UNIT – 4

Overview of ADO.Net

Overview of ADO.NET

Active-x Data Object (ADO.NET) allows you to interact relational databases and other data sources. ADO.NET is a technology that ASP.NET applications use to communicate with database whether they need to add new customer's records, purchase and display a product catalogue. ADO.NET is much more than a slightly enhanced .NET version to ADO data access technology.

Many real-world applications need to interact with a database. The .NET Framework provides a rich set of objects to manage database interaction; these classes are collectively referred to as ADO.NET.
ADO.NET looks very similar to ADO, its predecessor. The key difference is that ado.net is disconnected data architecture. In a disconnected architecture, data is retrieved from a database and cached on your local machine. You manipulate the data on your local computer and connect to the database only when you wish to alter records or acquire new data. There are significant advantages to disconnecting your data architecture from your database. The biggest advantage is that you avoid many of the problems associated with connected data objects that do not scale very well. Database connections are resource-intensive, and it is difficult to have thousands (or hundreds of thousands) of simultaneous continuous connections. A disconnected architecture is resource-frugal.

ADO.NET connects to the database to retrieve data, and connects again to update data when you've made changes. Most applications spend most of their time simply reading through data and displaying it; ADO.NET provides a disconnected subset of the data for your use while reading and displaying.

Disconnected data objects work in a mode similar to that of the Web. All web sessions are disconnected, and state is not preserved between web page requests. Disconnected data architecture makes for a cleaner marriage with the Web.

➔Database Access in the Internet World :-

Accessing a database in an internet application is a completely different scenario than accessing a database in a desktop client server Program. Web Application raise a hole new set of consideration and potential problem. These problems have ASP developer for years.

(1) Problem of Scale
(2) Problem of States
(3) Problem of communication & compatibility.

(1) Problem of Scale :

A web Application has the potential to be used by the 100 of simultaneous users. This means that it can't be casual about server means that it can't be casual about server memory or limited resources like data base connection all of this problem are possible with traditional client server database development. The differences is fearless lightly to have any negative effect because the typically load (The no. of simultaneous user) is lower or minimum.

In a web Application, database practices that might slightly hamper the performance have a client server application can multiply rapidly and cause significant.

(2) Problem of State :

You are familiar with the effect that HTTP state less protocol (Connection less) this means that connection between the browser and web server are not maintain, when a user

browse to a page in ASP.NET application, A Connection is made, code is process, an HTML page is return & the Connection is immediately severed (disconnected).

While user may have the illusion that they are interactive with a continuously running application they are really just receiving a string of static pages.

(3) Problem of Communication & Compatibility :

Sending data over internal network is easy but sending information over the internet is also easy as long as to stick and standard the http protocol. Generally, ADO could be used in a true multiplatform internet application all problem because all octal code is performed on the server.

ADO .NET

Microsoft ADO.NET is the latest improvement after ADO. ADO.NET provider platform interoperability and scalable data access. In the .NET framework data is transmitted in the Extensible Mark up Language (XML). Therefore, any application that can read XML format can process data it is not necessary for receiving components to be ADO.NET Component at all.

ADO.NET the foundation of data aware .NET application. ADO.NET brings together all the classes that follow data handling. For exp. Indexing, sorting, views etc.

The ADO Model uses the Concept of the record set all through this record set are very flexible to use and allow access to data. Even when disconnected from data sources. This usage of COM component object modelling but the interoperability is limited when using ADO.NET record set. So the Concepts of ADO.NET record set fails.

Unlike ADO.NET usage XML as the data format because XML is the universal data format being use. Instead of record set, ADO.NET usage the dataset and data reader objects, data adaptor objects to access and manipulate data. Thus ADO.NET designs to perform better and more flexible then ADO.

Ado .net is a technology used for working with data and database of all types. It provides access to data sources, such as Microsoft sql server, and to data sources exposed through ole DB and extensible markup language (xml).you can use ado .net to connect to data sources for retrieving, manipulating and updating data

Some features of ADO.NET are as follows:

- **Disconnected data architecture**-implies that applications connect to the database only when data needs to be retrieved or modified. After the database operation has been performed, the connection to the database closed. To perform any database operation again, the connection with the database will have to be re-established. Applications that do not follow the disconnected data architecture waste a lot of valuable system resources and time because these application need to maintain a connection with the database even when it is not required.

- **Cached data in datasets**-follows a disconnected architecture for accessing modifying data .hence, it is not required for the applications to connects to the database for processing each record. Therefore, the data is accessed and later stored in the datasets. A dataset is a cached set of database records, i.e. datasets is not dependent on the sources. Even when you are disconnected from the database on which you are working, you can make modifications in the database.

- **Scalability**-reduces the traffic on the database and saves the resources resulting in more efficiency of the database in meeting the demands of the user. Ado .net helps in scalability while working with datasets. All database operations are performed on the dataset instead of on the database.

- **Transfer of data in xml format**- transfer data from a database into and from the dataset to another component using xml. Xml IS THE standard format used for transferring data in ado.net. You can also use xml file as a data sources and store the data from it into a dataset. One needs not know xml since data conversions from and to xml is abstracted from the user.

- **Interaction with the database through data commands**- specifies that database operations are performed by executing the data commands. All operations on a database, such as retrieving, modifying, or updating of data are performed using data commands .a data command is either a structured query language (sql) statement or a stored procedure.

➔ Characteristics of ADO.NET:-

- Disconnected Access
- Based on XML
- Extended Data Format
- Managed Code
- Data Provides
- Maintainability
- Programmability
- Performance
- Scalability

(1) Disconnected Access :

Disconnected Access is the most important characteristic of the ADO.NET, In a typical ADO Application, you connect to the database, create a record set use the information to fill a Data Grid or calculate some type of summary information. And then abandon the record set and close the connection while the connection is open, you have a "Live" connection with the database that allows you to make immediate update and in some cases see the changes made by other users in real time.

(2) Based on XML :

ADO.NET uses XML natively to store data. This fact is not automatically obvious when you are working with ADO.NET datasets, because you will usually use the dataset Objects built in methods and properties to perform all the data manipulation you need. In this case, you will access data in a row based fashion that resembles ADO. You might also make the reasonable assumption that XML is just another import and export format supported by ADO.NET. For advanced computability. In fact XML plays a far greater role.

(3) Extended data Format:

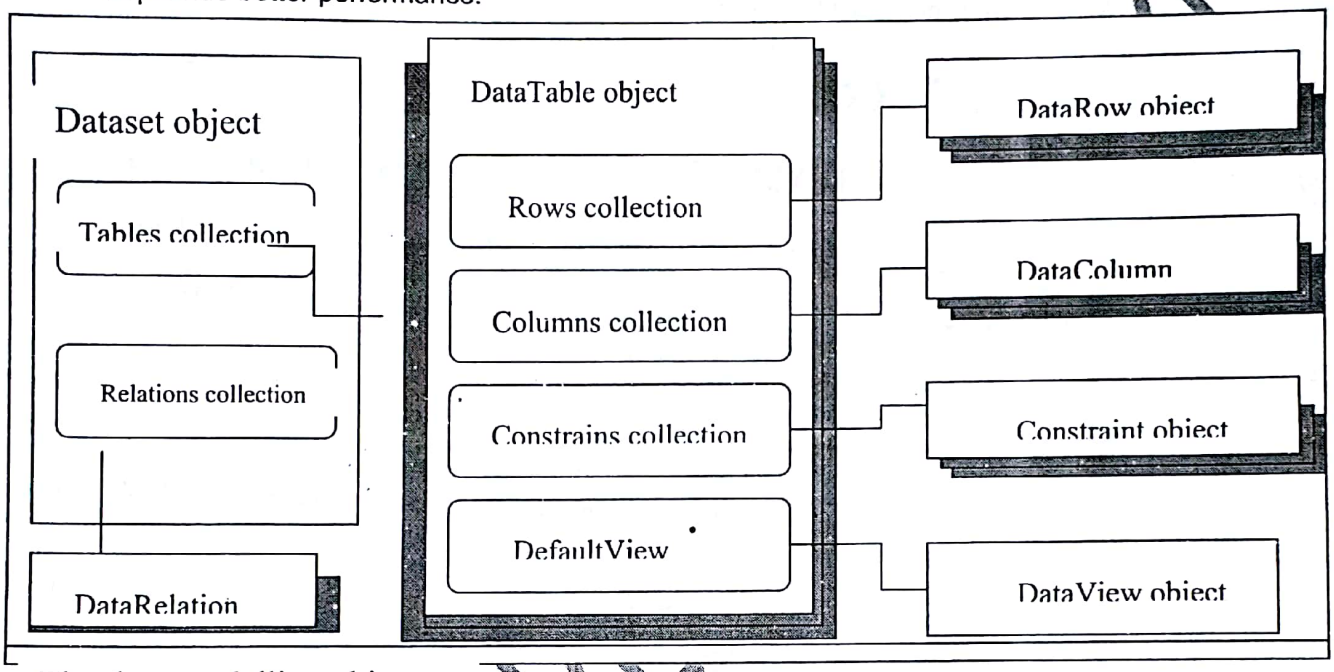
ADO.NET has a self sufficient that ADO never had part of this change result from its new role as a disconnected data manager. ADO.NET retains more information to allow you to effectively make changes without needing to rapidly reconnect to the database.

(4) Managed Code:

Like the entire .NET class library Components, ADO.NET is built in the managed Code Environment which means it can be used easily in the other .NET application. The traditional ADO Components on the Other hand use COM to Communicate.

(5) Data Providers :

You might wonder where OLEDB fits into all this. OLEDB was an important standard that supported ADO. In essence, ADO could communicate with any database that had an OLEDB provider. ADO.NET is exactly the same of course, communicating with the standards OLEDB providers still means a jump from managed to unmanaged code, but other providers are on the way to provide better performance.



The data modelling objects

Difference between ADO and ADO.net

1. **ADO**
   1. Classic ADO requires active connection with the data store.
   2. Data is stored in binary format.
   3. XML integration is not possible.
   4. It uses the object named Recordset to reference data from the data store.
   5. Using Classic ADO, we can obtain information from one table or set of tables through join. We cannot fetch records from multiple tables independently.
   6. Firewall might prevent execution of Classic ADO.
   7. Classic ADO architecture includes client side cursor and server side cursor.
   8. We cannot send multiple transactions using a single connection instance.

ADO.NET

1. ADO.NET architecture works while the data store is disconnected.
2. Data is stored in XML.
3. XML integration is possible.
4. It uses Dataset Object for data access and representation.
5. Dataset object of ADO.NET includes collection of DataTables wherein each

DataTable will contain records fetched from a particular table. Hence multiple table records are maintained independently.
6. ADO.NET has firewall proof and its execution will never be interrupted.
7. ADO.NET architecture doesn't include such cursors.
8. We can send multiple transactions using a single connection instance.

Other Difference

ADO used OLE DB to access data and is COM-based, while ADO.net uses XML as the format for transmitting data to and from your database and web application.

In ADO, it is sometime problematic because firewall prohibits many types of request, while in ADO.net there is no such problem because XML is completely firewall-proof.

ADO.NET Data Access

DISCONNECTED DATABASE ACCESS

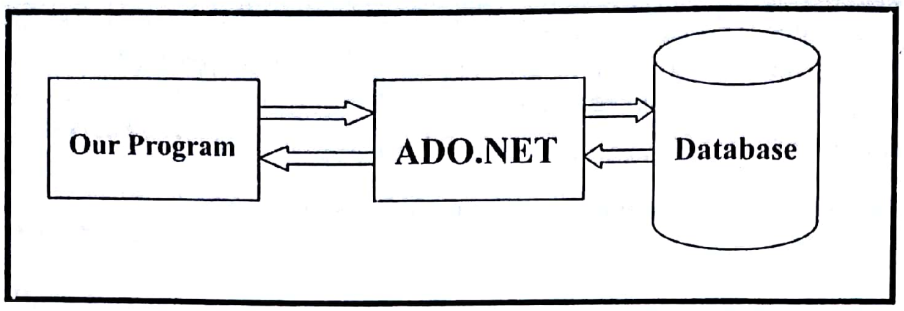
Previous data-access technologies provided continuously connected data access by default. In such a model, an application creates a connection to a database and keeps the connection open for the life of the application, or at least for the amount of time that data is required. However, as application become more complex and database serve more and more clients, connected data access is impractical (impossible) for a variety of season, including the following:

- Open database connections are expensive in terms of system resources. The more open connection there are, the less efficient system performance becomes.
- Applications with connected data access are difficult to scale. An application that can comfortably maintain connection with two clients might to poorly with 10 and be completely unusable with 100.
- Open database connection can quickly consume all available database licenses, which can be a significant expense. In order to work within a limited set of client licenses, connection must be reused whenever possible.

ADO.NET addresses these issues by implementing a disconnected data access model by default. In this model, data connections are established and left open only long enough to perform the requisite action. For example, if an application requests data from a database, the connection opens just long enough to load the data into the application, and then it closes. Likewise, if a database is updated, the connection opens to execute the UPDATE command, and then closes again. By keeping connection open only for the minimum required time, ADO.NET conserves system resources and allows data access to scale up with a minimal impact on performance.

ADO.NET

- Most application requires some from of data access.
- ADO is a part of the Microsoft.Net Framework.
- **The full from of the ADO.net is ActiveX Data Objects.**
- ADO.NET has the ability to separate data access mechanisms.
- ADO.NET is a set of classes that allow application to read and write information in databases.

