ISBN: 978-90-430-3242-1
NUR: 173
Trefw.: programmeertalen, C#

Dit is een uitgave van Pearson Benelux bv, Postbus 75598, 1070 AN Amsterdam
Website: www.pearson.nl – e-mail: amsterdam@pearson.com

Opmaak: CO2 Premedia, Amersfoort / Goedhart Ontwerp, Aarlanderveen
Omslag: Kok Korpershoek, Amsterdam

Vakinhoudelijke beoordeling: Koen Casier, Universiteit Gent; Tim Dams, Artesis Plantijn Hogeschool Antwerpen; Cees van Tilborg, Fontys.

Dit boek is gedrukt op een papiersoort die niet met chloorhoudende chemicaliën is gebleekt. Hierdoor is de productie van dit boek minder belastend voor het milieu.

HOOFDSTUK 25

Databases

Dit hoofdstuk behandelt:

- de aard van eenvoudige databases en de taal SQL;
- hoe SQL kan worden gebruikt in combinatie met C#;
- de C#-klassen die toegang bieden tot de database.

25.1 Inleiding

Dit hoofdstuk is bedoeld voor degenen die bekend zijn met een relationele database (bijvoorbeeld SQL Server) en die database willen bewerken met een C#-programma. De redenen hiervoor zouden kunnen zijn:

- Het C#-programma heeft een of andere vorm van gegevensopslag nodig en een database is hiervoor een zeer geschikt hulpmiddel.
- De programmeur wil de kracht van C# en zijn klassen benutten voor een krachtiger interface naar een database dan met een conventioneel, op zichzelf staand databaseproduct mogelijk is.

Misschien ben je (nog) niet helemaal vertrouwd met SQL (*Structured Query Language*), de taal om databases te bevragen. Daarom geven we hier een korte inleiding. We gebruiken hiervoor een voorbeelddatabase genaamd **MusicSales**. Een script genaamd `MusicSales.sql` kan je terugvinden op de website die bij dit boek hoort (www.pearsonmylab.nl). Dit script moet je uitvoeren om deze database aan te maken. Dit doe je als volgt:

1. Start Visual Studio.
2. Kies in het menu: **Tools > SQL Server > New Query.**
3. Er verschijnt een pop-up getiteld 'Connect to Server'. Bij het veld 'Server name' vul je in: `(localdb)\v11.0`
   Let hierbij op gebruik van ronde haakjes en backslash! De overige instellingen van dit venster laat je ongewijzigd.
4. Klik nu op 'Connect'. Er wordt een leeg bestand geopend met de naam `SQLQuery1.sql`.
5. Open het `MusicSales.sql`-script met een teksteditor (bijvoorbeeld Notepad) en kopieer de volledige inhoud van dit script naar `SQLQuery1.sql`.
6. Klik ten slotte op het 'Execute'-icoon links in dit venster. Dit is aangeduid in figuur 25.1.

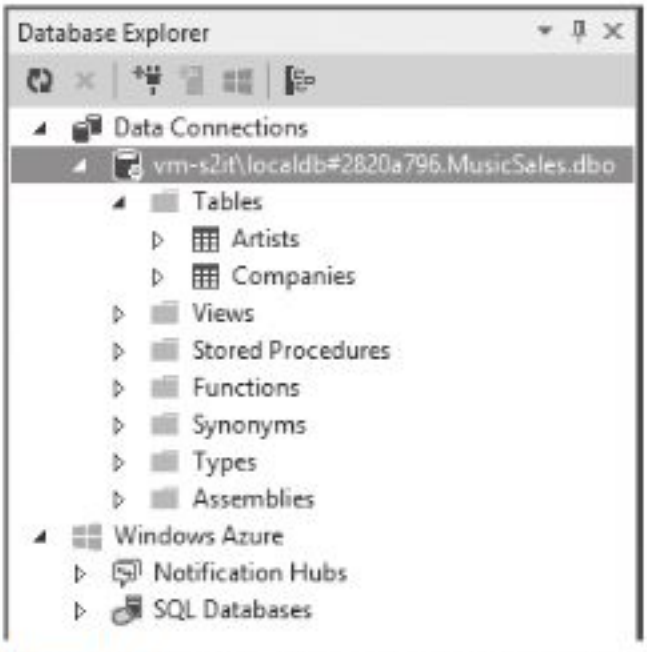
Figuur 25.1 Een script uitvoeren om de **MusicSales**-database aan te maken

Om te controleren of je script succesvol is uitgevoerd, onderneem je volgende stappen:

1. Gebruik je de Express-editie van Visual Studio, dan selecteer je in het menu **View > Other Windows > Database Explorer**; in de andere Visual Studio-versies selecteer je **View > Server Explorer**. Je kan voor beide versies ook gebruikmaken van een toetsenbordcombinatie: druk eerst Ctrl-W in en terwijl je de controltoets ingedrukt houdt, vervolgens ook de L-toets.
2. Klik op het stekkericoontje met als tooltip 'Connect to database'.

