ADO.Net

To access data from database like SQL server and oracle, .net framework provides ADO.Net. when we look at database technologies provided by Microsoft, they are as follows.

Database Technology	Description
DAO(Data Access Objects)	• This works based on JetEngine or DBEngine. • It is efficient in accessing desktop databases like MS Access, Foxpro and Excel. • It is not efficient in accessing remote databases like SQL Server and Oracle.
RDO(Remote Data Objects)	• This works based on ODBC(Open Database Connectivity). • It is efficient in accessing remote databases. • It can not access the databases that don't support ODBC.
ADO(Activex Data Objects)	• This works based on OLEDB (Object Linking and Embedding DataBase). • It can access almost any database that may be desktop database or remote database and that may support ODBC or may not support ODBC. • It is efficient than DAO and RDO. • It doesn't support pure disconnected architecture. • It doesn't support XML.
ADO.Net	To overcome the drawbacks of ADO, in .net Microsoft provides ADO.Net. Same as ADO it can access almost any database that may be desktop database or remote database and that may support ODBC or may not support ODBC. It supports pure disconnected architecture and XML.

Data Providers

ADO.net provides four data providers to work with various data sources and those data providers are as follows.

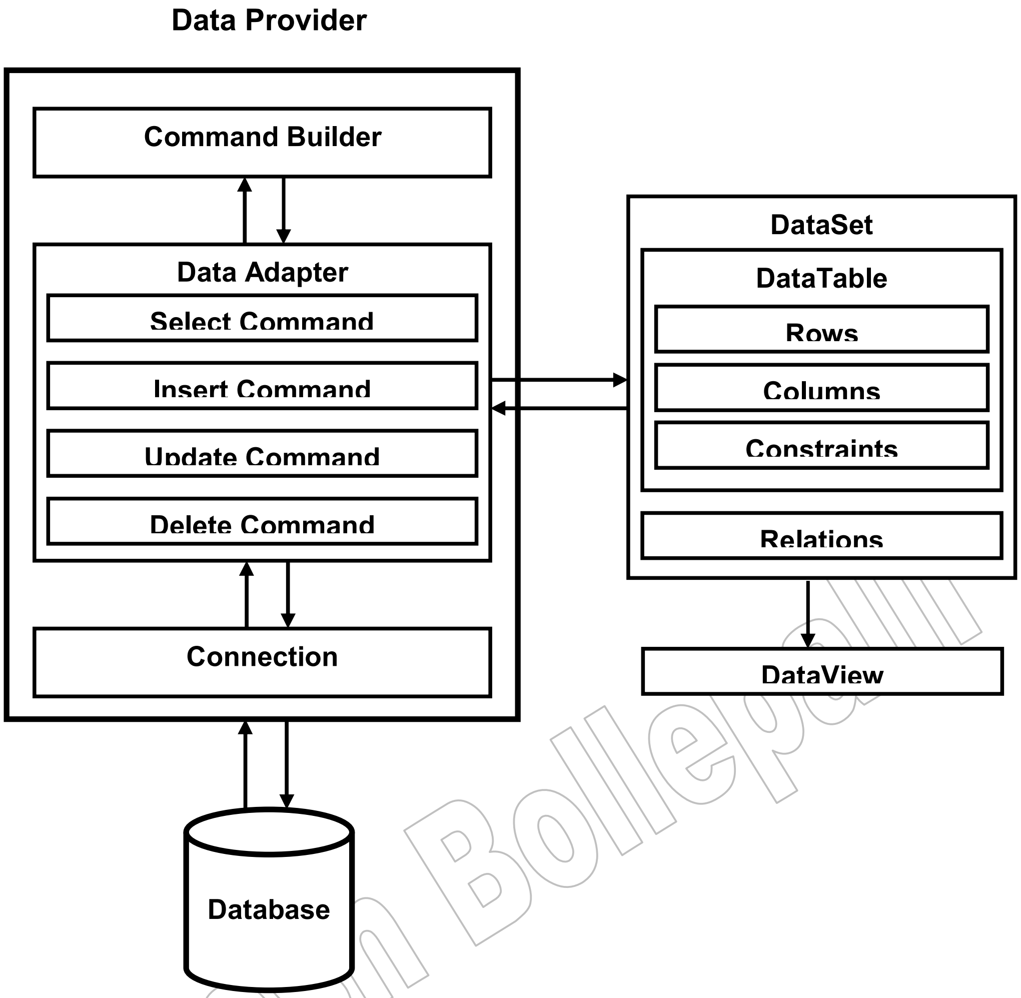
Data Provider	Description
SQLClient	Provides data access for Microsoft SQL Server version 7.0 or later versions. Uses the **System.Data.SqlClient** namespace.
OracleClient	For Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later, and uses the **System.Data.OracleClient** namespace.
OLEDB	For data sources exposed by using OLE DB. Uses the **System.Data.OleDb** namespace.
ODBC	For data sources exposed by using ODBC. Uses the **System.Data.Odbc** namespace.

ADO.Net Architecture

ADO.net supports both disconnected and connected architecture.

Disconnected Architecture

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 connection it 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't contain 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");

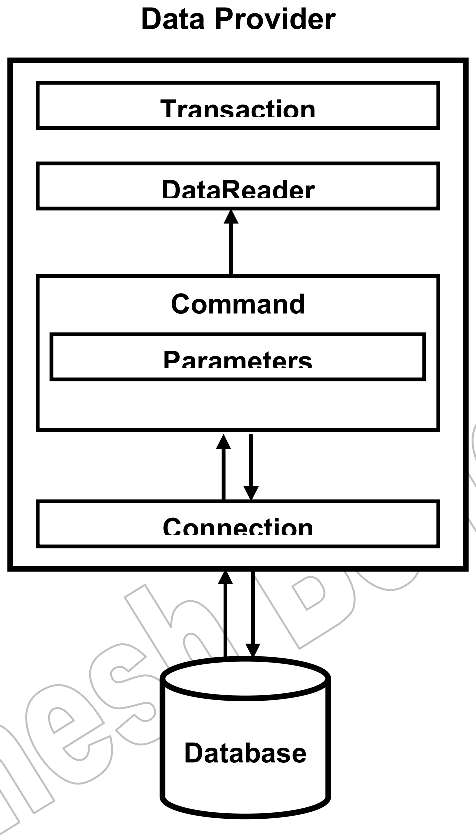
When Update method was called, dataadapter will again open the connection to database, executes insert, update and delete commands to send changes in dataset to database and immediately closes the connection. As connection is opened only when it is required and will be automatically closed when it was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

A dataset can contain data from multiple tables and each table may be from different data source like SQL Server, Oracle and MS Access. It has **Tables** collection that contains all the tables whose data was filled in to dataset and its data type is **DataTable**. Each table in Tables collection has **Rows, Columns** and **Constraints** collection that contains rows, columns and Constraints available in that table and their data type is **DataRow** and **DataColumn** and **ConstraintCollection** Respectively. Dataset along with Tables collection also contains **Relations** collection that contains information about **Foreign key** relation between tables available in the dataset.

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.

Connected Architecture

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 a connection to database and it self will not transfer any data.

Command : Command is used to execute almost any SQL command from within the .net application. The SQL command like insert, update, delete, select, create, alter, drop can be executed with command object and you can also call stored procedures with the command object. Command object has the following important properties.

1. **Connection :** used to specify the connection to be used by the command object.
2. **CommandType :** Used to specify the type of SQL command you want to execute. To assign a value to this property, use the enumeration **CommandType** that has the members **Text, StoredProcedure** and **TableDirect**. **Text** is the default and is set when you want to execute ant SQL command with command object. **StoredProcedure** is set

when you want to call a stored procedure or function and **TableDirect** is set when you want to retrieve data from the table directly by specifying the table name without writing a select statement.

3. **CommandText :** Used to specify the SQL statement you want to execute.
4. **Transaction :** Used to associate a transaction object to the command object so that the changes made to the database with command object can be committed or rollback.

Command object has the following important methods.

1. **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.
2. **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**.
3. **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.**

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.

Transaction : Transaction object is used to create a transaction and assign it to a command object so that the changes made to data with that command object can be committed or rollback. To create a transaction use **BeginTransaction()** method of the connection object.

Differences Between DataSet and DataReader

DataSet and DataReader are called fundamental objects of ADO.net as they are used to store data and make it available for .net application and they have the following differences.

