C# Windows Forms (Desktop app) Development Tutorial

By: Vincent Paul P. de Jesus

This tutorial will teach you how to learn programming using C# / VB.Net and SQL Server database development. In this tutorial we will use ADO.Net to connect the front-end UI (User Interface) to the back-end (SQL DB).

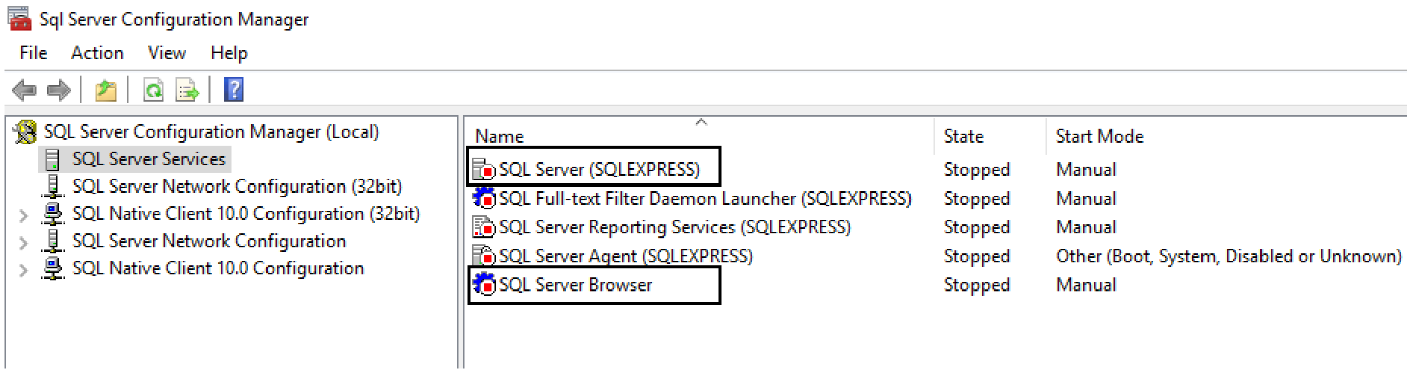
I will also tackle how to design reports using the free add-in tool for Visual Studio, the SAP Crystal Reports. As you know SAP Crystal Reports is not a product of Microsoft but from the SAP company. SAP provides free tool to install that will form part of Visual Studio 2013. I will also tackle how to make an installer for the project.

The pre-requisite tools to be installed for this tutorial are the following:

1. Visual Studio 2013 / .Net Framework
2. SAP Crystal Reports which can be downloaded from this link - http://scn.sap.com/docs/DOC-7824
3. Microsoft Visual Studio 2013 Installer Projects – this extension is used for making an installer for your project when you decided to deploy your project. The installer can be downloaded here - https://visualstudiogallery.msdn.microsoft.com/9abe329c-9bba-44a1-be59-0fbf6151054d. Or you can use InstallShield Limited Edition which can be downloaded and installed separately as a setup maker for your project. Installshied LE can be downloaded here - http://learn.flexerasoftware.com/content/IS-EVAL-InstallShield-Limited-Edition-Visual-Studio
4. Microsoft SQL Server Database Express – this is used to store the data for your project. There are other database server that we can use for this tutorial like MySQL, Oracle, PostgreSQL, etc. but we will use SQL Server for this example. Install the software and all its updates and Service Packs if still not installed on your PC. We are using SQL Server Express 2008 R2 for this tutorial. You can use other versions as well.

We'll begin our tutorial. Supposing you have installed all the pre-requisite tools.

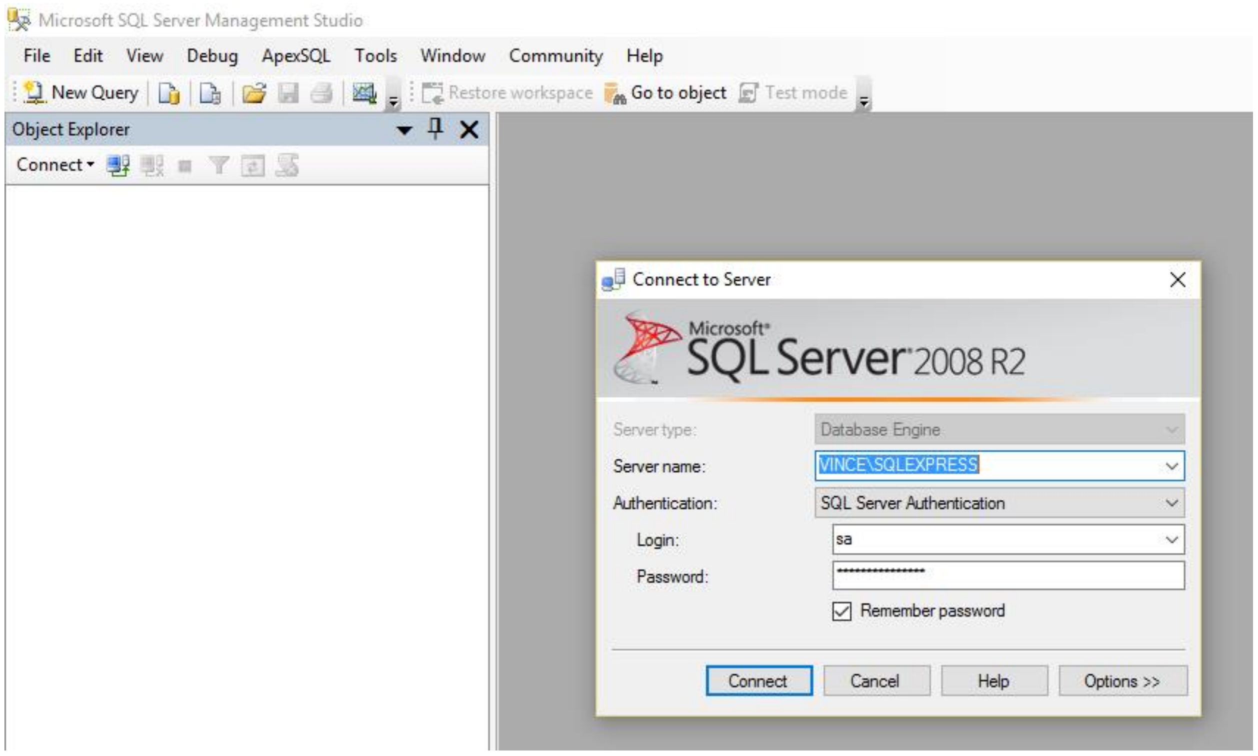
First, we should open the **SQL Server Configuration Manager** to start our Database Server. Click the shortcut on your desktop or open the tool from the Windows Startup menu. Upon opening you can see the image below:


Sql Server Configuration Manager
File
Action
View
Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration (32bit)
SQL Native Client 10.0 Configuration (32bit)
SQL Server Network Configuration
SQL Native Client 10.0 Configuration

Name	State	Start Mode
SQL Server (SQLEXPRESS)	Stopped	Manual
SQL Full-text Filter Daemon Launcher (SQLEXPRESS)	Stopped	Manual
SQL Server Reporting Services (SQLEXPRESS)	Stopped	Manual
SQL Server Agent (SQLEXPRESS)	Stopped	Other (Boot, System, Disabled or Unknown)
SQL Server Browser	Stopped	Manual

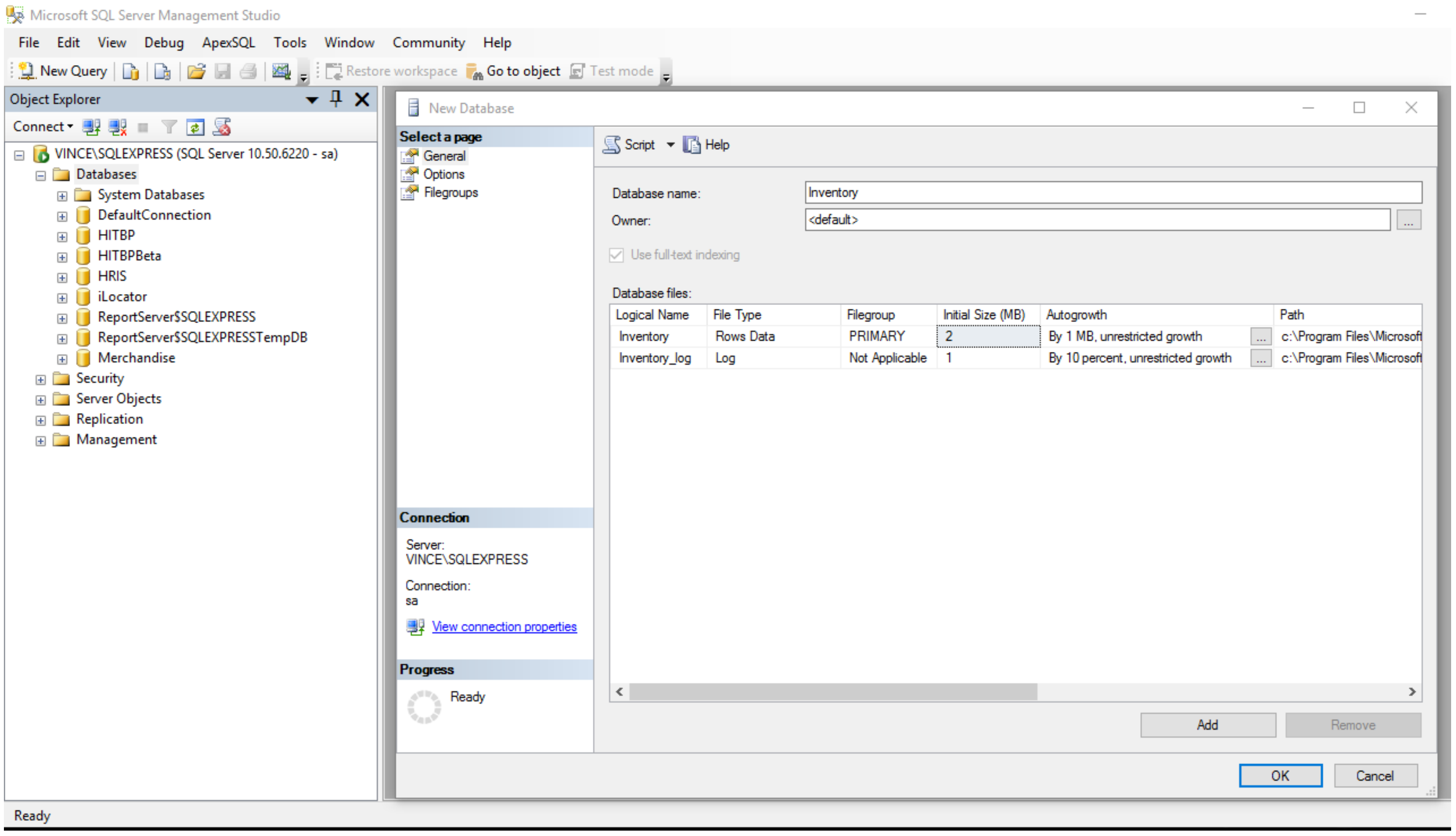
Right click the SQL Server (SQLExpress) Database Engine and click Start to start our server. Next, rigtt-click also the SQL Server Browser to start our Browser. You can enable also the other services if you are going to use them.

After starting our server, we will then open the SQL Server Management Studio. This is the tool that we are going to use to design our database like creating new databases, tables, views, stored procedures, etc. Click the shortcut on your desktop or from the Start menu. When the app is open, click the Connect from the Object Explorer. A pop-up window will open and the image below is displayed.


Microsoft SQL Server Management Studio
File Edit View Debug ApexSQL Tools Window Community Help
New Query
Restore workspace
Go to object
Test mode
Object Explorer
Connect
Connect to Server
Microsoft SQL Server 2008 R2
Connect
Cancel
Help
Options >>

Server type: Database Engine
Server name: VINCE\SQLEXPRESS
Authentication: SQL Server Authentication
Login: sa
Password: ***************
☑ Remember password

And the app requires you to enter your Login name and password. The server name is composed of two names separated by back-slash (\). The first is the name of the PC where the Server DB is installed called VINCE then followed by a back-slash with the name of the SQL Server instance called SQLExpress (the Default-name of the SQL Server Express upon installing). After entering all the information required, click the **Connect** button to finally open and connect to the server. Then right click on the **Database** and click **New Database**. The image below will show.



