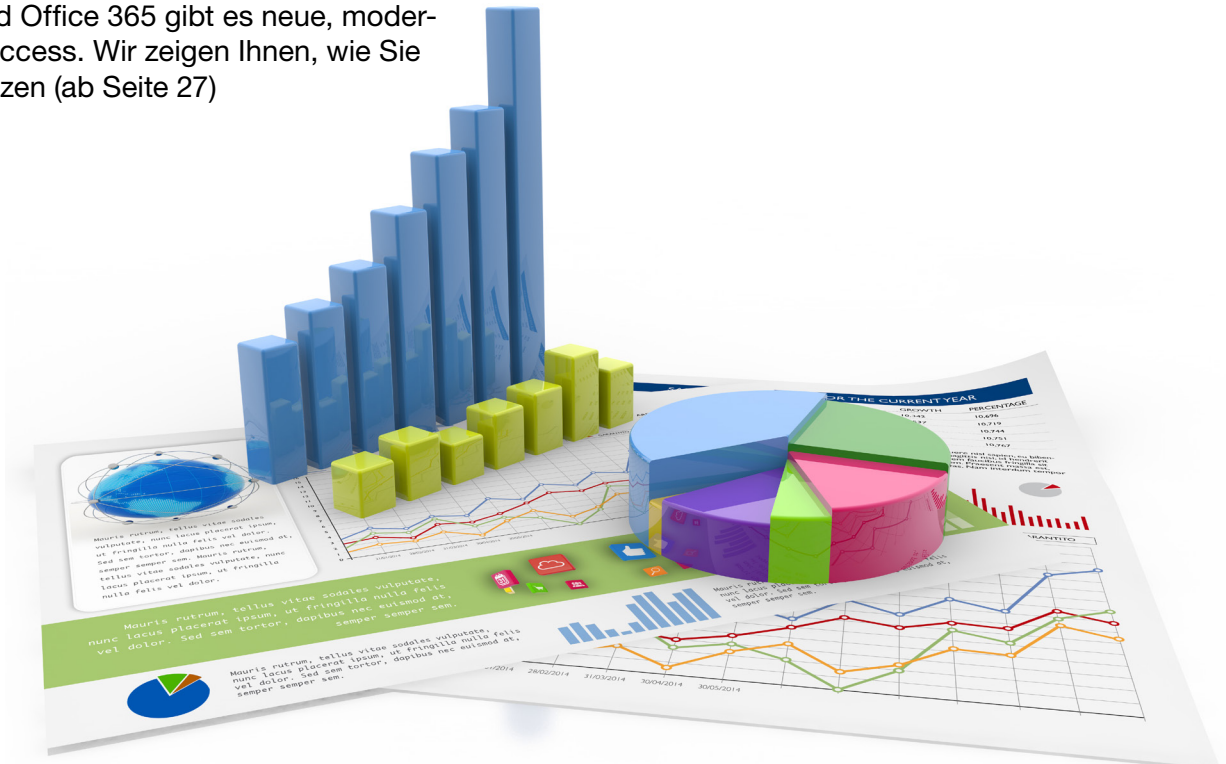
HAUFE.
Ausgabe 02/2019

ACCESS
IM UNTERNEHMEN

MODERNE DIAGRAMME

Mit Access 2019 und Office 365 gibt es neue, moderne Diagramme für Access. Wir zeigen Ihnen, wie Sie diese optimal einsetzen (ab Seite 27)



In diesem Heft:

Heimliche Updates

Microsoft hat mit Office 365 schon seit einiger Zeit die übliche Politik hinsichtlich der Softwarelizenzen geändert. Statt einer einmal zu zahlenden Lizenz bucht man ein jährlich zu bezahlendes Abonnement und erhält dafür Updates am laufenden Band. Für den Benutzer ist das praktisch, für den Entwickler mitunter nicht. Tatsache ist: Neuerungen finden nun wesentlich schneller zum Rechner der Lizenznehmer.

Früher gab es nur wenige Möglichkeiten, die Datenbanksoftware Access zu lizensieren: Entweder man kaufte eine Einzellizenz oder man beschaffte sich das große Office-Paket und holte sich damit auch die anderen Programme wie Outlook, Word, Excel oder SharePoint auf den Rechner. Damit war man dann bis zum Erscheinen der neuen Version auf dem aktuellen Stand, abgesehen von einigen Updates mit Bugfixes. Im Jahre 2019 sieht das anders aus: Entweder Sie kaufen sich Access oder ein Office-Paket mit Einmal-Lizenz und erhalten dann ein Produkt namens Access 2019. Oder Sie buchen das Office 365-Abonnement für knappe zehn Euro im Monat und erhalten dann für ein Jahr alle Programme inklusive regelmäßiger Aktualisierungen.

Für den Benutzer etwa von Word oder Excel ist das angenehm, denn so kann man direkt von Neuerungen profitieren. Für Entwickler von Datenbanksoftware auf Basis von Microsoft Access ist da eher anstrengend. Nicht, dass es ein Problem wäre, immer direkt die neuesten Features zu erhalten. Aber durch die verschiedenen Lizenzarten erhöht sich auch die Anzahl der auf den Zielrechnern beim Kunden verfügbaren Versionen von Microsoft Access. Zusätzlich werden jetzt auch die 64bit-Versionen salonfähig, zumal es die ActiveX-Steuerelemente wie das TreeView-Steuerelement nun auch für die 64bit-Version gibt.

Wir wollen jedoch nicht nur meckern, sondern erfreuen uns auch an den durch die regelmäßigen Updates auf unseren Rechner gespülten Neuerungen. Da wären zum Beispiel die modernen Diagramme – ein neues Steuerelement, mit dem Sie Diagramme etwas einfacher als zuvor einrichten und steuern können und die von Haus aus eine moderner anmutende Optik mitbringen. Mit diesem Thema beschäftigen wir uns in drei Beiträgen: In **Moderne Diagramme** schauen wir uns die Grundlagen des neuen Steuerelements an (ab Seite 11), unter dem Titel **Moderne Diagramme mit VBA** betrachten wir die Programmierung mit VBA (ab Seite 27) und unter **Wertebereiche in Diagrammen per VBA einstellen** zeigen wir ab Seite 35, wie Sie verschiedene Wertebereiche auf der primären und der sekundären Y-Achse abbilden und einstellen.

Der zweite Schwerpunkt beschäftigt sich mit der Vergabe von Berechtigungen für die Formulare in Access-Datenbanken mit SQL Server-Backend, wobei wir uns in **Berechtigungen für Access-Objekte per SQL Server II: Formulare** (ab Seite 51) auf die Programmierung der dazu benötigten Formulare konzentrieren und unter **Berechtigungen für Access-Objekte per SQL Server III: Anwenden** ab Seite 64 auf den Einsatz dieser Formulare.

Die für die Vergabe von Berechtigungen ebenfalls interessanten Schemas schauen wir uns im Beitrag **SQL Server: Sicherheit mit Schema** ab Seite 43 an.

Und für die Programmierung von Formularen gibt es auch noch Lesestoff: **Tabulator in Rich-Text-Feldern** zeigt ab Seite 2, wie Sie die Nutzung der Tab-Taste in Rich-Text-Feldern erlauben.

Viel Spaß mit der neuen Ausgabe!

Ihr André Minhorst

Tabulator in Rich-Text-Feldern

Ein Kunde fragte, ob es möglich ist, im Text von Memofeldern im Rich-Text-Format auch das Tabulator-Zeichen einzugeben und damit eine strukturiertere Darstellung von Inhalten zu erreichen – vor allem für die Ausgabe in Berichten. Dazu sind mir einige Ideen gekommen, von denen sich aber nicht alle so einfach umsetzen ließen, wie es gedacht war. Schließlich hat sich aber doch noch eine sehr gut programmierbare Lösung herauskristallisiert.

In der Beispieldatenbank legen wir zunächst eine Tabelle an, die neben dem Primärschlüsselfeld **TextID** noch die Felder **Betreff** mit dem Datentyp **Kurzer Text** und das Feld **Inhalt** mit dem Datentyp **Langer Text** enthält (siehe Bild 1). Für dieses Feld stellen wir außerdem die Eigenschaft **Textformat** auf **Rich-Text** ein.

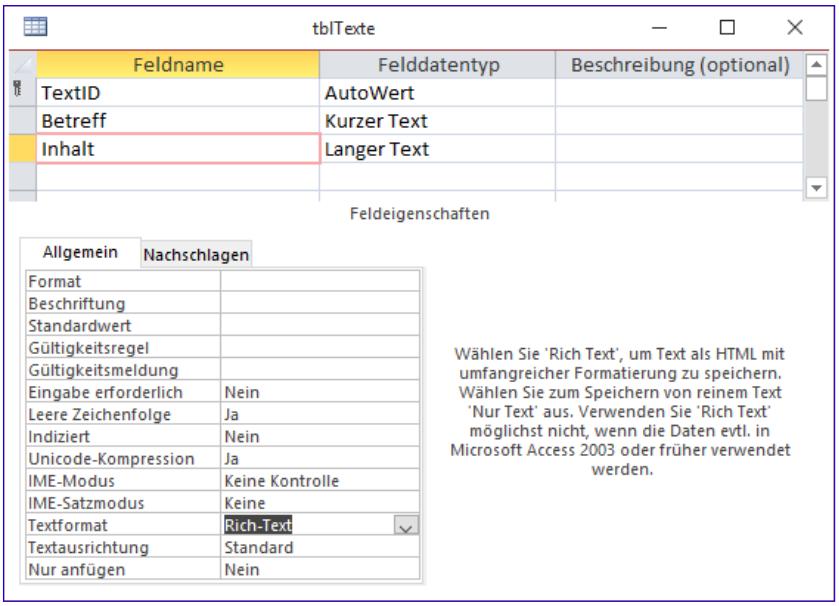


Bild 1: Hinzufügen eines Feldes mit dem Datentyp **Langer Text** und dem Textformat **Rich-Text**

Danach erstellen wir ein Formular, das wir über die Eigenschaft **Datensatzquelle** an die Tabelle **tblTexte** binden. Danach ziehen wir alle Felder dieser Tabelle aus der Feldliste in den Detailbereich des Formularentwurfs. Das Feld **Inhalt** gestalten wir etwas größer und stellen seine Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** auf **Beide** ein, damit es seine Größe mit der des Formulars verändert (siehe Bild 2).

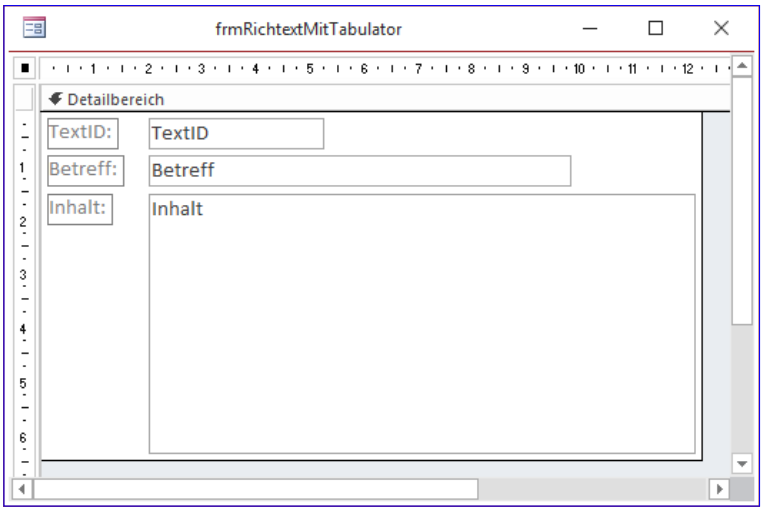


Bild 2: Formular zur Anzeige des Rich-Text-Feldes

Normales Tabulatur-Verhalten

Danach wird es interessant. Wir wechseln in die Formularansicht des Formulars und setzen den Fokus in das Textfeld, das an das Feld **Inhalt** gebunden ist und dessen Namen wir zuvor noch auf **txtInhalt** ändern.

Wenn wir nun ein paar Zeichen eingeben und dann die Tabulator-Taste drücken, erhalten wir das standardmäßige Verhalten: Access

verschiebt schlicht den Fokus auf das nächste Steuerelement, in diesem Fall auf das Feld **TextID**, und wechselt im gleichen Zuge den Datensatz, da unser Feld **txtInhalt** das letzte Steuerelement in der Aktivierreihenfolge ist (siehe Bild 3).

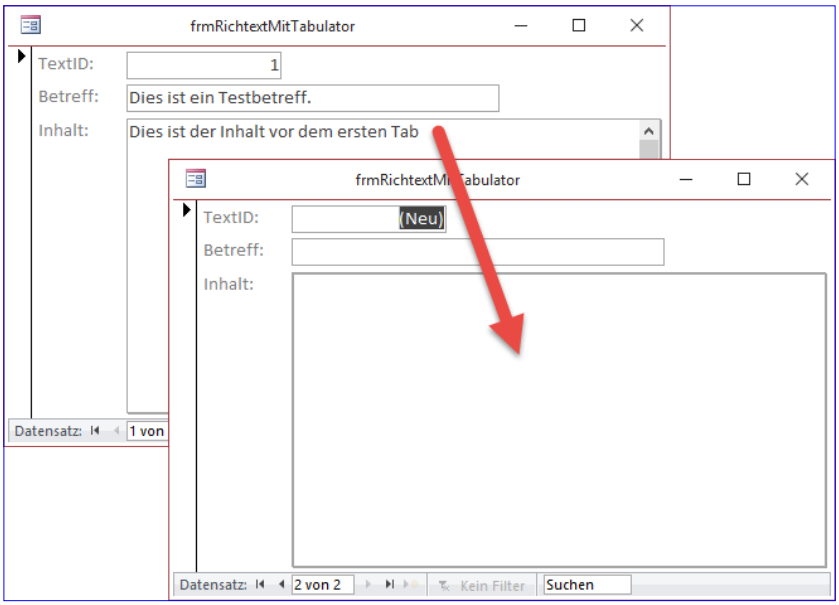


Bild 3: Die Tabulator-Taste im letzten Feld der Aktivierreihenfolge wechselt zum nächsten Datensatz.

Tabulator-Taste abfangen

Wir müssen also eine Möglichkeit finden, die Tabulator-Taste abzufangen, wenn sich der Fokus gerade im Textfeld **txtInhalt** befindet. Dann soll ein Tabulator-Zeichen in den Text eingefügt und die eigentliche Funktion der Tabulator-Taste unterbunden werden.

Dies erledigen wir mit einer passenden Ereignisprozedur. Wir müssen nur noch das passende Ereignis finden. Dabei handelt es sich vermutlich um das Ereignis **Bei Taste auf** oder **Bei Taste ab**. Um das Tabulator-Zeichen zum richtigen Zeitpunkt einzufügen, schauen wir uns an, wann eines der anderen Zeichen im Textfeld ausgegeben wird – beim Herabdrücken der Taste oder beim Loslassen.

Ein kurzes Experiment zeigt schnell, dass herkömmliche Zeichen immer direkt beim Herunterdrücken der Taste ausgegeben werden. Also verwenden wir das Ereignis **Bei Taste ab**. Für dieses stellen wir den Wert auf **[Ereignisprozedur]** ein und klicken dann auf die Schaltfläche mit den drei Punkten (**...**) rechts von der Eigenschaft.

Die im nun erscheinenden VBA-Editor erscheinende Ereignisprozedur ergänzen wir um eine Anweisung, die den Wert des Parameters **KeyCode** dieser Ereignisprozedur im Direktbereich des VBA-Editors ausgibt:

```
Private Sub Inhalt_KeyDown(KeyCode As Integer, _
        Shift As Integer)
    Debug.Print KeyCode
End Sub
```

Damit wechseln wir wieder in die Formularansicht und betätigen im Textfeld **txtInhalt** die Tabulator-Taste, um den Wert für **KeyCode** für diese Taste zu ermitteln. Das Ergebnis lautet **9**.

Wollen wir die eigentliche Funktion der Tabulator-Taste unterbinden, müssen wir also in der Prozedur prüfen, ob der Parameter **KeyCode** dem Wert **9** entspricht. Das erledigen wir in einer **Select Case**-Bedingung. Damit die übliche Funktion der Tabulator-Taste nicht ausgeführt wird, stellen wir **KeyCode** in diesem Fall auf den Wert **0** fest. Dies sieht dann in der Ereignisprozedur **txtInhalt_KeyDown** wie folgt aus:

```
Private Sub Inhalt_KeyDown(KeyCode As Integer, _
        Shift As Integer)
    Select Case KeyCode
        Case 9
            KeyCode = 0
    End Select
End Sub
```

Damit können Sie nun zumindest schon einmal die Tabulator-Taste im Textfeld **txtInhalt** betätigen, ohne dass

Spalten zweier Datenblätter synchronisieren

Im Beitrag »Neue Datensätze oben anfügen« haben wir uns angesehen, wie Sie ein Unterformular in der Datenblattansicht, das nur einen leeren, neuen Datensatz anzeigt, über einem normalen Datenblatt mit den Daten der gleichen Tabelle platzieren. Damit erhält der Benutzer die Möglichkeit, einen Datensatz oben einzufügen und nicht, wie sonst üblich, unten. Das Problem hierbei ist, dass der Benutzer immer noch die Spalten des oberen Formulars anders anordnen oder ihre Breite ändern kann. Das wirkt sich nicht automatisch auf die Spalten des darunter positionierten Formulars aus. Also müssen wir noch ein wenig Arbeit investieren, um dieses Verhalten zu dieser Lösung hinzuzufügen.

Wir rufen uns zuerst noch einmal die Konstellation der Lösung aus dem Beitrag **Neue Datensätze oben anfügen** (**www.access-im-unternehmen/1160**) ins Gedächtnis. Hier haben wir ganz oben im Formular **frmArtikel** ein Unterformular mit geringer Höhe platziert, welches nur die Spaltenköpfe sowie eine einzige Zeile anzeigen soll, die den neuen, leeren Datensatz anzeigt, der sonst unten im Datenblatt angezeigt wird (siehe Bild 1).

Wenn wir in die Formularansicht wechseln, sehen wir, dass das obere Unterformular genau so platziert ist, dass die Spaltenüberschriften des unteren Unterformulars nicht zu sehen sind. Dadurch erscheint die Ansicht so, als ob es sich tatsächlich nur um ein Formular in der Datenblattansicht handelt, bei dem die Zeile zum Einfügen eines neuen Datensatzes ganz oben angezeigt wird (siehe Bild 2).

Wenn wir nun allerdings die Breiten der Spaltenüberschriften ändern oder deren Anordnung ändern, wirkt sich dies nur auf die Spaltenköpfe selbst und die Zeile mit dem neuen Datensatz im oberen Unterformular aus

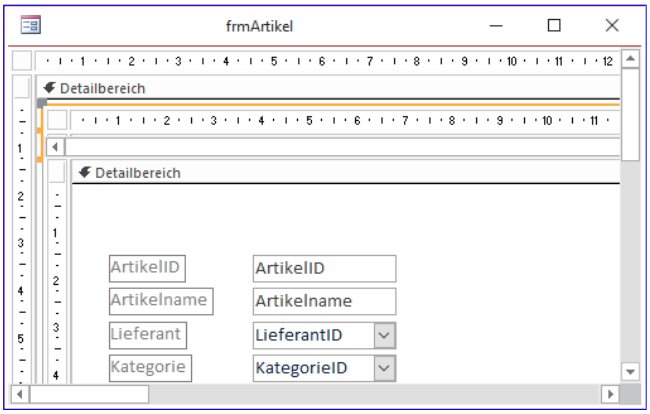


Bild 1: Entwurf der beiden übereinander angeordneten Unterformulare in der Datenblattansicht

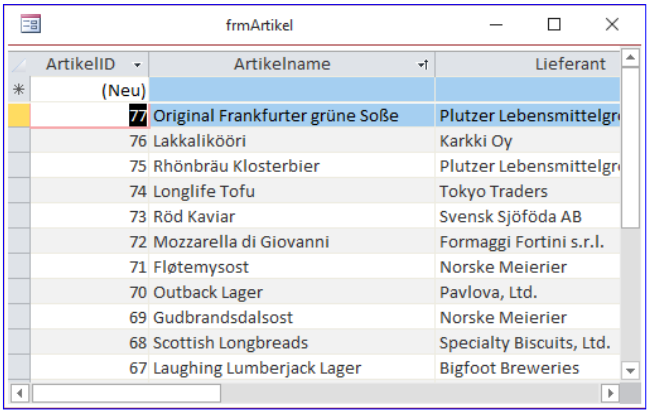


Bild 2: Formularansicht der beiden Unterformulare in der Datenblattansicht

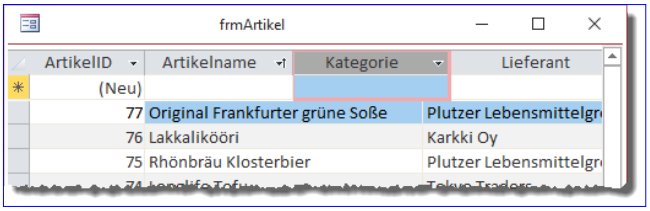


Bild 3: Das Anpassen der Spaltenköpfe wirkt sich nicht auf das untere Formular aus.