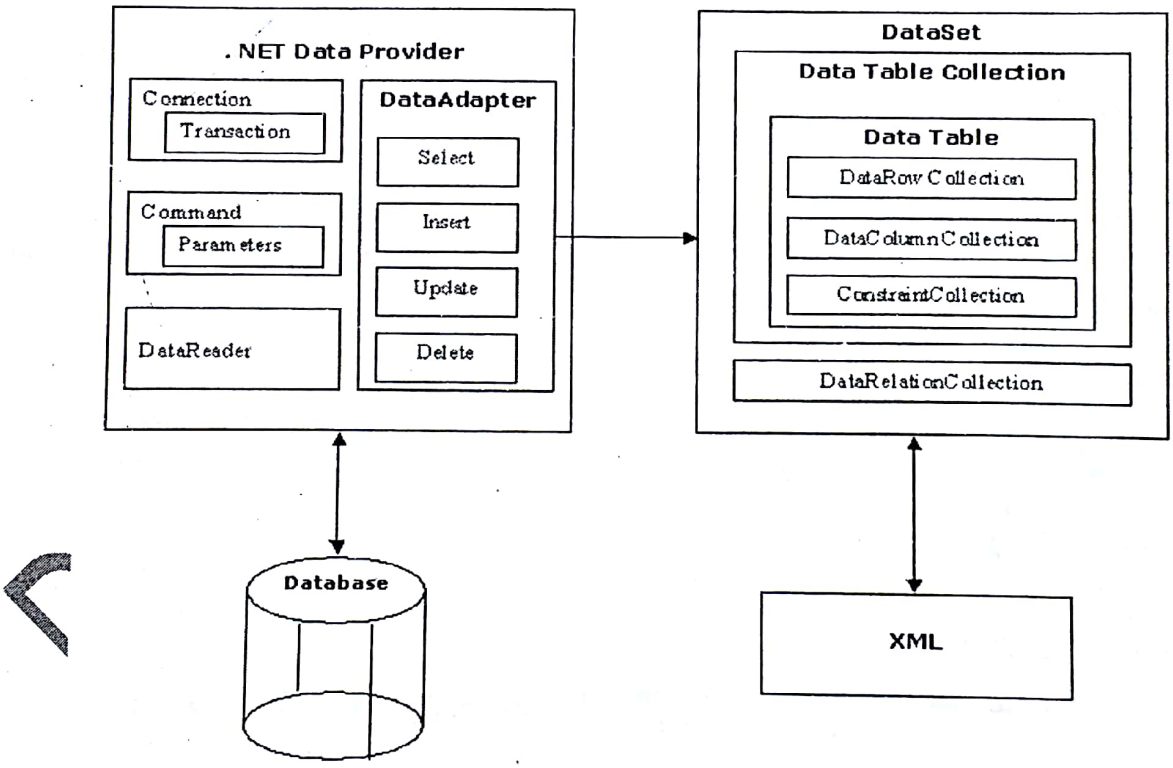


- ADO.NET can be used by .NET language.
- It's a concept. It's not a programming language.
- ADO.NET introduces the concept of **disconnected language**.
- We need to add System data namespace for work with ADO.Net.
- It's a next version of ActiveX Data Object (ADO) technology which was used in VB6.0.

ADO.NET Data Architecture

Data access in ADO.NET relies on two entities:

DataSet: This stores data on the local machine.
Data Provider: A set of components that mediates interaction between the program and the database.



ADO .NET Data Architecture

THE DATASET:

- The Dataset is a disconnected, in-memory representation of data.
- It can be thought of as a local copy of the relevant portions of a database.
- Data can be loaded into a Dataset from any valid data source, such as a SQL Server database, a Microsoft Access database, or an XML File.

- The Database persists in memory, any the data there in can be manipulated and updated independent of the database.
- When appropriate, the dataset can then act as a template for updating the central database.
- The Database object contains a collection of zero or more DataTable objects, each of which is an in-memory representation of a single table.
- The structure of a particular DataTable is defined by the Data Columns collection, which enumerates the columns in a particular table
- The constraint collection, which enumerates any constraints on the table.
- A DataTable also contains a DataRows collection, which contain the actual data in the DataSet.
- The DataRelations collection enumerates a set of Data Relation objects that define the relationship between tables in the DataSet. For Example, consider a DataSet that contains two related tables: an Employees table and a Projects table. In the Employee table, each employee is represented only once and is identified by a unique EmployeeID field. In the Project table, an employee in charge of a project is identified by the EmployeeID field, but can appear more than once if that employee is in charge of multiple projects. This is an example of a one-to-many relationship: you would use a DataRelation object to define relationship.
- Additionally, a DataSet contain an Extended Properties collection, which is used to store custom information about the Dataset.

THE DATA PROVIDER: **four core objects of the data provider:**

(1) The Connection object:

The connection objects represent the actual connection to the database1. Visual Studio .NET supplies two types of connection class: the SQL connection objects, which is designed specifically to connect to SQL server 7 or latter, and the OleDbConnection Object, which can provide connection to a wide range of database types. The connection object contains all of the information required to open a channel to the database in the Connection String property.

➔ SQL Connection and OLEDB Connection :

This object allows you to establish a connection which is the first step in any data base operation, causing the open Method.

(2) The command objects:

The command object is represented by two corresponding classes, SqlCommand and OleDbCOmmand. You can use Command Objects to execute command to a database across a data connection. Command objects can be used to execute stored procedure on the database and SQL commands, or return complete tables. Command Objects provide three methods that are use to execute commands to database.

Methods of command objects

ExecuteReader: Returns Data Reader.
Executes command that returns row

ExecuteNonQuery: Returns Row # affected.
For UPDATE, INSERT and DELETE statement, the return value is the number of rows affected by the command.

ExecuteScllar: Return single value.
Used with aggregate function.
It returns only first column of first row.
There are five aggregate function max, min, avg, count, and sum.

It returns only one value at a time.
It fetches the data from database.

➔ SQL Command and OLEDB Command ::

This object represents an SQL statement or store procedures that you can use against data source to retrieve, update or modify data.

To actually use this command you must use a data adaptor (SQL data adapter or OLEDB data adapter). Use one of the following Methods ....

(i) Execute Non Query
(ii) Execute Reader
(iii) Execute Scalar

(i) Execute Non Query :

Allows you to execute the command without retrieving any information. This is deal method for (insert, Update, Delete operation using SQL, SP).

(ii) Execute Reader :

Creates a data reader objects which provide a fast forward only connection that allows you to retrieve rows from data sources.

(iii) Execute Scalar :

It also creates a reader, but it only returns a single value. The first column from the first row from the row set.

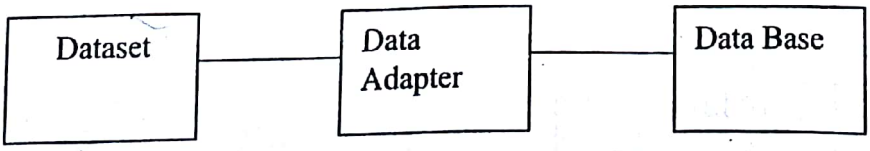
(3) The data reader objects:

The DataReader object provides a forward-only, read-only connected stream recordset from a database. Unlike other components of data provider, DataReader objects cannot be directly instantiated. Rather, the DataReader is returned as the result of a command object's ExecuteReader method. The SqlCommand.ExecuteReader method returns a SqlDataReader object and the OleDbCommand.ExecuteReader method return an OleDbDataReader objects. Likewise, the ODBC an Oracle Command.ExecuteReader method returns a DataReader specific to the ODBC and Oracle Data Provides respectively. The DataReader can supply rows of data directly to application logic when you do not need to keep the data cached in memory. Because only one row in memory at a time, the DataReader Provides the lowest overhead in terms of system performance. But it requires exclusive use of an open Connection object for the lifetime Of the DataReader.

➔ SQL Data Reader and OLEDB Data Reader ::

This object provides stream of Data they are fastest way to read data when all you want to display it in a web page and you don't need to any update or manipulate.

(4) The DataAdapter Object:



The DataAdapter is the class at the core of ADO.NET disconnected data access. It is essentially the middleman, facilitating all communication between the data access it is essentially the middleman, facilitating all communication between the database and dataset. The DataAdapter fills a DataTable or DataSet with Data from the database whenever the fill method is called. After the memory-resident data has been manipulated the DataAdapter can transmit changes to the database by calling the Update method the DataAdapter provides four properties that represent data base commands. the four properties are:--

- SelectCommand: Contains the command text or objects that select the data from the database. This command is executed when the fill method is called and fills a DataTable or a DataSet
- InsertCommand: Contents the command text or object that insert a row into a table
- DeleteCommand: Contents the command text or object that delete a row from a table
- UpdateCommand: Contents the command text or object that update the values of a database

When the update method is called, change in the DataSet are copied that to the database, and the appropriate insert command, delete command, or update command is executed.