Database name: Inventory
Owner: <default>
☑ Use full-text indexing

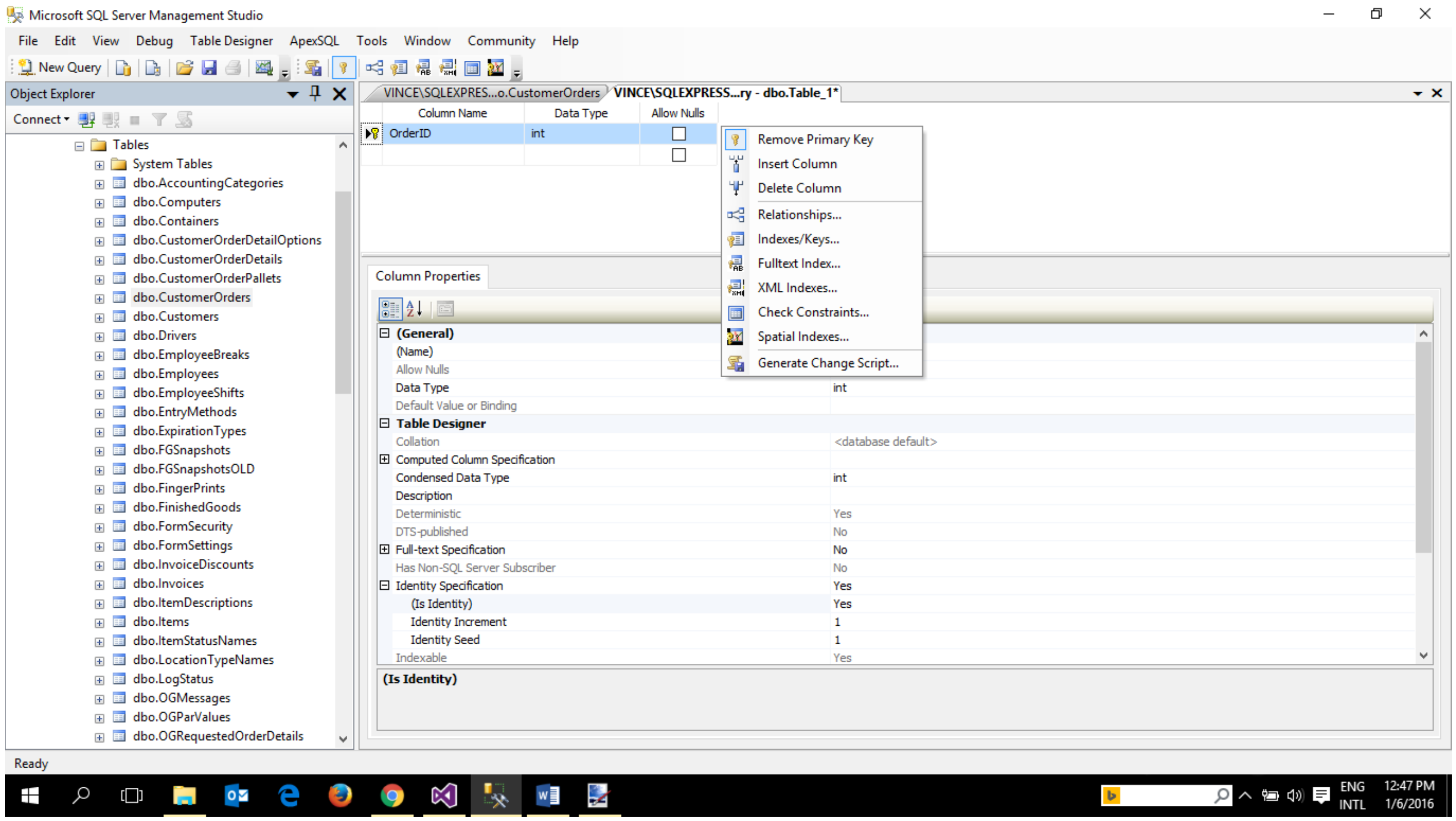
Server: VINCE\SQLEXPRESS
Connection: sa

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth	Path
Inventory	Rows Data	PRIMARY	2	By 1 MB, unrestricted growth	c:\Program Files\Microsoft
Inventory_log	Log	Not Applicable	1	By 10 percent, unrestricted growth	c:\Program Files\Microsoft

Enter the name of the Database. This time we'll call it **Inventory**. Database name is Inventory and Owner is <default>. The SQL Server Database is composed of two Logical Name. The Row Data type and the log file. These files will be saved in the **C:\Program Files\Microsoft SQL Server\MSSQL10_50.SQLEXPRESS\MSSQL\DATA** upon creating. Click OK button to create our database. You can see now a database named Inventory under **Databases** on the left pane of your SQL Server Management Studio.

After creating the Database Inventory, we are now going to create our first table called ***CustomerOrders***. For Microsoft standard naming convention, a table name should be plural. If it is in 2 words the first word is singular and the second word is plural just like in our case **CustomerOrders**. If it is in one word, the table name should be in plural form. Now under Inventory Database, right-click the **Table** menu and click **New Table.** A pop-up window will appear, enter on the first column OrderID and set it to a Primary Key. Right-click and click Primary Key then set the Identity Specification to the image below. Under Identity Specification of column properties, (Is Identity) is Yes, Identity Increment is 1 and Identity Seed is 1. This will make the OrderID field the primary key of the table and this will auto-increment once data is entered into the table.


Microsoft SQL Server Management Studio
File Edit View Debug Table Designer ApexSQL Tools Window Community Help
New Query
Object Explorer
Connect
VINCE\SQLEXPRES...o.CustomerOrders
VINCE\SQLEXPRESS...ry - dbo.Table_1*
Remove Primary Key
Insert Column
Delete Column
Relationships...
Indexes/Keys...
Fulltext Index...
XML Indexes...
Check Constraints...
Spatial Indexes...
Generate Change Script...
Column Properties
Ready
ENG INTL 12:47 PM 1/6/2016

Tables
- System Tables
- dbo.AccountingCategories
- dbo.Computers
- dbo.Containers
- dbo.CustomerOrderDetailOptions
- dbo.CustomerOrderDetails
- dbo.CustomerOrderPallets
- dbo.CustomerOrders
- dbo.Customers
- dbo.Drivers
- dbo.EmployeeBreaks
- dbo.Employees
- dbo.EmployeeShifts
- dbo.EntryMethods
- dbo.ExpirationTypes
- dbo.FGSnapshots
- dbo.FGSnapshotsOLD
- dbo.FingerPrints
- dbo.FinishedGoods
- dbo.FormSecurity
- dbo.FormSettings
- dbo.InvoiceDiscounts
- dbo.Invoices
- dbo.ItemDescriptions
- dbo.Items
- dbo.ItemStatusNames
- dbo.LocationTypeNames
- dbo.LogStatus
- dbo.OGMessages
- dbo.OGParValues
- dbo.OGRequestedOrderDetails

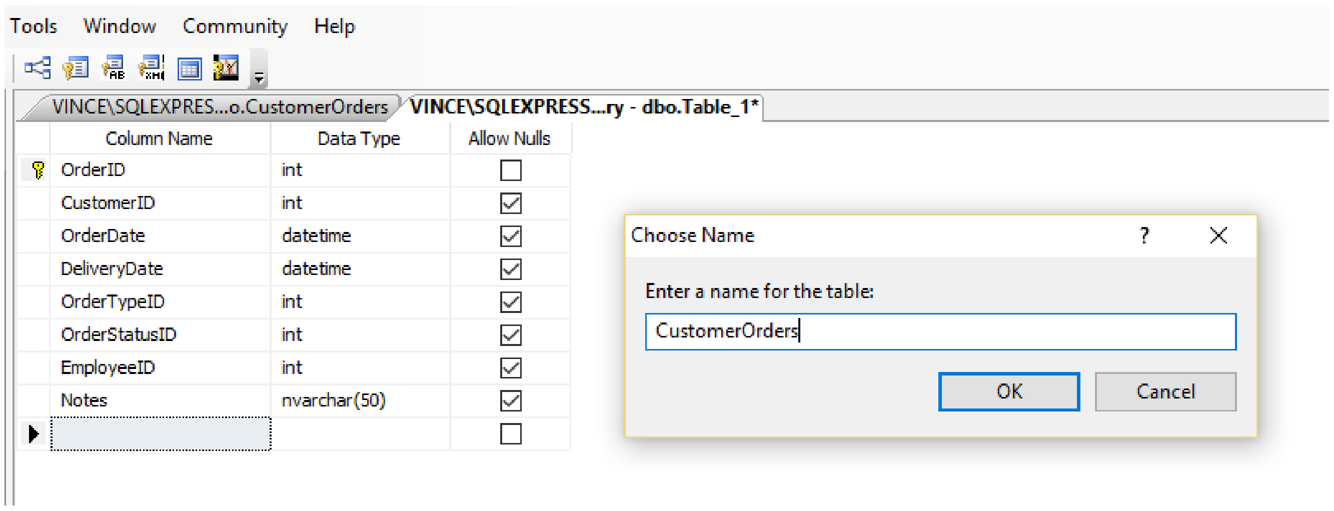
Column Name	Data Type	Allow Nulls
OrderID	int	☐
		☐

(General)	
(Name)	
Allow Nulls	
Data Type	int
Default Value or Binding	
Table Designer	
Collation	<database default>
Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes

(Is Identity)

NOTE: To enable to modify and save tables, please uncheck the "**Prevent saving changes that require table re-creation**". This can be done by accessing the **Tools** menu then **Options** then **Designers** then **Table and Database Designers** on the **SQL Server Management Studio**.

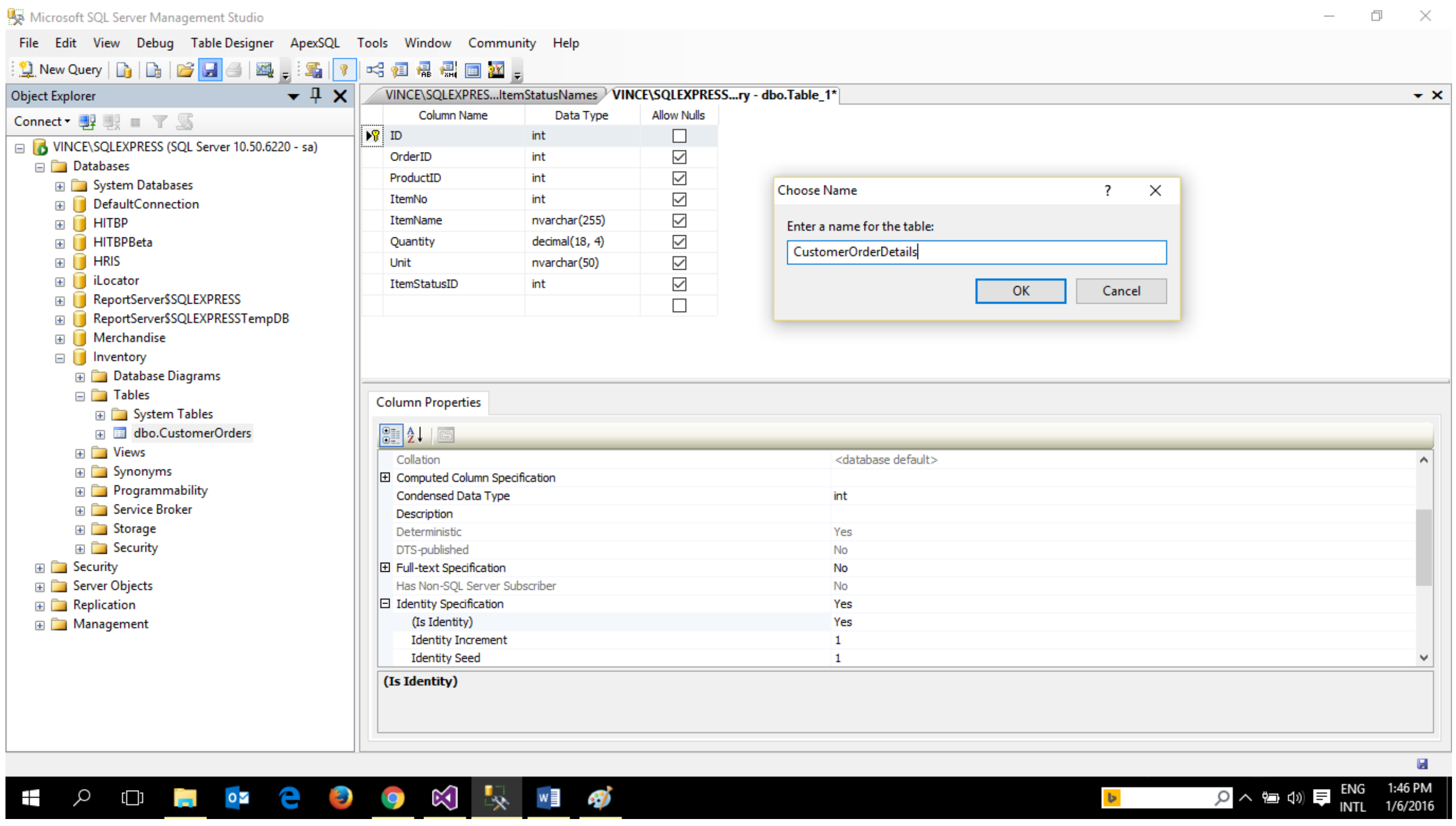
Create all additional fields by adding columns and set its Data Type as shown in the image below and save the table as **CustomerOrders** by clicking the disk icon on the SQL Server Management Studio or SSMS. Please see image on the next page.


Tools
Window
Community
Help
VINCE\SQLEXPRES...o.CustomerOrders
VINCE\SQLEXPRESS...ry - dbo.Table_1*
Choose Name
?
OK
Cancel

Column Name	Data Type	Allow Nulls
OrderID	int	☐
CustomerID	int	☑
OrderDate	datetime	☑
DeliveryDate	datetime	☑
OrderTypeID	int	☑
OrderStatusID	int	☑
EmployeeID	int	☑
Notes	nvarchar(50)	☑
		☐

Enter a name for the table: CustomerOrders

Now create a table called **CustomerOrderDetails** with the following fields and data types below. Set the ID field as the Primary Key. For every table that we make on our Inventory database, we should always create a Primary Key field. This will be a unique key on each table to be used for indexing, searching, etc. Take note CustomerOrderDetails table is the child table on our database. This has relationship with **CustomerOrders** table. The connection between two tables is on the **OrderID** field.



