

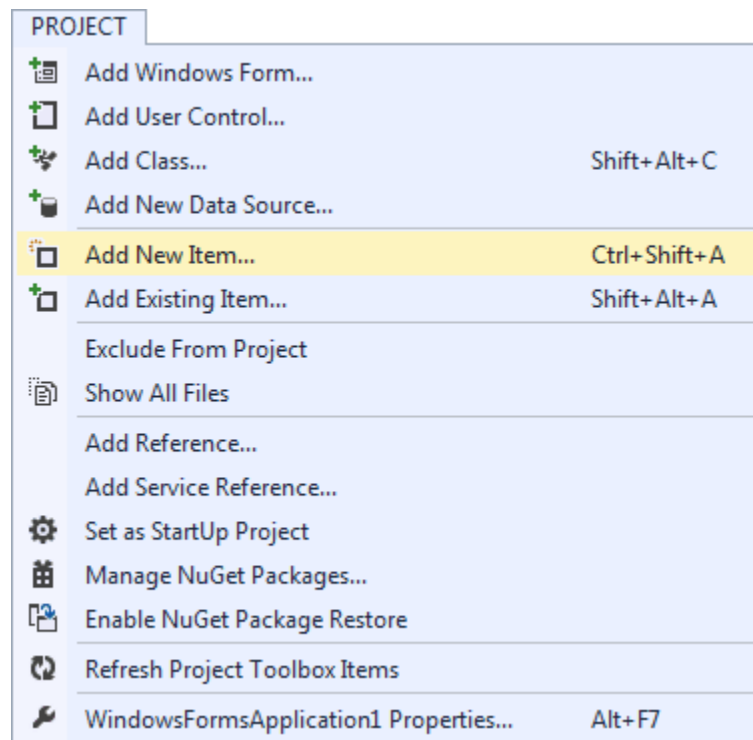
Sql Server Express And C# .Net

In this section, you'll learn how to create a database with SQL Server Express. You can do all this within the Visual C# .NET software. Once you have a database, you'll learn how to pull records from it, and display them on a Windows Form. You will also learn how to navigate through the records in your database, and how to add new records.

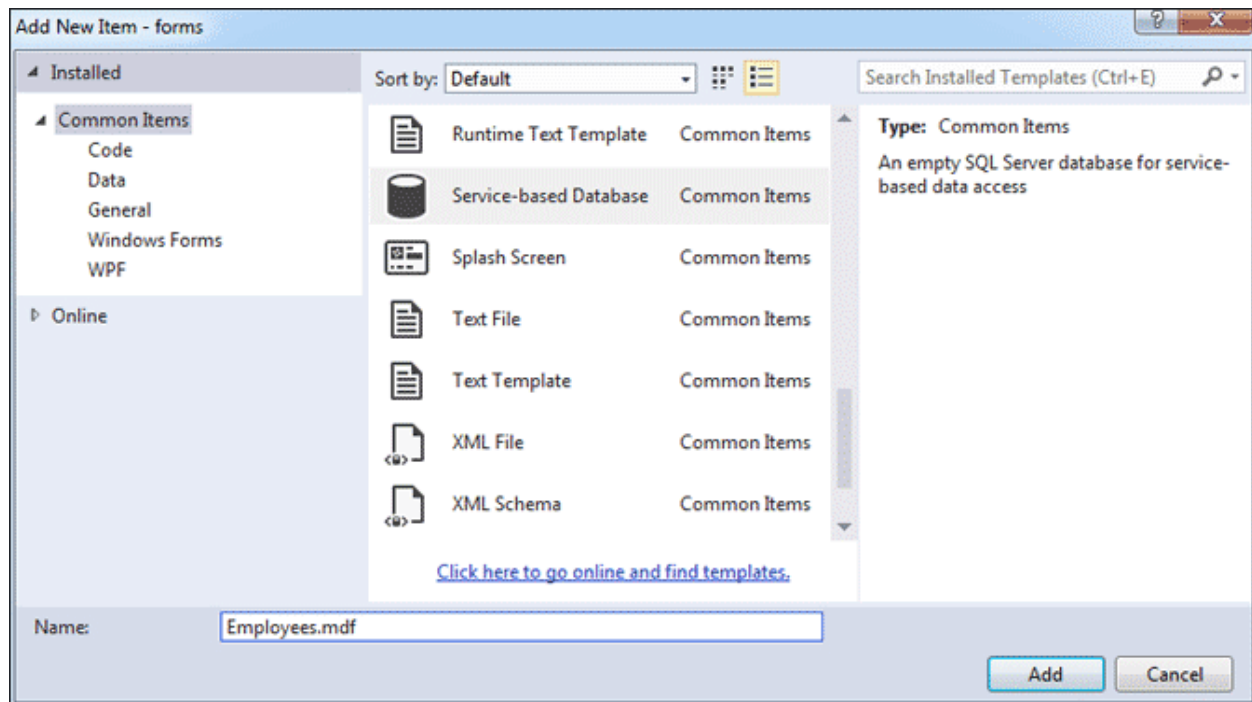
How to Create a SQL Server Express Database

Start a new project, and call it anything you like as we're only using this project to create a database. But do remember where on your hard drive you saved this project to, usually in your Documents folders, under Visual Studio, then Projects. When a database is created, it will be saved in the project folder. You'll need to browse to this folder to add the database as resource when we create a new project.

From the menu at the top, click on Project > Add New Item.



From the **Add New Item** dialogue box, look for the **Service-based database** item, as in the image below:

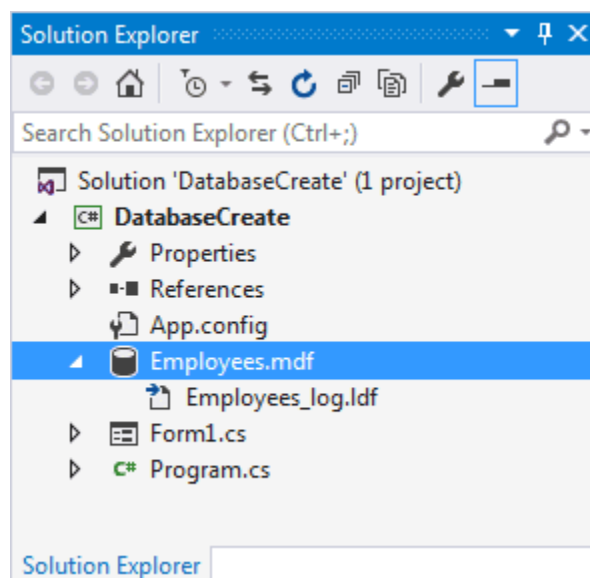


In the Name box at the bottom, type Employees.mdf. Then click the **Add** button.

If you're prompted to Install Missing Packages, download the extra files.

If you have Visual Studio Community 2015 to 2019, you'll go straight back to your Form, and it will look like nothing has happened.

In all versions, it might look as though nothing has happened, when you are returned to your form. But have a look at the Solution Explorer at the top right of your screen. You should see your database there:

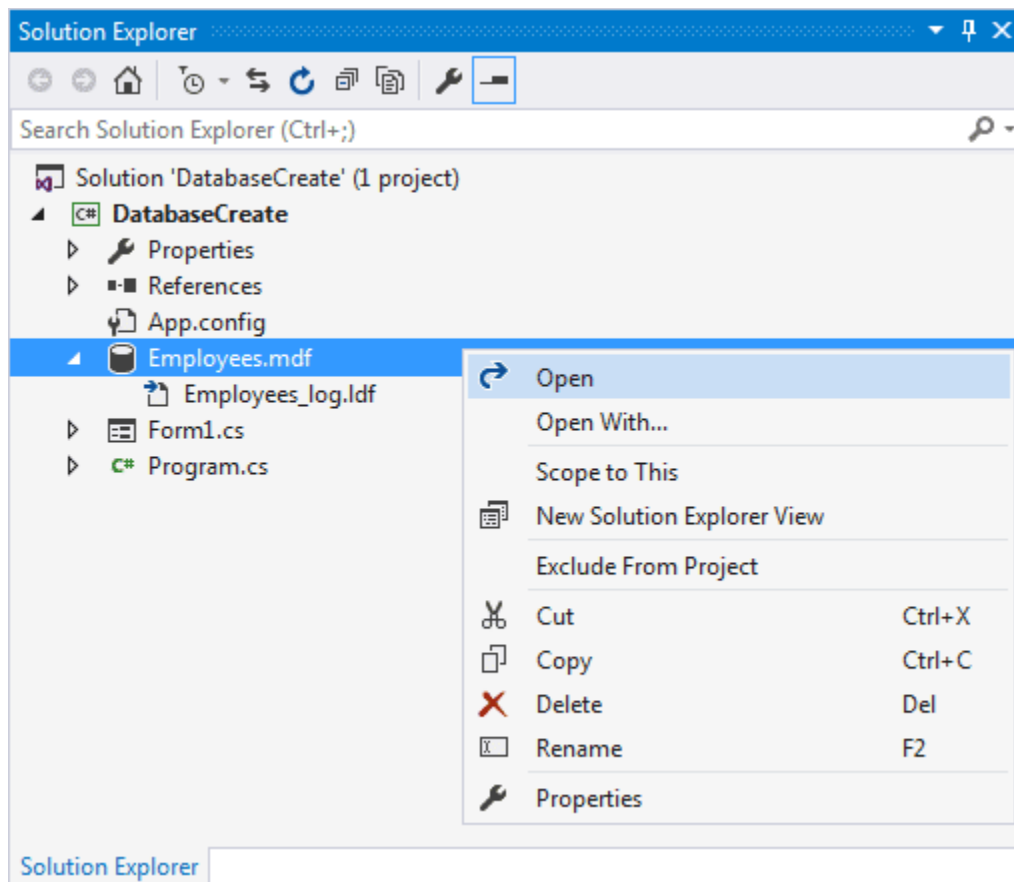


When you save your project, the database will get saved along with all the solution files.

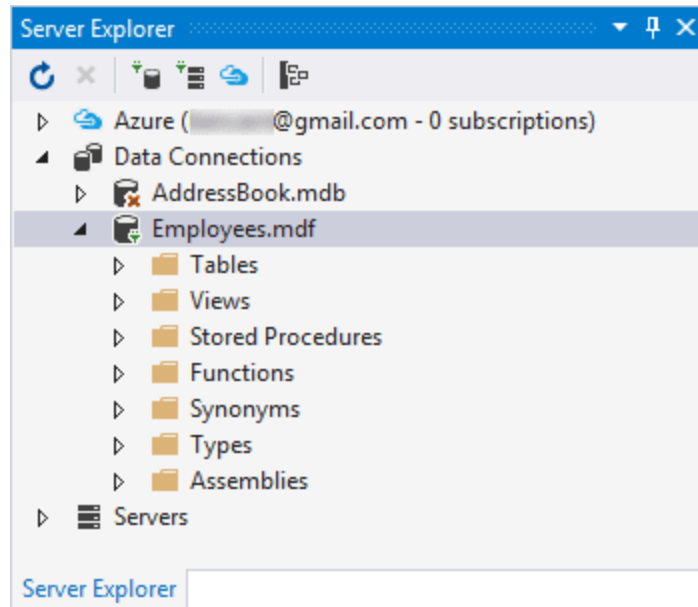
How To Create Tables In Your Sql Server Database

Now that you have created the database itself, you need to create at least one table to go in it.

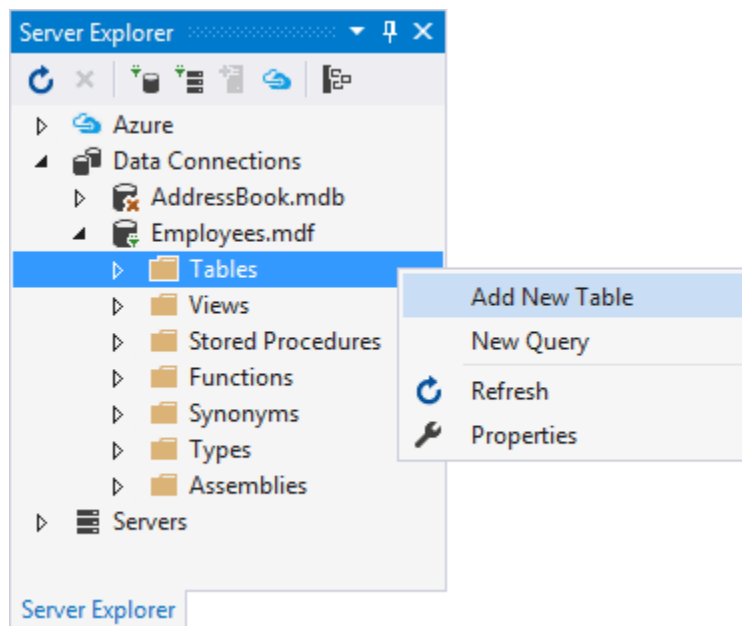
To create a table, right click on the database in the Solution Explorer. From the menu that appears, select **Open**:



When you click Open, you'll see the Server Explorer appear on the left-hand side of your screen:



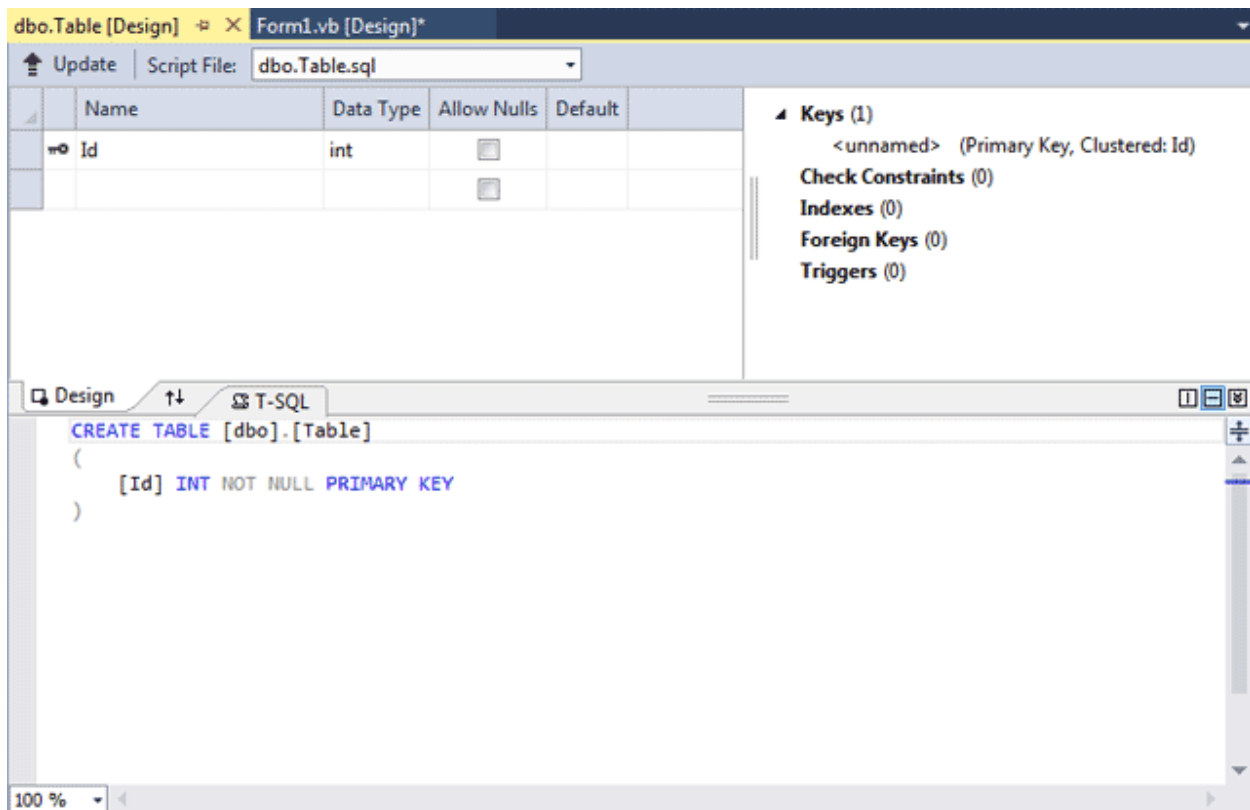
To create a new table, right click on **Tables**. From the menu that appears, select **Add New Table**:



When you click on Add New Table you should see the table designer open up in the centre of your screen. We'll now configure this new database table using the designer. To see how to do that click below, right.

Create A Database Table In Sql Server Express

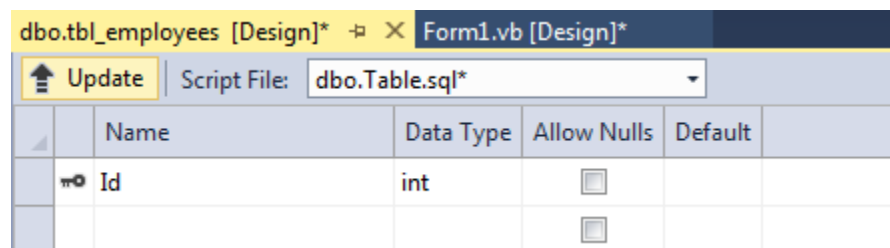
When you click Add New Table, as you did in [the previous section](#), you'll see this screen appear in the middle:



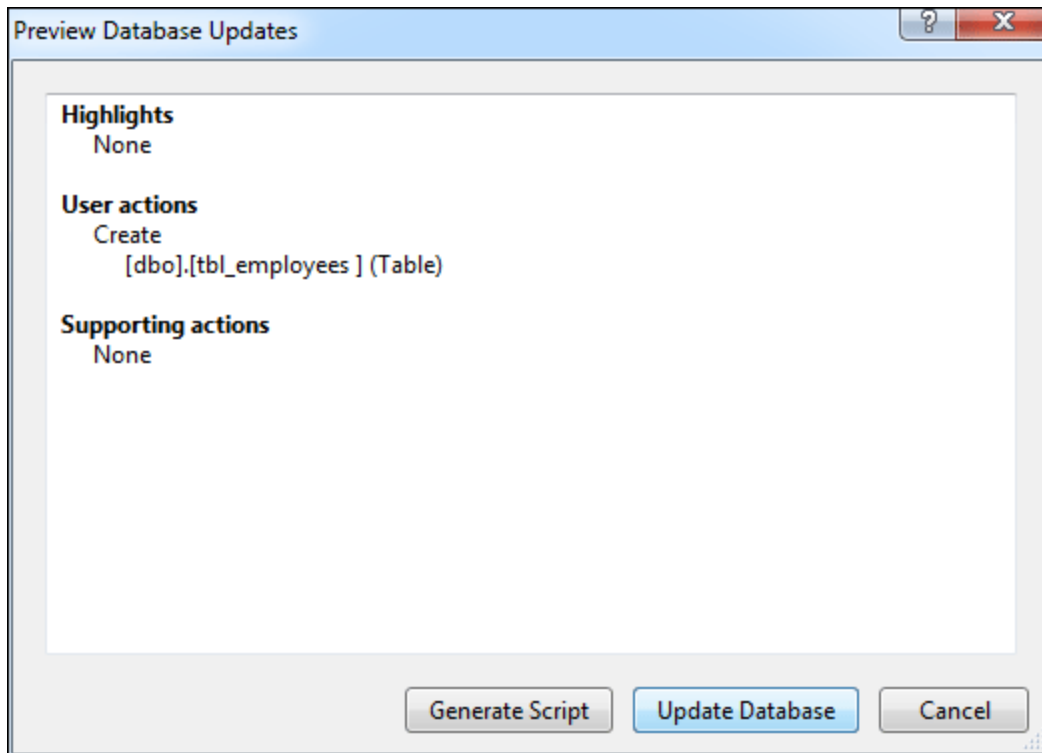
The first thing we can do is to give the table a name. In the bottom half of the screen above, delete the word Table between the square brackets. Type the name tbl_employees instead:

CREATE TABLE [dbo].[tbl_employees]

Now click the **Update** button, which is top left of the table designer:




You should then see the following screen:



Click **Update Database** to return to the table designer. We can now set up the columns that are going into the table.

The first column name, **Id**, has already been set up:

dbo.tbl_employees [Design] - X					
Update		Script File: dbo.Table.sql			
	Name	Data Type	Allow Nulls	Default	
	Id	int	<input type="checkbox"/>		
			<input type="checkbox"/>		

The Data Type is OK on **int**, which is short for Integer. The column has a key symbol to the left, which means it is the Primary Key. This is OK, too. The Allow Nulls is unchecked, which is what you want for a Primary Key. This means you can't have duplicate item for this column.

One thing we can do for the Id column is to have it update itself automatically. When you add a new item to the database, the next integer in the sequence will be added to the Id column. To set the Id column to Auto Increment, highlight the Id row then take a look at the properties area bottom right. Expand the **Identity Specification** item:

Properties

Id Column

Collation

Computed Column S

Data Type **int**

Default Value or Bindi

Description

Full Text Specification False

Identity Specification False

(Is Identity) **False**

Identity Increment

Identity Seed

Is Column Set False

Is File Stream False

Is ROWGUID Column False

Identity Specification
Expands to show properties specific to identity columns.

Now set **Is Identity** to **True**:

Properties

Id Column

Collation

Computed Column S

Data Type **int**

Default Value or Bindi

Description

Full Text Specification False

Identity Specification True

(Is Identity) **True**

Identity Increment **1**

Identity Seed **1**

Is Column Set False

Is File Stream False

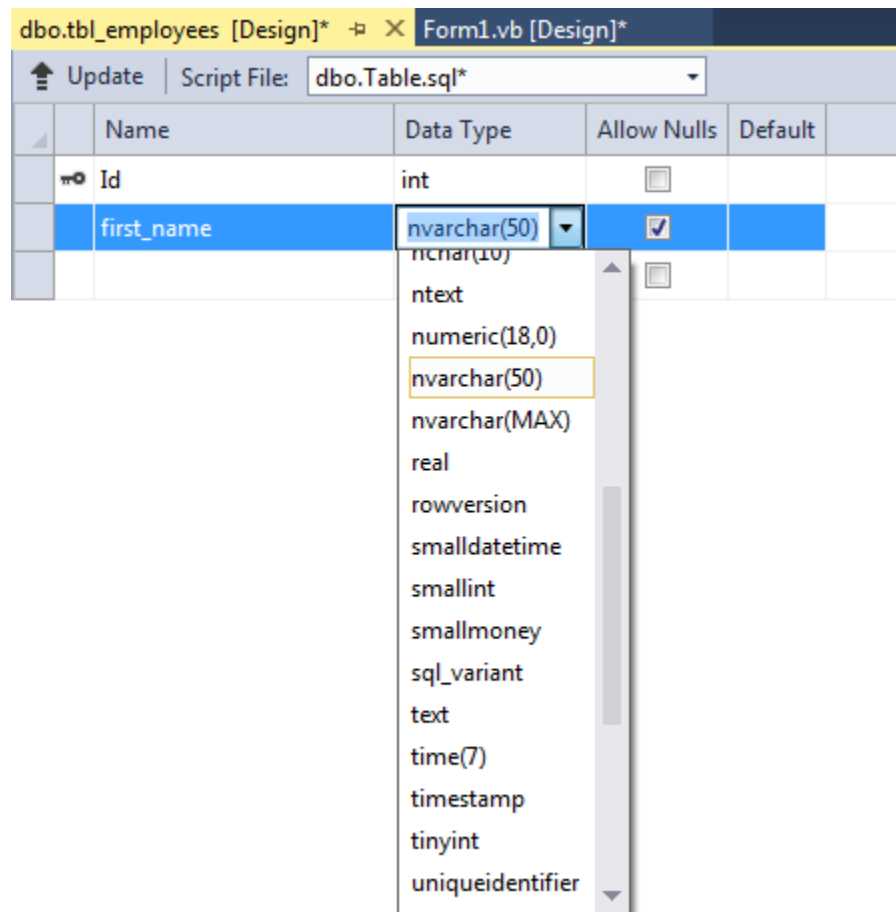
Is ROWGUID Column False

(Is Identity)
Specifies whether the column is the identity column for the table.

The Identity Increment has a default of 1, meaning 1 will get added to the Id column every time a new entry is added to the table.

With the Id column set up, we can add more columns.

Click in the **Name** box just below Id at the top of your table designer. Now type the new column heading **first_name**. We want this to be text. So for the **Data Type**, select **nvarchar(50)**, meaning a maximum of 50 characters:



The **Allow Nulls** is ok checked.

Add a third column by clicking into the Name box again. This time, type **last_name**. Set the Data Type to **nvarchar(50)**, just as before. Leave Allow Nulls checked.

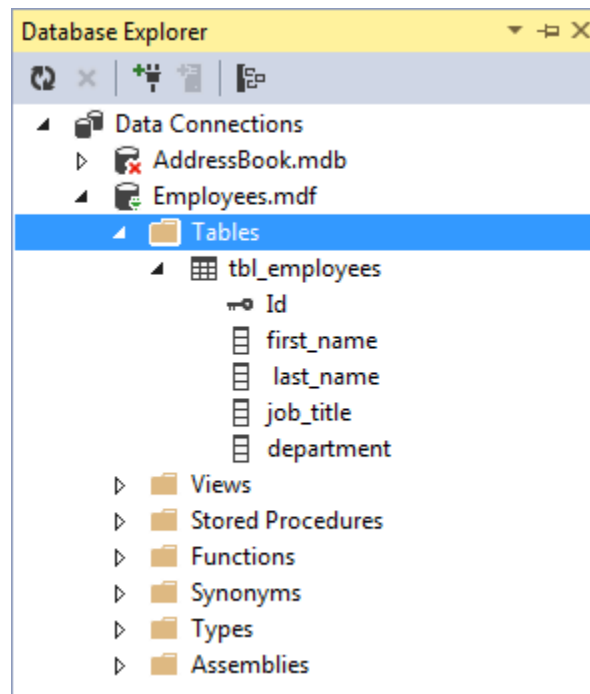
We only need two more columns, **job_title** and **department**. Add this using the same technique as above. When you're done, your table designer will look like this:

dbo.tbl_employees [Design]*

Update | Script File: `dbo.Table.sql*`

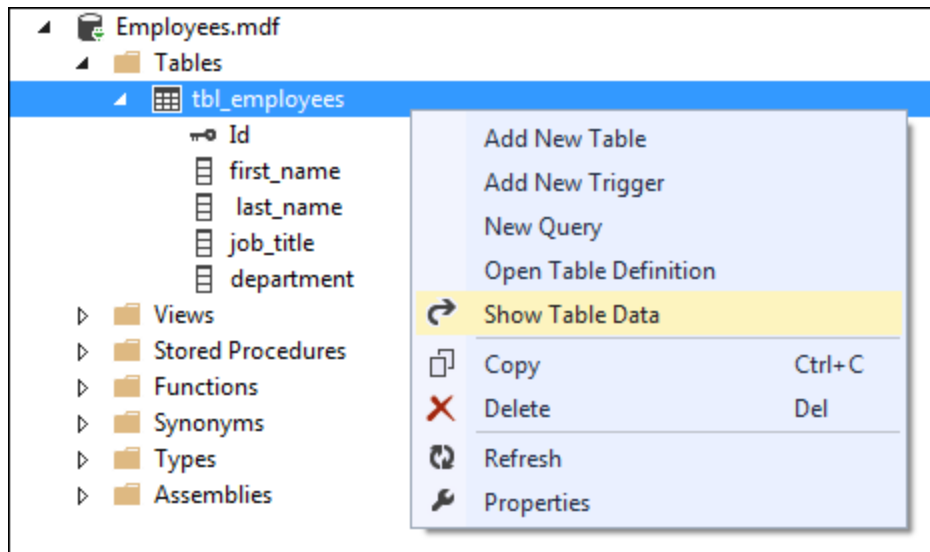
	Name	Data Type	Allow Nulls	Default	
	Id	int	<input type="checkbox"/>		
	first_name	nvarchar(50)	<input checked="" type="checkbox"/>		
	last_name	nvarchar(50)	<input checked="" type="checkbox"/>		
	job_title	nvarchar(50)	<input checked="" type="checkbox"/>		
	department	nvarchar(50)	<input checked="" type="checkbox"/>		
			<input type="checkbox"/>		

Again, click the Update button to save your changes. Now have a look at the Database Explorer on the left. You should find that your new columns are displayed:



Now that we have all the columns set up, we can add some data to the table.