DataSet	DataReader
It is disconnected object and can provide access to data even when connection to database was closed.	It is connected object and can not provide access to data when connection to database was closed.
It can store data from multiple tables	It can store data from only one table.
It allows insert, update and delete on data	It is read only and it doesn't allow insert, update and delete on data.
It allows navigation between record either forward or backward.	It allows only forward navigation that also only to immediate next record.
It can contain multiple records.	It can contain only one record at a time.
All the data of a dataset will be on client system.	All the data of a DataReader will be on server and one record at a time is retrieved and stored in datareader when you call the Read() method of datareader.

Creating Connection To Database

Creating Connection to SQLServer using SqlClient Data Provider

For this, use **SqlConnection** class available in the namespace **System.Data.SqlClient** that has the following syntax.

SqlConnection Cn = new SqlConnection("Connection String");

The **SqlConnection** class takes connection string as argument and the connection string for **SqlConnection** class has the following options.

1. **Server** or **Data Source :** Used to specify the name of the system in network that contains the database you want to access. When the database is available on same system on which you are working, then this option can be set to **localhost** or **.(dot)**. Otherwise this option can be completely eliminated from connection string.
2. **Database** or **Initial Catalog :** Used to specify the name of the database on server that you want to access.

3. **Integrated Security** or **Trusted_Connection :** This is a Boolean option and is used to specify whether windows authentication mode or SQL server authentication mode is used to login to SQL server. When this option was set to **true** then it indicates that you want to login to SQL server using windows authentication mode and **false** indicates that you want to login using SQL server authentication mode. When you are logging in windows authentication mode, no need to provide username and password.
4. **User Id** and **Password :** Used to specify user id and password required to login to SQL Server when the authentication mode was set to SQL Server authentication mode. When the authentication mode is windows, then no need to specify these options.

Creating Connection to Oracle using OracleClient Data Provider

For this, use **OracleConnection** class available in the namespace **System.Data.OracleClient** that has the following syntax.

OracleConnection Cn = new OracleConnection("Connection String");

The **OracleConnection** class takes connection string as argument and the connection string for **OracleConnection** class has the following options.

1. **Server** or **Data Source :** Used to specify the global Sid of the oracle database you want to access. When the database is available on same system on which you are working, then this option can be set to **localhost** or **.(dot)**. Otherwise this option can be completely eliminated from connection string.
2. **User Id** and **Password :** Used to specify user id and password required to login to SQL Server when the authentication mode was set to SQL Server authentication mode. When the authentication mode is windows, then no need to specify these options.

Database option is not available for connection string of **OracleConnection** class. Because in oracle, the global sid of the database it self is the name of the database. Oracle doesn't support windows authentication and hence it is mandatory to specify **user id** and **password**.

Creating Connection to SQLServer using Oledb Data Provider

For this, use **OledbConnection** class available in the namespace **System.Data.Oledb** that has the following syntax.

OledbConnection Cn = new OledbConnection("Connection String");

The **OledbConnection** class takes connection string as argument and the connection string for **OledbConnection** class has the following options.

1. **Provider :** Used to specify the Oledb provider to use to connect to the database. For Sql Server, **SqlOledb** provider is used and hence set this option to **SqlOledb** while connecting to SQL Server.
2. **Server** or **Data Source :** Used to specify the name of the system in network that contains the database you want to access. When the database is available on same system on which you are working, then this option can be set to **localhost** or **.(dot)**. Otherwise this option can be completely eliminated from connection string.
3. **Database** or **Initial Catalog :** Used to specify the name of the database on server that you want to access.
4. **Integrated Security** or **Trusted_Connection :** This is a Boolean option and is used to specify whether windows authentication mode or SQL server authentication mode is used to login to SQL server. When this option was set to **true** then it indicates that you want to login to SQL server using windows authentication mode and **false** indicates that you want to login using SQL server authentication mode. When you are logging in windows authentication mode, no need to provide username and password.
5. **User Id** and **Password :** Used to specify user id and password required to login to SQL Server when the authentication mode was set to SQL Server authentication mode. When the authentication mode is windows, then no need to specify these options.

Creating Connection to Oracle using Oledb Data Provider

For this, use **OledbConnection** class available in the namespace **System.Data.Oledb** that has the following syntax.

```
OledbConnection Cn = new OledbConnection(“Connection String”);
```

The **OledbConnection** class takes connection string as argument and the connection string for **OledbConnection** class has the following options.

1. **Provider :** Used to specify the Oledb provider to use to connect to the database. For Oracle, **MSDAORA** provider is used and hence set this option to **MSDAORA** while connecting to Oracle.
2. **Server** or **Data Source :** Used to specify the global Sid of the oracle database you want to access. When the database is available on same system on which you are working, then this option can be set to **localhost** or **.(dot)**. Otherwise this option can be completely eliminated from connection string.
3. **User Id** and **Password :** Used to specify user id and password required to login to SQL Server when the authentication mode was set to SQL Server authentication mode. When the authentication mode is windows, then no need to specify these options.

Database option is not available for connection string of **OracleConnection** class. Because in oracle, the global sid of the database it self is the name of the database. Oracle doesn’t support windows authentication and hence it is mandatory to specify **user id** and **password**.

Creating Connection to MSAccess using Oledb Data Provider

For this, use **OledbConnection** class available in the namespace **System.Data.Oledb** that has the following syntax.

```
OledbConnection Cn = new OledbConnection(“Connection String”);
```

The **OledbConnection** class takes connection string as argument and the connection string for **OledbConnection** class has the following options.

1. **Provider :** Used to specify the Oledb provider to use to connect to the database. For MS Access, **Microsoft.Jet.Oledb.4.0** provider is used and hence set this option to **Microsoft.Jet.Oledb.4.0** while connecting to MS Access.
2. **Data Source :** Used to specify the name and path of the database file of MS Access.
3. **User Id** and **Password :** Used to specify user id and password required to login to OMS Access.

Database option is not available for connection string of **OledbConnection** while connecting to MS Access database. When MS Access database was protected with user id and password then only you have to specify the user id and password and other wise you can eliminate these options from connection string.

Server Explorer

Server explorer is a new tool provided in visual studio.net 2005 that provides a list of servers available for the system and allow the user to create a connection to database and access the tables, view and stored sub programs in the database. Shortcut to open server explorer is Ctrl + Alt + S.

Creating Connection to Database

To create a connection to database using server explorer, follow the following steps.

1. Within the server explorer click on **connect to database** button or right click on **data connections** and choose **add connection** option to open **add connection dialog box.**
2. In the add connection dialog box, select a data source i.e SQL Server or Oracle or MS Access by clicking on change button, provide connection string information required to create a connection and click on ok button to add connection to server explorer.

Accessing Database Objects

To access database objects like tables, views and stored procedures, expand the connection and then expand the corresponding node. To access a table, expand tables node that displays available tables in that database. Right click on a table and choose **show table data** option to get a list of all rows available in the table. From this table data window it self you can perform insert, update and delete on the table. When the data source you selected is SQL server, you can also create new tables, views and stored sub programs from within server explorer. But this is not possible with other data sources like oracle and MS Access.

Declarative Access to Database

Accessing data from database without writing a single line of code is called as declarative access to database. For declarative access to database, first you have to configure a data source and to configure the data source, follow the following steps.

1. Start the data source configuration wizard, choose **add new datasource** option from **data** menu**.**
2. In the first step of the wizard, select **database** option and click on next button.
3. In the second step of the wizard, click on **new connection** button to create a connection to database and after creating the connection click on next button.
4. In the third step of the wizard, provide a name to save the connection string information in configuration file and click on next button.
5. In the last step of the wizard, select tables for which you want to create the data source by checking the check box left to it, provide a name for the data set to be created for the data source and click on finish button.

Example : The following example provides declarative access to Dept table available in MyDb database of SQL Server.

1. Create a windows application, add a data source to the project by choosing **add new datasource** option in the data menu by creating a connection to MyDb database, and selecting the table Dept and by providing the dataset name as DsDept.
2. Open Data Sources window by using the short cut **Shift + Alt + D.**
3. Within the data sources window click on the arrow key right to Dept table and choose **Details**.
4. Drag and Drop the table Dept from data sources window on to the form, which will automatically create required controls like labels and text boxes for displaying data and also creates the controls like BindingSource and BindingNavigator.

