AUSGABE 06/2019

DATENBANK ENTWICKLER

MAGAZIN FÜR DIE DATENBANKENTWICKLUNG MIT VISUAL STUDIO FÜR DESKTOP, WEB UND CO.

TOP-THEMEN:

André Minhorst Verlag

Spaltenbreiten von zwei Grid-Elementen anpassen

Im Grid-Steuerelement kann man nicht wie etwa unter Excel zwei Spalten verbinden. Die Alternative ist, zwei Grid-Steuerelemente untereinander anzuordnen und die Inhalte, die mehr als eine Grid-Spalte enthalten, zwischen den Grids zu platzieren.

Bild 1 zeigt, wie die Konstellation aussieht, die wir ändern wollen. Die zweite und die vierte Zeile mit den Textfeldern sollen so gestaltet werden, dass die Textfelder gleich ausgerichtet sind.

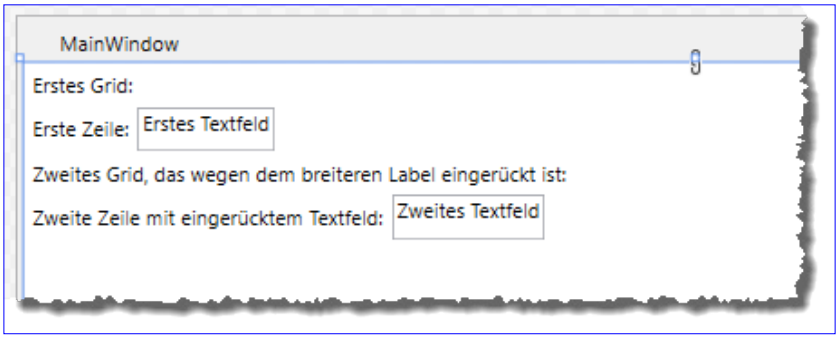

Bild 1: Nicht synchrone Spalten

Aktuell sieht der Code wie folgt aus und wir nutzen darin einfach ein StackPanel, das untereinander den Text über dem ersten Grid, ein Grid mit Label und Textfeld und darunter nochmal die gleiche Konstellation enthält:

```
<StackPanel>
    <Label>Erstes Grid:</Label>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Label>Erste Zeile:</Label>
        <TextBox Grid.Column="1">Erstes Textfeld</TextBox>
    </Grid>
    <Label>Zweites Grid, das wegen dem breiteren Label eingerückt ist:</Label>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Label>Zweite Zeile mit eingerücktem Textfeld:</Label>
        <TextBox Grid.Column="1">Zweites Textfeld</TextBox>
    </Grid>
</StackPanel>
```

Fenster mehrfach öffnen

Wenn Sie in einem Übersichtsfenster eine Liste von Datensätzen anzeigen, beispielsweise von Kunden, verwenden Sie vermutlich auch ein weiteres Fenster, indem Sie die Details eines Kunden anzeigen können. Mit einem gleichzeitig geöffneten Kundendetail-Fenster ist die Handhabung recht einfach – vor allem, wenn Sie dieses Fenster als modalen Dialog öffnen. Interessanter wird es, wenn Sie die Möglichkeit bieten wollen, mehr als einen Kunden gleichzeitig im Detailfenster anzuzeigen. Dann müssen Sie schon kontrollieren, welche Kunden bereits angezeigt werden und welche wieder geschlossen wurden. Dieser Artikel zeigt, wie Sie das mehrfache Öffnen eines Fensters realisieren und wie Sie die geöffneten Fenster vom öffnenden Fenster aus verwalten können.

Fenster mit Übersichtsliste

Als Erstes benötigen wir für unser Beispiel ein Fenster, das ein **ListBox**-Steuerelement zur Anzeige der Kundenübersicht enthält. Dieses sieht im Entwurf wie in Bild 1 aus. Das **ListBox**-Steuerelement soll die Daten einer Auflistung namens **Kunden** anzeigen, und von diesem das Feld **Nachname** verwenden.

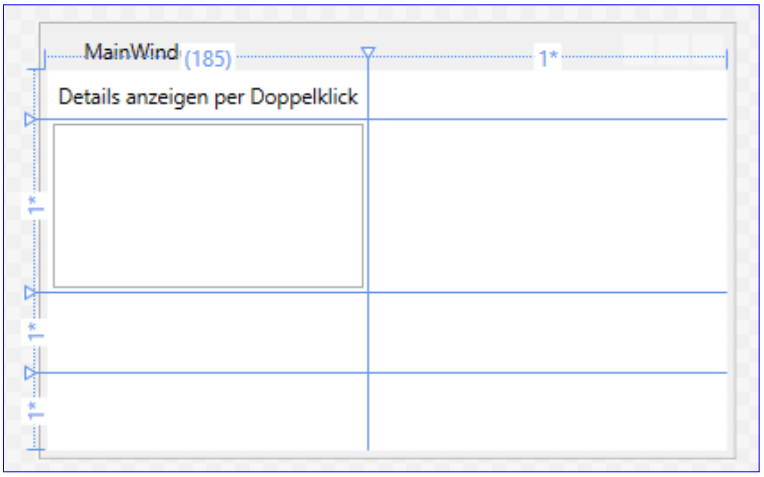

Bild 1: Fenster mit **ListBox** für die Auswahl von Kunden

Dazu richten wir dieses mit den folgenden XAML-Definitionen ein. Die Definition der **ListBox** enthält unter **ItemsSource** die Angabe der Auflistung, aus der die Daten stammen, mit **DisplayMemberPath** die Eigenschaft der Elemente der Auflistung, die angezeigt werden soll und mit **MouseDoubleClick** die Angabe einer Methode, die beim Doppelklicken auf einen der Einträge ausgelöst werden soll:

```
<Label>Details anzeigen per Doppelklick</Label>
<ListBox x:Name="lstKunden" Grid.Row="1" ItemsSource="{Binding Kunden}" DisplayMemberPath="Nachname"
MouseDoubleClick="LstKunden_MouseDoubleClick"></ListBox>
```

Die Kunden bilden wir in der folgenden Klasse ab, für die wir einen Konstruktor zur einfachen Übergabe der Daten beim Anlegen neuer Elemente hinzugefügt haben:

```
Public Class Kunde
    Public Sub New(iD As String, vorname As String, nachname As String)
        Me.ID = iD
```

```
        Me.Vorname = vorname
        Me.Nachname = nachname
    End Sub

    Public Property ID As String
    Public Property Vorname As String
    Public Property Nachname As String
End Class
```

In der Code behind-Klasse für das Hauptfenster deklarieren wir das **List**-Objekt, das die im **ListBox**-Steuerelement anzuzeigenden **Kunde**-Elemente aufnehmen soll. Außerdem initialisieren wir die Komponente in einer Konstruktor-Methode, rufen dort eine Methode zum Füllen des **List**-Objekts für die Kunden auf und stellen den **DataContext** auf das Code behind-Formular ein:

```
Class MainWindow
    Public Property Kunden As List(Of Kunde)
    Public Sub New()
        InitializeComponent()
        KundenFuellen()
        DataContext = Me
    End Sub
```

Die Methode **KundenFuellen** erstellt das **List**-Objekt für die Kunden und füllt diese mit fünf Beispielobjekten:

```
    Private Sub KundenFuellen()
        Dim Kunde As Kunde
        Kunden = New List(Of Kunde)
        Kunden.Add(New Kunde(1, "André", "Minhorst"))
        Kunden.Add(New Kunde(2, "Klaus", "Müller"))
        Kunden.Add(New Kunde(3, "Bernd", "Meier"))
        Kunden.Add(New Kunde(4, "Beate", "Klausing"))
        Kunden.Add(New Kunde(5, "Dörte", "Becker"))
    End Sub
```

Schließlich fügen wir dem Code behind-Modul noch die Methode hinzu, die durch einen Doppelklick auf einen der **ListBox**-Einträge ausgelöst wird. Diese schreibt zunächst den per **sender** übergebenen Auslöser der Prozedur in die Variable **lst**. Dann erstellt sie eine neue Instanz des Fensters **Kundendetails** und übergibt dieser das Objekt, das der Benutzer im **ListBox**-Element angeklickt hat:

```
    Private Sub LstKunden_MouseDoubleClick(sender As Object, e As MouseButtonEventArgs)
        Dim wnd As Kundendetails
        Dim lst As ListBox
```

Automatisch implementierte Eigenschaften

Bisher haben wir in vielen Artikeln Eigenschaften mit Getter und Setter definiert. Das ist in meistens sinnvoll, beispielsweise wenn Sie beim Setzen oder Lesen von Eigenschaften noch weiteren Code ausführen lassen wollen. Oft bleibt es jedoch beim einfachen Getter und Setter. In diesem Fall können Sie sich eine Menge Code sparen, indem Sie eine automatisch implementierte Eigenschaft nutzen. Dieser Artikel zeigt, wie Sie diese Art von Eigenschaften unter Visual Basic verwenden.

In vielen Beispielen haben wir für eine öffentliche Eigenschaft eine Variable wie die folgende erstellt:

```
Private _Vorname As String
```

Dazu passend haben wir die öffentliche Eigenschaft mit Getter und Setter definiert: :

```
Public Property Vorname As String
    Get
        Return _Vorname
    End Get
    Set(value As String)
        _Vorname = value
    End Set
End Property
```

Das hat durchaus seinen Sinn, wenn Sie beispielsweise für eine Klasse die **INotifyPropertyChanged**-Schnittstelle implementieren wollen. Dann benötigen Sie nämlich eine zusätzliche Anweisung, die Sie wunderbar im **Set**-Teil der öffentlichen Eigenschaft unterbringen können. Den Getter und den Setter können Sie auch für alle andere Aktionen verwenden, die beim Setzen oder Lesen der Eigenschaft ausgeführt werden sollen.

Wenn das jedoch nicht der Fall ist, können Sie sogenannte automatisch implementierte Eigenschaften verwenden. Dann würden Sie die obige Eigenschaft einfach wie folgt schreiben:

```
Public Property Vorname As String
```

Dies ist gleichbedeutend mit den oben vorgestellten Zeilen. Sie können mit automatisch implementierten Eigenschaften auch alle sonstigen Features nutzen, die Sie bei der Kombination aus privat deklarierter Variable und öffentlich deklarierter Eigenschaft definiert haben, zum Beispiel das Initialisieren mit einem bestimmten Wert. Angenommen, Sie haben zuvor das verwendet:

```
Private _Vorname As String = "André"
```

Klasse initialisieren mit Konstruktor

In der einfachsten Variante hat eine Klasse öffentliche Eigenschaften, mit denen Sie nach dem Initialisieren die Eigenschaften des erstellten Objekts festlegen können. Es gibt eine Alternative: Sie können eine Konstruktor-Methode für die Klasse definieren, der Sie die verschiedenen Eigenschaftswerte direkt in einer Anweisung übergeben. Dieser Artikel zeigt, wie das funktioniert.

Die einfachste Variante einer Klasse mit einigen Eigenschaften definieren wir wie folgt:

```
Public Class Person
    Public Property Vorname As String
    Public Property Nachname As String
    Public Property Geburtsdatum As Date
End Class
```

Diese deklarieren und initialisieren wir wie folgt und weisen den Eigenschaften die gewünschten Werte zu und geben diese anschließend in einem Meldungsfenster aus:

```
Private Sub BtnInitialisierenPerEigenschaft_Click(sender As Object, e As RoutedEventArgs)
    Dim Person As Person
    Person = New Person
    With Person
        .Vorname = "André"
        .Nachname = "Minhorst"
        .Geburtsdatum = "23.01.1971"
    End With
    MessageBox.Show("Vorname: " + Person.Vorname + vbCrLf + "Nachname: " + Person.Nachname + vbCrLf + "Geburtsdatum: " _
        + Person.Geburtsdatum.ToString)
End Sub
```

Das ist die herkömmliche Methode. Visual Basic bietet eine Alternative, bei der Sie eine Konstruktor-Methode verwenden. Eine Konstruktor-Methode wird beim Initialisieren eines Objekts ausgelöst. Sie können auch Parameter für die Konstruktor-Methode definieren. Und Sie können die Konstruktor-Methode auch überladen – das heißt, Sie können mehrere Konstruktor-Methoden mit verschiedenen Parametern definieren, die diese dann auf die gewünschte Weise auswerten.