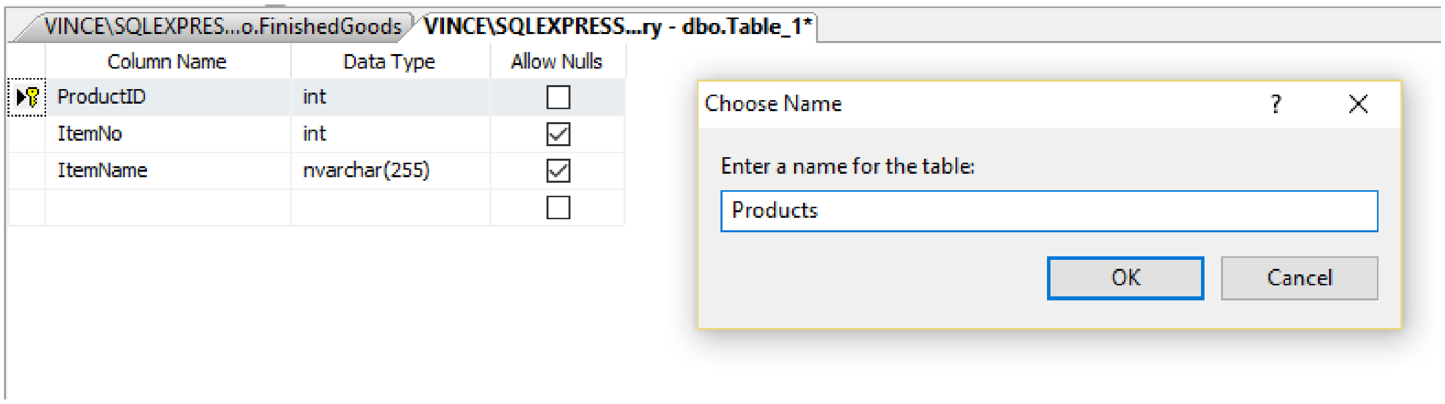
Column Name	Data Type	Allow Nulls
ID	int	☐
OrderID	int	☑
ProductID	int	☑
ItemNo	int	☑
ItemName	nvarchar(255)	☑
Quantity	decimal(18, 4)	☑
Unit	nvarchar(50)	☑
ItemStatusID	int	☑
		☐

Enter a name for the table: CustomerOrderDetails

Collation	<database default>
⊞ Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No
⊞ Full-text Specification	No
Has Non-SQL Server Subscriber	No
⊟ Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

(Is Identity)

The next table is called **Products** table that is used to enter the ProductID (Primary Key), ItemNo, ItemName of the products for the inventory. The data of this table will be used later on the Customer Order Details form that we are going to design. Please create the table by creating the fields and data types from the image below.


VINCE\SQLEXPRES...o.FinishedGoods | VINCE\SQLEXPRESS...ry - dbo.Table_1*

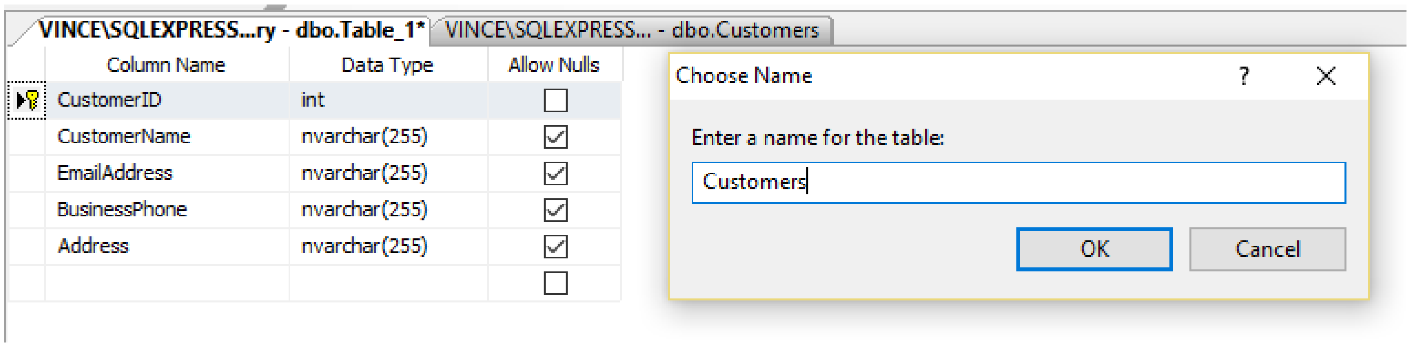
Column Name	Data Type	Allow Nulls
ProductID	int	☐
ItemNo	int	✓
ItemName	nvarchar(255)	✓
		☐

Choose Name

Enter a name for the table:

Products

Next is the Customers table. Create the table and save as **Customers.** Image as shown below:


VINCE\SQLEXPRESS...ry - dbo.Table_1* | VINCE\SQLEXPRESS... - dbo.Customers

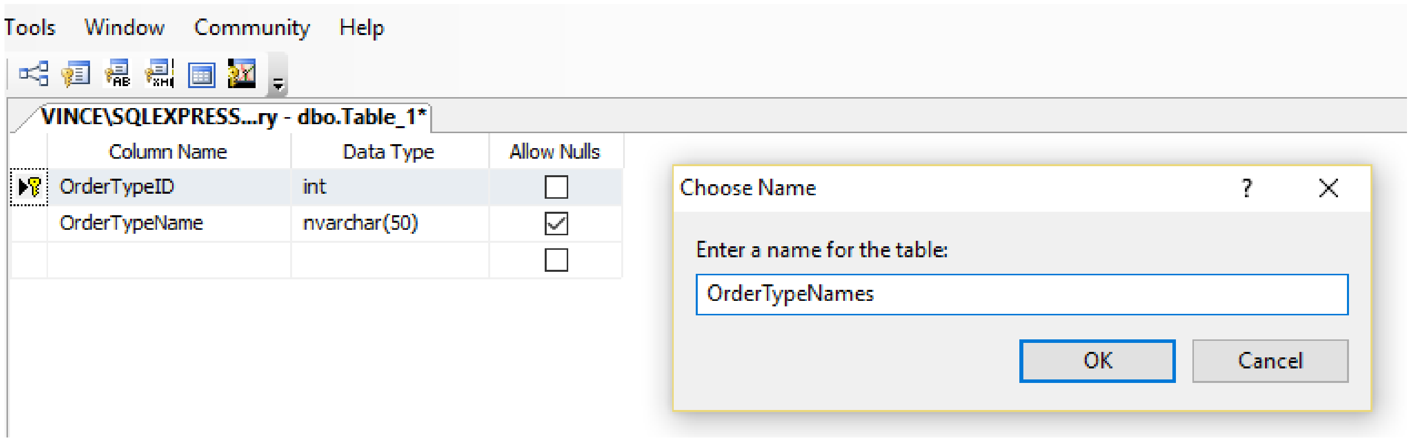
Column Name	Data Type	Allow Nulls
CustomerID	int	☐
CustomerName	nvarchar(255)	✓
EmailAddress	nvarchar(255)	✓
BusinessPhone	nvarchar(255)	✓
Address	nvarchar(255)	✓
		☐

Choose Name

Enter a name for the table:

Customers

Next is the OrderTypeNames table. This tables will be used if the Order is for Delivery or Pickup. Create as shown below:



VINCE\SQLEXPRESS...ry - dbo.Table_1*

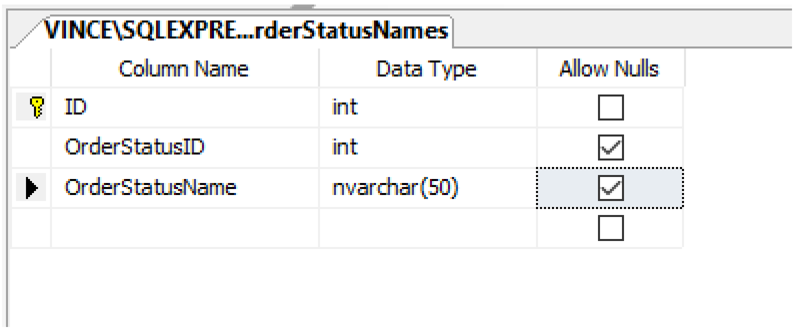
Column Name	Data Type	Allow Nulls
OrderTypeID	int	☐
OrderTypeName	nvarchar(50)	✓
		☐

Choose Name

Enter a name for the table:

OrderTypeNames

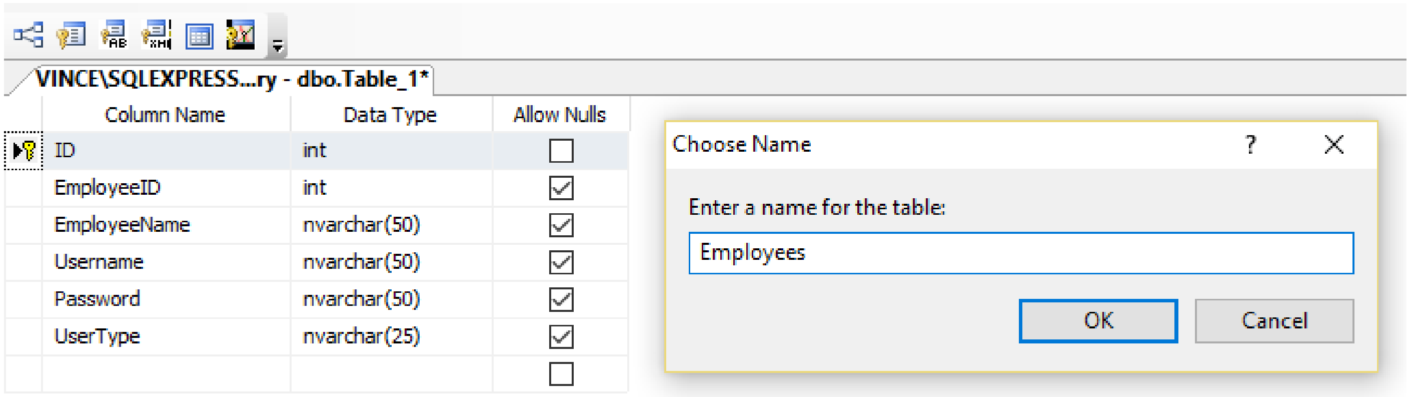
Next will be the OrderStatusNames table. Please create the table and save it as **OrderStatusNames**.



VINCE\SQLEXPRE...rderStatusNames

Column Name	Data Type	Allow Nulls
ID	int	☐
OrderStatusID	int	✓
OrderStatusName	nvarchar(50)	✓
		☐

Next will be the Employees table. This will serve as the Users of our Inventory system. Please create the table and follow the field names and data types below.



VINCE\SQLEXPRESS...ry - dbo.Table_1*

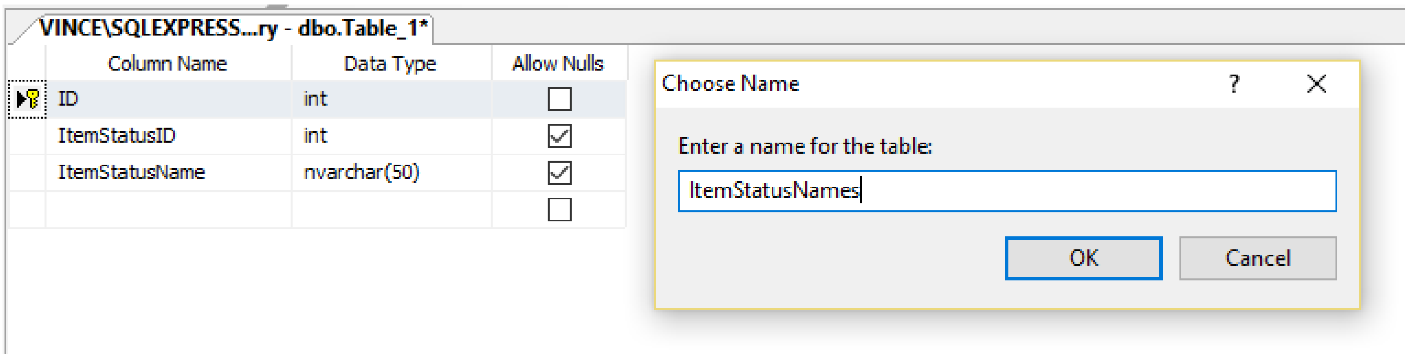
Column Name	Data Type	Allow Nulls
ID	int	☐
EmployeeID	int	✓
EmployeeName	nvarchar(50)	✓
Username	nvarchar(50)	✓
Password	nvarchar(50)	✓
UserType	nvarchar(25)	✓
		☐

Choose Name

Enter a name for the table:

Employees

The last table that we are going to create is the ItemStatusNames table. Please create the table with the field names and data types below.



VINCE\SQLEXPRESS...ry - dbo.Table_1*

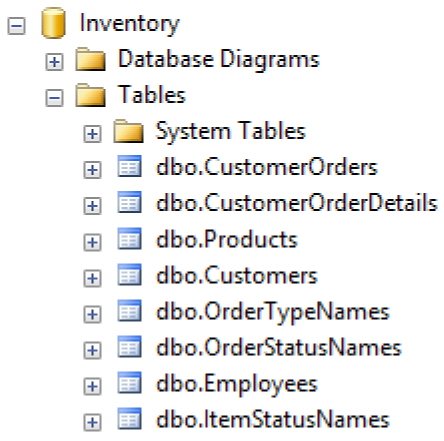
Column Name	Data Type	Allow Nulls
ID	int	☐
ItemStatusID	int	✓
ItemStatusName	nvarchar(50)	✓
		☐

Choose Name

Enter a name for the table:

ItemStatusNames

Our Inventory Database should have this following tables:

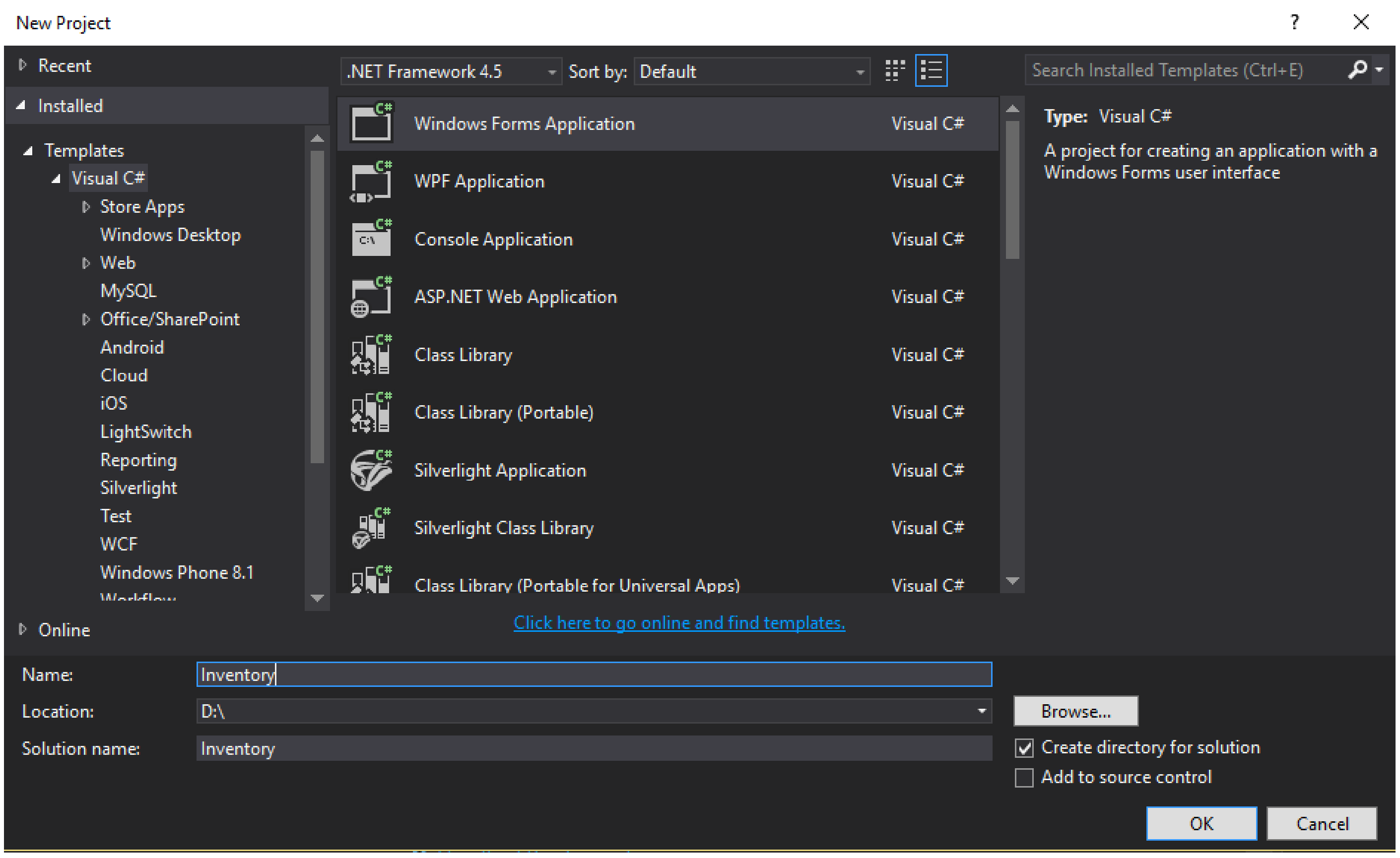


- Inventory
  - Database Diagrams
  - Tables
    - System Tables
    - dbo.CustomerOrders
    - dbo.CustomerOrderDetails
    - dbo.Products
    - dbo.Customers
    - dbo.OrderTypeNames
    - dbo.OrderStatusNames
    - dbo.Employees
    - dbo.ItemStatusNames

Project Development using Visual Studio 2013

The next thing that we should do after creating our database and tables is to create our project under Visual Studio 2013. Visual Studio 2013 is an Integrated Development Environment (IDE) that is developed by Microsoft to allow software developers to design a desktop, web or mobile applications using a programming language like C#, VB.Net, F# and other languages for the .Net platform.

Now open your Visual Studio 2013 IDE from the short cut on your desktop or the start menu. After opening, create the project by clicking the File menu, New and then Project. Choose the C# template and Windows Forms Application as project type since we are developing a desktop type application. There is another desktop type application that is called WPF (Windows Presentation Foundation). But in this case we are using Windows Forms Application or WinForms. A pop-up form will show. Enter the name of your project called Inventory and where would you like to save it. In this case the project files is saved into Drive D:\ of the hardisk. Please see image below:


New Project

Recent
Installed
Templates
Visual C#
Store Apps
Windows Desktop
Web
MySQL
Office/SharePoint
Android
Cloud
iOS
LightSwitch
Reporting
Silverlight
Test
WCF
Windows Phone 8.1
Online

.NET Framework 4.5 | Sort by: Default | Search Installed Templates (Ctrl+E)

Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
ASP.NET Web Application	Visual C#
Class Library	Visual C#
Class Library (Portable)	Visual C#
Silverlight Application	Visual C#
Silverlight Class Library	Visual C#
Class Library (Portable for Universal Apps)	Visual C#

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Click here to go online and find templates.

Name: Inventory
Location: D:\
Solution name: Inventory

[✓] Create directory for solution
[ ] Add to source control

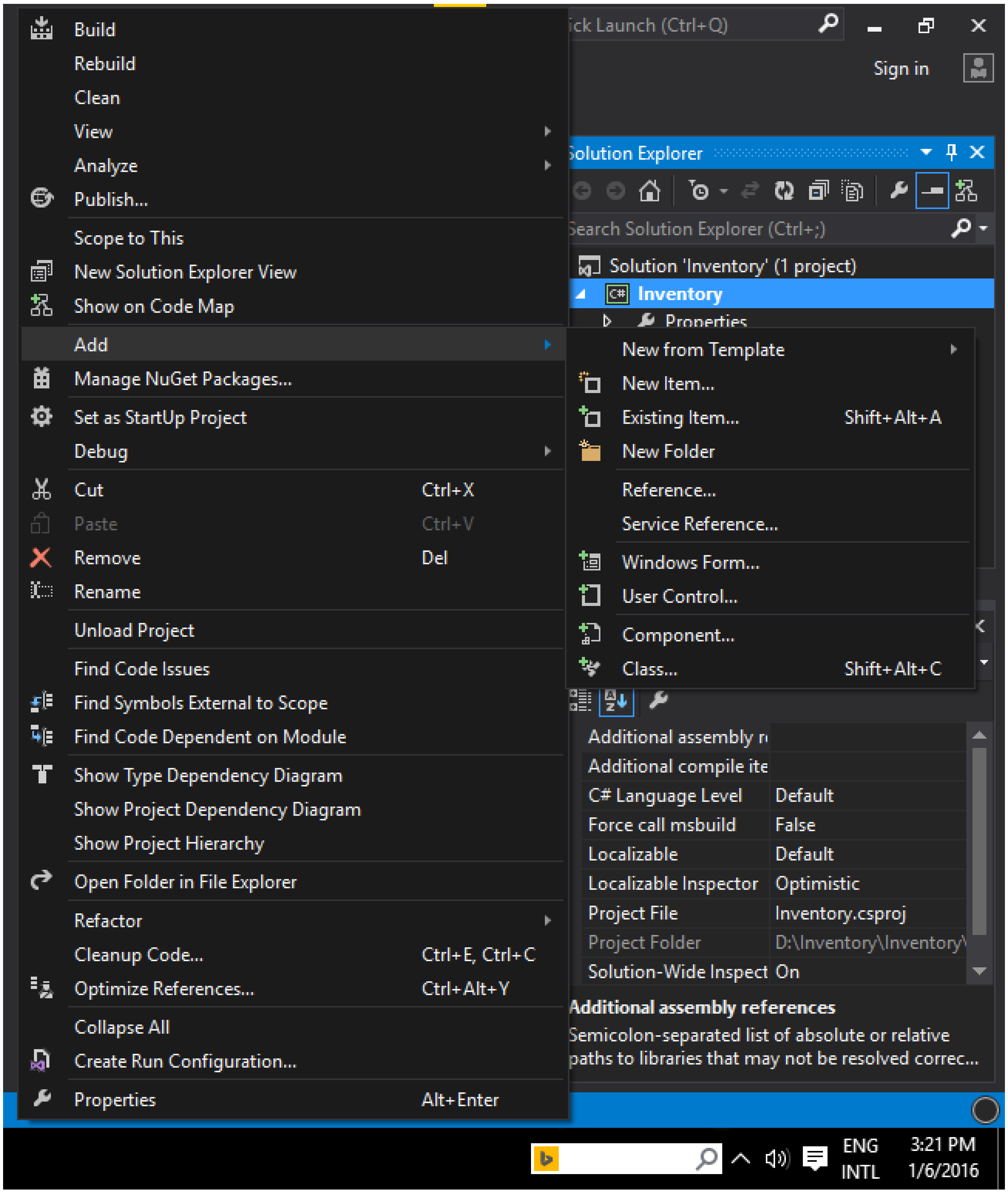
After clicking the OK button, an "**Inventory**" Solution project will be created, properties and a default Form1.cs default form. Please delete the Form1.cs file by right-clicking and clicking delete. Since we are developing desktop application using WinForms, there is a design pattern that we are going to use called MVP pattern.

A **Model View Presenter (MVP)** is a user interface architectural pattern engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic:

- The *model* is an interface defining the data to be displayed or otherwise acted upon in the user interface.
- The *presenter* acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.
- The *view* is a passive interface that displays data (the model) and routes user commands (events) to the presenter to act upon that data.

To have an overview about this pattern please research more about this on Microsoft sites or other programming site.

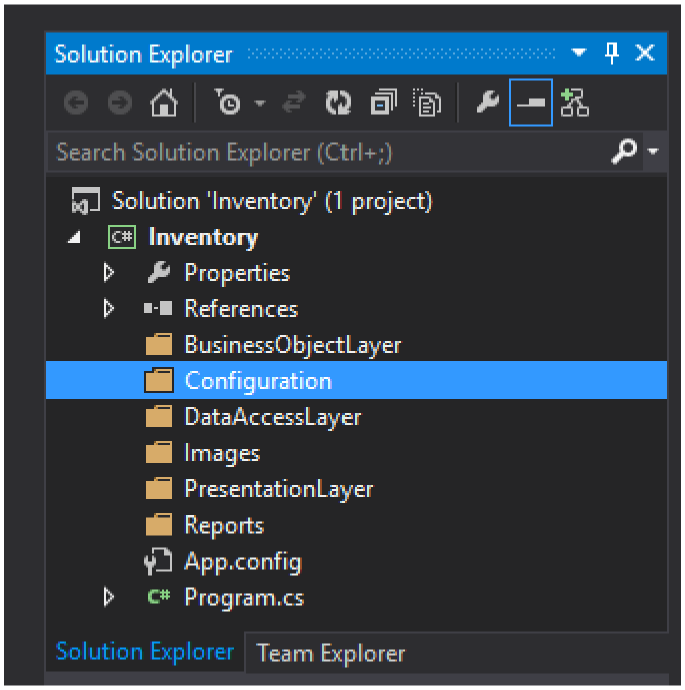
After deleting the Form1.cs file, we will now create a new folder on our project solution to develop this kind of pattern. We will use this pattern so that our code will be more organize and maintainable. Next right click our Inventory project under solution then click Add then click New Folder and name the new folder PresentationLayer. Create another and name it BusinessObjectLayer and another one for DataAccessLayer. Please see the image below on how to create.



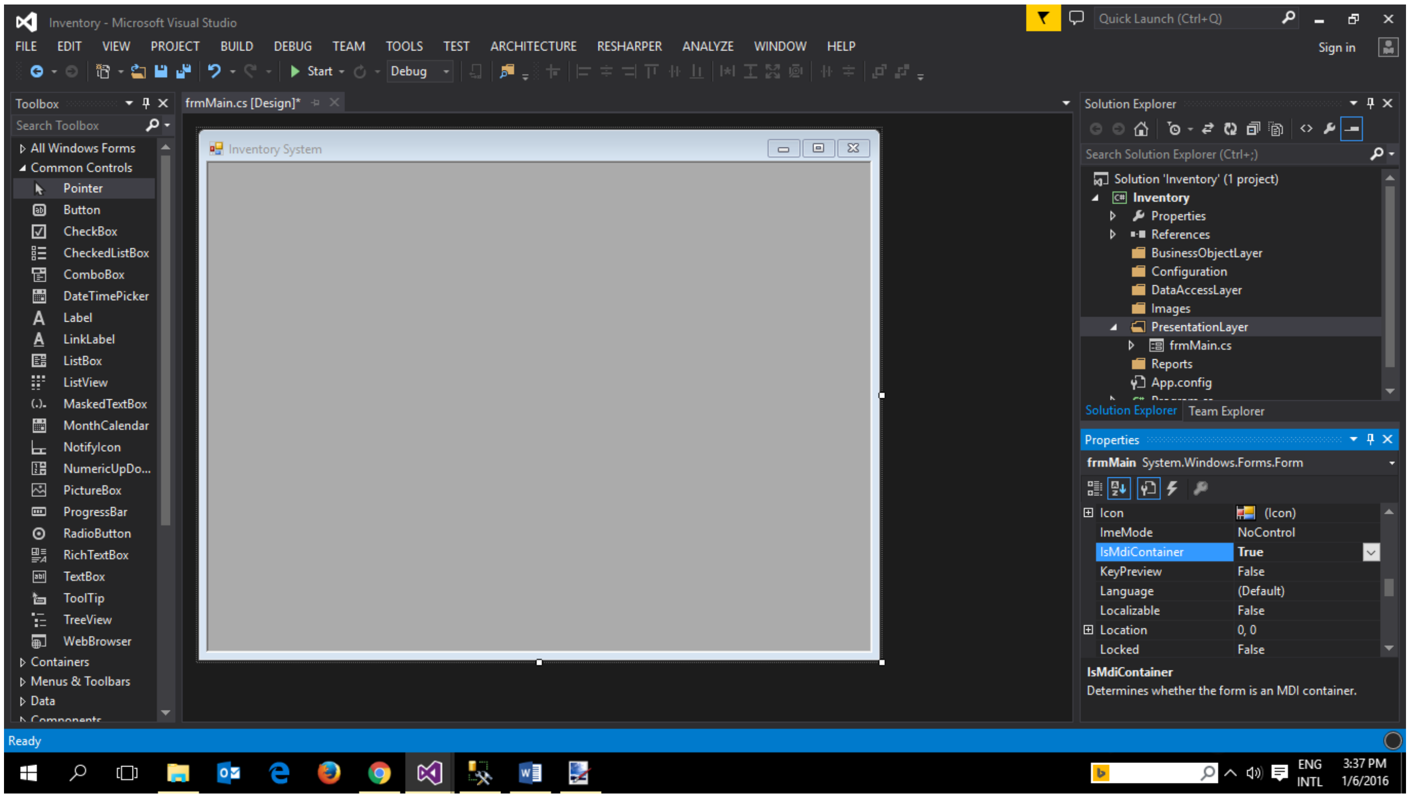
Additional assembly r	
Additional compile ite	
C# Language Level	Default
Force call msbuild	False
Localizable	Default
Localizable Inspector	Optimistic
Project File	Inventory.csproj
Project Folder	D:\Inventory\Inventory\
Solution-Wide Inspect	On

Additional assembly references
Semicolon-separated list of absolute or relative paths to libraries that may not be resolved correc...

