UNIT III APPLICATION DEVELOPMENT ON .NET

Building Windows Forms Applications -Windows Forms Controls – Simple Windows Forms Application with controls and Events - ADO.NET Introduction - ADO.NET Architecture : Connected and Disconnected Architecture – Data Provider:Connection Object,Command Object, Data Reader - Executing a Command using ExecuteNonQuery ,ExecuteScalar , ExecuteReader Datasets : DataTable , DataColumn , DataRow , DataConstraints , DataRelation , DataAdapters

BUILDING WINDOWS FORMS APPLICATIONS

Windows Forms offers a way to build Windows desktop applications using the .NET Framework. You develop applications with Windows Forms when you want the client Application to be responsible for much of the processing.

Windows Forms vs Web Forms

Features	Windows Forms	Web Forms
User Interfaces, data binding etc.	Easy to build	Difficult to build
Deployment and Maintenance	Complex. New versions of assemblies, configuration files, and other required files must be deployed on all client machines. Usually user interaction required.	Easy. Need to deploy assemblies and configuration files on the server only. Transparent to the client.
Performance	Faster	Slower
Robustness and Reliability	One client machine goes down, other users are still live.	Usually web servers are never down. However if the server goes down, all users are affected.
Network Congestion	Depending on the data transfer and connections made to the server from various clients.	Depends
Resources	Runs on the client machine.	Runs on a Web server.
Framework dependency	All client machines have to install required versions of .NET framework and other required libraries	Only server needs to have .NET framework and other required libraries.

WINDOWS FORMS CONTROLS

The commonly used controls are given below :

Control Name	Class Name
Textbox	System.Winforms.TextBox
Label	System.WinForms.Label
Button	System.Winforms.Button
RadioButton	System.Winforms.RadioButton
CheckBox	System.WinForms.CheckBox
ListBox	System.WinForms.ListBox
ComboBox	System.WinForms.ComboBox

Adding controls to the forms

Syntax :

<access_modifier> System.WinForms.<Control class> <identifier>
this.<identifier>=new System.WinForms.<Control_Class>

Example :

Public System.WinForms.Button new Button1;
This.<newbutton1>=new System.Winforms.Button();

Common Properties of the Controls are as follows

- Location
- Text
- TabIndex
- Size
- Visible
- Enabled
- Locked(whether control can be changed at runtime)

1. TextBox:

   The TextBox provides an area to enter or Display Text.

   Example :

```
Private System.WinForms.TextBox.txt;
this.txt=new System.WinForms.TextBox();
this.Location=new System.Drawing.Point(6,110)
txt.Text="Sample Text"
txt.TabIndex=0;
txt.Size=new System.Drawing.Size(200,20)
this.ontrl.Add(txt)
```

2. Label

A label Control Displays on a form that cannot be edited by the user.The System.WinForms.Label class is used to create a label.

Example :

```
Private System.WinForms.Label lbl;
this.lbl=new System.Winforms.Label();
lbl.Location=new System.Drawing.Point(40,172)
Lbl.Text="Name";
Lbl.TabIndex=1;
Lbl.Size=new System.Drawing.Size(172,24);
this.Conrtol.Add(lbl);
```

3.Button

The button executes a command or action when a user clicks on it.The System.WinForms.Button class is used to create a button.

Example :

```
Private System.WinForms.button  btn;
This.btn=new System.Winforms.Button();
btn.Location=new System.Drawing.Point(100,36)
btn.Size=new System.Drawing.Size(54,30);
btn.Text="Click";
btn.TabIndex=2;
this.Conrtol.Add(btn);
```

4.Radio Button

The Radio button control appear in a Group.Only One option can be selected from a group at a time. The System.WinForms.RadioButton class is used to create a radio button.

Example :

```
Private System.WinForms.Radiobutton  rb;
this.rb=new System.Winforms.RadioButton();
rb.Location=new System.Drawing.Point(116,172)
rb.Size=new System.Drawing.Size(140,20);
```

```
rb.Text="RadioButton1";
btn.TabIndex=1;
This.Conrtol.Add(rb);
```

5.CheckBox

The CheckBox control is used for a True/False Option.More than once check box can be selected in a form.The System.WinForms.CheckBox class is used to create a CheckBox.

Example :

```
Private System.WinForms.CheckButton  cb;
This.cb=new System.Winforms.CheckButton();
cb.Location=new System.Drawing.Point(120,172)
cb.Size=new System.Drawing.Size(146,26);
cb.Text="option1";
cb.TabIndex=1;
this.Conrtol.Add(cb);
```

6.ListBox

A ListBox Control Contains a list of Options from which the user can choose one or more options. The number of options in the listbox can be fixed.When the list becomes larger than the size of ListBox,scrollbars are added automatically.The System.WinForms.ListBox class is used to create a ListBox

Example :

```
Private System.WinForms.ListBox  lib;
This.lib=new System.Winforms.ListBox();
lib.Location=new System.Drawing.Point(32,48)
lib.Size=new System.Drawing.Size(64,70);
lib.TabIndex=2;
this.Conrtol.Add(lib);
```

The Code to Initialise the List

```
lib.items.All=new Object[]{"Rose","Lilly","Jasmine","Lotus"}
```

To add a new item to the list

```
lib.items.Add("MarieGold")
```

To remove an item from the list

```
lib.Items.Remove(lib.SelectedIndex)
```

To return the number of items in the list

```
int cnt=lib.Items.Count;
```

To sort the items in the list

```
list.Sorted=true;
```

6.ComboBox

The combobox control is similar to listbox .Manyitems can be added but it allows the user to select only one input at a time. The System.Winforms.ComboBox class is used to create a combobox.