Die Klasse sieht mit einem Konstruktor, der alle drei Eigenschaften als Parameter abfragt, wie folgt aus:

```
Public Class Person
    Public Sub New(vorname As String, nachname As String, geburtsdatum As Date)
```

```
        Me.Vorname = vorname
        Me.Nachname = nachname
        Me.Geburtsdatum = geburtsdatum
    End Sub
    Public Property Vorname As String
    Public Property Nachname As String
    Public Property Geburtsdatum As Date
End Class
```

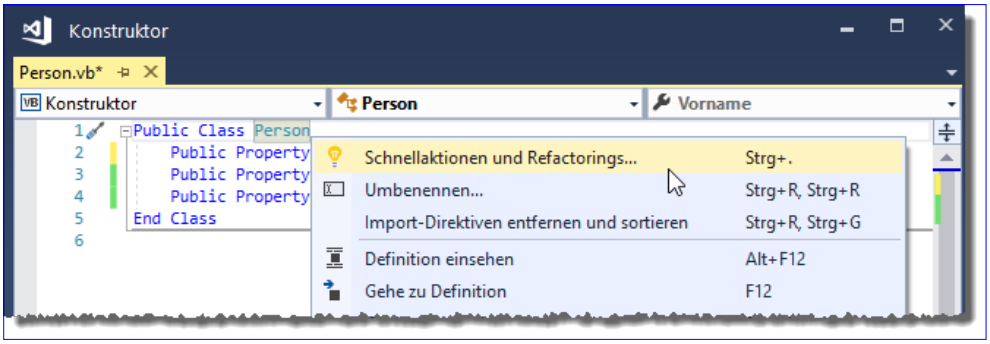

Bild 1: Schnellaktionen und Refactorings aufrufen

Wenn Sie denken, das wäre viel Tipparbeit, dann freuen Sie sich sicherlich darüber, dass Sie den Konstruktor vollständig mit wenigen Mausklicks zusammenstellen können. Dazu klicken Sie mit der rechten Maustaste auf den Kopf der Klasse in Visual Studio und wählen den Eintrag **Schnellaktionen und Refactorings...** aus (siehe Bild 1).

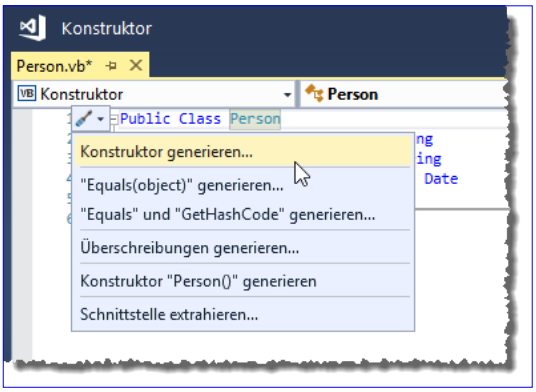

Bild 2: Generieren des Konstruktors

Danach erscheint ein weiteres Aufklappmenü, mit dem Sie den Eintrag **Konstruktor generieren...** aufrufen (siehe Bild 2).

Daraufhin erscheint der Dialog **Member auswählen**. Dieser bietet alle öffentlichen Eigenschaften in einer Liste an. Hier wählen Sie alle Eigenschaften aus, die vom Konstruktor berücksichtigt werden sollen (siehe Bild 3). Ein Klick auf die Schaltfläche **OK** legt den Konstruktor an, den wir weiter oben bereits vorgestellt haben.

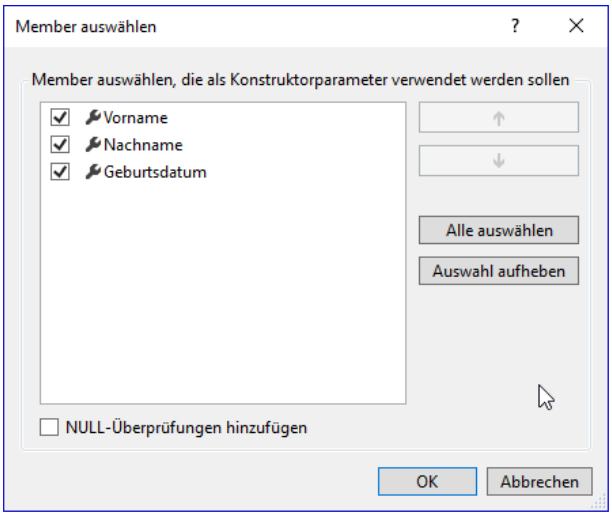

Bild 3: Auswählen der Parameter

Überladen des Konstruktors

Diesen Konstruktor können Sie nun noch überladen. Wir legen also weitere Konstruktoren an, die beispielsweise nur jeweils zwei der drei Parameter entgegennehmen. Damit der Code von ganz oben, in dem wir ein Objekt auf Basis der Klasse **Person** ohne Übergabe eines Parameters beim Erstellen anlegen, keinen Fehler enthält, müssen wir sogar noch einen Konstruktor anlegen – nämlich einen, der keinen Parameter entgegennimmt.

Dieser sieht wie folgt aus:

Textfelder an Daten binden

Eines der Hauptkonzepte von WPF ist es, dass Steuerelemente über entsprechende Bindungsattribute an die Datenquellen gebunden werden. Es gibt verschiedene Datenquellen: einfache Eigenschaften, Objekte oder Auflistungen. In dieser Artikelreihe schauen wir uns an, wie Sie die gängigsten Steuerelemente an verschiedene Datenquellen binden und welche Attribute dazu benötigt werden. Den Beginn macht das Textbox-Steuerelement.

Binden statt zuweisen

Die WPF-Philosophie setzt im Gegensatz etwa zu den Formularen unter Access verstärkt auf die Datenbindung und weniger darauf, dass Steuerelemente per Code auf bestimmte Datenquellen oder Werte eingestellt werden. Wir schauen uns den Unterschied in einem ersten Beispiel an.

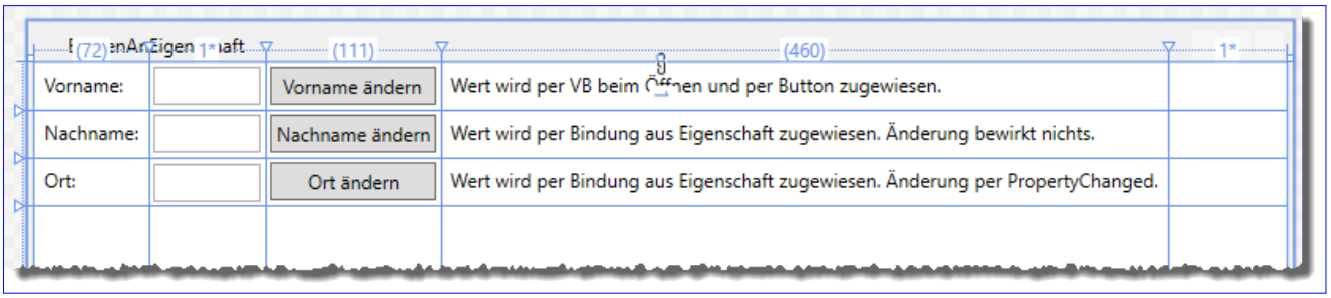

Bild 1: Beispiel für mit Daten gefüllte Textfelder im Entwurf

Zuweisen von Werten per VB

Hier experimentieren wir einfach nur mit Textfeldern für die drei Eigenschaften **Vorname**, **Nachname** und **Ort**. Das Fenster, in dem wir diese Beispiele darstellen, heißt **BindenAnEigenschaft** und enthält in einem Grid neben zwei Bezeichnungsfeldern jeweils ein Textfeld und eine Schaltfläche pro Zeile (siehe Bild 1). Das erste Textfeld namens **txtVorname** soll beim Öffnen des Fensters per Code in der Code behind-Klasse mit einem Wert gefüllt werden.

Ein Klick auf die Schaltfläche soll den Wert des Textfeldes dann ebenfalls durch Zuweisung per VB ändern. Das erledigen wir, indem wir dem Fenster die folgende Konstruktor-Methode zuweisen, die beim Erstellen des Fensters ausgelöst wird. Der Befehl **InitializeComponent** sorgt dafür, dass das Fenster auf Basis der XAML-Definition des Fensters erstellt wird und die zweite Anweisung weist der Eigenschaft **Text** des Textfeldes **txtVorname** den Wert **André** zu:

```
Public Sub New()
    InitializeComponent()
    txtVorname.Text = "André"
End Sub
```

Das sorgt dafür, dass das Textfeld direkt nach dem Öffnen den gewünschten Wert anzeigt. Neben dem Textfeld befindet sich die Schaltfläche **cmdNameAendern**. Für dieses hinterlegen wir die folgende Ereignismethode:

```
Private Sub cmdNameAendern_Click(sender As Object, e As RoutedEventArgs)
    txtVorname.Text = "Michael"
End Sub
```

Die Methode wird durch einen Klick auf die Schaltfläche ausgelöst. Sie weist der Eigenschaft **Text** den Wert **Michael** zu. Diese Art der Anzeige von Daten in Steuerelementen soll unter WPF optimalerweise nicht praktiziert werden.

Bindung an eine Eigenschaft der Code behind-Klasse

Beim zweiten Textfeld **txtNachname** verwenden wir den Ansatz der Bindung vom WPF-Steuerelement aus. Dazu hinterlegen wir im Code behind-Modul des Fensters eine öffentliche Eigenschaft namens **Nachname**:

```
Public Property Nachname As String
```

Dieser Eigenschaft weisen wir in der Konstruktor-Methode **New** einen Wert zu. Außerdem stellen wir für die Eigenschaft **DataContext** den Wert **Me** ein. Damit legen wir fest, dass die aktuelle Klasse als Datenquelle für das WPF-Fenster verwendet werden soll:

```
Public Sub New()
    InitializeComponent()
    DataContext = Me
    Nachname = "Minhorst"
End Sub
```

Somit können wir in den Steuerelementen im WPF-Fenster auf die öffentlich deklarierten Eigenschaften des Code behind-Moduls zugreifen, zum Beispiel auf die Eigenschaft **Nachname**. Das zweite Textfeld **txtNachname** wollen wir auf diese Weise mit dem Wert füllen, den wir in der **New**-Methode der Eigenschaft **Nachname** zugewiesen haben.

Die XAML-Definition des Textfeldes statten wir dazu mit dem Wert **{Binding Nachname}** für das Attribut **Text** aus:

```
<TextBox x:Name="txtNachname" ... Text="{Binding Nachname}"></TextBox>
```

Starten wir nun das Projekt, zeigt das Textfeld den im Konstruktor zur Eigenschaft **Nachname** zugewiesenen Wert **Minhorst** an. Die Schaltfläche **cmdNachnameAendernGebunden** soll den Wert im Textfeld ändern, indem es der Eigenschaft, an das dieses Textfeld gebunden ist, einen neuen Wert zuweist. Dazu programmieren wir seine Ereignismethode wie folgt:

```
Private Sub CmdNachnameAendernGebunden_Click(sender As Object, e As RoutedEventArgs)
    Nachname = "Müller"
End Sub
```

Das hat aber beim Testen keinen Effekt – der Wert des Textfeldes ändert sich nicht. Der Grund ist einfach: Wir müssen bei der Änderung der Eigenschaft ein Ereignis auslösen, durch die das daran gebundene Steuerelement erfährt, dass der Wert der

Eigenschaft geändert wurde. Dazu nutzen wir die **INotifyPropertyChanged**-Schnittstelle. Wie diese genutzt wird, zeigen wir am dritten Textfeld **txtOrt** und der im Code behind-Modul definierten Eigenschaft **Ort**.

Textfeld bei Eigenschaftsänderungen aktualisieren

Um die **IPropertyNotifyChanged**-Schnittstelle zu implementieren, sind folgende Schritte nötig:

- Hinzufügen eines Verweises auf den Namespace **System.ComponentModel**
- Bekanntgeben der Implementierung mit dem **Implements**-Schlüsselwort
- Umsetzen der Implementierung per Schnellaktion und Hinzufügen einer weiteren Methode
- Ändern der Eigenschaft durch Ergänzen von Getter und Setter
- Hinzufügen des Aufrufs von **OnPropertyChanged** im Setter der Eigenschaft

Im Code sieht das Ergebnis wie folgt aus. Im Kopf des Moduls platzieren wir den Verweis auf den **System.ComponentModel**-Namespace:

```
Imports System.ComponentModel
```

Die Implementierung der Schnittstelle geben wir durch das Implements-Schlüsselwort in einer neuen Zeile hinter der Klassenbezeichnung an:

```
Public Class BindenAnEigenschaft
    Implements INotifyPropertyChanged
```

Klicken Sie dann mit der rechten Maustaste auf **INotifyPropertyChanged** und wählen Sie den Eintrag **Schnellaktionen und Refactorings...** aus dem nun erscheinenden Kontextmenü aus. Dies liefert ein Popup, mit dem Sie den Befehl **Schnittstelle implementieren** auswählen.

