Cairo University
Faculty of Engineering
Computer Engineering Dept.
CMPN202/CMP2020

Fall 2021

LAB #4
Accessing SQL Server database from C# application

Objectives

After this lab, the student should be able to use Visual Studio to:

- Create simple windows application using C#.
- Write scripts to create database, create tables, or fill tables data.
- Connect C# application to a database.
- Execute simple SQL queries on the database by calling direct SQL statements in C# application.

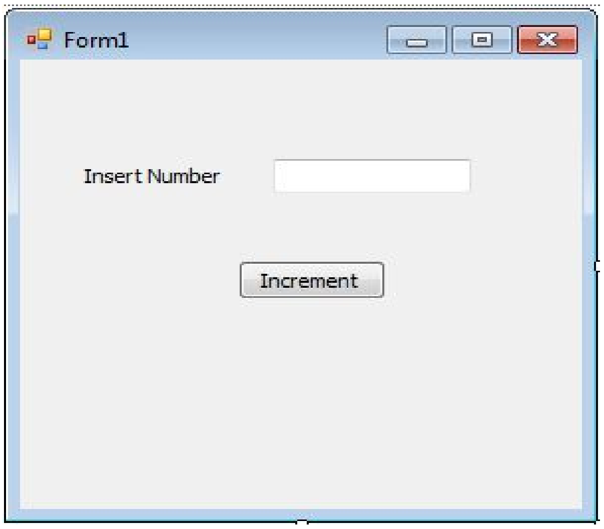
Lab Outline

I. Create a windows application

Creating Simple Windows Application

1. Open **Microsoft Visual Studio 2012** , From **File** select **New** then **Project**.
2. Select from **Visual C#** , **Windows Forms Application.**
3. The application creates a default form for you. And you can add more forms if you want by right-clicking on your project in the Solution Explorer and selecting **Add Windows Form** or **Add New Item Windows Form.** (But we will use the default form created "Suppliers")
4. Add **GUI elements:** Label, Textbox and Button to the form. This is done from the **Toolbox** (If it doesn't appear, choose it from view) and choose Textbox, Button and Label and drag them onto the Form Designer. You can **right click** any GUI element and modify its name, text, color, etc. from **Properties**. When the user clicks the Button "Increment", we want to increment the value entered in the textbox and display the new value instead. This can be done by capturing the onClick event of the Button by adding an event handler (function) to the onClick event of the button. Inside this function, we will add one (1) to the value entered in textbox and show the result in the same text box.

   Your form will look like this



5. Add **event handler** (function that will be executed when the button is clicked) for the button. This can be added by double clicking the button. You can also use the Events pane from the Properties Window.

```
 int value=0;
//TryParse in order not to prevent exception if non-integer value was entered
if (int.TryParse(textbox.Text, out value))
{
    value = value + 1;
    //set the result to the text box
    textbox.Text = value.ToString();
}
```

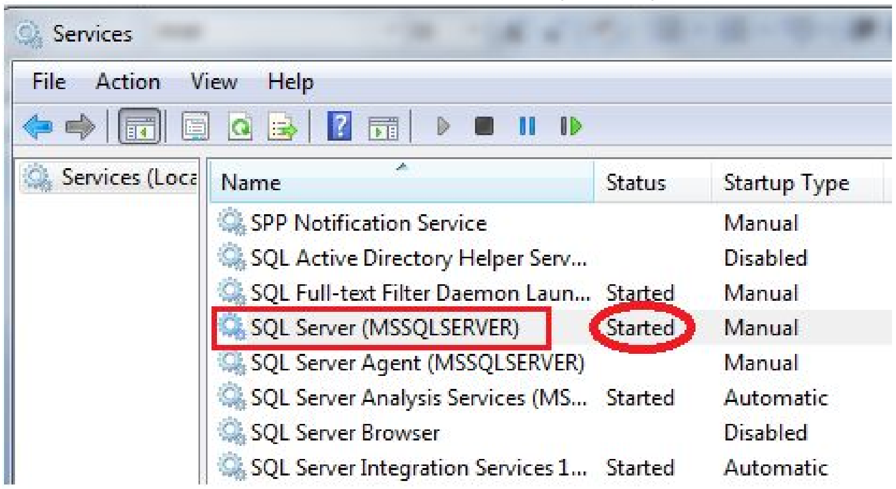
6. Run the application **(CTRL+F5**).

