Experiment 1:
An Islamic Banking Application using C# & SQL Server

Learning Objectives

Upon completion of this lab, you will be able to:

- Create database on SQL server 2005
- Create tables and Views
- Create relationship between tables
- Write SQL quarries and run it
- Define SQL service.
- Define Database file (mdf and ldf files)

Overview of Relational Databases and Database Applications

In 1970s, the *relational database model* which originated in the academic research community became available in commercial implementations such as IBM DB2 and Oracle. The relational data model specifies data stored in *relations* that have some *relationships* among them (hence the name *relational*).

In relational databases such as Sybase, Oracle, IBM DB2, MS SQL Server and MS Access, data is stored in *tables* made up of one or more *columns* (Access calls a column a *field*). The data stored in each column must be of a single *data type* such as Character, Number or Date. A collection of values from each column of a table is called a *record* or a *row* in the table. Different tables can have the same column in common. This feature is used to explicitly specify a relationship between two tables. Values appearing in column A in one table are shared with another table.

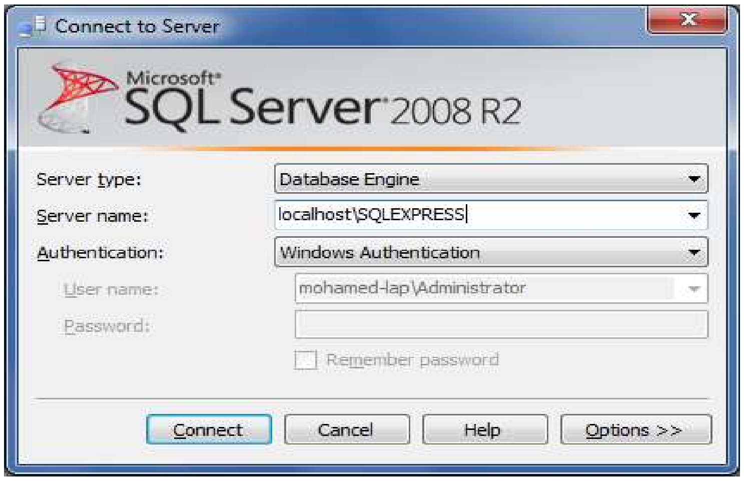
Task 1: Create Database

Step 1: From **start menu – All Programs - Microsoft SQL Server** run **SQL Server Management Studio.** At the **connect to server** window shown in Figure 1.1 and enter the following parameters :

Server name: <PCName>\SQLEXPRESS

Authentication: Windows Authentication

Then press connect



Server type: Database Engine
Server name: localhost\SQLEXPRESS
Authentication: Windows Authentication
User name: mohamed-lap\Administrator
Password:

Figure 1.1 server window

Step 2: From **Objet Explorer** window **Right click** on **Database,** and then select **New Database** as shown in Figure 1.2

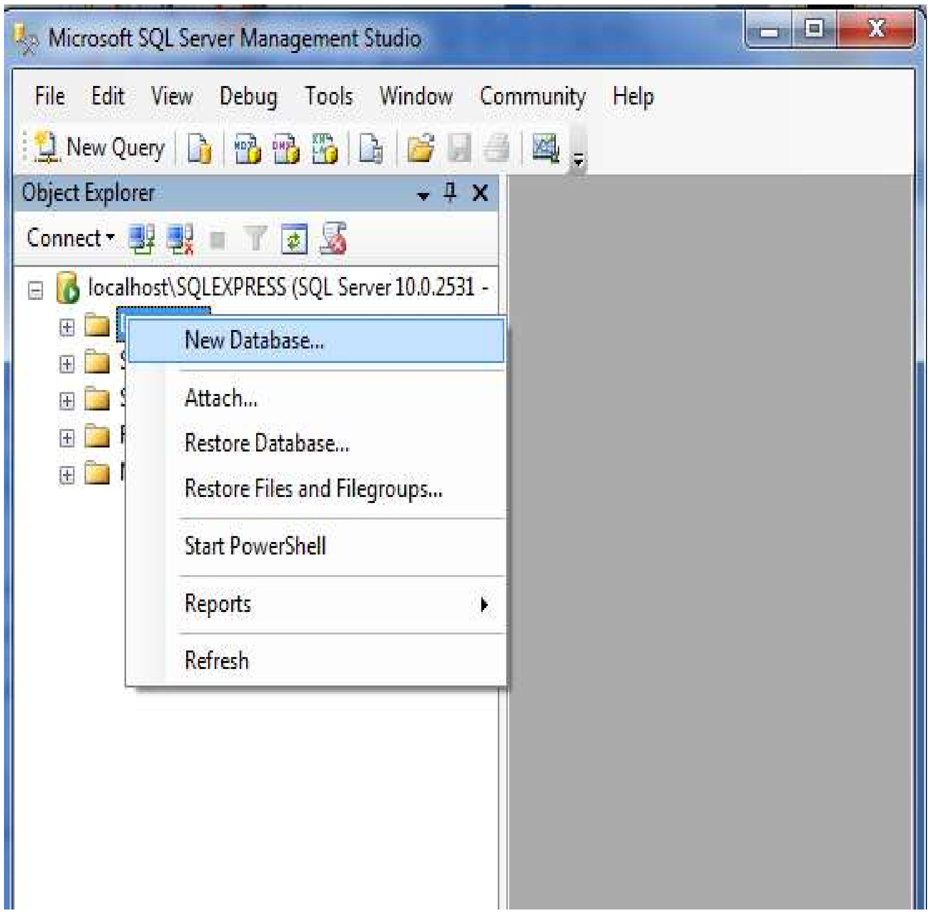
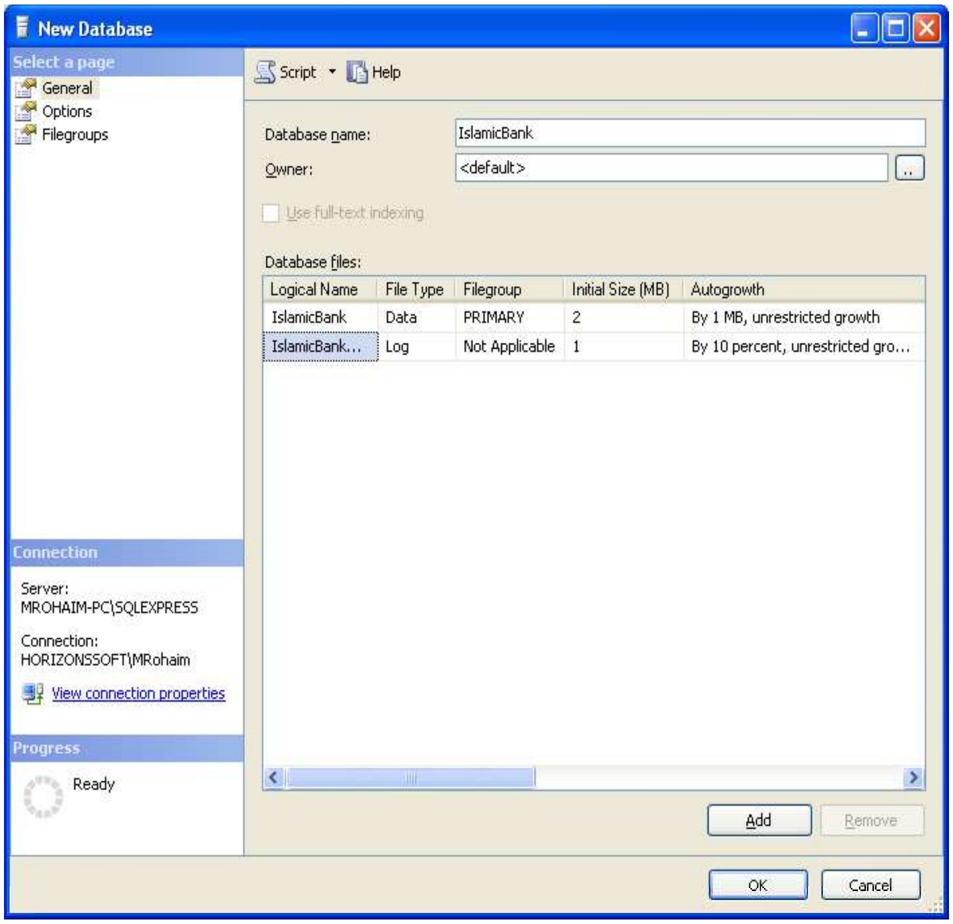