Create another folder to store reports, configuration classes and images. After creating our new folders our solution will now look like this image below:



PresentationLayer folder is used to store all our forms. Now we are going to create our first form. Right-click the PresentationLayer folder and click Add then Windows Form. Then name the form “**frmMain**” then click OK button. After this a new form is created under PresentationLayer folder. We will make this form our parent form to house our child forms. Rename the heading on the form from “frmMain” to “Inventory System”by changing the Text property on the right pane under Properties. Be sure the frmMain form has the focus. Then change the IsMdiContainer property to True and the WindowState property to Maximized. Please see image below.

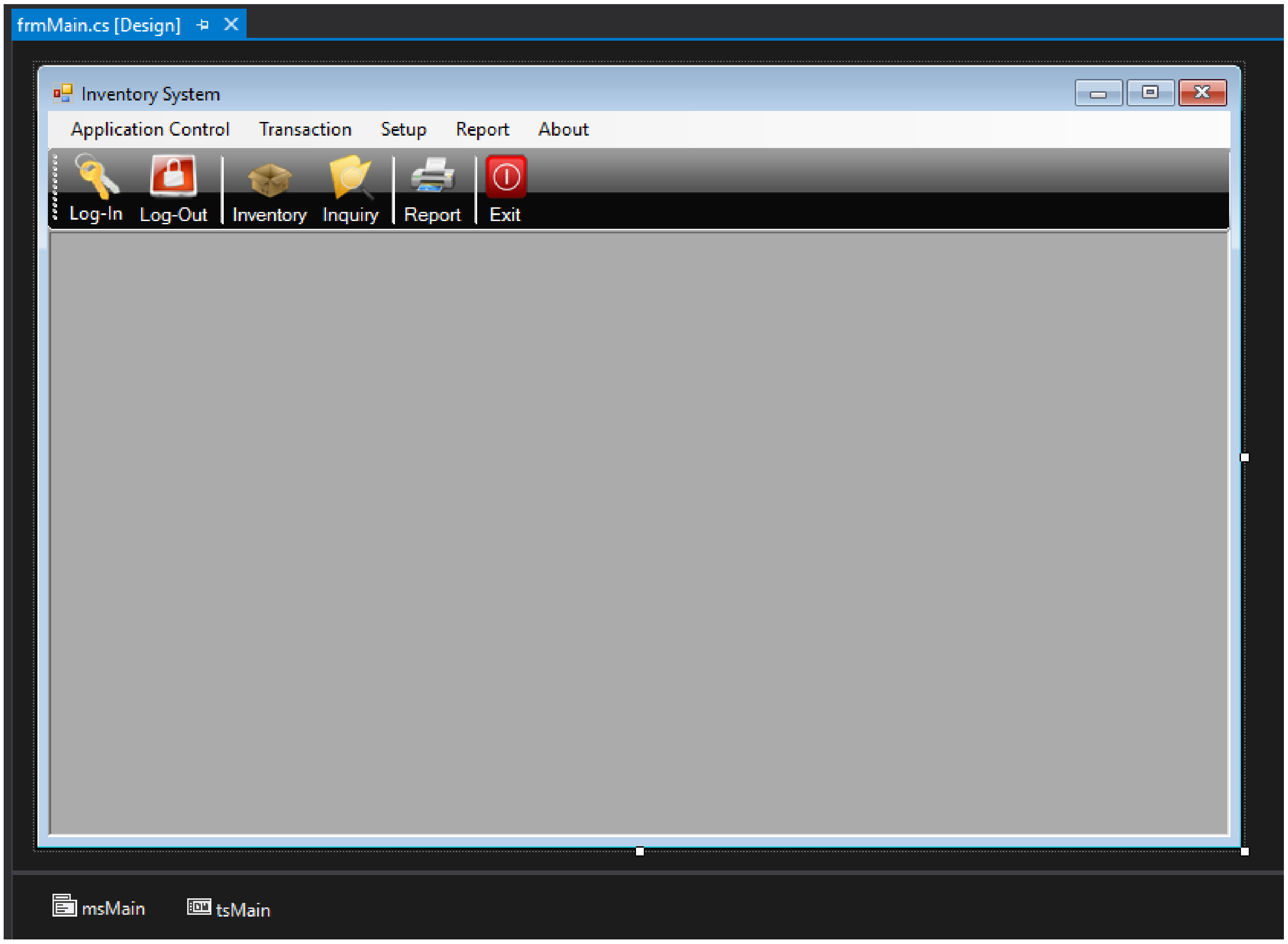


frmMain System.Windows.Forms.Form

Icon	(Icon)
ImeMode	NoControl
IsMdiContainer	True
KeyPreview	False
Language	(Default)
Localizable	False
Location	0, 0
Locked	False

IsMdiContainer
Determines whether the form is an MDI container.

Next is we should create a menu for our parent form. Under Menu & Toolbars on the Toolbox (left side of our IDE), click and hold then drag and drop the **MenuStrip** control on our form. You can now see a menu on the upper corner of our form and on the lower corner, you can see the control that is being added. Now rename the control under its properties as "***msMain***". On the upper corner type in the first name of our menu called "Application Control" and name the control **mnApplication**, next add another menu and type in "Transactions" and name it **mnTransaction**, next add another menu and name it "Setup"and name the control **mnSetup,** next add a menu and call it "Reports"and name the control **mnReport** and finally add a menu and call it "About" and name it **mnAbout**. Under our Application Control menu, add a Log-In (**mnLogin**) menu, Log-Out (**mnLogout**) menu, Separator, and Exit (**mnExit**) menu. Add also a **ToolStrip** control by clicking + holding then drag and drop the control on the parent form. Add then add then add a ToolStripButton with an image and text below. You can modify the properties of each control on the **Properties Window** on the right side of our IDE. Add a background image for our **ToolStrip** control from the images that came along with this tutorial. By the way, this tutorial has an accompanied source code. Please refer to it if you want to verify something. Our parent form should look like the one below:



Next we are now going to design other forms for this project like the **Login** form. But before that, we should add a database connection string on our **App.config** file under our solution. The App.config file should look like something below:

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

        <connectionStrings>

                <add name="ApplicationServices"

                        connectionString="Data Source=PAUL\SQLExpress;Initial Catalog=Inventory;User
Id=sa;Password=super;Max Pool Size=500;Min Pool Size=5;Connection Timeout=240"

                        providerName="System.Data.SqlClient"/>

        </connectionStrings>

  <startup>

    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

  </startup>

</configuration>
```

The config file is actually in XML file format. Under the Configuration tag is our database connectionStrings tag. The name for our connection string tag is **ApplicationServices** then we have our database connection string which is composed of our Data Source. Data Source is a combination of the name of the PC where the server is installed + \ + SQL Server name (SQLExpress). Initial Catalog is the name of our database. In our case the **Inventory** database. The User Id and password respectively of our SQL Server. To know more about ConnectionStrings please visit this link - https://www.connectionstrings.com/sql-server/.

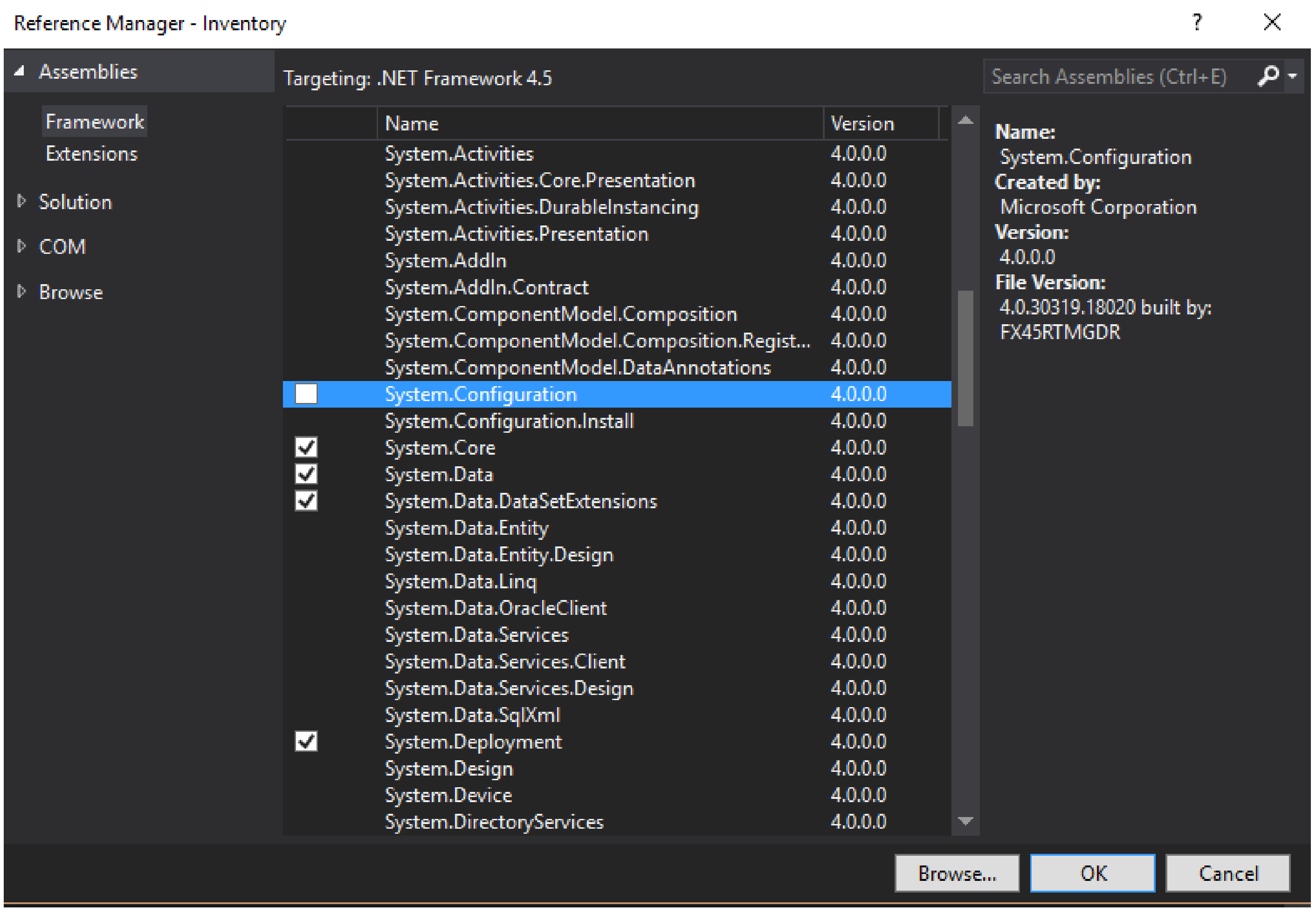
After this, we will now add a new class to pull this connection string and this class will be used on each form that needs connection with our database. Now, right click the folder **Configuration** and add a new class. You can do this by right-clicking the folder then click **Add** then click **Class**. After that a pop-up window will appear and asking you for the name of the class. A default Class1.cs appears. Now rename that one to MainConn.cs file and click OK button. Our default class should look like below:

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

namespace Inventory.Configuration
{
    class MainConn
    {
    }
}
```

Now replace this class with the one on the next page.

```
using System.Data.SqlClient;
using System.Configuration;

namespace Inventory.Configuration
{
    public class MainConn
    {
        private string _connectionString = string.Empty;
        private readonly string _serverName = string.Empty;
        private readonly string _databaseName = string.Empty;
        private readonly string _userId = string.Empty;
        private readonly string _password = string.Empty;

        public MainConn()
        {
            var objConnStringSettings =
ConfigurationManager.ConnectionStrings["ApplicationServices"];
            _connectionString = objConnStringSettings.ConnectionString;

            var builder = new SqlConnectionStringBuilder(_connectionString);
            _serverName = builder.DataSource;
            _databaseName = builder.InitialCatalog;
            _userId = builder.UserID;
            _password = builder.Password;
        }

        public string ConnectionString
        {
            get { return _connectionString; }
            set { _connectionString = value; }
        }

        public string ServerName
        {
            get {return _serverName;}
        }

        public string DatabaseName
        {
            get { return _databaseName; }
        }

        public string UserId
        {
            get { return _userId; }
        }

        public string Password
        {
            get { return _password; }
        }
    }
}
```

After pasting the code on our MainConn.cs file you will notice that the System.Configuration namespace from the .Net Framework is not yet added or an error on the code will appear when compiled. Now right-click the Reference tab/folder of the project and click Add Reference. A pop-up window will show. Go to Assemblies and then Framework. Choose the System.Configuration file namespace by checking the checkbox and clicking OK button. Please refer to the image below:


Reference Manager - Inventory

Assemblies
- Framework
- Extensions

Solution

COM

Browse

Targeting: .NET Framework 4.5

Search Assemblies (Ctrl+E)

	Name	Version
	System.Activities	4.0.0.0
	System.Activities.Core.Presentation	4.0.0.0
	System.Activities.DurableInstancing	4.0.0.0
	System.Activities.Presentation	4.0.0.0
	System.AddIn	4.0.0.0
	System.AddIn.Contract	4.0.0.0
	System.ComponentModel.Composition	4.0.0.0
	System.ComponentModel.Composition.Regist...	4.0.0.0
	System.ComponentModel.DataAnnotations	4.0.0.0
☐	System.Configuration	4.0.0.0
	System.Configuration.Install	4.0.0.0
✓	System.Core	4.0.0.0
✓	System.Data	4.0.0.0
✓	System.Data.DataSetExtensions	4.0.0.0
	System.Data.Entity	4.0.0.0
	System.Data.Entity.Design	4.0.0.0
	System.Data.Linq	4.0.0.0
	System.Data.OracleClient	4.0.0.0
	System.Data.Services	4.0.0.0
	System.Data.Services.Client	4.0.0.0
	System.Data.Services.Design	4.0.0.0
	System.Data.SqlXml	4.0.0.0
✓	System.Deployment	4.0.0.0
	System.Design	4.0.0.0
	System.Device	4.0.0.0
	System.DirectoryServices	4.0.0.0

Name:
System.Configuration
Created by:
Microsoft Corporation
Version:
4.0.0.0
File Version:
4.0.30319.18020 built by:
FX45RTMGDR

Browse... OK Cancel

After clicking the OK button, we can now see that our ConfigurationManager class is enabled and the color turned green from red. It means it's good to go. Visual Studio has default colors for the code, namespace and classes. As you can see our MainClass.cs file is composed of private variables like the one below:

```
private string _connectionString = string.Empty;
 private readonly string _serverName = string.Empty;
 private readonly string _databaseName = string.Empty;
 private readonly string _userId = string.Empty;
 private readonly string _password = string.Empty;
```

These variables are used to store a value for our database connection. The next code will pull the connection string from our ***App.config*** file that we made earlier. Please see below:

```
var objConnStringSettings = ConfigurationManager.ConnectionStrings["ApplicationServices"];
_connectionString = objConnStringSettings.ConnectionString;
```

It will pull the connection string settings using the tag name which is ApplicationServices from our App.config file and then store the value on the objConnStringSettings variable. It will then pass the actual value of the ConnectionString to the private variable that we declared which is the _connectionString variable.

