PROG6212

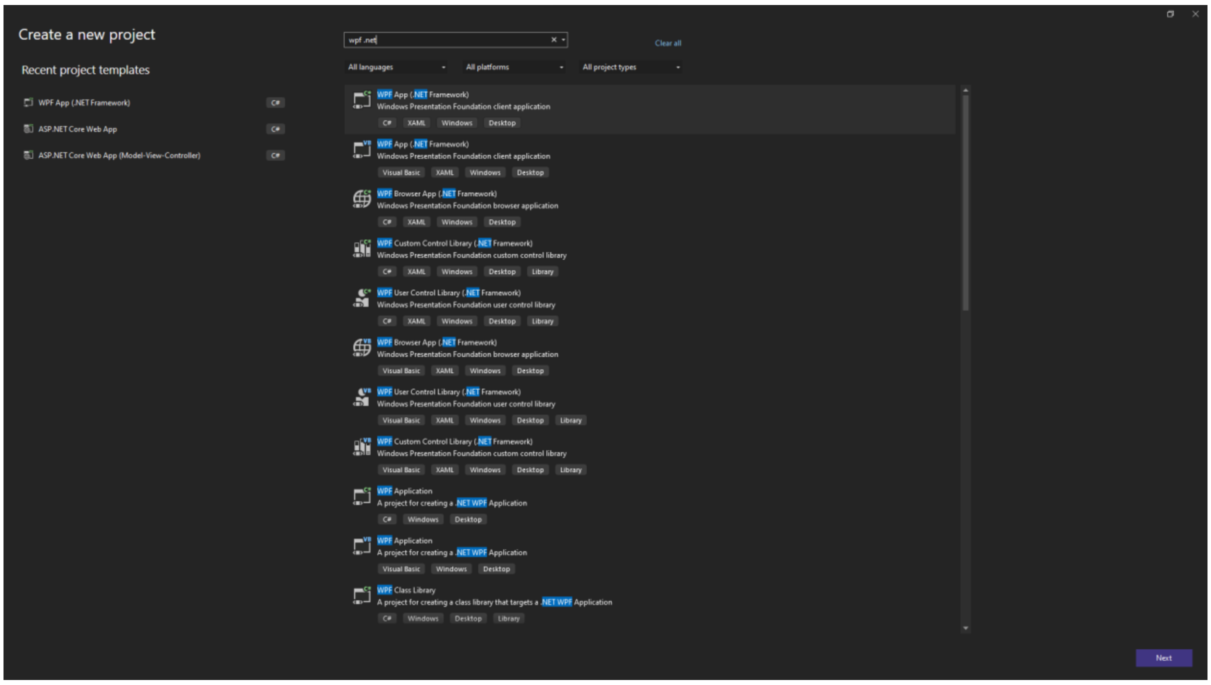
PART 2: Basic WPF App with SQL Database

By Aaron Fourie

Contents

Creating a new WPF App

Create a new **WPF App(.NET Framework)** Project



Create a new project

Recent project templates

- WPF App (.NET Framework) — C#
- ASP.NET Core Web App — C#
- ASP.NET Core Web App (Model-View-Controller) — C#

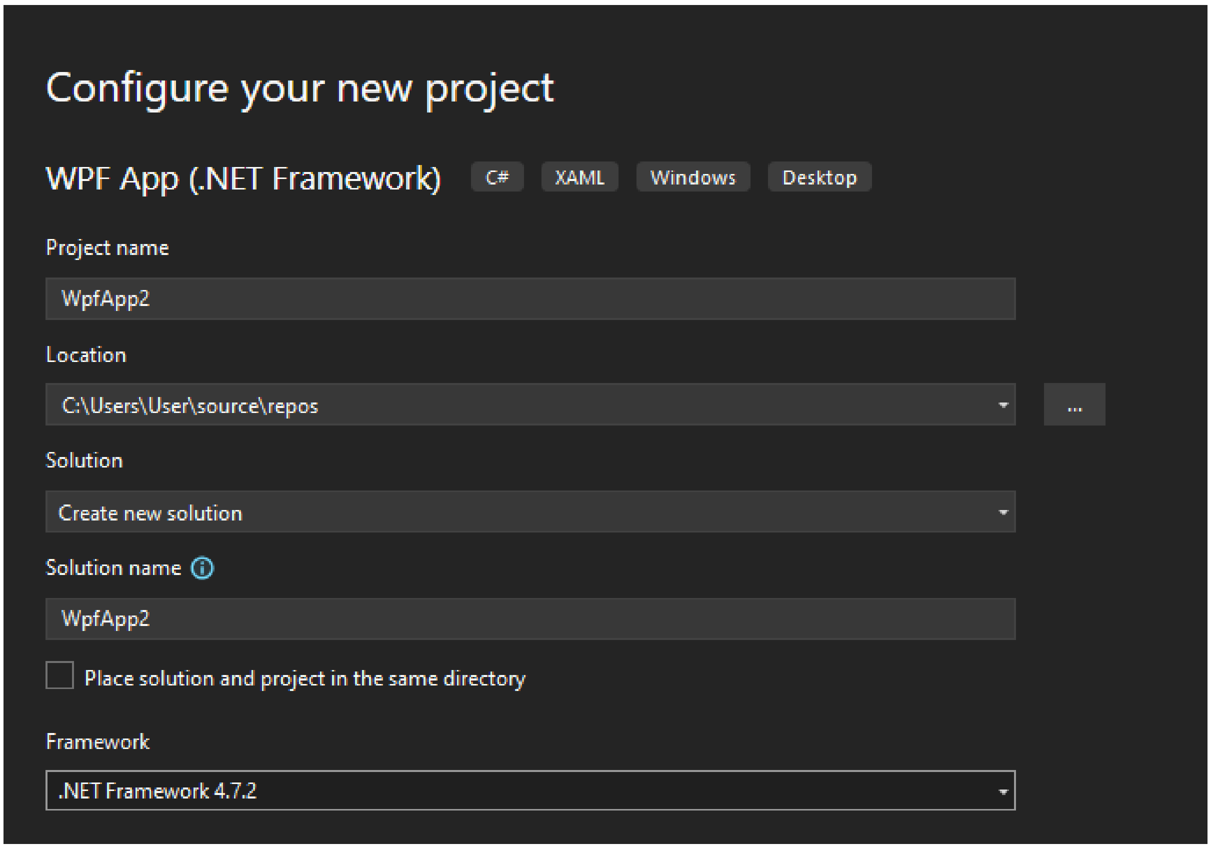
Search: wpf .net

Clear all

All languages | All platforms | All project types

- WPF App (.NET Framework)
  Windows Presentation Foundation client application
  C# · XAML · Windows · Desktop
- WPF App (.NET Framework)
  Windows Presentation Foundation client application
  Visual Basic · XAML · Windows · Desktop
- WPF Browser App (.NET Framework)
  Windows Presentation Foundation browser application
  C# · XAML · Windows · Desktop
- WPF Custom Control Library (.NET Framework)
  Windows Presentation Foundation custom control library
  C# · XAML · Windows · Desktop · Library
- WPF User Control Library (.NET Framework)
  Windows Presentation Foundation user control library
  C# · XAML · Windows · Desktop · Library
- WPF Browser App (.NET Framework)
  Windows Presentation Foundation browser application
  Visual Basic · XAML · Windows · Desktop
- WPF User Control Library (.NET Framework)
  Windows Presentation Foundation user control library
  Visual Basic · XAML · Windows · Desktop · Library
- WPF Custom Control Library (.NET Framework)
  Windows Presentation Foundation custom control library
  Visual Basic · XAML · Windows · Desktop · Library
- WPF Application
  A project for creating a .NET WPF Application
  C# · Windows · Desktop
- WPF Application
  A project for creating a .NET WPF Application
  Visual Basic · Windows · Desktop
- WPF Class Library
  A project for creating a class library that targets a .NET WPF Application
  C# · Windows · Desktop · Library

Select the **.NET Framework 4.7.2** version



Configure your new project

WPF App (.NET Framework) — C# · XAML · Windows · Desktop

Project name: WpfApp2

Location: C:\Users\User\source\repos

Solution: Create new solution

Solution name: WpfApp2

☐ Place solution and project in the same directory

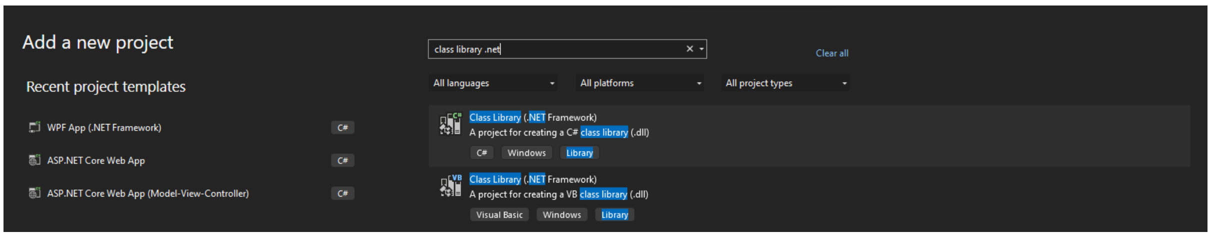
Framework: .NET Framework 4.7.2

Click **Create**

Adding a class library

In your Solution Explorer go to your **Solution tab** and click **Add -> New Project**

Create a **class library (.NET Framework)**



Add a new project

Recent project templates

WPF App (.NET Framework) C#
ASP.NET Core Web App C#
ASP.NET Core Web App (Model-View-Controller) C#

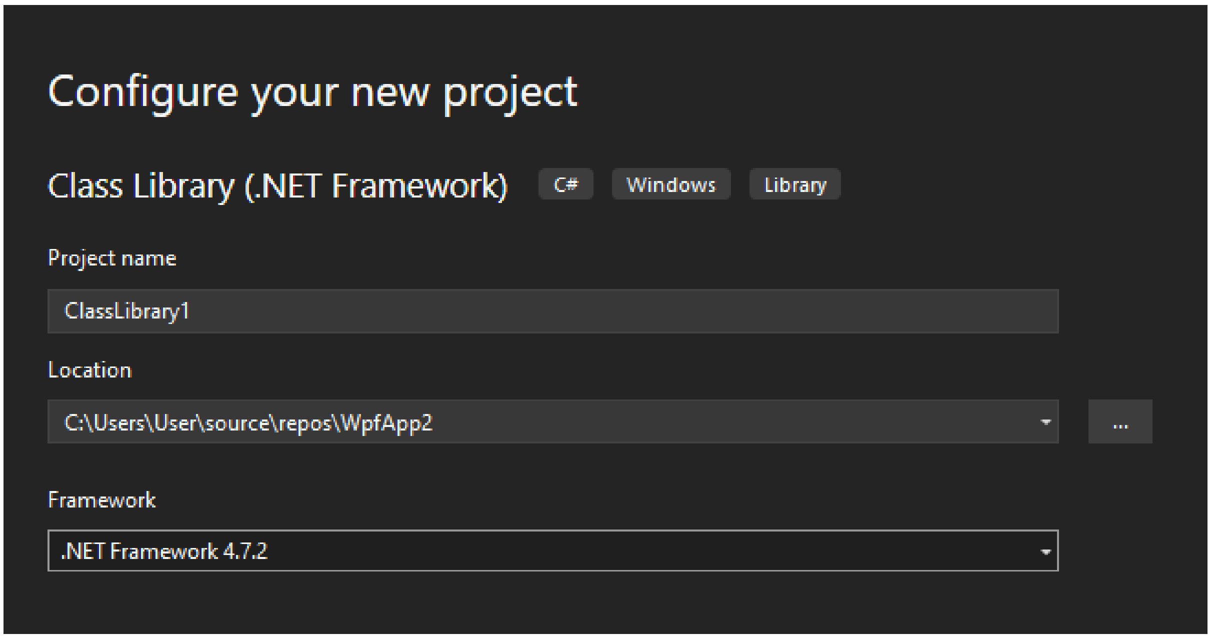
class library .net

All languages · All platforms · All project types · Clear all

Class Library (.NET Framework)
A project for creating a C# class library (.dll)
C# Windows Library

Class Library (.NET Framework)
A project for creating a VB class library (.dll)
Visual Basic Windows Library

Choose **.NET Framework 4.7.2** and click create



Configure your new project

Class Library (.NET Framework) C# Windows Library

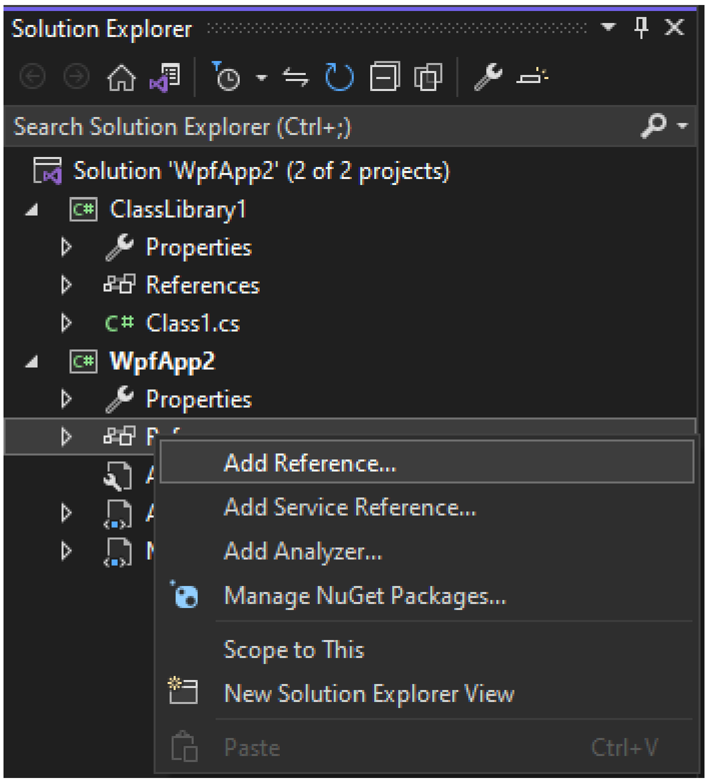
Project name: ClassLibrary1

Location: C:\Users\User\source\repos\WpfApp2

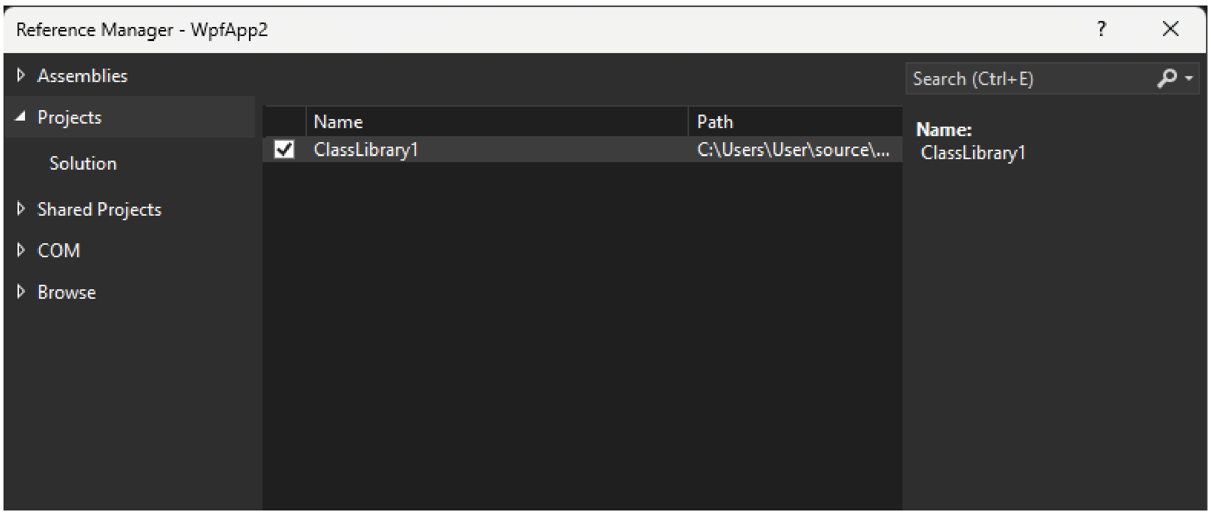
Framework: .NET Framework 4.7.2

To add your class library to your WPF Project as a reference

Go to your WPF Project in the solution explorer and click **References** -> **Add Reference**