Microsoft SQL Server Management Studio
File Edit View Debug Tools Window Community Help
New Query
Object Explorer
Connect
localhost\SQLEXPRESS (SQL Server 10.0.2531 -
New Database...
Attach...
Restore Database...
Restore Files and Filegroups...
Start PowerShell
Reports
Refresh

Figure 1.2 Select a new Database

Step 3: From **New Database** window enter **database name** as **IslamicBank as** shown in Figure 1.3. Then click **OK**



Database name: IslamicBank

Owner: <default>

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth
IslamicBank	Data	PRIMARY	2	By 1 MB, unrestricted growth
IslamicBank...	Log	Not Applicable	1	By 10 percent, unrestricted gro...

Connection

Server: MROHAIM-PC\SQLEXPRESS

Connection: HORIZONSSOFT\MRohaim

Figure 1.3. **Select IslamicBank**

<u>Task 2: Create Tables</u>

In this task we will create the following two tables:

- **Customer Table** Figure 1.4, and
- **Accounts Table** Figure 1.5

CustomerID	Name	Address	City	Zip
Number	***Character***	***Character***	***Character***	***Character***
1001	Ahmed Mohamed	Nasr city	Cairo	91232
1002	Mohamed Hosny	Tanta	Elgharbia	81992
1003	Mostafa Taha	Dmnhor	Elbehira	81992

Figure 1.4, **Customer Table**

CustomerID	Account Number	Account Type	DateOpened	Balance	Currency
Number	***Number***	***Character***	***Date***	***Number***	***Character***
1001	9987	Checking	10/12/1989	4000.00	EGP
1001	9980	Savings	10/12/1989	2000.00	USD
1002	8811	Savings	01/05/1992	1000.00	USD
1003	4422	Checking	12/01/1994	6000.00	EGP
1003	4433	Savings	12/01/1994	9000.00	USD

Figure 1.5 **Accounts Table**

To Create the Customer Table, do the following steps:

Step 1: Under **IslamicBank** database **right click** on **Tables** then select **New Table** as shown in Figure 1.6