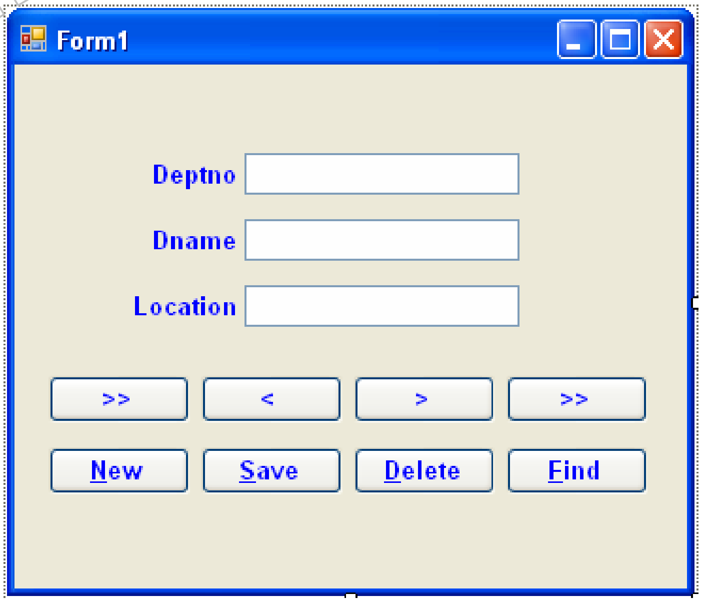
5. BindingSource is used to provide source of data for the controls on the form from table and BindingNavigator provides buttons for navigating between records and to perform insert, update and delete.
6. To add a record, click on **add new** button in BindingNavigator, enter the values for new record in controls on the form and click on **Save** button in BindingNavigator.
7. To Update a record, navigate to the record with the help of first, next, previous and last buttons in the BindingNavigator, make required changes to data and click on **Save** button in BindingNavigator.
8. To delete a record, navigate to the record with the help of navigation buttons in indingNavigator and click on delete button followed by save button in BindingNavigator.

Programmatic Access to Database

Accessing the data in database by writing code is called as programmatic access to the database.

Example : The following example provides programmatic access to Dept table available in MyDb Database of SQl Server using Disconnected Architecture of ADO.net.

1. Create a Windows Application and design the form as follows.



Set the following properties to the controls on the form.

TextBox1 Name : TxtDno

TextBox2	Name	: TxtDname
TextBox3	Name	: TxtLoc
Button1	Name	: BtnFirst
	Text	: <<
Button2	Name	: BtnPrev
	Text	: <
Button3	Name	: BtnNext
	Text	: >
Button4	Name	: BtnLast
	Text	: >>
Button5	Name	: BtnNew
	Text	: &New
Button6	Name	: BtnSave
	Text	: &Save
Button7	Name	: BtnDelete
	Text	: &Delete
Button8	Name	: BtnFind
	Text	: &Find

2. Within the code of the form write using statement to add reference to System.Data.SqlClient namespace and then create a connection, Data Adapter, Command Builder DataSet, DataRow, integer and bool variables as follows within the class of the form.

```
static SqlConnection Cn = new
   SqlConnection("Server=.;Database=Mydb;User Id=sa;password=abc");
static SqlDataAdapter Da = new SqlDataAdapter("Select * From Dept", Cn);
static SqlCommandBuilder Cmb = new SqlCommandBuilder(Da);
static DataSet Ds = new DataSet();
static DataRow Dr;
static int Pos;
static bool NewRec;
```

3. Create method with the name GetRow() to display the values of a row with given index from dataset in to text boxes on the form.

```
public void GetRecord(int P)
{
   TxtDno.Text = Ds.Tables["Dept"].Rows[P][0].ToString();
   TxtDname.Text = Ds.Tables["Dept"].Rows[P][1].ToString();
   TxtLoc.Text = Ds.Tables["Dept"].Rows[P][2].ToString();
}
```

4. Write the following code within the form load event to fill data from database in to dataset and display first record in to textboxes.

```
private void Form1_Load(object sender, EventArgs e)
{
   Da.Fill(Ds, "Dept");
   Pos = 0;
   NewRec = false;
   GetRecord(Pos);
   BtnFirst.Enabled = false;
   BtnPrev.Enabled = false;
}
```

5. Write the following code within the first, previous, next and last button click events to navigate between records.

```
private void BtnFirst_Click(object sender, EventArgs e)
{
   Pos = 0;
   GetRecord(Pos);
   BtnFirst.Enabled = false;
   BtnPrev.Enabled = false;
   BtnNext.Enabled = true;
   BtnLast.Enabled = true;
}

private void BtnPrev_Click(object sender, EventArgs e)
{
   Pos--;
   GetRecord(Pos);
   if (Pos == 0)
   {
         BtnFirst.Enabled = false;
         BtnPrev.Enabled = false;
   }
   BtnNext.Enabled = true;
   BtnLast.Enabled = true;
}

private void BtnNext_Click(object sender, EventArgs e)
{
   Pos++;
   GetRecord(Pos);
   if (Pos == Ds.Tables["Dept"].Rows.Count - 1)
   {
         BtnNext.Enabled = false;
         BtnLast.Enabled = false;
   }
   BtnFirst.Enabled = true;
   BtnPrev.Enabled = true;
}

private void BtnLast_Click(object sender, EventArgs e)
{
   Pos = Ds.Tables["Dept"].Rows.Count - 1;
   GetRecord(Pos);
   BtnFirst.Enabled = true;
   BtnPrev.Enabled = true;
   BtnNext.Enabled = false;
   BtnLast.Enabled = false;
}
```

6. Write the following code within new button click event to clear the text boxes and place cursor in first text box to allow the user to enter values for new record.

```
private void BtnNew_Click(object sender, EventArgs e)
{
   NewRec = true;
   TxtDno.Clear();
   TxtDname.Clear();
   TxtLoc.Clear();
   TxtDno.Focus();
   BtnNew.Enabled = false;
}
```

7. Write the following code within the Save button click event to insert or update the record based on the value in NewRec Boolean variable.

```
private void BtnSave_Click(object sender, EventArgs e)
{
   if (NewRec == true)
   {
         Dr = Ds.Tables["Dept"].NewRow();
         Dr[0] = TxtDno.Text;
         Dr[1] = TxtDname.Text;
         Dr[2] = TxtLoc.Text;
         Ds.Tables["Dept"].Rows.Add(Dr);
   }
   else
   {
         Dr = Ds.Tables["Dept"].Rows[Pos];
         Dr[0] = TxtDno.Text;
         Dr[1] = TxtDname.Text;
         Dr[2] = TxtLoc.Text;
   }
   Da.Update(Ds, "Dept");
   MessageBox.Show("One Row Saved");
   BtnFirst_Click(sender, e);
   BtnNew.Enabled = true;
}
```

8. Write the following code within Delete button click event to delete the record.

```
private void BtnDelete_Click(object sender, EventArgs e)
{
   DialogResult Res;
   Res = MessageBox.Show("Do You Really Want To Delete The Row?",
                  "Confirmation To Delete", MessageBoxButtons.YesNo,
                                          MessageBoxIcon.Question);
   if (Res == DialogResult.Yes)
   {
```

```
            Ds.Tables["Dept"].Rows[Pos].Delete();
            Da.Update(Ds, "Dept");
            MessageBox.Show("One Row Deleted");
            BtnFirst_Click(sender, e);
        }
    }
```

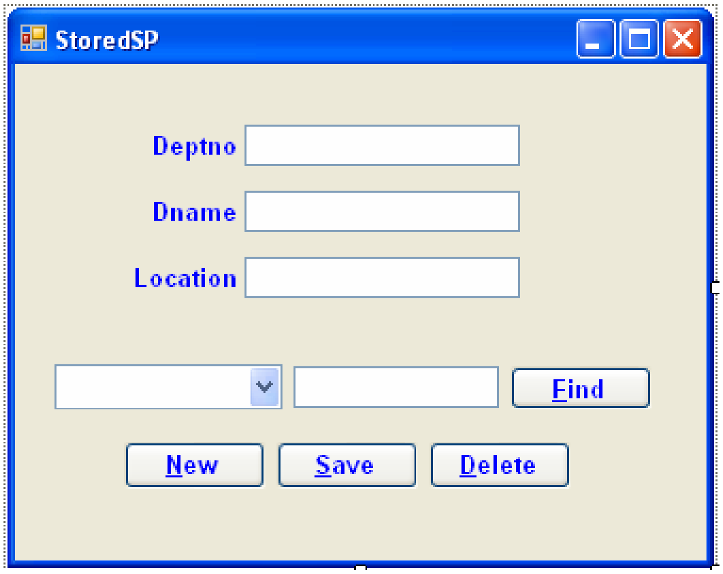
9. Write the following code within the find button click event to find a row based on the deptno entered in deptno textbox.

```
private void BtnFind_Click(object sender, EventArgs e)
{
    try
    {
            Dr = Ds.Tables["Dept"].Select("Deptno=" + TxtDno.Text)[0];
            Pos = Ds.Tables["Dept"].Rows.IndexOf(Dr);
            GetRecord(Pos);
    }
    catch (Exception Ex)
    {
            MessageBox.Show("Row Not Found");
            TxtDno.Clear();
            TxtDname.Clear();
            TxtLoc.Clear();
    }
}
```

10. Run the application using the short cut F5.

Example : The following example provides programmatic access to Dept table available in Oracle Database using connected Architecture of ADO.net.