II. Create a data connection to a SQL Server database

Before you create a connection to a database, you should:

A. Make sure SQL server is running

1. Click Windows Start Button and type "**local services**". Select "**view local service**"
2. Make sure that "**SQL Server**" is "**<u>Started</u>**". If not, right-click and start it. You can also make it start automatically with Windows through its **Properties.**
   Note: Here server name is **MSSQLSERVER**, you may have a different server name.



Name	Status	Startup Type
SPP Notification Service		Manual
SQL Active Directory Helper Serv...		Disabled
SQL Full-text Filter Daemon Laun...	Started	Manual
SQL Server (MSSQLSERVER)	Started	Manual
SQL Server Agent (MSSQLSERVER)		Manual
SQL Server Analysis Services (MS...	Started	Automatic
SQL Server Browser		Disabled
SQL Server Integration Services 1...	Started	Automatic

B. Make sure your database is created

You can create and fill a database either through **graphical interface (Management Studio)** or by **running a script.**

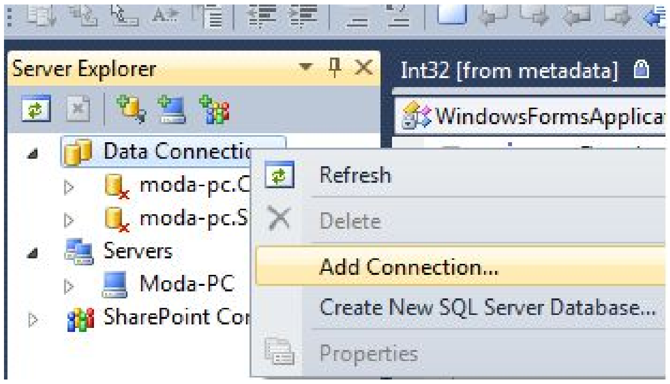
In this lab, you are given an SQL script that creates and fills the database (It is a set of SQL statements that creates database and table and insert tuples in tables).

1. Open **SQL Server Management Studio** and connect to the server (as in previous lab)
2. From **File** → **Open** → Browse to open the SQL script **SPJ_DB_Lab4.sql**. The script is provided with the lab material in folder **DB**.
3. Execute the script (click **Execute**)
4. This script creates a database **SPJ_DB_Lab4**, creates database tables, and fills tables with some sample data.

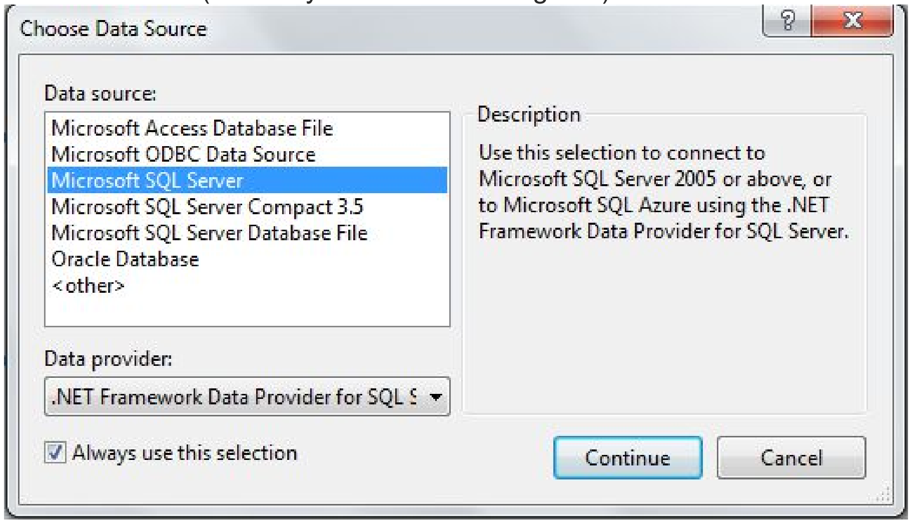
C. Create a database connection

Go back to your windows application in **Visual Studio** then:

1. In **Server Explorer/Database Explorer** right-click **Data Connections, select Add Connection.**

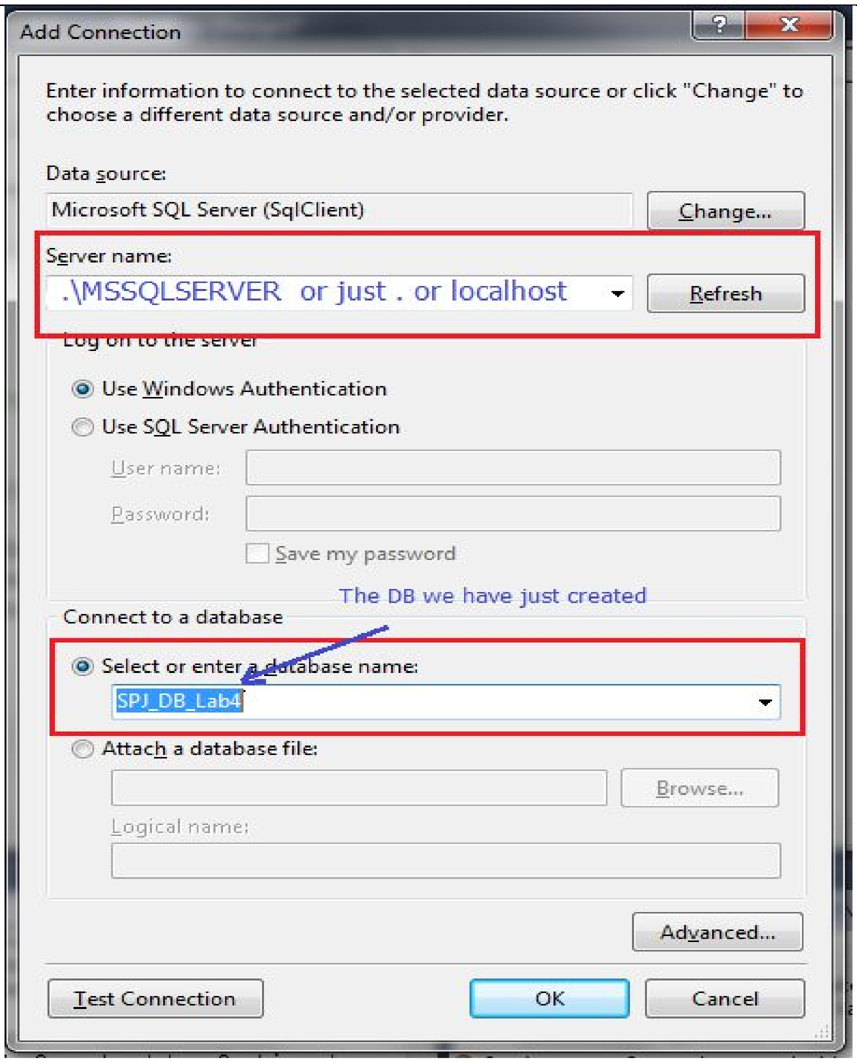


2. In the **Choose Data Source** dialog box, select **Microsoft SQL Server**, and then click **Continue**. (You may not see this dialog box)


Choose Data Source
Data source:
Microsoft Access Database File
Microsoft ODBC Data Source
Microsoft SQL Server
Microsoft SQL Server Compact 3.5
Microsoft SQL Server Database File
Oracle Database
<other>
Description
Data provider:
.NET Framework Data Provider for SQL S
Always use this selection
Continue
Cancel

Use this selection to connect to Microsoft SQL Server 2005 or above, or to Microsoft SQL Azure using the .NET Framework Data Provider for SQL Server.

