Note: This example exercise is based on ADO.Net Entity Data Model, Entity Framework and LINQ to Database.

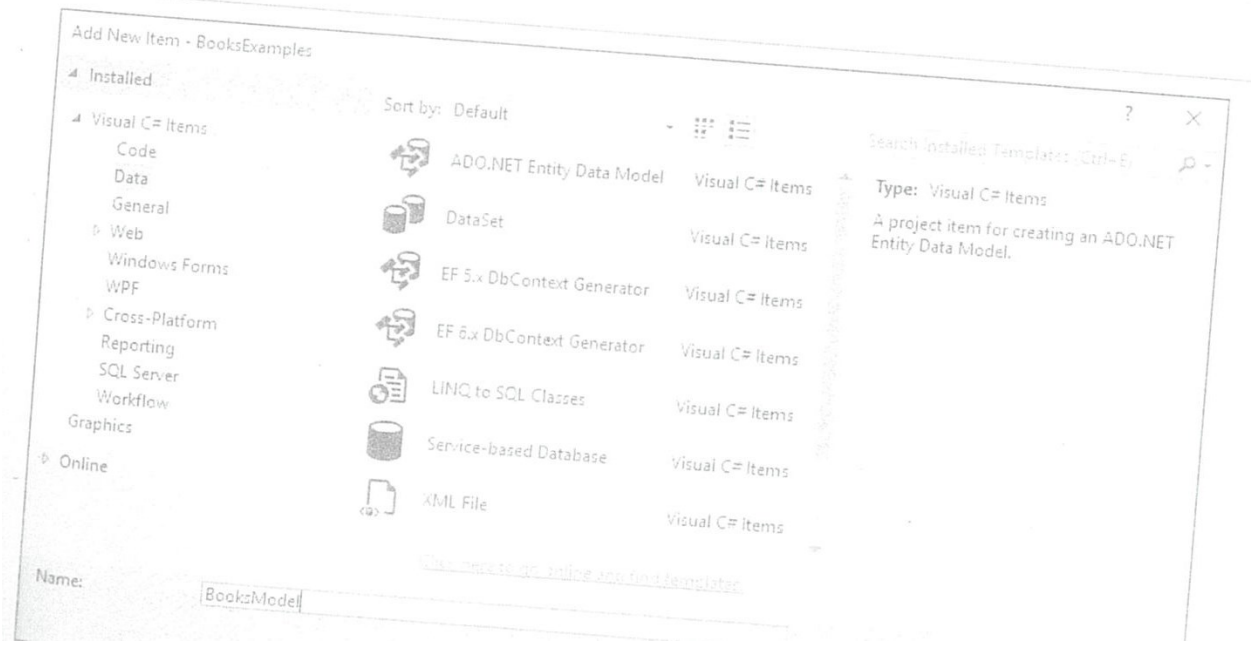
Exercise-01:

A) Creating the ADO.NET Entity Data Model Class Library

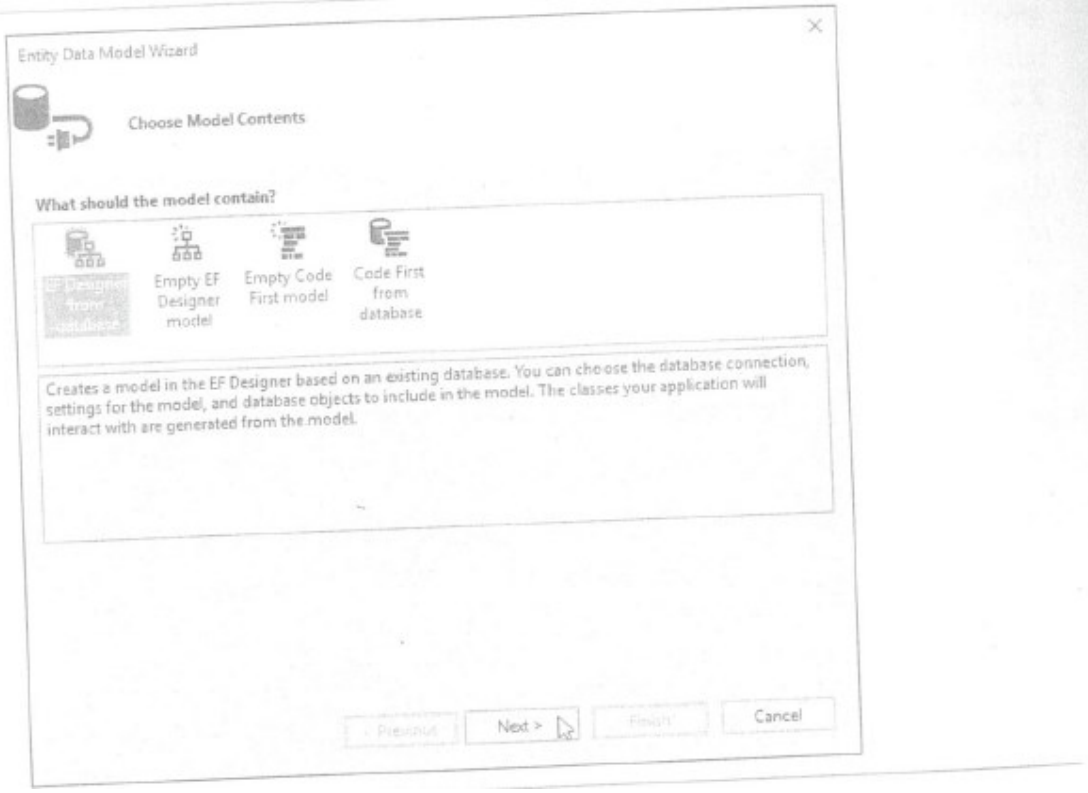
Step 1: Creating a Class Library Project for the ADO.NET Entity Data Model
Select **File > New > Project...** to display the **New Project** dialog, then select **Class Library** from the **Visual C#** templates and name the project `BooksExamples`. Click **OK** to create the project, then delete the `Class1.cs` file from the **Solution Explorer**.

Step 2: Adding the ADO.NET Entity Data Model to the Class Library
To interact with the database, you'll add an ADO.NET entity data model to the class library project. This will also configure the connection to the database.

1. *Adding the ADO.NET Entity Data Model.* Right click the `BooksExamples` project in the **Solution Explorer**, then select **Add > New Item...** to display the **Add New Item** dialog (Fig. 22.11). From the **Data** category select **ADO.NET Entity Data Model** and name the model `BooksModel`—this will be the name of a file (with the file-name extension `.edmx`) that configures the entity data model. Click **Add** to add the entity data model to the class library and display the **Entity Data Model Wizard** dialog.

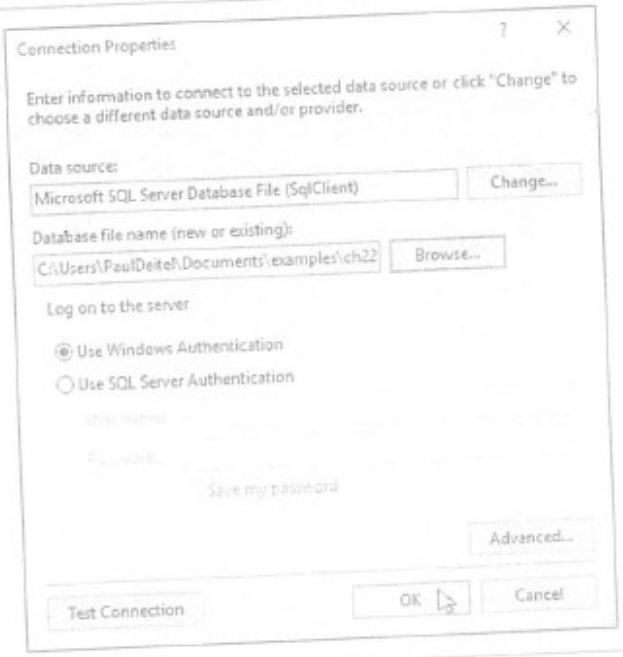


Choosing the Model Contents. The **Choose Model Contents** step in the **Entity Data Model Wizard** dialog (Fig. 22.12) enables you to specify the entity data model's contents. The model in these examples will consist of data from the **Books** database, so select **EF Designer from database** and click **Next >** to display the **Choose Your Data Connection** step.