3. Als er een pop-up zou verschijnen (optioneel) over het type database, kies je voor Microsoft SQL Server en vervolgens klik je 'Continue'.
4. In het 'Add Connection'-venster geef je als server name opnieuw in: `(localdb)\v11.0`. Als alles goed is gegaan, kan je nu de **MusicSales**-database selecteren en op 'OK' klikken, zoals aangegeven in figuur 25.2:

Figuur 25.2 Een connectie naar de **MusicSales**-database



Figuur 25.3 Het **Database/Server Explorer**-venster

Links verschijnt er een **Database/Server Explorer**-venster met onder de Data Connections node de **MusicSales**-database. Hieronder kan je doorklikken naar 'Tables', met daaronder 'Artist' en 'Companies'. Deze twee tabellen worden in een volgende paragraaf uitgelegd.

25.2 De elementen van een database

De database die we zonet hebben aangemaakt, bestaat uit twee tabellen, `Artists` en `Companies`.

In tabel 25.1 is de tabel `Artists` weergegeven, met de uitvoerende artiesten, managers en muziekomzet (in miljoenen euro's).

Tabel 25.1 De tabel `Artists`

Artist	Company	Sales
The Visuals	ABC	65,2
RadioStar	ABC	22,7
Mike Parr	Media Ltd	3,5
The Objects	Class UK	12,6
Assignment 182	Media Ltd	34,6
The Trees	United Co	3,72

Deze data kan je ook vanuit Visual Studio oproepen. Klik met de rechtermuisknop in het **Database Explorer**-venster (Figuur 25.3) op de 'Artist'-tabel en kies voor 'Show Table Data'.

Let erop dat:

- de tabel is samengesteld uit een verzameling gelijksoortige *records*. In tabel 25.1 hebben we elk record weergegeven als een rij;
- een record is opgebouwd uit een aantal *velden*. In tabel 25.1 zien we drie velden: `Artist`, `Company` en `Sales`. Eigenlijk is een veldnaam niets anders dan een kolomnaam;
- we ervan uit zijn gegaan dat de artiestennamen uniek zijn, dus we hebben die als primaire sleutel gebruikt. Als de tabel echter bijvoorbeeld had bestaan uit namen van universiteitsstudenten, dan hadden we een uniek studentidentiteitsnummer moeten invoeren. In de bovenstaande tabel hebben we bij het creëren van het `Artist`-veld de naam meteen ingesteld als de primaire sleutel, waardoor het niet toegestaan is meerdere artiesten met dezelfde naam in te voeren.

Een database kan bestaan uit een of meer van dergelijke tabellen. In tabel 25.2 is de tweede tabel te zien, met de naam `Companies` (waarin `Company` als primaire sleutel voor de tabel is ingesteld), die een opsomming geeft van de namen van de bedrijven met daarbij de locatie.

Tabel 25.2 De tabel `Companies`

Company	Location
ABC Co	London
Media Ltd	Chicago
Class UK	Madrid
United Co	London

De relatie tussen de tabellen is dat de kolom `Company` van de tabel `Artists` een verwijzende sleutel is van de kolom `Company` in de tabel `Companies`. Dit is gespecificeerd tijdens het ontwerpen van de database.

25.4 De databaseklassen van C#

Gedurende zijn bestaan heeft Microsoft een reeks methoden voor databasemanipulatie geleverd, waaronder ODBC (*Open DataBase Connectivity*) en ADO (*ActiveX Data Objects*). Het .NET-framework gaat vergezeld van ADO.NET. Dit is een API (*Application Programming Interface*): een reeks klassen met de bijbehorende methoden die toegang bieden tot een complex item. Zo zijn bestandsklassen zoals `Directory` een soort API, in die zin dat ze programma's in staat stellen toegang te krijgen tot het bestandssysteem van Windows.

De ADO.NET-klassen die we zullen gebruiken, zijn:

```
SqlConnection
SqlCommand
SqlDataAdapter
DataTable
DataSet
CollectionViewSource
DataGrid
```

De `System.Data`- en `System.Data.SqlClient`-namespaces bevatten veel van dergelijke klassen; het gebruik hiervan zul je zien als we instanties gaan declareren.

Het ADO.NET-framework, waarvan we hieronder enkele voorbeelden beschrijven, is het oudste databaseframework in .NET en vormt een solide basis voor modernere frameworks. We denken hierbij dan vooral aan Entity Framework[1] met LINQ (*Language Integrated Query*). Hierbij worden de SQL-opdrachten bijna volledig afgeschermd

1 Meer info op http://msdn.microsoft.com/nl-be/data/ef.aspx.

van de programmeur en kan deze rechtstreeks werken met klassen die afgeleid zijn van de tabellen uit de database. Bovendien kan de programmeur deze klassen (en dus de bijhorende database) ondervragen via LINQ, een taal die erg lijkt op SQL maar volledig in C# is geïntegreerd. Een studie van deze frameworks valt buiten het bestek van dit boek, maar het is zeker de moeite waard om ook de 'oudere' manier van werken onder de knie te krijgen. Ten eerste zijn de nieuwe frameworks gebouwd met de klassen van ADO.NET onder de motorkap, en ten tweede levert het vaak snellere programma's op als je rechtstreeks met de ADO.NET-klassen werkt.

We bespreken nu kort de belangrijkste klassen.