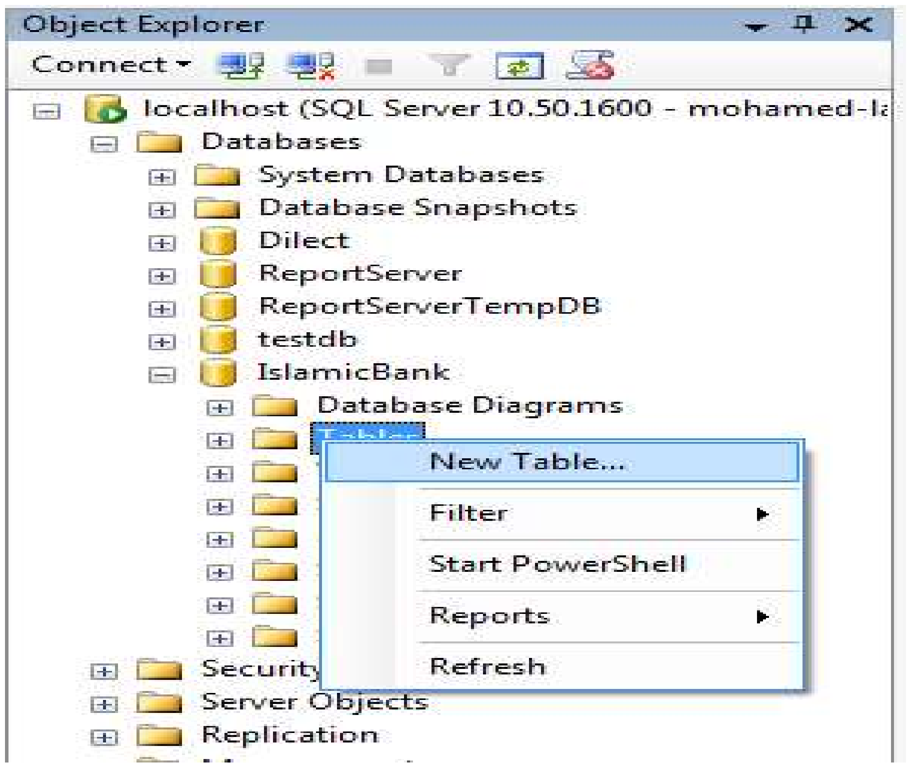
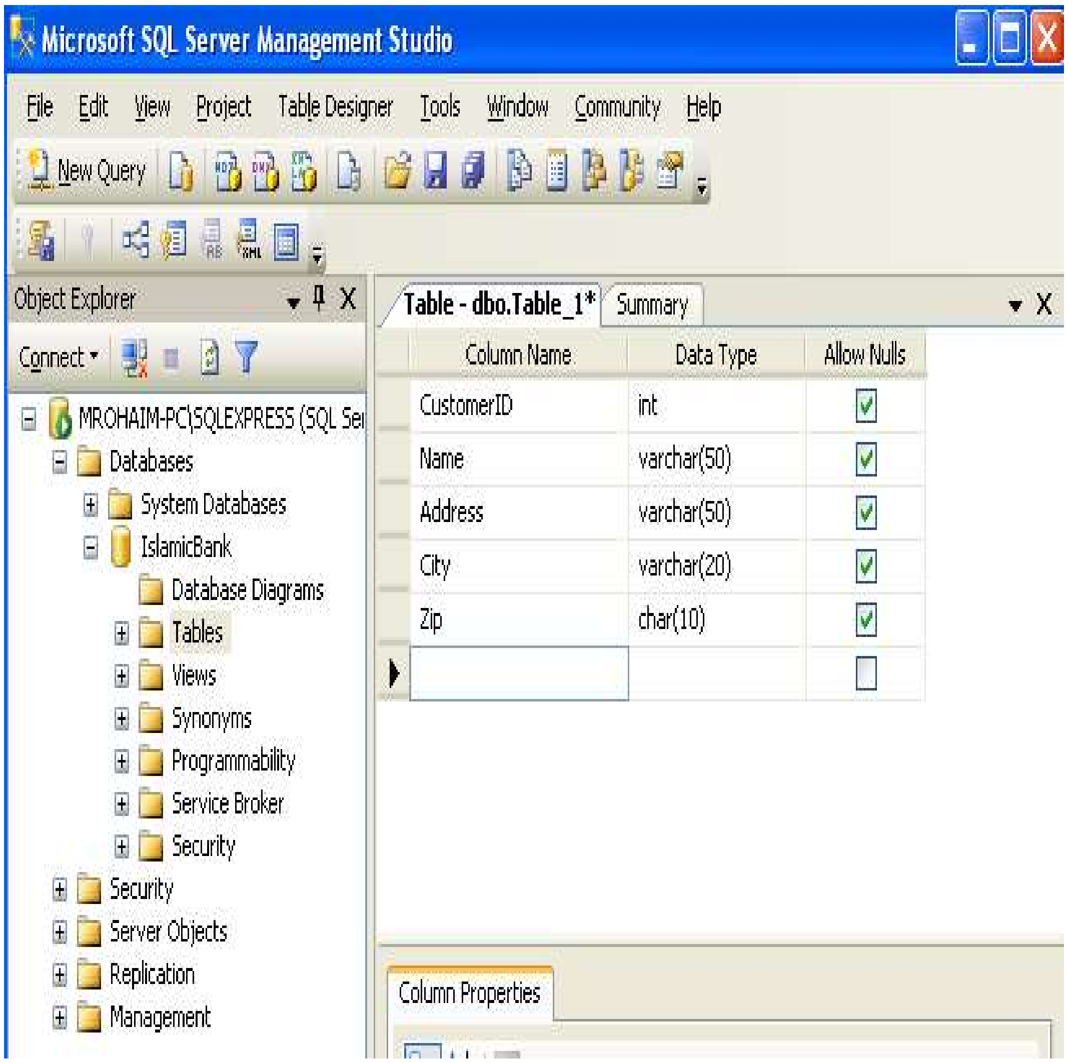


Figure 1.6 Select **New Table**

Step 2: Define customer table fields as shown in the Figure 1.7,

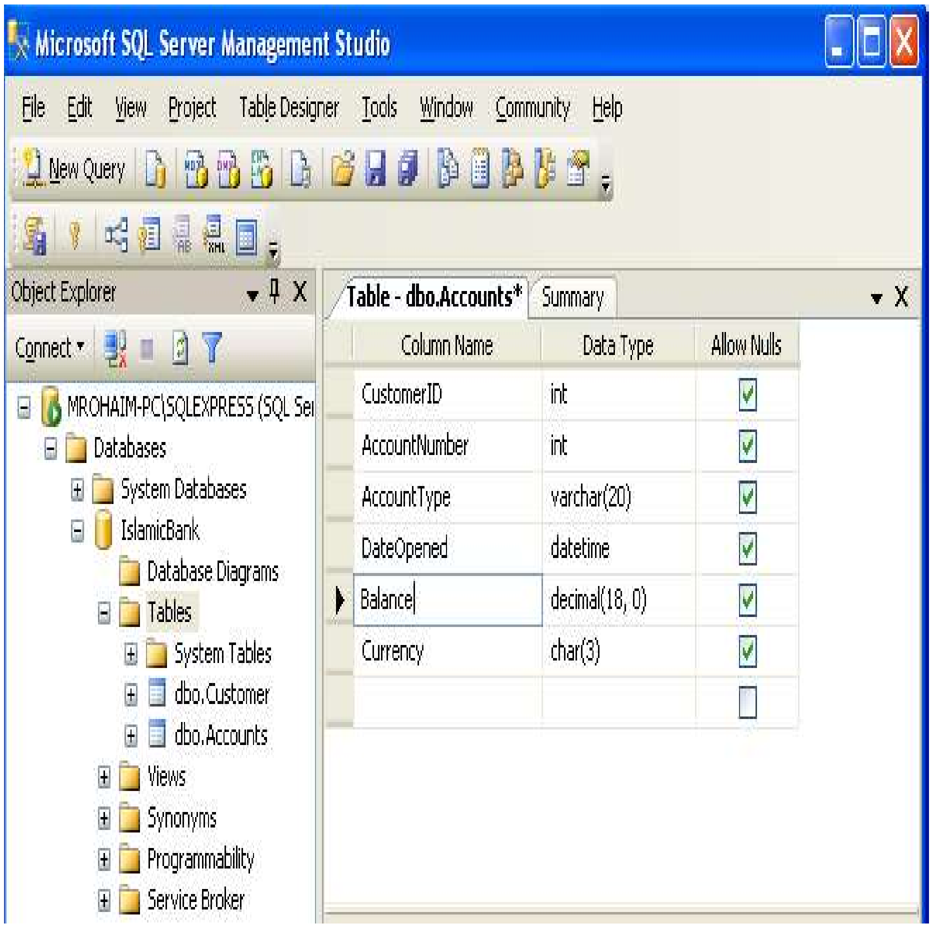


Column Name	Data Type	Allow Nulls
CustomerID	int	✓
Name	varchar(50)	✓
Address	varchar(50)	✓
City	varchar(20)	✓
Zip	char(10)	✓
		☐

Figure 1.7, Definition of customer table fields

Step 3: Save the created table as **Customer**

To Create the Accounts Table, do the same steps. Define it as shown in Figure 1.8. and save it.