➔ SQL Data Adapter and OLEDB Data Adapter ::

This object derives from data adapter class act as a bridge between command and data set.

For Ex. You can create an SQL Statement that select a set of rows, create a command that represent it and use the SQL data adapter or OLEDB Data Adapter object to automatically retrieve to all matching value and insert it into a dataset.

Once you have set the appropriate commands you can use the following methods.....

- **Fill:** Execute the select command and places the result into dataset.
- **Fill Scheme:** Usage the select command to retrieve information about the table such as column constraints.

Updates: Scan through the supplied data set and applies all he changes it finds to the data source.

(1) Understanding Some Other Data Provider

For SQL server	• Imports System.Data.SqlClient namespace. • It provides data access for Microsoft SQL Server.
For OLE DB	• Imports System.Data.OleDb namespace. • It provides data sources exposed using OLE DB. • We can use OLE DB for connect Microsoft access or Microsoft excel frequently.
For ODBC	• Imports System.Data.Odbc namespace. • It provides data source exposed using ODBC.
For Oracle	• Imports System.Data.oracleclient namespace. • It provides data access for oracle.

DataSource Interaction Objects

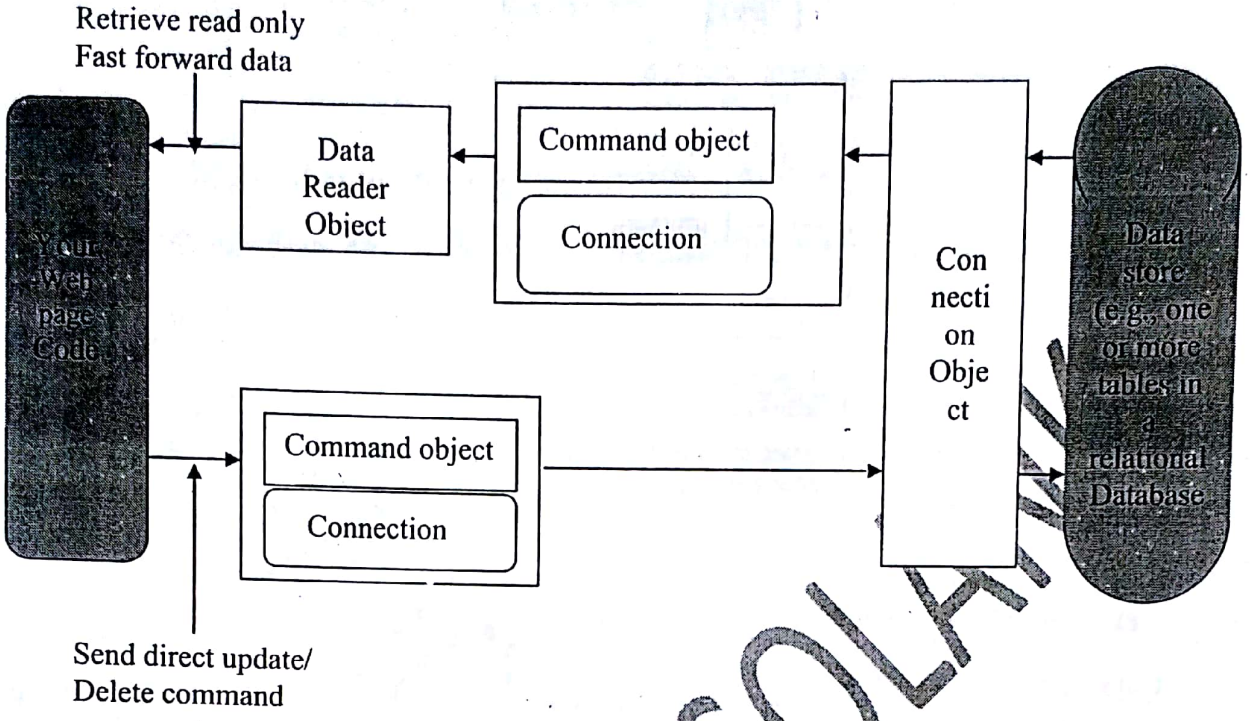


Fig:- Direct Data Access with ADO.NET

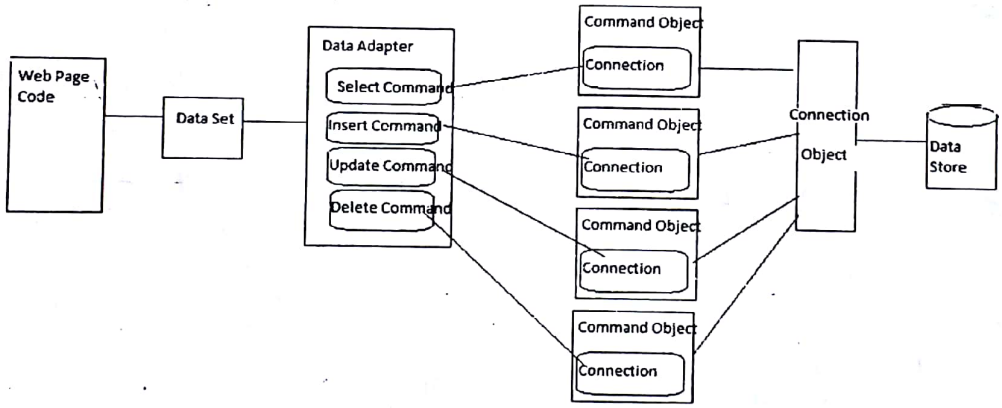
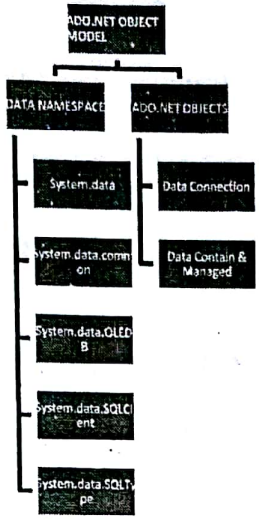


Fig:-Using a DataSet with ADO.NET

ADO.NET Object Model



1) **Data Namespaces**
2) **ADO.NET Objects**

1. **Data Namespaces :**

ADO.NET Components live in the five main namespaces in the .NET class Library.

(1) **System.data** : Contain fundamental classes with core ADO.NET functionality this include datasets and relation which allows manipulate structure relational data.

(2) **System.data.common :** These classes are not use directly in your code instead they are use by other classes in the system.data.SQLClient & system.data.OLEDB namespaces.

(3) **System.data.OLEDB :** Contain the classes you use to connect to the OLEDB provider including OLEDB command and OLEDB Connection.

(4) **System.data.SQLClient :** Contains the classes you used to connect to a Microsoft SQL server database. This classes like SQLDB Command and SQLDB Command.

(5) **System.data.SQLTypes :** Contain structure for SQL Server specific data type such as SQL many and SQLDateTime.

2. **ADO.NET OBJECTS :**

This object can be divided into two groups, those that are used to contain and managed data (dataset, data table, and data row & data relations) and those are used to connect specific data source (Connection, Command & Data Reader classes.)

The ADO.Net object in the second groups is provided two different categories….