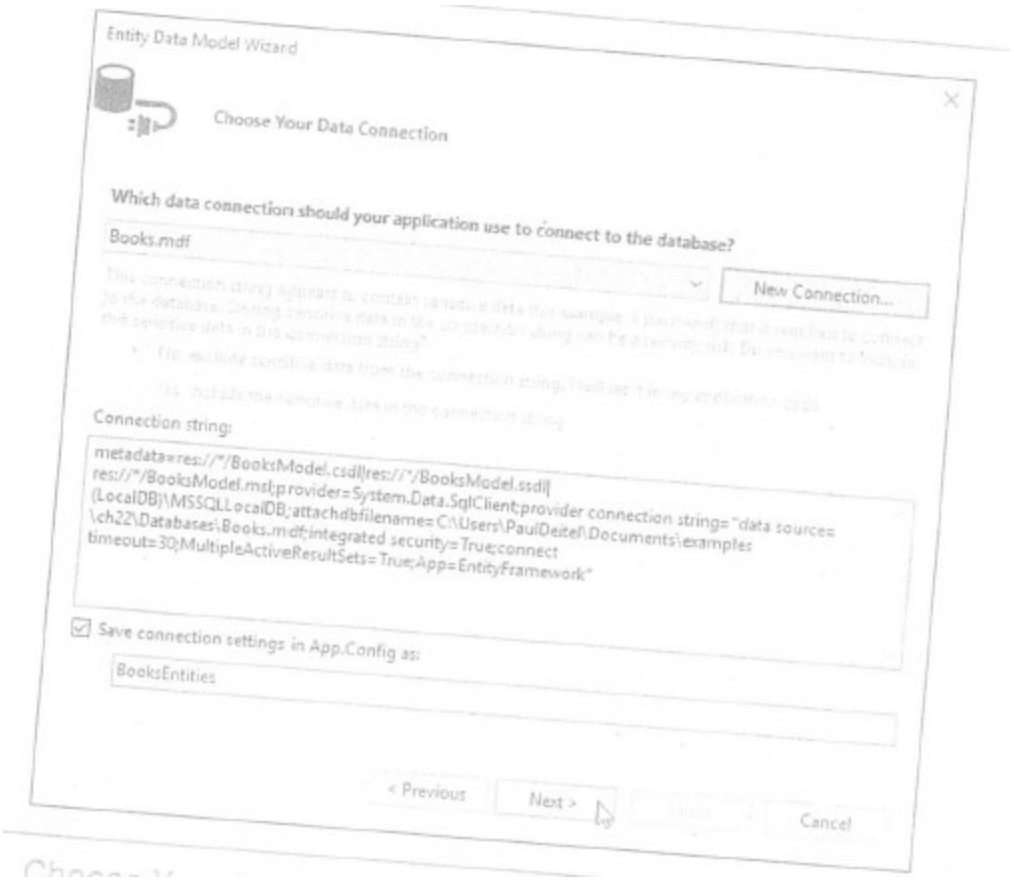


Step 03:

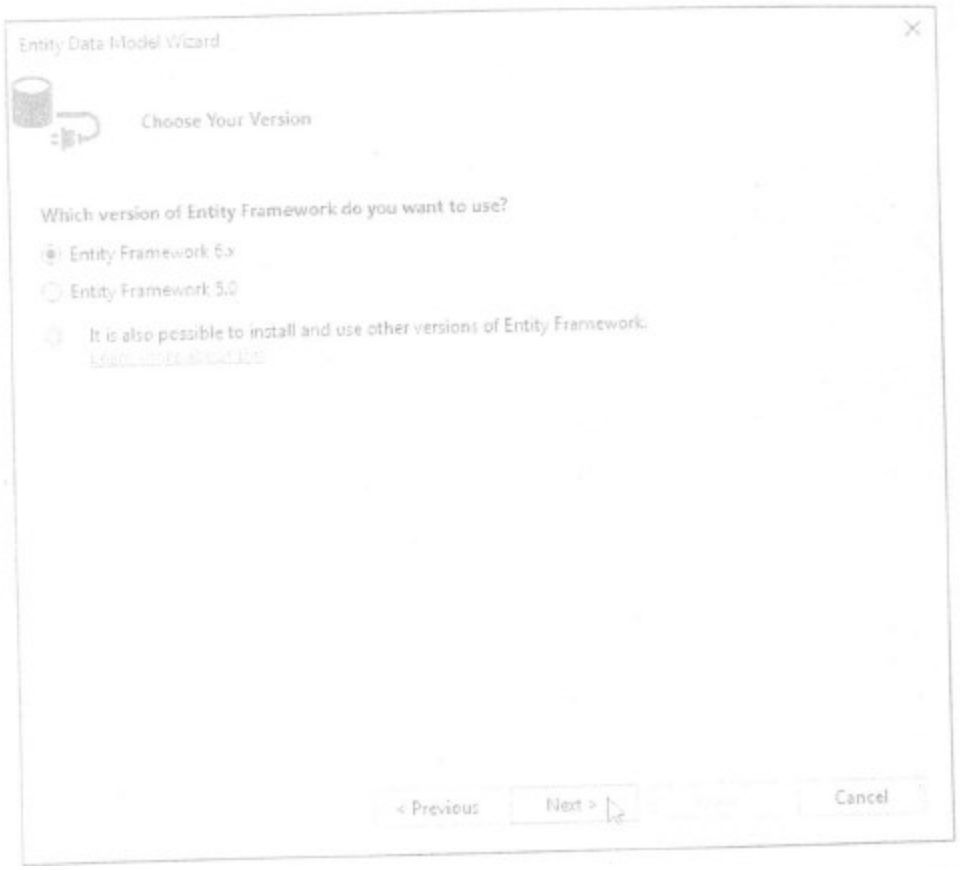
Choosing the Data Connection. In the **Choose Your Data Connection** step, click **New Connection...** to display the **Connection Properties** dialog (Fig. 22.13). (If



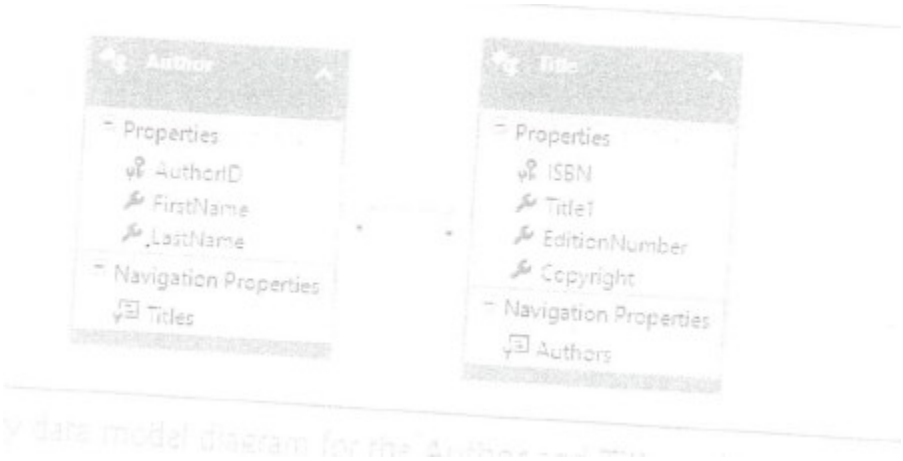
the IDE displays a **Choose Data Source** dialog, select **Microsoft SQL Server Database File** and click **Continue**.) For the **Data source** field, if **Microsoft SQL Server Database File (SqlClient)** is not displayed, click **Change...**, select **Microsoft SQL Server Database File (SqlClient)** and click **OK**. Next, click **Browse...** to the right of the **Database file name** field to locate and select the `Books.mdf` file in the `Databases` directory included with this chapter's examples. You can click **Test Connection** to verify that the IDE can connect to the database through SQL Server Express. Click **OK** to create the connection. Figure 22.14 shows the **Connection string** for the `Books.mdf` database. This contains the information that the ADO.NET Entity Framework requires to connect to the database at runtime. Click **Next >**. A dialog will appear asking if you'd like to add the database file to your project. Click **Yes** to move to the next step.