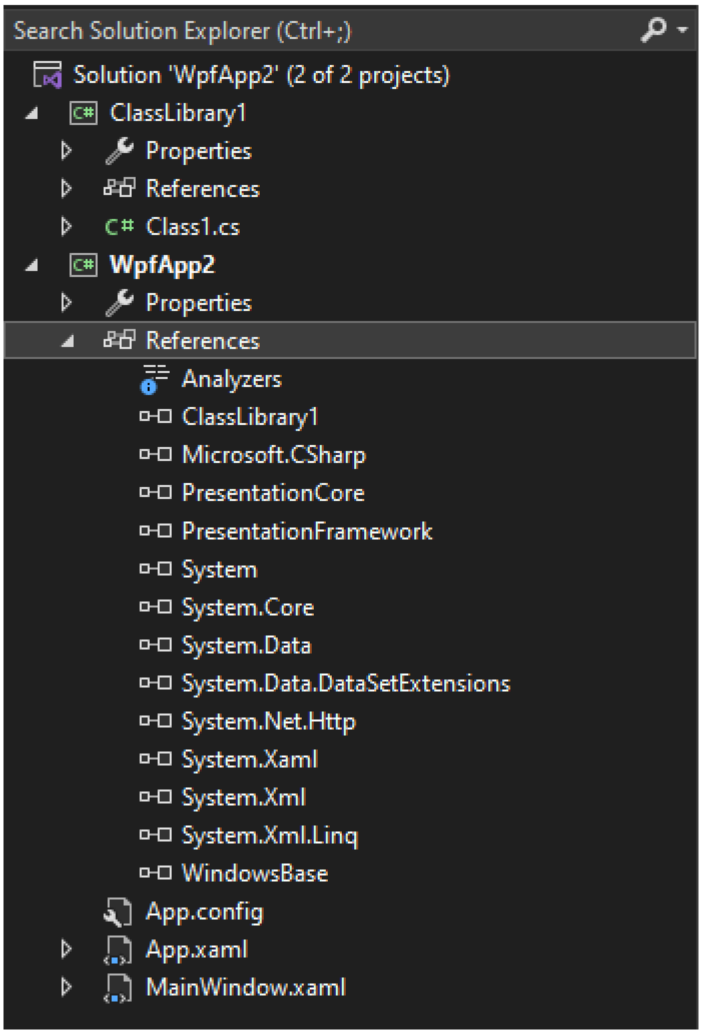
In the Reference Manager go to **Projects** and **select your class library**, then click **OK**



	Name	Path
✓	ClassLibrary1	C:\Users\User\source\...

Name:
ClassLibrary1

Your class library is now added to your project and your solution explorer should look similar to this



```
Solution 'WpfApp2' (2 of 2 projects)
  ClassLibrary1
    Properties
    References
    Class1.cs
  WpfApp2
    Properties
    References
      Analyzers
      ClassLibrary1
      Microsoft.CSharp
      PresentationCore
      PresentationFramework
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Net.Http
      System.Xaml
      System.Xml
      System.Xml.Linq
      WindowsBase
    App.config
    App.xaml
    MainWindow.xaml
```

We'll come back to your class library later in the project.

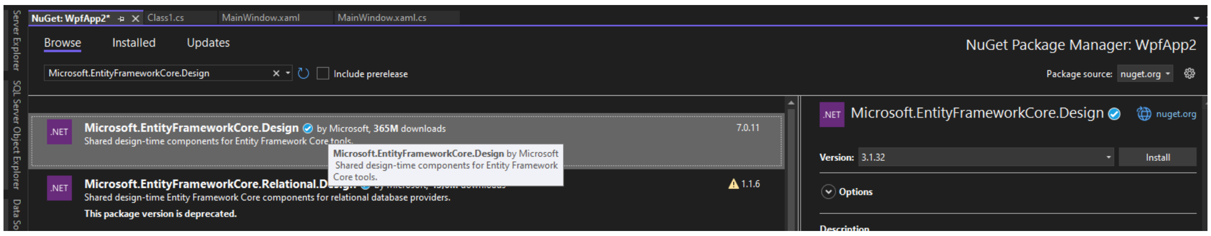
Installing the necessary NuGet Packages

Now let’s setup the necessary SQL database packages so we can use it in our WPF App

Go to your WPF Project in the solution explorer and open **Manage NuGet Packages…**

Go to **browse**, then **search** and **install** the following packages:

1. Microsoft.EntityFrameworkCore.Design (Version 3.1.32)
2. Microsoft.EntityFrameworkCore.Tools (Version 3.1.32)
3. Microsoft.EntityFrameworkCore.SqlServer (Version 3.1.32)



Microsoft.EntityFrameworkCore.Design

Package source: nuget.org

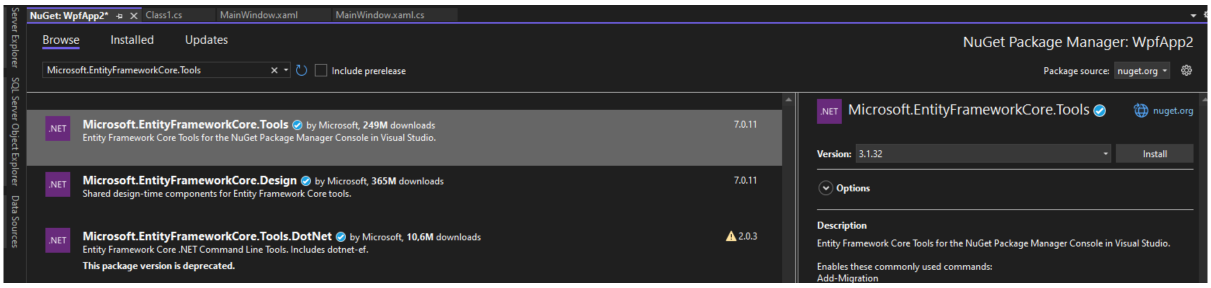
Microsoft.EntityFrameworkCore.Design by Microsoft, 365M downloads 7.0.11
Shared design-time components for Entity Framework Core tools.

Microsoft.EntityFrameworkCore.Relational.Design by Microsoft 1.1.6
Shared design-time Entity Framework Core components for relational database providers.
This package version is deprecated.

Microsoft.EntityFrameworkCore.Design

Version: 3.1.32

Figure 1: 1. Microsoft.EntityFrameworkCore.Design(Version 3.1.32)



Microsoft.EntityFrameworkCore.Tools

Package source: nuget.org

Microsoft.EntityFrameworkCore.Tools by Microsoft, 249M downloads 7.0.11
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Microsoft.EntityFrameworkCore.Design by Microsoft, 365M downloads 7.0.11
Shared design-time components for Entity Framework Core tools.

Microsoft.EntityFrameworkCore.Tools.DotNet by Microsoft, 10,6M downloads 2.0.3
Entity Framework Core .NET Command Line Tools. Includes dotnet-ef.
This package version is deprecated.

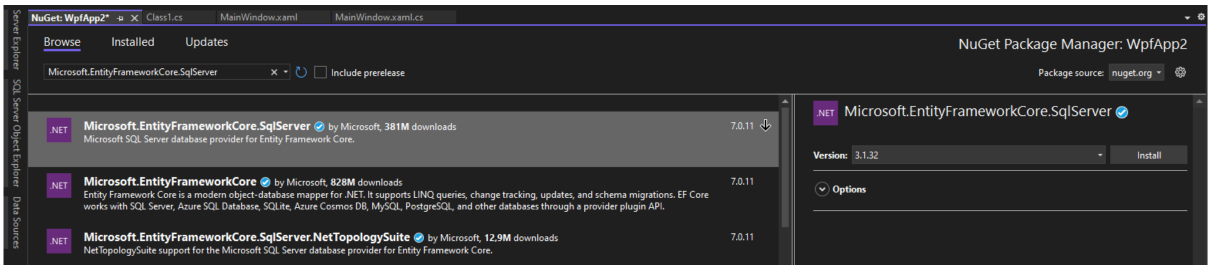
Microsoft.EntityFrameworkCore.Tools

Version: 3.1.32

Description
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Enables these commonly used commands:
Add-Migration

Figure 2: Microsoft.EntityFrameworkCore.Tools(Version 3.1.32)



Microsoft.EntityFrameworkCore.SqlServer

Package source: nuget.org

Microsoft.EntityFrameworkCore.SqlServer by Microsoft, 381M downloads 7.0.11
Microsoft SQL Server database provider for Entity Framework Core.

Microsoft.EntityFrameworkCore by Microsoft, 828M downloads 7.0.11
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.

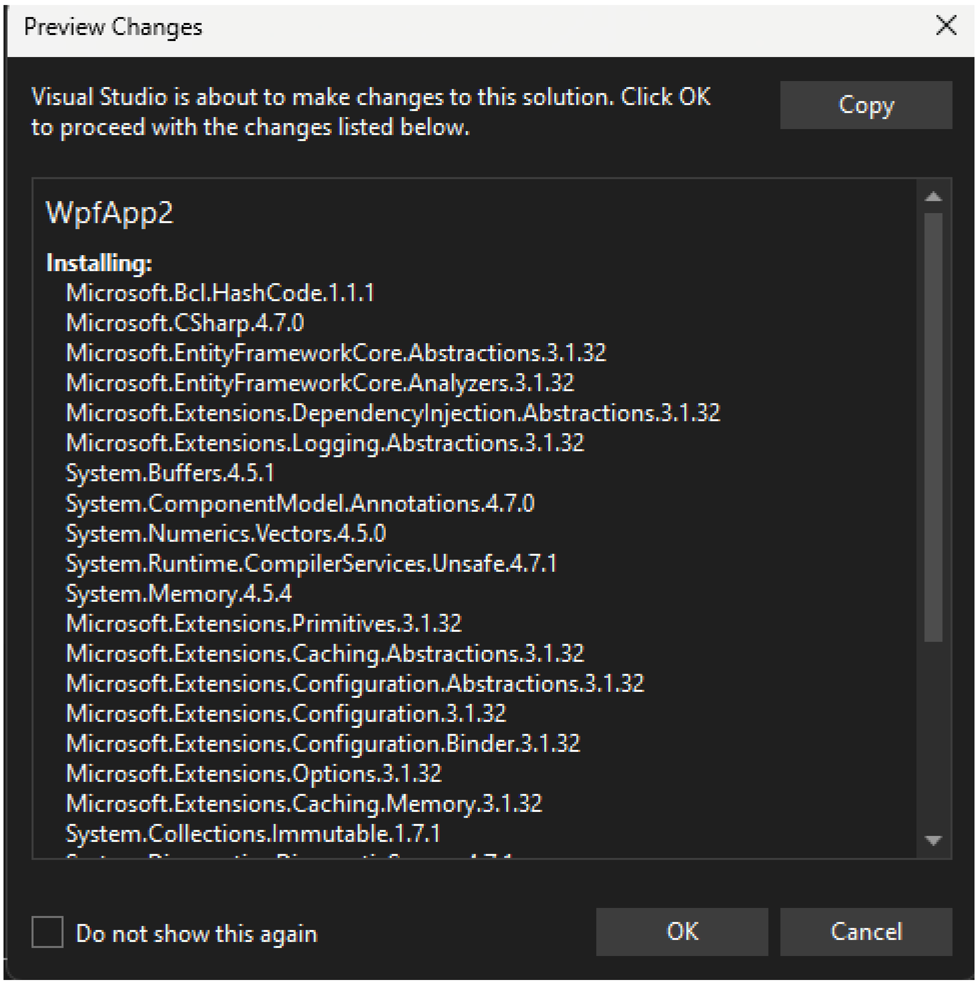
Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite by Microsoft, 12,9M downloads 7.0.11
NetTopologySuite support for the Microsoft SQL Server database provider for Entity Framework Core.

Microsoft.EntityFrameworkCore.SqlServer

Version: 3.1.32

Figure 3: Microsoft.EntityFrameworkCore.SqlServer(Version 3.1.32)

After each packages installs, Click **OK** in the **Preview Changes** Window



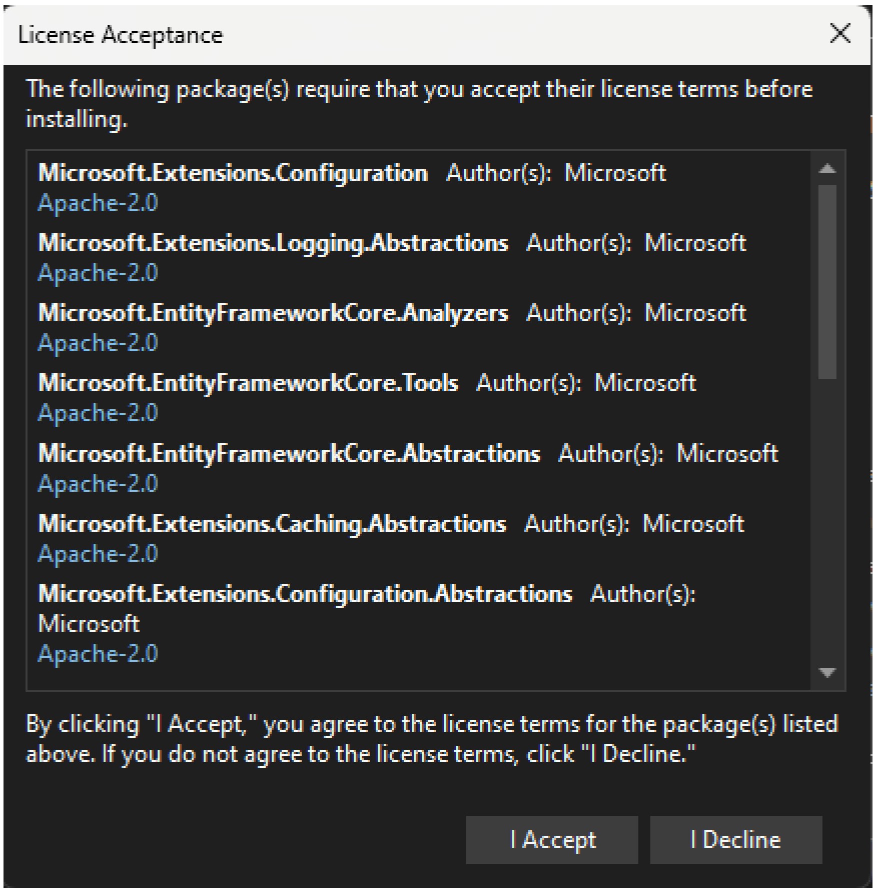
Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.