Das ergänzt die folgende Zeile mit der Definition des Ereignisses **PropertyChanged**:

```
    Public Event PropertyChanged As PropertyChangedEventHandler Implements INotifyPropertyChanged.PropertyChanged
```

Wir fügen dann diese Methode hinzu, um das enthaltene Auslösen des oben definierten Ereignisses nicht für jede Eigenschaft neu schreiben zu müssen:

```
    Protected Overridable Sub OnPropertyChanged(propname As String)
        RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(propname))
    End Sub
```

Stattdessen können wir einen vereinfachten Aufruf in den Setter der Eigenschaft schreiben. Den Setter erstellen wir nebst dem Getter, indem wir die Eigenschaft ebenfalls mit der rechten Maustaste anklicken und wieder den Befehl **Schnellaktionen und Refactorings...** aufrufen. Diesmal führen wir den Befehl **In vollständige Eigenschaft konvertieren** in dem danach angezeigten Popup aus und erzeugen aus der einfachen **Property**-Eigenschaft das folgende Konstrukt:

```
Private _Ort As String

Public Property Ort As String
    Get
        Return _Ort
    End Get
    Set
        _Ort = Value
        OnPropertyChanged("Ort")
    End Set
End Property
```

Wir haben hier schon die entscheidende Anweisung hinzugefügt, nämlich den Aufruf der Methode **OnPropertyChanged** für die als Parameter angegebene Eigenschaft **Ort**. Um zu prüfen, ob der Inhalt des Textfeldes nun automatisch geändert wird, wenn wir den Wert der Eigenschaft anpassen, nutzen wir die durch die Schaltfläche **btnOrtGebunden** ausgelöste Methode:

```
Private Sub BtnOrtGebunden_Click(sender As Object, e As RoutedEventArgs)
    Ort = "Mülheim"
End Sub
```

Und ein Test ergibt, dass die Bindung durch die Implementierung der Schnittstelle **INotifyPropertyChanged** wie gewünscht funktioniert.

Bindung ist keine Einbahnstraße

Was aber geschieht nun, wenn wir den Wert des Textfeldes **txtOrt** ändern – wird durch **INotifyPropertyChanged** automatisch der Wert der Eigenschaft im Code behind-Modul angepasst?

Um das zu probieren, fügen wir eine Schaltfläche hinzu, die den aktuellen Wert der Eigenschaft per Meldungsfenster ausgibt:

```
Private Sub BtnOrtAusgeben_Click(sender As Object, e As RoutedEventArgs)
    MessageBox.Show("Wert der Eigenschaft Ort: " + Ort)
End Sub
```

Damit finden wir heraus, dass sich die Änderung des Inhalts des Textfeldes auch auf die Eigenschaft auswirkt, an die das Textfeld gebunden ist.

Bindungsmodus

Das liegt in diesem Fall daran, dass die Bindung hier einen Standardwert für das Attribut **Mode** verwendet. Explizit würden wir wie folgt festlegen, dass die Bindung in beide Richtungen arbeiten soll:

```
<TextBox x:Name="txtOrtGebunden" ... Text="{Binding Ort, Mode=TwoWay}"></TextBox>
```

Genau genommen ist auch **Ort** bereits der Wert eines Parameters, und zwar **Path**. Explizit würden wir also schreiben:

```
Text="{Binding Path=Ort, Mode=TwoWay}"
```

Wenn das Textfeld geänderte Inhalte nicht in die Eigenschaft zurückschreiben soll, geben Sie für Mode den Wert **OneWay** an:

```
Text="{Binding Path=Ort, Mode=OneWay}"
```

Binden an Elemente einer Klasse

Nachdem wir uns angesehen haben, wie wir das Textfeld an eine einfache Eigenschaft binden, gehen wir nun einen Schritt weiter. Dabei schauen wir uns drei verschieden Konstellationen hinsichtlich der Aktualisierung der Anzeige nach dem Ändern der Daten an:

- Das Code behind-Modul enthält eine Eigenschaft, die ein Element der Klasse **Kunde** bereitstellt.
- Das Code behind-Modul enthält eine Eigenschaft, die ein Element der Klasse **Kunde** bereitstellt und diese Eigenschaft wird mit der **INotifyPropertyChanged**-Schnittstelle ausgestattet.
- Das Code behind-Modul enthält eine Eigenschaft, die ein Element der Klasse **Kunde** bereitstellt und die Eigenschaften der Klasse **Kunde** werden mit der **INotifyPropertyChanged**-Schnittstelle ausgestattet.

Beispiel für die Klasse ohne INotifyPropertyChange

Im ersten Beispiel fügen wir dem Code behind-Modul eine Klasse hinzu. Dann wollen wir den Inhalt der Eigenschaften der Klasse in einem Textfeld anzeigen. Dazu fügen wir eine Klasse mit dem folgenden Aufbau hinzu, wobei wir aus Platzgründen nur die Eigenschaft **Vorname** hinzugefügt haben. Die Klasse bringen wir in einer eigenen Datei unter:

```
Public Class Kunde
    Private _Vorname As String
    Public Property Vorname As String
        Get
            Return _Vorname
        End Get
        Set(value As String)
            _Vorname = value
        End Set
```

Ungebundene ComboBox-Steuerelemente

Auswahlfelder werden unter WPF meist mit dem ComboBox-Steuerelement abgebildet. Manchmal braucht man diese nur für die Auswahl von Daten aus einfachen Auflistungen und die ComboBox-Steuerelemente selbst sind nicht an eine Eigenschaft des Code behind-Moduls gebunden. Wie man solche ungebundenen ComboBox-Steuerelemente einfach abbildet, zeigt dieser Artikel.

Wir schauen uns die ungebundene ComboBox für verschiedene Fälle an – als erstes für eine einfache Liste von Zeichenketten, danach für eine Liste mit Objekten einer bestimmten Klasse. Die Beispiele finden Sie jeweils in eigenen Fenstern, die vom Hauptfenster **MainWindow** aus aufgerufen werden.

String-Liste in ComboBox darstellen

Im Entwurf sieht unser erstes Beispiel (siehe Fenster **ComboBoxMitEinfacherListe.xaml**) wie in Bild 1 aus. Neben einigen **Style**-Definitionen für die Steuerelemente im Bereich **Window.Resources** des XAML-Codes und der Definition des **Grid**-Elements, die Sie im Beispielprojekt einsehen können, werden das Bezeichnungsfeld und das **ComboBox**-Element wie folgt definiert:

```
<Label>Ungebundene ComboBox:</Label>
<ComboBox Grid.Column="1" x:Name="cboUngebunden" ItemsSource="{Binding Vornamen}"></ComboBox>
```

Das Ziel ist nun, die Angabe des Attributs mit **ItemsSource** und dem Wert **{Binding Vornamen}** im Code behind-Modul so mit Leben zu füllen, dass die gewünschten Einträge in der ComboBox erscheinen.

Dazu deklarieren wir zunächst eine private Variable für die Liste und eine entsprechende Eigenschaft, welche die Liste für die daran gebundenen Steuerelemente verfügbar macht:

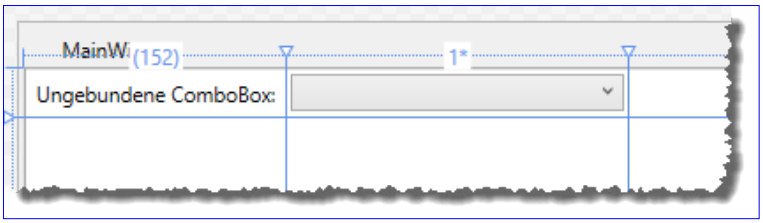

Bild 1: Entwurf der ComboBox

```
Class MainWindow
    Private _Vornamen As List(Of String)

    Public Property Vornamen As List(Of String)
        Get
            Return _Vornamen
        End Get
        Set(value As List(Of String))
            _Vornamen = value
        End Set
    End Property
```

Nun fügen wir die Konstruktor-Methode hinzu, die dafür sorgt, dass die Liste gefüllt wird und dass das XAML-Dokument das Code behind-Modul als Datenquelle verwenden kann (**DataContext = Me**). Danach füllen wir die Liste namens **Vornamen** noch mit drei Einträgen:

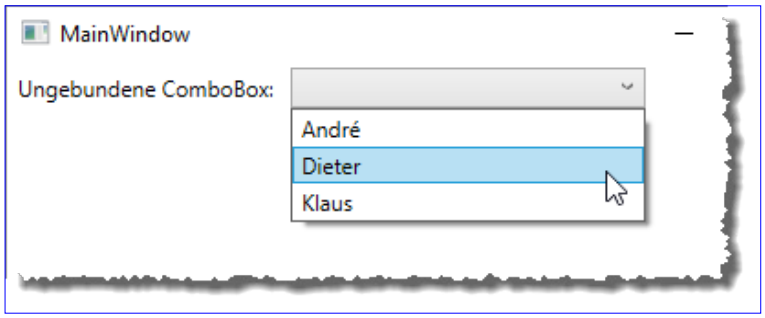

Bild 2: **ComboBox**-Steuerelement mit Einträgen

```
    Public Sub New()
        InitializeComponent()
        DataContext = Me
        Vornamen = New List(Of String)
        With Vornamen
            .Add("André")
            .Add("Dieter")
            .Add("Klaus")
        End With
    End Sub
End Class
```

Das Ergebnis sieht wie in Bild 2 aus. Das ist eine einfache Version für das Binden von Daten aus einer Liste als Datenquelle eines **ComboBox**-Steuerelements.

ComboBox mit Daten aus einer Liste von Objekten füllen

Nun gehen wir einen Schritt weiter, denn nicht in allen Fällen bestehen die Daten, die in einem **ComboBox**-Steuerelement zur Auswahl angeboten werden sollen, nur aus einem Feld. Die Daten kommen eher als Liste von Objekten einer bestimmten Klasse wie etwa **Kunde** oder **Person**. Diese enthaltenen die anzuzeigenden Daten als Eigenschaften.

Diese Klasse sieht so aus:

```
Public Class Person
    Private _ID As Integer
    Private _Vorname As String
    Private _Nachname As String
    Public Property ID As Integer
        Get
            Return _ID
        End Get
        Set(value As Integer)
            _ID = value
        End Set
    End Property
```

```
    Public Property Vorname As String
        Get
            Return _Vorname
        End Get
        Set(value As String)
            _Vorname = value
        End Set
    End Property

    Public Property Nachname As String
        Get
            Return _Nachname
        End Get
        Set(value As String)
            _Nachname = value
        End Set
    End Property
End Class
```

In der Code behind-Klasse des Fensters **ComboBoxMitObjektListe.xaml** fügen wir folgenden Code hinzu. Im Kopf beginnen wir mit den zwei privaten Variablen **_Person** für ein **Person**-Objekt und **_Personen** für eine Liste von Objekten des Typs **_Person**:

```
Public Class ComboBoxMitObjektListe
    Dim _Person As Person
    Dim _Personen As List(Of Person)
```

Um die Liste der Personen nach außen zugänglich zu machen, damit Sie beispielsweise als Datenquelle des **ComboBox**-Steuerelements verwendet werden können, fügen wir eine öffentliche Eigenschaft namens **Personen** mit Getter und Setter zur Verfügung:

```
    Public Property Personen As List(Of Person)
        Get
            Return _Personen
        End Get
        Set(value As List(Of Person))
            _Personen = value
        End Set
    End Property
```

In der Konstruktor-Methode **New** initialisieren wir zunächst die Komponente und erstellen dann eine neue Liste des Typs **Person**, die wir in der Variablen **Personen** speichern. Dann legen wir zwei neue Objekte des Typs **Person** an, die wir in **_Person** zwischenspeichern, und weisen diese dann der Liste **Personen** zu:

Abhängige ComboBox-Steuerelemente

Ein beliebter Anwendungsfall von ComboBox-Steuerelementen ist, dass der Benutzer in der ersten ComboBox einen Wert auswählt und die in der zweiten ComboBox angezeigten Daten entsprechendem Wert aus der ersten ComboBox angepasst werden. Dafür gibt es einige Beispiele – Hersteller und Produkte, Abteilungen und Mitarbeiter, Kategorien und Artikel und so weiter. Dieser Artikel zeigt, wie Sie zwei ComboBox-Steuerelemente so mit Daten füllen und programmieren, dass diese die Auswahl abhängiger Informationen erlauben.

Die Grundkonstellation besteht in der Definition von zwei **ComboBox**-Steuerelementen im WPF-Fenster **MainWindow.xaml** des Beispielprojekts. Das Ergebnis sieht wie in Bild 1 aus.