Column Name	Data Type	Allow Nulls
CustomerID	int	✓
AccountNumber	int	✓
AccountType	varchar(20)	✓
DateOpened	datetime	✓
Balance	decimal(18, 0)	✓
Currency	char(3)	✓
		☐

Figure 1.8. Definition of Accounts table

Task 3: Create Views

Objective: Create a view on the available tables.

Assume we want to create a view contains the columns shown in Figure 1.9. from the two tables, Customer and Accounts.

Column Name	Table
CustomerID	Customer
Name	Customer
AccountNumber	Accounts
AccountType	Accounts
Balance	Accounts
Currency	Accounts

Figure 1.9. Required fields in the view to be created

Step 1: Under **IslamicBank** database **right click** on **Views** then select **New View.** An Add Table window will be displayed to select the tables which contain the required fields **in the required view.**

Step 2: Select the required tables and click Add as shown in Figure 1.10. Then a view window will result

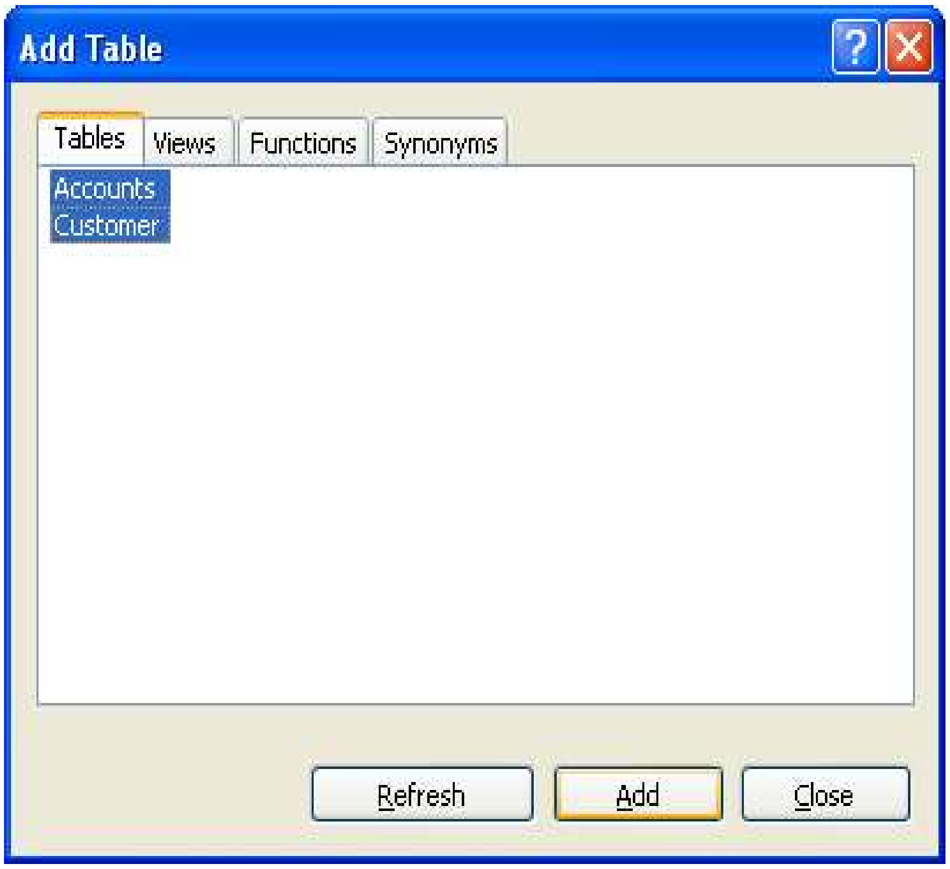
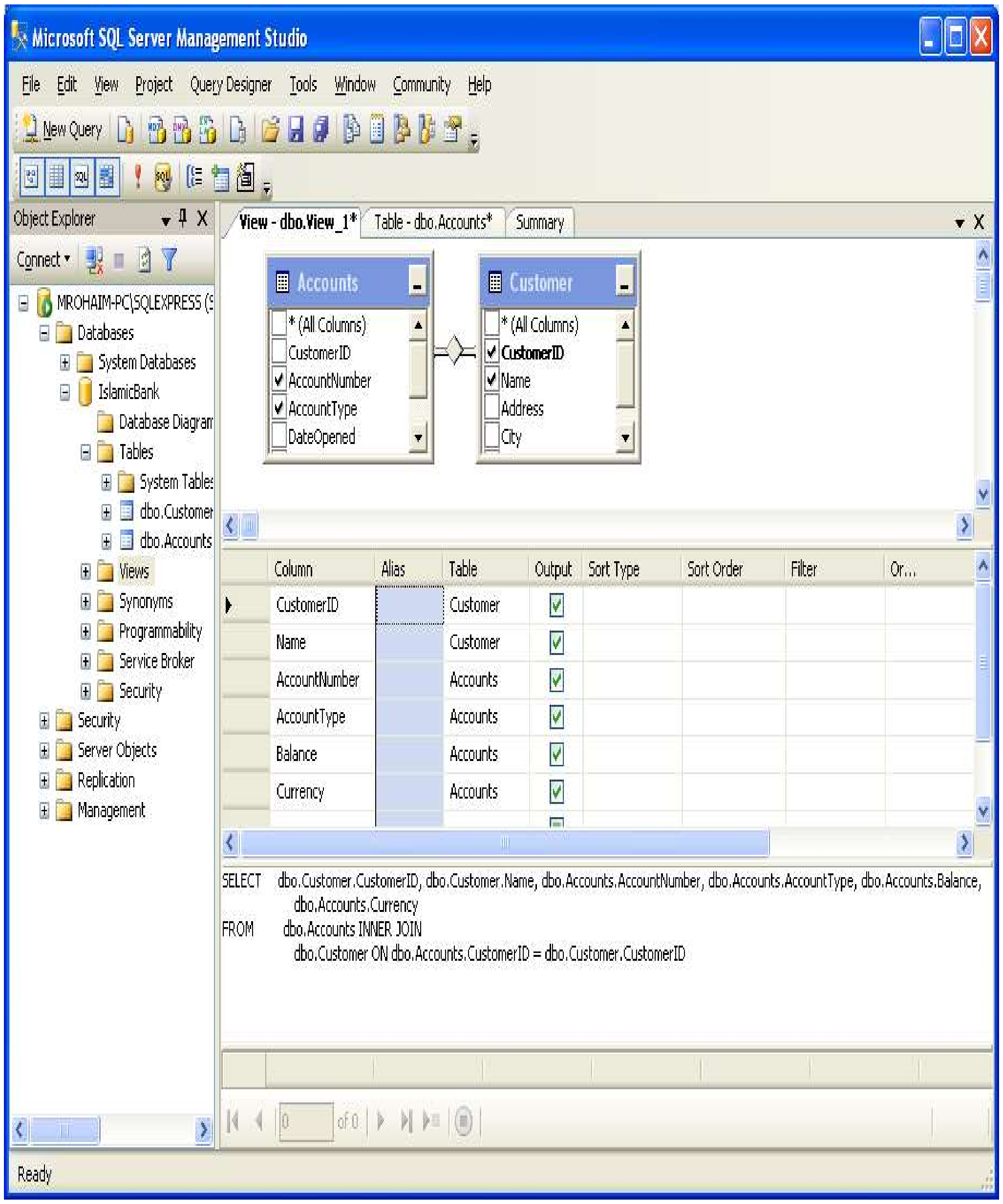