To add data to your table, right click the name of your table in the Database Explorer. Then click **Show Table Data**:



When you click on **Show Table Data** you'll see a new screen appear in the middle. This one:

dbo.tbl_employees_[Data] [X]					
Max Rows: 1000					
	Id	first_name	last_name	job_title	department
*	NULL	NULL	NULL	NULL	NULL

The columns are the ones we set up earlier. Each row will be a single entry in the table.

Because we set the Id column to Auto Increment, it means we don't have to type anything into this box. So click into the text box under **first_name**. Enter **Adara**. Now press the Tab key on your keyboard. You'll be taken to the next text box to the right, the **last_name** field. Enter **Hussein** as the last name. Press the Tab key again to go to the **job_title** text box. Enter **Lead Programmer** here. Tab across again to the **department** text box and enter **IT**. Your screen will then look like this:

dbo.tbl_employees_[Data] [X]					
Max Rows: 1000					
	Id	first_name	last_name	job_title	department
	NULL	Adara	Hussein	Lead Program..	IT
*	NULL	NULL	NULL	NULL	NULL

As you can see, there are red warning circles on the previous three entries. This is because the data hasn't been committed to the table. To get rid of the warning circles simply tab to the next row down. Tab to the first_name field again, on row two this time. (You can also just click inside a text box.)

Now enter the following data in your table:

first_name	last_name	job_title	department
Hamal	Ata	Network Engineer	IT
Harris	Hameed	Systems Analyst	IT
Tansy	Lakshman	Writing Assistant	Creative
Orenda	Khan	Head of Tuition	Teaching
Tadi	Patel	Network Engineer	IT
Zoe	Walker	Head of Design	Creative
Alice	Thyne	Tutor	Teaching
Jake	Jaloore	Graphic Artist	Creative

When you're finished, your table data screen should look like this:

dbo.tbl_employees_ [Data] X					
Max Rows: 1000					
	Id	first_name	last_name	job_title	department
	1	Adara	Hussein	Lead Programmer	IT
	2	Hamal	Ata	Network Engineer	IT
	3	Harris	Hameed	Systems Analyst	IT
	4	Tansy	Lakshman	Writing Assistant	Creative
	5	Orenda	Khan	Head of Tuition	Teaching
	6	Tadi	Patel	Network Engineer	IT
	7	Zoe	Walker	Head of Design	Creative
	8	Alice	Thyne	Tutor	Teaching
	9	Jake	Jaloore	Graphic Artist	Creative
▶*	NULL	NULL	NULL	NULL	NULL

Now that you have a database with some data in it, we can move on. Before you close this solution, remember where you saved it to, and the name of your project. If you left everything on the defaults when you installed Visual Studio, then your projects will be in your Documents folder. The database will be in the folder created for this database project.

Save your work, and you will have created your very first Compact SQL Server Express database! But it's a huge subject, and whole books have been written about SQL Server. We can only touch on the very basics here. What we do have, though, is a database we can open with C# .NET programming code. We'll do that next.

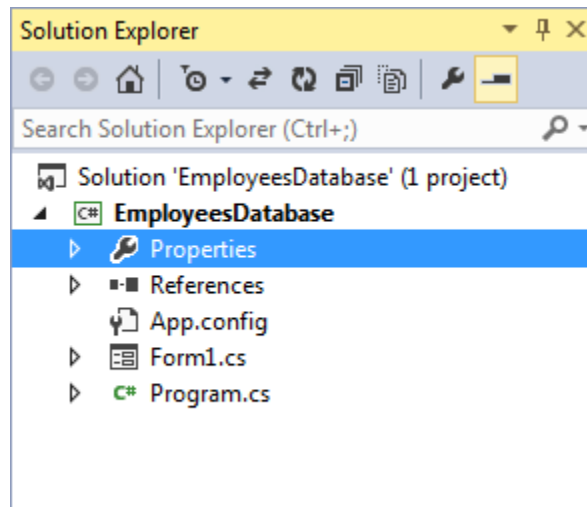
Creating A Database Project With C# .Net

What we're going to do now is to use the Employees database we created in the previous section. We'll add it as a resource to a new project. We can then create a form with buttons that allow us to scroll back and forward through the records in the database.

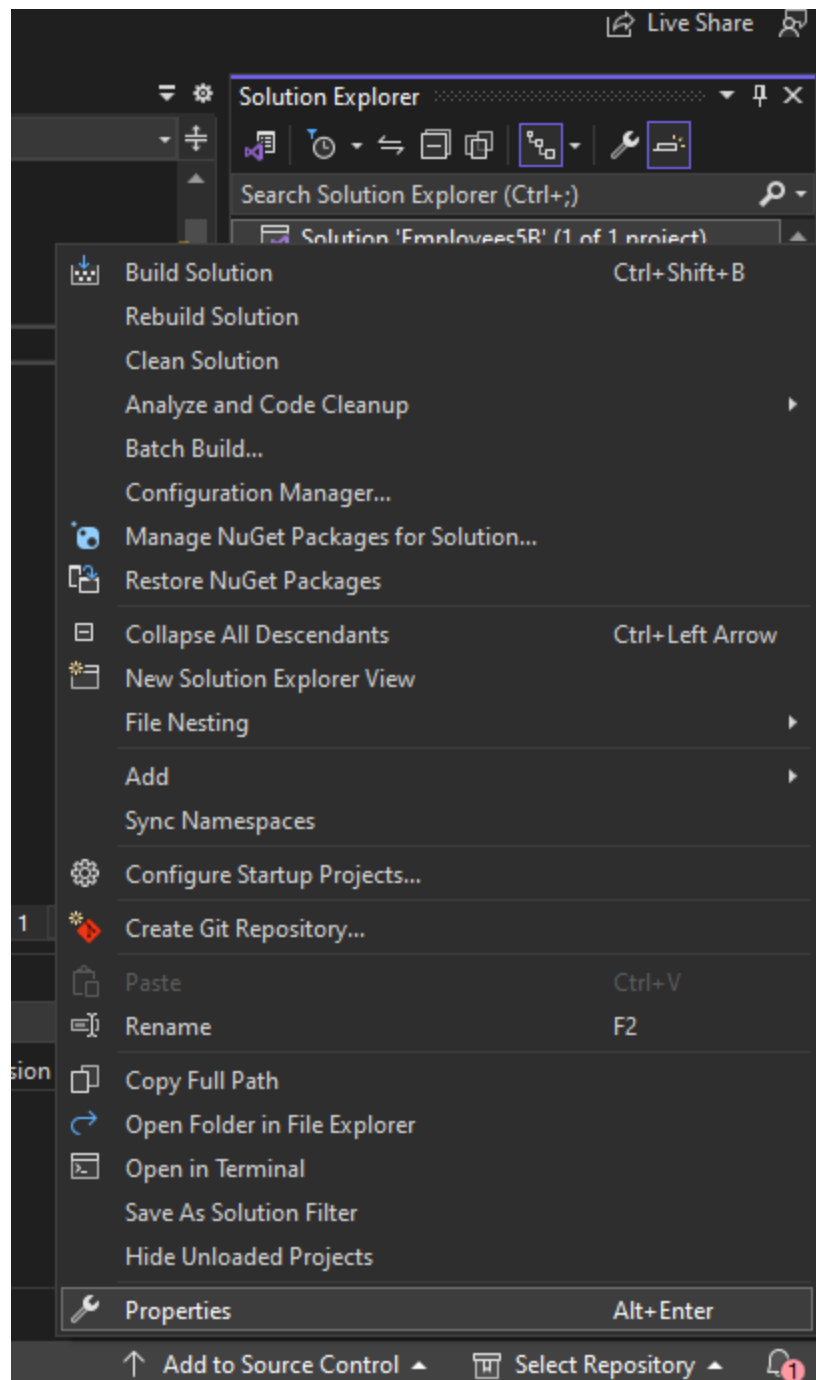
Adding a Database to a Project

Create a new project. Call it **EmployeesDatabase**. To add a database to your new project, have a look at the Solution Explorer on the right.

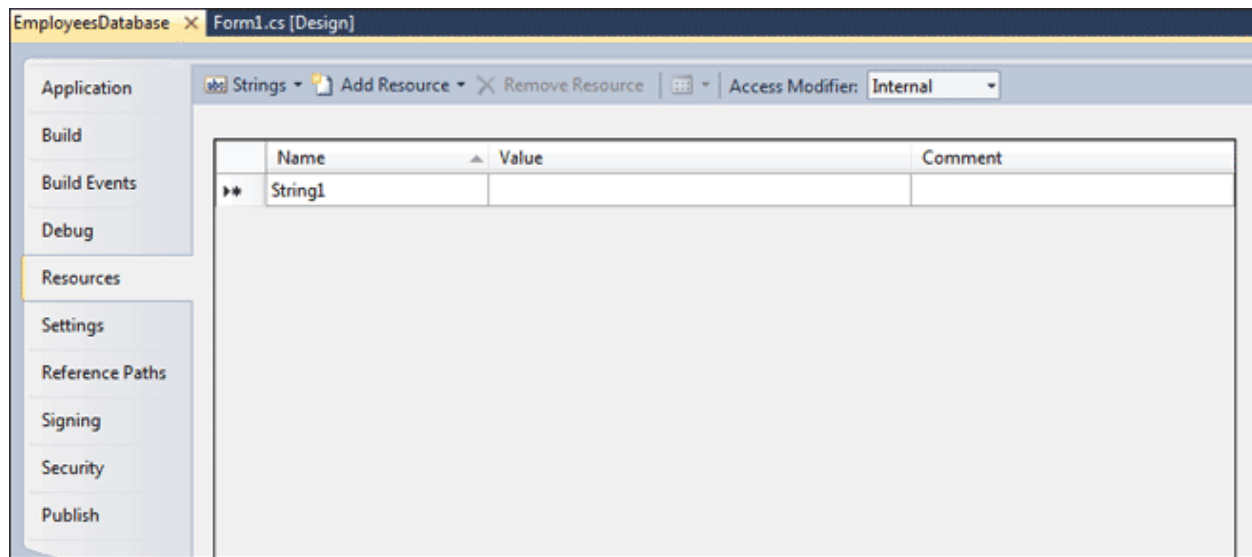
Locate the Properties item:



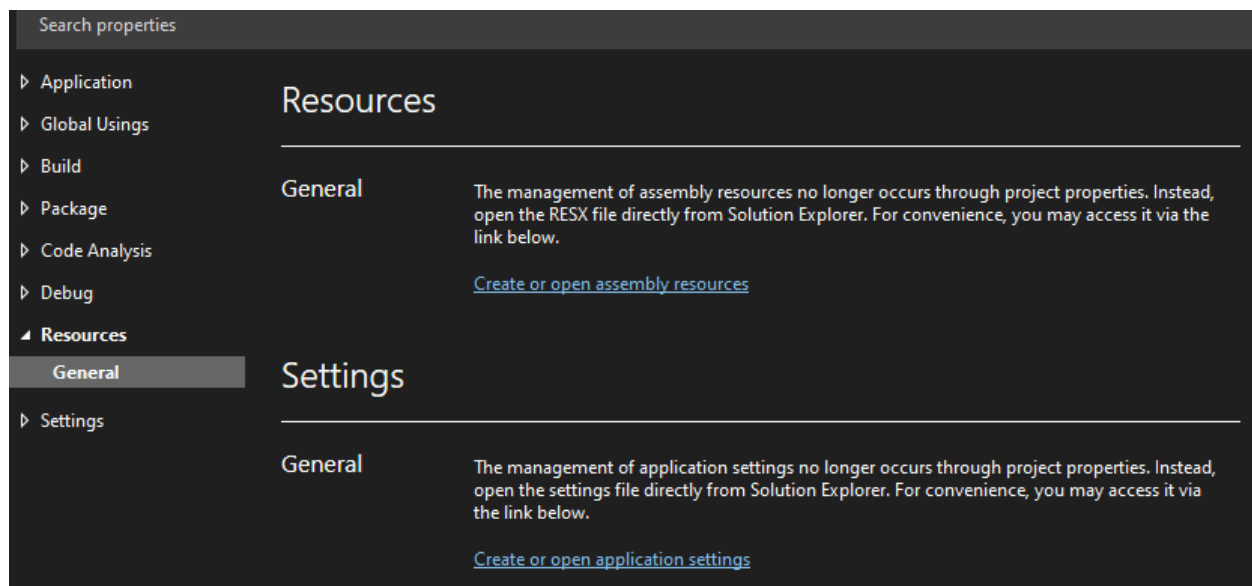
If you are using a newer version of the Visual studio, You might not see the properties item under the Solution Explorer. So, right click on the Project