(siehe Bild 3). Das Verhalten ist ganz normal, denn das untere Unterformular weiß ja nichts vom oberen Unterformular.

Spalten synchron halten

Wenn wir wollen, dass die Spalten des unteren Unterformulars synchron mit den Spalten des oberen Unterformulars geändert werden, müssen wir die Ereignisse identifizieren, die beim Ändern der Spaltenbreiten und beim Ändern der Reihenfolge der Spalten ausgelöst werden. Wie finden wir diese heraus?

Falls uns kein Ereignis aufgrund seines Namens auffällt, legen wir für die möglichen Ereignisse jeweils eine Ereignisprozedur an und fügen darin entweder eine **Debug.Print**-Anweisung ein, die den Namen der Ereignisprozedur ausgibt, oder fügen Haltepunkte zu den jeweiligen Prozedurköpfen hinzu.

Hier kommen die folgenden Ereigniseigenschaften in Frage:

- **Beim Klicken**
- **Bei Maustaste ab**
- **Bei Maustaste auf**
- **Bei Mausbewegung**

Also hinterlegen wir testweise die folgenden Ereignisprozeduren für die Ereigniseigenschaften:

```
Private Sub Form_Click()
    Debug.Print "Beim Klicken"
End Sub

Private Sub Form_MouseDown(Button As Integer, _
        Shift As Integer, X As Single, Y As Single)
    Debug.Print "Bei Maustaste ab"
End Sub

Private Sub Form_MouseMove(Button As Integer, _
        Shift As Integer, X As Single, Y As Single)
    If Not Button = 0 Then
        Debug.Print "Bei Mausbewegung"
    End If
End Sub

Private Sub Form_MouseUp(Button As Integer, _
        Shift As Integer, X As Single, Y As Single)
    Debug.Print "Bei Maustaste auf"
End Sub
```

Hier erkennen wir dann, dass diese Ereignisse immer ausgelöst werden, wenn der Benutzer auf einen der Bereiche mit Ausnahme der Felder zur Eingabe der Daten eine Aktion mit der Maus ausführt – also in den Spaltenköpfen, im Datensatzmarkierer und in dem grauen Bereich links von den Spaltenköpfen und oberhalb des Datensatzmarkierers.

Wann synchronisieren?

Bevor nun losprogrammieren, müssen wir noch festlegen, wann die Spalten des unteren Unterformulars an die des oberen Unterformulars angepasst werden sollen. Erledigen wir dies nur, wenn der Benutzer eine Aktion zum Vergrößern oder Verkleinern der Breite oder zum Verschieben einer Spalte durchgeführt hat (also beim Ereignis **Bei Maustaste auf**) oder immer, wenn der Benutzer die Maus bewegt? Wir entscheiden uns zunächst für die erstere Methode. Wenn wir dann tatsächlich die Synchronisierung zu jedem Zeitpunkt ausprobieren wollen, brauchen wir die entsprechenden Anweisungen ja nur noch in das Ereignis **Bei Mausbewegung** einzufügen.

Der Ereignisprozedur **Bei Maustaste auf** fügen wir nun den Aufruf der noch zu programmierenden Prozedur **SpaltenSynchronisieren** hinzu:

```
Private Sub Form_MouseUp(Button As Integer, _
        Shift As Integer, X As Single, Y As Single)
    SpaltenSynchronisieren
End Sub
```

Prozedur zum Synchronisieren der Spalten

Die Prozedur zum Synchronisieren der Spalten könnten wir im oberen Unterformular **sfmArtikelNeu** implementieren. Allerdings müssten wir dann von dort aus das Unterformular, dessen Spalten wir manipulieren wollen, über das übergeordnete Formular referenzieren. Das ist schlechter Programmierstil.

Also programmieren wir diesmal gleich eine Klasse, mit der wir die beiden betroffenen Formulare referenzieren und die wir wiederum im Hauptformular deklarieren und instanzieren. Auf diese Weise haben wir erstens eine sauber programmierte Lösung und zweitens eine Lösung, die wir mit wenigen Zeilen Code in anderen Formularen wiederverwenden können.

Also fügen wir ein Klassenmodul namens **clsSpaltenSynchronisieren** zum VBA-Projekt hinzu. Dieses erhält zwei private deklarierte Variablen, welche die Verweise auf die beiden betroffenen Unterformulare aufnehmen sollen. Beide Variablen versehen wir mit dem Schlüsselwort **WithEvents**, damit wir für diese auch Ereignisse in der aktuellen Klasse implementieren können:

```
Private WithEvents m_SubFormNeu As Form
Private WithEvents m_Subform As Form
```

Damit die beiden Variablen mit den Verweisen auf die entsprechenden Unterformulare gefüllt werden können, legen wir noch zwei **Property Set**-Prozeduren an:

```
Public Property Set SubformNeu(frm As Form)
    Set m_SubFormNeu = frm
    With m_SubFormNeu
        .OnMouseUp = "[Event Procedure]"
    End With
End Property

Public Property Set Subform(frm As Form)
    Set m_Subform = frm
End Property
```

Für das Formular in **m_SubFormNeu** legen wir noch den Wert **[Event Procedure]** für die Eigenschaft **OnMouseUp** fest. So weiß die Klasse auch, dass das Ereignis hier ausgeführt werden soll.

Vorbereitung des Hauptformulars

Damit wir gleich testen können, während wir die Prozedur zum Synchronisieren der Spalten programmieren, fügen wir dem Klassenmodul des Hauptformulars direkt die notwendigen Elemente hinzu. Als Erstes eine Objektvariable für die Klasse **objSpaltenSynchronisieren**:

```
Dim objSpaltenSynchronisieren As clsSpaltenSynchronisieren
```

Dann erweitern wir die Ereignisprozedur **Form_Load** um die folgenden Zeilen, in denen wir **clsSpaltenSynchronisieren** instanzieren und die beiden Eigenschaften für das obere und das untere Unterformular füllen:

```
Private Sub Form_Load()
    ...
    Set objSpaltenSynchronisieren = _
                            New clsSpaltenSynchronisieren
    With objSpaltenSynchronisieren
        Set .Subform = Me!sfmArtikel.Form
        Set .SubformNeu = Me.sfmArtikelNeu.Form
    End With
End Sub
```

Ereignisse in der Klasse clsSpaltenSynchronisieren

In der Klasse **clsSpaltenSynchronisieren** legen wir nun die Ereignisprozedur an, die beim Loslassen der Maustaste auf einem der Spaltenköpfe ausgelöst werden soll. Diese fügen wir für die Objektvariable **m_SubFormNeu** hinzu.

Dazu wählen Sie im linken Kombinationsfeld des Codefensters den Eintrag **m_SubFormNeu** aus und im rechten den Eintrag **MouseUp**. Die so hinzugefügte Ereignisprozedur erweitern wir wie folgt:

Moderne Diagramme

Wenn Sie ein Office 365-Abonnement mit Access verwenden, erhalten Sie neue Funktionen von Access automatisch mit den Office 365-Updates. Es gibt auch noch das nicht im Abo erhältliche Access 2019, mit dem Sie neue Features gegenüber Access 2016 erhalten. In beiden Fällen können Sie die neuen Diagramme von Access nutzen. Dieser Beitrag zeigt, welche Diagrammtypen es gibt, wie Sie diese mit Daten füllen und wie Sie die Diagramme für Ihre Bedürfnisse anpassen können.

Die neuen Diagramme umfassen elf verschiedene Diagrammtypen, die Sie mit den Daten aus Tabellen oder Abfragen füllen können und deren Anzeige direkt nach der Auswahl der Datenquelle aktualisiert und somit sichtbar wird.

Beispielabfrage vorbereiten

Wir benötigen eine Beispielabfrage, welche passende Daten für die Anzeige in Diagrammen bereitstellt. Wir arbeiten wieder mit dem Datenmodell der Datenbank **Südsturm**, die einige Tabellen etwa mit Bestelldaten bereithält.

Wir wollen die Umsätze über verschiedene Zeiträume darstellen. Die Tabelle **tblBestellungen** enthält die Bestelldaten, die Tabelle **tblBestellpositionen** die Positionen mit der Anzahl, dem Rabatt und den Preisen.

Daher fassen wir die beiden Tabellen zunächst in einer Abfrage zusammen.

Diese Abfrage nennen wir **qryUmsaetzeNachQuartal** und fügen dieser die beiden Tabellen **tblBestellungen** und **tblBestelldetails** hinzu. Aus der Tabelle **tblBestellungen** ziehen wir das Feld **Bestelldatum** in das Entwurfsraster der Abfrage, aus der Tabelle **tblBestelldetails** benötigen wir gleich mehrere Felder, die wir im folgenden Ausdruck zusammenfassen:

```
PreisPosition: Summe([Einzelpreis]*[Anzahl]*(1-[Rabatt]))
```

Der Entwurf der Abfrage sieht nun wie in Bild 1 aus.

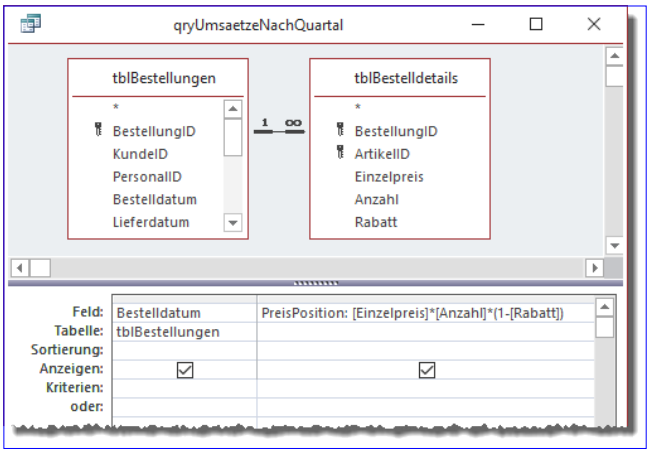


Bild 1: Zusammenführen der beiden Tabellen **tblBestellungen** und **tblBestelldetails**

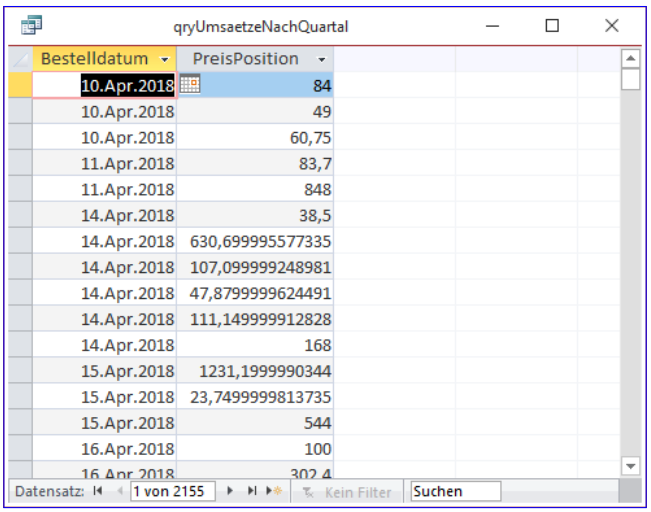


Bild 2: Datenblatt mit Bestelldatum und Preis je Position

Wechseln wir in die Datenblattansicht, erhalten wir das Ergebnis aus Bild 2. Die Abfrage liefert wie gewünscht das Datum der Bestellung und den Preis der Position, wobei

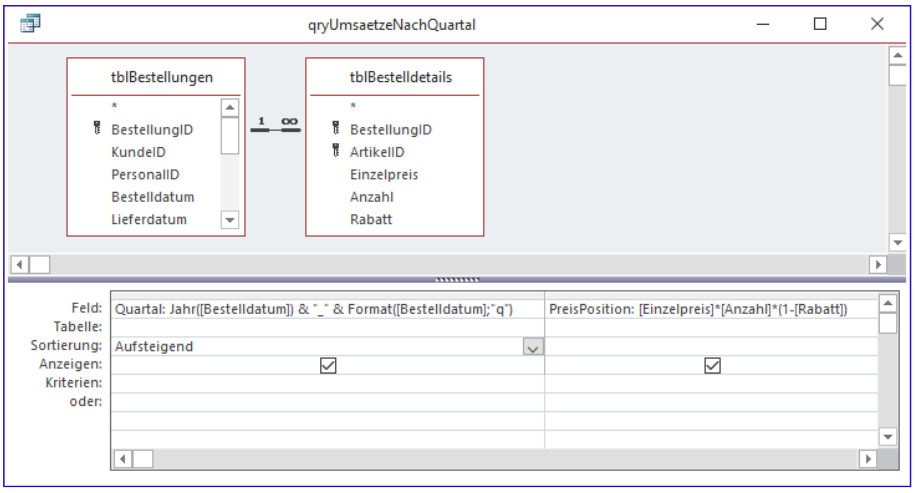


Bild 3: Ändern des Bestelldatums in einen Ausdruck, der das Jahr und das Quartal enthält.

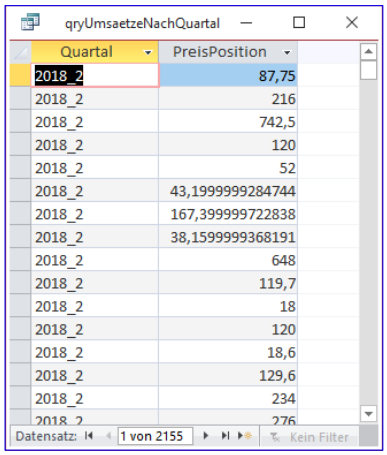


Bild 4: Angabe des Quartals einer Bestellposition und des Preises für diese Position

der Einzelpreis, die Anzahl und der Rabatt berücksichtigt werden.

Nun benötigen wir allerdings nicht das Bestelldatum, sondern wir wollen die Summe der Preise der Bestellpositionen für die einzelnen Quartale bilden. Dazu wollen wir den Wert des Feldes **Bestelldatum** zunächst in einen Ausdruck umwandeln, der das Jahr und das Quartal in der Form **yyyy_q** enthält, also etwa **2019_1**. Dazu ersetzen wir in der Abfrage das Feld **Bestelldatum** durch den folgenden Ausdruck:

```
Quartal: Jahr([Bestelldatum]) & "_" &
Format([Bestelldatum];"q")
```

Quartal gibt die Beschriftung des Feldes an. **Jahr(Bestelldatum)** ermittelt das Jahr der Bestellung. **Format([Bestelldatum];"q")** liefert eine Zahl, die dem Quartal des Datums entspricht (siehe Bild 3).

Das Zwischenergebnis sieht nun wie in Bild 4 aus. Hier sehen wir nun das Quartal und den Preis. Allerdings beziehen sich die Angaben immer noch auf die einzelne Bestellposition. Wir wollen aber die Summen je Quartal haben. Das erreichen wir durch den letzten Schritt.

Dazu aktivieren Sie die Anzeige der Funktionen, indem Sie in der Entwurfsansicht der Abfrage im Ribbon den Eintrag **Entwurf|Einblenden/Ausblenden|Summen** aktivieren. Dies fügt dem Entwurfsraster der Abfrage eine weitere Zeile hinzu, die den Zeilenkopf **Funktion** trägt. Für diese Zeile enthalten beide Spalten zunächst den Wert **Gruppierung**. Für die Spalte **Quartal** können wir diesen Wert beibehalten, da wir ja die Datensätze der Abfrage nach dem Quartal gruppieren wollen. Für die zweite Spalte mit den Preisen je Position stellen wir die Funktion auf den

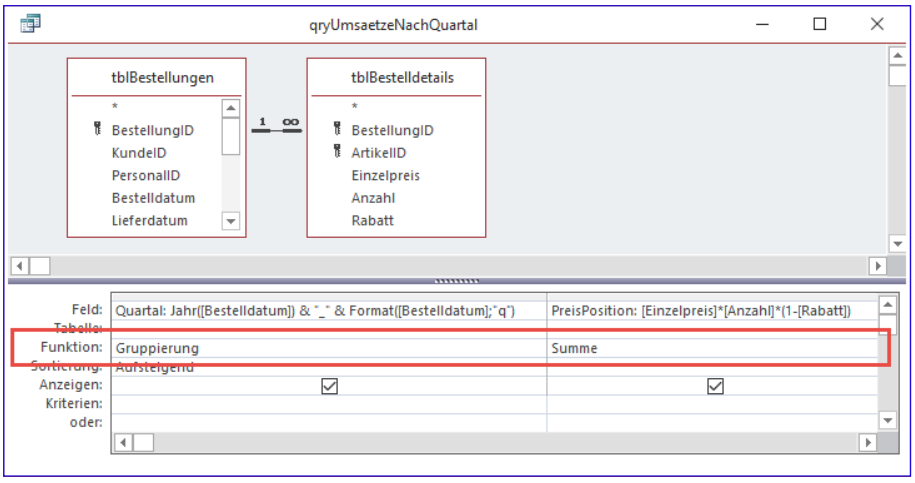


Bild 5: Gruppieren nach Quartal und Bilden der Summe der Preise für dieses Quartal

Wert **Summe** ein, damit für jede Gruppierung nach dem Quartal die Summe der Bestellpositionen berechnet wird (siehe Bild 5). Außerdem stellen wir die Sortierung für die erste Spalte noch auf **Aufsteigend** ein.

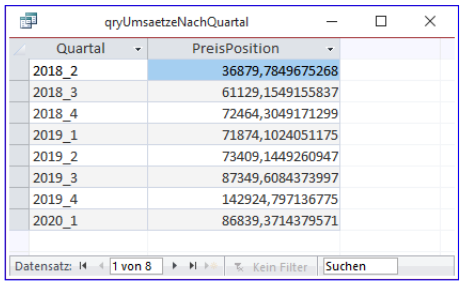


Bild 6: Umsatzsummen nach Quartal

Das Ergebnis sieht nun wie in Bild 6 und somit fast wie gewünscht aus.

Wir wollen nur noch den Namen der zweiten Spalte auf den passenderen Begriff **Quartalssumme** ändern. Dabei stellen wir nach erneutem Öffnen der Entwurfsansicht auch fest, dass Access aus der Funktion **Summe** die Funktion **Ausdruck** gemacht hat und die Summenfunktion in die Definition des Feldes übernommen hat (siehe Bild 7). Außerdem können wir in den Eigenschaften für dieses Feld noch die Eigenschaft **Format** auf **Euro** einstellen.

Diagramm anlegen

Damit erstellen wir ein neues Formular, dem wir das Diagramm hinzufügen wollen. Dies erledigen wir, nachdem wir ein neues, leeres Formular in der Entwurfsansicht geöffnet haben, über den Ribbon-Eintrag **Entwurf|Steuerelemente|Diagramm** (siehe Bild 8).

Nachdem Sie diesen Eintrag angeklickt haben, ziehen Sie im Formular einen Rahmen in der gewünschten Größe auf, damit das Diagramm-Steuerelement dort eingefügt wird. Gleich danach erscheint ein Dialog namens **Diagramm-Assistent** (siehe Bild 9).