We then get each value of the connection string by using a SqlConnectionStringBuilder variable. SqlConnectionStringBuilder class from the .Net Framework is used to provide a simple way to create and manage the contents of connection strings used by the SqlConnection class. For more information please visit this link - https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnectionstringbuilder(v=vs.110).aspx

Please see the code below:

```
            var builder = new SqlConnectionStringBuilder(_connectionString);
            _serverName = builder.DataSource;
            _databaseName = builder.InitialCatalog;
            _userId = builder.UserID;
            _password = builder.Password;
```

You have noticed that we define and store the values of our connection string under the Constructor of our MainConn.cs file. Our constructor looks like the one below:

```
public MainConn()
{
}
```

As per **Microsoft** reference: Using **Constructors** (**C#** Programming Guide) When a **class** or struct is created, its **constructor** is called. **Constructors** have the same name as the **class** or struct, and they usually initialize the data members of the new object.

You can also refer to any C# book to know more about constructor and classes.

The public properties is used to get or pass the value of the private variables that is used for getting the values from the connection string. The public properties will be used later on each form that needs database connection. The forms will then need to access the **MainConn.cs** class to use the connection string values. Please see public properties code below:

```
public string ConnectionString
 {
     get { return _connectionString; }
     set { _connectionString = value; }
 }
```

This code above declares a property set to public so that it will be accessible anywhere in the project. It is of property string value. It will return the value of _connectionString that is declared as private in our class.

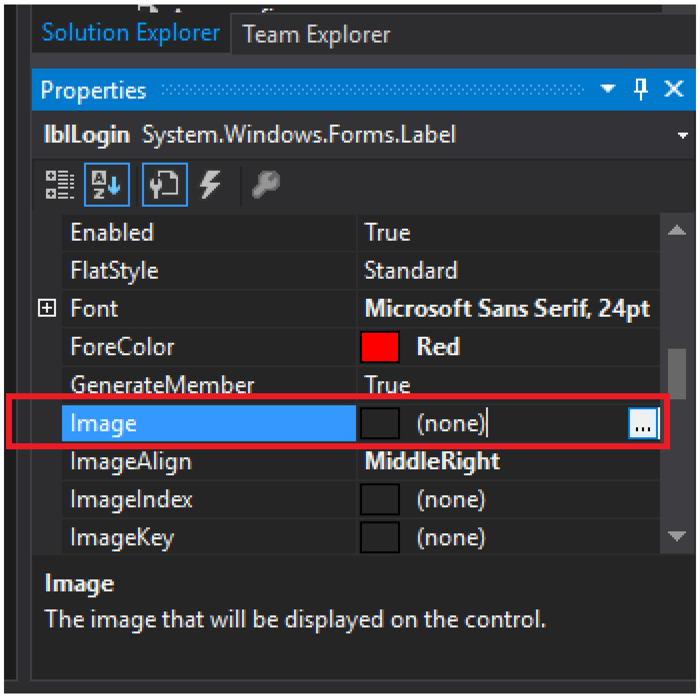
Next we need to add another class based from Singleton design pattern. As per description of Solutions Toolkit book by Rockford Lhotka, the Singleton pattern is used when we want to ensure that there is exactly one instance of a particular class available in an entire application. That is we don't want another instance of our connection string running in our application. We only need to use one instance since our connection string is the same throughout our application.

Please add a new class called **Config.cs** under Configuration folder and follow this code below:

```
using System;
using System.Runtime.Serialization;

namespace Inventory.Configuration
{
    class Config
    {
        private string connString;
        private static object @lock = new object();
        private static Config theInstance;

        public Config()
        {
            var dbAccess = new MainConn(); // Making a new instance of the MainConn Class
            connString = dbAccess.ConnectionString; // Getting the DataSource Property
        }

        public string Connection
        {
            get { return connString; } // Returning the database connection string from the class
        }

        public static Config Instance
        {
            get
            {
                lock (@lock)
                {
                    if (theInstance == null)
                    {
                        theInstance = new Config();
                    }
                }
                return theInstance;
            }
        }

        // Serialization functionality
        [Serializable()]
        public class SerializationHelper : ISerializable
        {
            public void GetObjectData(SerializationInfo info, StreamingContext context)
            {
                var theSingleton = Instance;
                lock (theSingleton) { }
            }
        }

    }
}
```

By the way, a namespace is used to organize code. As per Microsoft reference, the **namespace** keyword is used to declare a scope that contains a set of related objects. You can use a **namespace** to organize code elements and to create globally unique types. In our case, we have namespace Inventory.Configuration for this class. This namespace will be used later in conjunction with a “using” statement when we want access of this class on our forms for database connection.

Now we are ready to add another form for our project. Please add a new form under PresentationLayer folder and call it "**frmLogin**". Set the form's **FormBorderStyle** property value to None and **StartPosition** property value to CenterScreen. Add 4 Label controls. Drag and drop this control on the form. You can access the Label Control under the Common Controls of the ToolBox from the left pane of our IDE. Name the first control lblLogin and display Text to "Login" and BackColor property value to DimGray which you can set on the properties window on the right pane for this control. Set the font to Microsoft Sans Serif, 24pt. Next add an image for this control from its Image property. See image below:

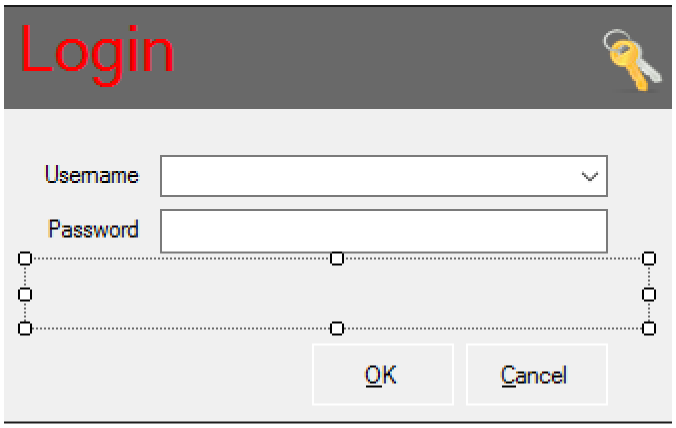


lblLogin System.Windows.Forms.Label

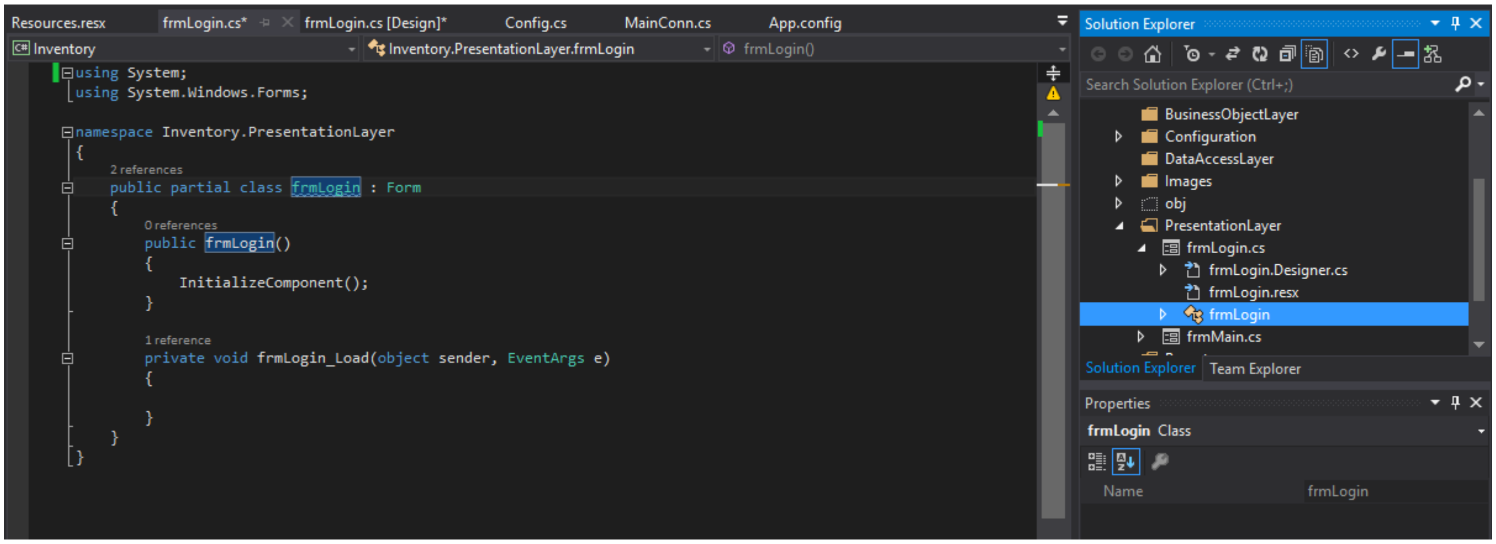
Enabled	True
FlatStyle	Standard
Font	Microsoft Sans Serif, 24pt
ForeColor	🟥 Red
GenerateMember	True
Image	(none)
ImageAlign	MiddleRight
ImageIndex	(none)
ImageKey	(none)

Image
The image that will be displayed on the control.

Then click the button with 3 small dots at the right side of the Image property to add an image. After clicking you will see a pop-up window that will ask you where to get the image. Under Project resource file: click Import button and browse to where your image Key is saved. In our case on this folder D:\Inventory\Inventory\Images. After that select the image and press OK button. You can see now the image under the Label. Add another Label control and name the Text property Username and name the Label control "lblUsername". Add another for the Password and name it "lblPassword". Now add 1 ComboBox control and name it cmbUsername and put it at the right side of the lblUsername control. Make the AutoCompleteMode value of the ComboBox property to SuggestAppend. This will enable the ComboBox append values automatically depending on the characters being type by the user at runtime. Also make the AutoCompleteSource value into ListItems. Now add a TextBox control and place it beside the Password label and name the control txtPassword. After this, add another Label control and name the control "lblStatus" right after the Password controls. Then add 2 buttons and name one btnOk (Name OK the Text property) and the other btnCancel (name Cancel for the Text property. The form should look like the one below:

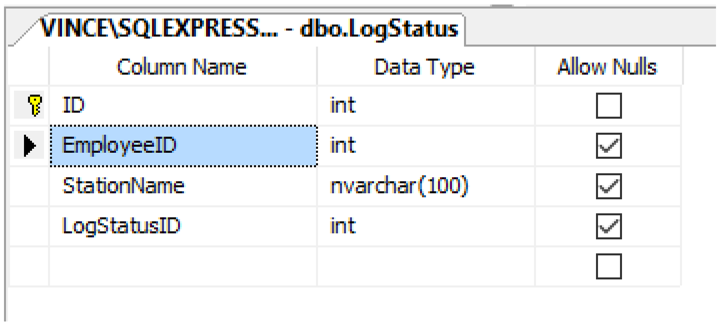


Now we will add the necessary code for our Login form. Expand the Login form from the PresentationLayer folder and click the frmLogin code file and you will see the code behind of the form. Please see image below:



```
using System;
using System.Windows.Forms;

namespace Inventory.PresentationLayer
{
    2 references
    public partial class frmLogin : Form
    {
        0 references
        public frmLogin()
        {
            InitializeComponent();
        }

        1 reference
        private void frmLogin_Load(object sender, EventArgs e)
        {

        }
    }
}
```

At the left side is the default code when we first added the form. Now we will need to modify this. Before this we need to add a new table called LogStatus. This will store data for the Log-In and Log-Out of the users. Open your SQL Server Management Studio and connect to your Database. Under Inventory DB and Table, add a new table with the following schema below and save it as LogStatus table.


VINCE\SQLEXPRESS... - dbo.LogStatus

Column Name	Data Type	Allow Nulls
ID	int	☐
EmployeeID	int	☑
StationName	nvarchar(100)	☑
LogStatusID	int	☑
		☐

Next we will add stored procedures for our Employees table. This will be used later on our DataAccess class. Go to your Inventory database and under Programmability, right click Stored Procedures and click New Stored Procedures. A new window will appear. Modify it and follow the code below:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[usp_Employee_Select]

AS
BEGIN
	SET NOCOUNT ON;
	SELECT *
	FROM dbo.Employees
END
```

After modifying, click the "! Execute" button above our SQL Server Management Studio to create our stored procedure. Right-click the Stored Procedures under Programmability and click Refresh. You will notice a new stored procedure called "usp_Employee_Select" is created. This stored procedure is a SELECT statement. This will select all records from the Employees table when executed and called on our class later.

Next add a new stored procedure for the INSERT statement. Please run the code below on your SQL Window.