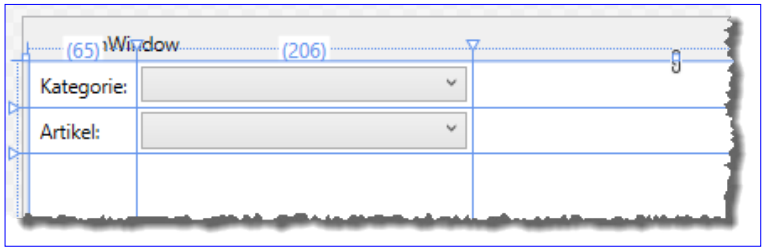

Bild 1: Die beiden **ComboBox**-Steuerelemente in der Entwurfsansicht

Die Steuerelemente definieren wir wie folgt:

```
<Label>Kategorie:</Label>
<ComboBox x:Name="cboKategorien" ItemsSource="{Binding Kategorien}"
    DisplayMemberPath="Bezeichnung"></ComboBox>
<Label>Artikel:</Label>
<ComboBox x:Name="cboArtikel" ItemsSource="{Binding Produkte}" DisplayMemberPath="Bezeichnung"></ComboBox>
```

Klassen für Kategorien und Produkte

Für die Kategorien verwenden wir die folgende Klassendefinition, die neben den beiden automatisch implementierte Eigenschaften noch einen Konstruktor enthalten:

```
Public Class Kategorie
    Public Sub New(ID As Integer, Bezeichnung As String)
        Me.ID = ID
        Me.Bezeichnung = Bezeichnung
    End Sub

    Public Property ID As Integer
    Public Property Bezeichnung As String
End Class
```

Die Klasse **Produkt** hat ein Feld mehr, weil sie noch eine Eigenschaft für die Referenzierung der Kategorie enthält, der das Produkt zugeordnet ist. Auch hier verwenden wir automatisch implementierte Eigenschaften und einen Konstruktor, der Parameter für alle Eigenschaften bereitstellt:

```
Public Class Produkt
    Public Sub New(ID As Integer, Bezeichnung As String, Kategorie As Kategorie)
        Me.ID = ID
        Me.Bezeichnung = Bezeichnung
        Me.Kategorie = Kategorie
    End Sub

    Public Property ID As Integer
    Public Property Bezeichnung As String
    Public Property Kategorie As Kategorie
End Class
```

Anlegen von Kategorien und Produkten

Damit wir ein wenig Spielmaterial zum Experimentieren haben, verwenden wir die folgende Methode namens **ListenFuellen**. Diese initialisiert zunächst die beiden **ObservableCollection**-Objekte. Dann erstellen wir jeweils eine Kategorie, speichern die neue Instanz in der Variablen **Kategorie** und fügen diese zur Auflistung **Kategorien** hinzu.

Anschließend erstellen wir jeweils ein paar **Produkt**-Elemente und weisen diesen die Kategorie aus der Variablen **Kategorie** zu. Auf diese Weise erstellen wir drei Kategorien und sechs Produkte, die den Kategorien zugeordnet sind:

```
Private Sub ListenFuellen()
    Dim Kategorie As Kategorie

    Produkte = New ObservableCollection(Of Produkt)
    Kategorien = New ObservableCollection(Of Kategorie)

    Kategorie = New Kategorie(1, "Getränke")
    Kategorien.Add(Kategorie)
    Produkte.Add(New Produkt(1, "König Pilsener", Kategorie))
    Produkte.Add(New Produkt(2, "Fanta", Kategorie))

    Kategorie = New Kategorie(2, "Süßwaren")
    Kategorien.Add(Kategorie)
    Produkte.Add(New Produkt(3, "Milka Sahne Creme", Kategorie))
    Produkte.Add(New Produkt(4, "Aero Luftschokolade", Kategorie))

    Kategorie = New Kategorie(3, "Gemüse")
    Kategorien.Add(Kategorie)
    Produkte.Add(New Produkt(5, "Brokkoli", Kategorie))
    Produkte.Add(New Produkt(6, "Chinakohl", Kategorie))
End Sub
```

In der Code behind-Klasse von **MainWindow** fügen wir nun noch zwei öffentliche Eigenschaften für die Auflistungen der Kategorien und Produkte hinzu.

Außerdem erstellen wir eine Konstruktor-Methode für die Klasse und rufen dort die Methode **Listen-Fuellen** auf. Außerdem stellen wir den **DataContext** des Fensters auf die Code behind-Klasse ein:

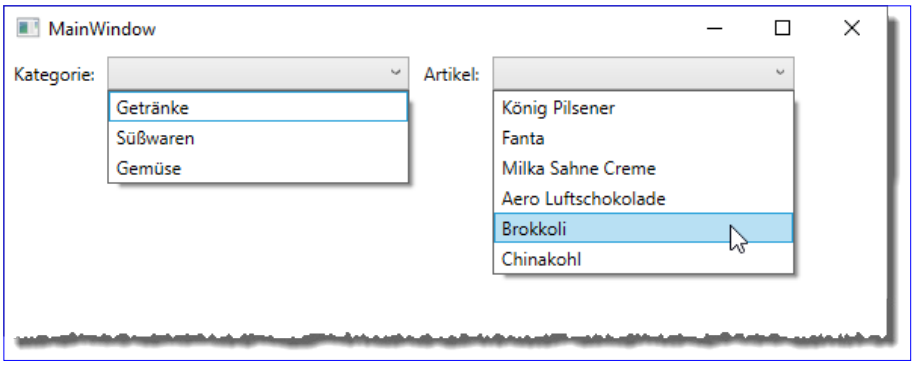

Bild 2: Die **ComboBox**-Elemente zeigen alle Daten an.

```
Imports System.Collections.ObjectModel

Class MainWindow
    Public Property Kategorien As ObservableCollection(Of Kategorie)
    Public Property Produkte As ObservableCollection(Of Produkt)
    Public Sub New()
        InitializeComponent()
        ListenFuellen
        DataContext = Me
    End Sub
    Private Sub ListenFuellen()
        ...
    End Sub
End Class
```

Nach dem Starten der Prozedur sieht das Fenster mit geöffneten Kombinationsfeldern wie in Bild 2 aus (den Screenshot haben wir so zusammengestellt, ein Öffnen zweier Kombinationsfelder gleichzeitig ist nicht möglich).

Inhalt der zweiten ComboBox nach dem Wert der ersten ComboBox filtern

Nun wollen wir zunächst eine Methode ansehen, um die Werte der zweiten **ComboBox** zu filtern, wie man sie per Ereignisprogrammierung verwenden würde.

Dafür erweitern wir die XAML-Definition der ersten ComboBox um das Ereignisattribut **SelectionChanged**:

```
<ComboBox x:Name="cboKategorien" ItemsSource="{Binding Kategorien}" DisplayMemberPath="Bezeichnung"
SelectionChanged="CboKategorien_SelectionChanged"></ComboBox>
```

Die dazugehörige Methode sieht so aus:

Reporting Services: Installation und Start

Der Vorteil von Microsoft Access ist: Man bekommt alles aus einer Hand. Tabellenentwurf, Abfragedesigner, Formulare, Programmierumgebung und – Tools zur Berichtserstellung. Bei der Datenbank mit Visual Studio gibt es so viele Möglichkeiten, die aber alle irgendwie viel größer und komplizierter erscheinen als das, was der Berichtsdesigner von Access bietet. Früher oder später wollen wir aber auch in DATENBANKENTWICKLER das Thema Reporting behandeln und deshalb schauen wir uns in dieser Artikelreihe die Möglichkeiten der SQL Server Reporting Services an. Da SQL Server mittlerweile in einer Community Version kommt, die für den privaten Einsatz kostenlos ist, steht dem Ausprobieren auch nichts im Wege. Im ersten Teil der Artikelreihe schauen wir uns an, wie Sie die Reporting Services unter Visual Studio startklar machen und einen einfachen Beispielbereicht erzeugen.

Reporting Services

Die SQL Server Reporting Services bieten aber nicht nur paginierte Berichte, sondern Sie können damit auch mobile Berichte anlegen, die in Abhängigkeit vom verwendeten Gerät angezeigt werden können. Das ist der Tatsache geschuldet, dass immer mehr Menschen mobile Geräte nutzen, die einfach nicht gut zur Anzeige von Berichten im DIN A4-Format geeignet sind.

Außerdem liefern sie ein Webportal, mit dem Sie in modernen Browsern die Berichte verwalten und anzeigen können. Zu diesen beiden Features kommen wir in weiteren Artikeln.

Vorbereitungen

Für die Beispiele diese Artikels verwenden wir Visual Studio 2017. Gegebenenfalls sind die Reporting Services noch nicht installiert. Dann laden Sie diese zunächst herunter. Die Reporting Services 2017 finden Sie beispielsweise unter diesem Link:

```
https://www.microsoft.com/de-DE/download/details.aspx?id=55252
```

Bild 1: Installation der Reporting Services

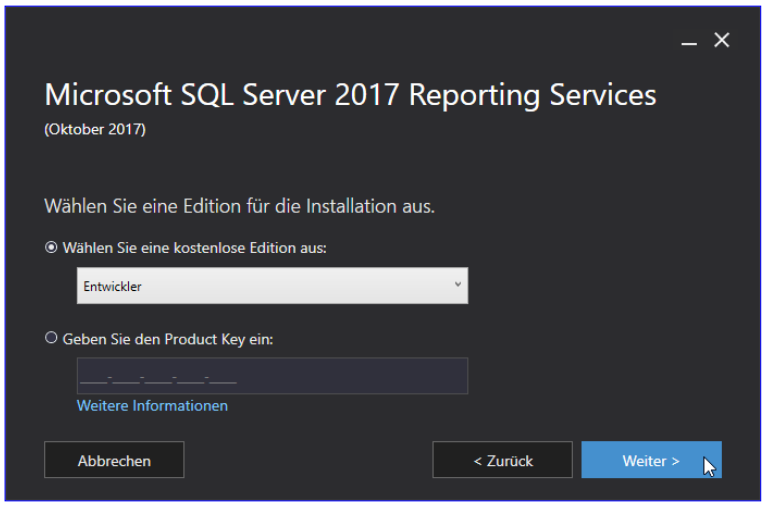

Bild 2: Auswahl der gewünschten Version

Nach dem Download der knapp 100 MB starten wir die **.exe**-Datei, die sich mit dem Startfenster aus Bild 1 zeigt.

Nach einem Klick auf **Reporting Services installieren** wählen Sie wie in Bild 2 die gewünschte Version aus. Wir möchten nicht die standardmäßig vorgeschlagene 180-Tage-Testversion, sondern die Entwicklerversion installieren. Anschließend bestätigen Sie die Nutzungsbestimmungen. Auf der Seite **Datenbankmodul installieren** klicken Sie auf **Weiter**. Auf der folgenden Seite können Sie den Installationsort anpassen, wir behalten diesen einfach bei. Danach startet die Installation.

Zum Abschluss können Sie mit einem Klick auf **Berichtsserver konfigurieren** oder mit einem Klick auf **Schließen** fortfahren. Wir wählen die erste Option und prüfen zunächst, welcher SQL Server als Berichtsserver verwendet werden soll. Hier behalten wir die Voreinstellung bei (siehe Bild 3). Die übrigen Einstellungen, die danach im Dialog **Report Server Configuration Manager** angezeigt werden, behalten wir auch bei. Den Konfigurationsmanager können Sie später jederzeit über das **Suchen**-Feld von Windows ausfindig machen und erneut starten.

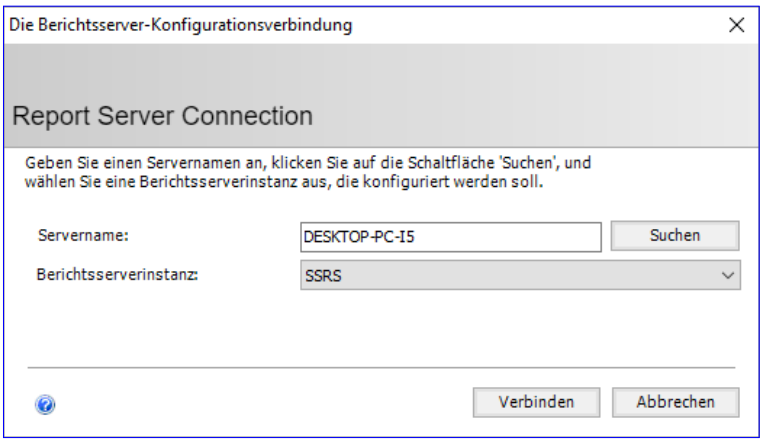

Bild 3: Festlegen des Servers

Bild 4: Installation der Microsoft SQL Server Data Tools

SQL Server Data Tools installieren

Danach installieren wir die SQL Server Data Tools, die Sie unter dem folgenden Link finden:

```
https://go.microsoft.com/fwlink/?linkid=2095463
```

Die hier heruntergeladene Datei liefert nach dem Start den Dialog aus Bild 4, wo Sie die Visual Studio-Instanz auswählen und unten die Option **SQL Server Reporting Services** aktivieren. Gegebenenfalls benötigen Sie noch eine Aktualisierung von Visual Studio. In jedem Fall müssen Sie Visual Studio schließen, bevor Sie die **SQL Server Data Tools** installieren.