WpfApp2

Installing:

```
Microsoft.Bcl.HashCode.1.1.1
Microsoft.CSharp.4.7.0
Microsoft.EntityFrameworkCore.Abstractions.3.1.32
Microsoft.EntityFrameworkCore.Analyzers.3.1.32
Microsoft.Extensions.DependencyInjection.Abstractions.3.1.32
Microsoft.Extensions.Logging.Abstractions.3.1.32
System.Buffers.4.5.1
System.ComponentModel.Annotations.4.7.0
System.Numerics.Vectors.4.5.0
System.Runtime.CompilerServices.Unsafe.4.7.1
System.Memory.4.5.4
Microsoft.Extensions.Primitives.3.1.32
Microsoft.Extensions.Caching.Abstractions.3.1.32
Microsoft.Extensions.Configuration.Abstractions.3.1.32
Microsoft.Extensions.Configuration.3.1.32
Microsoft.Extensions.Configuration.Binder.3.1.32
Microsoft.Extensions.Options.3.1.32
Microsoft.Extensions.Caching.Memory.3.1.32
System.Collections.Immutable.1.7.1
```

Accept the License for each installation


License Acceptance

The following package(s) require that you accept their license terms before installing.

Microsoft.Extensions.Configuration Author(s): Microsoft
Apache-2.0

Microsoft.Extensions.Logging.Abstractions Author(s): Microsoft
Apache-2.0

Microsoft.EntityFrameworkCore.Analyzers Author(s): Microsoft
Apache-2.0

Microsoft.EntityFrameworkCore.Tools Author(s): Microsoft
Apache-2.0

Microsoft.EntityFrameworkCore.Abstractions Author(s): Microsoft
Apache-2.0

Microsoft.Extensions.Caching.Abstractions Author(s): Microsoft
Apache-2.0

Microsoft.Extensions.Configuration.Abstractions Author(s): Microsoft
Apache-2.0

By clicking "I Accept," you agree to the license terms for the package(s) listed above. If you do not agree to the license terms, click "I Decline."

After the packages are installed, in your NuGet Package Manager go to **Installed** to view your installed packages.

If visual studio is not displaying all the installed packages, then **Save your project** then **close** and **open Visual Studio**.

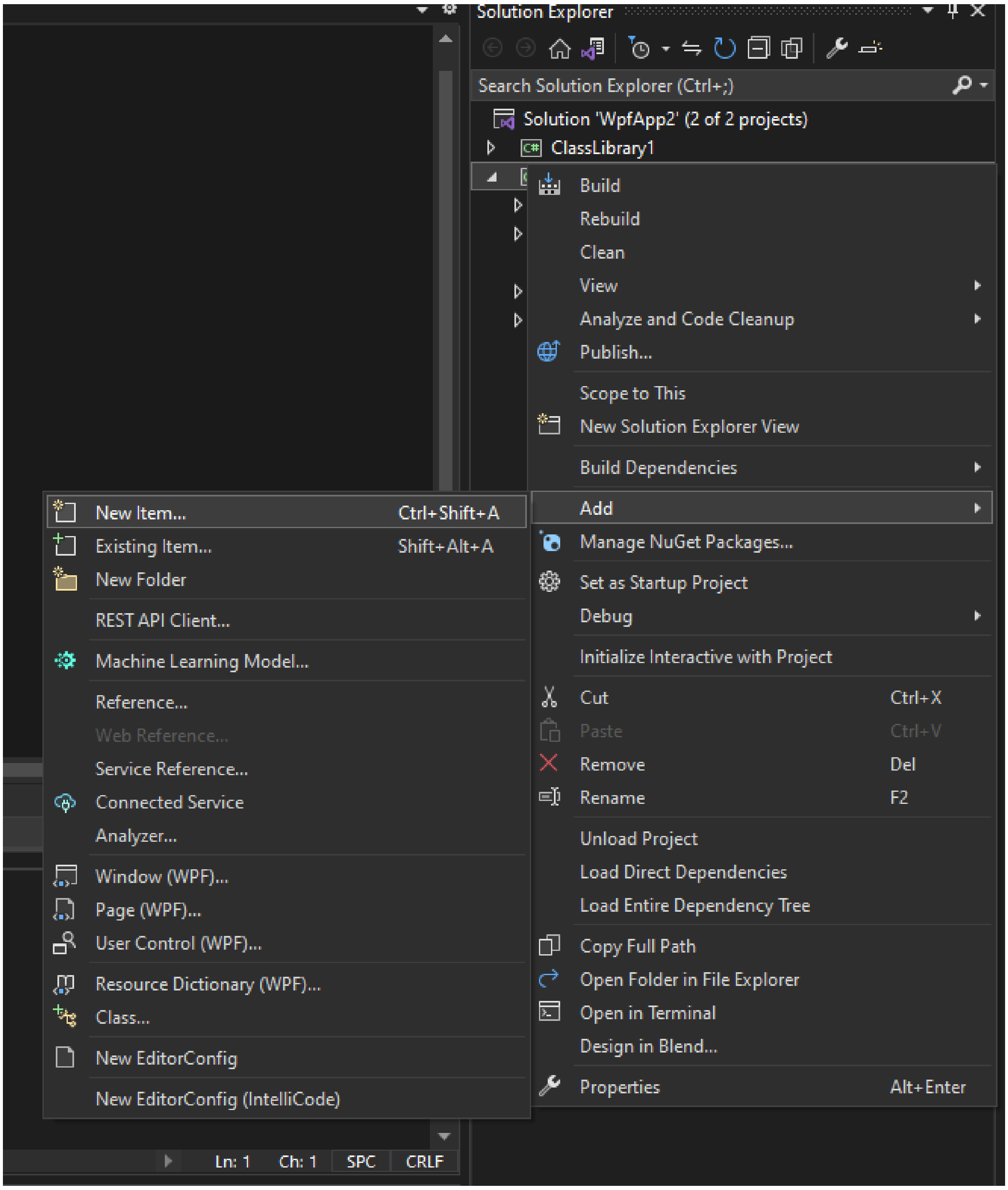
Now Open **Manage NuGet Packages…** -> and **refresh** your **Installed package**s

Now your project is setup in .NET Framework with a connected class library and all the required packages ready to perform SQL functions for data persistence

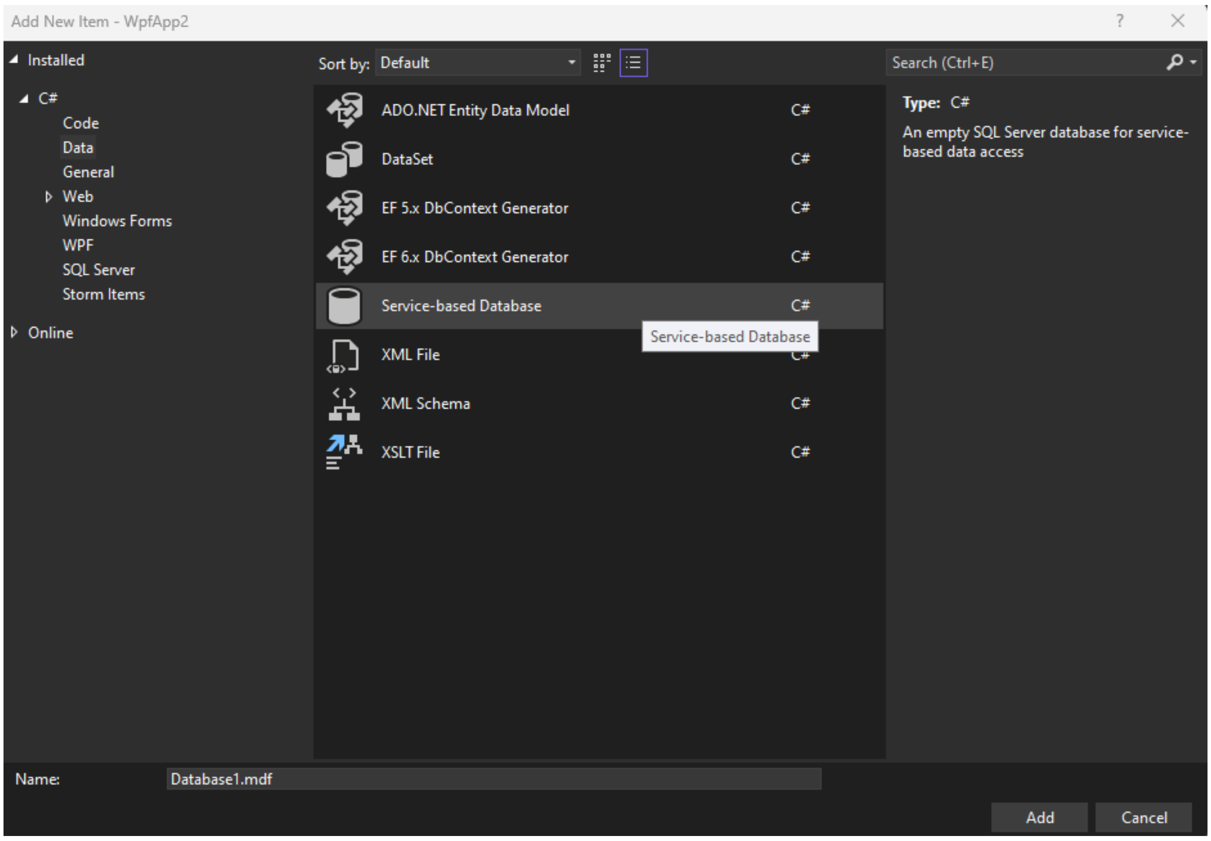
Creating the SQL Database

Let's move on to create the SQL Database where you'll be persisting your data to

Go to your project in the solution explorer and **Add** -> **New Item**

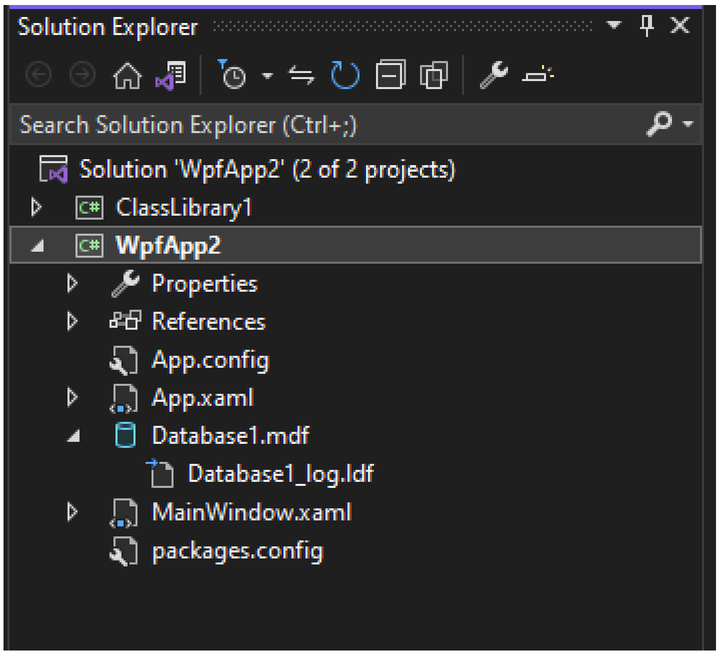


Under C# click on **Data** -> **Service Based Database**



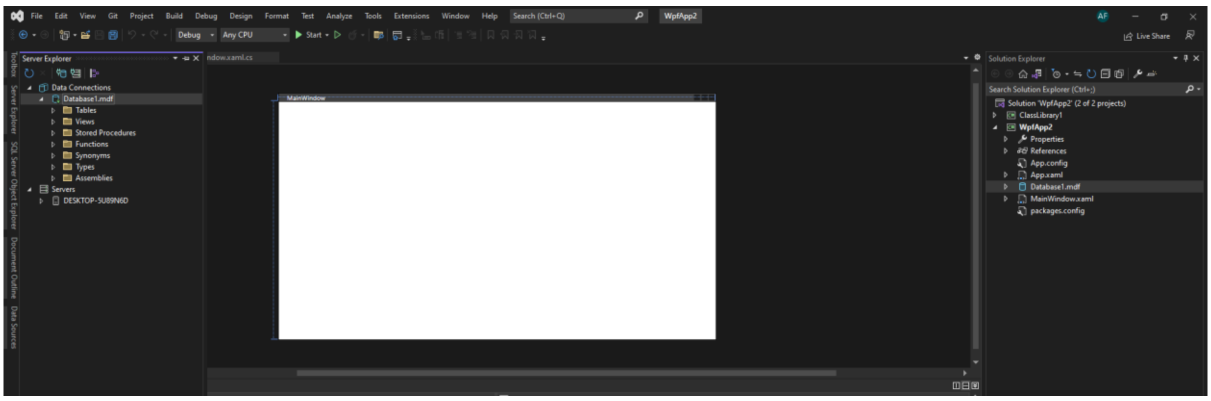
Change the name or leave it as default, then click **Add**

The database should then appear in your solution explorer as of type mdf like this