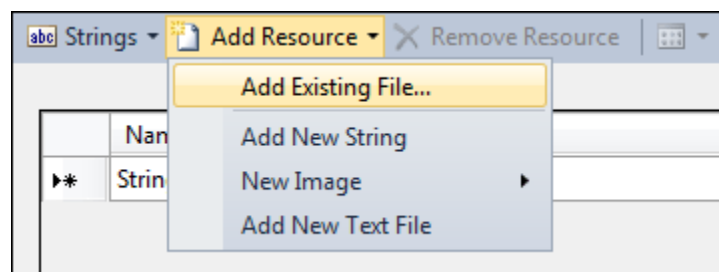
Double click on properties to see a new screen appear. Click on the Resources tab on the left of the new screen:



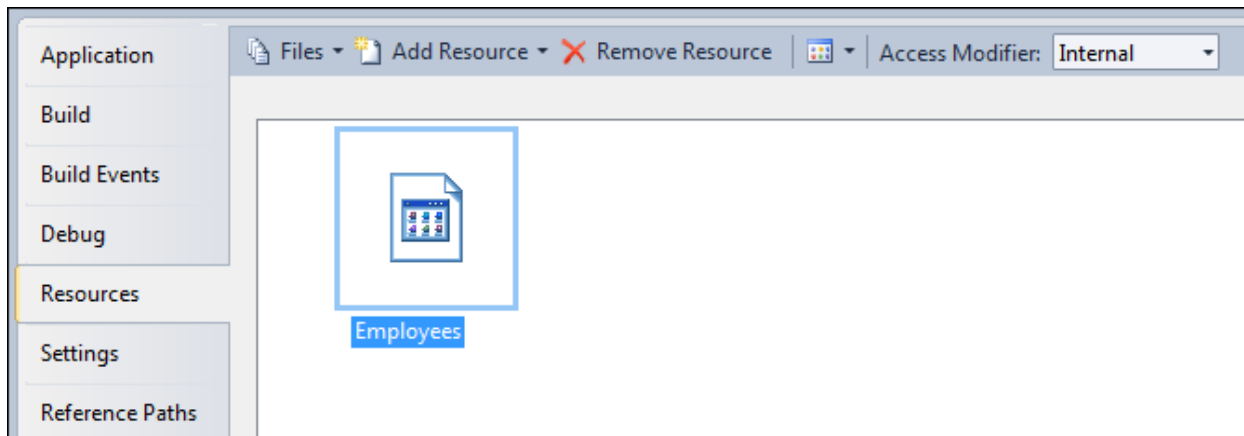
Or in Newer version, Click on the resources on the left and then Click on “Create or open assembly resources”



Click on the **Add Resource** dropdown list at the top and select **Add Existing File**:

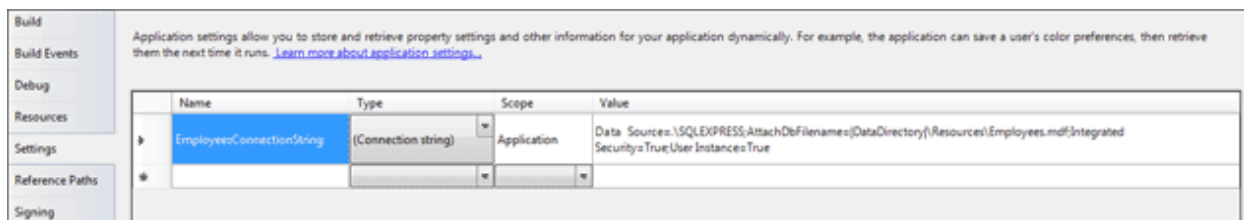


When you click on Add Existing File, you'll see a standard Open File dialogue box appear. Navigate to where you saved your **Employees.mdf** database. (If you didn't create a database, navigate to the one you downloaded from our extra files, here: [extra files](#).) Click **Open** on Open File dialogue box. You will then see this:



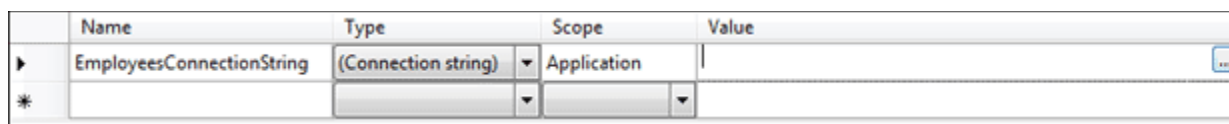
The database has been added to the project. You can see that it has been added by looking at the Solution Explorer on the right.

Have a look at the Properties screen again. Now click on the **Settings** tab, just below **Resources**. If you have version 2012 of Visual Studio Express, you'll see this:

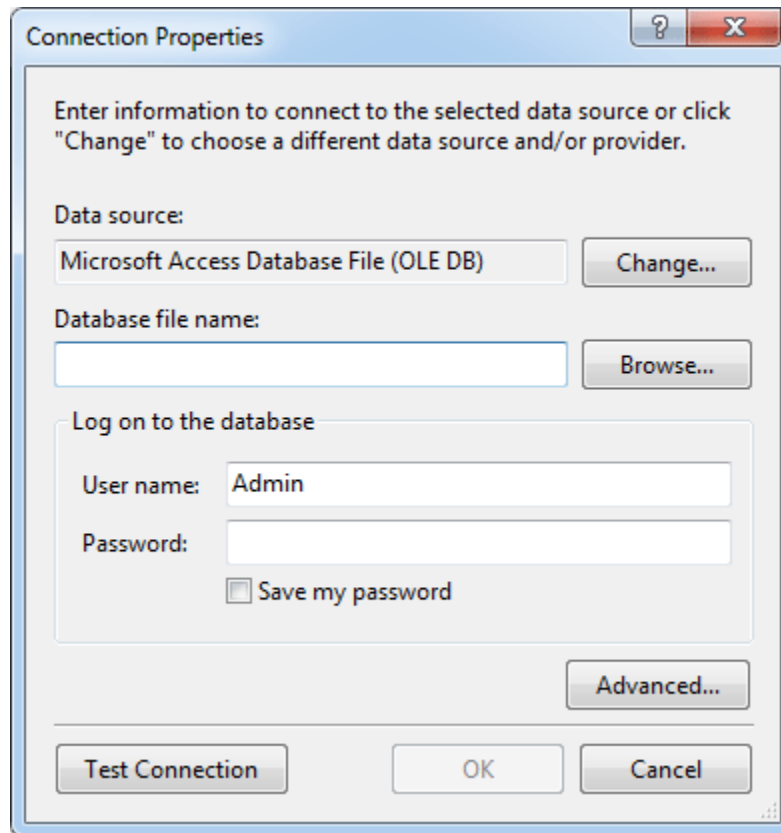


A Name, a Type, a Scope and a Value have been filled in for you. The Value is the connection string needed when we connect to the database. The Name will show up in the IntelliSense list a little later. For VS 2012 users, you can now scroll down this page a bit, until you come to the section All Visual Studio Users.

However, 2013 and 2015 users won't see any entries on the Settings page. You'll have to fill them out for yourself. To do that, click into the Name box and type **EmployeesConnectionString**. From the Type dropdown list select **Connection String**. Change the Scope to **Application**. For the Value, click inside the long text box. You'll see a button to the right:

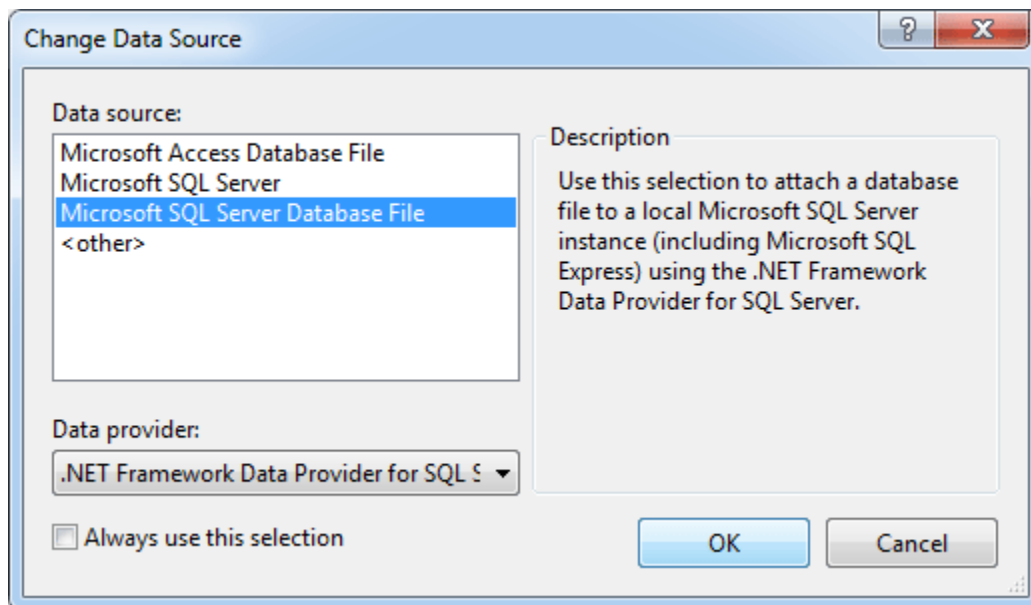


Click the button to see this dialogue box:



The 'Connection Properties' dialog box is shown. It has a title bar with a question mark and a close button. The main text says: 'Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.' Below this, there are two sections. The first section is 'Data source:' with a text box containing 'Microsoft Access Database File (OLE DB)' and a 'Change...' button to its right. The second section is 'Database file name:' with an empty text box and a 'Browse...' button to its right. Below these is a section titled 'Log on to the database' which contains 'User name:' with a text box containing 'Admin', 'Password:' with an empty text box, and a checkbox labeled 'Save my password' which is currently unchecked. At the bottom right of the dialog is an 'Advanced...' button. At the very bottom are three buttons: 'Test Connection', 'OK', and 'Cancel'.

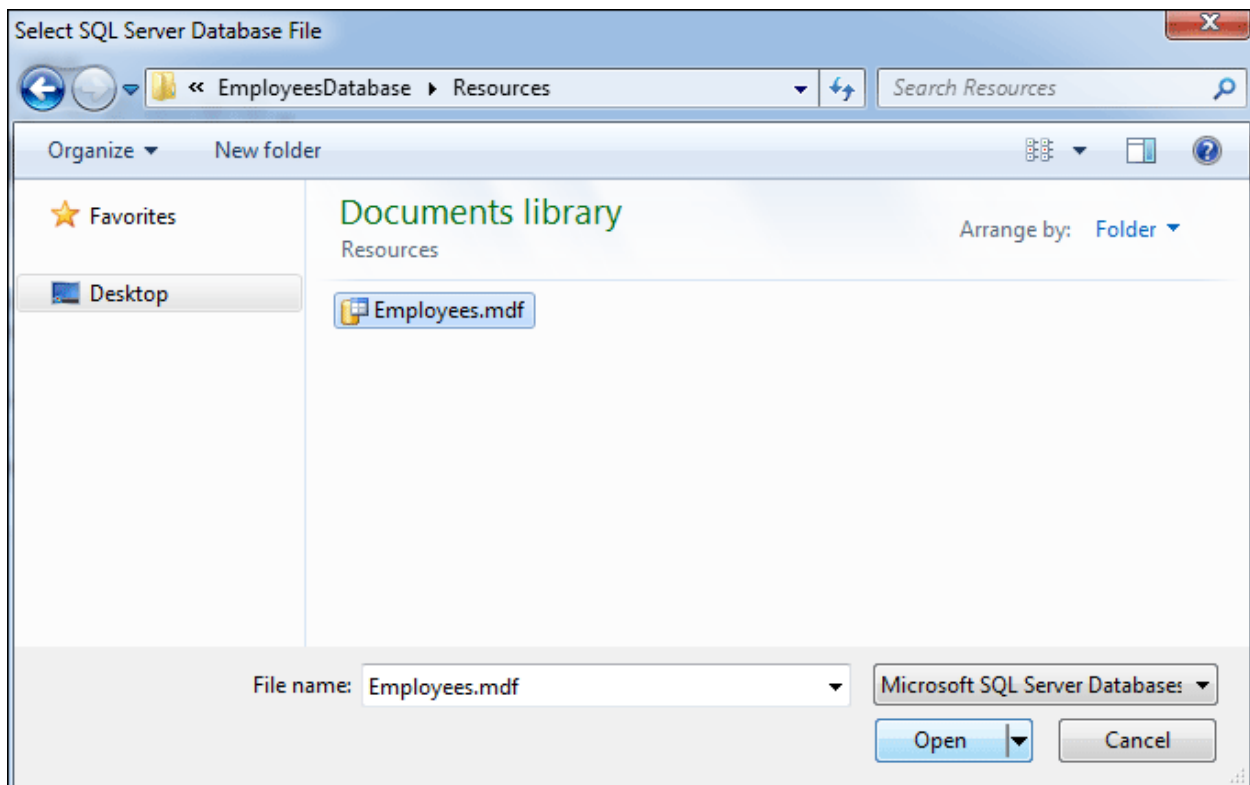
The default Data Source is for an Access database, so we need to change this. Click the Change button to see a new dialogue box:



The 'Change Data Source' dialog box is shown. It has a title bar with a question mark and a close button. The main area is divided into two panes. The left pane is titled 'Data source:' and contains a list box with four items: 'Microsoft Access Database File', 'Microsoft SQL Server', 'Microsoft SQL Server Database File' (which is highlighted with a blue background), and '<other>'. The right pane is titled 'Description' and contains the text: 'Use this selection to attach a database file to a local Microsoft SQL Server instance (including Microsoft SQL Express) using the .NET Framework Data Provider for SQL Server.' Below the list box is a section titled 'Data provider:' with a dropdown menu showing '.NET Framework Data Provider for SQL S'. At the bottom left is a checkbox labeled 'Always use this selection' which is currently unchecked. At the bottom right are two buttons: 'OK' and 'Cancel'.