De klasse `SqlConnection`

Deze klasse heeft te maken met de koppeling naar de database. We moeten deze voorzien van de verbindingsstring (*connection string*), waarin het type van de database is aangegeven (bijvoorbeeld Oracle, SQL Server) en de logingegevens. In ons voorbeeld hierna van de 'Record Navigator', stelt de wizard van Visual Studio deze string op, maar in het voorbeeld van de 'database example' doen we dat zelf.

De klasse `SqlDataAdapter`

Deze klasse biedt mogelijkheden om SQL-opdrachten naar een database te sturen. Hierbij kunnen gegevens worden overgestuurd van de database naar `DataTable`-objecten die tijdelijk in het RAM-geheugen zijn opgeslagen.

De klassenstructuur van `SqlDataAdapter` is tamelijk ingewikkeld: het heeft enkele properties (`UpdateCommand`, `InsertCommand` en `DeleteCommand`), die op hun beurt instanties zijn van een andere complexe klasse, `SqlCommand`. Deze klasse heeft weer andere properties en methoden (zoals `CommandText`).

Met de `Fill`-methode kunnen we de resultaten van een zoekhandeling in een gegevenstabel zetten. De laatste twee voorbeelden uit dit hoofdstuk tonen je hoe je deze klasse moet gebruiken.

De klasse `DataTable`

Instanties kunnen een tabel in het RAM-geheugen hebben. De gegevens worden aan de programmeur gepresenteerd in een rij-/kolom-vorm, vergelijkbaar met een tweedimensionale array. Merk op dat er niet wordt gezorgd voor een visuele representatie van een tabel.

We kunnen een gegevenstabel vullen met een exacte kopie van een tabel in een database, of we kunnen de records uitkiezen die we nodig hebben. Dit wordt bepaald door het SQL-statement dat we via de adapter naar de database sturen.

We kunnen toegang krijgen tot de afzonderlijke items van een gegevenstabel met behulp van de `Rows`-property, gevolgd door een rij- en kolomnummer, zoals in:

```
string name = Convert.ToString(dataTable.Rows[0][1]);
```

Let erop dat:

- de nummering van rijen en kolommen begint bij 0;
- de gegevenstabel instanties bevat van de klasse `object`. Dus wanneer we er een item uit halen, zetten we het item om naar het juiste type (als er bijvoorbeeld een item in een tekstvak moet worden weergegeven, zetten we het om naar een `string`).

Om te bepalen hoeveel rijen een gegevenstabel op een bepaald moment heeft, maken we gebruik van de `Count`-property, zoals in:

```
int size = dataTable.Rows.Count;
```

Om een gegevenstabel te wissen, gebruiken we de methode `Clear`.

De rij-/kolom-structuur zal je doen denken aan arrays en het is verleidelijk om de gegevenstabellen te gaan doorzoeken met lusconstructies. Dit moet echter worden vermeden. Probeer zo veel mogelijk in SQL te werken. Deze aanpak resulteert in eenvoudiger code. Bovendien worden deze SQL-instructies op de database uitgevoerd, die in de praktijk dikwijls op een erg performante server is geïnstalleerd. Als je lusconstructies in C# schrijft, dan worden deze door de applicatie uitgevoerd op een minder krachtige pc dan de databaseserver. Het voorbeeld getiteld 'DataGridView' toont het gebruik van de gegevenstabel.

De klasse `CollectionViewSource`

Deze klasse voorziet in de weergave van een `DataTable` in rij-/kolom-formaat. Via de XAML-code die de GUI opbouwt gaan we via een speciale notatie (genaamd data binding) een DataTable verbinden aan een CollectionViewSource, zodat we in de C#-code dit object kunnen gebruiken om door de records te navigeren:

```
<CollectionViewSource x:Key="artistsViewSource"
   Source="{Binding Artists, Source={StaticResource musicSalesDataSet}}"/>
```

Bovenstaande XAML-code hoef je niet zelf te schrijven, maar wordt door de DataSource Wizard gegenereerd. We illustreren dit in het 'Record Navigator'-voorbeeld.

25.5 Voorbeeld 1: Record Navigator

De eerste fase in het toegang krijgen tot een bestaande database vanuit een programma heeft betrekking op het toevoegen van een gegevensverbinding. Voordat je deze daadwerkelijk kunt toevoegen, moet je eerst de volgende stappen zetten:

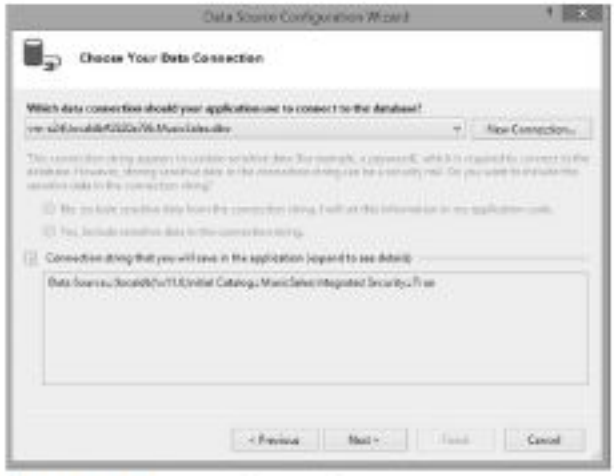
1. Creëer een nieuw WPF-project en noem dit **RecordNavigator**.
2. Zorg dat je de **MusicSales**-database correct hebt geïmporteerd, op de manier zoals we in de inleiding hebben beschreven. Je hoeft dit niet voor elk nieuw project te doen: deze database is na de import beschikbaar voor al je projecten. Als je echter met een schone lei wil starten, kan je deze import herhalen.
3. Start nu de Wizard om een Data Connection aan te maken. Ga hiervoor naar het **Data Sources**-venster (**View > Other Windows > Data Sources**) en klik op het icoon links-

boven met de tooltip 'Add New Data Source'. De start van deze wizard is voorgesteld in figuur 25.4:

Figuur 25.4 Data Source Configuration Wizard

4. Door op dit scherm op de 'Next'-knop te klikken, verschijnt de volgende stap (figuur 25.5) en kies je voor het databasemodel. De enige keuze is hier 'Dataset'. Hierna klik je weer op de 'Next'-knop.
5. Nu stelt de wizard voor om de gegevensverbinding in te stellen. Doordat je reeds een verbinding gemaakt hebt met de **MusicSales**-database (toen je de import testte), kan je deze via de combobox selecteren (figuur 25.6). Is dit niet het geval, kan je nog steeds een verbinding maken via de 'New Connection'-knop. Merk ook op dat de connection string nu wordt ingevuld. De details hiervan kan je uitlezen door op het +-teken te klikken op dit scherm.

Figuur 25.5 Databasemodel



Figuur 25.6 Connection string

6. De volgende stap (figuur 25.7) zal je vragen om deze connection string te bewaren. Bevestig dit door de checkbox aan te vinken. Later zal je deze connection string kunnen terugvinden in het bestand `App.config` van je project.
7. De laatste stap van de wizard bestaat eruit om te selecteren welke tabellen uit de database je nodig hebt voor je programma (figuur 25.8). Selecteer hier de beide tabellen: `Artists` en `Companies`. Ten slotte klik je op de 'Finish'-knop om de wizard te beëindigen.

De wizard heeft nu klassedefinities gemaakt om de `Artists`- en `Companies`-tabellen te benaderen vanuit C#. Deze definities zijn terug te vinden in het bestand `Music-`

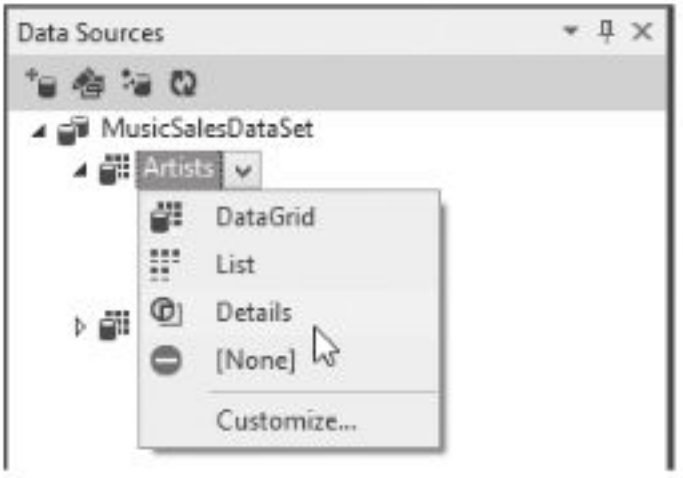
Figuur 25.7 Bewaren connection string

Figuur 25.8 Tabellen kiezen

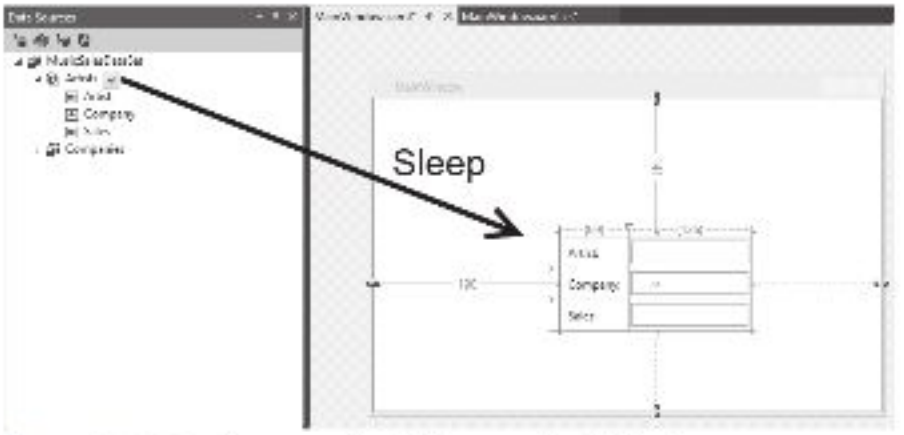
`SalesDataSet.xsd`, dat is toegevoegd aan de **Solution Explorer** van je project. In het **Data Sources**-venster zie je ook beide tabellen weergegeven.

Het is nu tijd om de GUI van het programma op te bouwen, maar voordat we hiermee kunnen starten, moeten we de klassedefinities compileren. Doe dit via het menu **Build > Rebuild Solution**. Als je dit zou vergeten, zouden de onderstaande stappen een foutmelding kunnen opleveren.