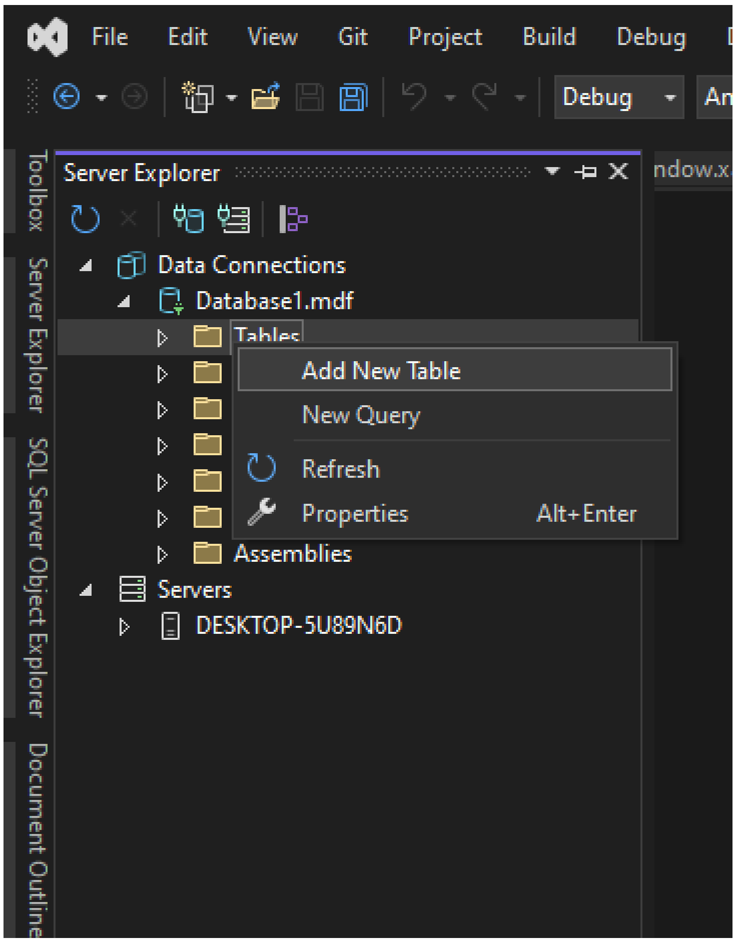


Creating a database table

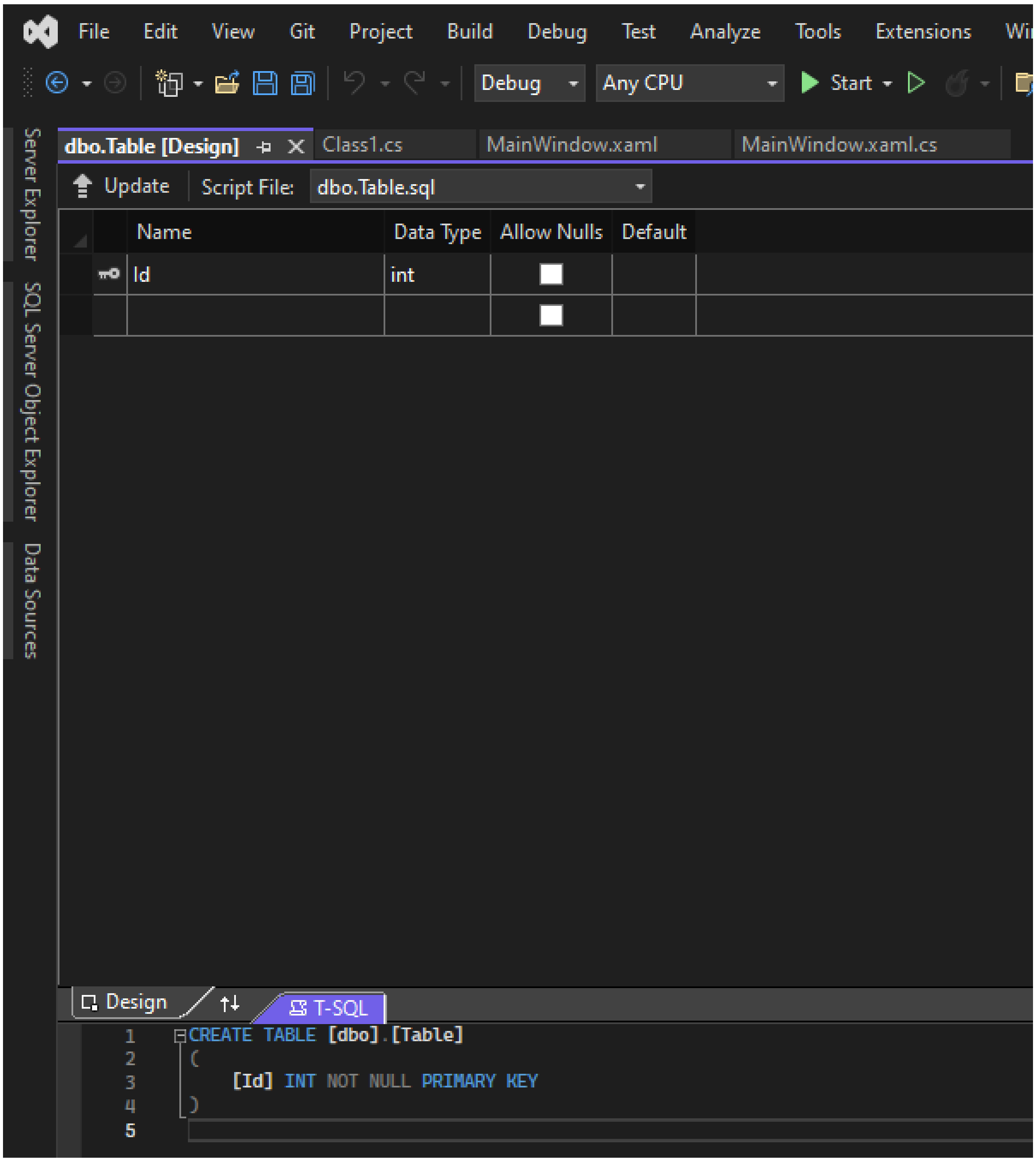
Now in the solution explorer **double click** on the **blue mdf file**. Your database should appear in the server explorer on the left under Data Connections



Now right click on **Tables** -> **Add New Table**



The Table Design window should appear with an area below to write T-SQL scripts

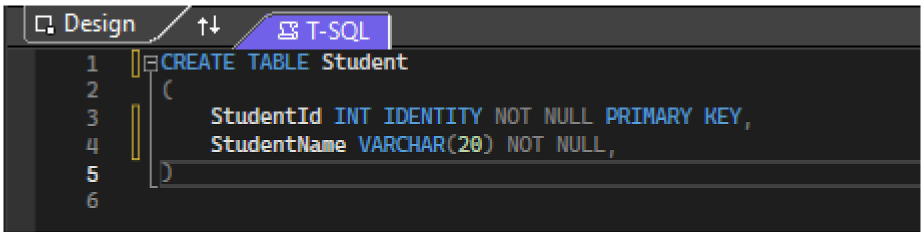


Name	Data Type	Allow Nulls	Default
Id	int	☐	
		☐	

```
CREATE TABLE [dbo].[Table]
(
    [Id] INT NOT NULL PRIMARY KEY
)
```

For an example database we are going to change the table name to Student with columns:

StudentID and StudentName

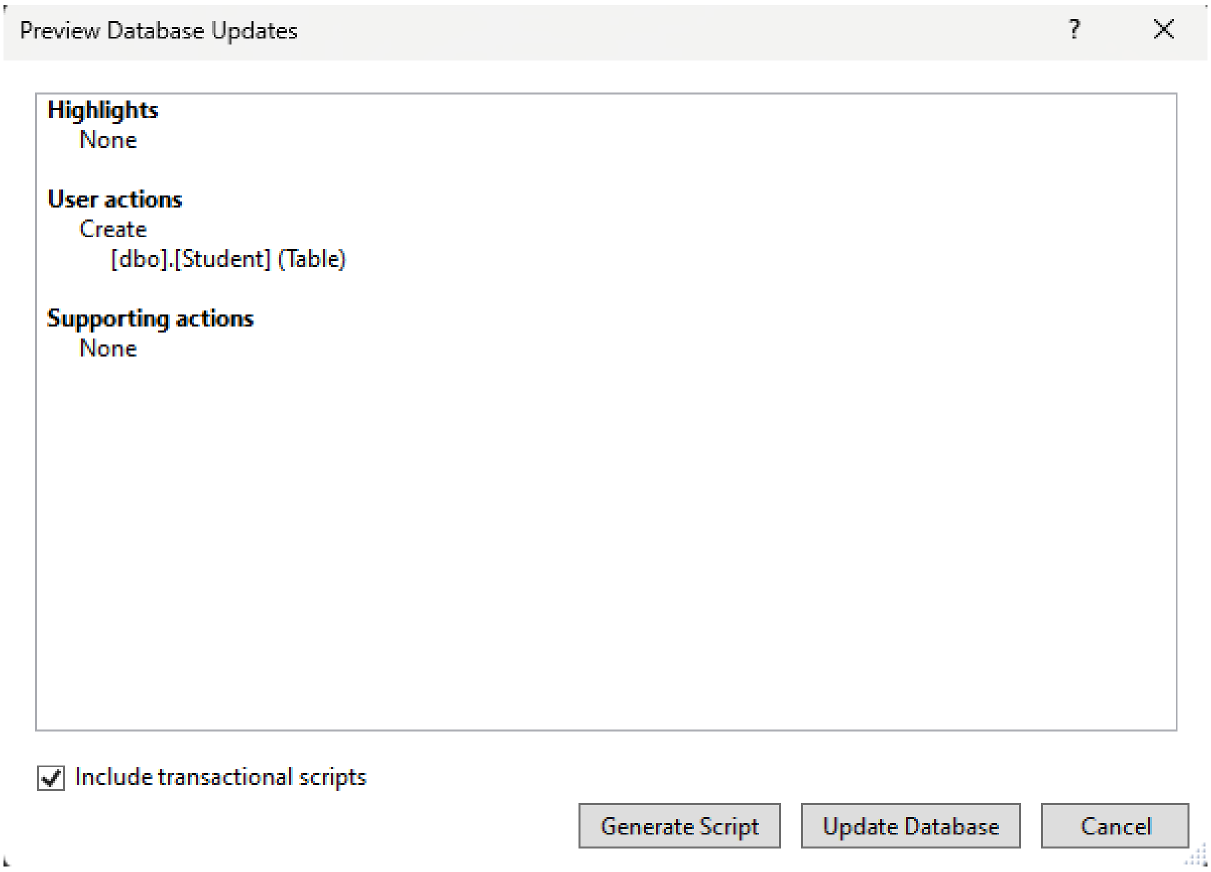


```
CREATE TABLE Student
(
    StudentId INT IDENTITY NOT NULL PRIMARY KEY,
    StudentName VARCHAR(20) NOT NULL,
)
```

Now **click enter**

It will **process the T-SQL script**

After the script is processed, navigate to the top just above the table and click **Update**


Preview Database Updates

Highlights
None

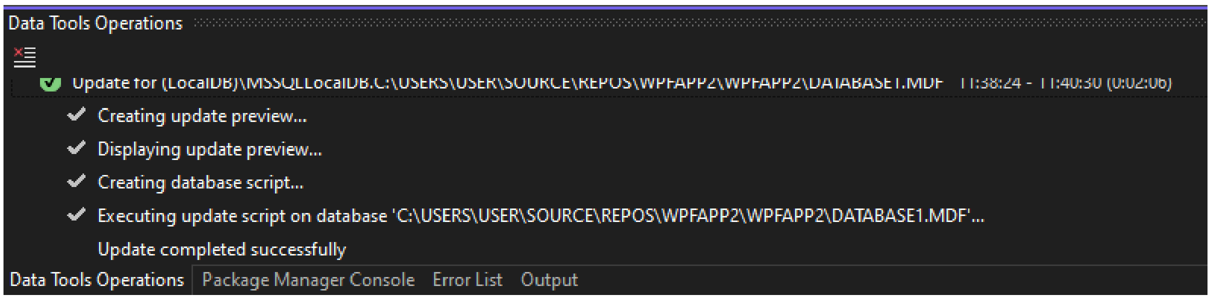
User actions
Create
[dbo].[Student] (Table)

Supporting actions
None

Include transactional scripts

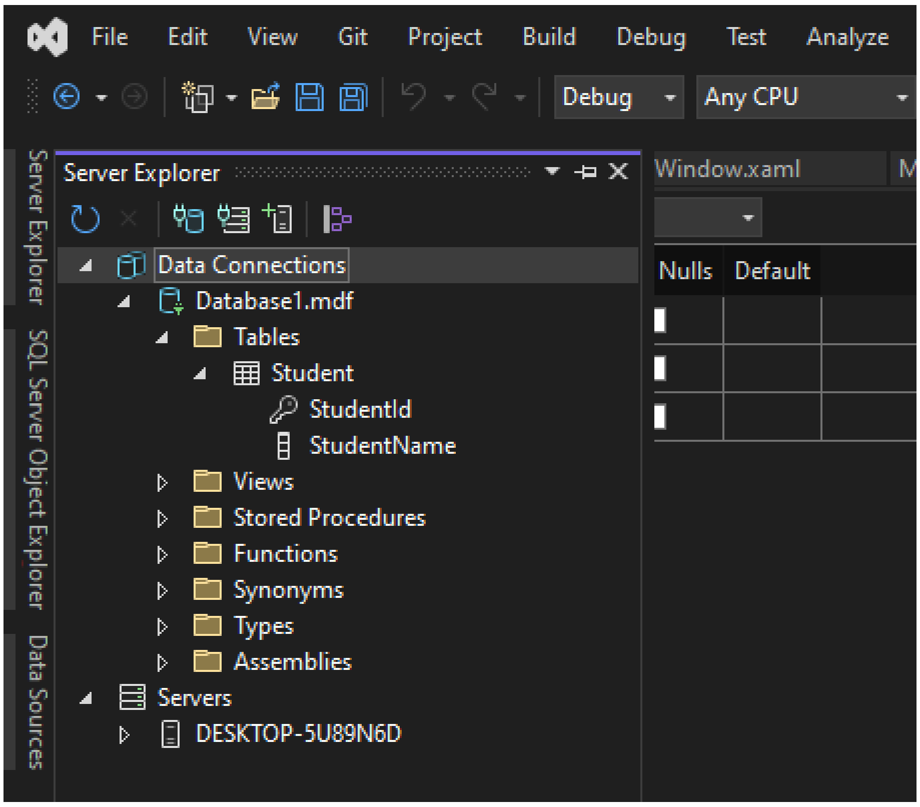
Make sure to **Include transactional scripts** and then click **Update Database**

Below the T-SQL in **Data Tools Operations** you should see your **Update completed successfully**



```
Update for (LocalDB)\MSSQLLocalDB.C:\USERS\USER\SOURCE\REPOS\WPFAPP2\WPFAPP2\DATABASE1.MDF   11:38:24 - 11:40:30 (0:02:06)
    ✓ Creating update preview...
    ✓ Displaying update preview...
    ✓ Creating database script...
    ✓ Executing update script on database 'C:\USERS\USER\SOURCE\REPOS\WPFAPP2\WPFAPP2\DATABASE1.MDF'...
      Update completed successfully
```