3. Select a server name from the drop-down list, or type the name of the server where the database you want to access is located. (**see next figure**)
4. Based on the requirements of your database or application, select either **Windows Authentication** or use a specific user name and password to log on to the SQL Server (**SQL Server Authentication**).
5. Select the database you want to connect to from the drop-down list. (select SPJ_DB_Lab4)
6. Click **OK**.


Add Connection
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 (SqlClient)
Change...
Server name:
.\MSSQLSERVER or just . or localhost
Refresh
Log on to the server
Use Windows Authentication
Use SQL Server Authentication
User name:
Password:
Save my password
The DB we have just created
Connect to a database
Select or enter a database name:
SPJ_DB_Lab4
Attach a database file:
Browse...
Logical name:
Advanced...
Test Connection
OK
Cancel

III. Get the connection string

Steps

1. In **Server Explorer/Database Explorer**, right click on the database connection you created and choose **Properties**
2. In the Properties list you will find a property called **"Connection String"(**)**

The **Connection string** specifies information about a data source, the way of connecting to it and other attributes for connection such as security.

**** Store the connection string as we will be using it in next steps**

Now, Open the C# application provided with the lab. You will find it under <u>Code\DBapplication</u>

To deal with the DB, the application code makes use of three classes; **SqlConnection**, **SQLCommand** and **SqlDataReader** which are in **System.Data.SqlClient** namespace and can be included by writing the statement: `using System.Data.SqlClient;` at the beginning part of your .cs file (see **DBManager.cs** code file)

IV. Open the connection to the database

Code **(see file DBManager.cs)**

```
// SqlConnection class is used to open a connection to a database.
string DB_Connection_String = @"Data Source=.;Initial
Catalog=SPJ_DB_Lab4;Integrated Security=True";

SqlConnection myConnection = new SqlConnection(DB_Connection_String);
try
{
    myConnection.Open(); //Open a connection with the DB
    //This is just used to write to Console for debugging purposes
    Console.WriteLine("Successfully connected to the database!");
    //just for illustration when the database is opened,
    // this should not be shown in GUI to the user
    MessageBox.Show("Successfully connected to the database!");
}
catch (Exception e)
{
    Console.WriteLine("The DB connection to :" + DB_Connection_String + " failed!");
    MessageBox.Show("An error occurred while connecting to the database!");
}
```

Question : What are the problems with writing connection string in the code???

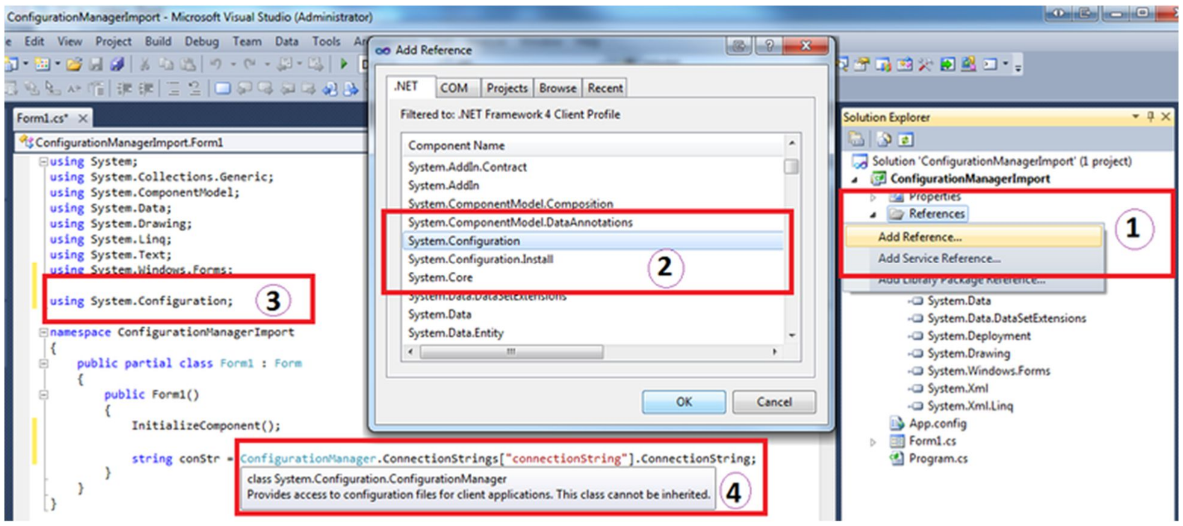
Connection string should be read from a **configuration file** so copy its value and add it to **App.config**.