Figure 1.10**.** Addition of the required Tables

Step 3: From view window select required columns then save this view as **CustomerAccounts**



Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...
CustomerID		Customer	✓				
Name		Customer	✓				
AccountNumber		Accounts	✓				
AccountType		Accounts	✓				
Balance		Accounts	✓				
Currency		Accounts	✓				

```
SELECT   dbo.Customer.CustomerID, dbo.Customer.Name, dbo.Accounts.AccountNumber, dbo.Accounts.AccountType, dbo.Accounts.Balance,
           dbo.Accounts.Currency
FROM     dbo.Accounts INNER JOIN
           dbo.Customer ON dbo.Accounts.CustomerID = dbo.Customer.CustomerID
```

Note That:-

1- The relation between the two tables is donning using **CustomerID** field.

2- SQL statement for this view is:

```
SELECT  dbo.Customer.CustomerID,
dbo.Customer.Name,
dbo.Accounts.   AccountNumber,
dbo.Accounts.AccountType, dbo.Accounts.Balance,
dbo.Accounts.Currency
FROM  dbo.Accounts INNER JOIN dbo.Customer   ON
dbo.Accounts.CustomerID = dbo.Customer.CustomerID
```

Step 3: Run this view to show its result and write this result as follow:

<u>Exercise:</u>

Do the following SQL statement and write their meaning and their results

1 – <u>Select Statement</u>

Select *   From Customer   Where   Name = 'Ahmed'

Select *   From Customer   Where   Name   Like '%Ahmed% '

Select *   From Accounts   Where   Balance > 1000 and   Balance   < 8000

Select Currency, sum(Balance) As sumOf Balance From Accounts Grouped By Currency

Select AccountType , Currency , sum(Balance) As sumOf Balance From Accounts Grouped By Currency

Select AccountType , Currency , sum(Balance) As sumOf Balance From Accounts Grouped By AccountType , Currency

```
Select *   From CustomerAccounts
```

2 – Insert Statement

```
insert into Customers  Values (1004, 'Mohamed', 'Nasr
City', 'Cairo',  '1234')
```

```
insert into Customers  Values (1005, 'Ramy', 'Nasr City',
'Cairo')
```

```
insert into Customers  Values (1005, 'Ramy', 'Nasr City',
'Cairo', NULL)
```

```
insert into Customers  (CustomerID, Name, Address,
City)  Values (1005, 'Ramy', 'Nasr City', 'Cairo')
```

3 – <u>Update Statement</u>

```
update Accounts set Balance Balance * 1.5
```

```
update Accounts set Balance Balance * 2
Where CustomerID = 1001
```

4 – <u>Delete Statement</u>

```
Delete from Accounts Where CustomerID = 1001
```

Experiment 2
An Islamic Banking Application (2)
using C# & SQL Server

Learning Objectives

Upon completion of this lab, you will be able to:

- Create Islamic Bank application using C#
- How to use ADO.NET to connect to SQL Server database
- How to retrieve data from SQL Server database.

Brief overview of ADO.NET Object Model

A programming language connects to and interacts with a relational database via a database interface software that facilitates communication between a database management system and a program. C# programs communicate with databases and manipulate their data through ADO.NET.

Experiment 2:

ADO.NET was created for the .NET framework to replace Microsoft's ActiveX Data Objects™ (ADO) technology. The ADO.NET object model provides an API for accessing database systems programmatically.

Namespace System.Data is the root namespace for the ADO.NET API. The other important ADO.NET namespaces, System.Data.OleDb and System.Data.SqlClient, contain classes that enable programs to connect with and manipulate data sourceslocations that contain data, such as a database or an XML file. Namespace System.Data.OleDb contains classes that are designed to work with any data source, whereas System.Data.SqlClient contains classes that are optimized to work with Microsoft SQL Server databases.

An object of class SqlConnection (namespace System.Data.SqlClient) represents a connection to a data sourcespecifically a SQL Server database. A SqlConnection object keeps track of the location of the data source and any settings that specify how the data source is to be accessed. A connection is either active (i.e., open and permitting data to be sent to and retrieved from the data source) or closed.

An object of class SqlCommand (namespace System.Data.SqlClient) represents a SQL command that a DBMS can execute on a database. A program can use SqlCommand objects to manipulate a data source through a SqlConnection. The program must open the connection to the data source before executing one or more SqlCommands and close the connection once no further access to the data source is required. A connection that remains active for some length of time to permit multiple data operations is known as a persistent connection.