1. Add a Win Form to the project and design it as follows.



Set the following properties to the controls on the form

TextBox1	Name : TxtDno
TextBox2	Name : TxtDname
TextBox3	Name : TxtLoc
ComboBox1	Name : CmbColumn
TextBox4	Name : TxtFind
Button1	Name : BtnFind Text : &Find Enabled : false
Button2	Name : BtnNew Text : &New
Button3	Name : BtnSave Text : &Save
Button4	Name : BtnDelete Text : &Delete

2. Add the reference to OracleClient namespace. For this right click on project in solution explorer and choose add reference option that opens **add reference** dialog box. In this dialog box select **System.Data.OracleClient** and click on ok button.

3. Within the code write using statement to add the reference to System.Data.OracleClient namespace and then create a connection, Command, a Boolean variable as follows.

```
static OracleConnection Cn = new OracleConnection("Server=NetSoft;User
                                             id=scott;password=tiger");
static OracleCommand Cmd = new OracleCommand();
static bool NewRec;
```

4. Write the following code within form load event to fill column details in to the combo box cmbcolumn.

```
private void FrmConnected_Load(object sender, EventArgs e)
{
   try
   {
   OracleDataAdapter Da = new OracleDataAdapter("Select * From Dept", Cn);
   DataSet Ds = new DataSet();
   Da.Fill(Ds, "Dept");
   foreach (DataColumn C in Ds.Tables["Dept"].Columns)
   {
         CmbColumn.Items.Add(C.ToString());
   }
   NewRec = false;
   Cmd.Connection = Cn;
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
}
```

5. Write the following code within the text changed event of txtfind text box to enable or disable the find button based on number of characters in txtfind text box.

```
private void TxtFind_TextChanged(object sender, EventArgs e)
{
   if (TxtFind.Text.Trim().Length > 0)
         BtnFind.Enabled = true;
   else
         BtnFind.Enabled = false;
}
```

6. Write the following code within the find button click event to find the record based on selected column in combo box and the value entered in the text box txtfind.

```
private void BtnFind_Click(object sender, EventArgs e)
{
   try
   {
   Cmd.CommandText = "Select * from dept Where " + CmbColumn.Text + "='" +
                                                   TxtFind.Text + "'";
   Cn.Open();
   OracleDataReader Dr = Cmd.ExecuteReader();
   if (Dr.Read())
   {
         TxtDno.Text = Dr[0].ToString();
         TxtDname.Text = Dr[1].ToString();
         TxtLoc.Text = Dr[2].ToString();
   }
   else
   {
         MessageBox.Show("No Row Found");
         TxtDno.Clear();
         TxtDname.Clear();
         TxtLoc.Clear();
   }
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
   finally
   {
         Cn.Close();
   }
}
```

7. Write the following code within the new button click event to automatically generate new deptno and indicate user wants to insert new record.

```
private void BtnNew_Click(object sender, EventArgs e)
{
   try
   {
   Cmd.CommandText = "Select NVL(max(Deptno),0)+10 from dept";
   Cn.Open();
   TxtDno.Text = Cmd.ExecuteScalar().ToString();
   TxtDname.Clear();
   TxtLoc.Clear();
   TxtDname.Focus();
   NewRec = true;
   BtnNew.Enabled = false;
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
   finally
   {
         Cn.Close();
   }
}
```

8. Write the following code within the save button click event to insert or update the record.

```
private void BtnSave_Click(object sender, EventArgs e)
{
   try
   {
         if (NewRec == true)
         {
         Cmd.CommandText = "Insert into Dept Values(" + TxtDno.Text + ",'"
                           + TxtDname.Text + "','" + TxtLoc.Text + "')";
         }
         else
         {
         Cmd.CommandText = "Update Dept Set Dname='" + TxtDname.Text +
               "',Loc='" + TxtLoc.Text + "' where Deptno=" + TxtDno.Text;
         }
         Cn.Open();
         Cmd.ExecuteNonQuery();
         MessageBox.Show("One Row Saved");
         BtnNew.Enabled = true;
         NewRec = false;
    }
    catch (Exception Ex)
    {
         MessageBox.Show(Ex.Message);
    }
    finally
    {
         Cn.Close();
    }
}
```

9. Write the following code within the delete button click event to delete the row.

```
private void BtnDelete_Click(object sender, EventArgs e)
{
   try
   {
         DialogResult Res;
         Res = MessageBox.Show("Do You Really Want To Delete The Row?",
               "Confirmation To Delete", MessageBoxButtons.YesNo,
                                       MessageBoxIcon.Question);
         if (Res == DialogResult.Yes)
         {
         Cmd.CommandText = "Delete Dept Where Deptno=" + TxtDno.Text;
         Cn.Open();
         Cmd.ExecuteNonQuery();
         MessageBox.Show("One Row Deleted");
         TxtDno.Clear();
         TxtDname.Clear();
         TxtLoc.Clear();
         }
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
   finally
   {
         Cn.Close();
   }
}
```

9. Run the application using the short cut F5.

Calling Stored Sub Programs : Writing individual SQL statements as in previous example is not recommended as it effects the performance of the application and recommended option is to create stored sub programs and call them. When you are executing individual SQL statements, before executing statement SQL has to perform various steps like **parsing, Execution plan generation** and **Cost estimation**. During parsing it will check for object availability, permissions for the current user on the object, syntax of the statement and whether the data type of values in SQL statement are same as the data type of column in table. All these steps will take time and hence execution will be slow and will affect your .net application performance.

When you are creating a stored a sub program then during creation it self parsing, execution plan generation and cost estimation will be done for every statement in the sub program and along with the compiled code, execution plan with low cost for every statement will be stored in database. As execution plan is readily available in the database for a stored sub program, execution will be fast when compared to executing individual SQL statement.

Example : The following example provides programmatic access to Dept table available in MyDb Database of SQL Server using connected Architecture of ADO.net by creating stored sub programs.

1. Create four stored sub programs in MyDb database of SQL Server, NewDno() for automatic generation of new deptno, InsertDept() for insertion, UpdateDept() for updating and DeleteDept() for deletion as follows.

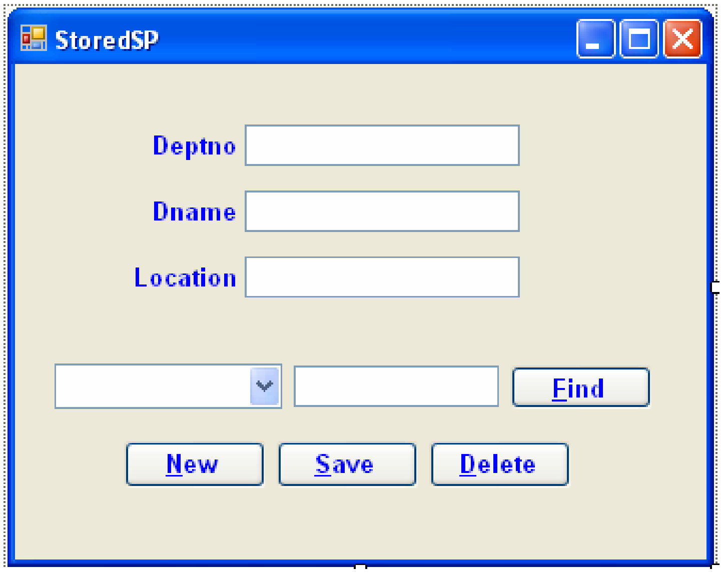
```
create function NewDno() returns int as
begin
      declare @Ndno int
      select @Ndno=isnull(max(Deptno),0)+10 from dept
      return @Ndno
end


create procedure InsertDept(@Dno int,@Dn varchar(30),@L varchar(20)) as
begin
      Insert into dept values(@Dno,@Dn,@L)
end


Create procedure UpdateDept(@Dno int,@Dn varchar(30),@L varchar(20)) as
begin
      Update Dept Set Dname=@Dn,Loc=@L where Deptno=@Dno
end


Create procedure DeleteDept(@Dno int) as
begin
      Delete Dept Where Deptno=@Dno
end
```