Reporting Services Projects installieren

Der dritte Schritt ist die Installation von Komponenten, die das Erstellen von Berichten über Visual Studio erlauben. Ob diese bereits installiert sind, können Sie prüfen, indem Sie den Menüpunkt **Datei|Neu|Projekt...** aufrufen und im nun erscheinenden

Dialog **Neues Projekt** auf der linken Seite nach Einträgen wie **Reporting Services** suchen. Sind diese wie in Bild 5 nicht vorhanden, müssen Sie diese noch hinzufügen.

Das erledigen Sie, indem Sie den Menübefehl **Extras|Extensions und Updates...** aufrufen. Hier wechseln wir auf der linken Seite zum Bereich **Online** und geben als Suchbegriff **Reporting Services** an. Das liefert nach kurzer Suche den Eintrag **Microsoft Reporting Services Projects** aus Bild 6, den wir durch einen Klick auf **Herunterladen** installieren.

Anschließend wird unten ein Text eingeblendet, der besagt, dass für die Änderung erst alle Visual Studio-Fenster geschlossen werden müssen. Das erledigen wir, indem wir Visual Studio schließen und neu starten. Auch diese Installation nimmt sich wieder ein paar Minuten Zeit.

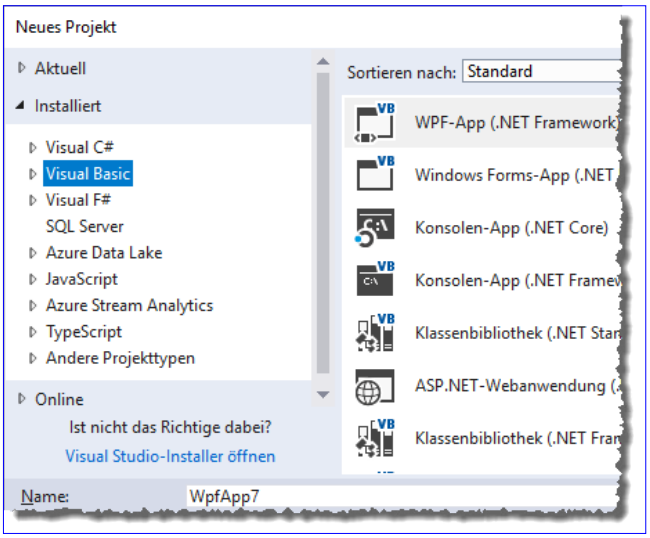

Bild 5: Die Reporting Services Projects sind offenbar noch nicht installiert.

Die Installation hat zunächst nicht funktioniert. Ein zweiter Anlauf, bei dem ich Visual Studio zuvor als Administrator geöffnet habe, war dann jedoch erfolgreich.

Ein erneutes Öffnen von Visual Studio und das Anzeigen des Dialogs zum Anlegen eines neuen Projekts lieferte dann die Ansicht aus Bild 7, wo wir links den Eintrag **Reporting Services** öffnen konnten und dann die Projektvorlagen **Berichtsserverprojekt-Assistent** und **Berichtsserverprojekt** vorfinden.

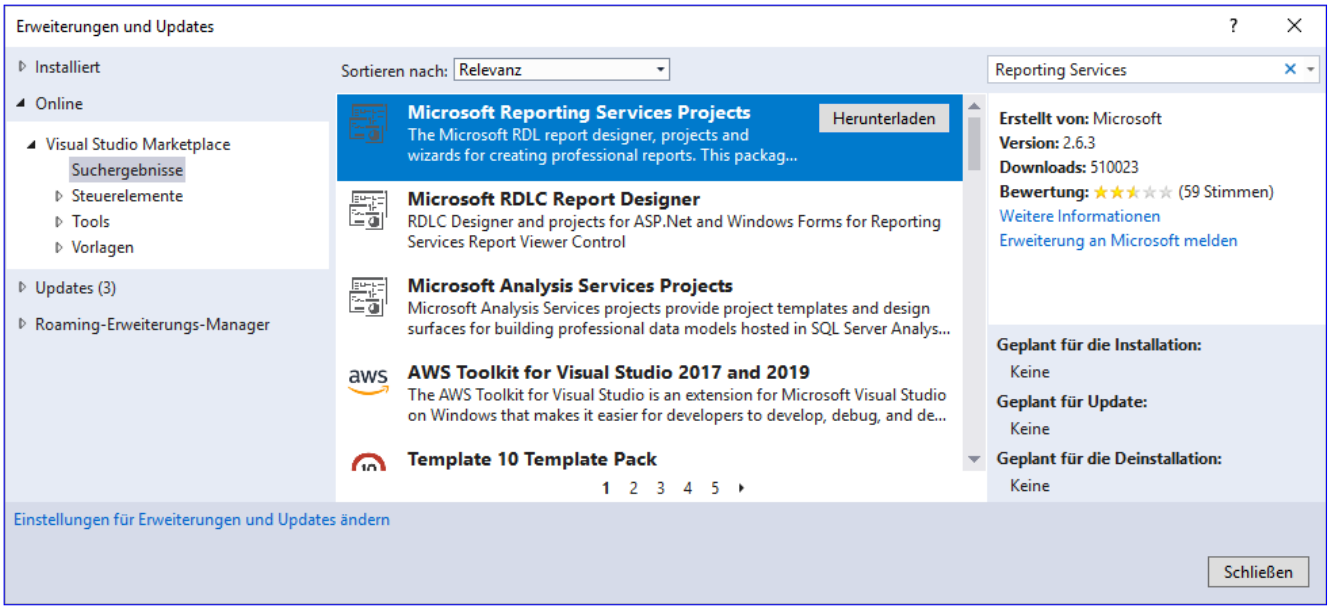

Bild 6: Installieren der Erweiterung **Microsoft Reporting Services Projects**

Reporting Services: Accessberichte importieren

Wenn Sie mit Access arbeiten, um Datenbankanwendungen zu programmieren, und damit Berichte entworfen haben, wissen Sie, wieviel Arbeit hinter einem Bericht stecken kann. Umso ärgerlicher, wenn man die Anwendung dann nach .NET migrieren möchte und die ganze Arbeit nochmal erledigen muss. Doch das muss gar nicht sein, denn: Die Microsoft Reporting Services Projects erlauben auch den Import von Berichten aus bestehenden Access-Datenbanken. Ob diese Funktion Arbeit bei der Migration spart und wie sie überhaupt funktioniert, klärt dieser Artikel.

Beim Herumspielen mit den Microsoft Reporting Services Projects schaute ich auch einmal in das Untermenü **Projekt**, um sicherzugehen, dass ich keine Möglichkeit übersehe, einen neuen Bericht anzulegen. Dort finde ich zu meiner Überraschung den Eintrag **Berichte importieren**, der noch überraschender den Befehl **Microsoft Access...** lieferte (siehe Bild 1). Und warum soll das nicht funktionieren? Immerhin gibt es in Access-Berichten auch nur einfach Steuerelemente, die an bestimmten Positionen platziert sind und die, wenn es sich um gebundene Steuerelemente handelt, eine Eigenschaft mit dem Verweis auf ein Feld der Datensatzquelle des Berichts oder einen berechneten Ausdruck enthalten. Kompliziert könnte es werden, die verschiedenen Berichtsbereiche und Gruppierungen von Access in einen Reporting Services-Bericht zu überführen, aber auch das sollte möglich sein. Schauen wir uns doch einfach an, was passiert!

Vorbereitung

Wichtig ist natürlich, dass wir die Datenquelle für den Bericht, den wir importieren wollen, in geeigneter Form bereitstellen können. Da die hier verwendete Software zur Reportgenerierung auf den SQL Server Reporting Services basiert, müssen die Daten logischerweise in einer SQL Server-Datenbank bereitgestellt werden. Sollten die Daten bisher nur in einer Access-Anwendung vorliegen, müssten wir diese also zuvor noch migrieren. Das ist aber kein Problem, denn wir haben ja in den vergangenen

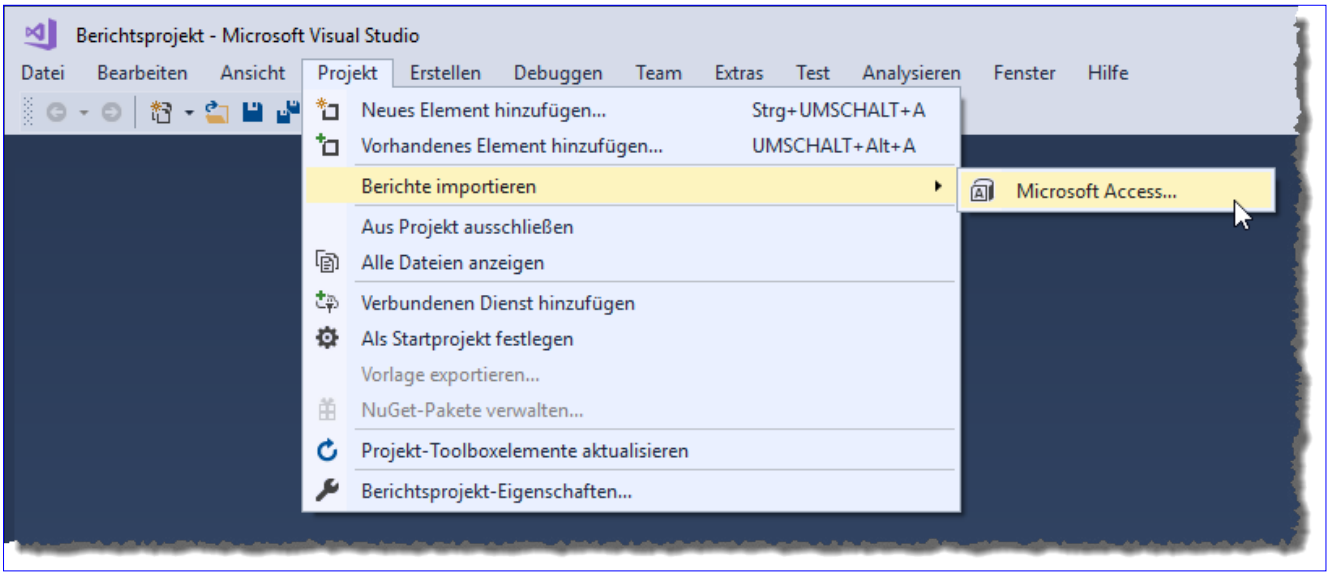

Bild 1: Menüeintrag zum Importieren von Berichten aus Access-Datenbanken

Ausgaben Code programmiert, mit dem wir ein Entity Data Framework auf Basis des Datenmodells einer Access-Datenbank erstellen können. Dieses wiederum können wir zur Generierung der entsprechenden SQL Server-Datenbank verwenden. Die Vorbereitung sollte also nur wenige Minuten in Anspruch nehmen. Und vielleicht ist das sogar noch nicht einmal notwendig und die Daten werden beim Import direkt mit importiert. Probieren wir es aus!

Die Vorbereitungen beschränken sich also zunächst darauf, ein neues Projekt mit der Vorlage **Reporting Services|Berichtsserverprojekt** zu erstellen.