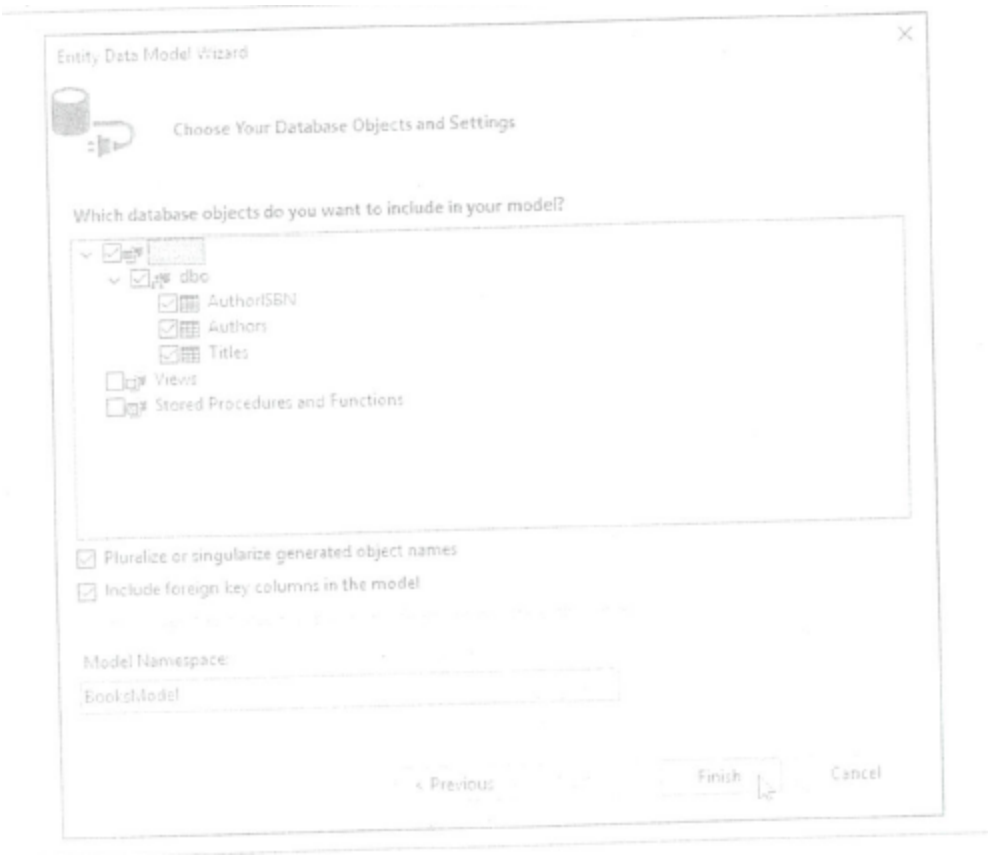


4. *Choosing the Entity Framework Version.* In the **Choose Your Version** step, select the **Entity Framework 6.x** (Fig. 22.15), then click **Next >**. This adds the latest version of the Entity Framework to your project.
5. *Choosing the Database Objects to Include in the Model.* In the **Choose Your Database Objects and Settings** step, you'll specify the parts of the database that should be used in the ADO.NET Entity Data Model. Select the **Tables** node as shown in Fig. 22.16, then click **Finish**. At this point, the IDE will download the Entity Framework 6.x templates you need and add them to your project. You may see one or more **Security Warning** dialogs—Visual Studio displays these when you attempt to use downloaded content in your projects. Click **OK** to dismiss each dialog. These warnings are intended primarily for cases in which a Visual Studio template is downloaded from an untrusted website.