Example :

```
Private System.WinForms.ComboBox cbx;
this.cbx = new System.Windows.ComboBox();
Cbx.Text="Combobox1";
Cbx.Location=new System.Drawing.Point(32,216)
Cbx.size=new System.Drawing.Size(62,20)
Cbx.Tabindex=3;
This.Controls.Add(cbx)
```

To initialise the combobox

```
cbx.Items.All= new object[] {"Rose","Lily","Jasmine","MarieGold"}
```

To add a new item to the list

```
cbx.items.Add("Lotus")
```

To remove an item from the list

```
cbx.Items.Remove(lib.SelectedIndex)
```

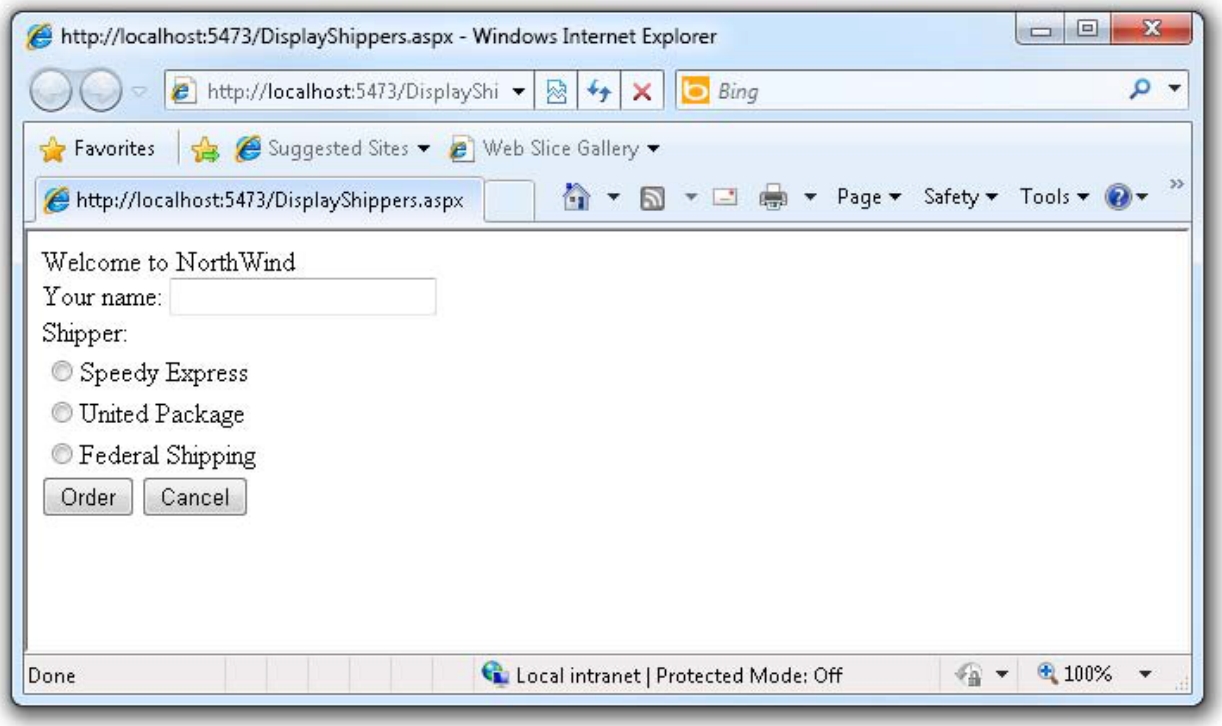
To return the number of items in the list

```
int cnt=cbx.Items.Count;
```

To sort the items in the list

```
cbx.Sorted=true;
```

SIMPLE WINDOWS FORMS APPLICATION WITH CONTROLS AND EVENTS



The .aspx file

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="DisplayShippers.aspx.cs"
Inherits="ProgrammingCSharpWeb.DisplayShippers" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>Welcome to NorthWind</div>
<div>
Your name:
<asp:TextBox ID="txtName" runat="server"></asp:TextBox></div>
<div>Shipper:</div>
<div>
<asp:RadioButtonList ID="rblShippers" runat="server"
DataSourceID="SqlDataSource1" DataTextField="CompanyName"
```

```
DataValueField="ShipperID">
</asp:RadioButtonList>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
SelectCommand="SELECT [ShipperID], [CompanyName] FROM [Shippers]">
</asp:SqlDataSource>
</div>
<div>
<asp:Button ID="btnOrder" runat="server" Text="Order" />
<asp:Button ID="Button2" runat="server" Text="Cancel" />
</div>
<div>
<asp:Label id="lblMsg" runat=server></asp:Label>
</div>
</form>
</body>
</html>
```

When the user clicks the Order button, you'll read the value that the user has typed in the Name text box, and you'll also provide feedback on which shipper was chosen. Remember, at design time, you can't know the name of the shipper, because this is obtained from the database at runtime, but we can ask the RadioButtonList for the chosen name or ID.

To accomplish all of this, switch to Design mode, and double-click the Order button. Visual Studio will put you in the code-behind page, and will create an event handlerfor the button's Click event.

You add the event-handling code, setting the text of the label to pick up the text from the text box, and the text and value from the RadioButtonList:

```
protected void btnOrder_Click(object sender, EventArgs e)
{

lblMsg.Text = "Thank you " + txtName.Text.Trim() +" whose ID is " +
rblShippers.SelectedValue;

}
```

When you run this program, you'll notice that none of the radio buttons are selected. Binding the list did not specify which one is the default. There are a number of ways to do this, but the easiest is to add a single line in the Page_Load method that Visual Studio created:

```
protected void Page_Load(object sender, EventArgs e)
{
rblShippers.SelectedIndex = 0;
}
```

This sets the RadioButtonList's first radio button to Selected. The problem with this solution is subtle. If you run the application, you'll see that the first button is selected, but if you choose the second (or third) button and click OK, you'll find that the first button is reset. You can't seem to choose any but the first selection. This is because each time the page is loaded, the OnLoad event is run, and in that event handler you are (re)setting the selected index.

The fact is that you only want to set this button the first time the page is selected, not when it is posted back to the browser as a result of the OK button being clicked.

To solve this, wrap the setting in an if statement that tests whether the page has been posted back:

```
protected void Page_Load(object sender, EventArgs e)
{
if (!IsPostBack)
{
rblShippers.SelectedIndex = 0;
}
}
```

When you run the page, the IsPostBack property is checked. The first time the page is posted, this value is false, and the radio button is set. If you click a radio button and then click OK, the page is sent to the server for processing (where the btnOrder_Click handler is run), and then the page is posted back to the user. This time, the IsPostBack property is true, and thus the code within the if statement isn't run, and the user's choice is preserved, as shown below fig

Code-behind form for DisplayShippers aspx.cs

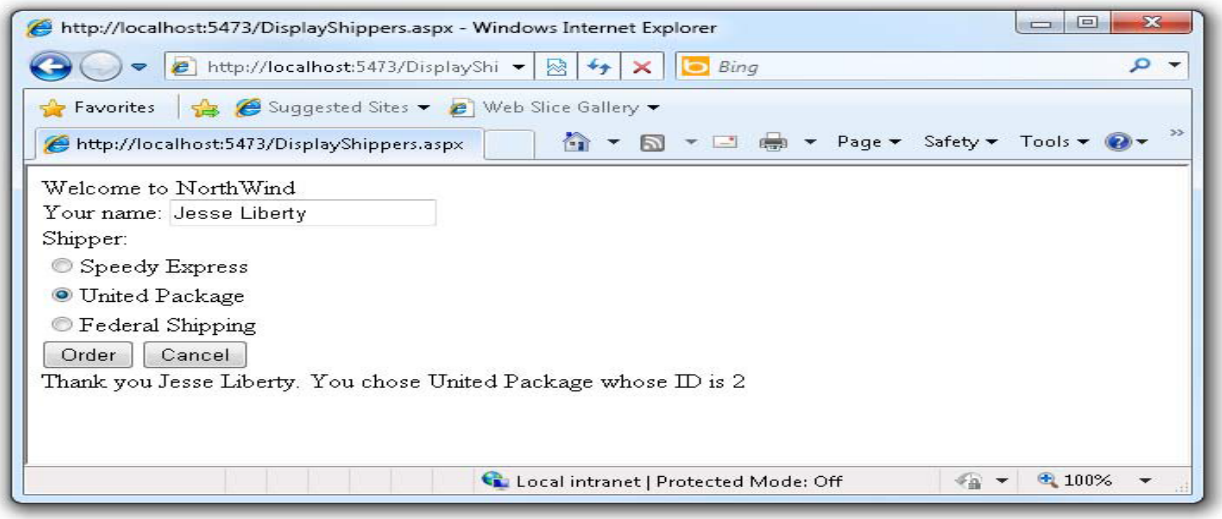
```
using System;
namespace ProgrammingCSharpWeb
{
public partial class DisplayShippers : System.Web.UI.Page
{

protected void Page_Load(object sender, EventArgs e)

{
if (!IsPostBack)
{
```

```
rblShippers.SelectedIndex = 0;
}
}
protected void btnOrder_Click(object sender, EventArgs e)
{
lblMsg.Text = "Thank you " + txtName.Text.Trim() +
". You chose " + rblShippers.SelectedItem.Text +
" whose ID is " + rblShippers.SelectedValue;
}
}