(i) OLEDB Providers
(ii) Special Version for SQL Server.

Note: SQL Server by passes the OLEDB layer but OLEDB Providers supports all types of Languages & data base like Dbase, Sybase.

- **Data Objects :** The data objects allow you to the local disconnected copy of the data they don't stored connection to data store.

- **Data Sets :** The data sets class stores disconnected information drawn from database and allows you to manipulate it as a single object.

  Data set object contain two main types of objects
  (a) Data tables
  (b) Data Relations

Data set support following Methods:

1. **Write XML and Read XML :** Dataset to be serialized to an XML File.

2. **Merge:** Adds Contains of one dataset to another.

3. **Clear:** Remove all the information in the dataset.

4. **Accept changes, Get changes and Reject changes:** It allows you to work with the modified data, Examining, implementing or reverting change. **Note:** This method doesn't update the original database.

→ **Data Table: You** can add information into a dataset one table at a time. The data table object represents one of these tables.

→ **Data Row :** Each data row represent single row of information in the Table.

Methods of Data Row:
- Begin Edit
- End Edit
- Cancel Edit
- Delete
- Get Child Rows

→ **Data Column:** The data column objects don't contain any actual data. Instead, they store information about the column.

→ **Data Relation:** Each relation object specify of single parents, child relationship between two different tables into dataset.

→ **Data View:** Data view object provide window on to a table. Each data table has an at least one data view, which is used for data binding.

Data Reader vs DataSet

Data Reader	Dataset
Is directly connected to database system.	Is generally used to employ disconnected architecture of ado.net.
Can hold one table at a time.	Can hold multiple tables at a time. It represents a complete set of data including related tables, constraints, and relationships among the table.
Use multiple user updated data every time	If multiple users are using the database and the database needs to be updated every time, you must not use the dataset.
It provide read only data. We cannot update records.	We can make change in the dataset.
Data reader is like a forward only record	Dataset is scrollable.

set.	
No local storage is required.	It reads data form database and stores in local system.
Improve application performance.	Dataset is always a bulky object that requires lot of memory space compare to data reader.
It holds one row at a time is stored in memory.	It holds multiple records at a time.
Less Memory occupying	It occupies more memory.
No relationship can be maintained.	Relationship can be maintained.
No xml storage available.	Can be stored as xml.

DataBound Controls:

List Control:

- BulletedList
- CheckboxList
- DropDownList
- Listbox
- RadioButtonList

Tabular databound Controls

- Display a set of data
  - GridView
  - DataList
  - Repeter
- Display a single data item at a time
  - DetailsView

Hierarchical databound Controls

- Menu
- TreeView

Data Source Controls

You can use the Data source control in conjunction with a data-bound control to retrieve data from a relational database and to display, edit, and sort data on a web page with little or no code;

Represent tabular data

- SQLDataSource
- AccessDataSource
- ObjectDataSource

Represent tabular and hierarchical data

- XMLDataSource
- SiteMapDataSource

➔ **Connection Objects::**

- Connection Object creates the data base.
- Connection object represent physical connection to data source
- The connection object contains all of the information requires to open a Connection to the database.

- **Methods of Connection String :**
  1) Open
  2) Close
  3) Dispose
  4) State

➔ Data Access Steps ::

To retrieve simple data access follows these steps:
1. Create Connection , Commands and Data reader Objects
2. Use the data reader to retrieve information from the database and display it in a control on a web form.
3. Close the Connection
4. Send the Page to the user. (When we want update)

➔ Coding Steps ::

1. **Add Name Spaces :**

C#:
```
using system.data
using system.data. SQLClient
using System.data.OLEDB
```

VB.NET:
```
import system.data
import system.data. SQLClient
import System.data.OLEDB
```

2. **Connection String**

```
Dim myconnection As new OLEDBConnection()
```

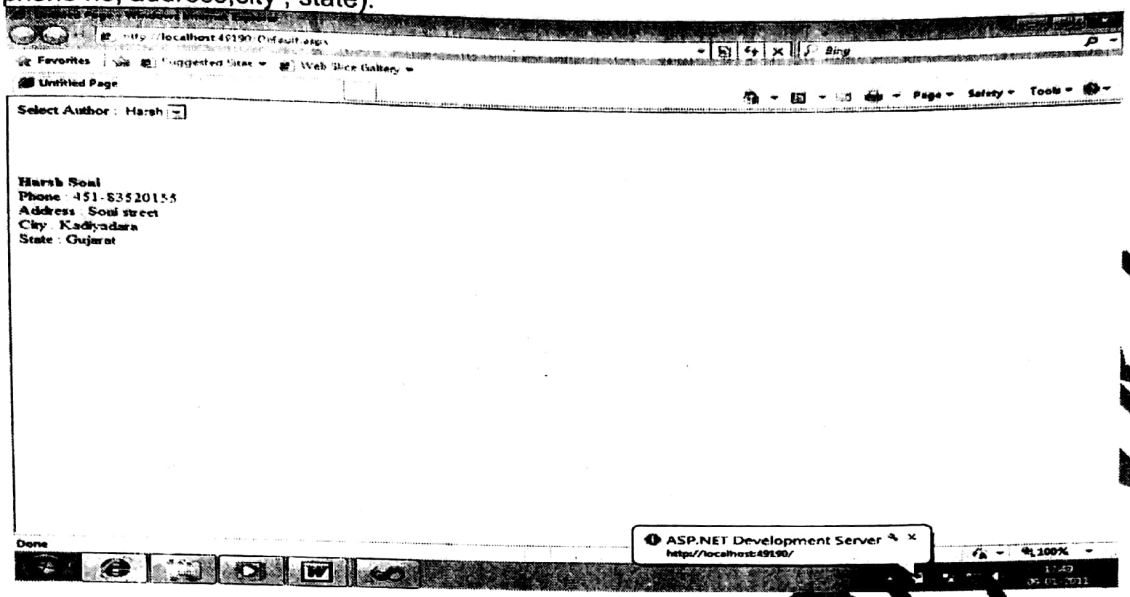
Myconnection.connectionString = "Provider = SQLOLEDB.1;"&DataSource=Localhost;&"Initial catalog= pubs;Integrated security = SSPI";

1. **Provider:** This is the name of the OLEDB Provider which allows Communication between ADO.NET and your database.
2. **Data Source:** This indicates the name of the server where the data source is located.
3. **Initial Catalogue:** This is the name of the database.
4. **User Id:** This is use to access the database.
5. **Password:** By Default "SA" account, doesn't have password.
6. **Connection time out:** This determines how long your code will wait, in seconds.
7. **Integrated Security:** In this case you could use connection string with the integrated security option.