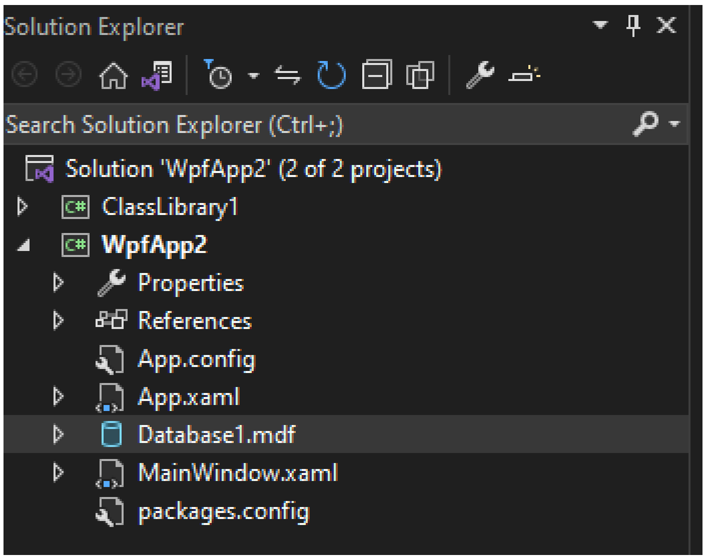
Now **open** your **Server Explorer** and **Refresh your Database and Tables**. Your database should now have the new Student Table added with the StudentId and StudentName columns

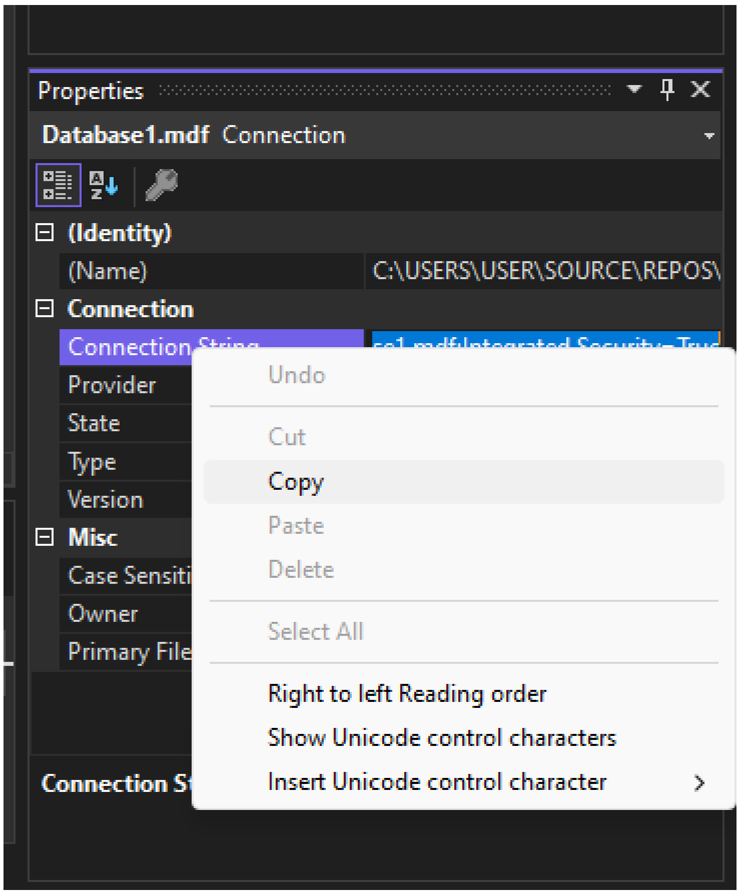


Now that our database is created, we now need a connection to the database.

Locating the database connection string

You can find your database connection string by double clicking on your [database name].mdf file in the Solution Explorer under Database properties

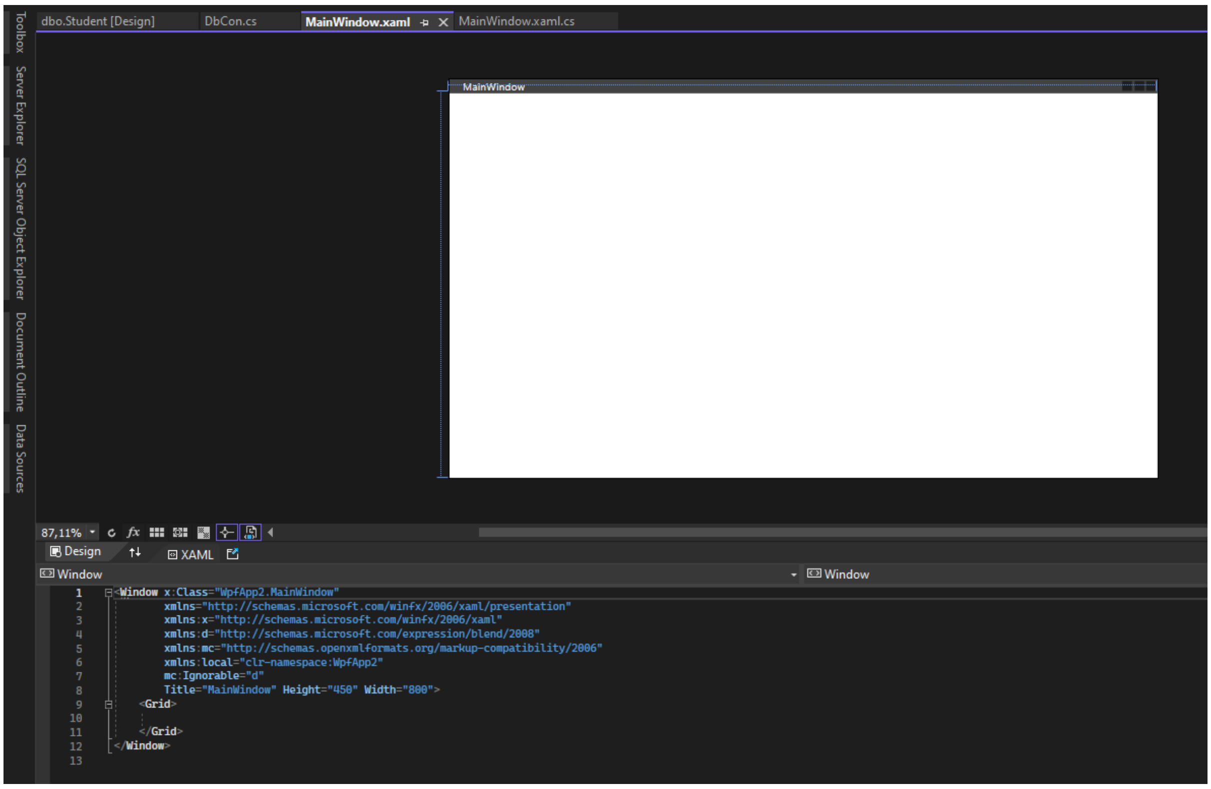




You will need this connection string later in your application

Creating the Form

Let’s move to the xaml design MainWidnow where we will create the Student form

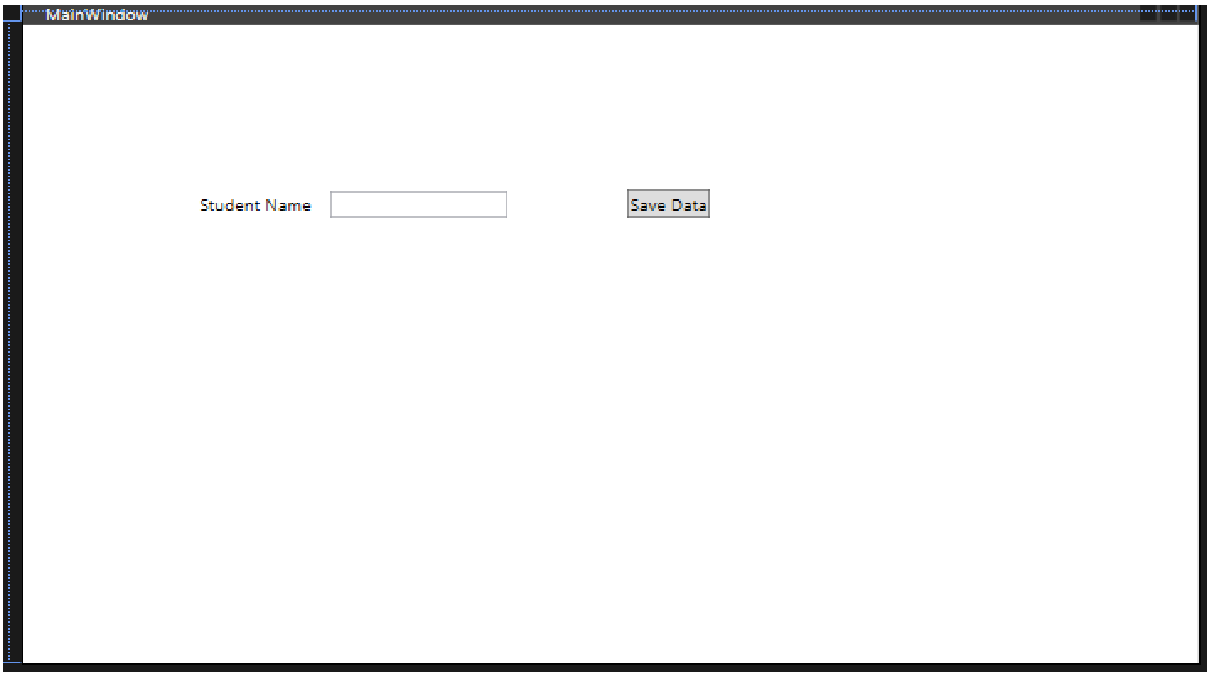


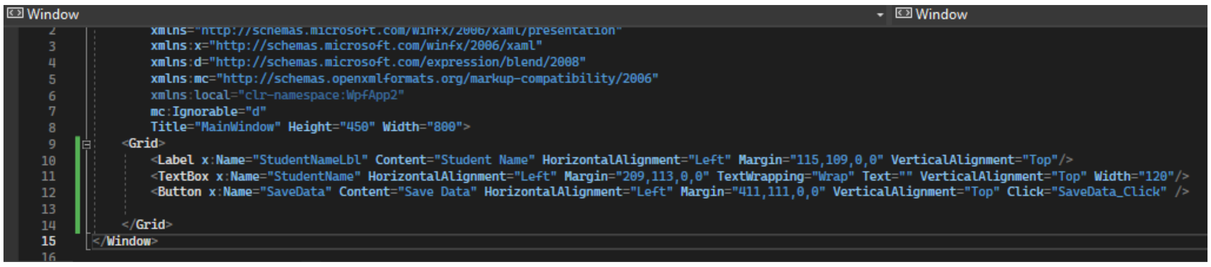
```
<Window x:Class="WpfApp2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApp2"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>

    </Grid>
</Window>
```

To continue the basic Student example

- Add a label for Student Name. Set the content=”Student Name”
- Add a Textbox alongside it and give it a name of x:Name=”StudentName”. Set the Text=””
- Add a button to save the data and double click on it to auto-generate an Onclick method

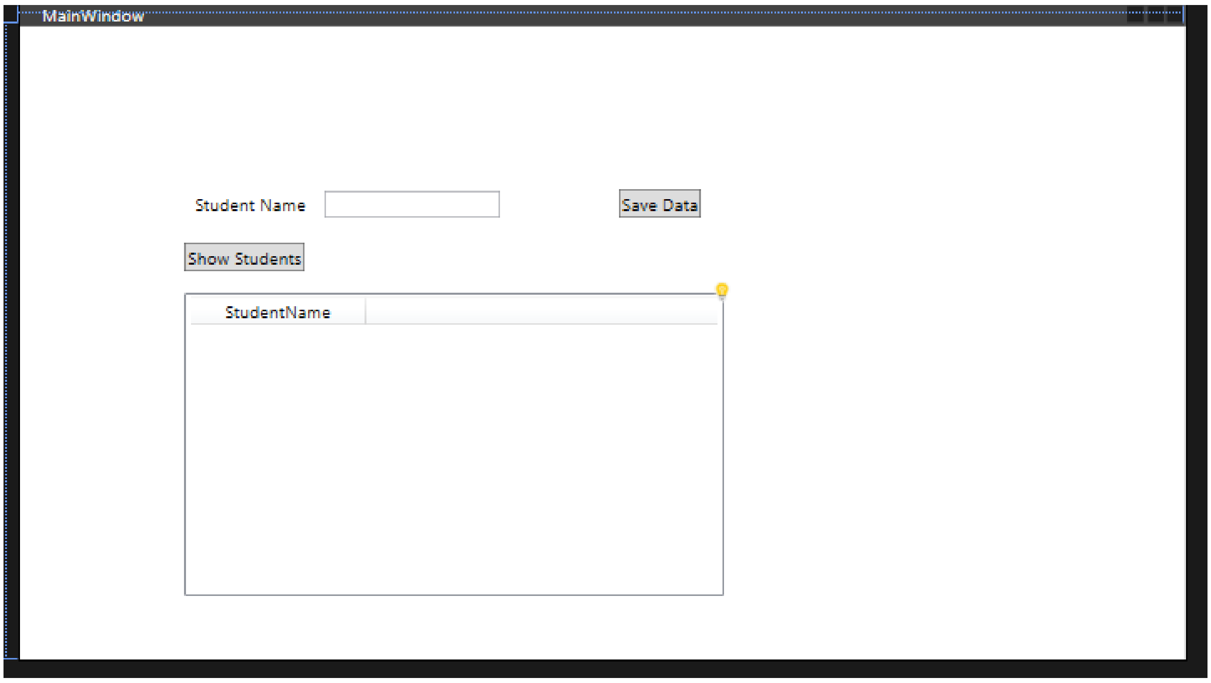
It should look something like this

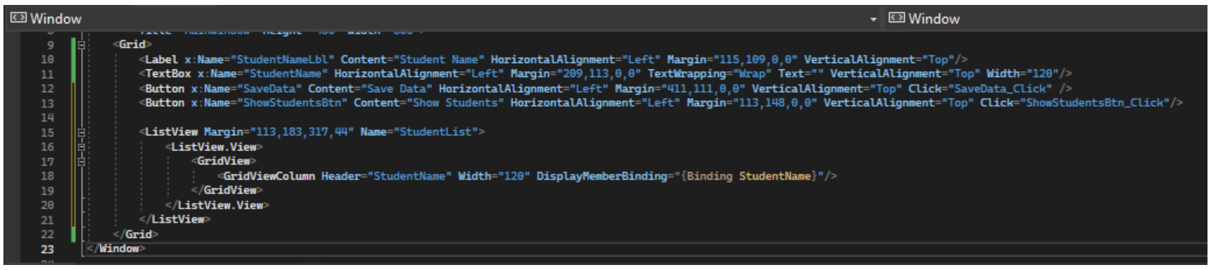




```
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApp2"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Label x:Name="StudentNameLbl" Content="Student Name" HorizontalAlignment="Left" Margin="115,109,0,0" VerticalAlignment="Top"/>
        <TextBox x:Name="StudentName" HorizontalAlignment="Left" Margin="209,113,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="120"/>
        <Button x:Name="SaveData" Content="Save Data" HorizontalAlignment="Left" Margin="411,111,0,0" VerticalAlignment="Top" Click="SaveData_Click" />

    </Grid>
</Window>
```