Als eerste gaan we instellen dat we in het programma de `Artists`-tabel record per record willen doorlopen. Hiervoor selecteer je in een combobox de waarde 'Details'. Deze combobox bevindt zich in het **Data Sources**-venster (figuur 25.9).



Figuur 25.9 Data Source Details



Figuur 25.10 Verslepen van `Artists` naar `MainWindow`

Vervolgens selecteer je met de muis in datzelfde venster de Artists-tabel en sleep je die naar het ontwerpvenster van MainWindow.xaml. Eventueel dien je eerst ervoor te zorgen dat het bestand MainWindow.xaml geopend is. Het resultaat van deze actie is te zien in figuur 25.10:

Visual Studio heeft door deze sleepbewerking volgende zaken gerealiseerd:

- Drie Label- en drie TextBox-elementen zijn aangemaakt en geplaatst in een nieuwe grid, genaamd grid1. Voor de Text-property van elke TextBox zie je in het XAML-bestand een uitdrukking met accolades, bijvoorbeeld:

```
Text={Binding Company, Mode=TwoWay}
```

  Deze uitdrukking zorgt ervoor dat gegevens uit de tabel Artists in de TextBox-elementen gezet worden en dat gewijzigde gegevens door de gebruiker naar de database kunnen worden weggeschreven.
- Een datastructuur wordt opgezet om tabelgegevens tijdelijk in het RAM-geheugen te plaatsen. Deze zie je bovenaan in het XAML-bestand:

```
<Window.Resources>
  <local:MusicSalesDataSet x:Key="musicSalesDataSet"/>
  <CollectionViewSource x:Key="artistsViewSource"
   Source="{Binding Artists,
            Source={StaticResource musicSalesDataSet}}"/>
</Window.Resources>
```

- Een stukje C#-code in het Loaded-event van de MainWindow-klasse zorgt ervoor dat de gegevens uit de database worden opgehaald en in de bovenstaande datastructuren geladen.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{

    RecordNavigator.MusicSalesDataSet musicSalesDataSet =
      ((RecordNavigator.MusicSalesDataSet)
            (this.FindResource("musicSalesDataSet")));
    // Load data into the table Artists.
    // You can modify this code as needed.
    RecordNavigator.MusicSalesDataSetTableAdapters.ArtistsTableAdapter
      musicSalesDataSetArtistsTableAdapter
      = new RecordNavigator.MusicSalesDataSetTableAdapters.
        ArtistsTableAdapter();
    musicSalesDataSetArtistsTableAdapter.Fill(musicSalesDataSet.Artists);
    System.Windows.Data.CollectionViewSource artistsViewSource =
      ((System.Windows.Data.CollectionViewSource)
            (this.FindResource("artistsViewSource")));
    artistsViewSource.View.MoveCurrentToFirst();
}
```

Deze code kan je vereenvoudigen, omdat alle klassen voorzien zijn van hun volledige namespace. Bovendien declareren we de aangemaakte objecten als private-membervariabelen, omdat we ze nog nodig gaan hebben in andere methoden. Je krijgt dan het volgende:

Project: h25\RecordNavigator | Bestand: MainWindow.xaml.cs

```
using RecordNavigator.MusicSalesDataSetTableAdapters;
...
public partial class MainWindow : Window
{
    CollectionViewSource artistsViewSource;
    ArtistsTableAdapter musicSalesDataSetArtistsTableAdapter;
    MusicSalesDataSet musicSalesDataSet;

    public MainWindow()
    {

        InitializeComponent();
    }

    private void Window_Loaded(object sender, RoutedEventArgs e)
    {
        // Load data into the table Artist.
        // You can modify this code as needed.
        musicSalesDataSet =
          ((MusicSalesDataSet)(this.FindResource("musicSalesDataSet")));
        musicSalesDataSetArtistsTableAdapter =
          new ArtistsTableAdapter();
        musicSalesDataSetArtistsTableAdapter.Fill(
                                              musicSalesDataSet.Artists);
        artistsViewSource =
         ((CollectionViewSource)(this.FindResource("artistsViewSource")));
        artistsViewSource.View.MoveCurrentToFirst();
    }
...
}
```

Als je het programma op dit moment zou starten, krijg je enkel het eerste record van de tabel `Artists` te zien. We willen echter stap voor stap door alle records gaan en eventuele aanpassingen weer wegschrijven. Daarom voegen we de nodige knoppen toe, zodat het geheel eruitziet zoals in figuur 25.11. De knoppen die we hebben toegevoegd zijn:

- `firstButton`: navigeert naar het eerste record;
- `backButton`: navigeert naar het vorige record als er één is;
- `nextButton`: navigeert naar het volgende record als er één is;
- `lastButton`: navigeert naar het laatste record;
- `saveButton`: bewaart de wijzigingen die in het huidige record zijn aangebracht.

Figuur 25.11 Het **Record Navigator**-programma

Daarnaast hebben we nog een `TextBlock` genaamd `positionTextBox` toegevoegd om de huidige positie weer te geven. Voor de concrete XAML-code verwijzen we naar de website die hoort bij dit boek (www.pearsonmylab.nl).

Door het implementeren van de nodige `Click`-event-handlers, maken we het programma compleet. Merk hierbij op dat we gebruikmaken van het object `artistsViewSource`, dat door de wizard is aangemaakt. Dit object zorgt ervoor dat we kunnen navigeren door alle records. Om de wijzigingen op te slaan, volstaat het om de `Update`-methode van `TableAdapter`-object aan te roepen.

Project: h25\RecordNavigator | Bestand: MainWindow.xaml.cs