   Two types of option:
   (1) SQL Server authentication
   (2) Integrated authentication

Lesson -1

When you select auther name from dropdownlistbox then label show all information of author.(first, last name, phone no, address,city , state).




```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Data;
using System.Data.SqlClient;

namespace DataAceess1
{
   public partial class _Default : System.Web.UI.Page
   {

SqlConnection con = new SqlConnection ("Data
Source=.\\SQLEXPRESS;AttachDbFilename=d:\\DataAceessfromcommand\\DataAceess\\App_Data\\Authord
atabase.mdf;Integrated Security=True;User Instance=True");

Private sub page_load(Object sender ,EventArgs e)
{
        If(me.Ispostback == false)
        {
                        FillAutherList();
        }
}

Private sub FillAutherList()
{
        lstAuthors.Iteams.Clear();
        String SelectSQL;
        SelectSQL = "Select au_lname,au_fname,au_id from authors";
```

```
SqlConnection  Con = new SqlConnection(ConnectionString);
SqLCommand cmd = new SqlCommand(SelectSql,con);
SqLDatareader reader;
```

➔ **Establishing the Connection :**

```
Try
{
        Con.open();
        Reader = cmd.Executereader();

        Do
        {
                ListItem newItem = new ListIem();
                newItem.text = reader("au_lname") & "," &reader("au_fname");
                newItem.value=reader("au_id);
                lstAuthor.Items.Add(newItem);
        }
        while(reader.read());
        reader.close();
}
Catch(Exception err)
{
        LbIresult.text="Error reading List of name...:" + err.Message;
}

Finally
{
        If(con != nothing)
        {
                Con.close();
        }
}
}
```

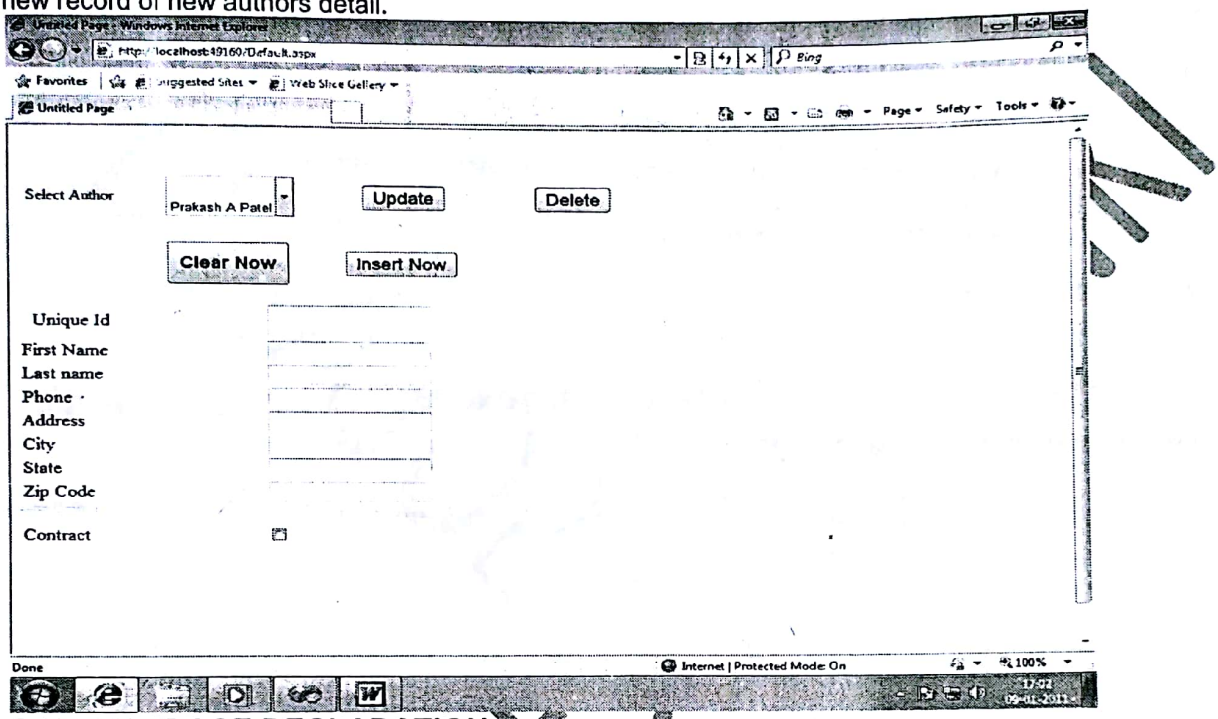
➔**Retrieving the Records :**

```
Private sub lstAuthor_SelectedIndexChanged(Object Sender, EventArgs e)

String SelectSql;
SelectSql = " Select * from Authors where au_id" + lstAuthor.selectedItem.value;

SqlConnection  Con = new SqlConnection(ConnectionString);
SqlCommand cmd = new SqlCommand(SelectSql,con);
SqlDatareader reader;

Try
{
        Con.open();
        Reader = cmd.ExecuteReader();
        Reader.read();
        LbIresult.text = "<b>" + reader("au_lname");
        LbIresult.text + =","+reader("au_fname") + "</b><br>";
        LbIresult.text + ="phone : " + reader("phone") +"<br>";
        LbIresult.text + ="Address : "+ reader("address") +"<br>";
        LbIresult.text + ="city : "+ reader("city") +"<br>";
        LbIresult.text + ="State : "+ reader("state") +"<br>";
        Reader.close();
}

Catch(Exception err)
{
        LbIresult.text = "Error Getting Author " + err.Message;
```

```
}
Finally
{
        Con.close();
}
}
```

Lesson -2

You can Update author detail by selecting from drop downlistbox and also Delete and can also insert(Add) new record of new authors detail.



→NAME SPACE DECLARATION :

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Data;
using System.Data.SqlClient;

namespace DataAceess
{
   public partial class _Default : System.Web.UI.Page
   {
```

//→PATH & DATABASE CONNECTION :

// we have use the path of express edition of sql server because we using the database of sql server which is inbuilt in the Visual Studio .Net 2008 edition.

Filling The List Box

```
SqlConnection con = new SqlConnection("Data
Source=.\\SQLEXPRESS;AttachDbFilename=d:\\DataAceessfromcommand\\DataAceess\\App_Data\\Authord
atabase.mdf;Integrated Security=True;User Instance=True");
    int Added;
    int del;
    protected void Page_Load(object sender, EventArgs e)
    {

      if (!IsPostBack)
      {
        fillAuthorname();
      }
    }
    private void fillAuthorname()
    {
      con.Open();
      string Selectsql;

      Selectsql = "select * from tb_author";
      SqlCommand cmd = new SqlCommand(Selectsql, con);
      SqlDataReader reader = cmd.ExecuteReader();
      //reader = cmd.ExecuteReader();
      while (reader.Read())
      {
        lstAuthor.Items.Add(reader.GetValue(1).ToString()+" "+reader.GetValue(2).ToString());
        // lstAuthor.Items.Add(reader.GetValue(2).ToString());
      }
      reader.Close();
      con.Close();
      //filldata();
    }
```

Retrieving The Record

```
  protected void lstAuthor_SelectedIndexChanged(object sender, EventArgs e)
  {
    if (IsPostBack)
    {
      filldata();
    }

  }
  private void filldata()
  {
    //lstAuthor.SelectedIndex.Equals
    string Str;
    Str = "select * from tb_author  where Uniquiid = ' " + Convert.ToInt16(lstAuthor.SelectedValue) + " ' ";
     SqlCommand cmd = new SqlCommand(Str , con);

  // SqlDataReader reader   SqlDataReader;

    try
    {
      con.Open();
     SqlDataReader reader = cmd.ExecuteReader();

        reader.Read();

{
```

```
txtUniqid.Text = reader.GetValue(0).ToString();
txtFirstname.Text = reader.GetValue(1).ToString();
txtLastname.Text = reader.GetValue(2).ToString();
txtPhone.Text = reader.GetValue(3).ToString();
txtAddress.Text = reader.GetValue(4).ToString();
txtState.Text = reader.GetValue(5).ToString();
txtZipcode.Text = reader.GetValue(7).ToString();
txtCity.Text = reader.GetValue(6).ToString();