2. Add a Win Form to the project and design it as follows.



Set the following properties to the controls on the form

TextBox1	Name : TxtDno
TextBox2	Name : TxtDname
TextBox3	Name : TxtLoc
ComboBox1	Name : CmbColumn
TextBox4	Name : TxtFind
Button1	Name : BtnFind
	Text : &Find
	Enabled : false
Button2	Name : BtnNew
	Text : &New
Button3	Name : BtnSave
	Text : &Save
Button4	Name : BtnDelete
	Text : &Delete

3. Within the code write using statement to add the reference to System.Data.OracleClient namespace and then create a connection, Command, a Boolean variable as follows.

```
static SqlConnection Cn = new SqlConnection("Server=.;Database=mydb;User
                                                    id=sa;password=abc");
static SqlCommand Cmd = new SqlCommand();
static bool NewRec;
```

4. Write the following code within the form load event to get column names in dept table and add them to the combo box.

```
private void StoredSP_Load(object sender, EventArgs e)
{
   try
   {
         Cmd.Connection = Cn;
         NewRec = false;
         SqlDataAdapter Da = new SqlDataAdapter("Select * From Dept", Cn);
         DataSet Ds = new DataSet();
         Da.Fill(Ds, "Dept");
         foreach (DataColumn C in Ds.Tables["Dept"].Columns)
         {
               CmbColumn.Items.Add(C.ToString());
         }
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
}
```

5. Write the following code in textchanged event of the text box txtfind to enable or disable the find button based on no. of characters in text box.

```
private void TxtFind_TextChanged(object sender, EventArgs e)
{
   if (TxtFind.Text.Trim().Length > 0)
         BtnFind.Enabled = true;
   else
         BtnFind.Enabled = false;
}
```

6. Write the following code within find button click event to find the record based on column selected in cmbcolumn combo box and the value entered in txtfind text box.

```
private void BtnFind_Click(object sender, EventArgs e)
{
   try
   {
         Cmd.CommandType = CommandType.Text;
         Cmd.CommandText = "Select * From Dept Where " + CmbColumn.Text +
                                                "='" + TxtFind.Text + "'";
         Cn.Open();
         SqlDataReader Dr = Cmd.ExecuteReader();

         if (Dr.Read())
         {
               TxtDno.Text = Dr[0].ToString();
               TxtDname.Text = Dr[1].ToString();
               TxtLoc.Text = Dr[2].ToString();
         }
         else
         {
               MessageBox.Show("Row Not Found");
               TxtDno.Clear();
               TxtDname.Clear();
               TxtLoc.Clear();
         }
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
   finally
   {
         Cn.Close();
   }
}
```

7. Write the following code within the click event of new button to automatically generate a new deptno by calling NewDno() stored sub program and display it in TxtDno text box.

```
private void BtnNew_Click(object sender, EventArgs e)
{
   try
   {
         Cmd.CommandType = CommandType.StoredProcedure;
         Cmd.CommandText = "NewDno";
         Cmd.Parameters.Clear();
         Cmd.Parameters.Add("NDno", SqlDbType.Int);
         Cmd.Parameters["NDno"].Direction =
                                       ParameterDirection.ReturnValue;
         Cn.Open();
         Cmd.ExecuteNonQuery();
         TxtDno.Text = Cmd.Parameters["NDno"].Value.ToString();
         TxtDname.Clear();
         TxtLoc.Clear();
         TxtDname.Focus();
         NewRec = true;
         BtnNew.Enabled = false;
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
   finally
   {
         Cn.Close();
   }
}
```

8. Write the following code within the click event of save button to insert or update the record based on the value of newrec Boolean variable by calling InsertDept() or DeleteDept() stored sub program.

```
private void BtnSave_Click(object sender, EventArgs e)
{
   try
   {
         Cmd.CommandType = CommandType.StoredProcedure;
         if (NewRec == true)
         {
               Cmd.CommandText = "InsertDept";
         }
         else
         {
               Cmd.CommandText = "UpdateDept";
         }
         Cmd.Parameters.Clear();
         Cmd.Parameters.Add("Dno", SqlDbType.Int);
         Cmd.Parameters.Add("Dn", SqlDbType.VarChar);
         Cmd.Parameters.Add("L", SqlDbType.VarChar);
         Cmd.Parameters["Dno"].Value = TxtDno.Text;
         Cmd.Parameters["Dn"].Value = TxtDname.Text;
         Cmd.Parameters["L"].Value = TxtLoc.Text;
         Cn.Open();
         Cmd.ExecuteNonQuery();
         MessageBox.Show("One Row Saved");
         NewRec = false;
         BtnNew.Enabled = true;
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
   finally
   {
         Cn.Close();
   }
}
```

9. Write the following code within the delete button click event to delete the record by calling DeleteDept() stored sub program.

```
private void BtnDelete_Click(object sender, EventArgs e)
{
   try
   {
         Cmd.CommandType = CommandType.StoredProcedure;
         DialogResult Res;
         Res=MessageBox.Show("Do You Really Want To Delete The Row?",
                     "Confirmation To Delete", MessageBoxButtons.YesNo,
                                             MessageBoxIcon.Question);
         if (Res == DialogResult.Yes)
         {
               Cmd.CommandText = "DeleteDept";
               Cmd.Parameters.Clear();
               Cmd.Parameters.Add("Dno", SqlDbType.Int);
               Cmd.Parameters["Dno"].Value = TxtDno.Text;
               Cn.Open();
               Cmd.ExecuteNonQuery();
               MessageBox.Show("One Row Deleted");
               TxtDno.Clear();
               TxtDname.Clear();
               TxtLoc.Clear();
         }
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
   finally
   {
         Cn.Close();
   }
}
```

10. Run the application using the short cut F5.

To call a stored sub program follow the following steps.

1. Set **CommandType** property of the command object to **CommandType.StoredProcedure.**
2. Set **CommandText** property of command object to the name of stored sub program you have to call.
3. Clear existing parameters if any by calling **Clear()** method of **Parameters** collection of command object.
4. Add required parameters for calling the stored sub program by using **Add()** method of **Parameters** collection of command object.
5. If any parameter is created for holding the return value from the stored sub program, set its **Direction** property to **ParameterDirection.ReturnValue.**
6. If parameters are created to pass value to the stored sub program, then assign values to those parameters from controls on the form, which you want to pass to the stored sub program.
7. Open the connection and call **ExecuteNonQuery()** method of the command object.

Storing And Retrieving Images

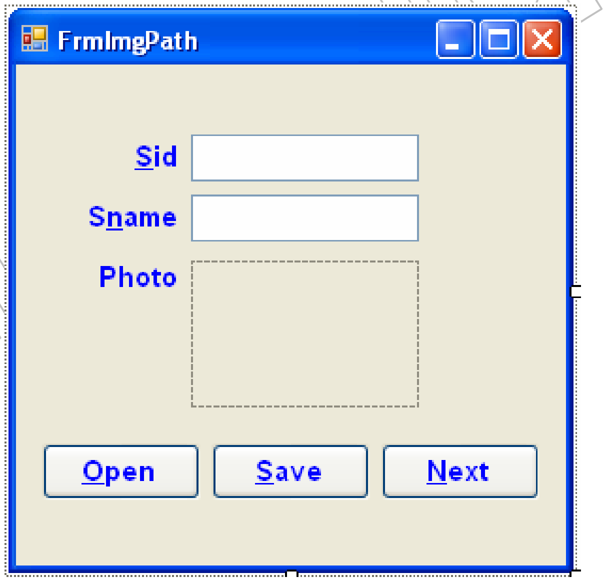
When you want to store and retrieve images from the database, you have two options, storing images in a folder on hard disk and store the path of images to database and storing image it self to the database. When you store images on hard disk and store path of the image to database, if the images are moved from their original location or deleted then the path stored in database will become invalid. Hence the recommended option for storing images is to store image it self to the database.