```
private void updatePositionStatus()
{
    positionTextBox.Text =
       String.Format("{0} of {1}",
                     artistsViewSource.View.CurrentPosition + 1,
                     musicSalesDataSet.Artists.Count);
}

private void backButton_Click(object sender, RoutedEventArgs e)
{
    if (artistsViewSource.View.CurrentPosition > 0)
    {
        artistsViewSource.View.MoveCurrentToPrevious();
        updatePositionStatus();
    }
}

private void nextButton_Click(object sender, RoutedEventArgs e)
{
    if (artistsViewSource.View.CurrentPosition <
          ((CollectionView)artistsViewSource.View).Count - 1)
    {
        artistsViewSource.View.MoveCurrentToNext();
        updatePositionStatus();
    }
}

private void firstButton_Click(object sender, RoutedEventArgs e)
{
    artistsViewSource.View.MoveCurrentToFirst();
    updatePositionStatus();
}

private void lastButton_Click(object sender, RoutedEventArgs e)
{
    artistsViewSource.View.MoveCurrentToLast();
    updatePositionStatus();
}

private void saveButton_Click(object sender, RoutedEventArgs e)
{
    musicSalesDataSetArtistsTableAdapter.Update(musicSalesDataSet.
      Artists);
}
```

In een volgend voorbeeld tonen we hoe we meerdere records tegelijk kunnen tonen via een `DataGrid`.

25.6 Voorbeeld 2: een `DataGrid`-programma

Dit programma toont de `DataGrid` in actie. In figuur 25.12 is het scherm weergegeven.

De gebruiker voert een omzetwaarde in en er wordt een SQL-query geconstrueerd die een gegevenstabel vult met records waarvan de omzet groter of gelijk is aan de ingevoerde waarde. Bij een waarde 0 wordt ieder record geselecteerd.

Net zoals in het eerste voorbeeld starten we met een nieuw project en voeren we de 'Data Source Configuration Wizard' uit. In het **DataSources**-venster zorgen we er nu echter voor dat de `Artists`-tabel ingesteld staat op 'DataGrid' en niet op 'Details' voordat we de tabel slepen naar `MainWindow.xaml`.

Vervolgens gaan we een nieuwe query definiëren die een parameter accepteert die het minimum aan sales bevat dat elk record dient te hebben. Hiervoor dubbelklik je op het `MusicSalesDataSet.xsd`-bestand. Vervolgens zie je een diagram verschijnen met de `Artists`-tabel en de `ArtistsTableAdapter`. Klik nu met de rechtermuisknop op die laatste en kies voor **Add > Query**. Een wizard wordt gestart om deze nieuwe query toe te voegen (figuur 25.13).

Figuur 25.12 Het `DataGrid`-programma

Figuur 25.13 Nieuwe query toevoegen

Kies in dit startscherm voor het gebruik van SQL-statements. Dit zou de standaard-instelling moeten zijn. Klik nu op de 'Next'-knop.

Figuur 25.14 Query Type

Figuur 25.15 SELECT-statement schrijven

In het volgende scherm duid je aan dat je een query wil schrijven die mogelijk meerdere rijen kan teruggeven. Klik weer op de 'Next'-knop.

Schrijf nu het SELECT-statement dat je wil gebruiken om enkel die artiesten te selecteren die meer dan een bepaald bedrag aan muziek verkocht hebben. Dit bedrag zal ingevuld worden door de gebruiker en kan nu nog niet meegegeven worden. Daarom werken we met een parameter. Parameters worden voorafgegaan door een @-teken. De query luidt dus:

```
SELECT Artist, Company, Sales FROM dbo.Artists
WHERE Sales >= @sales
```

In het voorlaatste scherm gaan we de methode benoemen, zoals aangegeven in figuur 25.16.

We kiezen voor de methodenaam `FillBySales`. Merk ook op dat we de tweede optie om een `DataTable` terug te geven uitvinken, omdat ze voor deze toepassing overbodig is. Klik vervolgens weer op de 'Next'-knop, zodat je in figuur 25.17 een overzicht krijgt van wat er is gegenereerd aan code:

Figuur 25.16 Methodenaam

Figuur 25.17 Resultaat van de wizard

Nu gaan we de C#-code schrijven. Net zoals in het vorige voorbeeld, is er code aangemaakt die de gegevens uit de database ophaalt en op het scherm zet. Na vereenvoudiging en declaratie van de nodige member-variabelen wordt dit:

Project: h25\DataGridView | Bestand: MainWindow.xaml.cs

```
private MusicSalesDataSet musicSalesDataSet;
private ArtistsTableAdapter artistsTableAdapter;
private CollectionViewSource artistsViewSource;

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    musicSalesDataSet =
        ((DataGridView.MusicSalesDataSet)
         (this.FindResource("musicSalesDataSet")));
    // Load data into the table Artists.
    // You can modify this code as needed.
    artistsTableAdapter = new ArtistsTableAdapter();
    artistsTableAdapter.Fill(musicSalesDataSet.Artists);
```