Select the option **Microsoft SQL Server Database File**, and then click OK to go back to the previous screen.

You now need to browse for your MDF database. So click the Browse button to see an Open File dialogue box. Navigate to your project folder (the **EmployeesDatabase** project that you have open). Double click your Resources folder to see your database:



Click on Open and you'll get back to the Connection Properties dialogue box:

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:

Microsoft SQL Server Database File (SqlClient) Change...

Database file name (new or existing):

EmployeesDatabase\Resources\Employees.mdf Browse...

Log on to the server

☒ Use Windows Authentication

☐ Use SQL Server Authentication

User name:

Password:

☐ Save my password

Test Connection OK Cancel

Click Test Connection and it should report success. In Visual Studio Community 2015, you may get an error when you click Test Connection. If you do, click the **Advanced** button at the bottom of the Connection Properties dialogue box:

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server Database File (SqlClient) Change...

Database file name (new or existing):
 Browse...

Log on to the server

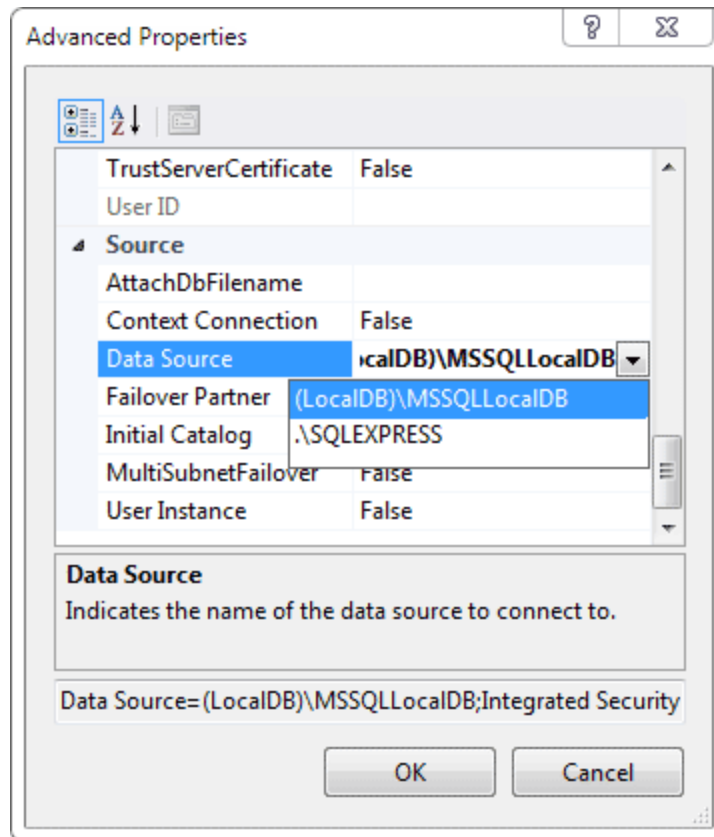
☒ Use Windows Authentication
☐ Use SQL Server Authentication

User name:
Password:
☐ Save my password

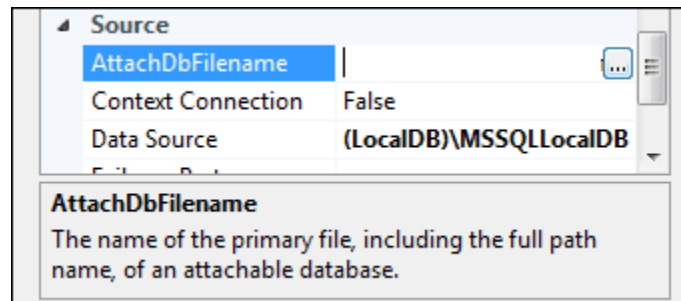
Advanced...

Test Connection OK Cancel

From the Advanced Properties box, select **LocalDb** from the Data Source drop down box:



Then click **AttachDbFilename** in the area just above **Data Source**. Click the button on the right of the text area:



Navigate to where your database is saved, which is in the Resources folder of the project you have open. Click the **Open** button when you have located your database. Click OK on your Advanced Properties dialogue box. Your Database file name area will then read something like this:

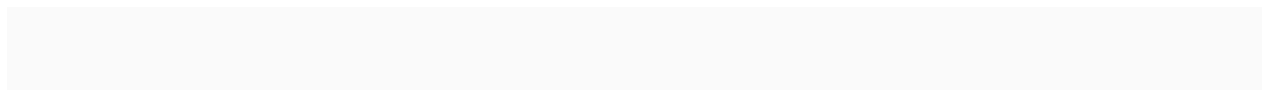
C:\Users\Ken\Documents\Visual Studio
2019\Projects\employees_database\employees_database\Resources\Employees.mdf

(Instead of Ken you'll have the name of your own computer. And instead of employees_database you'll have the name of your Visual Studio project.)

Click Test Connection and hopefully you'll see a message saying Test Connection Succeeded. Click OK to return to your Settings page. The Value area when look something like this (the year will be different, depending on which version of Visual Studio you have):

```
Data Source=(LocalDB)
\MSSQLLocalDB;AttachDbFilename="C:\Users\Ken\Documents\Visual Studio 2015
\Projects\employees_database\employees_database\Resources\Employees.mdf";Integrated
Security=True;Connect Timeout=30
```

This is a connection string we can use to connect to the Employees.mdf database. Because we've added it as setting, we don't need to type out the full string above.



All Visual Studio users

To gain access to the table in the database, you need something called a SQL String. This takes the form `SELECT * FROM tbl_employees`. The `*` symbol means "All records". There's quite a lot you can do with SQL as a language. For our purposes, though, we just want all the data from the database, so we can use a `SELECT ALL` statement.

Rather than typing the SQL statement in our code, we can add it as a setting. We can then retrieve this settings quite easily.

On the Settings tab, then, click in the Name text box, the one just below **EmployeesConnectionString**. Type **SQL**. For the Type, leave it on **string**. Change the Scope to **Application**. For the Value, type the following:

SELECT * FROM tbl_employees

Your Settings page will then look like this:

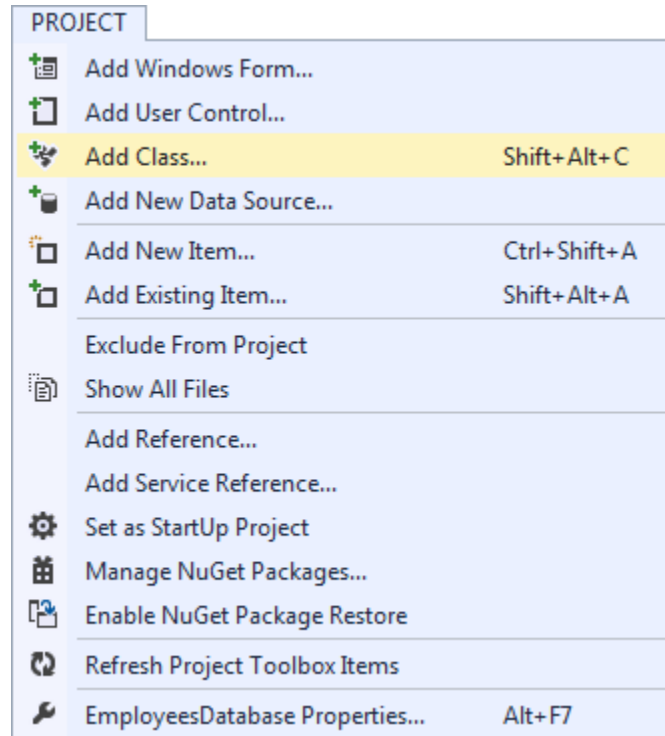
	Name	Type	Scope	Value
	EmployeesConnectionString	(Connection string)	Application	Data Source=.\SQLEXPRESS;AttachDbFilename= DataDirectory \Resources\Employees.mdf;Integrated Security=True;User Instance=True
▶	SQL	string	Application	SELECT * FROM tbl_employees
*				

Save your work and close the Properties page. We can now make a start with coding our database project.

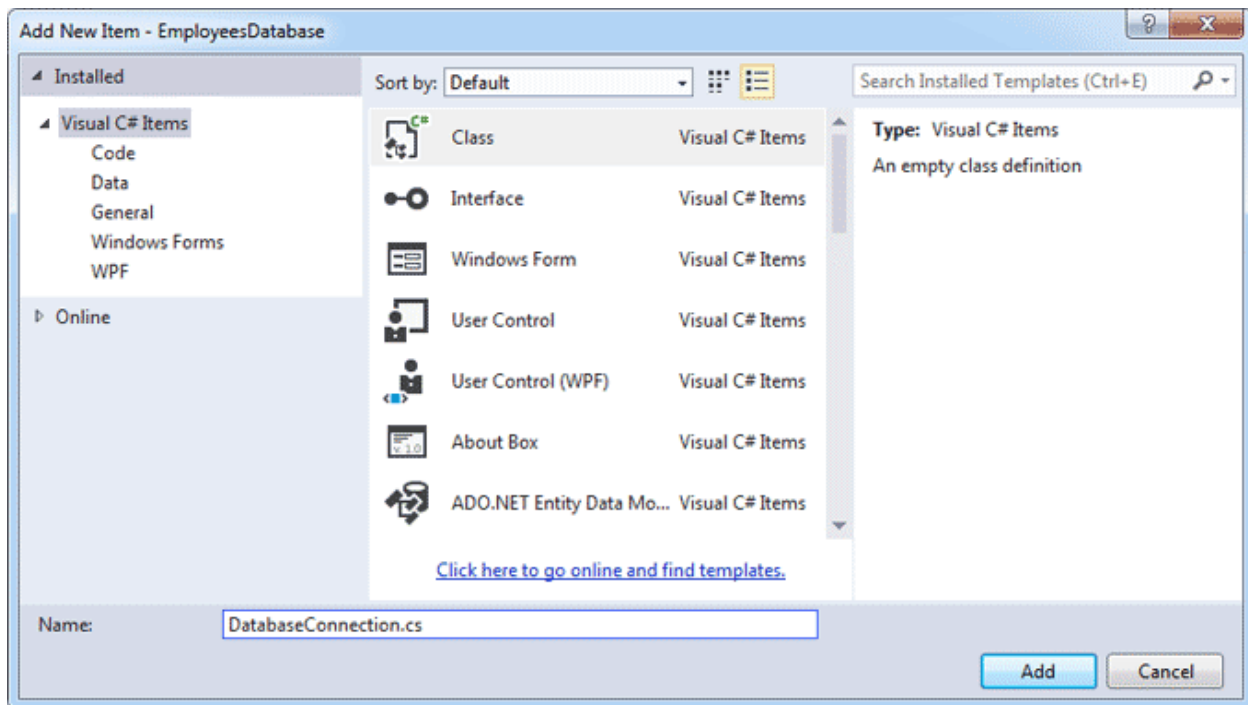
Creating A Database Connection Class

In order to connect to our database, we can create a class to handle all the connection issues.