Example : The following example creates a table with the name student with three columns Sid, Sname and Photo where type of Sid is int and Sname and Photo is varchar. This example stores the student details in to the student table along with the path of the student photo.

1. Create a table in mydb database of SQL server with the name student as follows.

```
Create Table Student
(Sid int Primary Key,
Sname Varchar(30) Not Null,
Photo Varchar(200) Not Null)
```

2. Add a Win Form to the project and design it as follows.



3. Set the following properties to the controls on the form.

TextBox1	Name	: TxtSid
TextBox2	Name	: TxtSname
PictureBox1	Name	: PbPhoto
	SizeMode	: StretchImage
Button1	Name	: BtnOpen
	Text	: &Open

Button2	Name	: BtnSave
	Text	: &Save
Button3	Name	: Next
	Text	: &Next

4. Within the code add the reference to SQL client namespace by writing using statement and then create a connection, data adapter, command builder, dataset and an integer within the class of the form as follows.

```
static SqlConnection Cn = new SqlConnection("Server=.;Database=mydb;user
                                             id=sa;password=abc");
static SqlDataAdapter Da = new SqlDataAdapter("Select * From Student",
                                                                  Cn);
static SqlCommandBuilder Cmb = new SqlCommandBuilder(Da);
static DataSet Ds = new DataSet();
static int Pos;
```

5. Write the following code within the load event of the form to fill data in to data set.

```
private void FrmImgPath_Load(object sender, EventArgs e)
{
   try
   {
         Da.Fill(Ds, "Stu");
         Pos = 0;
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
}
```

6. Write the following code within the open button click event to display open file dialog box and load the selected image in to picture box.

```
private void BtnOpen_Click(object sender, EventArgs e)
{
   try
   {
         Ofd.InitialDirectory = @"C:\Windows\";
         Ofd.Filter = "Bitmap Files|*.Bmp|All Files|*.*";
         if (Ofd.ShowDialog() == DialogResult.OK)
         {
               PbPhoto.Load(Ofd.FileName);
         }
   }
   catch (Exception Ex)
   {
```

```
            MessageBox.Show(Ex.Message);
    }
}
```

7. Write the following code within the save button click event to save the record to table.

```
private void BtnSave_Click(object sender, EventArgs e)
{
    try
    {
            DataRow Dr = Ds.Tables["Stu"].NewRow();
            Dr[0] = TxtSid.Text;
            Dr[1] = TxtSname.Text;
            Dr[2] = Ofd.FileName;
            Ds.Tables["Stu"].Rows.Add(Dr);
            Da.Update(Ds, "Stu");
            MessageBox.Show("One Row Saved");
    }
    catch (Exception Ex)
    {
            MessageBox.Show(Ex.Message);
    }
}
```

8. Write the following code within the Next button click event to display one by one record from table in to controls on the form.

```
private void BtnNext_Click(object sender, EventArgs e)
{
    try
    {
            TxtSid.Text = Ds.Tables["Stu"].Rows[Pos][0].ToString();
            TxtSname.Text = Ds.Tables["Stu"].Rows[Pos][1].ToString();
            PbPhoto.Load(Ds.Tables["Stu"].Rows[Pos][2].ToString());
            Pos++;
            if (Pos == Ds.Tables["Stu"].Rows.Count)
                  Pos = 0;
    }
    catch (Exception Ex)
    {
            MessageBox.Show(Ex.Message);
    }
}
```

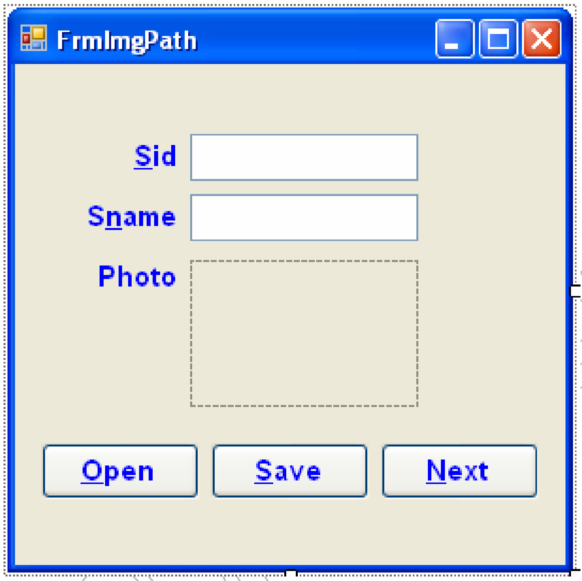
8. Run the application using the short cut F5.

Example : The following example creates a table with the name student with three columns Sid, Sname and Photo where type of Sid is int and Sname and Photo is varchar. This example stores the student details in to the student table along with Photo of the student. But not path of the photo on hard disk.

1. Create a table in mydb database of SQL server with the name students as follows.

```
Create Table Students
(Sid int Primary Key,
Sname Varchar(30) Not Null,
Photo Image Not Null)
```

2. Add a Win Form to the project and design it as follows.



3. Set the following properties to the controls on the form.

TextBox1	Name	: TxtSid
TextBox2	Name	: TxtSname
PictureBox1	Name	: PbPhoto
	SizeMode	: StretchImage
Button1	Name	: BtnOpen
	Text	: &Open
Button2	Name	: BtnSave
	Text	: &Save
Button3	Name	: Next
	Text	: &Next

4. Within the code add the reference to SQL client and System.IO namespaces by writing using statement and then create a connection, data adapter, command builder, dataset and Two integers within the class of the form as follows.

```
static SqlConnection Cn = new SqlConnection("Server=.;Database=mydb;user
                                                id=sa;password=abc");
static SqlDataAdapter Da = new SqlDataAdapter("Select * From Students",
                                                                  Cn);
static SqlCommandBuilder Cmb = new SqlCommandBuilder(Da);
static DataSet Ds = new DataSet();
static int N,Pos;
```

5. Also create a FileStrem, Binary Reader, Binary Writer and a byte array as follows.

```
static FileStream Fs;
static BinaryReader Br;
static BinaryWriter Bw;
static byte[] Photo;
```

6. Write the following code within the load event of the form to fill data in to data set.

```
private void FrmImage_Load(object sender, EventArgs e)
{
   try
   {
         Da.Fill(Ds, "Stu");
         Pos = 0;
         N=0;
   }
   catch (Exception Ex)
   {
         MessageBox.Show(Ex.Message);
   }
}
```

6. Write the following code within the open button click event to display open file dialog box, load the selected image in to picture box and convert the image to byte array.

```
private void BtnOpen_Click(object sender, EventArgs e)
{
   try
   {
         Ofd.InitialDirectory = @"C:\Windows\";
         Ofd.Filter = "Bitmap Files|*.Bmp|All Files|*.*";
         if (Ofd.ShowDialog() == DialogResult.OK)
         {
               PbPhoto.Load(Ofd.FileName);
               Fs = new FileStream(Ofd.FileName, FileMode.Open,
                                                 FileAccess.Read);
               Br = new BinaryReader(Fs);
               Photo = new byte[Fs.Length];
```

```
                Photo = Br.ReadBytes((int)Fs.Length);
                Br.Close();
                Fs.Close();
            }
        }
        catch (Exception Ex)
        {
            MessageBox.Show(Ex.Message);
        }
    }
```

7. Write the following code within the save button click event to save the record to table.

```
private void BtnSave_Click(object sender, EventArgs e)
{
    try
    {
        DataRow Dr = Ds.Tables["Stu"].NewRow();
        Dr[0] = TxtSid.Text;
        Dr[1] = TxtSname.Text;
        Dr[2] = Photo;
        Ds.Tables["Stu"].Rows.Add(Dr);
        Da.Update(Ds, "Stu");
        MessageBox.Show("One Row Saved");
    }
    catch (Exception Ex)
    {
        MessageBox.Show(Ex.Message);
    }
}
```

8. Write the following code within the Next button click event to display one by one record from table in to controls on the form by converting the image in byte format to image.

```
private void BtnNext_Click(object sender, EventArgs e)
{
    try
    {
        TxtSid.Text = Ds.Tables["Stu"].Rows[Pos][0].ToString();
        TxtSname.Text = Ds.Tables["Stu"].Rows[Pos][1].ToString();
        Photo =(byte[]) Ds.Tables["Stu"].Rows[Pos][2];
        string FPath=@"D:\Temp\Image" + N + ".Bmp";
        Fs = new FileStream(FPath, FileMode.Create, FileAccess.Write);
        Bw = new BinaryWriter(Fs);
        Bw.Write(Photo);
        Bw.Close();
        Fs.Close();
        PbPhoto.Load(FPath);
        Pos++;
        N++;
        if (Pos == Ds.Tables["Stu"].Rows.Count)
            Pos = 0;
    }
    catch (Exception Ex)
    {
         MessageBox.Show(Ex.Message);
   }
}
```

9. Run the application using the short cut F5.

Serialization And Deserialization of DataSet

Converting an object to XML is called as Serialization and creating an object from XML is called as de-serialization. Converting Data Set to XML is called as Data Set Serialization and Creating Data Set from XML is called as Data Set de-Serialization. To serialize the data set, use **WriteXml()** method of data set and to de-serialize the data set use **ReadXml()** method of the data set. This serialization and de-serialization concept can be used to work with xml files for storing and retrieving data without using a database.