Hier wählen wir die Option **Abfragen** aus, damit unsere frisch erstellte Abfrage zur Auswahl angeboten wird. Da dies der einzige Eintrag ist, können wir gleich auf **Weiter >** klicken. Damit landen wir dann beim zweiten Schritt, wo wir die Felder auswählen, die im Diagramm angezeigt werden können. Mit einem Klick auf die Schaltfläche **>>**

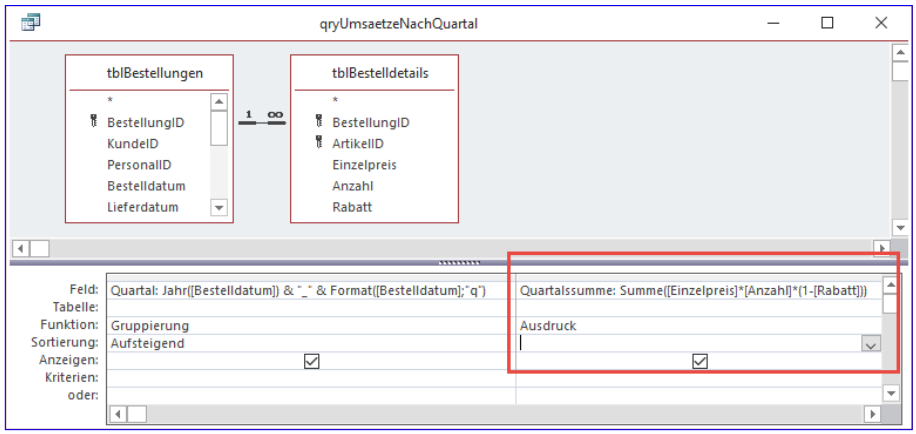


Bild 7: Letzte Änderungen

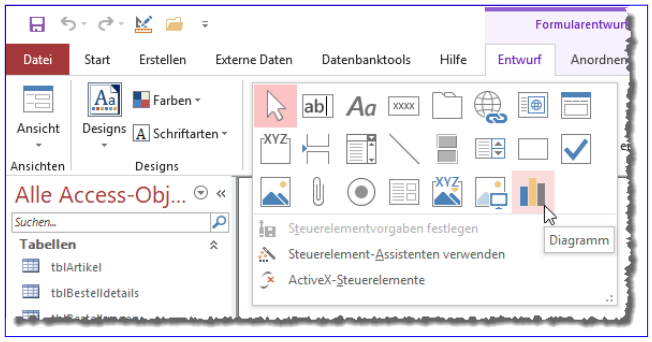


Bild 8: Hinzufügen eines Diagramms zu einem Formular

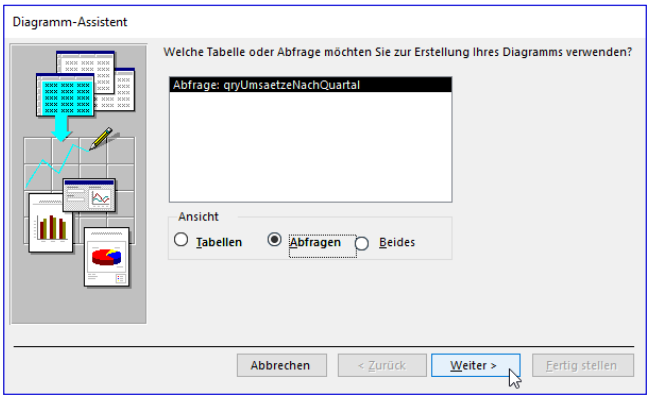


Bild 9: Erster Schritt im Diagramm-Assistent

fügen wir beide Felder auf einen Streich zur Liste der ausgewählten Felder hinzu (siehe Bild 10).

Im nächsten Schritt des Assistenten legen Sie fest, welchen Diagrammtyp Sie wählen wollen. Für unseren Zweck eigenen sich alle Diagrammtypen, mit denen Sie den zeitlichen Verlauf eines Wertes darstellen können. Wir wählen hier einfach das **Liniendiagramm** aus und wechseln dann zum nächsten Schritt des Assistenten (siehe Bild 11).

Danach zeigt der Assistent den Schritt aus Bild 12 an. Er versucht hier, eine intelligente Vorbelegung für die X- und die Y-Achse vorzunehmen. Das gelingt auch, denn wir wollen ja die Umsatzsummen auf der Y-Achse und die Quartale auf der X-Achse auftragen. Wenn die Zuordnung der Werte nicht passt, können Sie diese per Drag and Drop der rechts angezeigten Elemente, hier **Quartal** und **Quartalssumme**, anpassen. Beachten Sie, dass nach dem Ziehen eines der Elemente etwa zur Y-Achse dort zwei Elemente vorliegen und Sie eines wieder entfernen sollten. Das gelingt, indem Sie das zu entfernende Element ebenfalls per Drag and Drop greifen und zurück auf den Bereich rechts ziehen.

Schließlich legen wir im letzten Schritt des Assistenten noch fest, welchen Titel das Diagramm erhalten soll und ob Sie eine Legende anzeigen wollen oder nicht. Im Falle der Quartale und Quartalssummen scheint dies nicht notwendig zu sein, also wählen wir hier die zweite Option

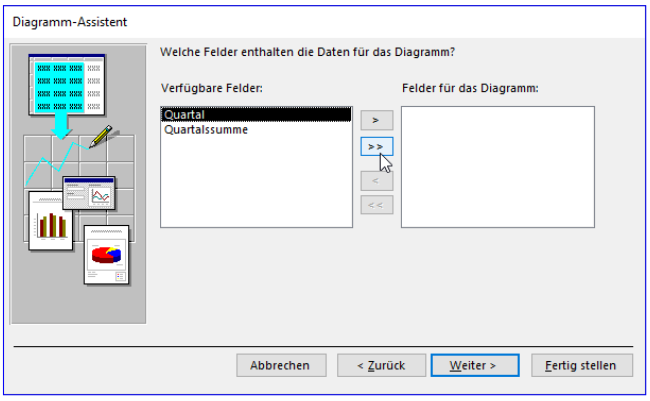


Bild 10: Auswahl der Felder für das Diagramm

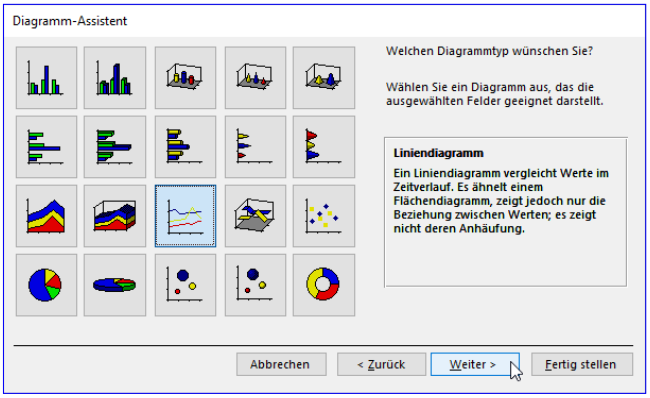


Bild 11: Auswahl des Diagrammtyps

(siehe Bild 13). Dann klicken Sie auf die Schaltfläche **Fertig stellen** und beenden so den Assistenten.

Diagramm in der Entwurfsansicht

Direkt nach dem Beenden des Assistenten zeigt Access eine Vorschau des Diagramms an, das allerdings mit den

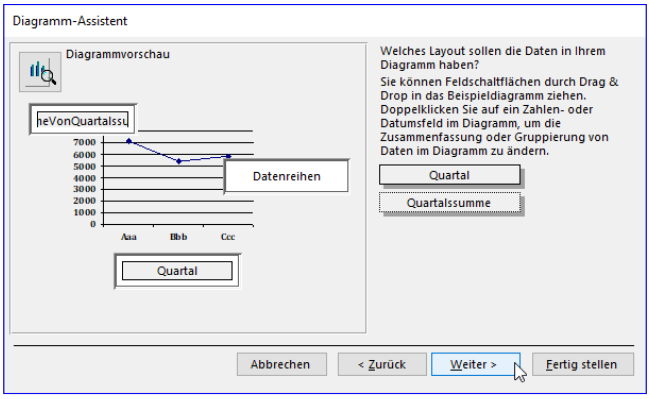


Bild 12: Festlegen des Layouts des Diagramms

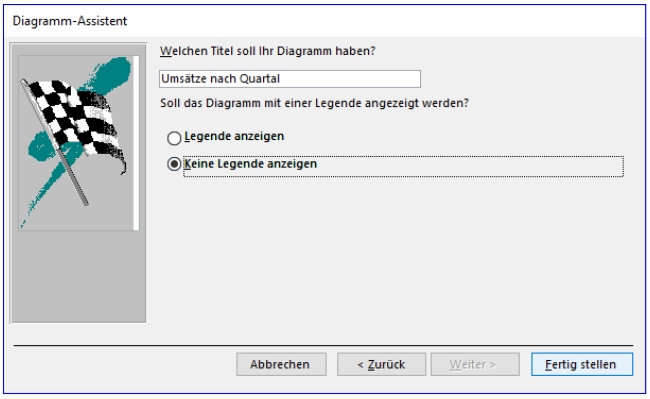


Bild 13: Festlegen von Titel und Legende

von uns angegebenen Informationen nicht viel zu tun hat – vielmehr stammen die Beschriftungen der Achsen sowie Werte nicht aus unserer Abfrage. Allein die Diagrammbeschriftung passt zu den im Assistenten gemachten Angaben (siehe Bild 14).

Erst wenn wir in die Formularansicht wechseln, erscheint das Diagramm mit den gewünschten Werten (siehe Bild 15).

Alt und neu statt »Aus alt mach neu«

Die Optik ist allerdings recht altbacken. Sollte die aktuelle Version von Access denn nicht eine Möglichkeit zum Erstellen modern aussehender Diagramme enthalten?

Irgendwie hatten wir schon beim Durchlaufen des Assistenten den Eindruck, als ob dieser nicht gerade auf dem aktuellsten Stand aussieht – eher nach dem Style von Access 97.

Und in der Tat ist dieser Assistent der alte Assistent zum Erstellen von Diagrammen. Wer sich ärgert, dass wir hier den alten Assistenten beschreiben, statt direkt auf die neuen Diagramme einzugehen – denken Sie an all die Benutzer, die noch kein Access 2016 besitzen und auch etwas vom Artikel haben wollen.

Neue Diagramm-Funktion

Die Funktion zum Erstellen der neuen Diagramme finden Sie ebenfalls im Ribbon, und zwar unter **Entwurf\|Steuerelemente\|Diagramm einfügen**. Hier finden Sie fünf Kategorien von Diagrammtypen, von denen die oberen drei jeweils drei Untereinträge enthalten – es gibt also elf verschiedene Diagrammtypen (siehe Bild 16).

Wie wollen die gleiche Abfrage wie zuvor nutzen und wählen auch den gleichen Diagrammtyp aus, nämlich **LinieLinie**. Nachdem wir diesen Eintrag angeklickt haben,

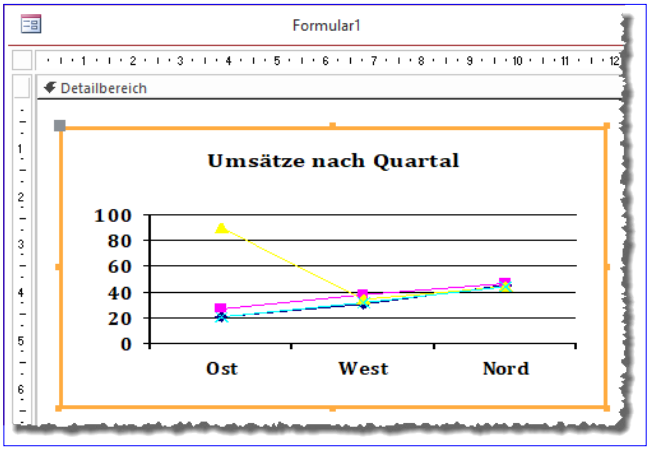


Bild 14: Entwurf des Diagramms

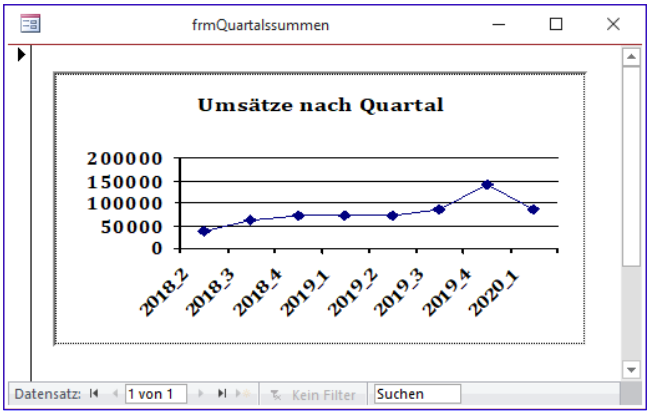


Bild 15: Das Diagramm in der Formularansicht

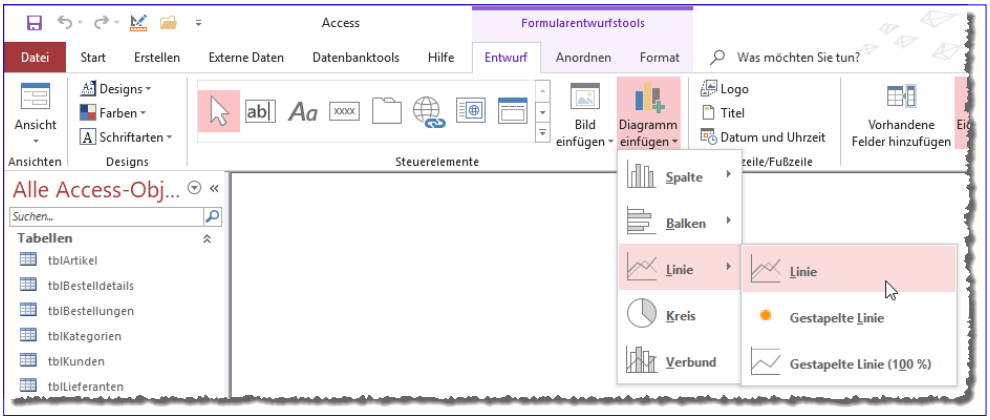


Bild 16: Aufruf der Funktion zum Erstellen eines modernen Diagramms

können wir wieder einen Rahmen mit der gewünschten Größe im Formularentwurf ausziehen. Das Ergebnis sieht wie in Bild 17 aus. Das Layout sieht schon im Entwurf wesentlich moderner aus, auch wenn auch hier noch nicht die richtigen Beschriftungen und Werte erscheinen.

Kein Wunder, denn im Falle des neuen Diagramms haben wir dieses ja auch einfach zum Entwurf hinzugefügt, ohne zuvor den Assistenten zu durchlaufen. Das wäre also ein Vorteil der alten Diagramme. Wir benennen das Diagramm-Steuerelement zuerst in **ctlDiagramm** um und schauen uns dann die Formularansicht an. Diese ist leer – kein Wunder, denn wir haben ja noch nicht einmal eine Datenquelle für das Diagramm festgelegt.

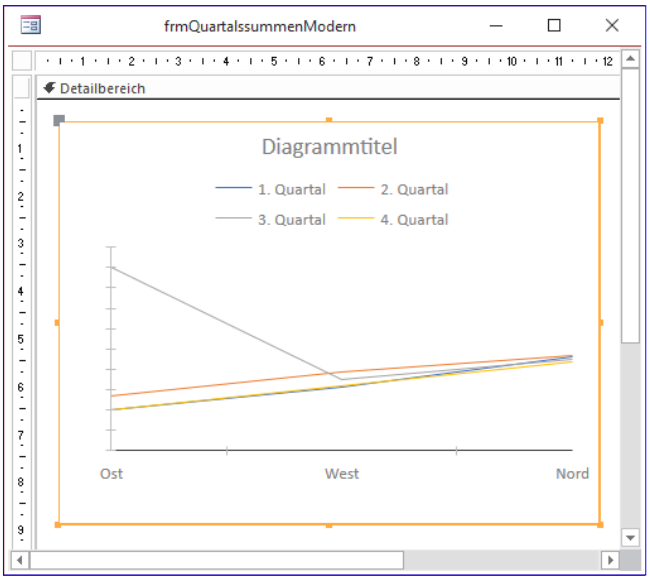


Bild 17: Entwurf des neuen Diagramms

Diagramm-Eigenschaften

Im Gegensatz zu den alten Diagrammen gibt es bei den neuen Diagrammen eine ganze Reihe mehr Eigenschaften, mit denen Sie das Aussehen des Diagramms selbst einstellen können. Schon auf der Seite **Daten** finden wir die folgenden Eigenschaften:

Vorschau von Livedaten anzeigen: Gibt an, ob schon während der Entwurfsansicht die gewählten Daten angezeigt werden sollen. So können wir, wenn diese Eigenschaft den Wert **Ja** aufweist, direkt den Diagrammtitel über die Eigenschaft **Diagrammtitel** einstellen (siehe Bild 18).

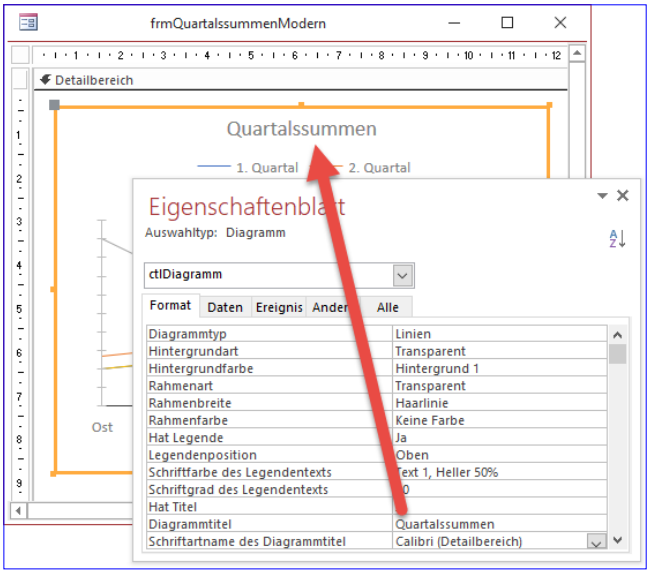


Bild 18: Einstellen des Diagrammtitels und Anzeige der Änderung in der Entwurfsansicht

Datensatzherkunft: Stellt die Tabelle oder Abfrage ein, aus der die Daten für das Diagramm stammen. Die Daten werden direkt im Diagramm angezeigt (siehe Bild 19). Außerdem füllen wir im gleichen Zuge automatisch die

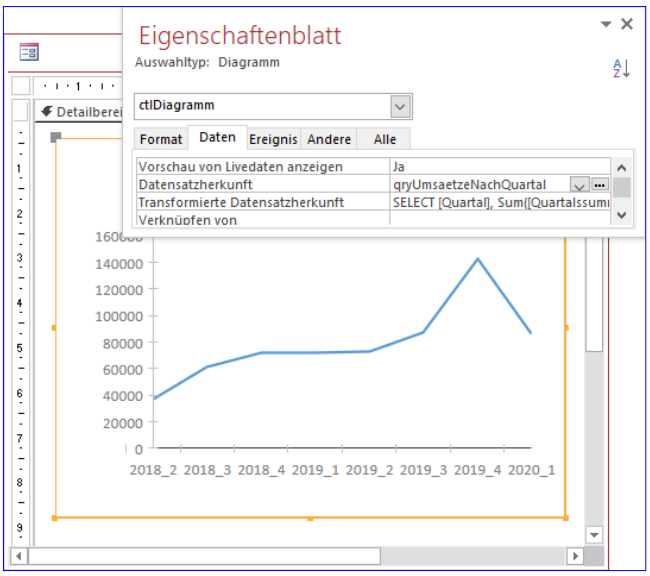


Bild 19: Einstellen der Datensatzherkunft des Diagramms

Eigenschaft **Transformierte Datensatzherkunft**.

Transformierte Datensatzherkunft: Diese Eigenschaft wird automatisch gefüllt, wenn Sie die Eigenschaft **Datensatzherkunft** gefüllt haben. In unserem Fall sieht der Wert dieser Eigenschaft wie folgt aus:

```
SELECT [Quartal],
Sum([Quartalssumme])
AS [SumOfQuartalssumme]
FROM [qryUmsaetzeNachQuartal]
GROUP BY [Quartal]
ORDER BY [Quartal]
```

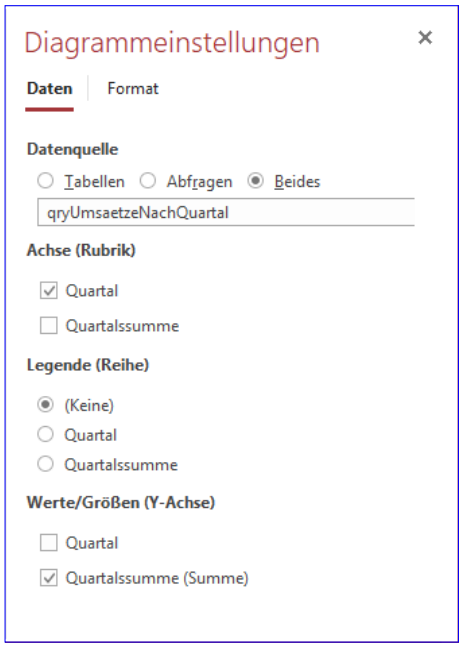


Bild 20: Diagrammeinstellungen, Seite **Daten**

Mit den Eigenschaften **Verknüpfen von** und **Verknüpfen nach** können Sie, ähnlich wie bei einem Unterformular-Steuerelement, festlegen, dass die Inhalte der Datensatzherkunft des Diagramms nach dem Wert eines Feldes der Datensatzherkunft gefiltert werden sollen. Es werden dann nur diejenigen Werte berücksichtigt, die mit dem Feld des übergeordneten Formulars übereinstimmen, dass mit der Eigenschaft **Verknüpfen nach** angegeben wurde. Ein Beispiel für den Einsatz dieser beiden Eigenschaften finden Sie im Beitrag **Diagramme mit gefilterten Daten** (**www.access-im-unternehmen.de/1186**) in der nächsten Ausgabe. Mit der Eigenschaft **Diagrammachse** wählen Sie ein Feld der Datensatzherkunft aus, dessen Werte auf der Diagrammachse oder X-Achse aufgetragen werden sollen.