}
```



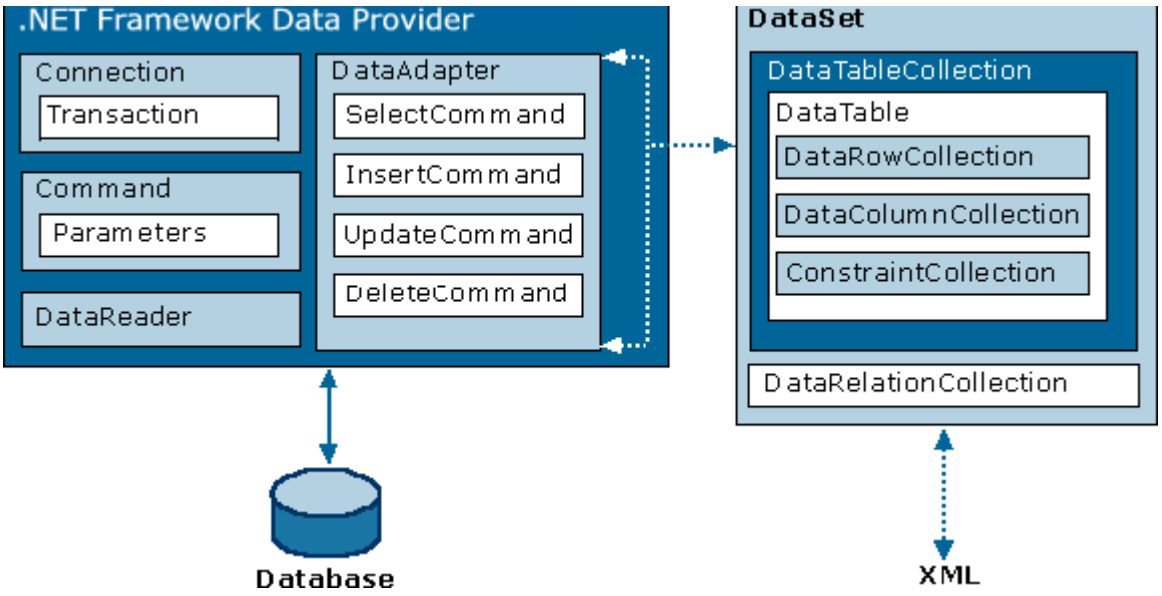
ADO.NET INTRODUCTION

ADO.NET is the new database technology of the .NET (Dot Net) platform, and it builds on **Microsoft ActiveX® Data Objects** (ADO).

ADO is a language-neutral object model that is the keystone of Microsoft's Universal Data Access strategy.

ADO.NET is an integral part of the .NET Compact Framework, providing access to relational data, XML documents, and application data. **ADO.NET** supports a variety of development needs.

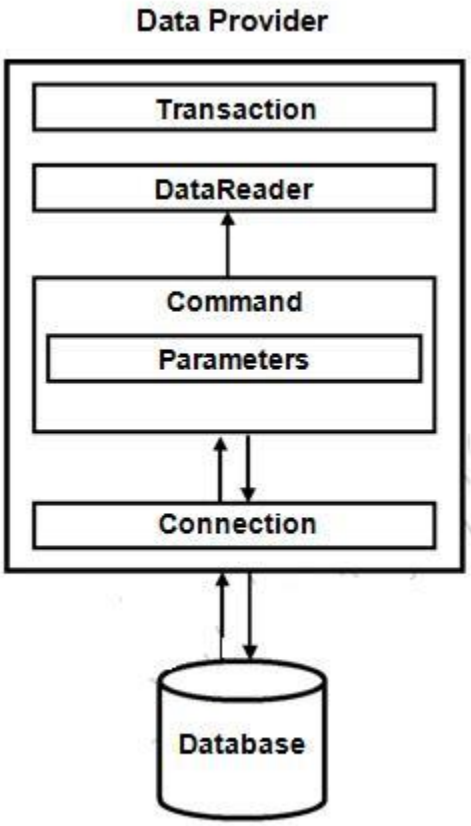
ADO.NET architecture



A .NET Framework data provider is used for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, or placed in an ADO.NET **DataSet** in order to be exposed to the user in an ad-hoc manner, combined with data from multiple sources, or remoted between tiers. The .NET Framework data provider is designed to be lightweight, creating a minimal layer between the data source and your code, increasing performance without sacrificing functionality.

CONNECTED ARCHITECTURE OF ADO.NET

The architecture of ADO.net, in which connection must be opened to access the data retrieved from database is called as connected architecture. Connected architecture was built on the classes connection, command, datareader and transaction.



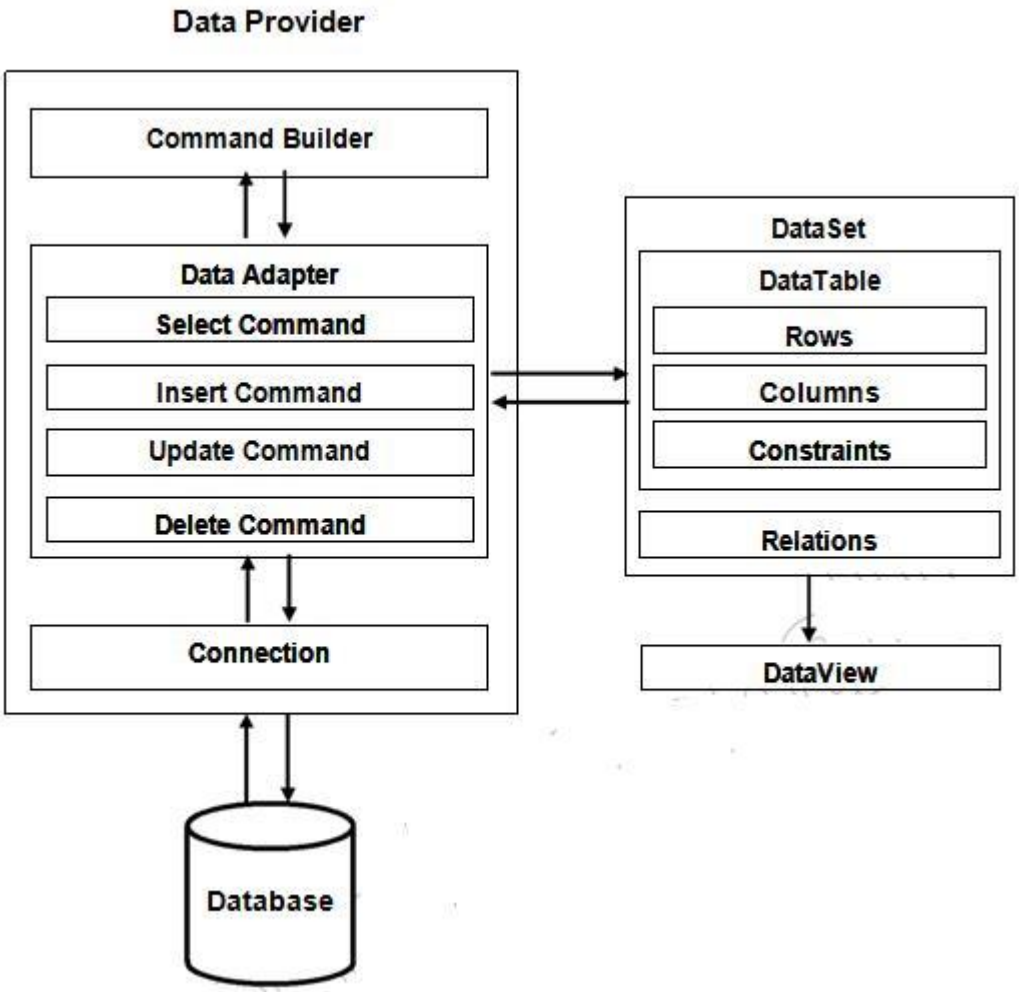
Connection : In connected architecture also the purpose of connection is to just establish aconnection to database and it self will not transfer any data.

DataReader : DataReader is used to store the data retrieved by command object and make it available for .net application. Data in DataReader is read only and within the DataReader you can navigate only in forward direction and it also only one record at a time.

To access one by one record from the DataReader, call **Read()** method of the DataReader whose return type is **bool.** When the next record was successfully read, the Read() method will return true and otherwise returns false.

DISCONNECTED ARCHITECTURE IN ADO.NET

The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture. Disconnected architecture of ADO.net was built on classes connection, dataadapter, commandbuilder and dataset and dataview.



Connection : Connection object is used to establish a connection to database and connectionit self will not transfer any data.

DataAdapter : DataAdapter is used to transfer the data between database and dataset. It has commands like select, insert, update and delete. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.

CommandBuilder : by default dataadapter contains only the select command and it doesn'tcontain insert, update and delete commands. To create insert, update and delete commands for the dataadapter, commandbuilder is used. It is used only to create these commands for the dataadapter and has no other purpose.

DataSet : Dataset is used to store the data retrieved from database by dataadapter and make it available for .net application.

To fill data in to dataset **fill()** method of dataadapter is used and has the following syntax.

Da.Fill(Ds,"TableName");

When fill method was called, dataadapter will open a connection to database, executes select command, stores the data retrieved by select command in to dataset and immediately closes the connection.

As connection to database was closed, any changes to the data in dataset will not be directly sent to the database and will be made only in the dataset. To send changes made to data in dataset to the database, **Update()** method of the dataadapter is used that has the following syntax.

Da.Update(Ds,"Tablename");

When Update method was called, dataadapter will again open the connection to database, executes insert, update and delete commands to send changes in dataset to database and immediately closes the connection. As connection is opened only when it is required and will be automatically closed when it was not required, this architecture is called disconnected architecture.

A dataset can contain data in multiple tables.

DataView : DataView is a view of table available in DataSet. It is used to find a record, sort the records and filter the records. By using dataview, you can also perform insert, update and delete as in case of a DataSet.

DATA PROVIDER

.NET provides data access services in the Microsoft **.NET** platform.

You can use **ADO.NET** to access data by using the new .NET Framework data providers which are:

- Data Provider for SQL Server (System.Data.SqlClient).
- Data Provider for OLEDB (System.Data.OleDb).
- Data Provider for ODBC (System.Data.Odbc).
- Data Provider for Oracle (System.Data.OracleClient).

ADO.NET is a set of classes that expose data access services to the .NET developer. The **ADO.NET** classes are found in *System.Data.dll* and are integrated with the XML classes in *System.Xml.dll*.

There are two central components of **ADO.NET** classes: the DataSet, and the **.NET** Framework **Data Provider**.

Data Provider is a set of components including:

- the **Connection object** (SqlConnection, OleDbConnection, OdbcConnection, OracleConnection)
- the **Command object** (SqlCommand, OleDbCommand, OdbcCommand, OracleCommand)
- the **DataReader object** (SqlDataReader, OleDbDataReader, OdbcDataReader, OracleDataReader)
- and the **DataAdapter object** (SqlDataAdapter, OleDbDataAdapter, OdbcDataAdapter, OracleDataAdapter).

Object	Description
Connection	Establishes a connection to a specific data source.
Command	Executes a command against a data source. Exposes **Parameters** and can execute within the scope of a **Transaction** from a **Connection**.
DataReader	Reads a forward-only, read-only stream of data from a data source.
DataAdapter	Populates a **DataSet** and resolves updates with the data source.

THE CONNECTION OBJECT

When creating a connection, you must specify several pieces of required information. Typically, this includes the type of authentication or user to authenticate, the location of the database server, and the name of the database.The following operations needs to be done in the connection object.

- Create a connection String
- Opening and closing a connection

Create a connection String

The `ConnectionString` contains a series of name/value settings delimited by semicolons (`;`). The order of these settings is unimportant, as is the capitalization. Taken together, they specify the information needed to create a connection.

Parameter	Description
`Data Source / Server / Address / Addr / Network Address`	The server name or network address of the database product to connect to. Use `localhost`

	for the current computer.
`Initial Catalog / Database`	The name of the database to use for all subsequent operations (insertions, deletions, queries, and so on).
`Integrated Security / Trusted_Connection`	Defaults to `false`. When set to `true` or `SSPI`, the .NET provider attempts to connect to the data source using Windows integrated security.
`User ID`	The database account user ID.
`Password/Pwd`	The password corresponding to the `User ID`.

The following code snippet shows how you might set the `ConnectionString` property on a `SqlConnection` object. The actual connection string details are omitted.