        }
            reader.Close();
            lblStates.Text = "";
    }
      catch(Exception err)
    {
         lblStates.Text = "Error getting author + ";
           lblStates.Text = err.Message;
       }
      finally

    {
       con.Close();

      }

}
```

→A CODE FOR UPDATE THE DATABASE:

```
protected void btnUpdate_Click(object sender, EventArgs e)
{
   string updatesql;

    updatesql = "update tb_author set Uniquiid = ' " + txtUniqid.Text +" ' ," + "Firstname = ' " +
txtFirstname.Text + " ' ," + "Lastname = ' " + txtLastname.Text + " ' " + " where Uniquiid = ' " +
Convert.ToInt32(lstAuthor.SelectedIndex) + " ";

    SqlCommand cmd = new SqlCommand(updatesql, con);
    try
    {
       con.Open();
          int update;
       update = cmd.ExecuteNonQuery();
          lblStates.Text = update.ToString() + " Record Update";
    }
    catch (Exception err)
    {
       lblStates.Text = "Error Updating Method + ";
       lblStates.Text = err.Message;
    }
    finally
    {
       con.Close();
    }
}
```

→ A CODE FOR THE INSERT THE DATA INTO DATABASE :

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

```
        if (txtUniqid.Text != "" && txtFirstname.Text != "" && txtLastname.Text != "" && txtPhone.Text != "" &&
txtAddress.Text != "" && txtCity.Text != "")
        {
            string inserdata;
            inserdata = " insert into
tb_author(Uniquiid,Firstname,Lastname,Phone,Address,City,State,Zipcode) values('" + txtUniqid.Text + " ' , ' "
+ txtFirstname.Text + " ' , ' " +
                    txtLastname.Text + " ' , ' " + txtPhone.Text + " ' , ' " + txtAddress.Text + " ' , ' " +
txtCity.Text + " ' , ' " + txtState.Text + " ' , ' " + txtZipcode.Text + " ')";

            SqlCommand cmd = new SqlCommand(inserdata, con);

            try
            {
               con.Open();
               Added = cmd.ExecuteNonQuery();
               lblStates.Text = Added.ToString() + "Record inserted";
            }
            catch (Exception err)
            {
               lblStates.Text = "Error inserted Record +";
               lblStates.Text = err.Message;
            }
            finally
            {
               con.Close();
            }
            if (!IsPostBack)
            {

               if (Added > 0)
               {
                  fillAuthorname();
               }
            }
         }
         else
         {
            lblStates.Text = "Not Enter the data";
         }
      fillAuthorname();
   }
   private void fillClear()
   {
      txtUniqid.Text = "";
      txtFirstname.Text = "";
      txtLastname.Text = "";
      txtPhone.Text = "";
      txtAddress.Text = "";
      txtCity.Text = "";
      txtState.Text = "";
      txtZipcode.Text = "";
   }
   protected void btnCreatenow_Click(object sender, EventArgs e)
   {
      fillClear();
   }
```

→A CODE FOR DELETE THE RECORD FROM THE DATABASE :

```
   protected void btnDelete_Click(object sender, EventArgs e)
   {
      if (!IsPostBack)
```

```
        {
           string delete;
        delete = "delete from tb_author where Uniquiid = ' " + Convert.ToInt16(lstAuthor.SelectedIndex) + " ' ";

           SqlCommand cmd = new SqlCommand(delete, con);

           try
           {
              con.Open();
              del = cmd.ExecuteNonQuery();
              lblStates.Text = "Data is Delete";
           }
           catch (Exception err)
           {
              lblStates.Text = "Error detected in author + ";
              lblStates.Text = err.Message;
           }
           finally
           {
              con.Close();
           }
           if (!IsPostBack)
           {
            if (del > 0)
           {
              fillAuthorname();
           }
           }
         }

      }
   }
}
```

Lesson -3

Practical using DataAdaptor and Datasets.

```
String SelectSQL="Select au_lname,au_fname,au_id from Authors";

SqlConnection con = new SqlConnection (ConnectionString);
SqlCommand cmd = new SqlCommand (SelectSQL,con);
SqlDataAdapter adapter = new SqlDataAdapter (cmd);
Dataset dsPubs = new Dataset ();

Try
{       Con.open();
        Adapter.Fill(dsPubs,"Authors");

        cmd.CommandText = "Select au_id,title_id from TitleAuthor";
        adapter.Fill(dsPubs,"Titles");
}
Catch (Exception err)
{
        lblList.text="Error reading list of Names..."+err.Message;
}
Finally
{       Cn.close();
}
```

Overview of Data Binding

Data Binding :

You can use Dataset or the DataReder to retrieve information and add them to an HTML on a web page. It still requires a lot of repetitive code to move through the data and display it in the correct order. Repetitive code may be easy, but it's also difficult to enhance. So ASP.NET adds a feature that allows you to skip this process and pop data directly into HTML elements and fully formatted controls. It's called "DATA BINDING"

The basic principle of data binding is where to find your data and how you want it displayed, and the control handles the rest of the details. Data binding in ASP.NET is superficially similar to data binding in the world of client/server application, but fundamentally different. Data binding refers to the creation of a direct connection between a data source and a control in a application window.

If the user changes a value in the onscreen control, the data in the linked database is modified automatically. Similarly, if the database changes while the user is working with it, the display can be refreshed automatically.

ASP.NET data binding has little in common with direct data binding. ASP.NET Data binding works in one direction only. Information moves from a data object into control. Then the objects are thrown away and the page is sent to the client

ASP.NET data binding is much more flexible than traditional data binding. Many of the most powerful data binding controls, such as the Repeater, DataList and DataGrid, allows you to configure formatting options and even add repeating controls and buttons for each record.

Types of ASP.NET Data Binding :

=> There are two type of ASP.NET Data binding.

1>Single value or "Simple" Data Binding.
2>Repeated Value or List Binding.

Single value or "Simple" Data Binding

This type of data binding is used to add information anywhere on an ASP.NET page. You can even place information into a control property or as plain text inside an HTML tag.

Single value data binding is rarely of much use. Single value data binding allows you to take a variable, property, or expression and insert it dynamically into page.

Repeated Value or List Binding:

This type of data binding is much more useful for handling complex tasks, such as displaying an entire table or all the values from single field in a table. Unlike single value data binding, this type of data binding requires special controls that support it. Typically, this will be a list control like CheckBoxList or ListBox, but it can also be a much more sophisticated control like data grid.

You will know that a control supports repeated value data binding if it provides a DataSource property. As with single value binding, repeated value binding does not necessarily need to use data from a database, and it doesn't have to use the ADO.NET object.

For example

You can use repeated value binding to bind data a collection or an array.

How Data Binding Works :

Data binding works a little differently depending on whether you are using single value or repeated value binding.

In single value binding, a data binding expression is inserted right into the display portion of the .aspx file.

Syntax:- <%# expression_goes_here %>
Example:- <%# Country %>

In repeated value binding, data binding is configured by setting the appropriate control properties (For Ex. In the load event for the page).

Once you specify data binding, you need to activate it. You accomplish this task by calling the DataBind method. The DataBind method is basic piece of functionality supplied in the Control Class. It automatically binds a control and any child controls that it contains. Alternatively, you can bind the whole page at once by calling the DataBind method for the current page.

A Single Data Binding Example :

You start with a special variable define in your Page class, which is called **TransactionsCount**

```
Public Class DataBindingPage
   Inherits Page
     Public TransactionsCount as Integer
          '(Additional code omitted.)
End Class
```

Note that this variable must be designated as Public, Not Private. Otherwise, ASP.NET will not be able to access it when it is evaluating the data binding expression.