From the Project menu at the top of Visual Studio, select **Add Class**:

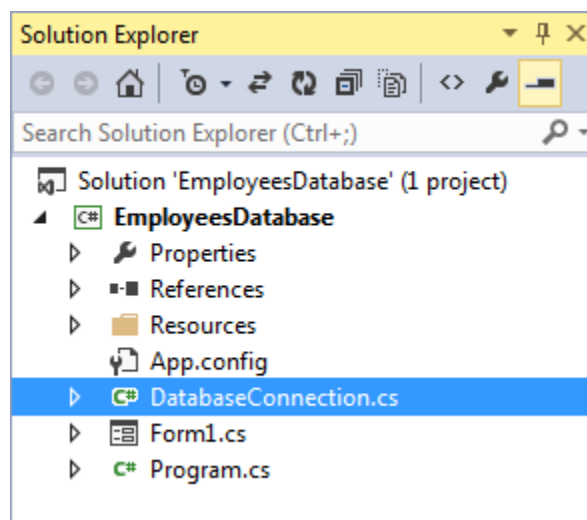


You'll see the following dialogue box appear:

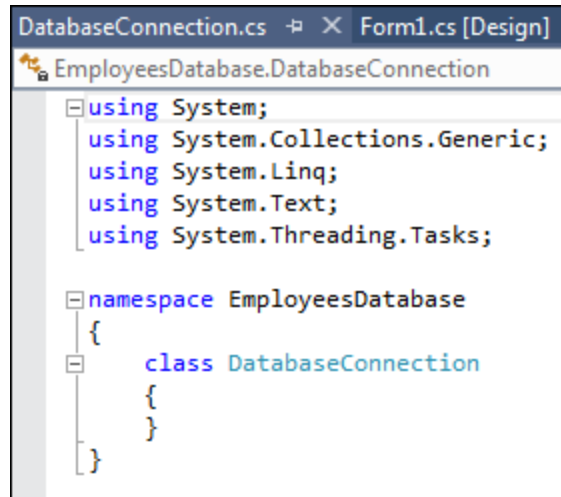


Make sure the **Class** item is selected. In the **Name** box at the bottom, type **DatabaseConnection.cs**. Then click the **Add** button at the bottom.

When you click Add, your new class will appear in the Solution Explorer on the right:



The code stub for your class will be open in the main window (if it's not, simply double click DatabaseConnection in the Solution Explorer):



```
DatabaseConnection.cs  Form1.cs [Design]
EmployeesDatabase.DatabaseConnection
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EmployeesDatabase
{
    class DatabaseConnection
    {
    }
}
```

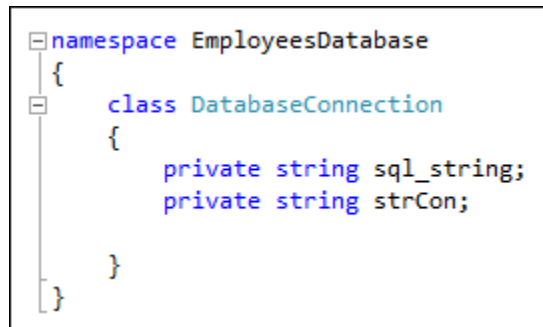
(2010 users won't have the Threading.Tasks using statement.)

The class we're going to create, as its name suggests, will help us to connect to a database. We'll add fields, properties and methods to the class.

The first thing to do is to set up two field variables:

```
private string sql_string;
private string strCon;
```

Add the two lines above to your class code and it will look like this



```
namespace EmployeesDatabase
{
    class DatabaseConnection
    {
        private string sql_string;
        private string strCon;
    }
}
```

The **sql_string** field variable will hold a SQL string like "SELECT * FROM table". We'll get this from the Settings we set up earlier. The **strCon** field variable will hold a location of the database. Again, we'll get this from the Settings we added.

The next code to add is a write-only property (meaning it has no get part):

```
public string Sql
{
    set { sql_string = value;
```



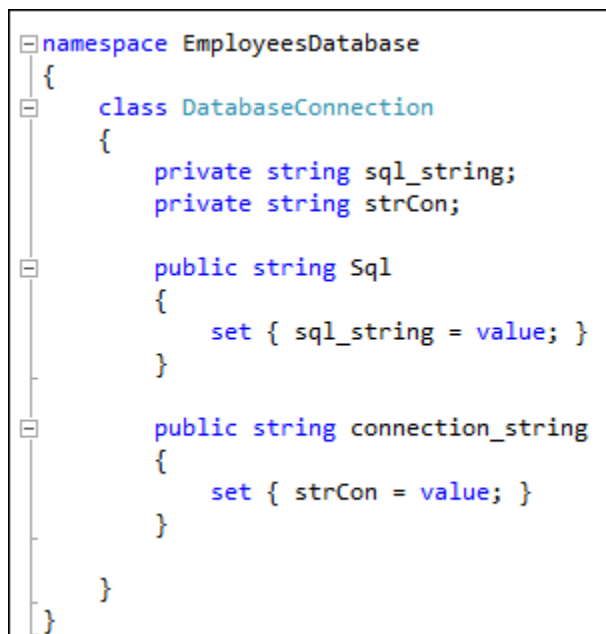
```
}
```

We've called this property **Sql**. It's going to be a **string**. It's **public** because we want to access this property from outside of this class. The **value** (the SQL itself) will be assigned to the **sql_string** field variable. We're saying, set the variable called **sql_string** to whatever is held in the **Sql** variable on line one.

The next bit of code sets something into the **strCon** field variable. This something is our connection string, the one we set up on the Settings page. Again, this is a write-only property.

```
public string connection_string  
{  
  
    set { strCon = value; }  
  
}
```

When you've added the two properties above, your code should look like this:



```
namespace EmployeesDatabase  
{  
    class DatabaseConnection  
    {  
        private string sql_string;  
        private string strCon;  
  
        public string Sql  
        {  
            set { sql_string = value; }  
        }  
  
        public string connection_string  
        {  
            set { strCon = value; }  
        }  
    }  
}
```

The next thing we need is something called a **DataSet**. Think of a DataSet as a grid that holds the data from our table. Each row in the DataSet grid will hold a row from our database table. The table is then closed, and the rows and columns in the DataSet is manipulated with code. In other words, you load table data into a DataSet, rather than manipulate the table in the database.

Add this code to you class:

```

public System.Data.DataSet GetConnection
{
    get { return MyDataSet( ); }
}

```

We've called this property **GetConnection**. This is a read-only property. For the get part we're calling a method with the name **MyDataSet**. The method connects to the database and fills a DataSet. It's this method that does all the work. Here's the code stub to add:

```

private System.Data.DataSet MyDataSet( )
{
}

```

The method is called **MyDataSet**. Instead of it being a string or an integer it's of this type:

System.Data.DataSet

So the type is **DataSet**. The inbuilt DataSet code lives in **System.Data**.

The first line to add to the new method is this rather long line:

```

System.Data.SqlClient.SqlConnection con = new
System.Data.SqlClient.SqlConnection( strCon );

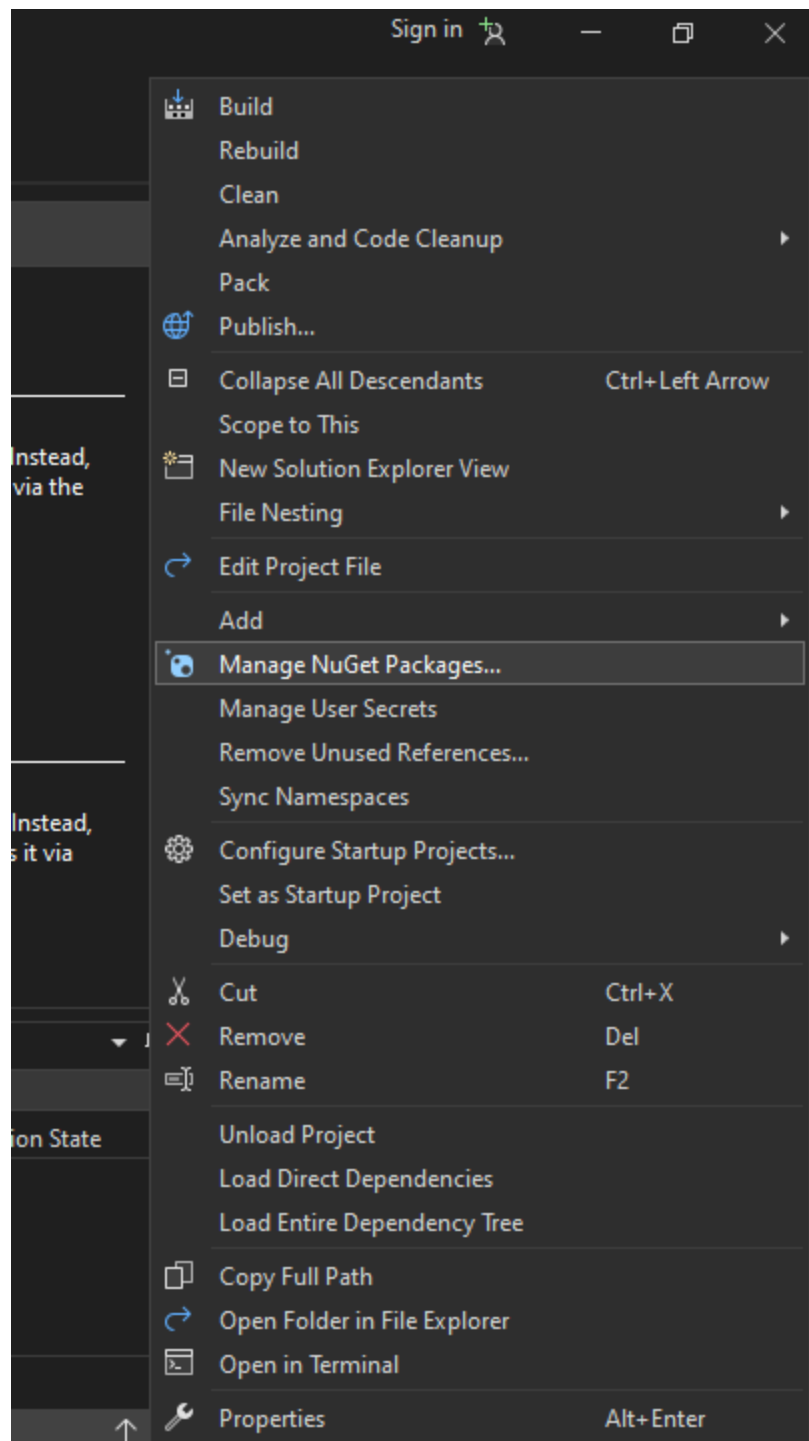
```

The variable we've created is called **con**. This variable is of type **SqlClient.SqlConnection**. Again, the SqlConnection code lives in **System.Data**. We're creating a new object of this type. In between the round brackets of **SqlConnection** we have our **strCon** variable. So SqlConnection will use our connection string in strCon to connect to the database.

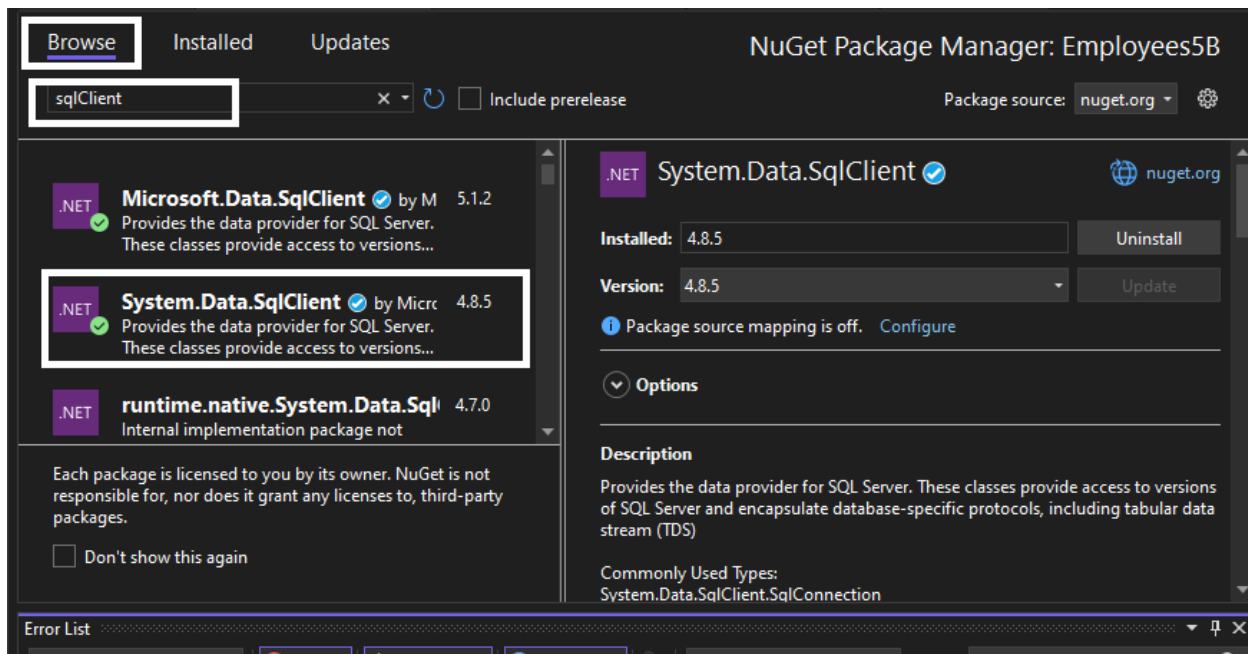
You might see an Error for SqlConnection because in newer version you need to separately install the package

So Right click on the project in the solution explorer shown in right top section.

And Click on **Manage NuGet Packages**



Click on Browse button on top left, Write SqlConnection in Search Box and locate System.Data.SqlClient. Download and install this package.



The next line to add is this:

```
con.Open( );
```

Here, we open a connection to the database.