```
SqlConnection con = new SqlConnection();
con.ConnectionString = "...";
```

All standard ADO.NET `Connection` objects also provide a constructor that accepts a value for the `ConnectionString` property. For example, the following code statement creates a `SqlConnection` object and sets the `ConnectionString` property in one statement. It's equivalent to the previous example.

```
SqlConnection con = new SqlConnection("...");
```

We will see now some sample connection strings with commonly used settings. Because the connection string varies depending on the provider.We will see the connecton string for SQL SERVER

When using a SQL Server database, you need to specify the server name using the `Data Source` parameter (use `localhost` for the current computer), the `Initial Catalog` parameter (the database name), and the authentication information.

You have two options for supplying the authentication information. If your database uses SQL Server authentication, you can pass a user ID and password defined in SQL Server. This account should have permissions for the tables you want to access:

```
SqlConnection con = new SqlConnection("Data Source=localhost;" +
          "Initial Catalog=Northwind;user id=userid;password=password");
```

If your database allows integrated Windows authentication, you can signal this fact with the `Integrated Security=SSPI` connection string parameter. The Windows operating system then supplies the user account token for the currently logged-in user. This is more secure because the login information doesn't need to be visible in the code (or transmitted over the network):

```
SqlConnection con = new SqlConnection("Data Source=localhost;" +
           "Initial Catalog=Northwind;Integrated Security=SSPI");
```

Keep in mind that integrated security won't always execute in the security context of the application user. For example, consider a distributed application that performs a database query through a web service. If the web service connects using integrated authentication, it uses the login account of the ASP.NET worker process, not the account of the client making the request. The story is similar with a component exposed through .NET remoting, which uses the account that loaded the remote component host

Opening and closing a connection

To open a connection use con.open .To close a connection use con.close
Give the commands between the two statements

```
SqlConnection con = new SqlConnection("Data Source=localhost;" +
          "Initial Catalog=Northwind;Integrated Security=SSPI");
con.Open();
   // (Execute an ADO.NET command here.)
con.Close();
```

THE COMMAND OBJECT

The Command object is the heart of data processing with ADO.NET. Typically, the Command object wraps a SQL statement or a call to a stored procedure. For example, you might use a Command object to execute a SQL UPDATE, DELETE, INSERT, or SELECT statement.

- Creating a Command
- Executing a command

Creating a command

As with the Connection object, the Command object is specific to the data provider. Two examples are:

- System.Data.SqlClient.SqlCommand executes commands against SQL Server data provider

- System.Data.OleDb.OleDbCommand executes commands against an OLE DB data provider.

SYNTAX for a command object :

```
SqlCommand cmd = new SqlCommand(commandText, con);
```

Example :

```
string connectionString = "Data Source=localhost;" +
                "Initial Catalog=Northwind;Integrated Security=SSPI";

string SQL = "UPDATE Categories SET CategoryName='Beverages'" +
                "WHERE CategoryID=1";

SqlConnection con = new SqlConnection(connectionString);
 SqlCommand cmd = new SqlCommand(SQL, con);
```

Executing a command

Once the Command is created we have to execute the command. Following are the options to execute the command.

- ExecuteNonQuery() : Used to execute an SQL statement that doesn't return any value like insert, update and delete. Return type of this method is int and it returns the no. of rows effected by the given statement.
- ExecuteScalar() : Used to execute an SQL statement and return a single value. When the select statement executed by executescalar() method returns a row and multiple rows, then the method will return the value of first column of first row returned by the query. Return type of this method is object.

- ExecuteReader() : Used to execute a select a statement and return the rows returned by the select statement as a DataReader. Return type of this method is DataReader.

Command	**Return Value**
ExecuteReader	Returns a **DataReader** object.
ExecuteScalar	Returns a single scalar value.
ExecuteNonQuery	Executes a command that does not return any rows.

SYNTAX for Execute command using Executescalar:

```
string SQL =  any sqlstatment which returns single cvalue
SqlCommand cmd = new SqlCommand(SQL, con);

cmd.executescalar()
```

EXAMPLE using execute scalar :

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

public class TotalOrders
{
   public static void Main()
   {
            StringconnectionString = "Data Source= .\\SQLEXPRESS;
               AttachDbFilename=C:\\project\\sampledb.mdf;
               Integrated Security=True;
               Connect Timeout=30;User Instance=True";


      string SQL = "SELECT COUNT(*) FROM Orders WHERE " +
            "OrderDate >= '1996-01-01' AND OrderDate < '1997-01-01'";

      // Create ADO.NET objects.
      SqlConnection con = new SqlConnection(connectionString);
      SqlCommand cmd = new SqlCommand(SQL, con);

      // Execute the command.
      con.Open();
      int result = (int)cmd.ExecuteScalar();
      con.Close();

      // Display the result of the operation.
      Console.WriteLine(result.ToString() + " rows in 1996");
   }
}
```