Step 6: Viewing the Entity Data Model in the model designer as shown below.





Step 7:

Building the Class Library. Select **Build > Build Solution** to build the class library that you'll reuse in the next several examples—this will compile the entity data model classes that were generated by the IDE.[1] When you build the class library, the IDE generates the classes that you can use to interact with the database. These include a class for each table you selected from the database and a derived class of `DbContext` named `BooksEntities` that enables you to programmatically interact with the database—the IDE created the name `BooksEntities` (Fig. 22.14) by adding `Entities` to the database file's base name (`Books` in `Books.mdf`). Building the project causes the IDE to execute a script that creates and compiles the entity data model classes.

Exercise 02:

>> Creating a Win Form Project and confirm it to use the Entity Data Model.

Step 01: Creating a project and add it to the existing solution.

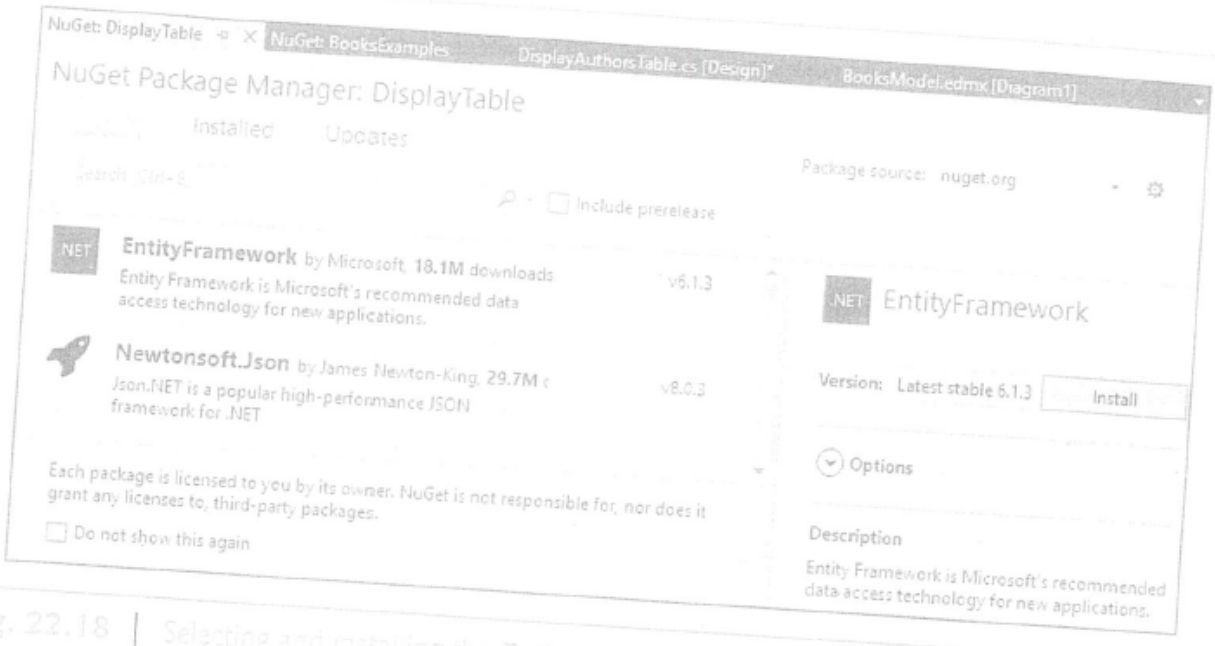
1. Right click Solution 'BooksExamples' (the solution name) in Solution Explorer and select Add > New Project... to display the Add New Project dialog.
2. Select Windows Forms Application from the Visual C# > Windows > Classic Desktop category, name the project `DisplayTable` and click OK.
3. Change the name of the `Form1.cs` source file to `DisplayAuthorsTable.cs`. The IDE updates the `Form`'s class name to match the source file. Set the `Form`'s Text property to `Display Authors Table`.
4. Right click the `DisplayTable` project's name in the Solution Explorer, then select Set as Startup Project to configure the solution so that project `DisplayTable` will execute when you select Debug > Start Debugging (or press *F5*).