```
USE [Inventory]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[usp_Employee_Insert]
      @EmployeeID INT,
      @EmployeeName NVARCHAR(50),
      @Username NVARCHAR(50),
      @Password NVARCHAR(50),
      @UserType NVARCHAR(25)
AS
BEGIN
      SET NOCOUNT ON;
      INSERT INTO dbo.Employees
               ( EmployeeID ,
             EmployeeName ,
             Username ,
             Password ,
             UserType
             )
      VALUES( @EmployeeID ,
                @EmployeeName ,
                @Username ,
                @Password ,
                @UserType
             );
    SELECT  ID = SCOPE_IDENTITY()
END
```

Next add a new stored procedure for the DELETE statement. Please run the code below on your SQL Window.

```
USE [Inventory]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[usp_Employee_Delete]
      @ID INT
AS
BEGIN
      SET NOCOUNT ON;
      DELETE
      FROM dbo.Employees
      WHERE ID = @ID
END
```

Next add a new stored procedure for the UPDATE statement. Please run the code below on your SQL Window.

```
USE [Inventory]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[usp_Employee_Update]
      @ID INT,
      @EmployeeID INT,
      @EmployeeName NVARCHAR(50),
      @Username NVARCHAR(50),
      @Password NVARCHAR(50),
      @UserType NVARCHAR(25)
AS
BEGIN
      SET NOCOUNT ON;
      UPDATE dbo.Employees
      SET EmployeeID = @EmployeeID,
            EmployeeName = @EmployeeName,
            Username = @Username,
            Password = @Password,
            UserType = @UserType
      WHERE ID = @ID
END
```

After creating the stored procedures, we are now going to go back to Visual Studio IDE to add another class. Under DataAccessLayer folder, create a new class called AppInfo.cs and overwrite the default code with the one below:

```
namespace Inventory.DataAccessLayer
{
    static class AppInfo
    {
        public static int CurrentUserID = -1;
        public static string CurrentUserName = "";
        public static string LocalComputerName = "";
        public static bool CurrentUserIsAdmin;
    }
}
```

The class is **static**. It means it is shareable in the whole project. The variables are declared public static also. These global variables are going to be used in our Login Form and in other classes. Next we add a new class called UserDataAccess class and still under DataAccessLayer folder. UserDataAccess classes consists of a function and procedures. A DataAdapter function is created for CRUD operation. DataAdapter is part of ADO.Net. Please paste the code below after adding the UserDataAccess class and replacing the default code.

```
using System;
using System.Data;
using System.Windows.Forms;
using System.Data.SqlClient;
using Inventory.Configuration;

namespace Inventory.DataAccessLayer
{
    class UserDataAccess
    {
        #region "Get Procedures"
        public static string GetUserName(int employeeId)
        {
            var user = string.Empty;

            try
            {
                using (var con = new SqlConnection(Config.Instance.Connection))
                {
                    using (var cmd = new SqlCommand())
                    {
                        con.Open();
                        cmd.CommandText = "SELECT EmployeeName FROM dbo.Employees WHERE EmployeeID =
@EmployeeID";
                        cmd.CommandType = CommandType.Text;
                        cmd.Connection = con;
                        cmd.CommandTimeout = 120;
                        cmd.Parameters.Add(new SqlParameter("@EmployeeID", SqlDbType.Int,
100)).Value = employeeId;

                        var rowId = cmd.ExecuteScalar();
                        user = (string)rowId;
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

            return user;
        }

        #endregion

        #region "User Verification"

        public static int VerifyCredentials(string username, string password)
        {
            var id = 0;

            try
            {
                using (var con = new SqlConnection(Config.Instance.Connection))
                {
                    using (var cmd = new SqlCommand())
                    {
                        con.Open();
                        cmd.CommandText = "SELECT EmployeeID FROM dbo.Employees WHERE Username =
@Username And Password = @Password";
                        cmd.CommandType = CommandType.Text;
                        cmd.Connection = con;
                        cmd.CommandTimeout = 120;
                        cmd.Parameters.Add(new SqlParameter("@Username", SqlDbType.NVarChar,
255)).Value = username;
                        cmd.Parameters.Add(new SqlParameter("@Password", SqlDbType.NVarChar,
255)).Value = password;

                        var userId = cmd.ExecuteScalar();

                        if (userId == null)
                        {
                            id = 0;
                        }
                        else
                        {
                            id = (Int32)userId;
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

            return id; // Return the ID
        }

        public static void LogIn()
        {
            try
            {
                using (var con = new SqlConnection(Config.Instance.Connection))
                {
                    con.Open();

                    if (IsStationExist())
                    {
                        using (var cmdUpdate = new SqlCommand())
                        {
                            cmdUpdate.CommandText = "UPDATE dbo.LogStatus Set LogStatusID =
@LogStatusID WHERE (EmployeeID = @EmployeeID AND StationName = @StationName)";
                            cmdUpdate.CommandType = CommandType.Text;
                            cmdUpdate.Connection = con;
                            cmdUpdate.CommandTimeout = 120;
                            cmdUpdate.Parameters.Add(new SqlParameter("@EmployeeID", SqlDbType.Int,
100)).Value = AppInfo.CurrentUserID;
                            cmdUpdate.Parameters.Add(new SqlParameter("@StationName",
SqlDbType.NVarChar, 100)).Value = AppInfo.LocalComputerName;
                            cmdUpdate.Parameters.Add(new SqlParameter("@LogStatusID", SqlDbType.Int,
100)).Value = 1;
                            cmdUpdate.ExecuteNonQuery();
                        }
                    }
                    else
                    {
                        using (var cmdInsert = new SqlCommand())
                        {
                            cmdInsert.CommandText = "INSERT dbo.LogStatus (EmployeeID, StationName,
LogStatusID) VALUES  (@EmployeeID, @StationName, @LogStatusID)";
                            cmdInsert.CommandType = CommandType.Text;
                            cmdInsert.Connection = con;
                            cmdInsert.CommandTimeout = 120;
                            cmdInsert.Parameters.Add(new SqlParameter("@EmployeeID", SqlDbType.Int,
100)).Value = AppInfo.CurrentUserID;
                            cmdInsert.Parameters.Add(new SqlParameter("@StationName",
SqlDbType.NVarChar, 100)).Value = AppInfo.LocalComputerName;
                            cmdInsert.Parameters.Add(new SqlParameter("@LogStatusID", SqlDbType.Int,
100)).Value = 1;
                            cmdInsert.ExecuteNonQuery();
                        }
                    }

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

        public static void LogOut()
        {
            try
            {
                using (var con = new SqlConnection(Config.Instance.Connection))
                {
                    using (var cmd = new SqlCommand())
                    {
                        con.Open();
                        cmd.CommandText = "UPDATE dbo.LogStatus Set LogStatusID = @LogStatusID WHERE
(EmployeeID = @EmployeeID AND StationName = @StationName)";
                        cmd.CommandType = CommandType.Text;
                        cmd.Connection = con;
                        cmd.CommandTimeout = 120;
                        cmd.Parameters.Add(new SqlParameter("@EmployeeID", SqlDbType.Int,
100)).Value = AppInfo.CurrentUserID;
                        cmd.Parameters.Add(new SqlParameter("@StationName", SqlDbType.NVarChar,
100)).Value = AppInfo.LocalComputerName;
                        cmd.Parameters.Add(new SqlParameter("@LogStatusID", SqlDbType.Int,
100)).Value = 0;
                        cmd.ExecuteNonQuery();
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public static bool IsStationExist()
        {
            var status = false;

            try
            {
                using (var con = new SqlConnection(Config.Instance.Connection))
                {
                    using (var cmd = new SqlCommand())
                    {
                        con.Open();
                        cmd.CommandText = "SELECT EmployeeID FROM dbo.LogStatus WHERE (EmployeeID =
@EmployeeID AND StationName = @StationName)";
                        cmd.CommandType = CommandType.Text;
                        cmd.Connection = con;
                        cmd.CommandTimeout = 120;
                        cmd.Parameters.Add(new SqlParameter("@EmployeeID", SqlDbType.Int,
100)).Value = AppInfo.CurrentUserID;
                        cmd.Parameters.Add(new SqlParameter("@StationName", SqlDbType.NVarChar,
100)).Value = AppInfo.LocalComputerName;

                        var rowId = cmd.ExecuteScalar();
                        var rows = (Int32)rowId;

                        if (rows != 0)
                        {
                            status = true;
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

            return status; // Returning Boolean Value
        }

        public static bool IsAdmin(int employeeId)
        {
            var status = false;

            try
            {
                using (var con = new SqlConnection(Config.Instance.Connection))
                {
                    using (var cmd = new SqlCommand())
                    {
                        con.Open();
                        cmd.CommandText = "SELECT UserType FROM dbo.Employees WHERE EmployeeID =
@EmployeeID";
                        cmd.CommandType = CommandType.Text;
                        cmd.Connection = con;
                        cmd.CommandTimeout = 120;
                        cmd.Parameters.Add(new SqlParameter("@EmployeeID", SqlDbType.Int,
100)).Value = employeeId;

                        var right = cmd.ExecuteScalar();
                        var userRight = (String)right;

                        if (userRight == "Administrator")
                        {
                            status = true;
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

            return status;
        }

        #endregion

        #region "CRUD Procedures"

        public static SqlDataAdapter UserDA(SqlConnection dbConnection)
        {
            var da = new SqlDataAdapter();

            try
            {
                da.SelectCommand = new SqlCommand();
                da.SelectCommand.CommandText = "[usp_Employee_Select]";
                da.SelectCommand.CommandType = CommandType.StoredProcedure;
                da.SelectCommand.Connection = dbConnection;
                da.SelectCommand.CommandTimeout = 120;

                da.InsertCommand = new SqlCommand();
                da.InsertCommand.CommandText = "[usp_Employee_Insert]";
                da.InsertCommand.CommandType = CommandType.StoredProcedure;
                da.InsertCommand.Connection = dbConnection;
                da.InsertCommand.CommandTimeout = 120;
                da.InsertCommand.Parameters.Add(new SqlParameter("@EmployeeID", SqlDbType.Int, 100,
"EmployeeID"));
                da.InsertCommand.Parameters.Add(new SqlParameter("@EmployeeName",
SqlDbType.NVarChar, 50, "EmployeeName"));
                da.InsertCommand.Parameters.Add(new SqlParameter("@Username", SqlDbType.NVarChar,
50, "Username"));
                da.InsertCommand.Parameters.Add(new SqlParameter("@Password", SqlDbType.NVarChar,
50, "Password"));
                da.InsertCommand.Parameters.Add(new SqlParameter("@UserType", SqlDbType.NVarChar,
25, "UserType"));

                da.DeleteCommand = new SqlCommand();
                da.DeleteCommand.CommandText = "[usp_Employee_Delete]";
                da.DeleteCommand.CommandType = CommandType.StoredProcedure;
                da.DeleteCommand.Connection = dbConnection;
                da.DeleteCommand.CommandTimeout = 120;
                da.DeleteCommand.Parameters.Add(new SqlParameter("@ID", SqlDbType.Int, 100, "ID"));

                da.UpdateCommand = new SqlCommand();
                da.UpdateCommand.CommandText = "[usp_Employee_Update]";
                da.UpdateCommand.CommandType = CommandType.StoredProcedure;
                da.UpdateCommand.Connection = dbConnection;
                da.UpdateCommand.CommandTimeout = 120;
                da.UpdateCommand.Parameters.Add(new SqlParameter("@ID", SqlDbType.Int, 100, "ID"));
                da.InsertCommand.Parameters.Add(new SqlParameter("@EmployeeID", SqlDbType.Int, 100,
"EmployeeID"));
                da.InsertCommand.Parameters.Add(new SqlParameter("@EmployeeName",
SqlDbType.NVarChar, 50, "EmployeeName"));
                da.InsertCommand.Parameters.Add(new SqlParameter("@Username", SqlDbType.NVarChar,
50, "Username"));
                da.InsertCommand.Parameters.Add(new SqlParameter("@Password", SqlDbType.NVarChar,
50, "Password"));
                da.InsertCommand.Parameters.Add(new SqlParameter("@UserType", SqlDbType.NVarChar,
25, "UserType"));
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

            return da; // Return Data Adapter
        }

        #endregion
    }
}
```

The first code of the UserDataAccess class access a Data SQL Client using directive since we are using SQL Server database (using System.Data.SqlClient), the namespace of the class for our connection string (using Inventory.Configuration), and the rest that are needed for the procedures and functions of this class.

You have notice this code below that we use from the procedure and function that we created:

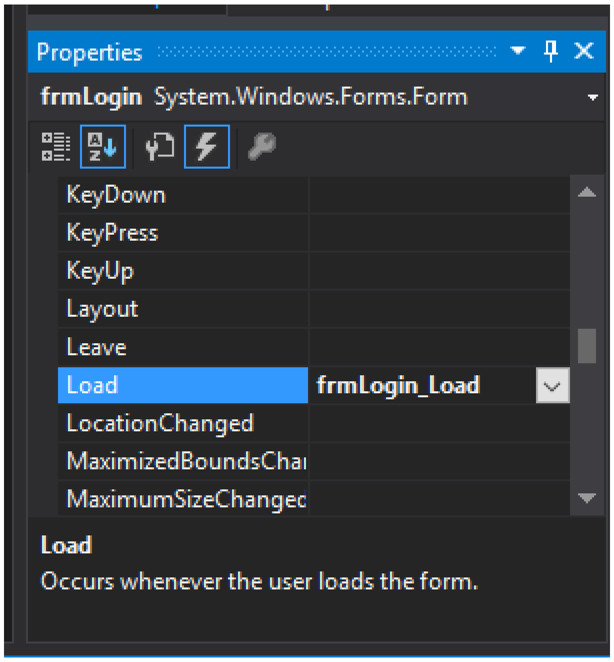
```
using (var con = new SqlConnection(Config.Instance.Connection))
```

The code is declaring a SqlConnection variable in C# called con and getting the connection string from the Config class that we created earlier in our project. This is really needed when you establish a connection from the database. Whether in a function or a procedure (private/public). Our procedure uses also SQLCommand after declaring a connection for the database. **SqlCommand** deals with databases. It executes SQL commands on a database. It sends an SQL command to a database that is specified by an SqlConnection object. We then call instance methods to physically apply the command to the database. For more information please refer to this link - http://www.dotnetperls.com/sqlcommand

Go to the frmLogin, and declare a private variable ds for the DataSet and da as SQLDataAdapter just after the class name. Like the following code below:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;
using Inventory.Configuration;
using Inventory.DataAccessLayer;

namespace Inventory.PresentationLayer
{
    public partial class frmLogin : Form
    {
        private DataSet ds;
        private SqlDataAdapter da;
```

Next enter the code for the Login form load event. You can access the events for the form or any control by clicking the event icon of the properties window like the one below:



Load: frmLogin_Load

Load
Occurs whenever the user loads the form.