- Now let's **add a GridView and a button** to retrieve all the Students saved in the database like this.
- Set the ListView x:Name="StudentList"
- Set the button Content="Show Students" and double click it to auto-generate an Onclick method





```
    <Grid>
        <Label x:Name="StudentNameLbl" Content="Student Name" HorizontalAlignment="Left" Margin="115,109,0,0" VerticalAlignment="Top"/>
        <TextBox x:Name="StudentName" HorizontalAlignment="Left" Margin="209,113,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="120"/>
        <Button x:Name="SaveData" Content="Save Data" HorizontalAlignment="Left" Margin="411,111,0,0" VerticalAlignment="Top" Click="SaveData_Click" />
        <Button x:Name="ShowStudentsBtn" Content="Show Students" HorizontalAlignment="Left" Margin="113,148,0,0" VerticalAlignment="Top" Click="ShowStudentsBtn_Click"/>

        <ListView Margin="113,183,317,44" Name="StudentList">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="StudentName" Width="120" DisplayMemberBinding="{Binding StudentName}"/>
                </GridView>
            </ListView.View>
        </ListView>
    </Grid>
</Window>
```

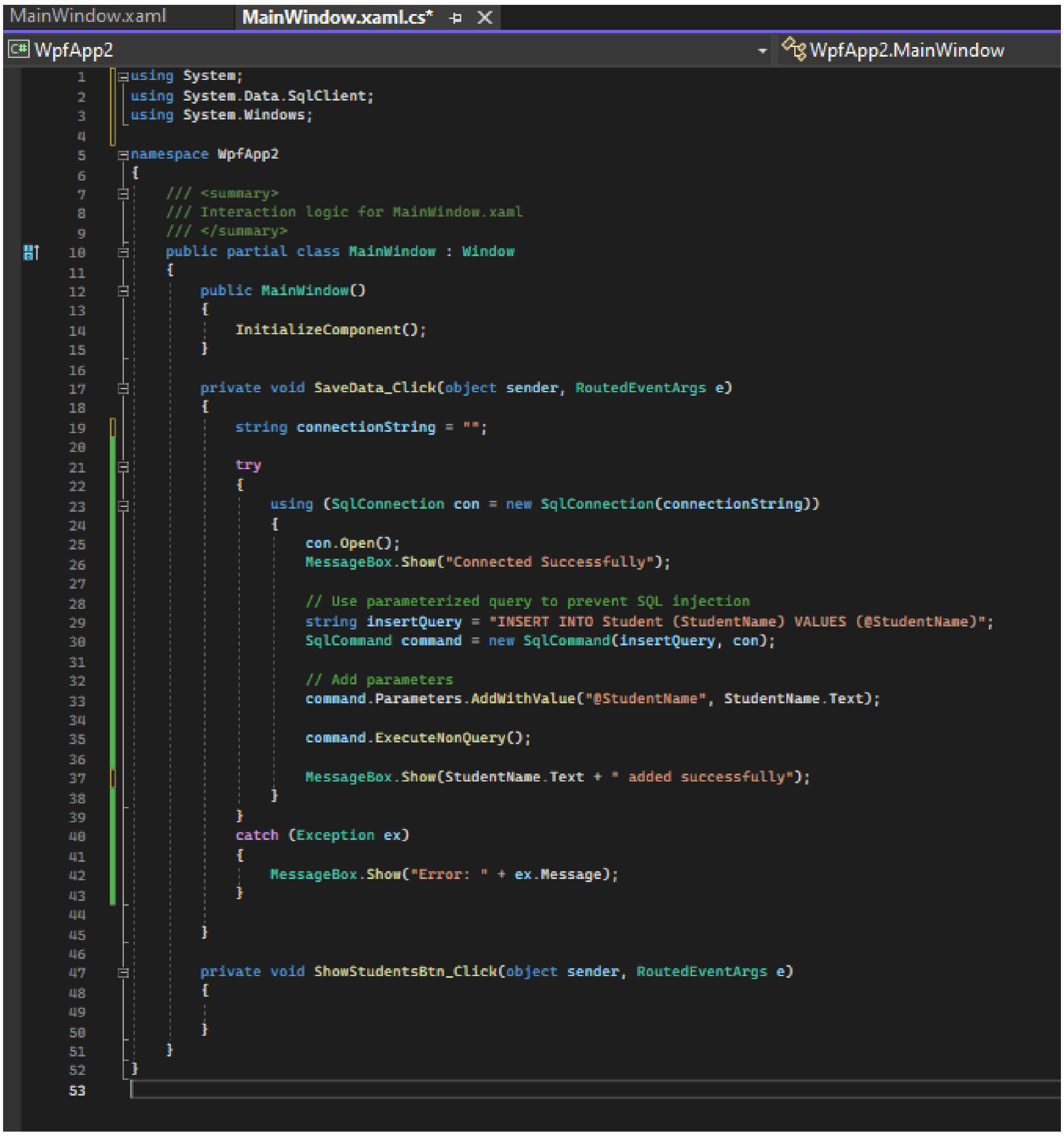
The student form is now setup.

From here you'll **INSERT** data into your database from the StudentName textbox when the Saved Data button is clicked. You can then click on the Show Students button to populate the GridView with a **SELECT** query showing all the records in the Students tables

We'll now be moving to the MainWindow.cs class

Writing to the database

Add the following code to within the SaveData Onclick method the functionality will be discussed



```
using System;
using System.Data.SqlClient;
using System.Windows;

namespace WpfApp2
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void SaveData_Click(object sender, RoutedEventArgs e)
        {
            string connectionString = "";

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    MessageBox.Show("Connected Successfully");

                    // Use parameterized query to prevent SQL injection
                    string insertQuery = "INSERT INTO Student (StudentName) VALUES (@StudentName)";
                    SqlCommand command = new SqlCommand(insertQuery, con);

                    // Add parameters
                    command.Parameters.AddWithValue("@StudentName", StudentName.Text);

                    command.ExecuteNonQuery();

                    MessageBox.Show(StudentName.Text + " added successfully");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }

        }

        private void ShowStudentsBtn_Click(object sender, RoutedEventArgs e)
        {

        }
    }
}
```

Code breakdown:

- In the SaveData_Click method we've created an empty string for now to store the database's connection string called connectionString
- We've then added a try-catch block to test the code, run it if there are no errors and catch any errors through a message box

In the try block

- We add a using clause to create a new connect to the database so that the connection can be disposed of properly
- We then open the connection and alert the user if the connection is successful

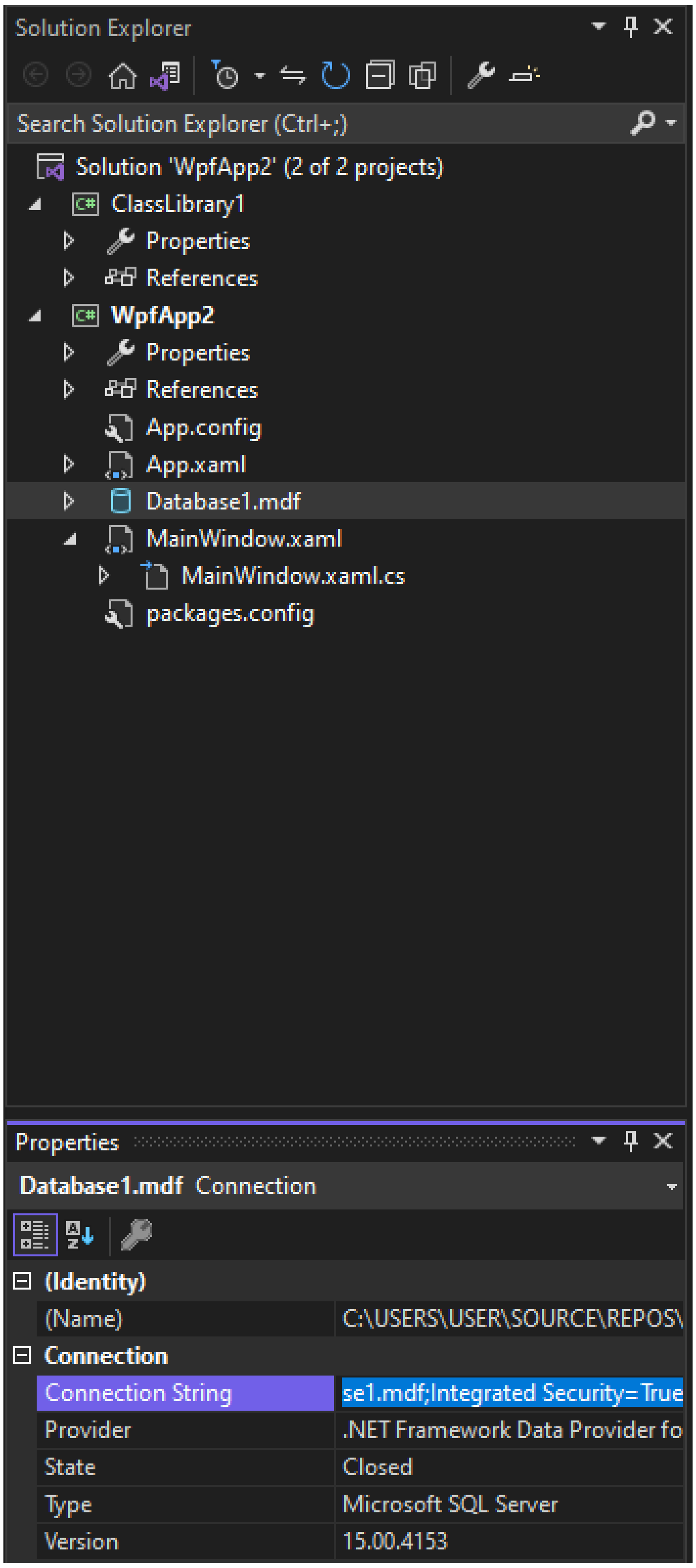
- Then we declare an INSERT query that'll insert the student name input using a StudentName Parameter which we define later
- The SqlCommand binds the query to the specific database through the database connection
- We then add a parameter for the student's name input that takes the input as StudentName.Text
- After setting up the command we execute the command to actually run it
- We then alert the user to show the input was added successfully

In the catch block

- The application will run the try block line-by-line and throw the appropriate error when its reached

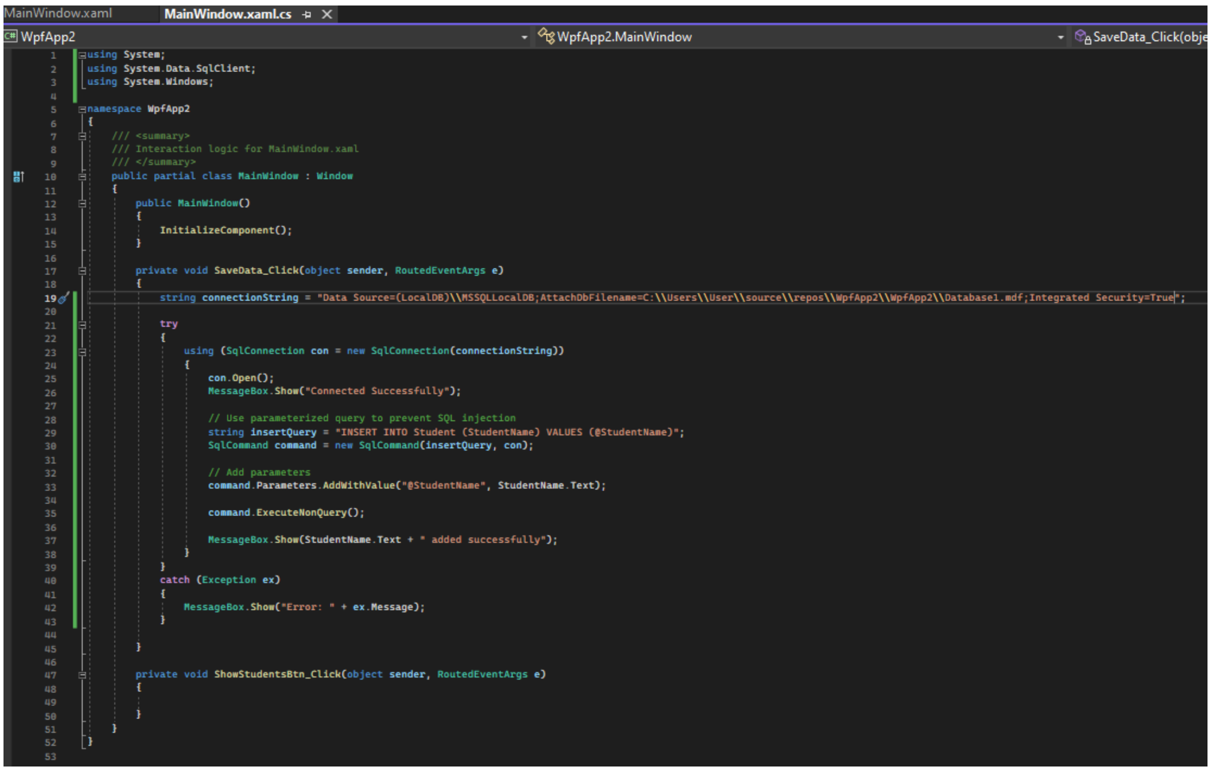
Adding the connection string

Remember we have to add the connection string. To do this double click on your database.mdf file



(Identity)	
(Name)	C:\USERS\USER\SOURCE\REPOS\
Connection	
Connection String	se1.mdf;Integrated Security=True
Provider	.NET Framework Data Provider fo
State	Closed
Type	Microsoft SQL Server
Version	15.00.4153

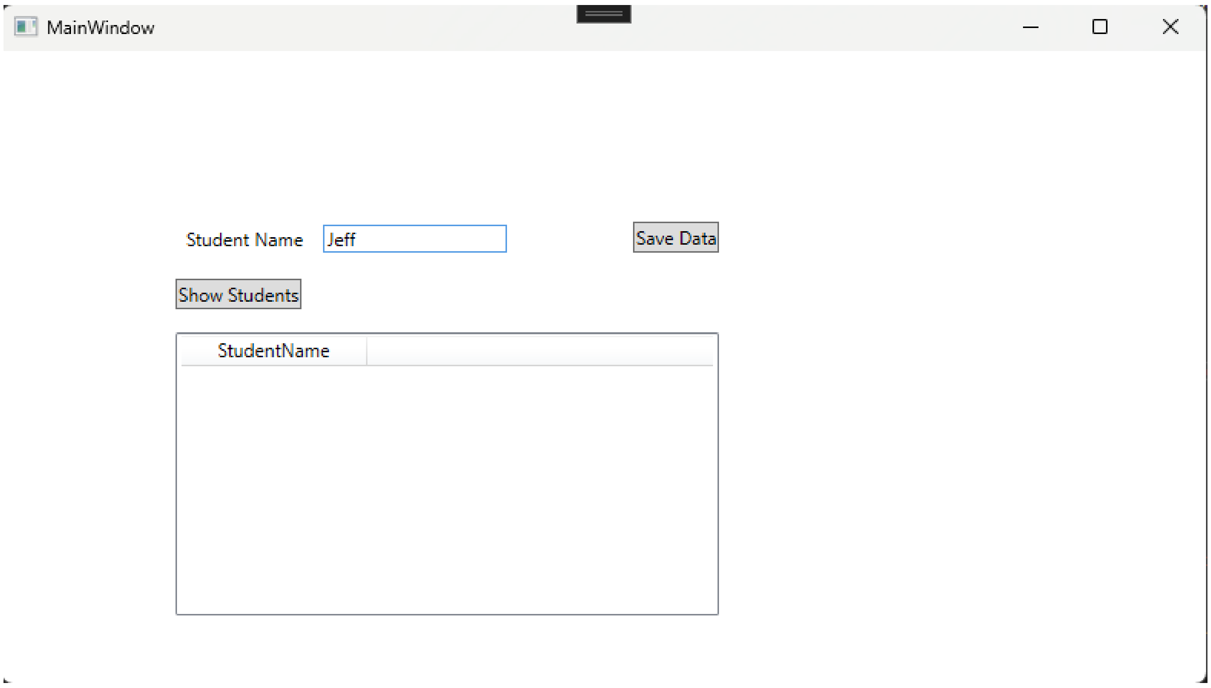
Copy the full connection string under database properties and paste it in the 2 quotation marks by your connectionString. It should look something like this