Here's the sample OUTPUT for this code:

```
152 rows in 1996
```

SYNTAX for execute command using executenonquery :

```
string SQL =  any sql statement which does not return any rows like
            INSERT,UPDATE,DELETE

SqlCommand cmd = new SqlCommand(SQL, con);
cmd.executenonquery()
```

EXAMPLE using execute nonquery

```
using System;
using System.Data.SqlClient;
public class UpdateRecord
{
   public static void Main()
   {
         StringconnectionString = "Data Source= .\\SQLEXPRESS;
                  AttachDbFilename=C:\\project\\sampledb.mdf;
                  Integrated Security=True;
                  Connect Timeout=30;User Instance=True";
        String SQL = "UPDATE Department SET DepartmentName='Sales'" +
       "WHERE Departmenttid=5";
      // Create ADO.NET objects.
      SqlConnection con = new SqlConnection(connectionString);
      SqlCommand cmd = new SqlCommand(SQL, con);
      // Execute the command.
      con.Open();
      Console.WriteLine("Connection is " + con.State.ToString());
      int rowsAffected = cmd.ExecuteNonQuery();
      con.Close();
      // Display the result of the operation.
      Console.WriteLine(rowsAffected.ToString() + " row(s) affected");
   }
}
```

Data Reader

Data Reader is used to store the data retrieved by command object and make it available for .net application. Data in Data Reader is read only and within the Data Reader you can navigate only in forward direction and it also only one record at a time.

To access one by one record from the DataReader, call **Read()** method of the Data Reader whose return type is **bool.** When the next record was successfully read, the Read() method will return true and otherwise returns false.

Typical DataReader access code follows five steps:

1. Create a Command object with an appropriate SELECT query.
2. Create a Connection, and open it.
3. Use the Command.ExecuteReader( ) method, which returns a live DataReader object.
4. Move through the returned rows from start to finish, one at a time, using the DataReader.Read( ) method. You can access a column in the current row by index number or field name.
5. Close the DataReader( ) and Connection( ) when the Read( ) method returns false to indicate there are no more rows.

Data Reader Example

```
SqlCommand command = new SqlCommand("SELECT CategoryID, CategoryName
FROM Categories;",connection);
connection.Open();
SqlDataReader reader = command.ExecuteReader();
if (reader.HasRows)
{
while (reader.Read())
        {
        Console.WriteLine("{0}\t{1}", reader.getvalue(0),reader.Getvalue(1));
        }
}
else
        {
        Console.WriteLine("No rows found.");
        }
reader.Close();
}
```

Returning Multiple Result Sets in Data Reader

It is possible to execute a query that returns multiple result sets. This technique can improve performance because you need to contact the database only once to initiate the query. All data is then retrieved in a read-only stream from start to finish.

There are two ways to return more than one result set. You might be executing stored procedures that contain more than one SELECT statement. Alternatively, you might set up a batch query to execute multiple SQL statements by separating them with a semicolon:

```
// Define a batch query.
string SQL = "SELECT * FROM Categories; SELECT * FROM Products";

SqlConnection con = new SqlConnection(connectionString);
SqlCommand cmd = new SqlCommand(SQL, con);

con.Open();

// Execute the batch query.
SqlDataReader r = cmd.ExecuteReader();
```

You need only one DataReader to process multiple result sets. To move from the first result set to the second, use the DataReader.NextResult( ) method:

```
while (reader.Read())
{
   // (Process the category rows here.)
}

reader.NextResult();

while (reader.Read())
{
   // (Process the product rows here.)
}
```

DATASETS

The DataSet is a memory-resident representation of data including tables, relationships between the tables, and both unique and foreign key constraints. It is used for working with and transporting data in a disconnected environment.

Data set is a set of components including:

- Data Table
- Datacolumn
- Data Row
- Constraints

Data Relation

There are several ways of working with a **DataSet**, which can be applied independently or in combination. You can:

- Programmatically create a DataTable, DataRelation, and Constraint within a **DataSet** and populate the tables with data.
- Populate the **DataSet** with tables of data from an existing relational data source using a **DataAdapter**.

There are four important characteristics of the DataSet:

- It's not provider-specific. It's impossible to tell by looking at the DataSet, or at the objects contained within the DataSet, which provider was used to retrieve the data or what the original data source was. The DataSet provides a consistent programming model regardless of the data source.
- It's always disconnected. Information is retrieved from the data source and placed in the DataSet using another ADO.NET object?the DataAdapter. At no point does a DataSet directly reference a Connection object.
- It can track changes made to its data. The DataSet contains multiple versions of each row of data in the tables, which allows changes to be updated back to the data source using a DataAdapter object, changes to be cancelled, and XML DiffGrams of the changes to be created.
- It can contain multiple tables. Unlike the traditional ADO Recordset, the DataSet approximates a relational database in memory.

Creating a DataSet

There are several ways a `DataSet` can be created. In the simplest case, a `DataSet` is created using the `new` keyword. The constructor accepts on optional argument that allows the `DataSetName` property to be set. If the `DataSetName` argument isn't supplied, the default name of the `DataSet` will be `NewDataSet`.

```
DataSet ds = new DataSet("MyDataSet");
```

DATA TABLE

The `DataTable` is an in-memory data store that contains `DataColumn` and `Constraint` objects that define the schema of the data. The actual data is stored as a collection of `DataRow` objects within the `DataTable`. Both the schema and the data can be created entirely programmatically, retrieved as the result of a query against a data source using a .NET managed data provider, or loaded from an XML document or stream through the `DataSet` to which it belongs.

The DataTable and the DataReader are similar because they both provide access to the results of a query that can then be exposed through collections of row and column objects. The primary difference is that the DataTable is a disconnected class that places little restriction on how the data within it is accessed and allows that data to be filtered, sorted, and modified. The DataReader is a connected class that provides little functionality beyond forward-only, read-only access to the results of the query.

Creating a Data table

ADO.NET enables you to create DataTable objects and add them to an existing DataSet. Tables are added to the DataSet using the Add( ) method of the DataTableCollection. The Add( ) method takes an optional table name argument. If this argument isn't supplied, the tables are automatically named Table, Table1, and so on. The following example adds a table to a DataSet:

```
DataSet ds = new DataSet("MyDataSet");
DataTable dt = new DataTable("MyTable");

ds.Tables.Add(dt);
```

Data Columns

Columns belonging to the DataTable are stored as DataColumn objects in a DataColumnCollection object and are accessed through the Columns property of the DataTable.