Die Eigenschaft **Diagrammlegende** gibt an, welches der Felder der Datensatzherkunft für die Anzeige einer Legende verwendet werden soll.

Mit der Eigenschaft **Diagrammwerte** wählen Sie ein Feld der Datensatzherkunft aus, dessen Werte auf der Y-Achse aufgetragen werden sollen.

Mit diesen Eigenschaften können wir bereits einige Einstellungen für das Diagramm vorstellen, die wir uns direkt in der Entwurfsansicht live ansehen können.

Der Bereich Diagrammeinstellungen

Rechts im Access-Fenster erscheint außerdem ein Bereich namens **Diagrammeinstellungen**. Dieser weist zwei Registerkarten namens **Daten** und **Format** auf. Auf der Registerseite **Daten** (siehe Bild 20) finden wir die folgenden Einstellungen:

Die Eigenschaft **Datenquelle** dient wie die Eigenschaft **Datensatzherkunft** der Auswahl der Tabelle oder Abfrage, deren Daten im Diagramm angezeigt werden sollen. Hier können Sie die verfügbaren Einträge nach **Tabellen**, **Abfragen** oder **Beides** filtern.

Nach der Auswahl einer der Tabellen oder Abfragen bieten die folgenden drei Eigenschaften jeweils alle Felder der gewählten Datenquelle zur Auswahl an. Im Falle der von uns gewählten Abfrage **qryUmsaetzeNachQuartal** handelt es sich hierbei nur um die beiden Felder **Quartal** und **Quartalssumme**.

- **Achse (Rubrik)**: Gibt das Feld an, dessen Daten auf der X-Achse aufgetragen werden sollen – in unserem Beispiel also das Feld **Quartal**.
- **Legende (Reihe)**: Diese Eigenschaft macht hier keinen Sinn, da wir dafür mehrere Linien auftragen müssten. Wir zeigen später, wie dies funktioniert.
- **Werte/Größen (Y-Achse)**: Hier wählen wir das Feld aus, deren Werte auf der Y-Achse aufgetragen werden sollen, in unserem Fall **Quartalssumme**.

Moderne Diagramme mit VBA

Die mit Access 2019/Office 365 eingeführten modernen Diagramme bieten eine VBA-Schnittstelle an, mit der Sie die Diagramme steuern können. Dieser Beitrag zeigt, wie das Objektmodell für moderne Diagramme aussieht und liefert einige Beispiele für die Nutzung der VBA-Steuerung für moderne Diagramme. In einem Fall zeigen wir etwa, wie Sie die Bereiche der Y-Achse für ein Diagramm abhängig von den enthaltenen Werten einstellen können.

Da die modernen Diagramme wie die übrigen Steuerelemente in Access integriert sind und nicht etwa als ActiveX-Steuerelement bereitgestellt werden (wie es beispielsweise beim alten Diagramm-Steuerelement der Fall war), finden Sie diese im Objektkatalog des VBA-Editors als eigene Klasse namens **Chart** (siehe Bild 1).

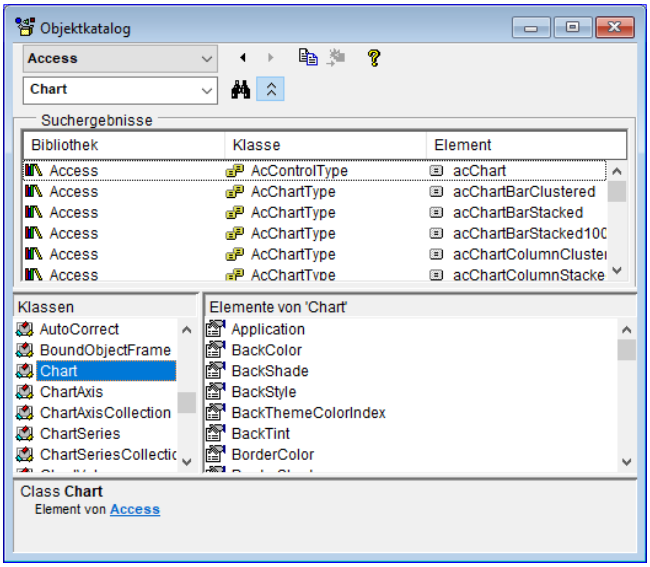


Bild 1: Chart-Klasse im Objektkatalog

Hier finden wir nicht nur bei der Suche nach dem Begriff **Chart** direkt die Konstante für den Steuerelementtyp, nämlich **acChart**, sondern auch die Klasse **Chart** mit all ihren Eigenschaften.

Darunter gibt es auch einige Elemente, die wiederum Auflistungen oder Klassen enthalten wie **ChartAxis**, **ChartAxisCollection** und so weiter.

Aber ist mit der **Chart**-Klasse tatsächlich das moderne Diagramm-Steuerelement gemeint oder referenzieren wir damit eventuell sogar das alte, etwas betagtere Diagramm-Element? Das müssten wir erst einmal herausfinden.

Dazu fügen wir einem Formular sowohl ein altes als auch ein modernes Diagramm-Steuerelement hinzu und benennen diese in **ctlOld** und **ctlNew** um (siehe Bild 2).

Unterschiede zwischen altem und neuem Diagramm

Dann öffnen wir das Formular in der Formularansicht und lassen uns im Direktbereich des VBA-Editors die Typen der beiden Diagramme ausgeben. Zuerst rufen wir die **TypeName**-Funktion für das alte Diagramm auf und erhalten als Ergebnis **ObjectFrame**:

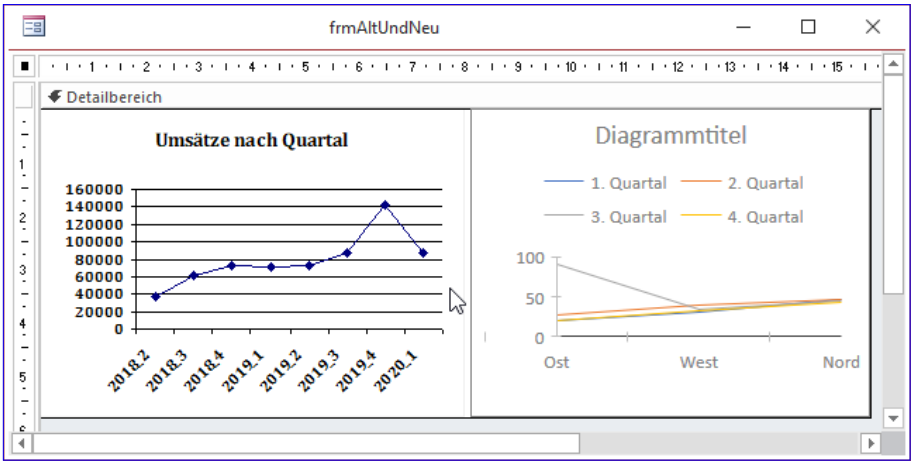


Bild 2: Ein altes und ein modernes Diagramm-Steuerelement

```
? Typename(Forms!frmAltUndNeu!ctlAlt)
ObjectFrame
```

ObjectFrame ist aber nur der Container, in dem das eigentliche Steuerelement enthalten ist. Also ermitteln wir noch den Typ des enthaltenen Steuerelements:

```
? Typename(Forms!frmAltUndNeu!ctlAlt.Object)
Chart
```

Danach führen wir den gleichen Befehl für das neue Diagramm aus:

```
? Typename(Forms!frmAltUndNeu!ctlNeu)
Chart
```

Das ist nun interessant, denn beide Steuerelemente entsprechen zumindest einem Objekttyp gleichen Namens. Sind die **Chart**-Eigenschaften, die wir im Objektkatalog gefunden haben, aber nun die Eigenschaften für das moderne oder für das herkömmliche **Chart**-Element? Wenn wir uns ein Detail anschauen, scheint es sich hier um die Eigenschaften für das moderne Element zu handeln.

Die Eigenschaft **ChartType** der **Chart**-Klasse liefert nämlich genau elf Elemente – und so viele Möglichkeiten bietet auch die Eigenschaft **Diagrammtyp** im Eigenschaftsfenster für das moderne Diagramm-Steuerelement im Formular.

Die herkömmliche **Chart**-Klasse bietet 18 verschiedene Diagrammtypen. Gehen wir also an dieser Stelle davon aus, dass die **Chart**-Klasse die für das moderne Diagramm passenden Eigenschaften liefert.

VBA-Eigenschaften der Chart-Klasse

Wir verschaffen uns zunächst einen Überblick über die Eigenschaften der **Chart**-Klasse und ordnen diese den Eigenschaften im Eigenschaftenblatt beziehungsweise im Bereich **Diagrammeinstellungen** für das moderne Diagramm-Steuerelement zu.

Allgemeine Eigenschaften

Die folgenden Eigenschaften sind die allgemeinen Eigenschaften des Diagramm-Steuerelements.

Wir geben jeweils den VBA-Namen der Eigenschaften an, dann die Entsprechung in der Benutzeroberfläche und, soweit notwendig, die Funktion der angegebenen Eigenschaft:

- **ChartAxis** (Bereich **DiagrammeinstellungenlDaten**, dort die markierten Einträge der Eigenschaft **Achse (Rubrik)**, getrennt durch Semikola und in eckige Klammern eingefasst.
- **ChartAxisCollection** (keine Entsprechung): siehe weiter unten unter **Auflistungen in Diagrammen**
- **ChartSeriesCollection** (keine Entsprechung): siehe weiter unten unter **Auflistungen in Diagrammen**
- **ChartType** (**Diagrammtyp**): Einer der Werte der Auflistung **acChartType**, also **acChartBarClustered** (**Gruppierte Balken**), **acChartBarStacked** (**Gestapelte Balken**), **acChartBarStacked100** (**Gestapelte Balken 100%**), **acChartColumnClustered** (**Gruppierte Säulen**), **acChartColumnStacked** (**Gestapelte Säulen**), **acChartColumnStacked100** (**Gestapelte Säulen 100%**), **acChartCombo** (**Verbund**), **acChartLine** (**Linien**), **acChartLineStacked** (**Gestapelte Linien**), **acChartLineStacked100** (**Gestapelte Linien 100%**), **acChartPie** (**Kreis**)
- **ChartValues** (Bereich **DiagrammeinstellungenlDaten**, dort die markierten Einträge der Eigenschaft **Werte/Größen (Y-Achse)**, getrennt durch Semikola und in eckige Klammern eingefasst)
- **ChartValuesCollection** (Bereich **DiagrammeinstellungenlDaten**, dort die markierten Einträge der Eigenschaft **Werte/Größen (Y-Achse)**): Details siehe weiter unten unter **Auflistungen in Diagrammen**

- **HasAxisTitles** (**Hat Achsentitel**): Stellt ein, ob die Titel der Rubrikenachse sowie der primären und sekundären Größenachsen angezeigt werden.

- **LinkChildFields** (**Verknüpfen von**): Gibt an, über welches Feld der Datensatzherkunft des Diagramm-Steuerelements die Verknüpfung hergestellt werden soll.

- **LinkMasterFields** (**Verknüpfen nach**): Gibt an, über welches Feld der Datensatzherkunft des Formulars beziehungsweise über welches Steuerelement des Formulars die Verknüpfung hergestellt werden soll.

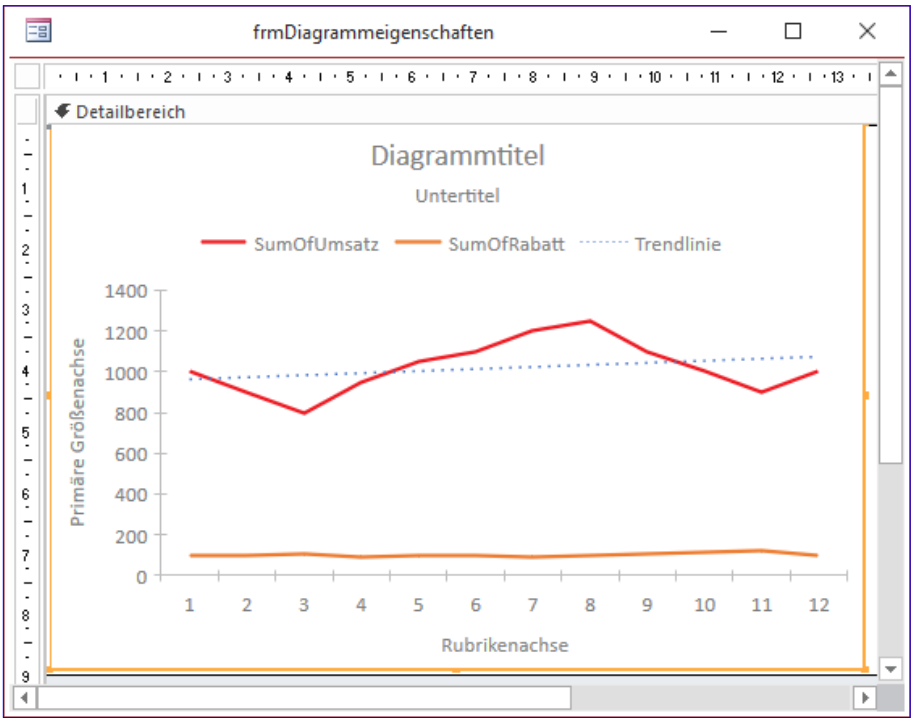


Bild 3: Einige mit dem jeweiligen Namen beschrifteten Eigenschaften eines Diagramms

- **TransformedRowSource** (**Transformierte Datensatzherkunft**): Schreibgeschützte Eigenschaft, die die über die verschiedenen Eigenschaften ermittelte transformierte Datensatzherkunft enthält.

Eigenschaften des Titels des Diagramms

Diese Eigenschaften verwenden Sie zum Einstellen des Aussehens der Titelzeile des Diagramms:

- **ChartTitle** (**Diagrammtitel**): Gibt den Titel des Diagramms an (siehe Bild 3).

- **ChartTitleFontName** (**Schriftartname des Diagrammtitels**)

- **ChartTitleFontSize** (**Schriftgrad des Diagrammtitels**)

- **ChartTitleFontColor**, **ChartTitleFontShade, ChartTitleFontTint, ChartTitleThemeColorIndex** (zusammengefasst in **Schriftfarbe des Diagrammtitels**)

- **HasTitle** (**Hat Titel**): Gibt an, ob ein Titel angezeigt werden soll.

Eigenschaften für den Untertitel des Diagramms

Die folgenden Eigenschaften verwenden Sie, um den Untertitel eines Diagramms und seine Eigenschaften einzustellen:

- **ChartSubtitle** (Diagrammuntertitel): Gibt den Untertitel des Diagramms an.

- **ChartSubtitleFontColor**, **ChartSubtitleFontShade**, **ChartSubtitleFontTint**, **ChartSubtitleThemeColorIndex** (zusammengefasst in **Schriftfarbe des Diagrammuntertitels**)

- **ChartSubtitleFontSize** (**Schriftgrad des Diagrammuntertitels**)

- **HasSubtitle** (**Hat Untertitel**): Gibt an, ob ein Untertitel angezeigt werden soll.

Eigenschaften für die X-Achse/Rubrikenachse

Diese Eigenschaften geben an, wie die X-Achse/Rubrikenachse ausgestattet werden soll:

- **CategoryAxisFontColor**, **CategoryAxisFontShade**, **CategoryAxisFontTint**, **CategoryAxisThemeColorIndex** (zusammengefasst in **Schriftfarbe der Rubrikenachse**)
- **CategoryAxisFontSize** (**Schriftgrad der Rubrikenachse**)
- **CategoryAxisTitle** (Titel der Rubrikenachse)

Eigenschaften für die Legende

Auch für die Anzeige der Legende gibt es verschiedene Eigenschaften.

Diese finden Sie in der folgenden Auflistung:

- **ChartLegend** (Bereich **Diagrammeinstellungen|Daten**, Eigenschaft **Legende (Reihe)**): Eine leere Zeichenkette entspricht der Auswahl des Eintrags **(Keine)**. Bei der Auswahl eines der anderen Einträge für diese Eigenschaft wird die Bezeichnung der entsprechenden Reihe eingetragen.
- **HasLegend** (**Hat Legende**): Gibt an, ob eine Legende angezeigt werden soll.
- **LegendPosition** (**Legendenposition**): kann die Werte der Auflistung **acLegendPosition** annehmen, und zwar **acLegendPositionBottom**, **acLegendPositionLeft**, **acLegendPositionRight** und **acLegendPositionTop**.
- **LegendTextFontColor**, **LegendTextFontShade**, **LegendTextThemeColorIndex** (zusammengefasst in **Schriftfarbe des Legendentitels**)
- **LegendTextFontSize** (**Schriftgrad des Legendentitels**)

Primäre und sekundäre Y-Achse

Die folgenden Eigenschaften stehen für die primäre und die sekundäre Y-Achse zur Verfügung und sind meist im Bereich **Format** des Eigenschaftenblatts zu finden:

- **PrimaryValuesAxisDisplayUnits** (**Anzeigeeinheiten der primären Größenachse**): Einer der Werte der Auflistung **acAxisUnits**, also **acAxisUnitsThousands** et cetera
- **PrimaryValuesAxisFontColor**, **PrimaryValuesAxisFontShade**, **PrimaryValuesAxisFontSize**, **PrimaryValuesAxisFontTint**, **PrimaryValuesAxisThemeColorIndex** (zusammengefasst in Schriftfarbe der primären Größenachse): Eigenschaften für die Schrift der Beschriftung der Punkte der Achsen
- **PrimaryValuesAxisFormat** (**Format der primären Größenachse**): Gibt das Format für die Werte der primären Größenachse an.
- **PrimaryValuesAxisMaximum** (**Maximum der primären Größenachse**): Größter Wert für die primäre y-Achse
- **PrimaryValuesAxisMinimum** (**Minimum der primären Größenachse**): Kleinster Wert für die primäre y-Achse
- **PrimaryValuesAxisRange** (**Bereich der primären Größenachse**): Einer der Werte der Auflistung **acAxisRange**, also **acAxisRangeAuto** (**Automatisch**) oder **acAxisRangeFixed** (**Fest**)
- **PrimaryValuesAxisTitle** (**Titel der primären Größenachse**): Titel der primären y-Achse
- **SecondaryValuesAxisDisplayUnits** (**Anzeigeeinheiten der sekundären Größenachse**): Einer der Werte der Auflistung **acAxisUnits**, also **acAxisUnitsThousands** et cetera

Wertebereiche in Diagrammen per VBA einstellen

Die neuen Diagramme unter Access 2019 und aktuellen Versionen von Office 365 bieten eine moderne Darstellung von Daten. Dem Benutzer bleiben die Möglichkeiten der Anpassung der angezeigten Daten jedoch verwehrt, wenn er nicht gerade in den Entwurf des Formulars oder Berichts mit dem Diagramm eingreift. Also bieten wir ihm die Möglichkeit, verschiedene Aspekte der Darstellung über von uns bereitgestellte Steuerelemente nach seinen Wünschen anzupassen. Im vorliegenden Beitrag schauen wir uns dabei die Wertebereiche von Diagrammen für die primäre und sekundäre y-Achse an.

Benutzern das Anpassen der Diagramme ermöglichen

Der Benutzer sollte in der Regel keine Gelegenheit erhalten, in den Entwurf von Formularen einzugreifen. Damit hat er aber leider auch keine Möglichkeit, die Ansicht der Diagramme anzupassen.