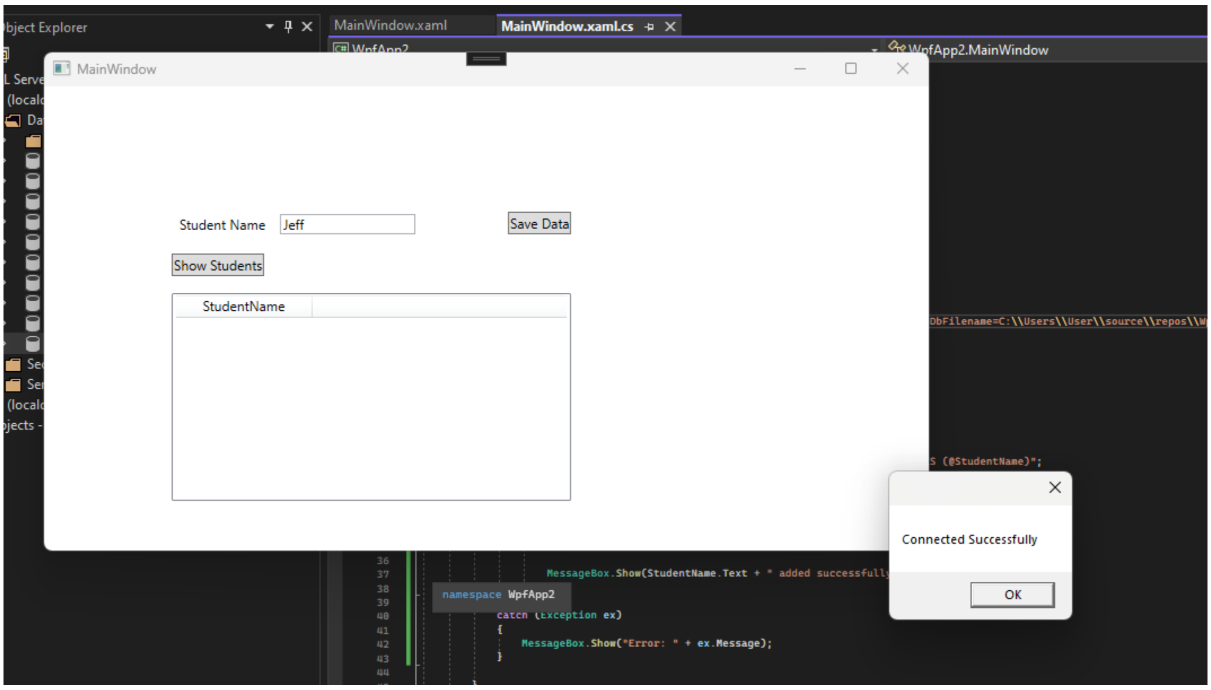


```csharp
using System;
using System.Data.SqlClient;
using System.Windows;

namespace WpfApp2
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void SaveData_Click(object sender, RoutedEventArgs e)
        {
            string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=C:\\Users\\User\\source\\repos\\WpfApp2\\WpfApp2\\Database1.mdf;Integrated Security=True";

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    MessageBox.Show("Connected Successfully");

                    // Use parameterized query to prevent SQL injection
                    string insertQuery = "INSERT INTO Student (StudentName) VALUES (@StudentName)";
                    SqlCommand command = new SqlCommand(insertQuery, con);

                    // Add parameters
                    command.Parameters.AddWithValue("@StudentName", StudentName.Text);

                    command.ExecuteNonQuery();

                    MessageBox.Show(StudentName.Text + " added successfully");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }

        }

        private void ShowStudentsBtn_Click(object sender, RoutedEventArgs e)
        {

        }
    }
}

```

Writing to the database: Test

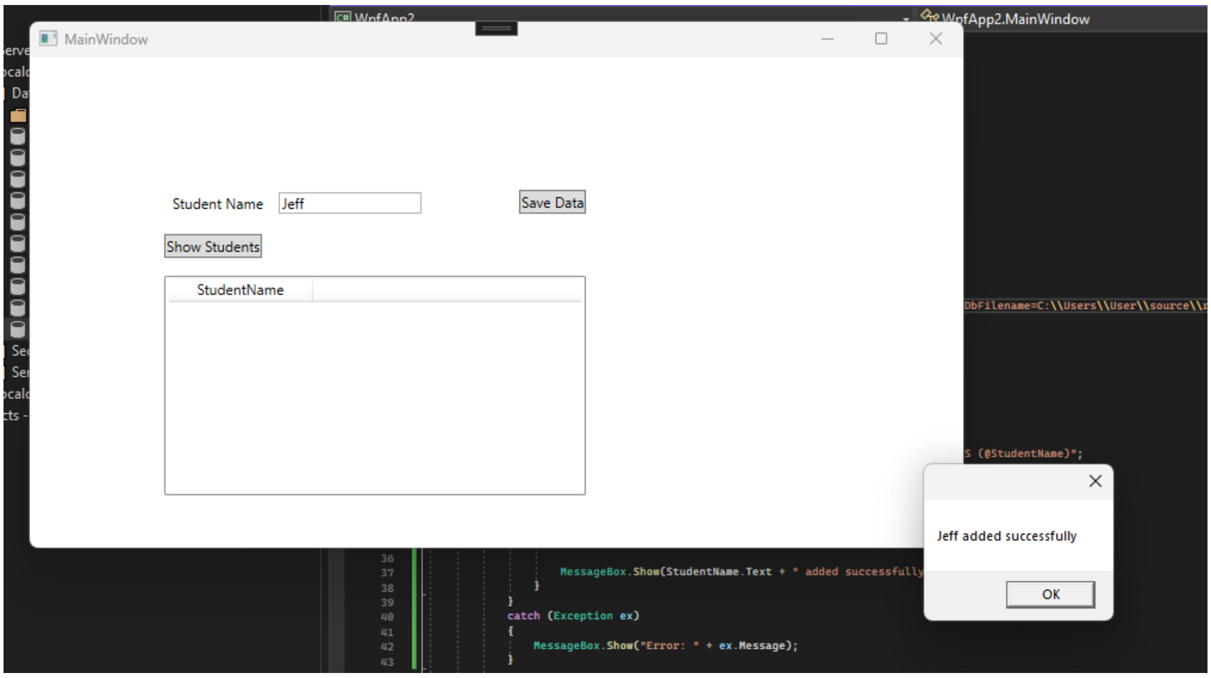
Now start the application and try entering some text into the textbox then click the Save Data button





Connected Successfully

```
                MessageBox.Show(StudentName.Text + " added successfully
namespace WpfApp2
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
```

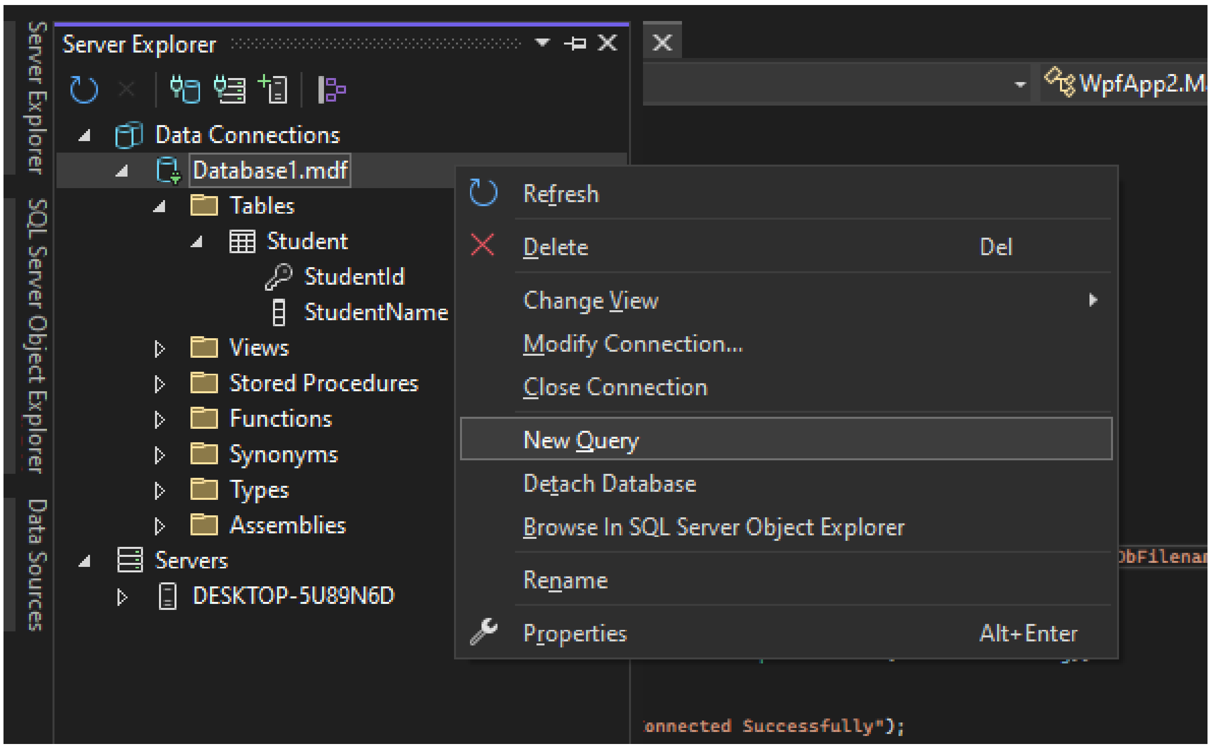


Jeff added successfully

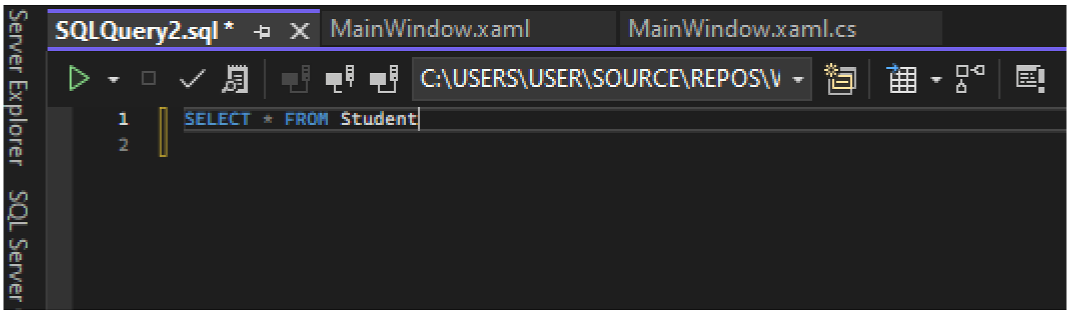
```
                    MessageBox.Show(StudentName.Text + " added successfully
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
```

This is good but let's query the database too see if Jeff was actually added.

To do this open your **Server Explorer ->** Right click on your database -> **New Query**

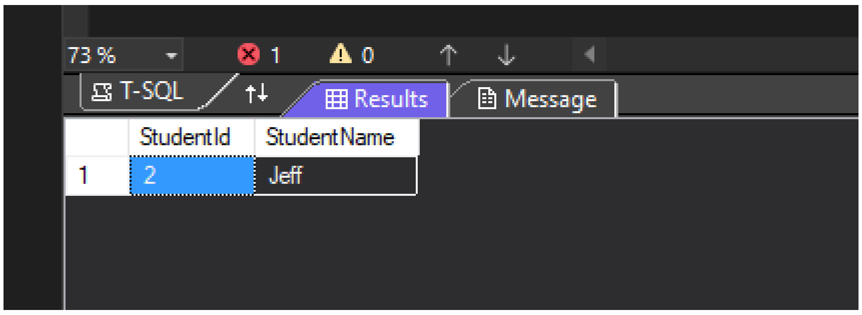


Write a SELECT query to pull all the records in the Students table



```
SELECT * FROM Student
```

Click the triangular green execute button to run the query



	StudentId	StudentName
1	2	Jeff

Congratulations! You now know how to insert data from your WPF app to your SQL database.