```
Private Sub Page_Load (sender as object, e as EventArgs)_Handles MyBase.Load
     'You could use database code here to lock up a value for TransactionsCount.
 TransactionsCount=10
     'Now convert all the data binding expressions on the page.
 Me.DataBind()
End Sub
```

NOTE:

Two Things actually happen in this event handler:The TransactionsCount variable is set to 10,and all the data binding expressions on the page are bound.This examplle Uses the "ME" keyword to refers to the current page.

If you are using Visual Studio.NET to create your controls, you should probably add a label control, and then configure the data binding expression in the HTML View by clicking HTML button.

You could also type this data binding expression in the Property Window; But Visual Studio.NET does not always refresh this information correctly. The Data binding expression have been replaced with the appropriate values.

Simple Data Binding With Properties:

You can also use single value data binding to set other types of information on your page, including control properties. To do this, you simply have to know where to put the data binding expression.

```
Public Class DataBindingPage
```

```
Inherits Page

 Public URL as String
 Private Sub Page_Load(sender as object,e as EventArgs) _
  Handles MyBase.Load
  URL=Server.MapPath("picture.jpg")
 End Sub
End Class
```

This URL can now be used to create a lable.

<asp:Label id=lblDynamic runat="server"><%# URL%></asp:Label>

Problem With Single Value Data Binding

You should consider some of the serious "Drawbacks" that this approach can present.

Putting code into a page's user interface: One of ASP.Net's great advantages is that it finally allows developers to separate the user interface code from the actual code used for data access and all other tasks. However, over enthusiastic use of single value data binding can encourage you to disregard that distinction and start coding function calls and even operations into your page?

Code Fragmentation: When data binding expression are used, it may not be obvious to other developers where the functionality resides for different operations. This is particularly problem if you blend both approaches. If the page code changes, or a variable or function is removed or renamed, the corresponding data binding expression could stop providing valid information without any explanation or even an obvious error.

No design time error checking: When you type a data binding expression, you need to remember the corresponding variable, property, or method name and enter it exactly. If you don't, you won't realize your mistakes until you try to run the page and you receive an error.

Repeated Value Data Binding :

While using simple data binding is an optional choice, repeated value binding is so useful that almost every ASP.NET application will want to make use of it. Repeated value data binding uses one of the special list controls included with ASP.NET. You link one of these controls to a data list source, and the control automatically creates a full list using all the corresponding values. This saves you from having to write code that loops through the array or data table and manually adds elements to control.

Repeated value data binding provide template options that automatically configure how the data should look when it is placed in the control. To create a data expression for list binding (repeated data value binding), you need to use a list control that explicitly supports data binding.

There are some controls like...

1>The ListBox, DropDownList, CheckBoxLIst, and RadioButtonList web control:
These controls provide a list for a single column of information.

2>The HtmlSelect server side HTML control:
This control represent the HTML <select> element, and works essentially the same as the ListBox web control.

3>The DataList, DataGrid, and Repeater controls:

These controls allow you to provide repeating lists or grids that can display more than one column (or field) of information at a time. For Ex. If you were binding to a full-fledged table in a Dataset, you could display multiple fields in any combination. These Control offer the most powerful and flexible options for data binding.

With repeated value data binding, you can write a data binding expression in your .aspx file, or you can apply the data binding by setting control properties.

Data Binding With Simple List Controls :

In some ways, data binding to a list control is the simplest kind of data binding. You only need to follow three steps:

1>Create and Fill some kind of data object.
2>Link the object to the appropriate control.
3>Activate the binding.

Example

Add a ListBox control to a new web page. Use the Page Load event handler to create a collection to use as a data source.

```
Dim colFruit As New Collection ()
    colFruit.Add("Kiwi")
    colFruit.Add("Pear")
    colFruit.Add("Mango")
    colFruit.Add("Blueberry")
    colFruit.Add("Apricot")
    colFruit.Add("Banana")
    colFruit.Add("Peach")
    colFruit.Add("Plum")
```

Now you can link this collection to the list box control.

```
    lstItems.DataSource = colFruit
```

To activate the binding, use the DataBind method.

```
Me.DataBind()
'Could also use lstItems.DataBind to bind just the list box.
```

Multiple Binding :

You can bind the same data list object to multiple different controls. Consider following example, which compares all the different types of list controls at your disposal by loading them with the same information.

```
Private Sub Page_Load(sender as object, e as EventArgs) _
Handles MyBase.Load

    'Create and Fill the Collection
     Dim colFruit As New Collection ()
    colFruit.Add("Kiwi")
    colFruit.Add("Pear")
    colFruit.Add("Mango")
    colFruit.Add("Blueberry")
    colFruit.Add("Apricot")
    colFruit.Add("Banana")
    colFruit.Add("Peach")
    colFruit.Add("Plum")
```

```
'Define the binding for the list controls.
MyListBox.DataSource = colFruit
MyDropDownListBox.DataSource = colFruit
MyHTMLSelect.DataSource = colFruit
MyCheckBoxList.DataSource = colFruit
MyRadioButtonList.DataSource = colFruit

'Activate the binding.
Me.DataBind ()
End Sub
```

Data Binding With Databases:

The greatest advantage from data binding is achieved when you use it in conjunction with a database. Remember, in order to work with databases you should import the related namespaces:

```
Imports System. Data
Imports System.Data.OleDb      'or System.Data.SqlDB
```

The Data Binding process steal takes plays in the same three steps with a database. First you create your data source, which will be a DataReader or DataSet object. A DataReader generly offers the best performance, but it limits your DataBinding to a single control, so a DataSet is a more common choice.

```
' Define a DataSet With a single DataTable.

Dim dsInternal as New DataSet ()
dsInternal.Tables.Add("Users")

' Define two columns for this table.
dsInternal.Tables("Users").columns.Add("Name")
dsInternal.Tables("Users").columns.Add("country")

'Add some actual information into table.
Dim rowNew as DataRow
rowNew=dsInternal.Tables("Users").NewRow ()
rowNew("Name")="John"
rowNew("country")="Uganda"
dsInternal.Tables("Users").Rows.Add(rowNew)

rowNew=dsInternal.Tables("Users").NewRow ()
rowNew("Name")="Samantha"
rowNew("country")="Belgium"
dsInternal.Tables("Users").Rows.Add(rowNew)

rowNew=dsInternal.Tables("Users").NewRow ()
rowNew("Name")="Rico"
rowNew("country")="Japan"
dsInternal.Tables("Users").Rows.Add(rowNew)
```

Next, a table from the DataSet Is bound to the appropriate control. In this case, you need to specify the appropriate field using the DataTextField property:

```
'Define the binding
lstUser.DataSource= dsInternal.Tables("Users")
lstUser.DataTextField ="Name"
```

Alternatively, you could use the entire DataSet for the datasource, instead of just the appropriate table. In this case, you have to select a table by setting the control's DataMember property. This is an equivalent approach, but the code looks slightly different:

```
'Define the binding
lstUser.DataSource = dsInternal
lstUser.DataMember = "Users"
lstUser.DataTextField = "Name"
```

As always,the last step is to activate the binding :

```
Me.DataBind()
'You could also use lstItems.DataBind to bind just the listbox.
```

THE END