Syntax :

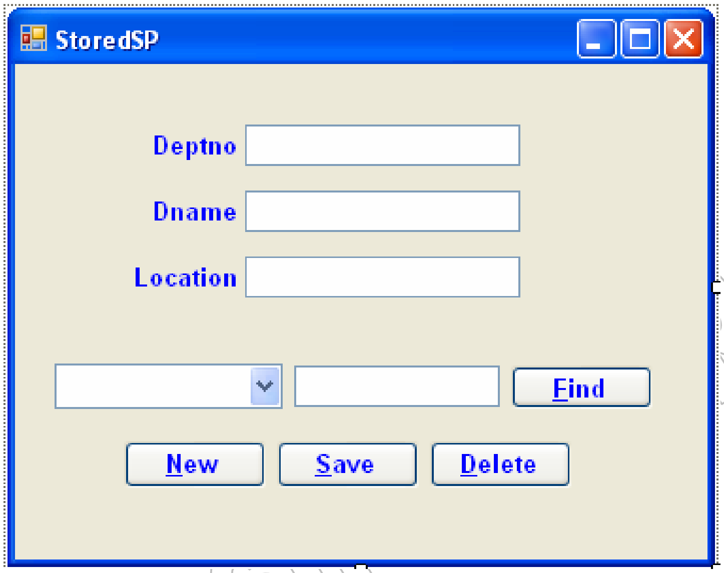
Ds.WriteXml("Xml File Path");
Ds.ReadXml("Xml File Path");

Example : The following example creates an XML file for storing departments information and provides access to it with data set serialization and de-serialization concept.

1. Create an XML file in the project with the name **Dept** by selecting **XML File** item in new item dialog box that can be displayed by right clicking on project in solution explorer and choosing **Add new item** and type the following content in it. Save the xml file to debug folder of bin folder of the project.

```
<?xml version="1.0" standalone="yes"?>
<Department>
  <Dept>
    <Deptno>10</Deptno>
    <Dname>Accouting</Dname>
    <Loc>New York</Loc>
  </Dept>
  <Dept>
    <Deptno>20</Deptno>
    <Dname>Research</Dname>
    <Loc>Dallas</Loc>
  </Dept>
  <Dept>
    <Deptno>30</Deptno>
    <Dname>Sales</Dname>
    <Loc>Boston</Loc>
  </Dept>
  <Dept>
    <Deptno>40</Deptno>
    <Dname>Operations</Dname>
    <Loc>Chicago</Loc>
  </Dept>
</Department>
```

2. Add a Win Form to the project and design it as follows.



3. Set the following properties for the controls on the form.

TextBox1	Name : TxtDno
TextBox2	Name : TxtDname
TextBox3	Name : TxtLoc
ComboBox1	Name : CmbColumn
TextBox4	Name : TxtFind
Button1	Name : BtnFind
	Text : &Find
	Enabled : false
Button2	Name : BtnNew
	Text : &New
Button3	Name : BtnSave
	Text : &Save
Button4	Name : BtnDelete
	Text : &Delete

4. Within the code add the reference to System.IO namespace and then create one dataset, datarow and a boolean variable as follows.

```
static string Path = Directory.GetCurrentDirectory() + @"\Dept.Xml";
static DataSet Ds = new DataSet();
static DataRow Dr;
static bool NewRec;
```

5. Write the following code within the load event of the form to create data set from XML file and add column names to combo box.

```
private void FrmSerialize_Load(object sender, EventArgs e)
{
      try
      {
            Ds.ReadXml(Path);
            foreach (DataColumn C in Ds.Tables["Dept"].Columns)
            {
                  CmbColumn.Items.Add(C.ToString());
            }
      }
      catch (Exception Ex)
      {
            MessageBox.Show(Ex.Message);
      }
}
```

6. Write the following code within the textchanged event of the text box TxtFind to enable or disable the find button based on no. of cahracters in the textbox.

```
private void TxtFind_TextChanged(object sender, EventArgs e)
{
      if (TxtFind.Text.Trim().Length > 0)
            BtnFind.Enabled = true;
      else
            BtnFind.Enabled = false;
}
```

7. Write the following code within the find button click event to find the row based on selected column in combo box and the value in text box.

```
private void BtnFind_Click(object sender, EventArgs e)
{
      try
      {
            string Condition = CmbColumn.Text + "='" + TxtFind.Text + "'";
            Dr = Ds.Tables["Dept"].Select(Condition)[0];
            TxtDno.Text = Dr[0].ToString();
```

```
        TxtDname.Text = Dr[1].ToString();
        TxtLoc.Text = Dr[2].ToString();
    }
    catch (Exception Ex)
    {
        MessageBox.Show("Row Not Found");
        TxtDno.Clear();
        TxtDname.Clear();
        TxtLoc.Clear();
    }
}
```

8. Write the following code within the new button click event to generate a new deptno and set newrec booloean variable to true indicating user wants to insert a new record.

```
private void BtnNew_Click(object sender, EventArgs e)
{
    try
    {
        NewRec = true;
        BtnNew.Enabled = false;
        int MaxDno=0;
        foreach (DataRow Dr in Ds.Tables["Dept"].Rows)
        {
            if (int.Parse(Dr[0].ToString()) > MaxDno)
                MaxDno = int.Parse(Dr[0].ToString());
        }
        TxtDno.Text = (MaxDno + 10).ToString();
        TxtDname.Clear();
        TxtLoc.Clear();
        TxtDname.Focus();
     }
     catch (Exception Ex)
     {
        MessageBox.Show(Ex.Message);
     }
}
```

9. Write the following code within the save button click event to insert or update the row based on the value in newrec boolean variable.

```
    private void BtnSave_Click(object sender, EventArgs e)
    {
        try
        {
          if (NewRec == true)
          {
              Dr = Ds.Tables["Dept"].NewRow();
              Dr[0] = TxtDno.Text;
              Dr[1] = TxtDname.Text;
              Dr[2] = TxtLoc.Text;
              Ds.Tables["Dept"].Rows.Add(Dr);
              NewRec = false;
              BtnNew.Enabled = true;
          }
          else
          {
              Dr[0] = TxtDno.Text;
              Dr[1] = TxtDname.Text;
              Dr[2] = TxtLoc.Text;
          }
          Ds.WriteXml(Path);
          MessageBox.Show("One Row Saved");
      }
      catch (Exception Ex)
      {
          MessageBox.Show(Ex.Message);
      }
  }
```

10. Write the following code within the delete button click event to delete the row.

```
private void BtnDelete_Click(object sender, EventArgs e)
  {
      try
      {
          DialogResult Res;
          Res = MessageBox.Show("Do You Really Want To Delete?",
            "Confirmation To Delete", MessageBoxButtons.YesNo,
                                    MessageBoxIcon.Question);
          if (Res == DialogResult.Yes)
          {
              Dr.Delete();
              Ds.WriteXml(Path);
              MessageBox.Show("One Row Deleted");
              TxtDno.Clear();
              TxtDname.Clear();
              TxtLoc.Clear();
          }
      }
      catch (Exception Ex)
      {
          MessageBox.Show(Ex.Message);
      }
  }
```

11. Run the application using the short cut F5.

Typed DataSet

A data set with strongly typed properties and methods is called as typed data set. In case of an untyped data set, properties and methods are not strongly typed and hence it has the following drawbacks.