Hier kommt VBA ins Spiel. Sie können mit geeigneten Steuerelementen die Werte, die für die verschiedenen Eigenschaften des Diagramm-Steuerelements eingestellt wurden, anzeigen und zur Bearbeitung anbieten. Die grundlegenden Einstellungen wie etwa die gewählte Diagrammart oder die Datensatzherkunft sollte man nicht bearbeiten lassen, da dies unter Umständen zu viele Gelegenheiten mit sich bringt, ein unerwünschtes Ergebnis zu produzieren. Also beschränken wir uns zu Beispielzwecken mit einigen wenigen Einstellungen.

Beispieldiagramm

Für dieses Beispiel verwenden wir ein Diagramm, dass die Daten der Tabelle **tblUmsaetzeUndRabattNachMonat** anzeigt. Dazu stellen Sie die Eigenschaft **Diagrammtyp** auf **Linien** und **Datensatzherkunft** auf den Namen der Tabelle ein.

Im Bereich **DiagrammeinstellungenlDaten** legen wir für die Anzeige auf der y-Achse die beiden Felder **Umsatz (Summe)** und **Rabatt (Summe)** fest (siehe Bild 1), indem wir die entsprechenden Kontrollkästchen aktivieren.

Die angezeigten Texte für den Diagrammtitel und die Legendenbeschriftung stellen wir über die Eigenschaften **Diagrammtitel** (im Eigenschaftenblatt) sowie **Anzeigename** (im Bereich **DiagrammeinstellungenlFormat**) ein.

Außerdem möchten wir noch die sekundäre y-Achse einblenden. Das erledigen wir, indem wir die Eigenschaft **Datenreihe zeichnen auf** im Bereich **DiagrammeinstellungenlFormat** für die Datenreihe **SumOfRabatt** auf den Wert **Sekundär** einstellen.

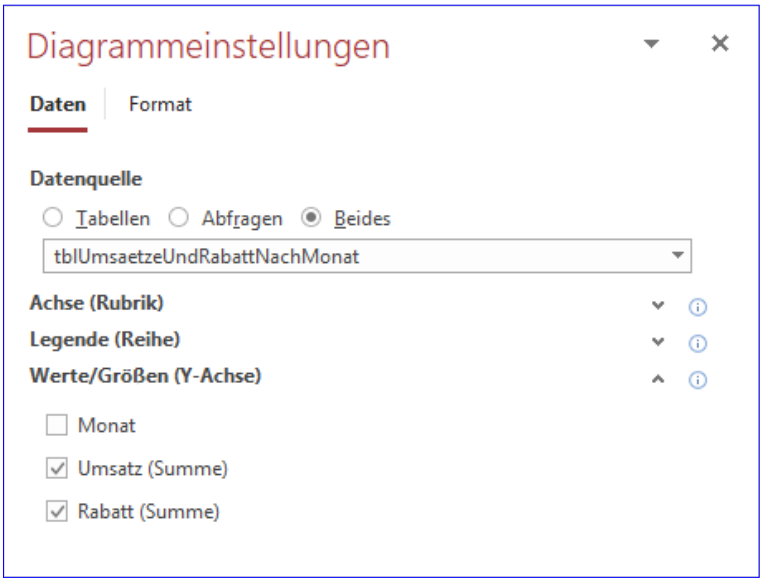


Bild 1: Auswahl der Felder für die Werte der y-Achse

Schließlich wünschen wir uns noch Beschriftungen der beiden y-Achsen. Dazu tragen wir für die Eigenschaft **Titel der primären Größenachse** den Wert **Umsatz** und für **Titel der sekundären Größenachse** den Wert **Rabatt** ein.

Das Ergebnis sieht nun wie in Bild 2 aus.

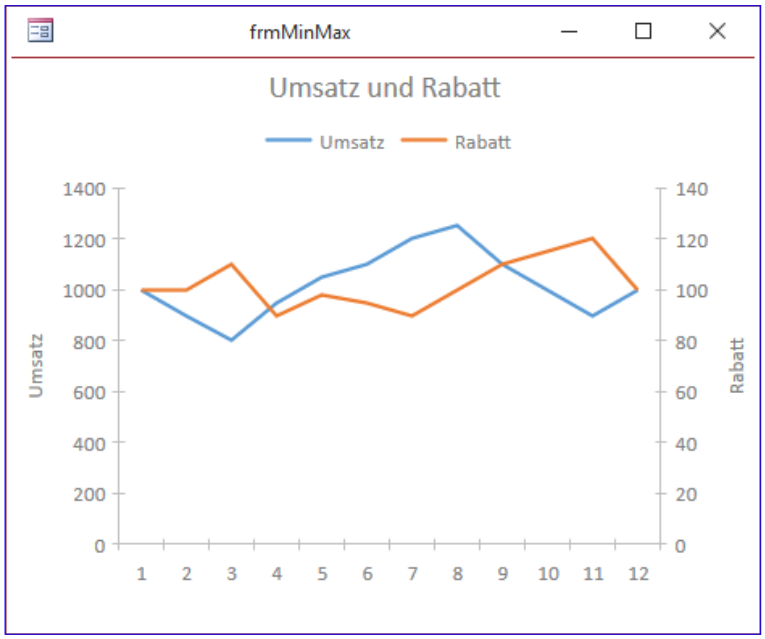


Bild 2: Diagramm ohne Benutzereinstellungen

Minimale und maximale Werte für y-Achse automatisch einstellen

Nun wollen wir dem Benutzer die Möglichkeit geben, die Bereiche für die primäre und die sekundäre y-Achse einzustellen. Normalerweise sind hier die beiden Eigenschaften **PrimaryValuesAxisRange** und **SecondaryValuesAxisRange** auf den Wert **acAxisRangeAuto** für **Automatisch** eingestellt.

Wenn wir dem Benutzer die Möglichkeit geben wollen, diese Werte manuell einzustellen, müssen wir diese beiden Eigenschaften zunächst auf den Wert **acAxisRangeFixed** einstellen. Die weiteren beiden Eigenschaften je y-Achse heißen **PrimaryValuesAxisMinimum** und **PrimaryValuesAxisMaximum** sowie **SecondaryValuesAxisMinimum** und **SecondaryValuesAxisMaximum**. Mit diesen stellen wir den Startwert und den Endwert für den anzuzeigenden Wertebereich ein.

Die Idee ist, dem Benutzer ausgehend von der automatischen Anzeige der Bereiche, die Möglichkeit zu bieten, die Ober- und die Untergrenze für den Wertebereich manuell einzutragen. Oder er kann durch das Betätigen eines Steuerelements den aktuellen Wertebereich ermitteln und einstellen lassen.

Wir entscheiden uns, für den Wert der Eigenschaft **PrimaryValuesAxisRange** eine Optionsgruppe zum Formular hinzuzufügen, der wir den Namen **ogrPrimaerWertebereich** geben. Für diese Optionsgruppe stellen wir den Standardwert auf **0** ein (was der Konstanten **acAxisRangeAuto** entspricht).

Der Optionsgruppe fügen wir dann zwei Optionsfelder hinzu, denen wir die Werte **0** und **1** zuweisen und die wir mit **Automatisch** und **Manuell** beschriften. Die zweite Optionsgruppe für den Wertebereich der sekundären y-Achse erstellen wir auf die gleiche Weise, vergeben aber den Namen **ogrSekundaerWertebereich**.

Nun benötigen wir noch vier Textfelder, welche die minimalen und maximalen Werte für die primäre und die sekundäre y-Achse aufnehmen. Diese platzieren wir unter der Optionsgruppe und so, dass die Felder für die primäre Achse links und die für die sekundäre Achse rechts erscheinen. Das Formular sieht nun wie in Bild 3 aus.

Steuerelemente programmieren

Nun wollen wir die Steuerelemente mit Funktion füllen. Als Erstes sorgen wir dafür, dass die Optionsgruppe den richtigen Wert enthält. Diesen lesen wir aus der entsprechenden Eigenschaft aus und weisen ihn in der Ereignisprozedur **Beim Laden** des Formulars zu:

Hier rufen wir gleichzeitig eine Prozedur namens **TextfelderAktivieren** auf:

```
Private Sub Form_Load()
    Me!ogrPrimaerWertebereich = _
            Me!ctlMinMax.PrimaryValuesAxisRange
    Me!ogrSekundaerWertebereich = _
           Me!ctlMinMax.SecondaryValuesAxisRange
    TextfelderAktivieren
End Sub
```

Diese Prozedur prüft, ob die Wertebereiche automatisch ermittelt werden sollen oder nicht.

Falls nicht, sollen die Textfelder zur manuellen Eingabe der kleinsten und größten Werte aktiviert werden. Das erledigen die vier Anweisungen der Prozedur wie folgt:

```
Private Sub TextfelderAktivieren()
    Me!txtPrimaerMax.Enabled = _
                  Me!ogrPrimaerWertebereich = 1
    Me!txtPrimaerMin.Enabled = _
                  Me!ogrPrimaerWertebereich = 1
    Me!txtSekundaerMax.Enabled = _
                Me!ogrSekundaerWertebereich = 1
    Me!txtSekundaerMin.Enabled = _
                Me!ogrSekundaerWertebereich = 1
End Sub
```

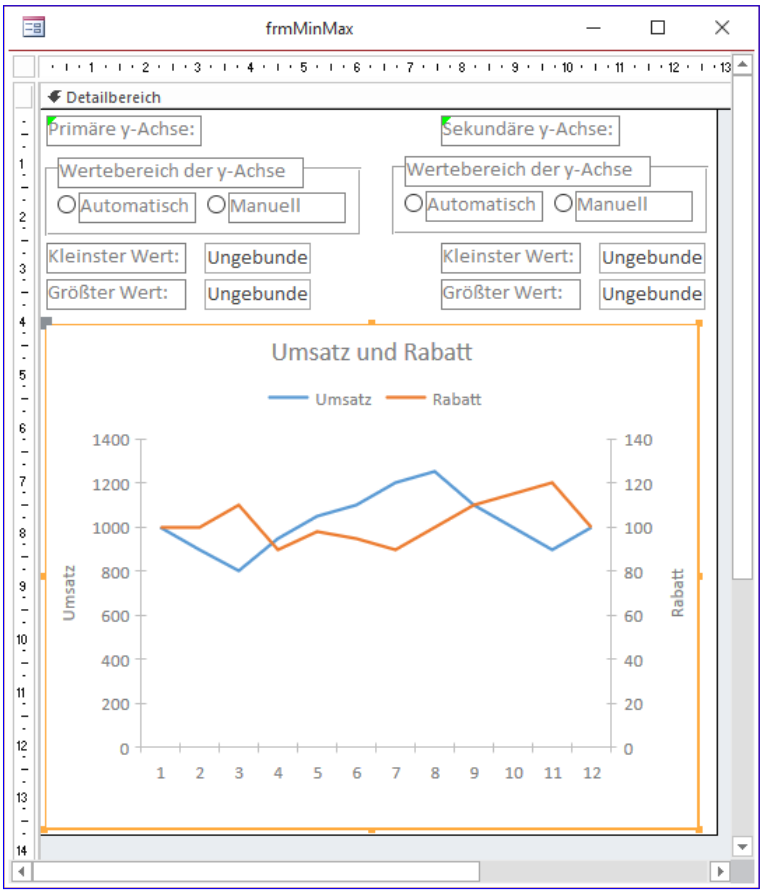


Bild 3: Entwurf des Formulars mit Steuerelementen für die Eigenschaften

Wenn der Benutzer den Wert einer der beiden Optionsgruppen ändert, also möchte, dass die Wertebereiche entweder automatisch festgelegt werden oder manuell, weisen wir der entsprechenden Eigenschaft des Diagramms, also **PrimaryValuesAxisRange** oder **SecondaryValuesAxisRange**, den entsprechenden Wert der jeweiligen Optionsgruppe zu.

Außerdem rufen wir dann erneut die Prozedur **TextfelderAktivieren** auf, um diese zu aktivieren oder deaktivieren:

```
Private Sub ogrPrimaerWertebereich_AfterUpdate()
    Me!ctlMinMax.PrimaryValuesAxisRange = _
                                Me!ogrPrimaerWertebereich
    TextfelderAktivieren
End Sub

Private Sub ogrSekundaerWertebereich_AfterUpdate()
    Me!ctlMinMax.SecondaryValuesAxisRange = _
                                Me!ogrSekundaerWertebereich
    TextfelderAktivieren
End Sub
```

Damit haben wir allerdings noch nicht viel erreicht, denn wir blenden so nur die Zahlenwerte an der primären und sekundären y-Achse ein und aus.

Kleinsten und größten Wert vorbelegen

Damit der Benutzer Standardwerte für die Wertebereiche hat, müssen wir diese entsprechend berechnen. Das können wir nur näherungsweise so tun, wie es das

Diagramm-Steuerelement im automatischen Modus tut, da wir ja den dort verwendeten Algorithmus nicht kennen. Also wollen wir den größten und den kleinsten Wert ermitteln und einen bestimmten Prozentsatz aufschlagen, damit der größte Wert nicht gerade ganz oben auf der jeweiligen y-Achse angezeigt wird.

Wir aber können wir diese Werte berechnen? Normalerweise ist das einfach, da wir ja wissen, welche Datensatzherkunft wir eingestellt haben und welche Felder für die Darstellung der Werte verwendet werden. Aber wir wollen die Lösung so gestalten, dass wir auch einmal die Datensatzherkunft wechseln und unsere Lösung immer noch verwenden können. Also schauen wir uns an, welche Möglichkeiten die Eigenschaften bieten, um die größten und kleinsten Werte der jeweiligen y-Achse zu berechnen.

Dabei stoßen wir schnell auf die Eigenschaften der Datenreihen, wo wir Bezeichnungen wie **SumOfUmsatz** oder **SumOfRabatt** finden. Diese werden interessanterweise auch in dem Ausdruck für die Eigenschaft **Transformierte Datensatzherkunft** verwendet, die in unserem Fall wie folgt aussieht:

```
SELECT [Monat], Sum([Umsatz]) AS [SumOfUmsatz],
Sum([Rabatt]) AS [SumOfRabatt]
FROM [tblUmsaetzeUndRabattNachMonat]
GROUP BY [Monat] ORDER BY [Monat]
```

Und wir wissen, dass die auf den beiden y-Achsen aufgetragenen Werte **SumOfUmsatz** und **SumOfRabatt** heißen. Daraus können wir dann durch geschicktes Formulieren einer Abfrage zum Beispiel wie folgt den maximalen Wert für **SumOfUmsatz** ermitteln – hier für die Ausgabe im Direktfenster:

```
Debug.Print CurrentDb.OpenRecordset("SELECT
Max(SumOfUmsatz) FROM (SELECT [Monat], Sum([Umsatz])
AS [SumOfUmsatz], Sum([Rabatt]) AS [SumOfRabatt] FROM
[tblUmsaetzeUndRabattNachMonat] GROUP BY [Monat] ORDER BY
[Monat])").Fields(0)
```

Das Ergebnis lautet korrekterweise **1250**.

Damit können wir dann auch noch den minimalen Wert für die primäre y-Achse ermitteln und den maximalen und den minimalen Wert für die sekundäre y-Achse.

Dabei dürfen wir allerdings nicht außer acht lassen, dass wir ja noch gar nicht wissen, welcher Wert auf welcher Achse aufgetragen ist. Und vielleicht sind ja sogar beide Datenreihen auf der gleichen y-Achse aufgetragen.
Aber auch diese Information können wir über die Eigenschaften ermitteln. Dazu benötigen wir die **Collection** namens **ChartSeriesCollection**. Diese liefert alle Datenreihen. Für jede Datenreihe finden wir im Bereich **Diagrammeinstellungen|Format** einen Satz von Eigenschaften. Durch Auswahl eines der Einträge des Kombinationsfeldes oben können wir die Eigenschaften für eine andere Datenreihe anzeigen lassen (siehe Bild 4). Hier interessieren uns besonders der Name der Datenreihe, denn diesen finden wir jeweils auch in der Abfrage aus der Eigenschaft **Transformierte Datensatzherkunft** wieder (hier **SumOfUmsatz** und **SumOfRabatt**). Außerdem interessiert uns hier, auf welcher y-Achse die Werte für die Datenreihe aufgetragen werden – auf der primären oder der sekundären y-Achse. Diese Eigenschaft finden wir unter **Diagrammeinstellungen|Format** unter **Datenreihe zeichnen auf**.

Unter VBA können wir ebenfalls auf diese Eigenschaften zugreifen. Die Datenreihen finden wir dabei in der bereits oben erwähnten Collection namens **ChartSeriesCollection**.

Datenreihen analysieren

Genau genommen kann es, im Gegensatz zu dem hier gezeigten Beispiel, auch Diagramme geben, wo es nur eine Datenreihe gibt, deren Werte auf der primären y-Achse aufgetragen werden, Diagramme mit zwei Datenreihen auf einer einzigen y-Achse oder auch Datenreihen, deren Werte sich auf die primäre und die sekundäre y-Achse aufteilen. In all diesen Fällen haben wir jedoch das gleiche

SQL Server: Sicherheit mit Schema

Die älteren SQL Server-Versionen haben einen Nachteil: Sie benötigen hier einen Benutzer, dem eine neu erstellte Datenbank gehört. Genauso gibt es auch für alle weiteren Objekte wie Tabellen, Sichten, gespeicherte Prozeduren und so weiter einen Besitzer – nämlich den Benutzer, der das Objekt erstellt hat. Meist legt man diese Elemente im Kontext eines Windows-Benutzers an, was zum Beispiel dazu führt, dass Sie diesen Benutzer nicht einfach löschen können, ohne zuvor alle Elemente dieses Benutzers an einen anderen Benutzer übertragen zu haben. Aus diesem Grund hat Microsoft mit SQL Server 2005 den Typ namens »Schema« eingeführt. Diesem werden nun nicht nur neu erstellte Objekte zugeordnet, sondern Sie können auch mehrere Schemas pro Datenbank verwenden, um beispielsweise Elemente nach verschiedenen Gruppen zu sortieren.

Schemas sind also grundsätzlich eine Art Namensraum für Datenbankobjekte wie Tabellen, Sichten, gespeicherte Prozeduren et cetera. Der Zweck ist dabei, wie oben bereits erwähnt, die direkte Zuordnung von Windows-Benutzern oder -Benutzergruppen als Besitzer der Datenbankobjekte zu unterbrechen, da sonst, wenn der Benutzer oder die Benutzergruppe entfernt werden soll, die Besitzer aller betroffenen Datenbankobjekte geändert werden müssten.

Wenn Sie hingegen mit Schemas arbeiten, erhalten Sie eine Zwischenschicht. Es gibt zwar nun immer noch einen Besitzer des Schemas, das wiederum die Datenbankobjekte enthält, aber Sie brauchen nur noch den Besitzer des Schemas zu wechseln, wenn Sie den alten Besitzer in Form eines Benutzers oder einer Benutzergruppe nicht mehr verwenden können.

Fachliche Gruppierung

Mit einem Schema können Sie die Objekte einer Datenbank nach dem fachlichen Zusammenhang gruppieren oder auch nach den gewünschten Berechtigungen. Damit können wir, um beim Beispiel der **Suedsturm_SQL**-Datenbank zu bleiben, etwa für die beiden Benutzergruppen **Management** und **Bestellannahme** zwei geeignete Schemas erstellen und diesen die relevanten Datenbankobjekte zuordnen.

Schemas für die Sicherheit

Außerdem können Sie natürlich auch Schemas einrichten, um über diese dann die Berechtigungen an den verschiedenen Elementen der Datenbank zu definieren. Die fachliche Gruppierung und die Gruppierung der Elemente nach Sicherheits- beziehungsweise Berechtigungsaspekten schließt sich ja nicht aus. Wenn Sie etwa, wie soeben vorgeschlagen, ein Schema namens Management und eines namens Bestellannahme erstellen, könnten Sie alle Elemente, die nur von den Mitarbeitern der Bestellannahme benötigt werden, im Kontext eines Schemas namens **Bestellannahme** speichern. Die Elemente, auf welche die Mitarbeiter der Bestellannahme keinen Zugriff haben sollen, aber das Management, erstellen Sie im Kontext eines Schemas namens **Management**. Wenn ein Mitarbeiter des Managements nun Zugriff nicht nur auf die Elemente des Schemas **Management** haben soll, sondern auch auf die Elemente des Schemas **Bestellannahme**, weisen Sie einen Windows-Benutzer einfach den Windows-Benutzergruppen **Management** und **Bestellannahme** zu.