```
    artistsViewSource = ((CollectionViewSource)
                         (this.FindResource("artistsViewSource")));
    artistsViewSource.View.MoveCurrentToFirst();
}
```

Door het klikken op de knop, wordt een getal van het tekstvak ingelezen en omgezet naar een `decimal` (het correcte type om met geldbedragen te werken). Daarna volgt een aanroep van `FillBySales` die de query zal uitvoeren die we via de wizard hebben aangemaakt. Dankzij het mechanisme van data binding, zal het scherm automatisch worden ververst.

Project: h25\DataGridView | Bestand: MainWindow.xaml.cs

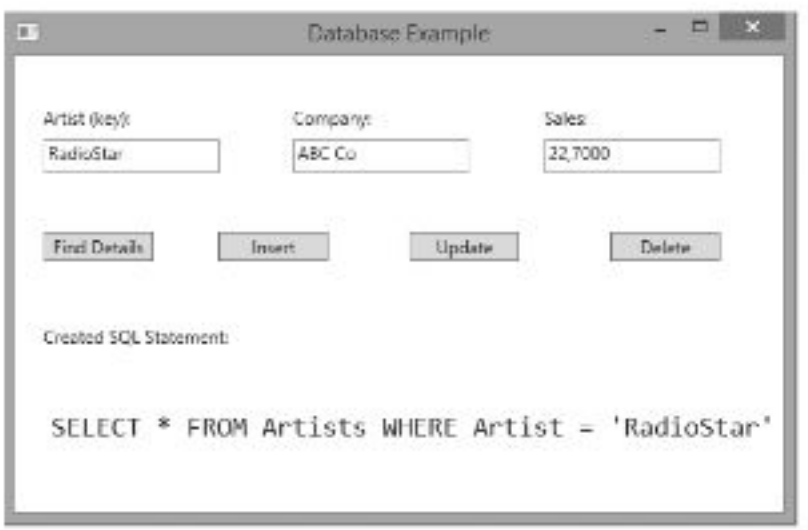
```
private void highSalesButton_Click(object sender, RoutedEventArgs e)
{
    decimal minsales = Convert.ToDecimal(salesAboveTextBox.Text);
    artistsTableAdapter.FillBySales(musicSalesDataSet.Artists, minsales);
}
```

TESTVRAAG

25.1 *Hoe zou je kunnen zorgen voor een 'Show All Records'-knop voor de gebruiker?*

25.7 Voorbeeld 3: SQL-voorbeeld

Dit programma levert een formulier waarmee de gebruiker records kan zoeken, verwijderen, invoegen en aanpassen. Bovendien is dit programma een voorbeeld waarbij geen wizards meer gebruikt worden en alle code zelf ingegeven moet worden.



Figuur 25.18 Het SQL-voorbeeldprogramma

In figuur 25.18 is het hoofdvenster weergegeven. De code luidt:

Project: h25\DatabaseExample | Bestand: MainWindow.xaml.cs

```
private string connectionString =
  @"Data Source=(localdb)\v11.0;" +
  @"Initial Catalog=MusicSales;Integrated Security=True";

...
private void findButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        DataTable table = new DataTable();

        // set up an SQL query
        string command = "SELECT * FROM Artists WHERE " +
                         "Artist = '" + artistTextBox.Text + "'";
        SqlDataAdapter adapter =
                  new SqlDataAdapter(command, connectionString);
        sqlLabel.Content = command;

        // do the query
        table.Clear();
        int recordCount = adapter.Fill(table);

        // display results
        if (recordCount != 0)
        {
            companyTextBox.Text = Convert.ToString(table.Rows[0][1]);
            salesTextBox.Text = Convert.ToString(table.Rows[0][2]);
        }
        else
        {
            MessageBox.Show("Artist not found!");
        }
    }
    catch (Exception obj)
    {
        MessageBox.Show(obj.Message);
    }
}

private void insertButton_Click(object sender, RoutedEventArgs e)
{
    SqlConnection connection = new SqlConnection(connectionString);
    SqlCommand insertCommand = new SqlCommand();
    SqlDataAdapter adapter = new SqlDataAdapter();

    try
    {
        connection.Open();
        string command = "INSERT INTO Artists" +
                         "(Artist, Company, Sales) " +
                         "VALUES('" + artistTextBox.Text + "', '" +
                         companyTextBox.Text + "', " +
                         salesTextBox.Text + ")";
        sqlLabel.Content = command;
```

```
        // put the command in the adapter
        insertCommand.Connection = connection;
        insertCommand.CommandText = command;
        adapter.InsertCommand = insertCommand;

        // do the insert
        adapter.InsertCommand.ExecuteNonQuery();
    }
    catch (Exception obj)
    {
        MessageBox.Show(obj.Message);
    }
    finally
    {
        connection.Close();
    }
}

private void updateButton_Click(object sender, RoutedEventArgs e)
{
    SqlConnection connection = new SqlConnection(connectionString);
    DataTable table = new DataTable();
    SqlCommand updateCommand = new SqlCommand();
    try
    {
        connection.Open();
        SqlDataAdapter adapter = new SqlDataAdapter();