1. The data types of columns in a data row in untyped dataset are not same as the type of the column in table and hence assigning a different type of value to a column will not cause error during compilation and will cause error at run time.

2. It doesn’t contain any constraint information and hence providing any data that violate the constraint will not cause any error during compilation and will cause error at runtime.

3. Code to access the data available in untyped data set is complex.

The drawbacks of untyped data set are overcome by typed data set as follows.

1. The data types of columns in a data row in typed dataset are same as the type of the column in table and hence assigning a different type of value to a column will cause error during compilation it self.

2. All the constraints available in table are automatcally available in typed data set and hence assigning a value that violates the constraint will cause error during compilation it self.

3. Code to access the data available in typed data set will be easy when compared to untyped data set.

Creating Untyped DataSet

To create a typed data set, follow the following steps.

1. Add a dataset to the project by choosing **Dataset** item in new item dialog box that is displayed when you right click on project in solution explorer and choose **add new item**.

2. Dataset will be added to project and will be opened. Now open server explorer and create a connection to database that contains the table for which you want to create the typed dataset.

3. After creating the connection, expand the connection and then tables in server explorer and then drag and drop the table for which you want to create the typed dataset from server explorer on to Dataset, save the Dataset and close it, which is now a typed data set.

An untyped data set can not be used in the following situations.

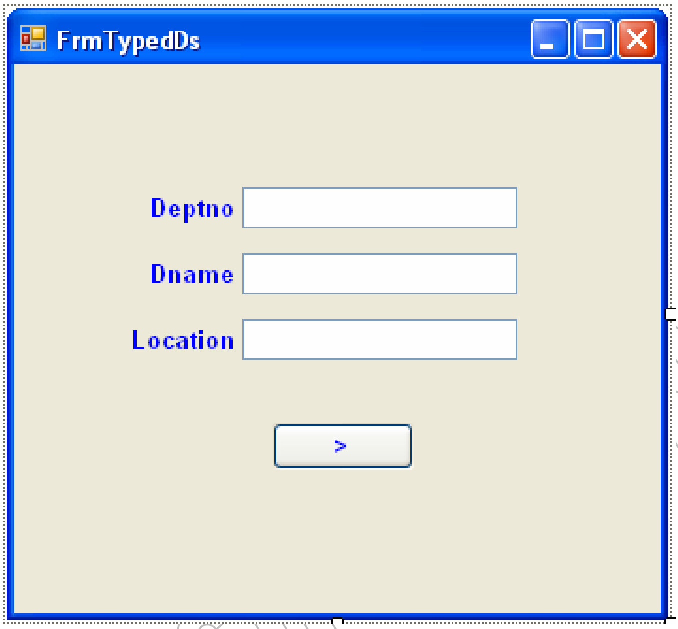
1. When you want to use the data set to hold the data set returned by a function.

2. When There is a chance for changing the structure of the table after creating the data set.

Example : The following example creates a typed dataset for Dept table available in mydb database of SQL server and provides access to it.

1. Create a typed data set for the table Dept in Mydb database of SQL Server with the name **DsDept** by following the above procedure specified for creating typed data set.

2. Add a winform to the project and design it as follows.



3. Set the following properties for the controls on the form.

TextBox1	Name : TxtDno
TextBox2	Name : TxtDname
TextBox3	Name : TxtLoc
Button1	Name : BtnNext
	Text : >

4. Within the code write using statement to add the reference to SQLClient namespace and then a create a connection, data adapter and a dataset of type DsDept and an integer as follows.

```
static SqlConnection Cn = new SqlConnection("Server=.;Database=mydb;user
                                         id=sa;password=abc");
static SqlDataAdapter Da = new SqlDataAdapter("Select * From Dept", Cn);
static DsDept Ds = new DsDept();
static int Pos;
```

5. Write the following code within the form load event.

```
private void FrmTypedDs_Load(object sender, EventArgs e)
{
      try
      {
          Da.Fill(Ds, "Dept");
          Pos = 0;
      }
      catch (Exception Ex)
      {
          MessageBox.Show(Ex.Message);
      }
}
```

6. Write the followin code within the next button click event to display one by one record.

```
private void BtnNext_Click(object sender, EventArgs e)
{
      try
      {
          TxtDno.Text = Ds.DEPT[Pos].DEPTNO.ToString();
          TxtDname.Text = Ds.DEPT[Pos].DNAME;
          TxtLoc.Text = Ds.DEPT[Pos].LOC;
          Pos++;
          if (Pos == Ds.DEPT.Rows.Count)
              Pos = 0;
      }
      catch (Exception Ex)
      {
          MessageBox.Show(Ex.Message);
      }
}
```

7. Run the application using the short cut F5.

Connection Pooling

Connecting to a database server typically consists of several time-consuming steps. A physical channel such as a socket or a named pipe must be established, the initial handshake with the server must occur, the connection string information must be parsed, the connection must be authenticated by the server, checks must be run for enlisting in the current transaction, and so on.

In practice, most applications use only one or a few different configurations for connections. This means that during application execution, many identical connections will be repeatedly opened and closed. To minimize the cost of opening connections, ADO.NET uses an optimization technique called *connection pooling*.

Connection pooling reduces the number of times that new connections need to be opened. The *pooler* maintains ownership of the physical connection. It manages connections by keeping alive a set of active connections for each given connection configuration. Whenever a user calls **Open** on a connection, the pooler looks to see if there is an available connection in the pool. If a pooled connection is available, it returns it to the caller instead of opening a new connection. When the application calls **Close** on the connection, the pooler returns it to the pooled set of active connections instead of actually closing it. Once the connection is returned to the pool, it is ready to be reused on the next **Open** call.

Only connections with the same configuration can be pooled. ADO.NET keeps several pools concurrently, one for each configuration. Connections are separated into pools by connection string, as well as by Windows identity when integrated security is used. Connections are also pooled based on whether they are enlisted in a transaction.

Pooling connections can significantly enhance the performance and scalability of your application. Connection pooling is enabled by default in ADO.NET. Unless you explicitly disable it, the pooler will optimize the connections as they are opened and closed in your application. You can also supply several connection string modifiers to control connection pooling behavior.

Pool Creation and Assignment

When a connection is first opened, a connection pool is created based on an exact matching algorithm that associates the pool with the connection string in the connection. Each connection pool is associated with a distinct connection string. When a new connection is opened, if the connection string is not an exact match to an existing pool, a new pool is created. Connections are pooled per process, per application domain, per connection string and when using integrated security, per Windows identity. Connection strings must also be an exact match; keywords supplied in a different order for the same connection will be pooled separately.

Adding Connections

A connection pool is created for each unique connection string. When a pool is created, multiple connection objects are created and added to the pool so that the minimum pool size requirement is satisfied. Connections are added to the pool as needed, up to the maximum pool size specified (100 is the default). Connections are released back into the pool when they are closed or disposed.

When a SqlConnection object is requested, it is obtained from the pool if a usable connection is available. To be usable, a connection must be unused, have a matching transaction context or be unassociated with any transaction context, and have a valid link to the server.

The connection pooler satisfies requests for connections by reallocating connections as they are released back into the pool. If the maximum pool size has been reached and no usable connection is available, the request is queued. The pooler then tries to reclaim any connections until the time-out is reached (the default is 15 seconds). If the pooler cannot satisfy the request before the connection times out, an exception is thrown.

We strongly recommend that you always close the connection when you are finished using it in order for the connection to be returned to the pool. You can do this using either the Close or Dispose methods of the Connection object

Removing Connections

The connection pooler removes a connection from the pool after it has been idle for an extended period of time, or if the pooler detects that the connection with the server has been severed. Note that a severed connection can be detected only after attempting to communicate with the server. If a connection is found that is no longer connected to the server, it is marked as invalid. Invalid connections are removed from the connection pool only when they are closed or reclaimed.

Within the connection string specified for the connection it self you can specify the options related to connection pooling and are as follows.

Keyword	Default	Description
Connect Timeout Or Connection Timeout	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
Connection Lifetime	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by Connection Lifetime. This is useful in clustered configurations to force load balancing between a running server and a server just brought online. A value of zero (0) causes pooled connections to have the maximum connection timeout.
Max Pool Size	100	The maximum number of connections allowed in the pool.
Min Pool Size	0	The minimum number of connections allowed in the pool.
Pooling	True	Indicates whether the connection pooling is enabled
Load Balance Timeout	0	The minimum time, in seconds, for the connection to live in the connection pool before being destroyed.