Anlegen eines Schemas

Wir legen nun in unserer Beispieldatenbank **Suedsturm_SQL** zunächst ein neues Schema namens **Bestellannahme** an. Dazu navigieren Sie im Objekt-Explorer des SQL Server Management Studios zunächst zu dieser Datenbank und dann zum Eintrag **Sicherheit|Schemas**.

Wenn Sie diesen Eintrag öffnen, erkennen Sie, dass es dort schon eine Reihe vorgefertigter Schemas gibt, zum Beispiel **db_accessadmin**, **db_datareader**, **db_datawriter** und so weiter (siehe Bild 1). Einige dieser Schemas können Sie auch entfernen, wenn Sie verhindern wollen, dass diese versehentlich für Benutzer festgelegt werden.

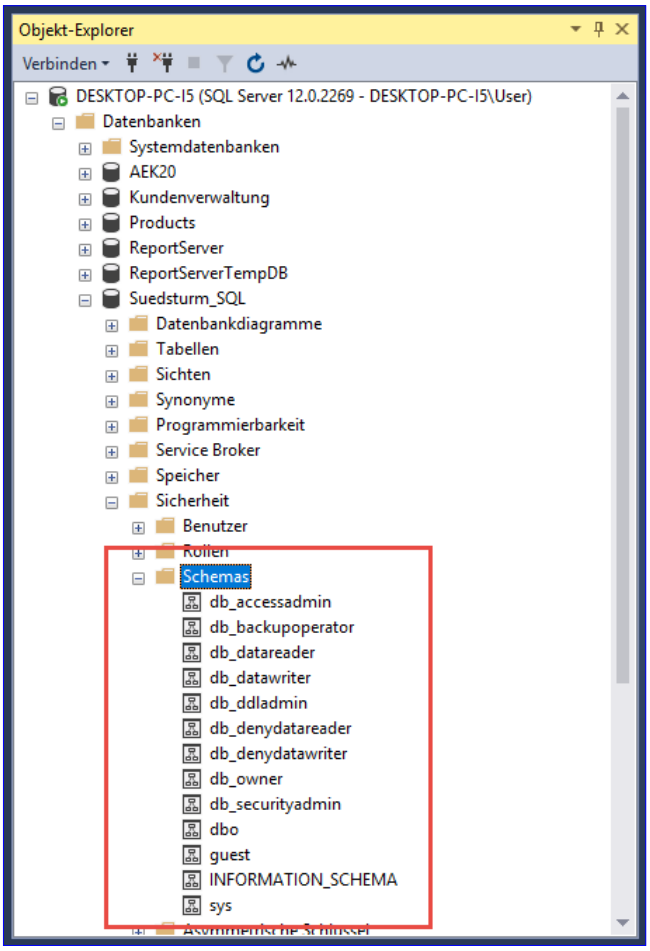


Bild 1: Schemas im Objekt-Explorer

Klicken Sie nun mit der rechten Maustaste auf den Eintrag **Schemas**, erscheint ein Kontextmenü, das unter anderem den Befehl **Neues Schema...** anbietet. Damit öffnen Sie nun den Dialog zum Anlegen eines neuen Schemas namens **Schema - Neu**. Hier geben Sie im Bereich **Allgemein** nun zunächst den Namen des Schemas ein, also **Bestellannahme**, und als Schemabesitzer **dbo** (siehe Bild 2). Das bedeutet, dass der Besitzer des Schemas identisch ist mit dem Besitzer der Datenbank.

Danach wechseln Sie zum Bereich **Berechtigungen**. Hier finden Sie die Schaltfläche **Suchen...**, mit der Sie einen weiteren Dialog namens Benutzer oder Rollen auswählen öffnen. Dieser bietet wiederum eine Schaltfläche namens **Durchsuchen...** an, mit der Sie den Dialog **Nach Objekten suchen** öffnen können. Hier werden alle Anmeldungen aufgeführt, die für den aktuellen SQL Server angelegt wurden (siehe auch Beitrag **Berechtigungen für Access-Objekte per SQL Server III: Anwenden**, **www.access-im-unternehmen.de/1185**).

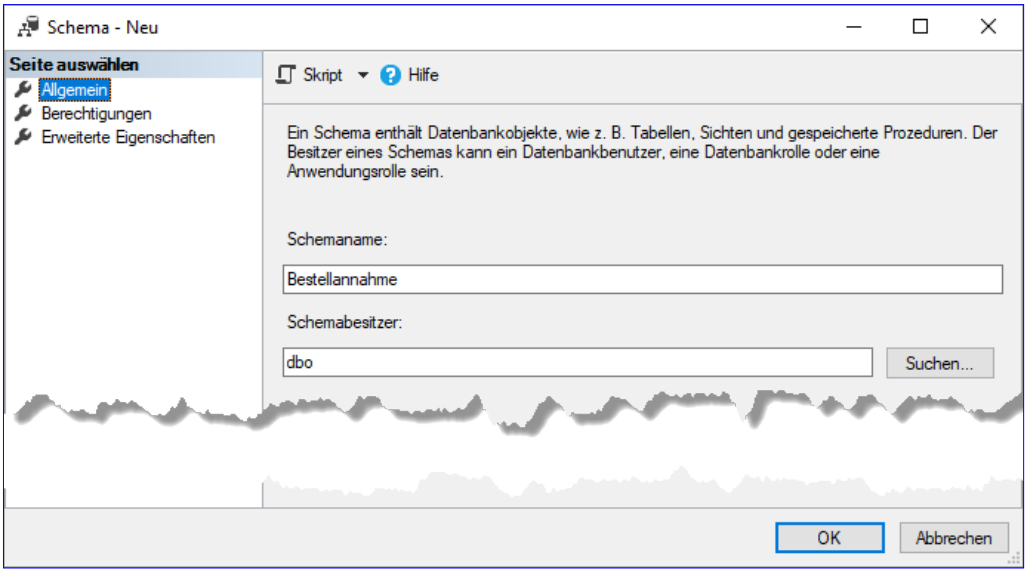


Bild 2: Anlegen eines neuen Schemas

Darunter finden Sie auch die Anmeldung **[<Servername>]\Bestellannahme**, die wir durch das Setzen eines Hakens zum Hinzufügen markieren (siehe Bild 3).

Danach finden Sie im Dialog **Schema - Neu** den gewählten Da-

tenbankbenutzer, also die Gruppe **Bestellannahme**, in der Liste der **Benutzer oder Rollen**.

Im Bereich darunter werden die expliziten Berechtigungen für diese Anmeldung angezeigt. Aktuell sind noch keine Berechtigungen markiert, aber das holen wir nun nach – dazu setzen Sie einen Haken in der Spalte Erteilen für die folgenden Berechtigungen:

- **Aktualisieren** (**UPDATE**),
- **Ausführen** (**EXECUTE**)
- **Auswählen** (**SELECT**),
- **Einfügen** (**INSERT**) und
- **Löschen** (**DELETE**).

Vielleicht fällt Ihnen in der Liste noch der Eintrag **Ändern** auf, dies bezieht sich aber nicht auf das Ändern von Daten – hierzu verwenden wir die Berechtigung **Aktualisieren** (siehe Bild 4).

Wenn Sie nun auf **OK** klicken, wird das neue Schema angelegt, das dann auch gleich in der Liste der Schemas im Objekt-Explorer erscheint. Die

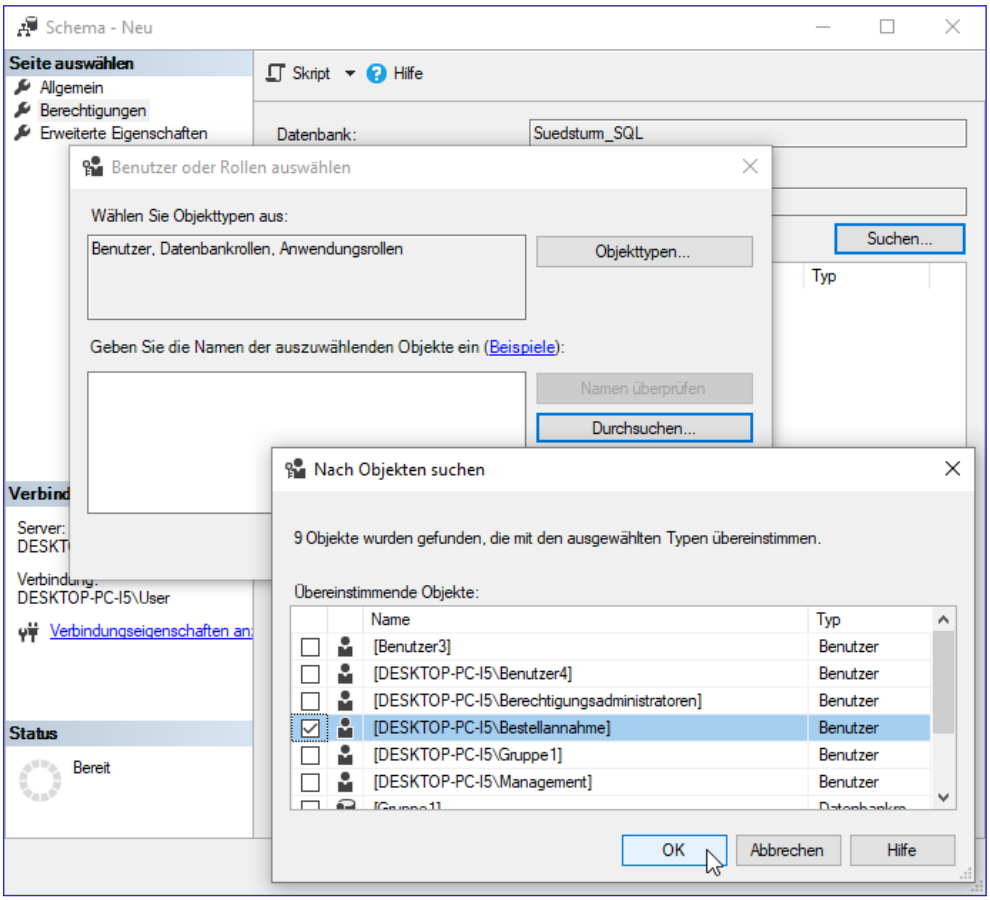


Bild 3: Auswählen der Benutzer des Schemas

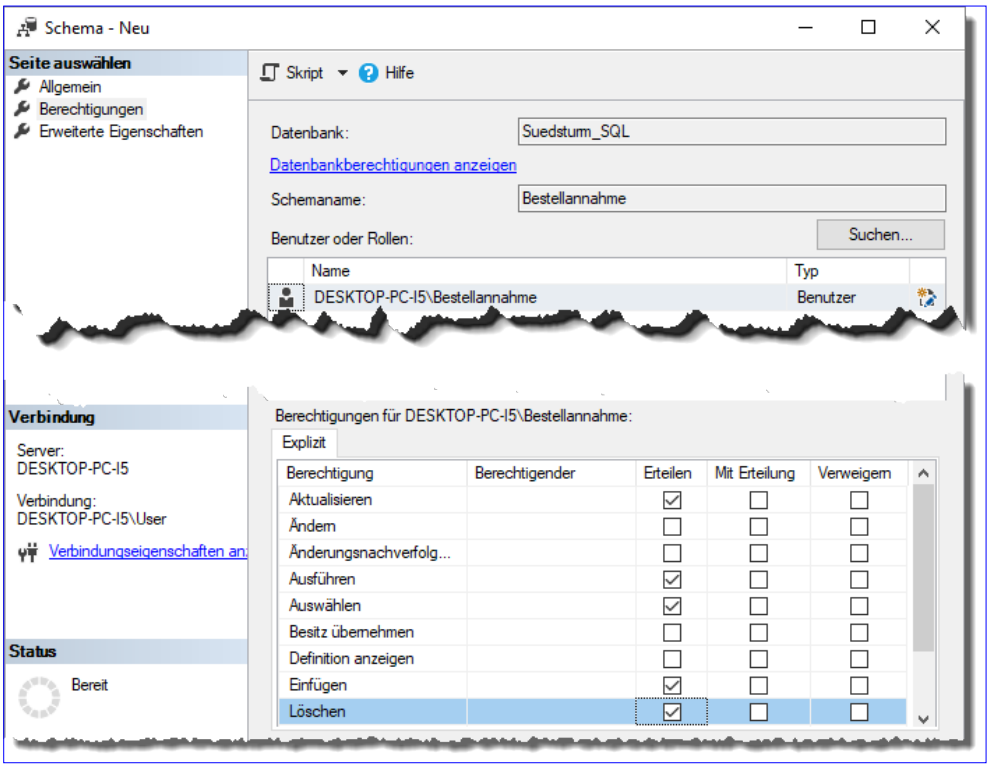


Bild 4: Festlegen der Berechtigungen für das Schema **Bestellannahme**

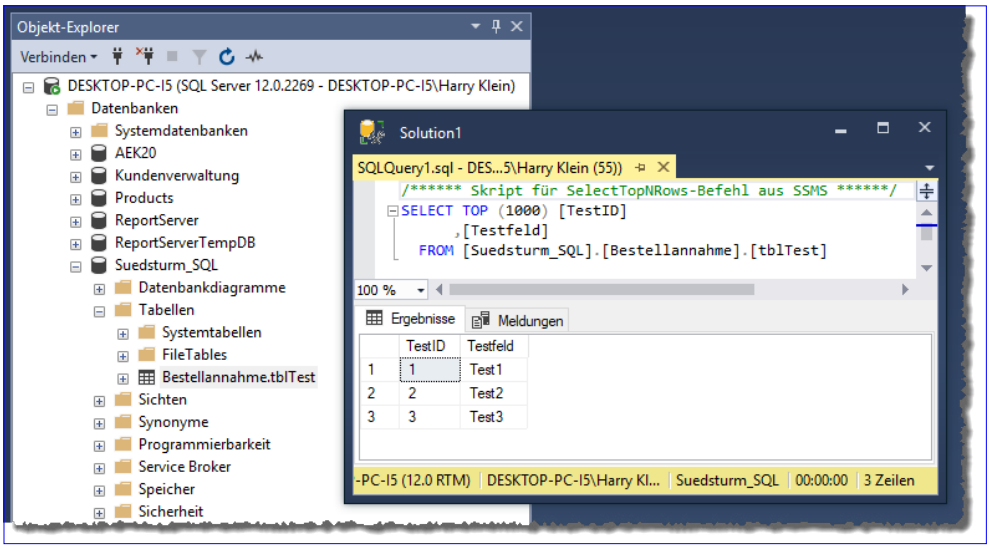


Bild 5: Zugriff auf die Daten des Schemas

Benutzer der Windows-Gruppe **Bestellannahme**, die über die Anmeldung dem Schema zugeordnet wurden, können nun auf die Elemente dieses Schemas zugreifen. Wenn ein Windows-Benutzer als neuer Benutzer zu dieser Windows-Benutzergruppe hinzugefügt wird, kann er ebenfalls auf die Elemente des Schemas zugreifen.

Objekte zu Schema hinzufügen

Andersherum greifen die Mitglieder der Windows-Benutzergruppe **Bestellannahme** automatisch auf alle Tabellen, Sichten, gespeicherte Prozeduren und andere Elemente zu, die im Kontext des Schemas erstellt werden. Bisher allerdings gehören noch keine Elemente zu diesem Schema, was wir auch kurz ausprobieren können.

Wenn die Datenbank **Suedsturm_SQL** die Benutzergruppe **Bestellannahme** bereits enthielt, müssen Sie diese alle Berechtigungen an der Datenbank **Suedsturm_SQL** entziehen, also nicht nur die Berechtigungen, die im Dialog Datenbankbenutzer für die Anmeldung **Bestellannahme** festgelegt waren, sondern auch die Mitgliedschaften in anderen Schemas wie **db_datareader** oder **db_datawriter**.

Wenn Sie sich im Kontext eines der Benutzer der Windowsgruppe **Bestellannahme** wie etwa **Harry Klein** den SQL Server Management Studio starten anmelden, sollte für diesen Benutzer zu diesem Zeitpunkt noch nicht einmal eine der Tabellen der Datenbank **Suedsturm_SQL** zu sehen sein. Dies sollte sich erst ändern, wenn Sie dem Schema eine Tabelle hinzufügen. Der wichtige Unterschied gegenüber dem bisherigen Hinzufügen einer Tabelle ist, dass wir nun statt **dbo** als Objektbesitzer den Namen des Schemas angeben, hier also **Bestellannahme**:

```
CREATE TABLE Bestellannahme.tblTest
(
    TestID int IDENTITY(1,1) PRIMARY KEY,
        Testfeld nvarchar(max)
)
```

Anschließend fügen wir noch ein paar Datensätze zu der neuen Tabelle hinzu:

```
INSERT INTO Bestellannahme.tblTest(Testfeld)
VALUES('Test1'), ('Test2'), ('Test3')
```

Wenn wir nun wieder im Kontext des Benutzers **Harry Klein** beziehungsweise der Benutzergruppe **Bestellannahme** auf die Tabellen der Datenbank zugreifen, sollte die neue Tabelle dort im Objekt-Explorer erscheinen.

Und genau das ist der Fall: Es erscheint genau diese eine Tabelle und wir können auch im Kontext dieses Benutzers über das Schema auf die Daten dieser Tabelle zugreifen (siehe Bild 5).

Wenn Sie Tabelle über den Tabellenentwurf anlegen wollen, geben Sie den Namen des Schemas über die Eigenschaft **Schema** des Entwurfs an (siehe Bild 6).

Berechtigungen für Access-Objekte per SQL Server II: Formulare

Im Beitrag »Berechtigungen für Access-Objekte per SQL Server I: Datenmodell« haben wir ein Datenmodell entwickelt für die Verwaltung der Berechtigungen verschiedener Benutzergruppen auf die Formulare, Berichte und Steuerelemente einer Access-Anwendung – gesteuert über die jeweilige Anmeldung an der SQL Server-Datenbank. Im zweiten Teil dieser Beitragsreihe verknüpfen wir die Access-Anwendung mit diesen Tabellen und erstellen die Formulare, die zur Bearbeitung der für die Berechtigungsverwaltung notwendigen Tabellen erforderlich sind.

Für die im ersten Teil dieser Beitragsreihe namens **Berechtigungen für Access-Objekte per SQL Server I: Datenmodell**, die Sie unter **www.access-im-unternehmen.de/1159** finden, benötigen wir prinzipiell zwei verschiedene Datenbanken: eine, mit der wir die Objektberechtigungen verwalten und eine, welche die Berechtigungen umsetzt. Die Formulare, mit denen wir die Objekte, Benutzergruppen und Berechtigungen einander zuordnen wollen, müssen nicht unbedingt in der eigentlichen Anwendung stecken. Andererseits kann es auch nicht schaden, damit gleich auszuprobieren, ob die Berechtigungen funktionieren – denn diese sollen weder durch Mitglieder der Gruppe **Bestellannahme** noch der Gruppe **Management** verwendet werden, sondern nur von den Administratoren. Wir werden also noch eine dritte Gruppe namens **Berechtigungsadministratoren** zu Windows hinzufügen, die wir dann auch in den SQL Server übernehmen. Das erledigen Sie in der Eingabeaufforderung wie folgt:

```
net.exe LOCALGROUP /ADD Berechtigungsadministratoren
```

Im SQL Server Management Studio fügen wir diese Anmeldung wie folgt hinzu (Details siehe Beitrag **SQL Server: Sicherheit und Benutzerverwaltung**, **www.access-im-unternehmen.de/1154**) – **<Server>** muss durch den entsprechenden Servernamen ersetzt werden:

```
USE [master]
GO
CREATE LOGIN [<Server>\Berechtigungsadministratoren] FROM
WINDOWS WITH DEFAULT_DATABASE=[Suedsturm_SQL]
GO
USE [Suedsturm_SQL]
GO
CREATE USER [<Server>\Berechtigungsadministratoren] FOR
LOGIN [<Server>\Berechtigungsadministratoren]
GO
ALTER ROLE [db_datareader] ADD MEMBER [<Server>\Berechti-
gungsadministratoren]
GO
ALTER ROLE [db_datawriter] ADD MEMBER [<Server>\Berechti-
gungsadministratoren]
GO
```

Auf den folgenden Seiten zeigen wir Ihnen, wie Sie erstens die Verknüpfung zu den relevanten Tabellen herstellen – also zu den Tabellen, die wir im ersten Teil entworfen haben –, und dann die Formulare programmieren, die zur Verwaltung der Berechtigungen erforderlich sind.