To add an **application configuration** file to your C# project [1]

1.On the menu bar, choose **Project**, **Add New Item**.
The **Add New Item** dialog box appears.
2.Expand **Installed**, expand **Visual C# Items**, and then choose the **Application Configuration File** template.
3.In the **Name** text box, enter a name, and then choose the **Add** button.
A file that's named **app.config** is added to your project.

Adding connection string to App.config

```
<configuration>
   <connectionStrings>
     <add name="MyConnectionString"
            connectionString="Data Source=.;Initial
Catalog=SPJ_DB_Lab4;Integrated Security=True"/>
    </connectionStrings>
</configuration>
```


ConfigurationManagerImport - Microsoft Visual Studio (Administrator)
Edit View Project Build Debug Team Data Tools
Form1.cs*
ConfigurationManagerImport.Form1
Add Reference
.NET COM Projects Browse Recent
Filtered to: .NET Framework 4 Client Profile
Component Name
System.AddIn.Contract
System.AddIn
System.ComponentModel.Composition
System.ComponentModel.DataAnnotations
System.Configuration
System.Configuration.Install
System.Core
System.Data.DataSetExtensions
System.Data
System.Data.Entity
OK
Cancel
Solution Explorer
Solution 'ConfigurationManagerImport' (1 project)
ConfigurationManagerImport
Properties
References
Add Reference...
Add Service Reference...
Add Library Package Reference...
System.Data
System.Data.DataSetExtensions
System.Deployment
System.Drawing
System.Windows.Forms
System.Xml
System.Xml.Linq
App.config
Form1.cs
Program.cs
1
2
3
4

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

using System.Configuration;

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

            string conStr = ConfigurationManager.ConnectionStrings["connectionString"].ConnectionString;
        }
    }
}
```

class System.Configuration.ConfigurationManager
Provides access to configuration files for client applications. This class cannot be inherited.

Get the connection string in C# code

```
using System.Configuration;
//needed to use ConfigurationManager


String DB_Connection_String =
ConfigurationManager.ConnectionStrings["MyConnectionString"].ConnectionSt
ring;
//You can now use the connection string as before
```

V. Insert Query Command

To execute Insert/Delete/Update statement, you should use method (function) `SqlCommand::ExecuteNonQuery( )`

Code

```
//First take input from user through the GUI "Supplier.cs"
private void Insert_Click(object sender, EventArgs e)
{
    // should validate user input first !!!
    int result = controllerObj.InsertSupplier(sNumTextBox.Text,
        sNameTextBox.Text, cityTextBox.Text, Int16.Parse(statusTextBox.Text));
    if (result == 0)
    {
        MessageBox.Show("The insertion of a new supplier failed");
    }
    else
    {
        MessageBox.Show("The row is inserted successfully!");
    }
}
```

```
//Second the query statement to be executed is prepared in the controller
"Controller.cs"
public int InsertSupplier(string snum, string sname, string city, int status)
{
    string query = "INSERT INTO S (S#, Name, City, Status) " +
          "Values ('" + snum + "','" + sname + "','" + city + "'," + status + ");";
    return dbMan.ExecuteNonQuery(query);
}

// In "DBManager.cs" myConnection is the connection opened in the previous step
//SqlCommand class is used for executing queries and stored procedures on the
database use in "DBManager.cs"
public int ExecuteNonQuery(string query)
{
    try
    {
        SqlCommand myCommand = new SqlCommand(query, myConnection);
        return myCommand.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        //This is just for debugging purposes
        Console.WriteLine(ex.Message);
        return 0;
    }
}
```

VI. Delete Query Command

Code