Step 02: Adding a reference of the **BooksExamples** class library.

1. Right click the `DisplayTable` project's References node in the Solution Explorer and select Add Reference....
2. In the left column of the Reference Manager dialog that appears, select Projects to display the other projects in this solution, then in center of the dialog ensure that the checkbox next to `BooksExamples` is checked and click OK. `BooksExamples` should now appear in the projects References node.

Step 03: Adding a reference to Entity Framework

1. Right click the project's name in the Solution Explorer and select Manage NuGet Packages... to display the NuGet tab in Visual Studio's editors area. NuGet is a tool (known as a *package manager*) that that helps you download and manage libraries (known as *packages*) used by your projects.
2. In the dialog that appears, click Browse, then select the EntityFramework by Microsoft and click Install (Fig. 22.18).
3. The IDE will ask you to review the changes. Click OK.
4. The IDE will ask you to accept the `EntityFramework` license. Click I Accept to complete the installation.



g. 22.18 | Selecting and installing the EntityFramework in the NuGet GUI.

Step 4: Adding the Connection String to the Windows Forms App

Each app that will use the entity data model also requires the *connection string* that tells the Entity Framework how to connect to the database. The connection string is stored in the `BooksExamples` class library's `App.Config` file. In the **Solution Explorer**, open the `Books-Examples` class library's `App.Config` file, then copy the `connectionStrings` element (lines 7–9 in our file), which has the format:

```
<connectionStrings>
   Connection string information appears here
</connectionStrings>
```

Next, open the `App.Config` file in the `DisplayTable` project and paste the connection string information *after* the line containing `</entityFramework>` and *before* the line containing `</configuration>`. Save, then close the `App.Config` file.

22.5.3 Data Binding

>> Databinding between the Controls and Entity Data Model

Step 01: Adding a data source for the Authors Table

To use the entity data model classes for the data binding, you must first add them as a data source. To do so:

#1. Select View > Other Windows > Data Sources to display the **Data Sources** window at the left side of the IDE, then in that Window click the **Add New Data Source…** link to display the **Data Sources Configuration Wizard**

#2. The Entity Data Model classes are used to create objects representing the tables in the Database, so we will use an **Object** data source. And follow the dialog windows.