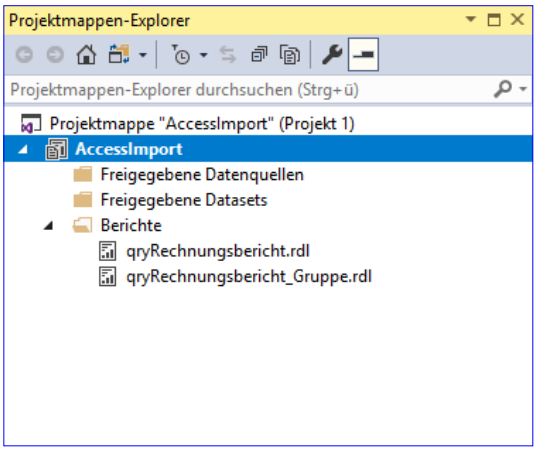

Bild 2: Projektmappen-Explorer mit zwei frisch importierten Berichten

Berichte importieren

Wir probieren es direkt mit einem recht komplizierten Bericht, nämlich mit einem Rechnungsbericht aus einem Artikel aus der Zeitschrift **Access im Unternehmen**. Den Artikel mit der Beschreibung zur Erstellung des Berichts finden Sie unter dem folgenden Link:

`http://access-im-unternehmen.de/Rechnungsbericht/`

Danach rufen wir direkt den Menüeintrag **Projekt|Berichte importieren|Microsoft Access...** auf, was einen Dateiauswahl-Dialog öffnet. Hier wählen wir die Datenbankdatei aus, deren Berichte wir importieren wollen und bereiten uns darauf vor, die zu importierenden Berichte zu selektieren. Das geschieht jedoch nicht, denn die Funktion öffnet die

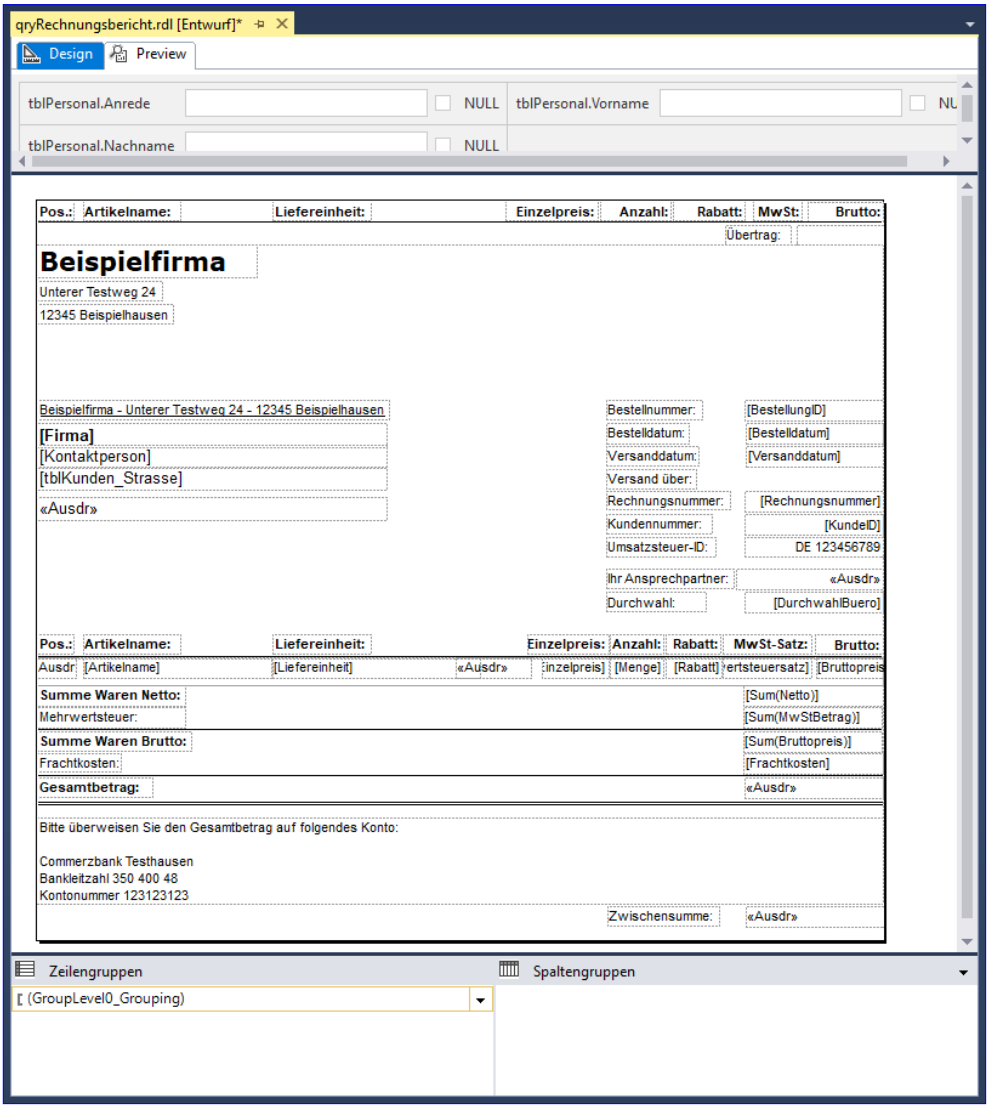

Bild 3: Entwurf des importierten Berichts

Reporting Services: Berichte entwerfen

Im Artikel »Reporting Services: Installation und Start« haben wir gezeigt, wie Sie die Reporting Services auf dem lokalen Rechner installieren. Außerdem haben Sie dort erfahren, wie Sie einen ersten kleinen Beispielbericht unter Visual Studio mithilfe der Reporting Services Projects erstellen. Im vorliegenden Beitrag gehen wir etwas mehr ins Detail und schauen uns an, wie Sie Berichte erzeugen, ohne dazu den Assistenten zu nutzen und wie Sie Berichte nachträglich anpassen können.

Vorbereitungen

Für die Arbeit mit den Reporting Services in Visual Studio benötigen Sie eine Installation der **Microsoft SQL Server Reporting Services**, der **Microsoft SQL Server Data Tools** und für Visual Studio die **Microsoft Reporting Services Projects**. Details hierzu finden Sie im Artikel **Reporting Services: Installation und Start**. Auf dieser Basis legen wir ein neues Projekt mit der Vorlage **Reporting Services|Berichtsserverprojekt** namens **BerichteEntwerfen** an.

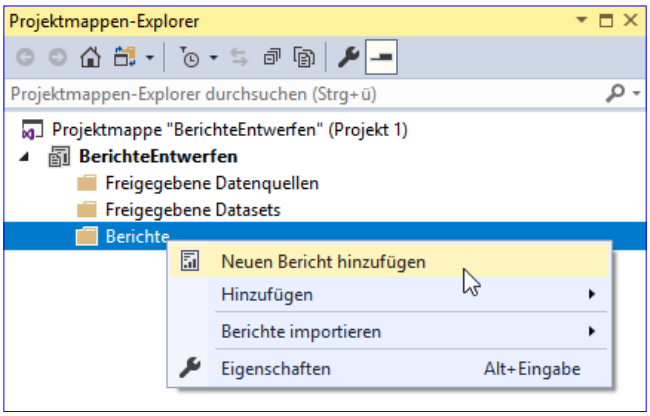

Bild 1: Projektmappen-Explorer eines Reporting Services-Projekts

Danach finden Sie ein recht leeres Visual Studio vor. Selbst der Projektmappen-Explorer zeigt nur drei leere Ordner namens **Freigegebene Datenquellen**, **Freigegebene Datasets** und **Berichte** an (siehe Bild 1). Hier finden wir außerdem einige spezifische Kontextmenü-Einträge für die verschiedenen Ordner. Der Ordner **Freigegebene Datenquellen** liefert den Eintrag **Neue Datenquelle hinzufügen**, der Ordner **Freigegebene Datasets** den Eintrag **Neues Dataset hinzufügen** und **Berichte** bietet die Befehle **Neuen Bericht hinzufügen** und **Berichte importieren** an. Letzterer enthält wiederum den Befehl **Microsoft Access...**, den wir im Artikel **Reporting Services: Import von Access** beschreiben.

Neuen Bericht hinzufügen

Der Befehl **Neuen Bericht hinzufügen** öffnet den Berichts-Assistent, den wir schon im Artikel **Reporting Services: Installation und Start** beschrieben haben. Es gibt aber noch eine zweite Möglichkeit, einen Bericht hinzuzufügen – und dieser verwendet nicht den Assistenten. Dabei handelt es sich um den Dialog zum Hinzufügen beliebiger Elemente zu einem Projekt. Diesen öffnen Sie beispielsweise mit dem Menübefehl **Projekt|Neues Element hinzufügen...** oder mit dem Kontextmenü-Eintrag **Hinzufügen|Neues Element...** der Ordner im Projektmappen-Explorer. Der Dialog **Neues Element hinzufügen** zeigt dann direkt die drei möglichen Einträge an, nämlich **Bericht**, **Datenquelle** oder **Dataset** (siehe Bild 2). Weitere Elemente werden nicht angeboten.

Datenquellen

Wie wir sehen, haben Sie die Möglichkeit, neben den eigentlichen Berichten Elemente der Typen **DataSource** und **DataSet** zum Projekt hinzuzufügen. Da stellt sich die Frage, ob wir nur mit den klassischen Methoden direkt auf den SQL Server zugreifen

können oder auch über das Entity Data Model, das zum Beispiel die Daten einer SQL Server-Datenbank bereitstellt, aber auch Daten aus anderen Quellen liefern kann. Wir schauen uns zunächst den klassischen Weg an. Später verwenden wir gegebenenfalls Daten aus dem Entity Data Model als Datenquelle.

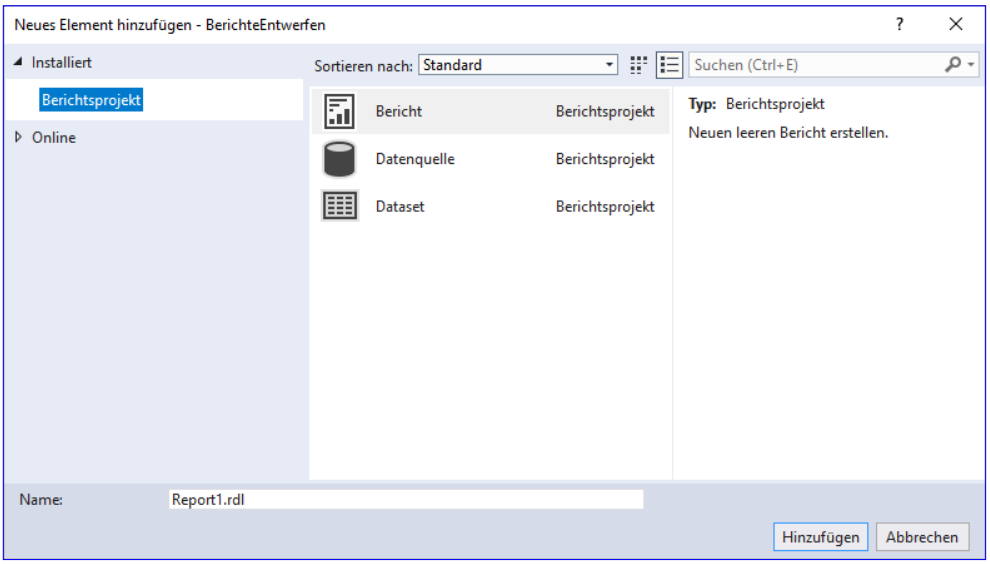

Bild 2: Hinzufügen neuer Elemente zu einem Reporting Services-Projekt

Neuer Bericht

Wenn wir einen neuen Bericht hinzufügen, ohne eine Datenquelle oder ein Dataset zu definieren, finden wir den Entwurf des Berichts wie in Bild 3 vor. Hier sehen Sie verschiedene Bereiche. Links sehen Sie den Bereich, aus dem Sie die Berichtsdaten zum Bericht hinzufügen können. In der Mitte können Sie im Entwurfsbereich zwischen **Design** und **Preview** wählen. Das Raster oben in diesem Bereich erläutern wir später. Darunter ist der eigentliche Entwurf, in dem Sie die Steuerelemente anordnen. Darunter definieren wir Zeilen- und Spaltengruppen –

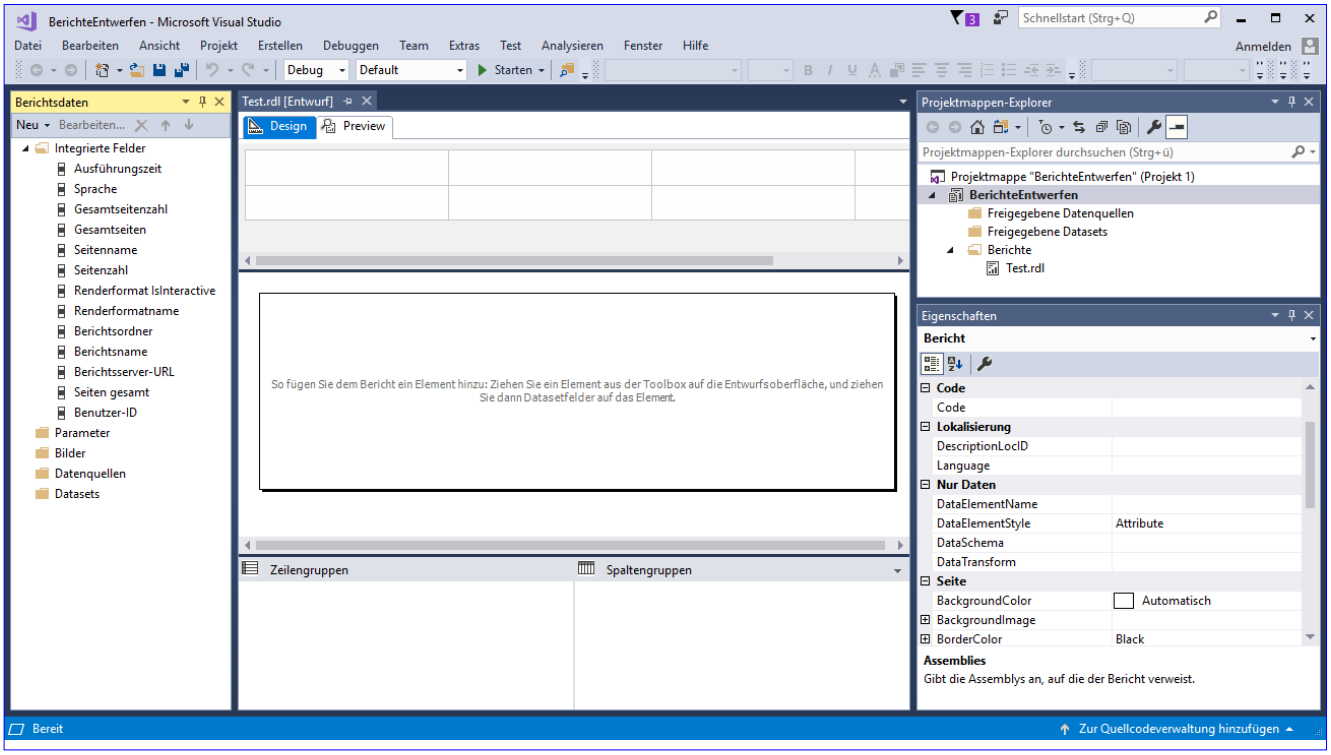

Bild 3: Entwurf eines leeren Berichts

auch dazu später mehr. Rechts finden Sie wie gewohnt den Projektmappen-Explorer sowie das Eigenschaften-Fenster. Um fortzufahren, fügen wir dem Bericht nun zunächst eine Datenquelle hinzu.