Clicking the Load event will bring you to the code behind of the form under the Load Event. Enter the following code below under private void:

```
        private void frmLogin_Load(object sender, EventArgs e)
        {
            try
            {
                using (var con = new SqlConnection(Config.Instance.Connection))
                {
                    da = new SqlDataAdapter();
                    da = UserDataAccess.UserDA(con);
                    da.TableMappings.Add("Table", "Employees");
                    ds = new DataSet();
                    da.Fill(ds);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

This code will establish a connection from the database and instantiate the private variable for the SQLDataAdapter. It will then call the UserDataAccess class and the UserDA function that will return a SQLDataAdapter value. This will then add table mappings for our Employees table. Then this will instantiate a new DataSet from ds variable. The da variable will then fill in our DataSet. Our DataSet ds variable now has been loaded with the data from our Employees table.

Then create the code for the form's Shown event. The final code should look like this.

```
        private void frmLogin_Shown(object sender, EventArgs e)
        {
            try
            {
                txtPassword.Text = "";
                txtPassword.Multiline = false;
                txtPassword.UseSystemPasswordChar = false;
                cmbUsername.SelectedIndex = -1;
                cmbUsername.Focus();
                lblStatus.Text = "";
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Next create a new procedure that will fill our ComboBox control with data. Use the following code below:

```
private void BindComboBox()
        {
            try
            {
                using (var con = new SqlConnection(Config.Instance.Connection))
                {
                    using (var da = new SqlDataAdapter("SELECT * FROM dbo.Employees ORDER BY
EmployeeName", con))
                    {
                        da.TableMappings.Add("Table", "Employees");

                        using (var ds = new DataSet())
                        {
                            da.Fill(ds, "Employees");
                            cmbUsername.DataSource = ds.Tables["Employees"];
                            cmbUsername.DisplayMember = "Username";
                            cmbUsername.ValueMember = "Username";
                            cmbUsername.SelectedIndex = -1;
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Next provide a code for the ComboBox’s (cmbUsername) KeyDown, Enter, and DropDown events. Please follow the code below for these events:

```
private void cmbUsername_KeyDown(object sender, KeyEventArgs e)
        {
            try
            {
                using (var dvUser = new DataView(ds.Tables[0]))
                {
                    var strUsername = this.cmbUsername.Text;

                    if (e.KeyCode == Keys.Enter | e.KeyCode == Keys.Tab)
                    {
                        e.SuppressKeyPress = true;
                        dvUser.RowFilter = string.Format("Username='{0}'", strUsername);

                        if (dvUser.Count > 0)
                        {
                            txtPassword.Focus();
                        }
                        else
                        {
                            lblStatus.Text = "You must choose a Valid Username!";
                            cmbUsername.Focus();
                            cmbUsername.Select();
                            cmbUsername.Text = "";
                        }
                    }
                    else if (e.KeyCode == Keys.Back | e.KeyCode == Keys.Delete)
                    {
                        return;
                    }
                    else if (e.KeyCode == Keys.Escape)
                    {
                        cmbUsername.Text = "";
                        cmbUsername.SelectedIndex = -1;
                        cmbUsername.Focus();
                        cmbUsername.Select();
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void cmbUsername_Enter(object sender, EventArgs e)
        {
            try
            {
                BindComboBox();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

```
        private void cmbUsername_DropDown(object sender, EventArgs e)
        {
            try
            {
                BindComboBox();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

You should put this code in a region to organize the code. Like #region "ComboBox Procedures" for the cmbUsername control procedures. Then create the code for our OK button. Just double-click the OK button to bring you to the Click event of the button and enter the code under the event:

```
private void btnOK_Click(object sender, EventArgs e)
        {
            try
            {
                if (string.IsNullOrEmpty(cmbUsername.Text) | string.IsNullOrEmpty(txtPassword.Text))
                {
                    lblStatus.Text = "Please choose username and enter your password to login!";
                    cmbUsername.Focus();
                    cmbUsername.Select();
                    AppInfo.CurrentUserID = -1;
                }
                else
                {
                    // Get the userId or EmployeeID from the function that was created on
UserDataAccess class
                    var userId = UserDataAccess.VerifyCredentials(cmbUsername.Text,
txtPassword.Text);

                    if (userId > 0)
                    {
                        AppInfo.CurrentUserID = userId;
                        AppInfo.CurrentUserName = cmbUsername.Text;
                        AppInfo.LocalComputerName = Environment.MachineName;
                        AppInfo.CurrentUserIsAdmin = UserDataAccess.IsAdmin(userId);
                        UserDataAccess.LogIn();
                        this.Close();
                    }
                    else
                    {
                        lblStatus.Text = "Password is incorrect! Try again.";
                        txtPassword.Focus();
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

The OK button will verify of the user has access or not. It will call the `UserDataAccess.VerifyCredentials` function that we created earlier.

Next create the code for the cancel button. Follow the code below:

```
       private void btnCancel_Click(object sender, EventArgs e)
        {
            try
            {
                this.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Next create the code for the txtPassword_KeyDown event. Please follow the code below:

```
       #region "TextBox Procedures"

        private void txtPassword_KeyDown(object sender, KeyEventArgs e)
        {
            try
            {
                if (e.KeyCode == Keys.Enter | e.KeyCode == Keys.Tab)
                {
                    btnOK.Focus();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        #endregion
```

Now we are finished coding our Login form. We should now call this form on our Main Form. Now go back to our frmMain and at design time add a **StatusStrip** control and drag & drop it to our frmMain at design time. Name it **ssMain** as the control name. Then add a StatusLabel inside the ssMain control to the left and right side. Call it LBLBottomLeft and LBLBottomRight as the control name. Now double-click the form and add these procedures just right after the closing code of the frmMain_Load event. Like the code below:

```
       private void frmMain_Load(object sender, EventArgs e)
        {

        }

        #region "Private Procedures"

        private void ShowLoginForm()
        {
            try
            {
                var frm = new frmLogin();
                frm.btnOK.Click += ReEnableControls;
                frm.MdiParent = this;
                frm.Show();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public void SetFormState()
        {
            try
            {
                if (AppInfo.CurrentUserID == -1)
                {
                    LBLBottomLeft.Text = "";
                }
                else
                {
                    LBLBottomLeft.Text = AppInfo.CurrentUserName.ToUpper() +
(AppInfo.CurrentUserIsAdmin == true ? " - (Administrator)" : "");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public void EnableMenuItems(bool isEnabled)
        {
            try
            {
                if (isEnabled == false | AppInfo.CurrentUserID == -1)
                {
                    mnLogin.Enabled = true;
                    mnLogout.Enabled = false;
                    mnExit.Enabled = true;
                    mnTransaction.Enabled = false;
                    mnReport.Enabled = false;
                    mnSetup.Enabled = false;
                    mnLogout.Enabled = false;
                }
                else if (AppInfo.CurrentUserIsAdmin == false)
                {
                    mnLogin.Enabled = false;
                    mnLogout.Enabled = true;
                    mnExit.Enabled = false;
                    mnTransaction.Enabled = true;
                    mnSetup.Enabled = false;
                    mnReport.Enabled = true;
                }
                else if (AppInfo.CurrentUserIsAdmin == true)
                {
                    mnLogin.Enabled = false;
                    mnLogout.Enabled = true;
                    mnExit.Enabled = false;
                    mnTransaction.Enabled = true;
                    mnSetup.Enabled = true;
                    mnReport.Enabled = true;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public void EnableToolBarButtons(bool isEnabled)
        {
            try
            {
                if (isEnabled == false | AppInfo.CurrentUserID == -1)
                {
                    tsbLogin.Enabled = true;
                    tsbLogout.Enabled = false;
                    tsbInventory.Enabled = false;
                    tsbInquiry.Enabled = false;
                    tsbReport.Enabled = false;
                    tsbExit.Enabled = true;
                }
                else
                {
                    tsbLogin.Enabled = false;
                    tsbLogout.Enabled = true;
                    tsbInventory.Enabled = true;
                    tsbInquiry.Enabled = true;
                    tsbReport.Enabled = true;
                    tsbExit.Enabled = false;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public void EnableControls(bool isEnable)
        {
            try
            {
                EnableMenuItems(isEnable);
                EnableToolBarButtons(isEnable);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void ReEnableControls(object sender, EventArgs e)
        {
            try
            {
                if (AppInfo.CurrentUserID != -1)
                {
                    EnableControls(true);
                    SetFormState();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
        private void CloseAllActiveForms()
        {
            try
            {
                foreach (var frm in this.MdiChildren)
                {
                    if (!frm.Focused)
                    {
                        frm.Close();
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        #endregion
```

Now under the Application Control menu, double click the Login menu to bring you to the mnLogin_Click event and enter the code just like below:

```
        #region "Menu Procedures"
        private void mnLogin_Click(object sender, EventArgs e)
        {
            try
            {
                ShowLoginForm();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        #endregion
```

Put the code in a region to organize it. Next add a code for the tsbLogin control's click event. Double click the control at design time and enter the code below:

```
      #region "ToolStripButton Procedures"

        private void tsbLogin_Click(object sender, EventArgs e)
        {
            try
            {
                ShowLoginForm();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        #endregion
```

Do the same for the Logout menu's click event. Double-click the Logout menu under the Application Control menu and enter the code below:

```
        private void mnLogout_Click(object sender, EventArgs e)
        {
            try
            {
                EnableControls(false);
                CloseAllActiveForms();
                ShowLoginForm();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Place this code under the region "Menu Procedures" just before the #endregion code. Now add a click event for the tsbLogout control. Double-click the tsbLogout control to bring you to the click event of the control and enter the code below:

```
        private void tsbLogout_Click(object sender, EventArgs e)
        {
            try
            {
                EnableControls(false);
                CloseAllActiveForms();
                ShowLoginForm();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Move the code to the "ToolStripButton Procedures" just before the #endregion code. Now add the code for our tsbExit click event. Double-click the control and add the code just like below:

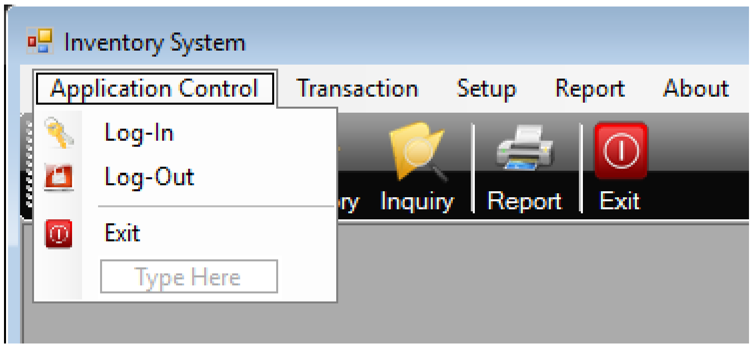
```
        private void tsbExit_Click(object sender, EventArgs e)
        {
            try
            {
                if ((MessageBox.Show("Are you sure you want to exit?", "Confirmation",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No))
                {
                    return;
                }
                else
                {
                    this.Close();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Move this code to its proper region.

Next add code for the mnExit under Application Control menu. Double-click the mnExit menu and enter the code below:

```
        private void mnExit_Click(object sender, EventArgs e)
        {
            try
            {
                if ((MessageBox.Show("Are you sure you want to exit?", "Confirmation",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No))
                {
                    return;
                }
                else
                {
                    this.Close();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

Next move this code to its proper region. Now let's add an image to our Login, Logout and Exit menu just like the image below:



Next let's add a code for our **"frmMain"** form's Load event and a code to disable the X or close button of this form. Enter the code below under a region:

```
#region "Form Procedures"

        private void frmMain_Load(object sender, EventArgs e)
        {
            try
            {
                EnableControls(false);
                ShowLoginForm();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        // Procedure to disable right above X button on this form
        protected override CreateParams CreateParams
        {
            get
            {
                var IsClose = false;
                var CloseButton = 0x200;
                var Param = base.CreateParams;

                if (IsClose)
                {
                    Param.ClassStyle = Param.ClassStyle & CloseButton;
                }
                else
                {
                    Param.ClassStyle = Param.ClassStyle | CloseButton;
                }
                return Param;
            }
        }

        #endregion
```

The using directive for this form should only be the following:

```
using System;
using System.Windows.Forms;
using Inventory.DataAccessLayer;
```

Remove the other using directive that is not needed by the form's procedure and methods.

Next open the Program.cs file from our Solution. This is after the App.config file. After opening, the code should look like this below:

```
using System;
using System.Windows.Forms;
using Inventory.PresentationLayer;

namespace Inventory
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmMain());
        }
    }
}
```

The Main form is our entry form and will be the first one to load.

Before you run the program, please enter a data on your Employees table. Enter your name and other details. Go back to your Inventory database and open the Employees table by right-clicking and clicking **Edit Top 200 Rows.** Then enter your name. This will be used to test our Login Form.

Now save all changes, build and run your program by clicking the Start button on our Visual Studio IDE.