Class DataTable (namespace System.Data) represents a table of data. A DataTable contains a collection of DataRows that represent the table's data. A DataTable also has a collection of DataColumns that describe the columns in a table. DataRow and DataColumn are both located in namespace System.Data. An object of class System.Data.DataSet, which consists of a set of DataTables and the relationships among them, represents a cache of datadata that a program stores temporarily in local memory. The structure of a *DataSet* mimics the structure of a relational database.

ADO.NET's Disconnected Model

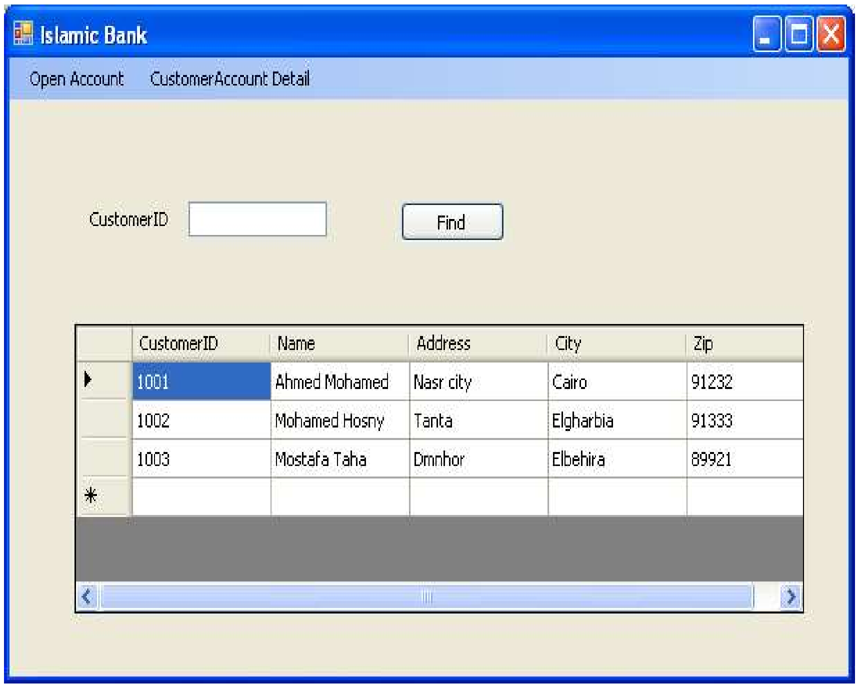
An advantage of using class DataSet is that it is disconnected the program does not need a persistent connection to the data source to work with data in a DataSet. Instead, the program connects to the data source to populate the DataSet (i.e., fill the DataSet's DataTables with data), but disconnects from the data source immediately after retrieving the desired data. The program then accesses and potentially manipulates the data stored in the DataSet. The program operates on this local cache of data, rather than the original data in the data source. If the program makes changes to the data in the DataSet that need to be permanently saved in the data source, the program reconnects to the data source to perform an update then disconnects promptly. Thus the program does not require any active, persistent connection to the data source.

An object of class SqlDataAdapter (namespace System.Data.SqlClient) connects to a SQL Server data source and executes SQL statements to both populate a DataSet and update the data source based on the current contents of a DataSet. A SqlDataAdapter maintains a SqlConnection object that it opens and closes as needed to perform these operations using SqlCommands. We demonstrate populating DataSets and updating data sources later in this chapter.

Task : Build Islamic Bank Application Main window

In this task will build the main window of Islamic bank application as shown in Figure 2.1.

This main window display customers' details and contains an option to search for a specified customer by his ID. Also it contains two menu buttons (Open Account and Customer Account detail).



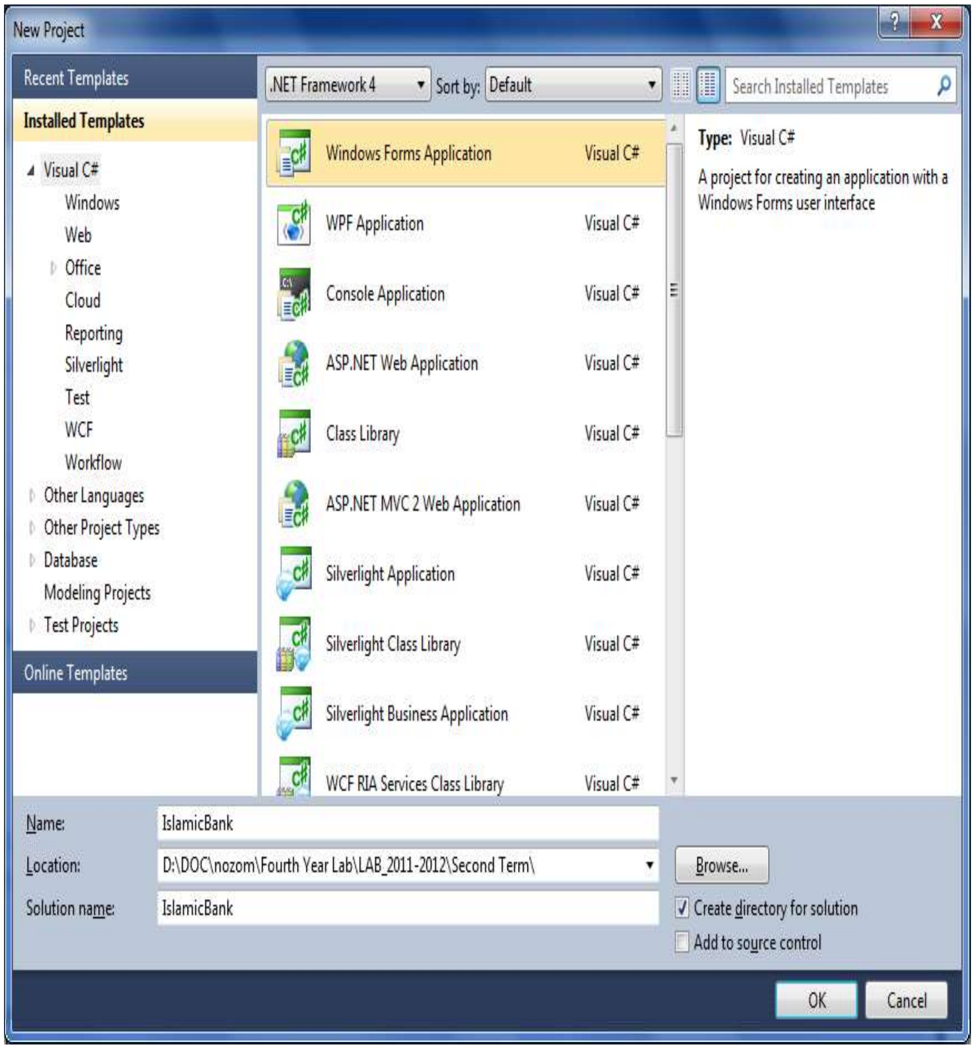
	CustomerID	Name	Address	City	Zip
▸	1001	Ahmed Mohamed	Nasr city	Cairo	91232
	1002	Mohamed Hosny	Tanta	Elgharbia	91333
	1003	Mostafa Taha	Dmnhor	Elbehira	89921
*					