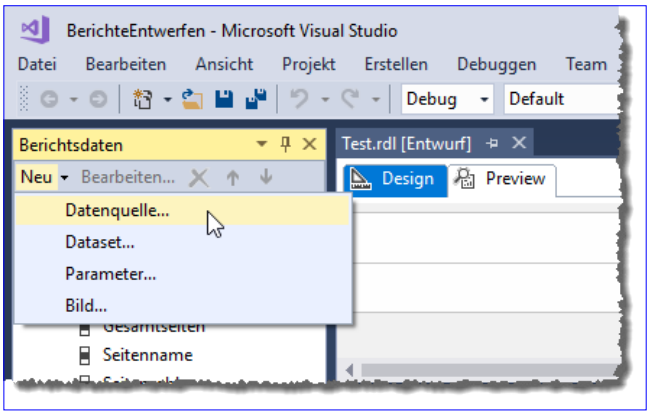

Bild 4: Aufruf des Befehls zum Hinzufügen einer Datenquelle

Datenquelle definieren

Zum Hinzufügen der Datenquelle rufen wir den Befehl **NeulDatenquelle...** des Bereichs **Berichtsdaten** auf (siehe Bild 4). Das können Sie allerdings auch über die zuvor erwähnten Stellen erledigen.

Es erscheint dann der Dialog **Datenquelleneigenschaften**, in dem wir zunächst den Namen der Datenquelle einstellen, und zwar auf **dsBestellverwaltung**. Danach wählen wir den Typ der Datenverbindung aus. Hier finden wir, im Gegensatz zu dem Assistent, den wir im Artikel **Reporting Services: Installation und Start** verwendet haben, einige weitere mögliche Datenquellen. Wir wählen dennoch den Eintrag **SQL Server** aus (siehe Bild 5).

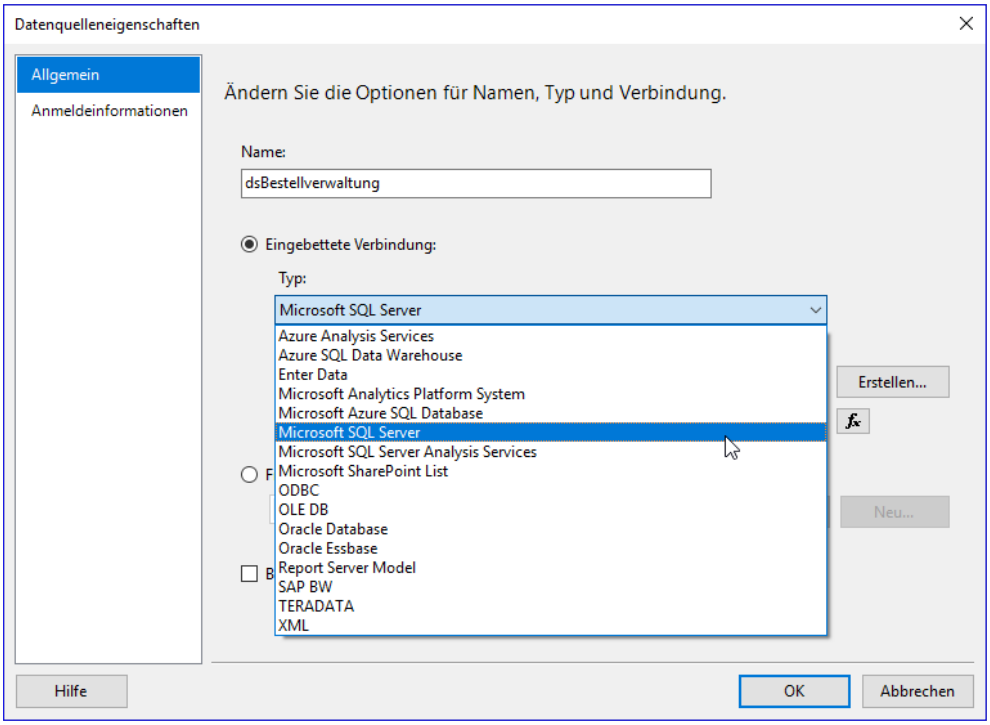

Bild 5: Einstellen der Eigenschaften der Datenquelle

Der nächste Schritt ist die Angabe der Verbindungszeichenfolge. Wenn Sie diese nicht aus dem Kopf zusammenstellen können, bietet der Dialog mit der Schaltfläche **Erstellen...** die Möglichkeit, sich unterstützen zu lassen. Hier geben Sie den SQL Server ein, der die zu verwendende Datenbank enthält, in unserem Fall **(localdb)\MSSQLLocalDb**. Nach der Eingabe können Sie die gewünschte Datenbank aus dem Feld **Datenbankname auswählen oder eingeben** selektieren.

Schließlich können Sie die Verbindung mit einem Klick auf die Schaltfläche **Testverbindung** testen (siehe Bild 6). Nach dem Schließen des Dialogs **Verbindungseigenschaften** erhalten wir die folgende Verbindungszeichenfolge:

```
Data Source=(localdb)\\MSSQLLocalDb;Initial Catalog=Bestellverwaltung
```

Nach dem Schließen beider Dialoge wird die neue Datenquelle im Bereich **Berichtsdaten** unter **Datenquellen** angezeigt (siehe Bild 7).

Warum erscheint dieser aber nicht im Projektmappen-Explorer? Weil zunächst davon ausgegangen wird, dass diese Verbindung nur für den aktuellen Bericht verwendet werden soll. Wollen Sie diese auch noch für weitere Berichte dieses Berichtsprojekts nutzen, können Sie diesen aber leicht freigeben. Dazu rufen Sie den Kontextmenü-Befehl **In freigegebene Datenquelle konvertieren** des Eintrags für die Datenquelle auf (siehe Bild 8). Danach erscheint der Eintrag **dsBestellverwaltung.rds** auch im Projektmappen-Explorer unter **Freigegebene Datenquellen**.

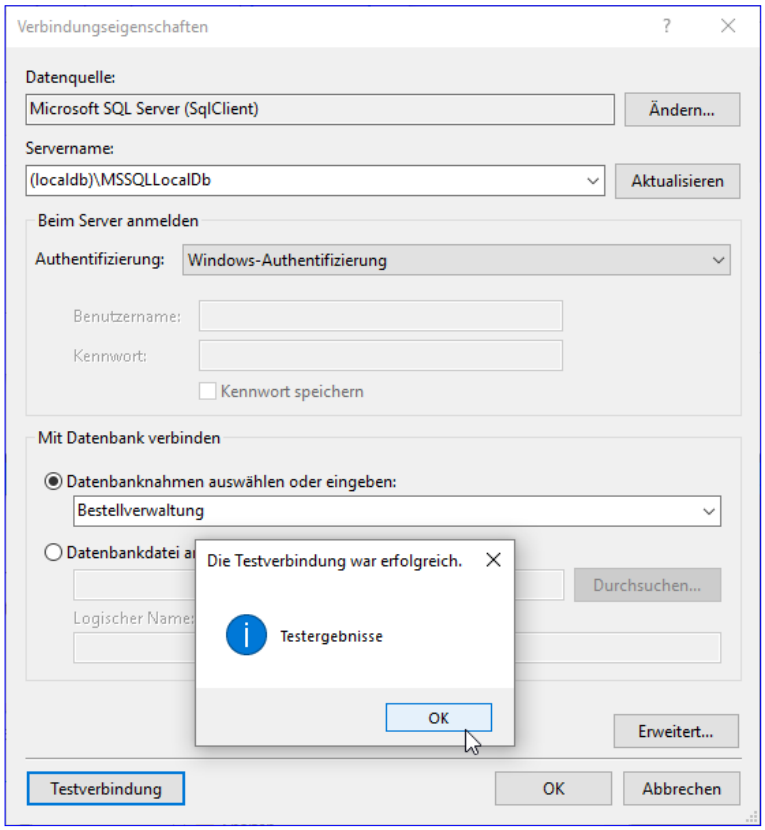

Bild 6: Einstellen und Testen der Verbindungseigenschaften

Dataset hinzufügen

Nachdem wir festgelegt haben, aus welcher Datenbank die Daten unseres Berichts stammen sollen, benötigen wir noch eine Tabelle oder Abfrage. Unsere Datenbank enthält keine Abfragen, daher müssen wir diese gegebenenfalls für den Bericht selbst erstellen.

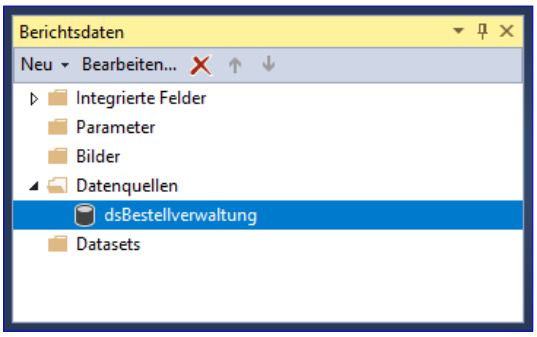

Bild 7: Neue Datenquelle für die Berichtsdaten

Aber erstmal wollen wir eine einfache Tabelle hinzufügen. Dazu verwenden wir den Kontextmenü-Eintrag **Dataset hinzufügen...** des **Datasets**-Eintrags im Bereich **Berichtsdaten**. Damit öffnen wir den Dialog **Dataseteigenschaften**. Hier können Sie wählen, ob Sie ein freigegebenes Dataset verwenden wollen oder ein nur in diesen Bericht eingebettetes Dataset. Erstere Option geht nicht, da wir noch kein freigegebenes Dataset erstellt haben. Also wählen wir die zweite Option.

Hier wählen wir dann als Datenquelle den Eintrag **dsDatenquelle** aus. Unter **Abfragetyp** finden wir dann allerdings nur die beiden Optionen **Text** und **Gespeicherte Prozedur** als aktive Elemente vor. Die Option **Tabelle** ist deaktiviert. Warum das? Wir probieren testweise, über die Schaltfläche **Abfrage-Designer...** eine Abfrage zu erstellen (siehe Bild 9).

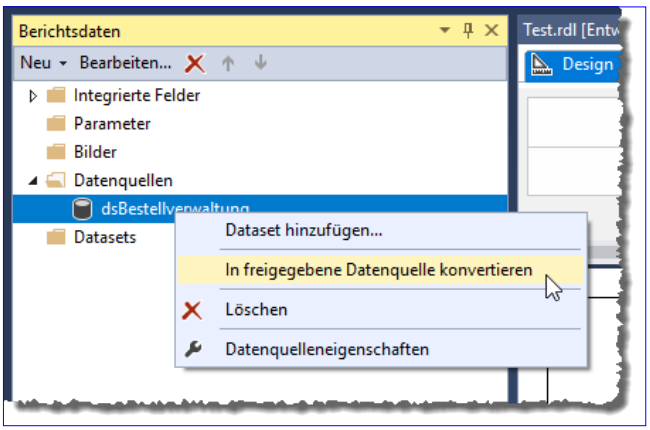

Bild 8: Datenquelle freigeben

Mit dem nun erscheinenden Dialog erscheint auch die Fehlermeldung aus Bild 10. Diese besagt, dass es Probleme mit der

Authentifizierungsmethode gibt. In den Verbindungseigenschaften der Datenquelle hatten wir jedoch Windows-Authentifizierung gewählt, was korrekt ist.