Verknüpfung herstellen

Als Erstes benötigen wir eine Verknüpfung von einer Access-Datenbank zu den im ersten Teil der Beitragsreihe erstellten Tabellen. Dazu bemühen wir einfache ODBC-Verknüpfungen. Diese legen wir mit einem Tool an, dass wir bereits im Beitrag **SQL Server-Tools** vorgestellt haben

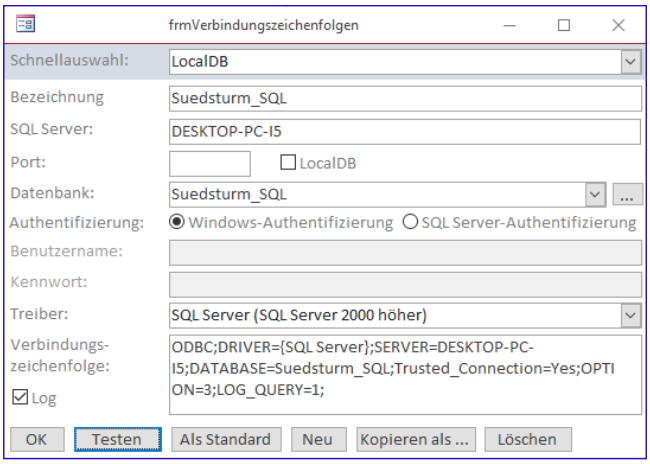


Bild 1: Einrichten der Verbindungszeichenfolge

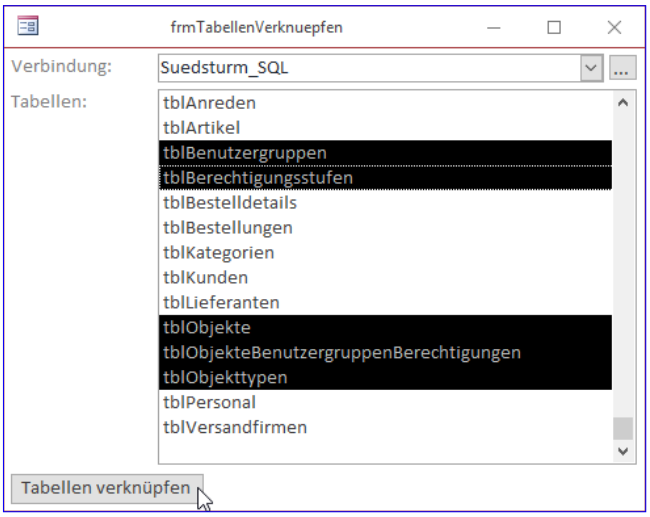


Bild 2: Hinzufügen der Verknüpfungen

(**http://www.access-im-unternehmen.de/1061**). Genaugenommen können wir die dort erstellte Datenbank auch als Grundlage für unsere Datenbank zur Verwaltung der Berechtigungen nutzen, denn sie enthält einige praktische Werkzeuge, die wir noch brauchen.

Das beginnt mit dem Formular **frmVerbindungszeichenfolgen** aus Bild 1, mit dem wir uns gleich eine passende Verbindungszeichenfolge zusammenstellen.

Nachdem wir dies erledigt haben, können wir gleich das Formular **frmTabellenVerknuepfen** öffnen, mit der wir die soeben angelegte Verbindung auswählen. Es erscheinen alle Tabellen der Datenbank **Suedsturm_SQL**, von denen wir die Tabellen **tblBenutzergruppen**, **tblBerechtigungsstufen**, **tblObjekte**, **tblObjekteBenutzergruppenBerechtigungsen** und **tblObjekttypen** auswählen – bei gedrückter Umschalttaste (siehe Bild 2). Diese Tabellen erscheinen kurz darauf im Navigationsbereich der Datenbank.

Warum ODBC-Verknüpfungen?

Aus Performance-Gesichtspunkten wäre es günstiger, über Abfragen und gespeicherte Prozeduren auf die Tabellen der SQL Server-Datenbank zuzugreifen. Allerdings werden wir nicht so viele Objekte, Benutzer oder Berechtigungen anlegen, dass die Performance bei Nutzung von ODBC-Verknüpfungen in die Knie geht – und wir werden auch nicht mit vielen Benutzern gleichzeitig auf diese Daten zugreifen.

Also ist die einfache Variante akzeptabel. Wenn wir später die Berechtigungen beim Anzeigen des Ribbons oder beim Öffnen von Formularen und Berichten abfragen, werden wir auf die performantere Variante zurückgreifen.

Tabellen füllen

Im ersten Teil dieser Beitragsreihe haben wir die Tabellen vorgestellt, mit denen wir die Objekte, die Berechtigungsstufen, die Berechtigungen und die Objekttypen speichern wollen. Nun kann es bei einer umfangreichen Datenbank relativ aufwendig werden, alle Objekte wie Formulare und Berichte sowie die enthaltenen Steuerelemente in die Tabelle **tblObjekte** zu schreiben, um dann über die anderen Tabellen die Berechtigungen zuzuordnen.

Also programmieren wir uns eine Prozedur, mit der wir die Formulare und Berichte sowie die enthaltenen Steuerelemente in die Tabelle **tblObjekte** schreiben können.

Diese heißt **ObjekteSchreiben** und erwartet einen optionalen Parameter, mit dem Sie den Namen eines oder mehrerer Formulare angeben können, die nicht von der Prozedur berücksichtigt werden sollen, die also nicht samt Steuerelementen in die Tabelle **tblObjekte** geschrieben werden sollen. Der Aufruf der Prozedur sieht also so aus, wenn alle Formulare berücksichtigt werden sollen:

```
Public Sub ObjekteSchreiben(ParamArray strNicht() As Variant)
    Dim db As DAO.Database, frm As Form, strForm As String
    Dim i As Integer, j As Integer, ctl As Control
    Dim bolNicht As Boolean, strControl As String, rst As DAO.Recordset
    Set db = CurrentDb
    For i = 0 To CurrentProject.AllForms.Count - 1
        strForm = CurrentProject.AllForms(i).Name
        bolNicht = False
        For j = LBound(strNicht) To UBound(strNicht)
            If strForm = strNicht(j) Then
                bolNicht = True
                Exit For
            End If
        Next j
        If Not bolNicht Then
            DoCmd.OpenForm strForm, acDesign
            Set frm = Forms(strForm)
            On Error Resume Next
            db.Execute "INSERT INTO tblObjekte(Bezeichnung, ObjekttypID) VALUES('" & strForm & "', 1)", dbFailOnError
            db.Execute "INSERT INTO tblObjekte(Bezeichnung, Uebergeordnet, ObjekttypID) VALUES('Form', '" _
                & strForm & "', 4)", dbFailOnError
            For Each ctl In frm.Controls
                If Not ctl.ControlType = 100 Then
                    strControl = ctl.Name
                    db.Execute "INSERT INTO tblObjekte(Bezeichnung, Uebergeordnet, ObjekttypID, SteuerelementtypID) " _
                        & "VALUES('" & strControl & "', '" & strForm & "', 4, " & ctl.ControlType & ")", dbFailOnError
                End If
            Next ctl
            On Error GoTo 0
            Set rst = db.OpenRecordset("SELECT * FROM tblObjekte WHERE Uebergeordnet = '" & strForm _
                & "' AND ObjekttypID = 4 AND NOT Bezeichnung = 'Form'", dbOpenDynaset, dbSeeChanges)
            Do While Not rst.EOF
                Set ctl = Nothing
                On Error Resume Next
                Set ctl = frm.Controls(rst!Bezeichnung)
                Debug.Print Err.Number, Err.Description
                On Error GoTo 0
                If ctl Is Nothing Then
                    db.Execute "DELETE FROM tblObjekte WHERE ObjektID = " & rst!ObjektID, dbFailOnError Or dbSeeChanges
                End If
                rst.MoveNext
            Loop
            DoCmd.Close acForm, strForm
        End If
    Next i
    FormularePruefen
End Sub
```

Listing 1: Prozedur zum Einlesen der Formulare und Steuerelemente in die Tabelle **tblObjekte**

```
ObjekteSchreiben
```

Wenn Sie die Prozedur, wie wir es später tun werden, von einem Formular wie **frmObjektberechtigungen** aus aufrufen werden, können Sie dieses auf folgende Weise ausschließen:

```
ObjekteSchreiben "frmObjektberechtigungen"
```

Wenn Sie mehr als ein Formular ausschließen wollen, geben Sie die Namen der Formulare einfach in einer durch Kommata getrennten Liste an:

```
ObjekteSchreiben "frmObjektberechtigungen", "frmArtikel", ...
```

Die Prozedur **ObjekteSchreiben** finden Sie in Listing 1. Damit Sie kein, ein oder mehrere Formulare von der Verarbeitung ausschließen können, verwenden wir ein **ParamArray** als Parameter. In einer **For...Next**-Schleife durchläuft die Prozedur alle Formulare der aktuellen Anwendung. Dabei ermitteln wir die Anzahl der zu durchlaufenden Formulare über die **Count**-Eigenschaft der Auflistung **CurrentProject.AllForms**.

Danach erfassen wir den Namen des ersten Formulars und speichern diesen in der Variablen **strForm**. Die **Boolean**-Variable **bolNicht** stellen wir auf den Wert **False** ein, da wir in der folgenden Schleife alle Elemente des per Parameter übergebenen ParamArrays durchlaufen und prüfen, ob das in **strForm** gespeicherte Formular einem der auszuschließenden Formulare entspricht. Dies erledigen wir in einer weiteren **For...Next**-Schleife, diesmal mit der Laufvariablen **j**.

In dieser Schleife durchlaufen wir alle Elemente des ParamArrays und referenzieren diese über den mit **j** ermittelten Indexwert. Stimmen **strForm** und das aktuelle Element aus **strNicht(j)** überein, stellen wir **bolNicht** auf **True** ein und verlassen die Schleife. Erhalten wir dabei den Wert **True** für die Variable **bolNicht**, dann wird der Inhalt der folgenden **If...Then**-Bedingung nicht ausgeführt, denn das aktuelle Formular samt Steuerelementen soll nicht eingelesen werden.

Anderenfalls beginnt hier jedoch der Einlesevorgang. Dabei öffnen wir das Formular aus **strForm** in der Entwurfsansicht und referenzieren es mit der Objektvariablen **objForm**. Danach deaktivieren wir die eingebaute Fehlerbehandlung, da es sein kann, dass beim Versuch, das aktuelle Formular als neuen Datensatz in die Tabelle **tblObjekte** zu schreiben, ein Fehler wegen einer Schlüsselverletzung ausgelöst wird.

Dies kommt vor, wenn Sie beispielsweise neue Formulare oder Steuerelemente hinzugefügt haben und diese Elemente erneut einlesen. Wir haben der Prozedur keine Überprüfung hinzugefügt, mit der wir das Vorhandensein eines Formulars oder Steuerelements prüfen, sondern lassen es auf den Fehler ankommen, den wir dann ignorieren. Die **INSERT INTO**-Anweisung schreibt den Namen des Formulars in das Feld **Bezeichnung** eines neuen Datensatzes der Tabelle **tblObjekte**.

Außerdem trägt es dort den Wert **1** für Formular in das Feld **ObjekttypID** ein. Die zweite **INSERT INTO**-Anweisung trägt einen zweiten Datensatz für dieses Formular ein, wobei allerdings das Feld **Bezeichnung** mit dem Wert **Form** und das Feld **Uebergeordnet** mit dem Namen des Formulars gefüllt wird. Wozu wir das benötigen, erfahren Sie weiter unten.

Danach durchlaufen wir in einer **For Each**-Schleife alle in dem mit **frm** referenzierten Formulare enthaltenen Steuerelemente. Dabei prüfen wir zunächst, ob es sich nicht um ein Bezeichnungsfeld handelt – diese sollen hier nicht berücksichtigt werden. Ist das aktuelle Steuerelement kein Bezeichnungsfeld, schreiben wir zunächst den Namen des aktuellen Steuerelements in die Variable **strControl**.

Danach fügen wir, wiederum bei deaktivierter Fehlerbehandlung, einen neuen Datensatz für das aktuelle Steuerelement in die Tabelle **tblObjekte** ein. Diesmal landet

```
Public Sub FormularePruefen()
    Dim db As DAO.Database
    Dim i As Integer
    Dim bolBehalten As Boolean
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblObjekte WHERE ObjekttypID = 1", dbOpenDynaset, dbSeeChanges)
    Do While Not rst.EOF
        bolBehalten = False
        For i = 0 To CurrentProject.AllForms.Count - 1
            If CurrentProject.AllForms(i).Name = rst!Bezeichnung Then
                bolBehalten = True
                Exit For
            End If
        Next i
        If bolBehalten = False Then
            db.Execute "DELETE FROM tblObjekte WHERE Uebergeordnet = '" & rst!Bezeichnung _
                & "'", dbFailOnError Or dbSeeChanges
            db.Execute "DELETE FROM tblObjekte WHERE ObjektID = " & rst!ObjektID, dbFailOnError Or dbSeeChanges
        End If
        rst.MoveNext
    Loop
End Sub
```

Listing 2: Prozedur zum Prüfen und gegebenenfalls Löschen von Einträgen für Formulare aus der Tabelle **tblObjekte**

der Name des Steuerelements im Feld **Bezeichnung**. Der Name des übergeordneten Formulars landet im Feld **Uebergeordnet**. Für den Objekttyp landet der Wert **4** für Steuerelement im Feld **ObjekttypID**. Außerdem speichern wir noch den Steuerelementtyp aus der Eigenschaft **ControlType**, und zwar im Feld **SteuerelementtypID**.

Danach aktivieren wir die Fehlerbehandlung wieder und öffnen ein Recordset, das alle Elemente der Tabelle **tblObjekte** enthält, deren Feld **Uebergeordnet** dem Namen des aktuell untersuchten Formulars entspricht und die im Feld **ObjekttypID** den Wert **4** aufweisen – mit Ausnahme der Datensätze, die im Feld **Bezeichnung** den Wert **Form** enthalten.

Das Recordset enthält also jeweils einen Datensatz für alle Steuerelemente des aktuellen Formulars. Damit durchlaufen wir dann in einer **Do While**-Schleife alle Datensätze dieses Recordsets. Wir leeren die Objektvariable **ctl** und versuchen dann, bei deaktivierter Fehlerbehandlung das Steuerelement mit dem Namen aus dem Feld **Bezeichnung** im Formular zu referenzieren.

Entspricht **ctl** danach dem Objekt **Nothing**, ist also leer, ist das Steuerelement zwar noch in der Tabelle **tblObjekte** gespeichert, aber nicht mehr im angegebenen Formulare vorhanden – vermutlich hat der Entwickler es in der Zwischenzeit gelöscht. In diesem Fall können wir auch den dazu gehörenden Datensatz aus der Tabelle **tblObjekte** löschen.

Danach schließen wir schließlich noch das aktuell untersuchte Formular und durchlaufen die Schleife für die weiteren Formulare, bis wir alle Formular durchlaufen haben.

Auf gelöschte Formulare prüfen

Eine weitere Prozedur, die wir mit der letzten Anweisung der zuvor beschriebenen Prozedur aufrufen, heißt **Formu-**

larePruefen (siehe Listing 2). Sie untersucht alle Einträge der Tabelle **tblObjekte**, die den Wert **1** im Feld **Objekttyp-ID** aufweisen dahingehend, ob die angegebenen Formular noch in der Datenbank vorhanden sind. Ist dies nicht der Fall, werden die Formulare-Einträge auch aus der Tabelle **tblObjekte** gelöscht – inklusive der Steuerelement-Einträge für diese Formulare.

Wenn Sie die Prozedur **ObjekteSchreiben** einmal aufgerufen haben, etwa für die Beispieldatenbank zu diesem Beitrag, finden Sie Daten wie die aus Bild 3 in der Tabelle **tblObjekte**.

Formular zum Verwalten der Berechtigungen

Das Formular zum Verwalten der Berechtigungen heißt **frmBerechtigungen**. Im linken Teil beherbergt es ein Listenfeld mit allen Formularen, die wir aus Einträgen der Tabelle **tblObjekte** ermitteln, für die das Feld **ObjektID** den Wert **1** enthält. Den Entwurf dieses Formulars finden Sie in Bild 4.

tblObjekte

ObjektID	Bezeichnung	Uebergeordnet	ObjekttypID
3332	sfmKategorienArtikel		1
3333	ArtikelID	sfmKategorienArtikel	4
3334	Bezeichnungsfeld0	sfmKategorienArtikel	4
3335	Artikelname	sfmKategorienArtikel	4
3336	Bezeichnungsfeld1	sfmKategorienArtikel	4
3337	Liefereinheit	sfmKategorienArtikel	4
3338	Bezeichnungsfeld4	sfmKategorienArtikel	4
3339	Einzelpreis	sfmKategorienArtikel	4
3340	Bezeichnungsfeld5	sfmKategorienArtikel	4
3341	Lagerbestand	sfmKategorienArtikel	4
3342	Bezeichnungsfeld6	sfmKategorienArtikel	4
3343	BestellteEinheiten	sfmKategorienArtikel	4
3344	Bezeichnungsfeld7	sfmKategorienArtikel	4
3345	Mindestbestand	sfmKategorienArtikel	4
3346	Bezeichnungsfeld8	sfmKategorienArtikel	4
3347	Auslaufartikel	sfmKategorienArtikel	4
3348	Bezeichnungsfeld9	sfmKategorienArtikel	4
3349	LieferantID	sfmKategorienArtikel	4
3350	Bezeichnungsfeld2	sfmKategorienArtikel	4
3351	KategorieID	sfmKategorienArtikel	4
3352	Bezeichnungsfeld3	sfmKategorienArtikel	4

Datensatz: 1 von 216 Ungefiltert Suchen

Bild 3: Daten über die mit Berechtigungen zu versehenden Objekte

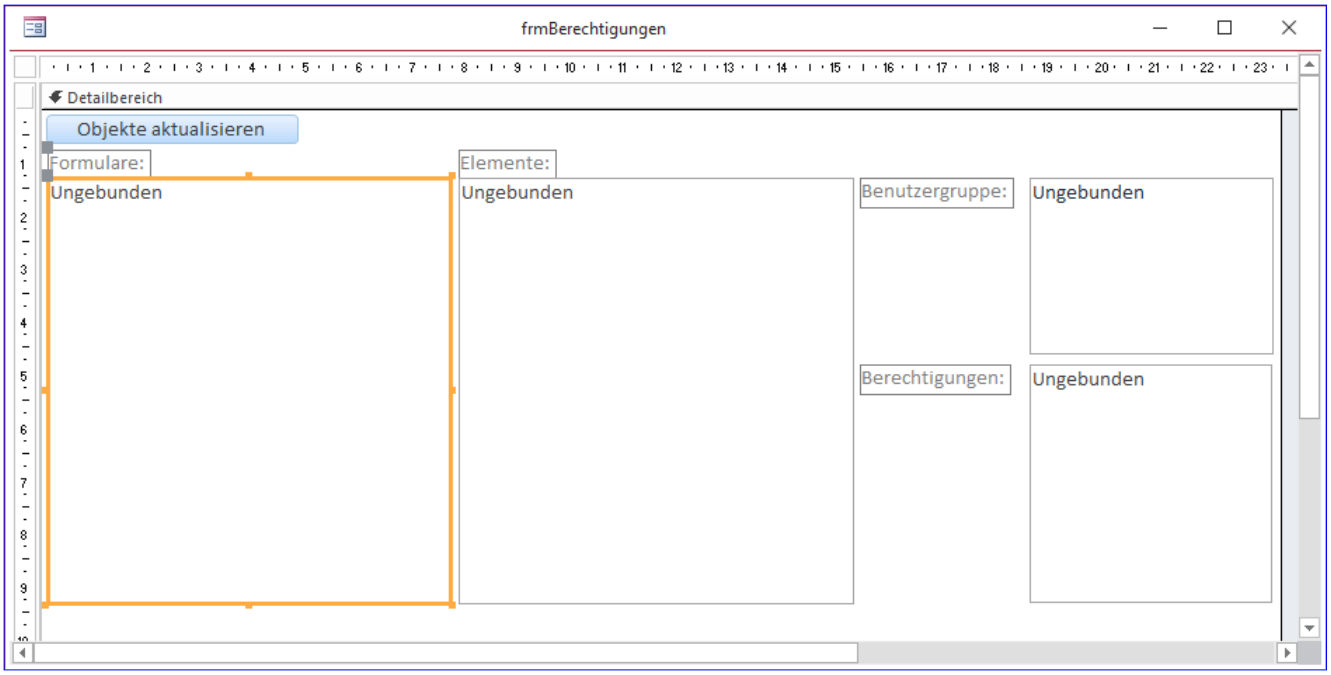


Bild 4: Entwurf des Formulars **frmBerechtigungen**

Berechtigungen für Access-Objekte per SQL Server III: Anwenden

Im Beitrag »Berechtigungen für Access-Objekte per SQL Server I: Datenmodell« haben wir ein Datenmodell entwickelt für die Verwaltung der Berechtigungen verschiedener Benutzergruppen auf die Formulare, Berichte und Steuerelemente einer Access-Anwendung – gesteuert über die jeweilige Anmeldung an der SQL Server-Datenbank. Im zweiten Teil dieser Beitragsreihe haben wir das Formular zum Einstellen der Berechtigungen erstellt. Im dritten Teil zeigen wir nun, wie wir die gespeicherten Berechtigungen nutzen, um Formulare und Steuerelemente vor dem unbefugten Zugriff zu schützen.