```
//First take input from user through the GUI "Supplier.cs" about which supplier to
delete
private void deleteButton_Click(object sender, EventArgs e)
{
    // should validate user input first !!!
    int result = controllerObj.DeleteSupplier(sNumDelTextBox.Text);
    if (result == 0)
    {
        MessageBox.Show("No rows were deleted");
    }
    else
    {
        MessageBox.Show("The row is deleted successfully!");
    }
}

//Second the query statement to be executed is prepared in the controller
"Controller.cs"
public int DeleteSupplier(string snum)
{
    string query = "DELETE FROM S WHERE S#='" + snum + "';";
    return dbMan.ExecuteNonQuery(query);

    // This will call the same ExecuteNonQuery method defined in "DBManager.cs"
    // and used in the previous section
}
```

VII. Update Query Command

Code

```
//First take input from user through the GUI "Supplier.cs" about which supplier to
update
private void updateButton_Click(object sender, EventArgs e)
{
    // should validate user input first !!!
    int result = controllerObj.UpdateSupplier(sNumUpdateTextBox.Text,
         cityUpdateTextBox.Text);
    if (result == 0)
    {
         MessageBox.Show("No rows are updated");
    }
    else
    {
         MessageBox.Show("The row is updated successfully");
    }
}
//Second the query statement to be executed is prepared in the controller
"Controller.cs"
public int UpdateSupplier(string snum, string city)
{
    string query = "UPDATE S SET City='" + city + "' WHERE S#='" + snum + "';";
    return dbMan.ExecuteNonQuery(query);
    // This will call the same ExecuteNonQuery method defined in "DBManager.cs"
    // and used in section V
}
```

VIII. Select Query Command that returns a Scalar

`SqlCommand::ExecuteScalar( )`

Code

```
//eventhandler for the Count Button to count all suppliers, the result will be
shown on the readonly text box: countTextBox
private void countButton_Click(object sender, EventArgs e)
{
    countTextBox.Text = controllerObj.CountSuppliers().ToString();
}
//Second the query statement to be executed is prepared in the controller
"Controller.cs"
public int CountSuppliers()
{
    string query = "SELECT COUNT(S#) FROM S;";
    return (int) dbMan.ExecuteScalar(query);
}
// In "DBManager.cs" SqlCommand ExecuteScalar will process the query passed to it
from the controller
public object ExecuteScalar(string query)
{
    try
    {
         SqlCommand myCommand = new SqlCommand(query, myConnection);
          return myCommand.ExecuteScalar();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return 0;
    }
}
```

IX. Select Query Command that returns a DataTable

`SqlCommand::ExecuteReader( )` method

Code

```
//eventhandler for the Select All Suppliers Button to get data for all suppliers,
the result will be shown in a datagrid view
private void selectAllButton_Click(object sender, EventArgs e)
{
    DataTable dt = controllerObj.SelectAllSuppliers();
    suppliersDataGrid.DataSource = dt;
    suppliersDataGrid.Refresh();
}
//Second the query statement to be executed is prepared in the controller
"Controller.cs"
public DataTable SelectAllSuppliers()
{
    string query = "SELECT * FROM S;";
    return dbMan.ExecuteReader(query);
}
// In "DBManager.cs" SqlCommand ExecuteReader will process the query passed to it
from the controller
public DataTable ExecuteReader(string query)
{
    try
    {
        SqlCommand myCommand = new SqlCommand(query, myConnection);
        SqlDataReader reader = myCommand.ExecuteReader();
        if (reader.HasRows)
        {
            DataTable dt = new DataTable();
            dt.Load(reader);
            reader.Close();
            return dt;
        }
        else
        {
            reader.Close();
            return null;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return null;
    }
}
```

X. Close the connection to the database

Code

```
public void CloseConnection()
{
    try
    {
        myConnection.Close();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```