There are two methods that can add a column to a table. The Add( ) method optionally takes arguments that specify the name, type, and expression of the column to be added. An existing column can be added by passing a reference to an existing column. If no arguments are passed, the default names Column1, Column2, Column3, and so on are assigned to the new columns. The following examples show how to create columns within the table:

Creating data columns

```
// adding a column using a reference to an existing column
DataColumn col = new DataColumn("MyColumn, typeof(System.Int32));
dt.Columns.Add(col);

// adding and creating a column in the same statement
dt.Columns.Add("MyColumn", typeof(System.Int32));
```

The second method for adding columns is the AddRange( ) method, which allows more than one column stored in a DataColumn array to be added to the table in a single statement, as shown in the following example:

```
DataTable dt = new DataTable("MyTable");

// create and add two columns to the DataColumn array
DataColumn[] dca = new DataColumn[]
   {new DataColumn("Col1", typeof(System.Int32)),
   new DataColumn("Col2", typeof(System.Int32))};

// add the columns in the array to the table
dt.Columns.AddRange(dca);
```

Constraints

The primary key is a column or collection of columns that uniquely identify each row in the table. The PrimaryKey property accesses one or more DataColumn objects that define the primary key of the DataTable. The primary key acts both as a unique constraint for the table and allows records to be located using the Find( ) method of the DataTableRows collection, as discussed in the next section.

The primary key for a table is set by specifying an array of DataColumn objects from the table. The following example illustrates creating a primary key based on two columns:

```
// set the primary key based on two columns in the DataTable

DataTable dt = new DataTable("MyTable");
dt.Columns.Add("PK_Field1", typeof(System.Int32));
dt.Columns.Add("PK_Field2", typeof(System.Int32));

// ... add other table columns

// set the primary key

dt.PrimaryKey = new DataColumn[]
{ dt.Columns["PK_Field1"],
   dt.Columns["PK_Field2"]
};
```

To remove the primary key, simply set the primary key object to null.

```
// remove the primary key

dt.PrimaryKey = null;
```

Data row

The `DataRow` class represents a single row of data in the `DataTable`. The `DataRow` class can retrieve, update, insert, and delete a row of data from the `DataTable`. Using the `DataRow` class, each column value for the row can be accessed.

The DataRow maintains the RowState property that is used by ADO.NET to track the changes that have been made to a DataRow. This property allows changed rows to be identified, and the appropriate update command to be used to update the data source with the changes.The following operations can be performed

1) Creating a datarow
2) Getting a data row
3) Deleting a datarow

Creating a datarow :

A DataRow is created by calling the NewRow( ) method of a DataTable, a method that takes no arguments. The DataTable supplies the schema, and the new DataRow is created with default or empty values for the fields:

```
// Here we create a DataTable with four columns.
    DataTable tbl = new DataTable();
    tbl.Columns.Add("Weight", typeof(int));
    tbl.Columns.Add("Name", typeof(string));
    tbl.Columns.Add("Breed", typeof(string));
    tbl.Columns.Add("Date", typeof(DateTime));

    // Here we add five DataRows.
    tbl.Rows.Add(57, "Koko", "Shar Pei", DateTime.Now);
    tbl.Rows.Add(130, "Fido", "Bullmastiff", DateTime.Now);
    tbl.Rows.Add(92, "Alex", "Anatolian Shepherd Dog",
    DateTime.Now);
    tbl.Rows.Add(25, "Charles", "Cavalier King Charles
    Spaniel", DateTime.Now);
    tbl.Rows.Add(7, "Candy", "Yorkshire Terrier",
    DateTime.Now);
    return tbl;
```

Getting a Datarow :

The contents of a row can be modified. First, you can simply replace the values of the column with a new value:

```
// Get the first row from the DataTable.
DataTable tbl = GetTable();
DataRow row = tbl.Rows[0];
```

```
Console.WriteLine(row["Name"]);
// Get the last row in the DataTable.
DataRow last = tbl.Rows[table.Rows.Count - 1];
Console.WriteLine(last["Name"]);
```

Deleting a datarow:

The Delete( ) method deletes rows from the DataTable. If the RowState is added, the row is removed; otherwise the RowState of the existing DataRow is changed to Deleted. The row is permanently removed from the table only when AcceptChanges( ) is called on the row either explicitly or implicitly when the Update( ) method of the DataAdapter successfully updates the changes to the row back to the data source.

```
DataRow row = table.Rows[0];
// Delete the first row. This means the second row is the
first row.
row.Delete();
// Display the new first row.
row = table.Rows[0];
Console.WriteLine(row["Name"]);
```

Data Relation

Every DataSet contains a DataRelationCollection object, which contains DataRelation objects. Each DataRelation defines a relationship between two tables. There are two reasons why you might define DataRelation objects:

- To provide better error checking. This functionality is provided through the ForeignKeyConstraint, which the DataRelation can create implicitly.
- To provide better navigation.

A typical DataRelation requires three pieces of information:

- a descriptive name of your choosing (which doesn't signify anything or relate to the data source),
- a reference to the parent DataColumn,
- a reference to the child DataColumn.

SYNTAX:

DataRelation relation = new DataRelation("Name", ParentCol, ChildCol);
ds.Relations.Add(relation); //adding to the dataset

For example, to create a relationship between product categories and products, use the following code:

```
DataColumn parentCol = ds.Tables["Categories"].Columns["CategoryID"];
DataColumn childCol = ds.Tables["Products"].Columns["CategoryID"];
DataRelation relation = new DataRelation("Cat_Prod", parentCol, childCol);
ds.Relations.Add(relation);
```

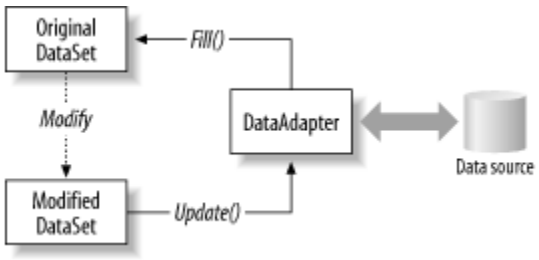
DATA ADAPTER

A Data Adapter is a bridge between the Data Source and the data set.Select,Insert,Update,Delete operations in the Data Source can be performed by data adapter. A Data Adapter contains a connection object and a command object. It manages data in a disconnected mode. And opens and closes connection automatically, when reading or writing to a database.