Figure 2.1 Customers' details window

Open Account button opens a new form used to add account to customers. Customer Account Detail button opens a new form displays customer’s accounts. To build this application follow the following steps:

Step 1: Select **IslamicBank** Application as follow:

From **Start menu – All Programs -** run **Microsoft Visual Studio - Microsoft Visual Studio.** From **File** menu select **New Project**. At **New Project** window select **Windows Form Application** and its **Name** is **IslamicBank** as shown in Figure 2.2. Then press **OK**


New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Office
Cloud
Reporting
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Database
Modeling Projects
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
ASP.NET Web Application Visual C#
Class Library Visual C#
ASP.NET MVC 2 Web Application Visual C#
Silverlight Application Visual C#
Silverlight Class Library Visual C#
Silverlight Business Application Visual C#
WCF RIA Services Class Library Visual C#
Browse...
OK
Cancel

Type: Visual C#

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

Name: IslamicBank

Location: D:\DOC\nozom\Fourth Year Lab\LAB_2011-2012\Second Term\

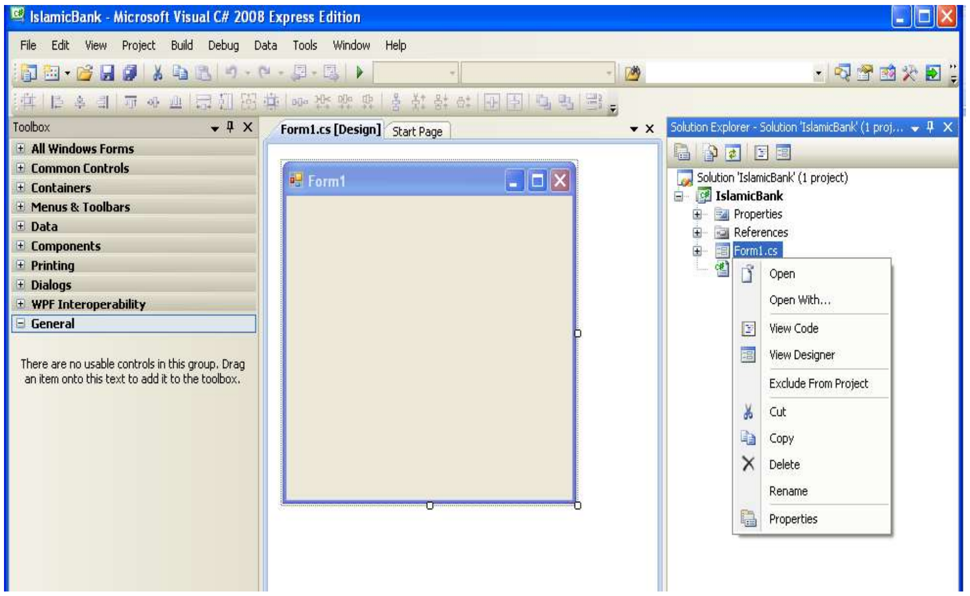
Solution name: IslamicBank

[✓] Create directory for solution

[ ] Add to source control

Figure 2.2 Selecting IslamicBank

Step 2: On **Solution Explore window** right click on **Form1.cs** file then select **Rename** and change its name to be **Customer**.



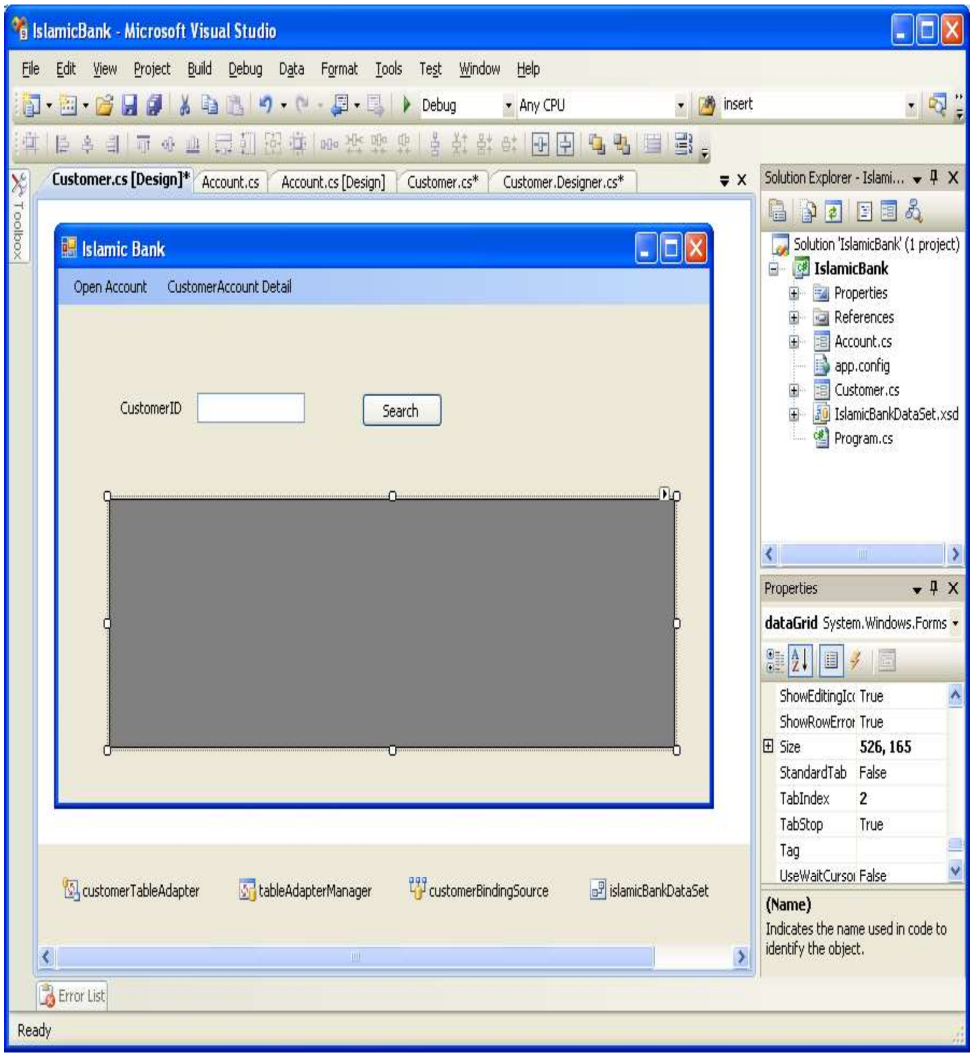
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Figure 2.3 Create Form1