However, although we have opened a connection to the database, we haven't yet opened the table in the database. To do that, we need something called a **DataAdapter**. A DataAdapter is used to open up a table in a database. We'll need to reference this DataAdapter later in our project, so we'll need to add a new line to the top of the Class. Add the following line just below the **sql_string** and **strCon** variables:

```
private string sql_string;  
private string strCon;  
System.Data.SqlClient.SqlDataAdapter da_1;
```

The variable we've set up is called **da_1**. It's of type **SqlConnection.SqlDataAdapter**.

Go back to your MyDataSet method and add this line:

```
da_1 = new System.Data.SqlClient.SqlDataAdapter(sql_string, con);
```

In between round brackets we have our **sql_string** variable. This is used to tell C# which records we want from the table. (We're going to be passing it that Settings string we added earlier.) After a comma, you need a connection object. This tells C# which database the table is in.

The next line is this:

```
System.Data.DataSet dat_set = new System.Data.DataSet( );
```

This sets up a **DataSet** object. The variable we've created is called **dat_set**. The whole method is going to be returning this DataSet, which will hold all the records from the table.

The final three lines are these:

```
da_1.Fill(dat_set, "Table_Data_1");  
  
con.Close( );  
  
return dat_set;
```

The first of those lines uses the **Fill** method of our DataAdapter object. This is used to fill a DataSet. In between the round brackets of Fill you need the name of a DataSet to fill. After a comma, you can add a name for the Fill, if you want. We've added one called **Table_Data_1**.

The final two lines close the connection object, and return the DataSet.

The whole of the code for your class should look like this:

```

namespace EmployeesDatabase
{
    class DatabaseConnection
    {
        private string sql_string;
        private string strCon;

        public string Sql
        {
            set { sql_string = value; }
        }

        public string connection_string
        {
            set { strCon = value; }
        }

        public System.Data.DataSet GetConnection
        {
            get
            { return MyDataSet(); }
        }

        private System.Data.DataSet MyDataSet()
        {
            System.Data.SqlClient.SqlConnection con = new System.Data.SqlClient.SqlConnection(strCon);

            con.Open();

            System.Data.SqlClient.SqlDataAdapter da_1 = new System.Data.SqlClient.SqlDataAdapter(sql_string, con);

            System.Data.DataSet dat_set = new System.Data.DataSet();
            da_1.Fill(dat_set, "Table_Data_1");
            con.Close();

            return dat_set;
        }
    }
}

```

Save your work and we'll put this new class to use. We'll do that in the next section below.

Using The Database Connection Class

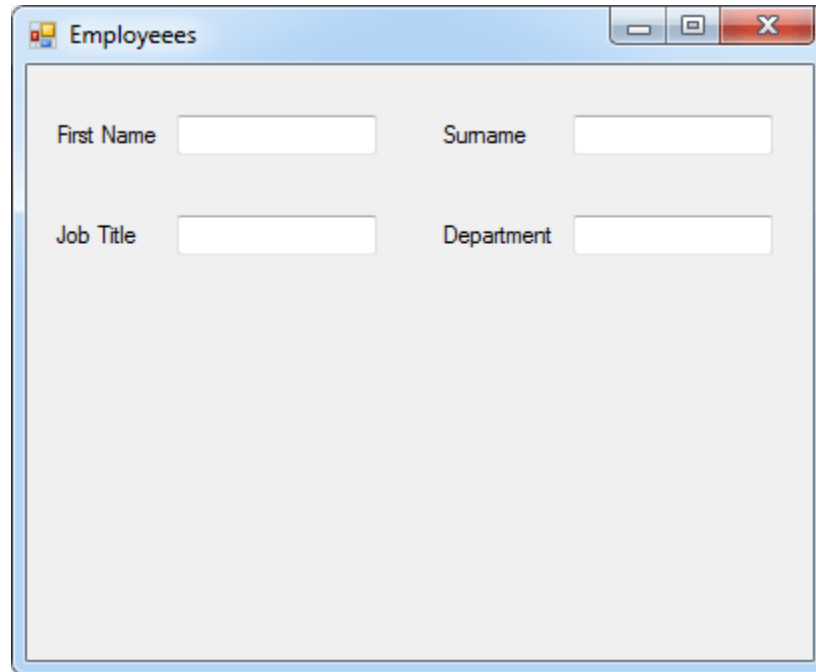
You now have a Windows form and a Database class that you created yourself. Go back to the form and add four textboxes. Change the Name property of each textbox to the following:

txtFirstName
txtSurname
txtJobTitle
txtDepartment

Add four labels to your form and place them next to the textboxes. Add the following text to the labels:

First Name
Surname
Job Title
Department

Your form will then look something like this:

A screenshot of a Windows application window titled "Employees". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. The main content area is light gray and contains four text input fields arranged in a 2x2 grid. The labels "First Name", "Surname", "Job Title", and "Department" are positioned to the left of each respective text box. The text boxes are empty.

What we're going to do now is to connect to the database using our class. We'll do this from the Form Load event. We'll place the first record from the table into the textboxes. Once this is done, we can then add buttons that will allow us to scroll forwards and backwards through all the records in the table.

To add the Form Load code stub, simply double click anywhere on your form that is not a textbox or a label. The code stub that opens up will look like this:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace EmployeesDatabase
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        }
    }
}

```

Because we're going to be adding buttons later, we need to set up variables where all the buttons can see them. The first one we need is a variable to store our connection object. Add the following line just outside of the Form Load event:

DatabaseConnection objConnect;

The variable is called **objConnect**. It is of type **DatabaseConnection**, which is the name of our class.

We also need a string to hold our connection string from the Setting page we set up earlier. So add this, as well:

string conString;

This is just a normal string variable that we've called **conString**.

Your coding window should now look like this:


```

namespace EmployeesDatabase
{
    public partial class Form1 : Form
    {
        public Form1()...

        DatabaseConnection objConnect;
        string conString;

        private void Form1_Load(object sender, EventArgs e)
        {
        }
    }
}

```

We also need a **DataSet** object. This is because the **GetConnection** method in our class is set up to load all the database data into a DataSet. When we call our GetConnection method we'll need somewhere to put the DataSet, which will be another DataSet. So add this line to your code, just below the other two:

DataSet ds;

A DataSet contains rows, which correspond to a row in the database table. To manipulate each row, you work with a **DataRow** object. You'll see how this works shortly. But add this line to your code, as well:

DataRow dRow;

The final two variables we need to add outside of the form load event are these two:

int MaxRows;
int inc = 0;

The two variables are both integers. The first one, MaxRows, will tell us how many rows there are in the DataSet, which is how many rows were pulled from the database table. The other integer variable, inc, will be used to move from one record to another, and back again. We'll need this for the buttons we'll add later.

But your coding window should now look like this:

```
DatabaseConnection objConnect;  
string conString;  
  
DataSet ds;  
DataRow dRow;  
  
int MaxRows;  
int inc = 0;  
  
private void Form1_Load(object sender, EventArgs e)  
{  
    try  
    {  
  
    }  
    catch (Exception err)  
    {  
        MessageBox.Show(err.Message);  
    }  
}
```

We can now turn our attention to the Form Load event. We'll place the code for this event in a try ... catch statement. So add the following to your form load event:

```
try  
{  
  
}  
catch (Exception err)  
{  
  
    MessageBox.Show(err.Message);  
  
}
```

Your code window should look like this:

```

DatabaseConnection objConnect;
string conString;

DataSet ds;
DataRow dRow;

int MaxRows;
int inc = 0;

private void Form1_Load(object sender, EventArgs e)
{
    try
    {

    }
    catch (Exception err)
    {
        MessageBox.Show(err.Message);
    }
}

```

The first thing to do in the try part is to set up an object from our class. Here's the line that does that:

objConnect = new DatabaseConnection();

This creates a new object for us, of type **DatabaseConnection**. (Don't forget the round brackets on the end.)

We can now grab that connection we set up in the Settings page earlier. To do that, start with your conString variable then an equal sign:

conString =

After the equal sign, type the word **Properties**, then a dot. As soon as you type the dot, you'll see the IntelliSense list appear:

The screenshot shows a code editor with the following text:

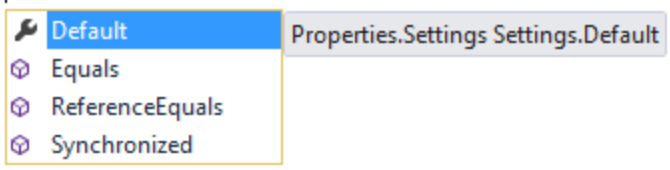

```

try
{
    objConnect = new DatabaseConnection();
    conString = Properties.|
}
    
```

 An IntelliSense dropdown menu is visible below the cursor, showing two options: 'Resources' and 'Settings'. The 'Settings' option is highlighted in blue. To the right of the dropdown, the text 'class EmployeesDatabase.Properties.Settings' is visible.

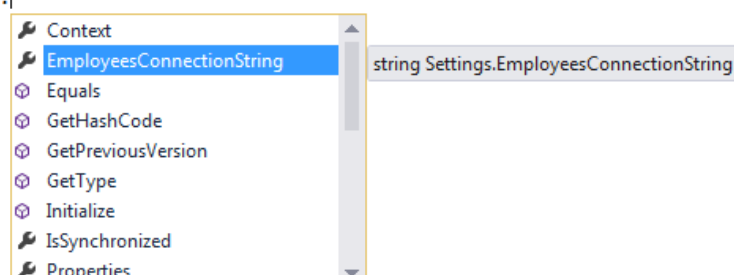
Select **Settings** (which refers to the Settings page), then type another dot. You'll see the IntelliSense list again:

```
try
{
    objConnect = new DatabaseConnection();
    conString = Properties.Settings.|
}
```



Select **Default**, and type another dot:

```
try
{
    objConnect = new DatabaseConnection();
    conString = Properties.Settings.Default.|
}
```

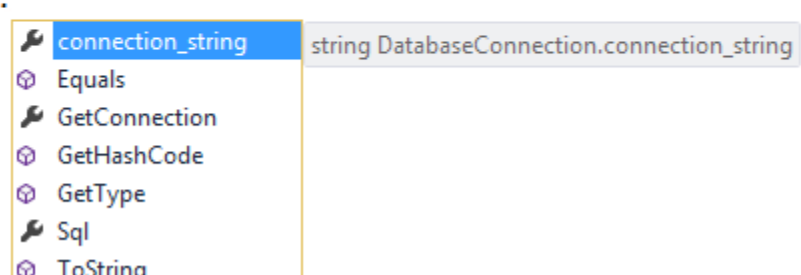


You should see your connection string on the list, which was **EmployeesConnectionString** for us. Select this and then end the line with a semicolon.

Now that we've placed the connection string into our **conString** variable, we can hand it over to our class. Type **objConnect** (the name of our class object in the Form), then a dot. You should see this:

```
private void Form1_Load(object sender, EventArgs e)
{
    try
    {
        objConnect = new DatabaseConnection();
        conString = Properties.Settings.Default.EmployeesConnectionString;

        objConnect.|
    }
}
```



Our **connection_string** property is showing up on the IntelliSense list. This allows us to pass over the connection string to our DatabaseConnection class.

Finish the line by adding the conString variable:

```
objConnect.connection_string = conString;
```

The next thing we need to do is to pass over some SQL to our DatabaseConnection class. We set this up on the Settings page. The SQL was this:

```
SELECT * FROM tbl_employees
```

This reads "Select all the records from the table called tbl_employees".

Add the following line to your code:

```
objConnect.Sql = Properties.Settings.Default.SQL;
```

You should see the IntelliSense list appear as you're typing the above line. At the end, is the name of the setting we added - **SQL**. This all gets passed to our DatabaseConnection class via the Sql property before the equal sign.

What we've done so far is to hand our **DatabaseConnection** class a connection string, which contains the name and location of the database; and we've also handed it some SQL, so that we can pull records from a table in the database. We set up a method in our DatabaseConnection class that takes these two values (connection string and SQL), and uses them to place data from the database into a DataSet. Let's now add code to return that DataSet.

The name of our DataSet in the Form code is **ds**. This is the variable we set up at the top of the Form Load event. We can call our **GetConnection** method from the DatabaseConnection class and hand its DataSet over to the variable ds. Add this code:

```
ds = objConnect.GetConnection;
```

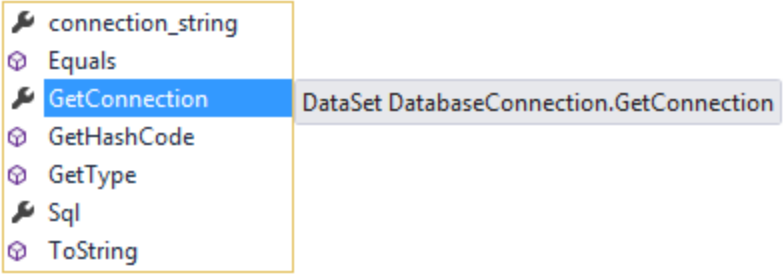
So **objConnect** is our object created from the DatabaseConnection class. Type a dot and you'll see the IntelliSense list appear:

```
try
{
    objConnect = new DatabaseConnection();
    conString = Properties.Settings.Default.EmployeesConnectionString;

    objConnect.connection_string = conString;

    objConnect.Sql = Properties.Settings.Default.SQL;

    ds = objConnect.
}
```



The **GetConnection** method is on the list. Double click to add it to your code, and then type a semicolon to end the line.

We can count how many records are in the DataSet. (These are the same records that are in the database table, remember.) You do it like this:

```
MaxRows = ds.Tables[0].Rows.Count;
```

MaxRows is the integer variable we set up at the top of the Form Load event. After an equal sign, we have this:

```
ds.Tables[0].Rows.Count;
```

The name of our DataSet is **ds**. The DataSet has a property called **Tables**. This is a list of all the tables in your DataSet. (We only have one table.) The first table is at position 0. The 0 goes between square brackets. After a dot, you type **Rows.Count**. This counts how many rows are in the DataSet.

To fill the textboxes, we can add a method. Let's call it **NavigateRecords**:

```
private void NavigateRecords()
{
}

```

We don't need to return a value so the method is set up as **void**.

Because we want to access one row at a time, we can use the DataRow object we set up earlier. If you wanted to access row 1 from the DataSet, the code would be this:

```
dRow = ds.Tables[0].Rows[1];
```

Here, the entire first row will be placed in the **dRow** variable. To place this first row into the textboxes we can use the **ItemArray** property of DataRow objects. The ItemArray property has a method called **GetValue**. In between the round brackets of GetValue you type the column you want to access. For example, take this code:

```
txtFirstName.Text = dRow.ItemArray.GetValue(1).ToString();
```

The value in between the round brackets of **GetValue** is 1. This will place row 1, column 1 into the textbox called **txtFirstName**. If we then advanced the row counter to 2, row 2 column 1 will be placed into the textbox. Advanced the row counter again and row 3 column 1 will end up in the textbox. (The ToString at the end just ensures that any non-text values get converted to strings.)

To advance the row counter, we can use our **inc** variable:

```
dRow = ds.Tables[0].Rows[inc];
```

So instead of hard-coding a value of 1 between Rows, we're using whatever value is inside of the **inc** variable. We can then increment the **inc** variable later from buttons on the form.

So add the following code, just below the Form Load event:

```
private void NavigateRecords()  
{  
  
    dRow = ds.Tables[0].Rows[inc];  
  
    txtFirstName.Text = dRow.ItemArray.GetValue(1).ToString();  
    txtSurname.Text = dRow.ItemArray.GetValue(2).ToString();  
    txtJobTitle.Text = dRow.ItemArray.GetValue(3).ToString();  
    txtDepartment.Text = dRow.ItemArray.GetValue(4).ToString();  
  
}
```

Now add a call to NavigateRecords from your Form Load event:

```
MaxRows = ds.Tables[0].Rows.Count;  
  
NavigateRecords( );
```

The whole of your code should now look like this:

```
DatabaseConnection objConnect;
string conString;

DataSet ds;
DataRow dRow;

int MaxRows;
int inc = 0;

private void Form1_Load(object sender, EventArgs e)
{
    try
    {
        objConnect = new DatabaseConnection();
        conString = Properties.Settings.Default.EmployeesConnectionString;

        objConnect.connection_string = conString;
        objConnect.Sql = Properties.Settings.Default.SQL;

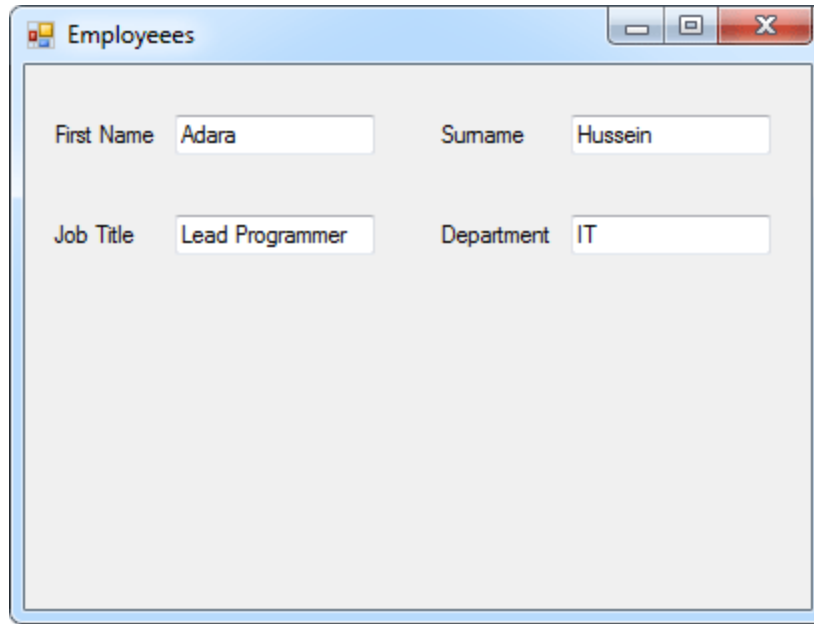
        ds = objConnect.GetConnection;
        MaxRows = ds.Tables[0].Rows.Count;

        NavigateRecords();

    }
    catch (Exception err)
    {
        MessageBox.Show(err.Message);
    }
}

private void NavigateRecords()
{
    dRow = ds.Tables[0].Rows[inc];
    txtFirstName.Text = dRow.ItemArray.GetValue(1).ToString();
    txtSurname.Text = dRow.ItemArray.GetValue(2).ToString();
    txtJobTitle.Text = dRow.ItemArray.GetValue(3).ToString();
    txtDepartment.Text = dRow.ItemArray.GetValue(4).ToString();
}
}
```

You can try out your code, now. Run our programme and you should find that the first record from the database table appears in your textboxes:

A screenshot of a Windows application window titled "Employees". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. Inside the window, there is a form with four text input fields arranged in a 2x2 grid. The first row contains "First Name" with the value "Adara" and "Surname" with the value "Hussein". The second row contains "Job Title" with the value "Lead Programmer" and "Department" with the value "IT".

In the next lesson, you'll add some buttons to the form so that you can scroll forwards and backwards through all the records in the database table.

Moving Forward And Backward Through The Database

To move forward through the database table, add a button to your form. Give it the Name **btnNext**. Add some text to the button.

Double click your button to get at the coding window. For the code, we need to check what is inside of the **MaxRows** variable and make sure we don't go past it. MaxRows, remember, is holding how many records are in the DataSet. We also need to increment the **inc** variable. It is this variable that will move us on to the next record.

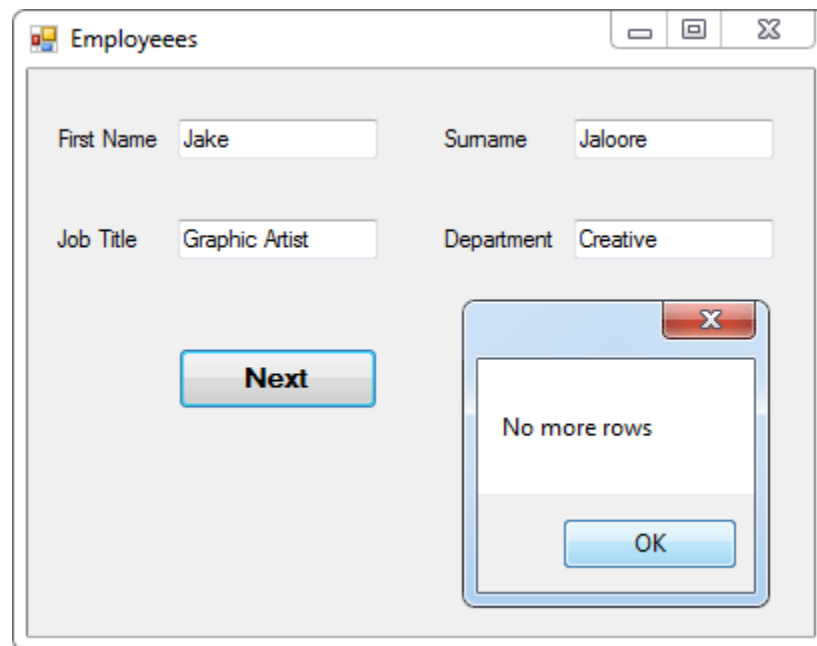
Add the following if statement to your button:

```
private void btnNext_Click(object sender, EventArgs e)
{
    if (inc != MaxRows - 1)
    {
        inc++;
        NavigateRecords();
    }
    else
    {
        MessageBox.Show("No More Rows");
    }
}
```

The first line of the If Statement says "If **inc** does not equal **MaxRows** minus 1". If it doesn't then we increment the **inc** variable and call **NavigateRecords**. But can you see why we need to say **MaxRows - 1**? It's because of the **Rows[inc]** line in our **NavigateRecords** method. The count for Rows starts at zero. So if we only have 4 records in the database, the count will be for 0 to 3. MaxRows, however, will be 4. If we don't deduct 1, the programme will crash with an error: **IndexOutOfRangeException**.

If MaxRows is reached, then we can display a message for the user: "No more rows".

Run your programme and test it out. You should be able to move forward through your database. Here's what your form should look like when the last record is reached:



Move Backwards through the Database

We can use similar code to move backwards through the records in the database.

Add another button to your form. Change the Text property to **Previous Record**. Change the Name property to **btnPrevious**.

Double click your new button to get at the coding window. Now add the following:

```
if (inc > 0)
{
```

```

        inc--;
        NavigateRecords( );
    }
    else
    {

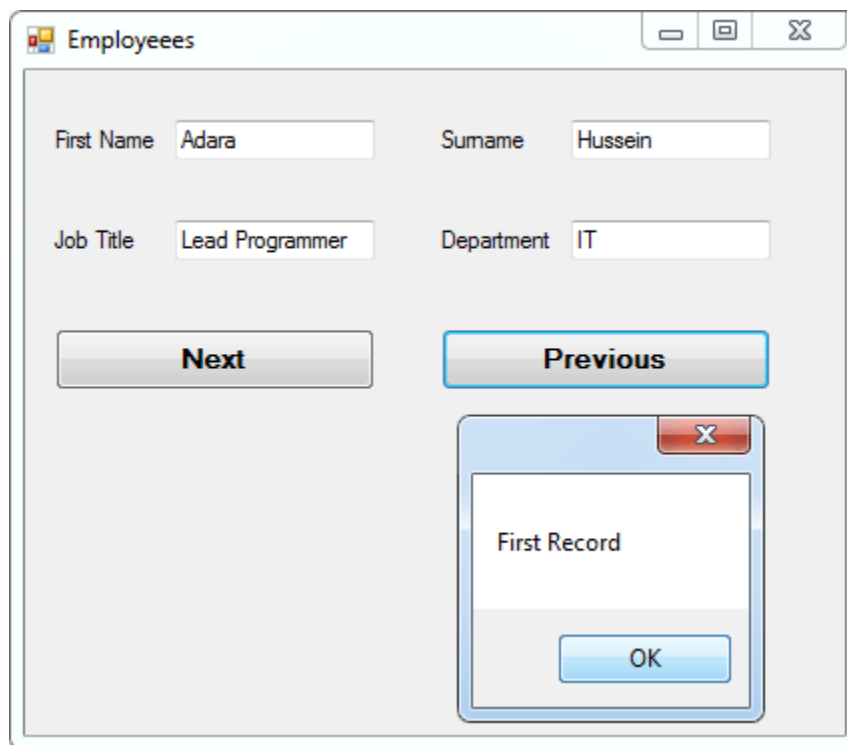
        MessageBox.Show("First Record");

    }

```

The if statement is now only checking the **inc** variable. We need to check if it's greater than zero. If it is, we can deduct 1 from **inc**, and then call our **NavigateRecords** methods. When the form loads, remember, **inc** will be 0. So if we tried to move back one record after the form first loads the programme would crash. It would crash because we'd be trying to access **Rows[-1]**.

Run your programme and test it out. Click you Previous button and you should see this:



Jump to the Last Record in your Database

To move to the last record of your database, you only need to make sure that the **inc** variable and **MaxRows** have the same value.

Add a new button to your form. Set the Text property as **Last Record**, and the Name property as **btnLast**. Double click the button, and add the following code:

```
if (inc != MaxRows - 1)
{

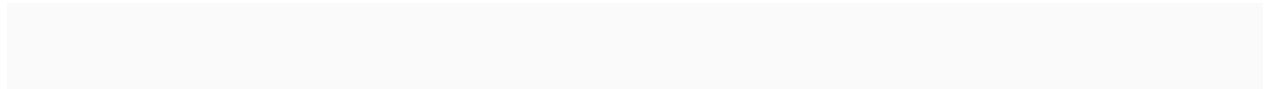
    inc = MaxRows - 1;
    NavigateRecords( );

}
```

The If Statement again checks that **inc** is not equal to **MaxRows** minus 1. If it isn't, we have this:

```
inc = MaxRows - 1;
```

MaxRows minus 1 would equal 3 in a four record database. Because **Rows[inc]** goes from 0 to 3, this is enough to move to the last record after the call to **NavigateRecords**.



Jump to the First Record in your Database

To move to the first record in the database, we only need to set inc to zero.

Add another button to your form. Change the Text property to First Record. Change the Name property to **btnFirst**. Double click your new button and add the following code:

```
if (inc != 0)
{

    inc = 0;
    NavigateRecords( );

}
```

This just checks to see if inc isn't already zero. If it isn't, we set the inc variable to 0. Then we call the **NavigateRecords** method.

Run your programme and test it out. You should now be able to move through the records in your database without the programme crashing. What we'll do now is to allow the user to add a new record to the database. This is more complex than the navigation, so you may need to pay close attention!