        // set up an SQL update
        string command = "UPDATE Artists SET Company = '" +
                         companyTextBox.Text + "', Sales = '" +
                         salesTextBox.Text + "' WHERE Artist = '" +
                         artistTextBox.Text + "'";
        sqlLabel.Content = command;

        // put the command in the adapter
        updateCommand.CommandText = command;
        updateCommand.Connection = connection;
        adapter.UpdateCommand = updateCommand;

        // do the update
        adapter.UpdateCommand.ExecuteNonQuery();
    }
    catch (Exception obj)
    {
        MessageBox.Show(obj.Message);
    }
    finally
    {
        connection.Close();
    }
}

private void deleteButton_Click(object sender, RoutedEventArgs e)
{
    SqlConnection connection = new SqlConnection(connectionString);
    SqlCommand deleteCommand = new SqlCommand();
```

```
    SqlDataAdapter adapter = new SqlDataAdapter();

    try
    {
        connection.Open();
        string command = "DELETE FROM Artists WHERE Artist = '" +
                         artistTextBox.Text + "'";
        sqlLabel.Content = command;

        // put the command in the adapter
        deleteCommand.Connection = connection;
        deleteCommand.CommandText = command;
        adapter.DeleteCommand = deleteCommand;

        // do the insert
        int rowsAffecting = adapter.DeleteCommand.ExecuteNonQuery();
        if (rowsAffecting == 0)
        {
            MessageBox.Show("Deletion not executed.");
        }
        else
        {
            MessageBox.Show("Artist deleted.");
        }
    }
    catch (Exception obj)
    {
        MessageBox.Show(obj.Message);
    }
    finally
    {
        connection.Close();
    }
}
```

Er zijn enkele punten die we bij het programma moeten opmerken:

- De gebruikersinterface is niet foutenvrij. Zo kan er worden geprobeerd om iets in te voegen, ook als alle velden leeg zijn. Dit is een opzettelijke omissie om de databasecode overzichtelijk en eenvoudig te houden.
- De drie tekstvakken hebben de volgende namen gekregen: `artistTextBox`, `companyTextBox` en `salesTextBox`.
- Het label onder aan het formulier heet `sqlLabel`. De knoppen heten respectievelijk `findButton`, `insertButton`, `updateButton` en `deleteButton`.
- De code voor het invoegen, verwijderen en aanpassen is vergelijkbaar. Deze omvat het creëren van een instantie van `SqlCommand`, het instellen van de properties (`Connection` en `CommandText`) hiervan en het toekennen van de juiste property van een adapter. Ten slotte voert `ExecuteNonQuery` de feitelijke opdracht uit.
- Telkens als we een SQL-instructie creëren, geven we deze eerst weer in `sqlLabel` onder aan het formulier en voeren deze daarna pas uit. De reden hiervoor is dat hoewel de C#-compiler gedetailleerde controles uitvoert op je C#-code, deze niet de inhoud van strings controleert en niets van SQL weet. Daarom worden fouten in SQL-opdrachten pas ontdekt in de uitvoerfase en is het nuttig om een scherm

met de SQL-code te hebben. Veel fouten zijn terug te voeren op fouten met enkele aanhalingstekens.

- Let ook op voor het fenomeen genaamd *SQL Injection*: hierbij geeft een kwaadwillige gebruiker in een tekstvak van het programma SQL code, bijvoorbeeld:

```
;drop table Sales
```

  Let op de puntkomma vooraan, wat in SQL een manier is om aan te geven dat er een nieuwe opdracht begint. Als de programmeur de invoer van de gebruiker niet, of onvolledig, controleert, dreigt dus het gevaar dat er belangrijke gegevens verloren gaan!
- Het is van groot belang om exception-handling te gebruiken. In dit programma tonen we de `Message`-property, die een verklaring geeft van alle fouten. Voorbeelden van fouten die zouden kunnen worden ontdekt zijn:
  1. Proberen een nieuwe record toe te voegen met een bedrijf dat niet in de tabel `Companies` voorkomt.
  2. Proberen een record toe te voegen met een niet-numerieke omzetwaarde.
- Bij het verwijderen, invoegen en aanpassen moeten we de verbinding openen en sluiten. We plaatsen de `Close`-aanroep binnen de `finally`-sectie van de exception-handlingcode, zodat deze wordt uitgevoerd ongeacht wat er binnen het `try`-blok gebeurt.

TESTVRAAG

25.2 *Welke tekstvakken in het SQL-voorbeeldprogramma hebben waarden nodig bij:*
a. het verwijderen van een record?
b. het invoegen van een record?

Programmeerprincipes

Dit hoofdstuk introduceert geen nieuw C#-principe; we gebruiken de kracht van de ADO.NET-klassen. Bovendien hebben we kennisgemaakt met enkele wizards die het programmeerwerk verlichten.

Programmeervalkuilen

- Bij het creëren van SQL-strings worden snel fouten gemaakt. Denk aan de enkele aanhalingstekens en geef, bij het debuggen, de SQL-instructie weer.
- Bij het uitvoeren van queries met `Fill`, hoef je geen `Open` en `Close` te gebruiken. In alle andere gevallen is dit wel nodig.
- Voldoende controle inbouwen om SQL Injection te vermijden.