Step 3: From the **Toolbox** window add the following controls

Control Type	**Properties**	
	Name	**Text**
MenuStrip	menuStrip1	menuStrip1
Label	label1	CustomerID
TextBox	txtCustomerID	
Button	buSearch	Search
DataGridView	dataGrid	

After adding these control the result will be like Figure 2.4.



ShowEditingIc	True
ShowRowError	True
Size	526, 165
StandardTab	False
TabIndex	2
TabStop	True
Tag	
UseWaitCursor	False

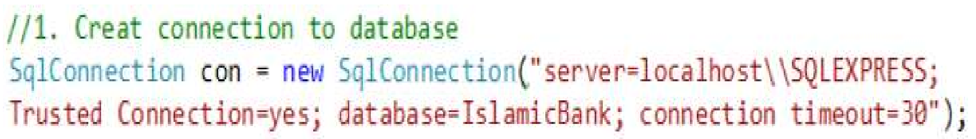
(Name)
Indicates the name used in code to identify the object.

Figure 2.4 Adding controls to the Form

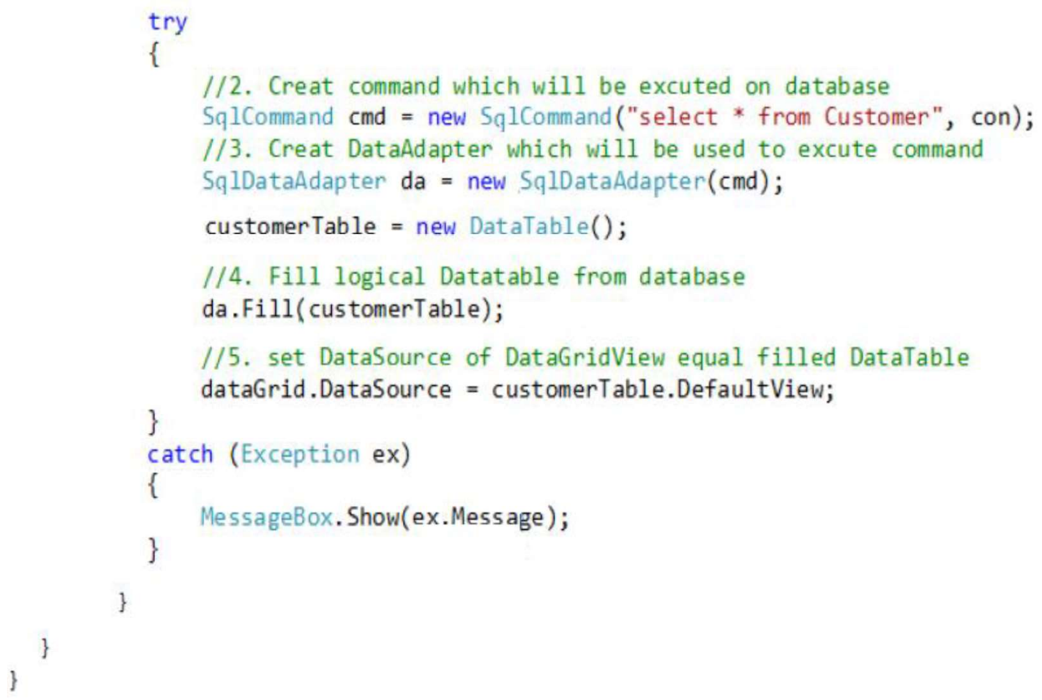
Step 4: Right click on the From then select **View Code**.

Then change the code to be like the following:

```
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.Data.SqlClient;
namespace IslamicBank
{
   public partial class Customer : Form
   {
      DataTable customerTable;
       public Customer()
      {
         InitializeComponent();
```



```
//1. Creat connection to database
SqlConnection con = new SqlConnection("server=localhost\\SQLEXPRESS;
Trusted_Connection=yes; database=IslamicBank; connection timeout=30");
```



```
          try
          {
              //2. Creat command which will be excuted on database
              SqlCommand cmd = new SqlCommand("select * from Customer", con);
              //3. Creat DataAdapter which will be used to excute command
              SqlDataAdapter da = new SqlDataAdapter(cmd);

              customerTable = new DataTable();

              //4. Fill logical Datatable from database
              da.Fill(customerTable);

              //5. set DataSource of DataGridView equal filled DataTable
              dataGrid.DataSource = customerTable.DefaultView;
          }
          catch (Exception ex)
          {
              MessageBox.Show(ex.Message);
          }

      }

   }
}
```

Step 5: **Double click** on **Search** button
then add the following code to its **Click** event

```
private void buSearch_Click(object sender, EventArgs e)
        {
          //Construct filter expression
          string exp = "CustomerID =" + txtCustomerID.Text;

          //Creat new DataView
          DataView customerView = new DataView();

          //set the DataView table equal customerTable
          customerView.Table = customerTable;

          if (txtCustomerID.Text != "")
          {
              //execute filter on DataView
              customerView.RowFilter = exp;
          }

          dataGrid.DataSource = customerView;

        }
```

Step 6: To run the application: from **Debug** menu
select **Start Debugging** or press **F5**