The DataAdapter retrieves data into a DataSet or a DataTable from a data source using the Fill( ) method. Schema information can be retrieved using the FillSchema( ) method. The DataAdapter updates any changes made to the DataSet or DataTable back to the data source using the Update( ) method.

In simple scenarios, the updating logic that the DataAdapter uses to reconcile changes made to the DataSet can be generated automatically from the query used to retrieve the data by using a CommandBuilder object. For more complex scenarios, custom update logic can be written to control the logic the DataAdapter uses when adding, deleting, and modifying records in the data source in response to changes made to the DataSet. In either case, the updating logic is used when the Update( ) method is called.

Once data is retrieved using the DataAdapter, no information about the connection, database, tables, columns, or any other details about the source of the data is available in the disconnected objects. The data can be persisted or passed between applications without the risk of exposing details about the location or structure of the data source or access credentials used. Figure 14-1 shows the structure of the DataAdapter and the contained classes.



Fill( ) method of the DataAdapter retrieves data from the data source into a DataSet or a DataTable.

Update( ) method of the dataadapter can submit DataSet changes back to the data source.

FILL ()

The Fill( ) method of the DataAdapter retrieves data from the data source into a DataSet or aDataTable. When the Fill( ) method for the data adapter is called, the select statement defined in theSelectCommand is executed against the data source and retrieved into a DataSet or DataTable. In addition to retrieving data, the Fill( ) method retrieves schema information for columns that don't exist. This schema that it retrieves from the data source is limited to the name and data type of the column. If more schema information is required, the FillSchema( ) method, described later in this chapter, can be used. The following example shows how to use the Fill( ) method to retrieve data from the Orders table .

```
// connection string and the select statement

String connString = "Data Source=(local);Integrated security=SSPI;" +
 "Initial Catalog=Northwind;";

String selectSQL = "SELECT * FROM Orders";

SqlDataAdapter da = new SqlDataAdapter(selectSQL, connString);

// create a new DataSet to receive the data
DataSet ds = new DataSet();

// read all of the data from the orders table and loads it into the
// Orders table in the DataSet
da.Fill(ds, "Orders");

//A DataTable can also be filled similarly:
DataTable dt = new DataTable("Orders");

// use the data adapter to load the data into the table Orders
da.Fill(dt);
```

UPDATE ()

The Update( ) method can submit DataSet changes back to the data source. It uses the statements in theDeleteCommand, InsertCommand, and UpdateCommand objects to attempt to update the data source with records that have been deleted, inserted, or updated in the DataSet. Each row is updated individually and not as part of a batch process. Furthermore, the order in which the rows are processed is determined by the indexes on the DataTable and not by the update type.

```
// connection and select command strings
String connString = "Data Source=(local);Integrated security=SSPI;" +
   "Initial Catalog=Northwind;";
String selectSql="SELECT * FROM Orders";

// create a new DataSet to receive the data
DataSet ds = new DataSet();

SqlDataAdapter da = new SqlDataAdapter(selectSql, connString);

// create the command builder
// this creates SQL statements for the DeleteCommand, InsertCommand,
// and UpdateCommand properties for the data adapter based on the
// select command that the data adapter was initialized with
SqlCommandBuilder cb = new SqlCommandBuilder(da);

// read all of the data from the orders table and load it into the
// Orders table in the DataSet
da.Fill(ds, "Orders");

// ... code to modify the data in the DataSet

// update the data in the Orders table in the DataSet to the data source
da.Update(ds, "Orders");
```

2-MARKS

1. Explain two differences between windows based and Web based application

Features	Windows Forms	Web Forms
User Interfaces, data binding etc.	Easy to build	Difficult to build
Deployment and Maintenance	Complex. New versions of assemblies, configuration files, and other required files must be deployed on all client machines. Usually user interaction required.	Easy. Need to deploy assemblies and configuration files on the server only. Transparent to the client.
Performance	Faster	Slower
Robustness and Reliability	One client machine goes down, other users are still live.	Usually web servers are never down. However if the server goes down, all users are affected.
Network Congestion	Depending on the data transfer and connections made to the server from various clients.	Depends
Resources	Runs on the client machine.	Runs on a Web server.
Framework dependency	All client machines have to install required versions of .NET framework and other required libraries	Only server needs to have .NET framework and other required libraries.

2. Define disconnected architecture of ADO.NET ?
   The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture.

3. What is the namespace used to connect to sqlclient ?

System.Data.SqlClient is the namespace used to connect to sqlclient

4. What are the different data providers supported by ADO.NET ?
   Data Provider for SQL Server (System.Data.SqlClient).
   Data Provider for OLEDB (System.Data.OleDb).
   Data Provider for ODBC (System.Data.Odbc).
   Data Provider for Oracle (System.Data.OracleClient).

5. What is the use of ExecuteNonQuery command ?

   ExecuteNonQuery() is used to execute an SQL statement that doesn't return any value like insert, update and delete. Return type of this method is int and it returns the no. of rows effected by the given statement.

6. What is the syntax for the command object ?

   SqlConnection con = new SqlConnection(connectionString);
   SqlCommand cmd = new SqlCommand(commandText, con);

   Where command text is the SQL Statement

7. What does executescalar will return ?
   ExecuteScalar is used to execute an SQL statement and return a single value. When the select statement executed by executescalar() method returns a row and multiple rows, then the method will return the value of first column of first row returned by the query. Return type of this method is object.

8. Name the different components of the data provider .

   Data Provider is a set of components including:

   - **Connection object**
   - **Command object**
   - **DataReader object**
   - **DataAdapter object**

9. What are the different objects in Dataset ?
   Data set is a set of components including:
   Data Table
   Datacolumn
   Data Row
   Constraints
   Data Relation

10. Name any two important characteristic of dataset .

- It's not provider-specific. It's impossible to tell by looking at the DataSet, or at the objects contained within the DataSet, which provider was used to retrieve the data or what the original data source was. The DataSet provides a consistent programming model regardless of the data source.
- It's always disconnected. Information is retrieved from the data source and placed in the DataSet using another ADO.NET object?the DataAdapter. At no point does a DataSet directly reference a Connection object.
- It can track changes made to its data. The DataSet contains multiple versions of each row of data in the tables, which allows changes to be updated back to the data source using a DataAdapter object, changes to be cancelled, and XML DiffGrams of the changes to be created.
- It can contain multiple tables. Unlike the traditional ADO Recordset, the DataSet approximates a relational database in memory.