Berechtigungen anwenden

Damit ist der erste Teil abgeschlossen – wir haben eine Möglichkeit geschaffen, mit der wir die Formulare einer Anwendung und die enthaltenen Steuerelemente einlesen können. Außerdem können wir die Benutzergruppen, die am SQL Server in Form von Anmeldungen vorliegen, ermitteln und für die Kombination aus Elementen und Benutzergruppen eine der drei Berechtigungsstufen **Keine**, **Lesen** oder **Alle** einstellen. Nun ändert dies allein noch nichts im Umgang mit den Elementen der Benutzeroberfläche, denn noch wissen diese ja nichts davon, dass der Zugriff auf die jeweiligen Formulare oder Steuerelemente gegebenenfalls eingeschränkt ist. Die Interpretation der drei Berechtigungsstufen sieht für Formulare wie folgt aus:

- **Keine**: Das Formular darf nicht geöffnet werden.
- **Lesen**: Das Formular darf geöffnet werden, aber nur zum Lesen von Daten.
- **Alle**: Das Formular darf geöffnet und auch zum Ändern, Anlegen oder Löschen von Daten verwendet werden.

Für die Steuerelemente sollen die drei so auswirken:

- **Keine**: Das Steuerelement ist deaktiviert.
- **Lesen**, **Alle**: Das Steuerelement ist aktiviert.

Um ein Formular und seine Steuerelemente mit den entsprechenden Werten für die betroffenen Eigenschaften zu versehen, benötigen wir noch ein wenig Code. Die Hauptfunktion dabei heißt **BerechtigungenAnwenden** und erwartet einen Verweis auf das zu öffnende Formular als Parameter. Diese Prozedur erläutern wir weiter unten. Bevor Sie diese Prozedur nutzen können, sind nämlich noch einige Vorbereitungen nötig – siehe weiter unten.

Berechtigungen ermitteln per gespeicherter Prozedur

Die Prozedur verwendet die Funktion **SPRecordsetMitParameter** (siehe weiter unten), um eine gespeicherte Prozedur in der SQL Server-Datenbank auszuführen und ein darauf basierendes Recordset zu erstellen. Diese gespeicherte Prozedur müssen wir zunächst anlegen, und zwar wahlweise im SQL Server Management Studio oder im Formular **frmSQLBefehle**, das wir im Beitrag **SQL Server Tools** vorgestellt haben (**www.access-im-unternehmen.de/1061**).

Anschließend steht diese gespeicherte Prozedur für die Verwendung in der Datenbank **Suedsturm_SQL** zur Verfügung. Sie können diese ausprobieren, werden aber gegebenenfalls keinen Datensatz als Ergebnis erhalten. Wenn Sie die im zweiten Teil dieser Beitragsreihe erstellten Beispielberechtigungen für das Formular **frmIntro** für die beiden Gruppen **Bestellannahme** und

Management testen wollen, sollten Sie sich bewusst sein, unter welchem Benutzerkonto Sie gerade am Windows-Rechner angemeldet sind und ob dieses Benutzerkonto einer dieser beiden Gruppen zugehört. Falls ja, werden Sie einen entsprechenden Datensatz erhalten, den wir uns gleich ansehen werden. Falls nicht, müssen wir zunächst die Voraussetzungen schaffen, damit Sie sich unter einer der beiden Benutzergruppen an der SQL Server-Datenbank anmelden. Dazu gibt es wiederum zwei Möglichkeiten:

- Sie melden sich unter einem geeigneten Benutzerkonto am System an oder
- Sie starten die Access-Anwendung im Kontext des gewünschten Benutzerkontos.

Wir wollen testweise, sofern noch nicht vorhanden, die Benutzergruppe **Management** mit dem Benutzer **Peter Gross** und die Benutzergruppe **Bestellannahme** mit dem Benutzer **Harry Klein** anlegen. Das erledigen Sie über die Eingabeaufforderung ganz schnell mit den folgenden Anweisungen. Die Benutzerkonten fügen Sie so hinzu:

```
net user /add "Peter Gross"
net user /add "Harry Klein"
```

Die Gruppen legen Sie so an:

```
net localgroup /add Bestellannahme
net localgroup /add Management
```

Schließlich fügen Sie die Benutzer noch den Gruppen hinzu:

```
net localgroup Management /add "Peter Gross"
net localgroup Bestellannahme /add "Harry Klein"
```

Dass die Gruppen und Benutzer angelegt wurden, können Sie auch prüfen. Geben Sie den folgenden Befehl ein, um alle Benutzer auszugeben:

```
net user
```

Mit dem nächsten Befehl geben Sie alle Benutzergruppen aus:

```
net localgroup
```

Wenn Sie die Details einer Gruppe inklusive Benutzer ermitteln wollen, gelingt dies durch Angabe des Gruppennamens:

```
net localgroup Management
```

Und auch über die Benutzerdetails können Sie die Gruppenzugehörigkeit ermitteln:

```
net user "Peter Gross"
```

Damit haben Sie die passenden Benutzerkonten und Benutzergruppen für die folgenden Experimente erstellt.

Außerdem müssen wir die Benutzergruppen noch als neue Anmeldung zum SQL Server hinzufügen. Das erledigen wir im SQL Server Management Studio, indem wir mit der rechten Maustaste auf das Element **<Servername>|Sicherheit|Anmeldungen** klicken und den Kontextmenü-Eintrag **Neue Anmeldung** betätigen.

Es erscheint der Dialog **Anmeldung - Neu**. Hier klicken Sie neben dem Feld **Anmeldename** auf die Schaltfläche **Suchen**. Im nächsten Dialog **Benutzer oder Gruppe auswählen** klicken Sie zunächst auf **Objekttypen**. Im Dialog **Objekttypen aktivieren** Sie, sofern noch nicht geschehen, den Eintrag **Gruppen** und schließen den Dialog wieder.

Dann klicken Sie im Dialog **Benutzer oder Gruppe auswählen** auf die Schaltfläche **Erweitert**. Im neuen Dialog gleichen Namens klicken Sie rechts auf die Schaltfläche **Jetzt suchen** und wählen dann unten aus den Suchergebnissen die Gruppe **Bestellannahme** aus. Schließen Sie den Dialog mit **OK** und den aufrufenden Dialog ebenfalls,

nachdem Sie sich vergewissert haben, dass unter **Geben Sie die zu verwendenden Objektnamen ein** die gewünschte Gruppe aufgeführt wird.

Zurück im Dialog **Anmeldung - Neu** wechseln Sie noch zum Bereich Benutzerzuordnung und aktivieren dort die Datenbank **Suedsturm_SQL**. Dann erstellen Sie Anmeldung mit einem Klick auf **OK**.

Auf die gleiche Weise verfahren Sie anschließend mit der Benutzergruppe **Management**. Sie können auch direkt im ersten Dialog **Anmeldung - Neu** die Benutzergruppe in der Form **<Servername>\<Gruppenname>** eingeben.

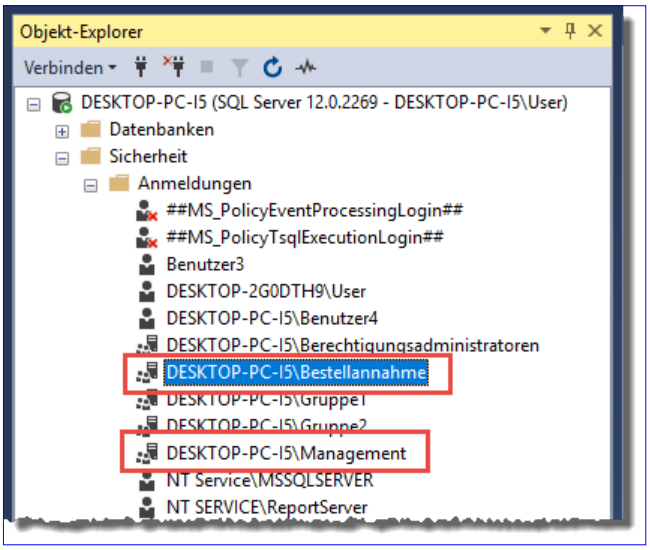


Bild 1: Einrichten der Benutzergruppen als SQL Server-Anmeldungen

```
CREATE PROCEDURE [dbo].[spObjektberechtigungen]
    @Objekt varchar(255)
AS
SELECT
(
    SELECT COUNT(*)
    FROM dbo.tblObjekteBenutzergruppenBerechtigungen t1 INNER JOIN dbo.tblBenutzergruppen t2
    ON t1.BenutzergruppeID = t2.BenutzergruppeID
    WHERE t1.ObjektID = t4.ObjektID AND IS_MEMBER(t2.Bezeichnung) = 1
) AS AnzahlFestgelegterBerechtigungen,
(
    SELECT COUNT(*)
    FROM dbo.tblBenutzergruppen t3
    WHERE IS_MEMBER(t3.Bezeichnung) = 1
) AS AnzahlGruppenDesBenutzers,
t4.ObjektID, t6.Bezeichnung, MAX(t4.BerechtigungsstufeID) AS MaxBerechtigungID
FROM
(
    (
        dbo.tblObjekteBenutzergruppenBerechtigungen t4 INNER JOIN dbo.tblBerechtigungsstufen t5
            ON t4.BerechtigungsstufeID = t5.BerechtigungsstufeID
    )
    INNER JOIN dbo.tblObjekte t6 ON t4.ObjektID = t6.ObjektID
)
INNER JOIN dbo.tblBenutzergruppen t7 ON t4.BenutzergruppeID = t7.BenutzergruppeID
WHERE t6.Uebergeordnet = @Objekt
AND IS_MEMBER(t7.Bezeichnung) = 1
GROUP BY t6.Bezeichnung, t4.ObjektID;
```

Listing 1: Gespeicherte Prozedur, welche die Berechtigungen für den aktuellen Benutzer für das übergebene Formular ermittelt.

Danach sollten Sie im Bereich **Sicherheit|Anmeldungen** die beiden neuen Anmeldungen mit den Benutzergruppen vorfinden (siehe Bild 1).

Gespeicherte Prozedur spObjektberechtigungen erstellen

Bevor wir uns gleich im Kontext eines der beiden Benutzer Peter Gross oder Harry Klein am SQL Server anmelden, erstellen wir noch die gespeicherte Prozedur, die wir dort ausprobieren wollen. Den dazu notwendigen Code finden Sie in Listing 1 (die Erläuterung folgt weiter unten).

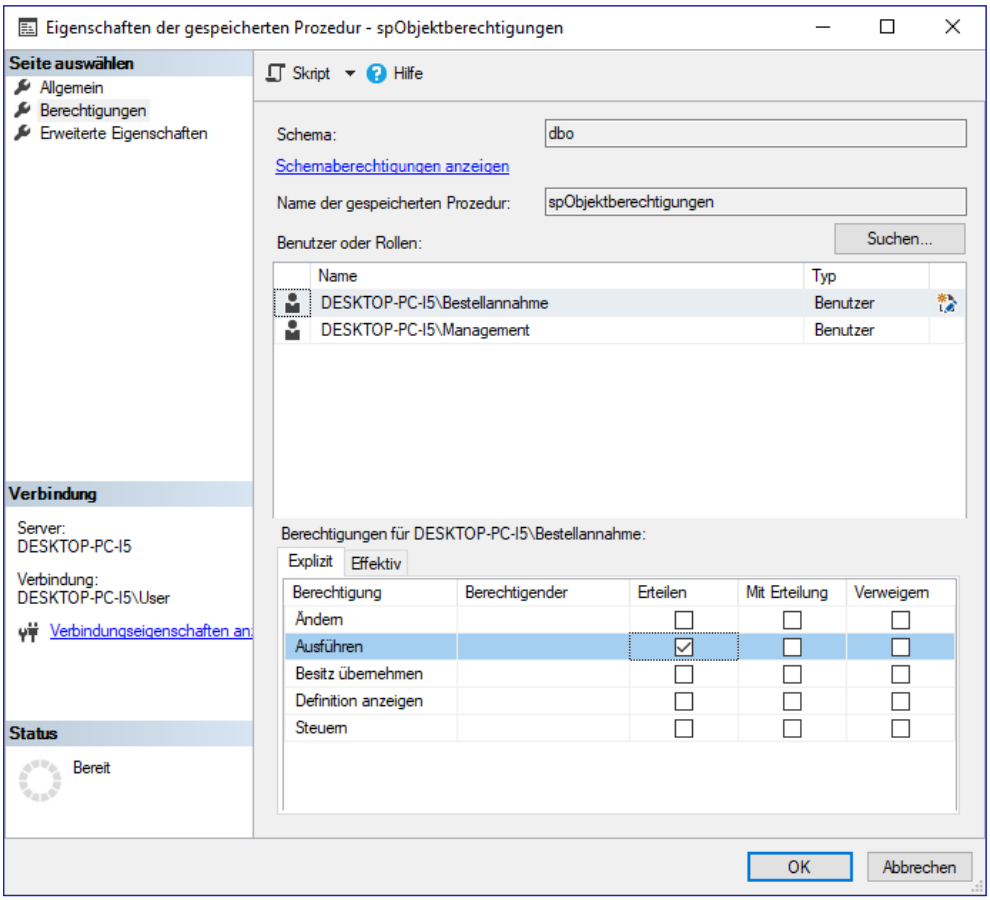


Bild 2: Erteilen der Berechtigungen für die gespeicherte Prozedur

Um die gespeicherte Prozedur anzulegen, führen Sie den **CREATE PROCEDURE**-Code in einem Abfragefenster der Datenbank **Suedsturm_SQL** aus. Nach der Aktualisierung des Elements **<Servername>|Datenbanken|Suedsturm_SQL|Programmierbarkeit|Gespeicherte Prozeduren** sollte darunter ein neuer Eintrag namens **spObjektberechtigungen** auftauchen.

Damit die neuen Benutzergruppen nun auch noch die gespeicherte Prozedur **spObjektberechtigungen** nutzen können, müssen wir diesen noch die passenden Berechtigungen geben. Dazu klicken Sie mit der rechten Maustaste auf den Eintrag für diese gespeicherte Prozedur im Objekt-Explorer und wählen aus dem Kontextmenü den Eintrag **Eigenschaften** aus. Wechseln Sie hier zum Bereich **Berechtigungen**, markieren Sie nacheinander die beiden betroffenen Gruppen **Management** und **Bestellannahme** und aktivieren Sie die Option **Erteilen** für die Berechtigung **Ausführen** (siehe Bild 2).

Aufbau der gespeicherten Prozedur spObjektberechtigungen

Diese Prozedur sieht etwas kompliziert aus, aber wir lösen das jetzt auf. Der größte Teil der **SELECT**-Anweisung ist das Konstrukt zum Herstellen der Verknüpfungen der vier beteiligten Tabellen **tblObjekteBenutzergruppenBerechtigungen**, **tblBerechtigungsstufen**, **tblObjekte** und **tblBenutzergruppen** per **INNER JOIN** – das ist der Teil von **FROM** bis **WHERE** im unteren Bereich. Dies würde grundsätzlich alle Datensätze liefern, die in den vier Tabellen angelegt wurden. Da wir nur bestimmte Berechtigungen benötigen, in diesem Fall für ein spezielles Objekt, dass wir mit dem Parameter **@Objekt** übergeben, haben wir dafür eine Bedingung in der **WHERE**-Klausel festgelegt

und noch ein weiteres. Die folgende Bedingung sorgt also dafür, dass wir nur die Berechtigungen für die Elemente erhalten, deren übergeordnetes Objekt das mit **@Objekt** übergebene Formular ist (**t6** ist ein Alias für die Tabelle **tblObjekte**):

```
t6.Uebergeordnet = 'frmIntro'
```

Außerdem möchten wir nur solche Datensätze für die aktuelle Benutzergruppe zurückliefern. Dabei unterstützt uns die T-SQL-Funktion **IS_MEMBER**. Sie erwartet den Namen einer Anmeldung, in diesem Fall etwa die Gruppe **<Servername>\Management**, und prüft, ob der Benutzer, der gerade am Windows-Rechner angemeldet ist, zu dieser Gruppe gehört. **t7** ist dabei ein Alias für die Tabelle **tblBenutzergruppen** und Bezeichnung der Name der Benutzergruppe:

```
IS_MEMBER(t7.Bezeichnung) = 1
```

Welche Werte soll die gespeicherte Prozedur nun zurückliefern? Die ersten beiden Werte sind keine einfachen Werte, die aus der per **INNER JOIN** verknüpften Kombination der vier Tabellen stammen, sondern die Ergebnisse von Unterabfragen. Hier ist die erste Unterabfrage:

```
(
    SELECT COUNT(*)
    FROM dbo.tblObjekteBenutzergruppenBerechtigungen t1
    INNER JOIN dbo.tblBenutzergruppen t2
    ON t1.BenutzergruppeID = t2.BenutzergruppeID
    WHERE t1.ObjektID = t4.ObjektID
    AND IS_MEMBER(t2.Bezeichnung) = 1
) AS AnzahlFestgelegterBerechtigungen
```

Diese Abfrage bestimmt die Anzahl der Datensätze der beiden per **INNER JOIN** verknüpften Tabellen **tblObjekteBenutzergruppenBerechtigungen** und **tblBenutzergruppen**, wobei hier nur die Datensätze berücksichtigt werden sollen, deren **ObjektID** der Tabelle **tblObjekteBenutzergruppenBerechtigungen** auch in der gleichnamigen Tabelle der Hauptabfrage vorkommen. Da wir zweimal die gleiche Tabelle referenzieren, haben wir diesen jeweils einen Alias (hier **t1** und **t4**) vergeben und dies auch direkt für alle referenzierten Tabellen der Haupt- und der beiden Unterabfragen erledigt. Das Ergebnis dieser Unterabfrage, das in Form des Feldes **AnzahlFestgelegterBerechtigungen** in der Hauptabfrage landet, entspricht der Anzahl der expliziten Berechtigungen an diesem Objekt für den aktuellen Benutzer. Wenn der Benutzer etwa in einer Benutzergruppe ist, die eine Lesen-Berechtigung für ein Formular hat, liefert **AnzahlFestgelegterBedingungen** den Wert **1** zurück. Ist der Benutzer in zwei Gruppen, von denen die eine Lesen-Rechte und die andere das Recht **Keine** hat, liefert **AnzahlFestgelegterBedingungen** den Wert **2**. Ist der Benutzer aber in einer Gruppe, die Lesen-Rechte hat und in einer anderen Gruppe, für die keine expliziten Rechte für das Formular festgelegt sind, dann liefert **AnzahlFestgelegterBedingungen** den Wert **1**. Wozu das dient, wird gleich deutlich.

Zunächst schauen wir uns noch die zweite Unterabfrage an. Diese lautet wie folgt:

```
(
    SELECT COUNT(*)
    FROM dbo.tblBenutzergruppen t3
    WHERE IS_MEMBER(t3.Bezeichnung) = 1
) AS AnzahlGruppenDesBenutzers
```

Diese Unterabfrage gibt einfach nur die Anzahl der Gruppen des Benutzers zurück. Sie zählt alle Einträge der Tabelle **tblBenutzergruppen**, für welche die Funktion **IS_MEMBER** den Wert **1** zurückgibt.

Der Sinn der beiden Unterfunktionen ist folgender: Wenn die Anzahl von **AnzahlFestgelegterBerechtigungen** und **AnzahlGruppenDesBenutzers** gleich groß sind, können wir schlussfolgern, dass für jede der Benutzergruppen des Benutzers eine explizite Berechtigung an dem jeweiligen Element festgelegt wurde. Wenn **AnzahlFestgelegterBerechtigungen** kleiner ist als **AnzahlGruppenDesBe-**