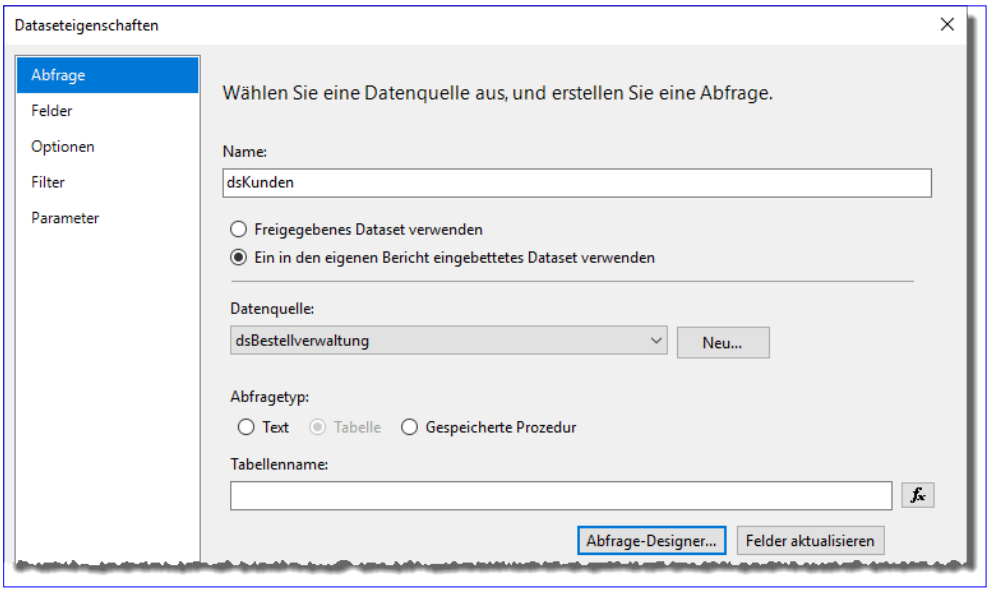

Bild 9: Einstellungen des Datasets

Anmeldeinformationen der Datenquelle anpassen

Also schauen wir uns nun nochmal die Datenquelleneigenschaften an, indem wir doppelt auf den Eintrag **dsBestellverwaltung** im Bereich **Berichtsdaten** klicken. Dies öffnet den Dialog **Datenquelleneigenschaften**, in dem sich Änderungen ergeben haben, seit wir die Datenquelle in eine freigegebene Datenquelle konvertiert haben.

Bild 10: Fehlermeldung beim Versuch, eine Verbindung mit der Datenquelle aufzubauen

Die Datenquelle des Berichts ist jetzt keine eingebettete Datenquelle mehr, sondern eine Datenquelle auf Basis der freigegebenen Datenquelle gleichen Namens.

Und obgleich wir weiter oben die Windows-Authentifizierung als Authentifizierungsmethode eingestellt haben, so ist diese nun nicht mehr aktiviert – und in den Eigenschaften der Datenquelle für den Bericht als eingebundene Datenquelle können wir auch gar keine Änderungen im Bereich **Anmeldeinformationen** des Dialogs **Datenquelleneigenschaften** vornehmen – die Eigenschaften sind alle deaktiviert (siehe Bild 11).

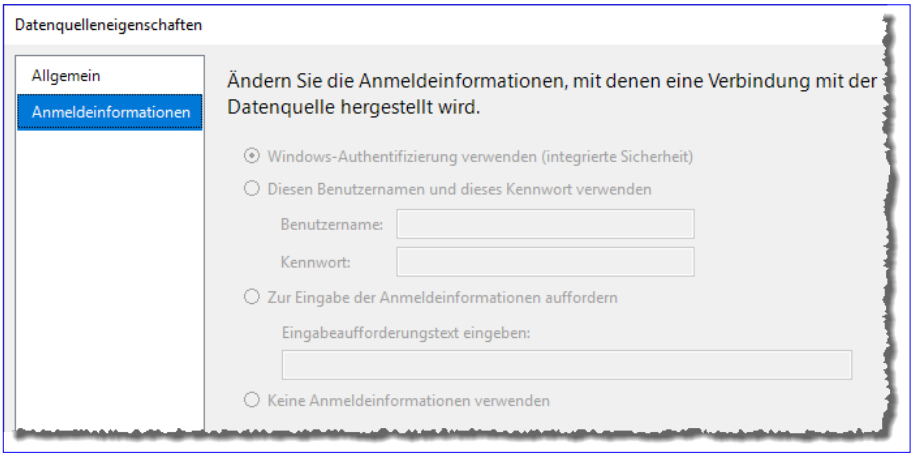

Bild 11: Die Anmeldeinformationen können in einer Datenquelle, die auf einer freigebebenen Datenquelle basiert, nicht aktualisiert werden.

Um diese Einstellungen zu ändern, müssen Sie den Dialog **Datenquelleneigenschaften** für die freigegebene Datenquelle im Projektmappen-Explorer öffnen. Hier stellen wir

dann fest, dass die Option **Keine Anmeldeinformationen verwenden** aktiviert ist (siehe Bild 12). Also stellen wir die Option **Windows-Authentifizierung verwenden** ein und schließen den Dialog wieder. Danach öffnen wir erneut den Dialog mit den Eigenschaften des neu hinzugefügten Datasets.

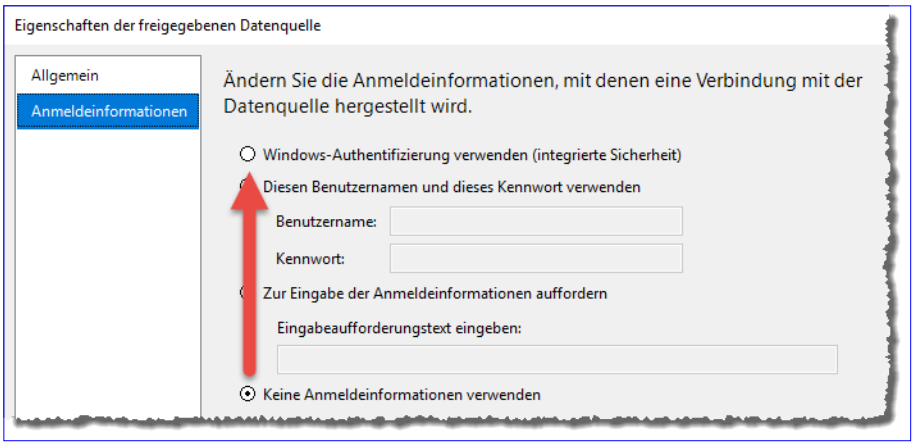

Bild 12: In der freigegebenen Datenquelle ist die Windows-Authentifizierung nicht aktiviert, was wir nun nachholen.

Dieser bietet nach der Auswahl der Datenquelle **dsBestellverwaltung** zwar immer noch nicht den Eintrag **Tabelle** als **Abfragetyp** an, aber wenn wir auf die Schaltfläche **Abfrage-Designer...** klicken, erscheint dieser immerhin ohne Fehlermeldung.

Bild 13: Steuerelemente im Abfrage-Designer

Abfrage entwerfen

Die Abfrage entwerfen wir nun im Dialog **Abfrage-Designer**. Dieser verfügt im oberen Bereich über eine Reihe von Steuerelementen (siehe Bild 13). Diese haben die folgenden Funktionen:

- **Als Text bearbeiten**: Blendet die Spalten-Auflistung aus und zeigt nur ein Textfeld zur Eingabe des SQL-Codes sowie den Bereich zur Ausgabe des Ergebnisses an.
- **Importieren...**: Importiert eine Abfrage aus einer Datei mit der Endung **.sql** oder **.rdl**.
- **Diagrammbereich anzeigen/ausblenden**: Blendet den Diagrammbereich ein oder aus.
- **Rasterbereich anzeigen/ausblenden**: Blendet den Rasterbereich ein oder aus.
- **SQL-Bereich anzeigen/ausblenden**: Blendet den SQL-Bereich ein oder aus.
- **Ergebnisbereich ein-/ausblenden**: Blendet den Ergebnisbereich ein oder aus.
- **Ausführen**: Führt die Abfrage aus und zeigt das Ergebnis im Ergebnisbereich an.
- **SQL überprüfen**: Überprüft die SQL-Syntax. Wird nur aktiviert, wenn eines der Felder einer der Tabellen im oberen Bereich markiert ist.

- **Sortieren**: Stellt die Sortierreihenfolge für das ausgewählte Feld beziehungsweise die ausgewählten Felder im oberen Bereich ein. Wird nur aktiviert, wenn eines der Felder eines der Tabelle im oberen Bereich markiert ist.

- **Filter entfernen**: Entfernt hinzugefügte Filter. Wird nur aktiviert, wenn eines der Felder im oberen Bereich markiert ist, für das zuvor ein Filter festgelegt wurde.

- **Gruppieren nach anzeigen/ausblenden**: Blendet die Spalte **Gruppieren nach** im Bereich der anzuzeigenden Felder ein.

- **Tabelle hinzufügen**: Öffnet den Dialog zum Hinzufügen der Tabellen aus der angegebenen Datenquelle.

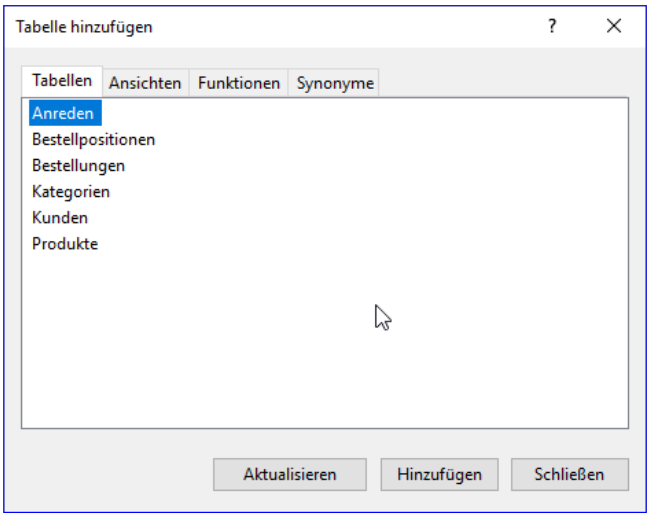

Bild 14: Auswahl der Tabellen für die Abfrage

Tabelle zur Abfage hinzufügen

Der erste Schritt zum Erstellen der Abfrage im Entwurfsmodus ist das Betätigen der Schaltfläche **Tabelle hinzufügen**, mit der Sie den Dialog aus Bild 14 öffnen. Hier wählen wir nun die Tabellen **Anreden** und **Kunden** aus und klicken auf die Schaltfläche **Aktualisieren**. Dadurch landen die beiden Tabellen im Diagrammbereich des Abfrage-Designers (siehe Bild 15). Danach schließen wir den Dialog **Tabelle hinzufügen**. Die Tabellen erscheinen direkt mit der zwischen ihnen definierten Beziehung.

Um Felder aus den Tabellen zum Rasterbereich hinzuzufügen, können Sie diese einfach mit einem Haken im jeweiligen Kontrollkästchen versehen. Die Felder werden dabei in der Reihenfolge hinzugefügt, in der Sie diese anhaken. Wenn Sie die Reihenfolge nachträglich ändern wollen, brauchen Sie nur in den Datensatzmarkierer vorn in der Zeile mit dem angedeuteten Pfeil nach rechts zu klicken und den Bereich an die gewünschte Stelle zu ziehen.

Währenddessen können Sie ständig die durch die Änderungen am Entwurf resultierenden Änderungen am SQL-Code im Bereich unter dem Rasterbereich verfolgen. Diesen Bereich können Sie ebenfalls manuell anpassen. Die Änderungen werden nach dem Verlassen des SQL-Bereichs auf die anderen Bereiche übertragen.

Sortierungen hinzufügen

Um eine Sortierung hinzuzufügen, klicken Sie im Diagrammbereich oben auf das Feld, nach dem sortiert werden soll. Dann betätigen Sie den Menüeintrag **Aufsteigend sortieren** oder **Absteigend sortieren**. Dies fügt dann in der Spalte **Sortierungsart** für das angegebene Feld den Wert **Aufsteigend** oder **Absteigend** hinzu. Wozu aber dient die Spalte **Sortierreihenfolge**? Das stellen Sie fest, wenn Sie mehrere Felder sortieren. In der Abbildung haben wir erst aufsteigend nach dem Feld **Nachname** sortiert und dann ebenfalls aufsteigend nach dem Feld **Vorname**. Das sorgt dafür, dass beide mit der Sortierungsart **Aufsteigend** versehen werden. Außerdem erhalten beide in der Spalte Sortierreihenfolge einen Index, der angibt, nach welchem Feld zuerst sortiert werden soll. Wenn Sie dies im Abfrageentwurf von Access erreichen wollen, müssen Sie die Felder in der entsprechenden Reihenfolge anordnen. Hier ist das etwas einfacher gelöst und es ist dazu kein Ändern der Reihenfolge der Felder für die Ausgabe nötig.