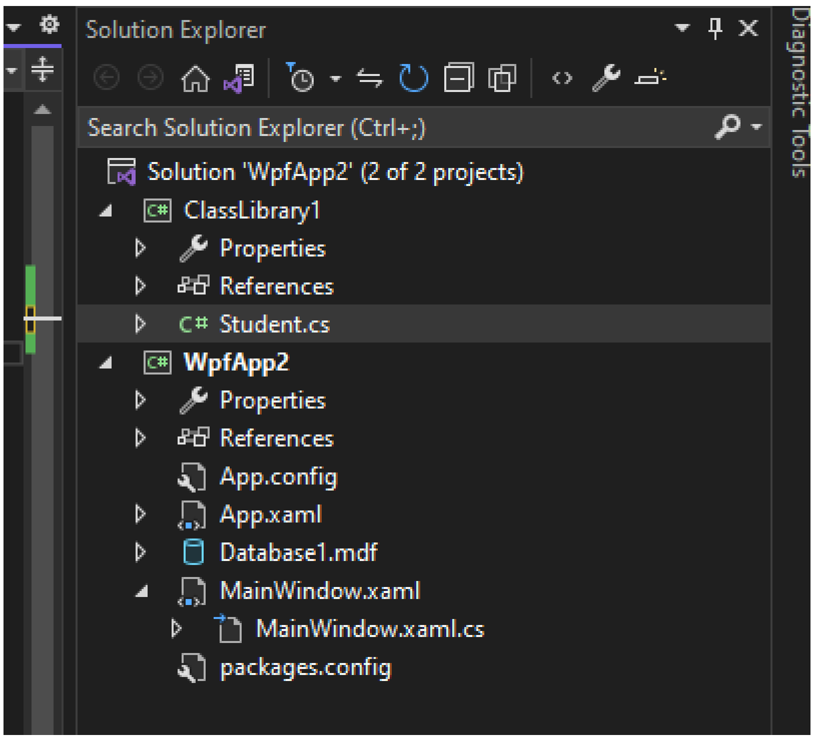
Now that we’ve added data to the database lets look at how to pull data from it and display the results in a GridView. You’ll now be working with the Show Students button Onclick method

In this case it’s the **ShowStudentsBtn_Click** method

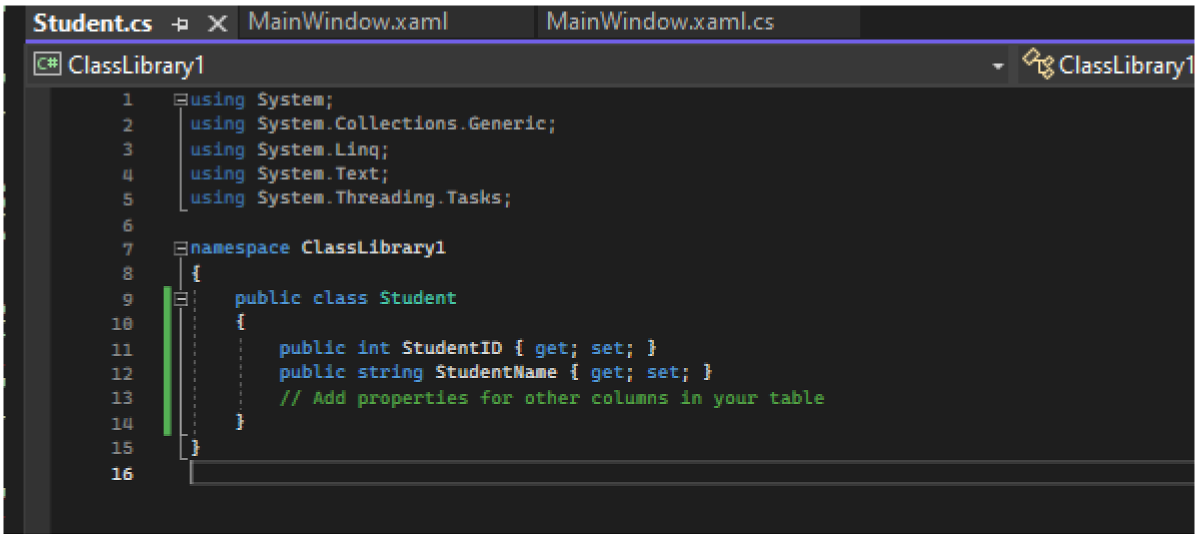
Now would be a good time to **Save All** your work

Adding the Student class to the class library

You'll need to use your class library now. Let's rename the default class (Should be Class1.cs) to Student.cs



In the student class add the two columns from the Student table in the database that we've earlier created StudentId and StudentName



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ClassLibrary1
{
    public class Student
    {
        public int StudentID { get; set; }
        public string StudentName { get; set; }
        // Add properties for other columns in your table
    }
}
```

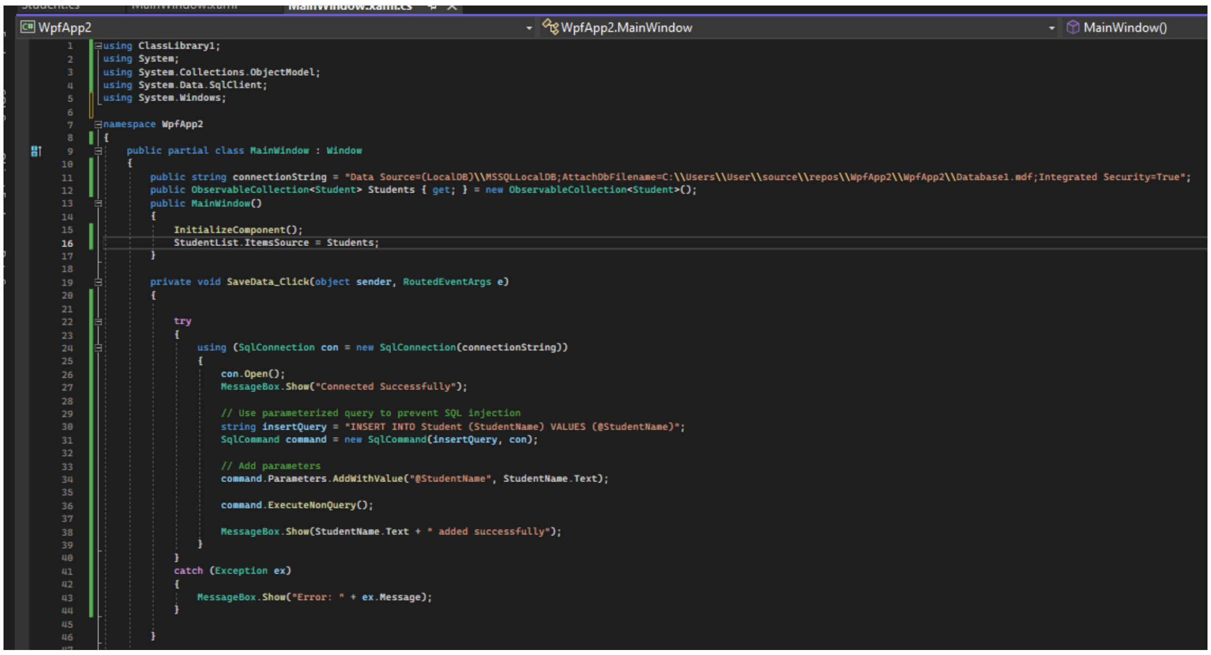
Cleaning up the code & adding a Collection of Students

Now let’s head back to MainWIndow.xaml.cs

First, let’s clean up our code by removing all the unnecessary usings

Since we are going to work with the database connection in 2 methods, we can optimize our code to declare the connection string outside the SaveData_Click method so we can use it anywhere in this class.

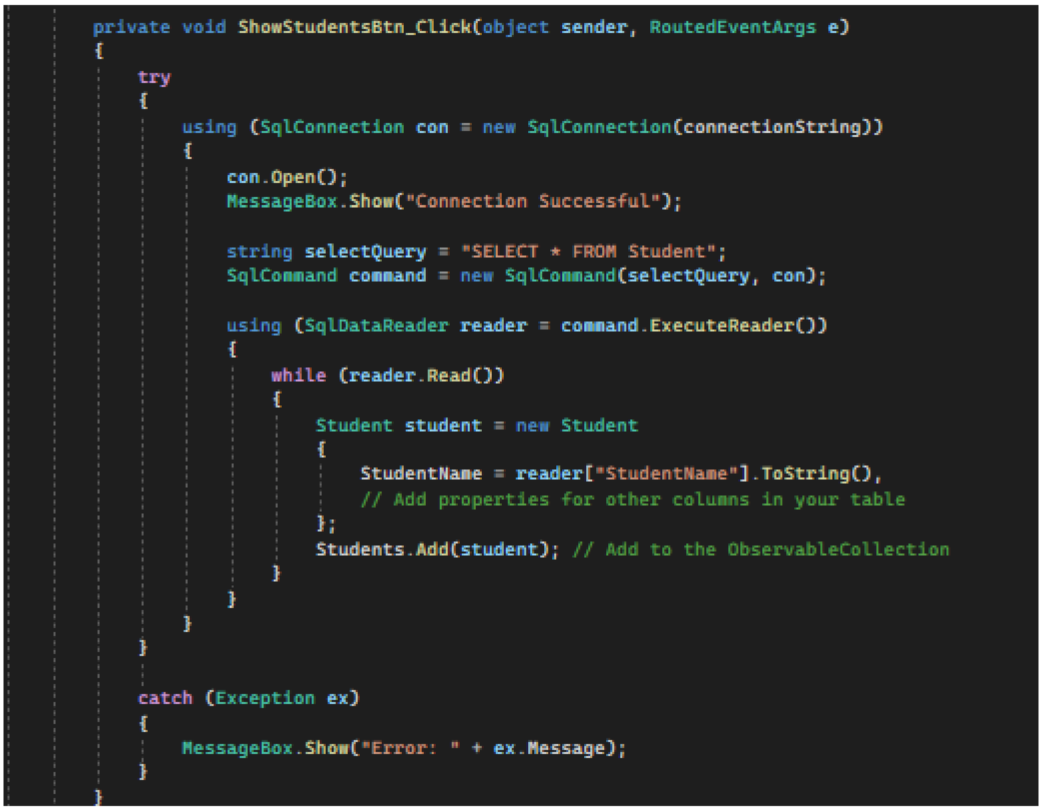
You can use any List or collection as long as it’s a data structure that you can manage properly but, in this case, let’s use an ObservableCollection of type Student that’ll serve as our GridViews Item Source



```
using ClassLibrary1;
using System;
using System.Collections.ObjectModel;
using System.Data.SqlClient;
using System.Windows;

namespace WpfApp2
{
    public partial class MainWindow : Window
    {
        public string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=C:\\Users\\User\\source\\repos\\WpfApp2\\WpfApp2\\Database1.mdf;Integrated Security=True";
        public ObservableCollection<Student> Students { get; } = new ObservableCollection<Student>();
        public MainWindow()
        {
            InitializeComponent();
            StudentList.ItemsSource = Students;
        }

        private void SaveData_Click(object sender, RoutedEventArgs e)
        {

            try
            {
                using (SqlConnection con = new SqlConnection(connectionString))
                {
                    con.Open();
                    MessageBox.Show("Connected Successfully");

                    // Use parameterized query to prevent SQL injection
                    string insertQuery = "INSERT INTO Student (StudentName) VALUES (@StudentName)";
                    SqlCommand command = new SqlCommand(insertQuery, con);

                    // Add parameters
                    command.Parameters.AddWithValue("@StudentName", StudentName.Text);

                    command.ExecuteNonQuery();

                    MessageBox.Show(StudentName.Text + " added successfully");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }

        }
```

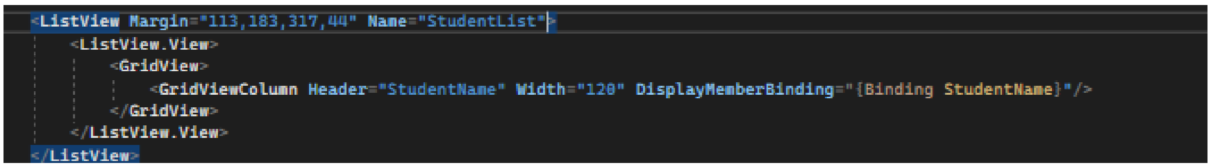
Retrieving data from the database and displaying it in a list

Add the following code to your ShowStudentsBtn_Click method and let's unpack it in the code breakdown



```
private void ShowStudentsBtn_Click(object sender, RoutedEventArgs e)
{
    try
    {
        using (SqlConnection con = new SqlConnection(connectionString))
        {
            con.Open();
            MessageBox.Show("Connection Successful");

            string selectQuery = "SELECT * FROM Student";
            SqlCommand command = new SqlCommand(selectQuery, con);

            using (SqlDataReader reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    Student student = new Student
                    {
                        StudentName = reader["StudentName"].ToString(),
                        // Add properties for other columns in your table
                    };
                    Students.Add(student); // Add to the ObservableCollection
                }
            }
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show("Error: " + ex.Message);
    }
}
```

Let's pop up the GridView we earlier created



```
<ListView Margin="113,183,317,44" Name="StudentList">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="StudentName" Width="120" DisplayMemberBinding="{Binding StudentName}"/>
        </GridView>
    </ListView.View>
</ListView>
```

Notice how the DisplayMemberBinding is set to a Binding of StudentName

Code Breakdown:

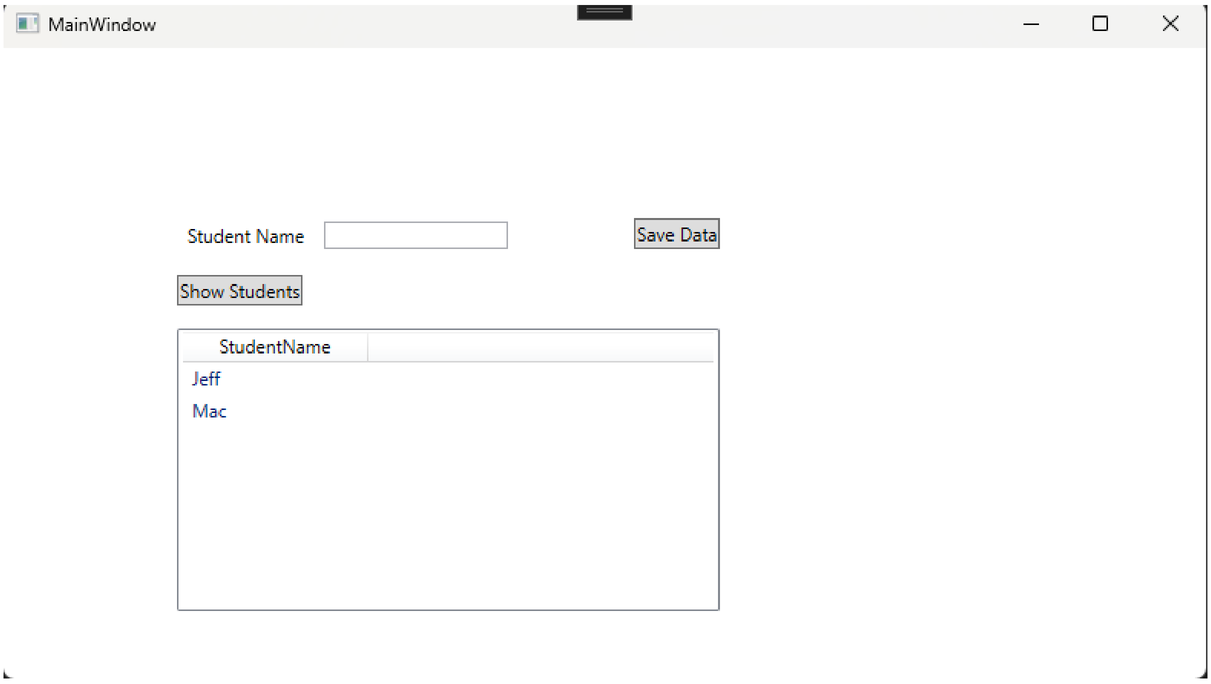
- We establish a database connection and open it.
- We create a SQL query (selectQuery) to retrieve all data from the Student table.
- We create a SqlCommand object and execute the query.
- We use a SqlDataReader to read the result set row by row.
- For each row, we create a Student object and populate its properties with data from the database.
- The Student object is then added to the Students collection (an ObservableCollection<Student>).
- When you run the application and click the "Show Students" button, the data retrieved from the database is stored in the Students collection.

- The ListView is configured to display this data in the GridView columns, with each row representing a Student object.
- The DisplayMemberBinding property of the GridViewColumn specifies which property of the student object to display in that column.

Retrieving data from the database and displaying it in a list: Test

Run the application and see if the data displays all the students you've added

It should display something like when you click the Show students button


MainWindow
Student Name
Save Data
Show Students
StudentName
Jeff
Mac