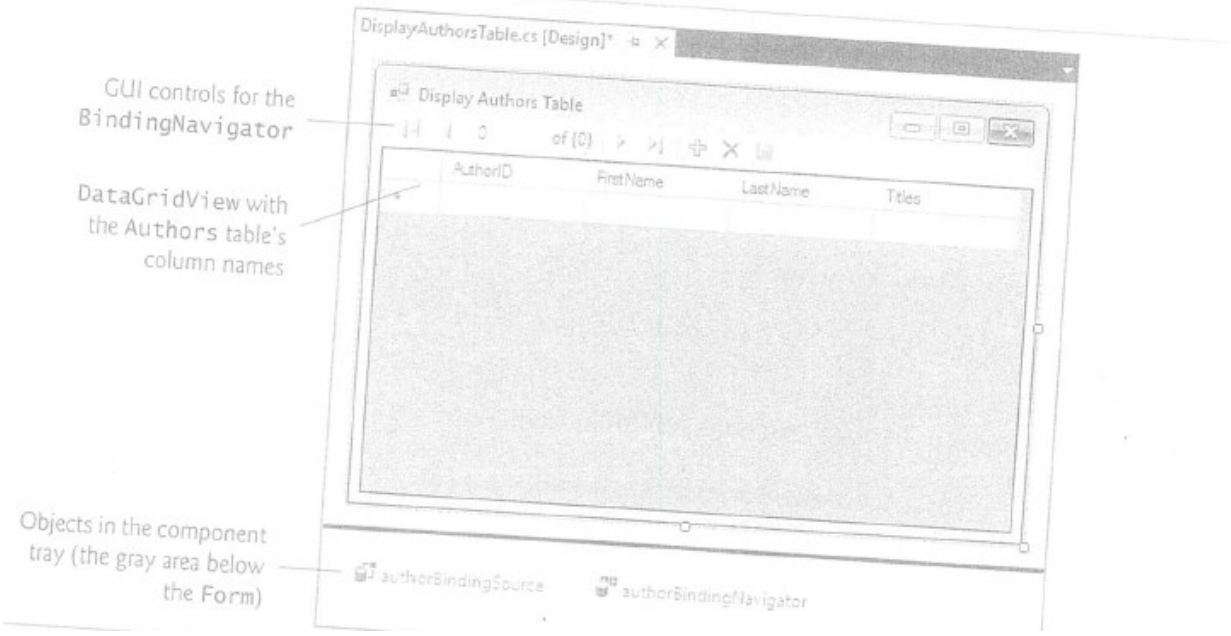
Step 02: Creating GUI elements.

Step 2: Creating GUI Elements

Next, you'll use the **Design** view to create a `DataGridView` control that can display the `Authors` table's data. To do so:

1. Switch to **Design** view for the `DisplayAuthorsTable` class.
2. Click the **Author** node in the **Data Sources** window—it should change to a drop-down list. Open the drop-down by clicking the down arrow and ensure that the `DataGridView` option is selected—this is the GUI control that will be used to display and interact with the data.
3. Drag the **Author** node from the **Data Sources** window onto the `Form` in **Design** view. You'll need to resize the `Form` to fit the `DataGridView`.

The IDE creates a `DataGridView` (Fig. 22.21) with column names representing all the properties for an `Author`, including the `Titles` navigation property.



Step 3: Connecting the Data Source to the **`authorBindingSource`**

The final step is to connect the data source to the `authorBindingSource`, so that the app can interact with the database. Figure 22.22 shows the code needed to obtain data from the database and to save any changes that the user makes to the data back into the database.

```
// Displaying data from a database table in a DataGridView.
using System;
using System.Data.Entity;
using System.Data.Entity.Validation;
using System.Linq;
using System.Windows.Forms;

namespace DisplayTable
{
   public partial class DisplayAuthorsTable : Form
   {
      // constructor
      public DisplayAuthorsTable()
      {
         InitializeComponent();
      }

      // Entity Framework DbContext
      private BooksExamples.BooksEntities dbcontext =
         new BooksExamples.BooksEntities();

      // load data from database into DataGridView
      private void DisplayAuthorsTable_Load(object sender, EventArgs e)
      {
         // load Authors table ordered by LastName then FirstName
         dbcontext.Authors
            .OrderBy(author => author.LastName)
            .ThenBy(author => author.FirstName)
            .Load();

         // specify DataSource for authorBindingSource
         authorBindingSource.DataSource = dbcontext.Authors.Local;
      }

      // click event handler for the Save Button in the
      // BindingNavigator saves the changes made to the data
      private void authorBindingNavigatorSaveItem_Click(
         object sender, EventArgs e)
      {
         Validate(); // validate the input fields
         authorBindingSource.EndEdit(); // complete current edit, if any
```

```
43
44          // try to save changes
45          try
46          {
47              dbcontext.SaveChanges(); // write changes to database file
48          }
49          catch(DbEntityValidationException)
50          {
51              MessageBox.Show(
52
53          }                                   );
54      }
55  }
56  }
```