Unit - 4
ADO.NET & Database

ADO.NET ARCHITECTURE

The .NET Framework includes its own data access technology i.e. ADO.NET. ADO.NET is the latest implementation of Microsoft's Universal Data Access strategy. ADO.NET consists of managed classes that allows .NET applications to connect to data sources such as Microsoft SQL Server, Microsoft Access, Oracle, XML, etc., execute commands and manage disconnected data..t

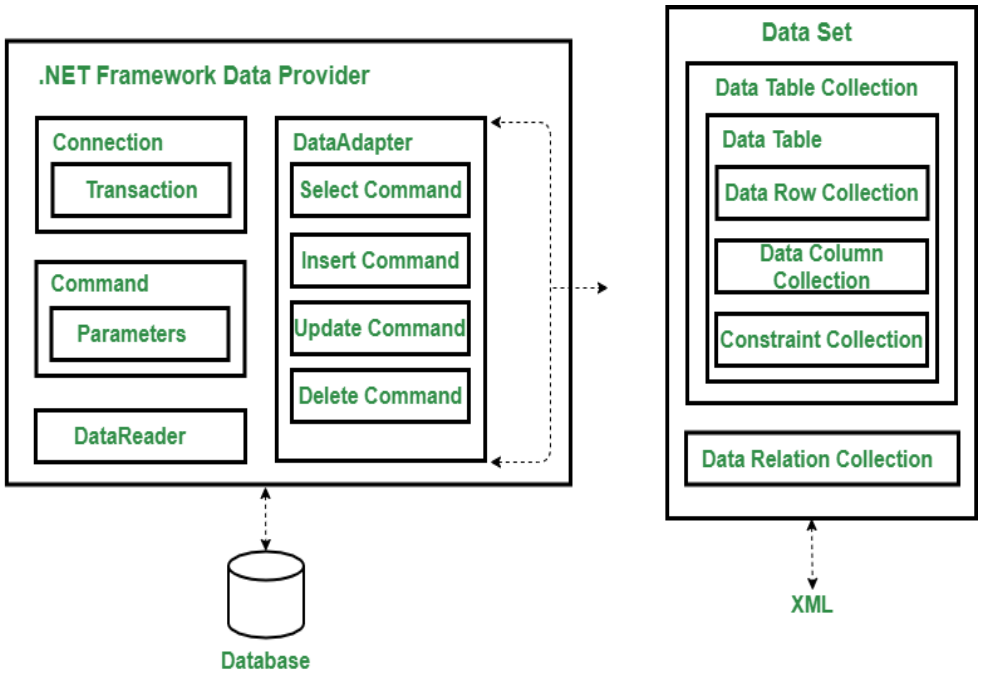
Microsoft ADO.NET is the latest improvement after ADO. Firstly, ADO.NET was introduced in the 10th version of the .NET framework, which helps in providing an extensive array of various features to handle data in different modes, such as connected mode and disconnected mode. In connected mode, we are dealing with live data and in disconnected mode, data is provided from the data store.

ADO.NET was primarily developed to address two ways to work with data that we are getting from data sources. The two ways are as follows :

1. The first is to do with the user's need to access data once and to iterate through a collection of data in a single instance.
2. The second way to work with data is disconnected architecture mode, in which we have to grab a collection of data and we use this data separately from the data store itself.

The ADO.NET architecture comprises six important components. They are as follows:

1. Connection
2. Command
3. DataReader
4. DataAdapter
5. DataSet
6. DataView



Connection: The connection object is the first important component of your application. It is required to connect to a backend database that may be SQL Server, Oracle, MySQL, etc. You must have two things to create a connection object. Your database Machine name or IP address or someplace where it is stored is where it is. The second thing is security credentials, such as whether it's a Windows authentication or username and password-based authentication. The connection is created using the connection object and a backend data source must be connected to using the connection.

Command: The second important component is the command object. When we discuss databases such as SQL Server, Oracle, MySQL, then speak SQL, it is the command object that we use to create SQL queries. After you create your SQL queries, you can execute them over the connection using the command object. You can go either the DataSet or the DataReader way with DTS. In general, you should choose which method you require based on the situation. Note: You can go either the DataSet or the DataReader way with DTS.

DataReader: We can only read the records in the forward mode with DataReader. Here, you should familiarize yourself with three things: read-only, connected, and forward modes.

DataSet: A disconnected recordset can be browsed in both directions, and it is also possible to insert, update, or delete data sets. The DataAdapter fills a DataSet using data.

DataAdapter: The DataAdapter performs an operation on the data from the command object and then writes the data set to the dataset.

DataView Class: A DataView enables you to modify the appearance of the data stored in a DataTable, a data-binding skill that is frequently employed in data-view applications. You may alter the sort order of data in a table or filter it based on row state or on a filter expression using a DataView.

Features :

1. When viewed as text-based formats, XML documents are obviously negotiable. ADO.NET exchanges data using XML, regardless of its complexity, and for internal purposes.
2. We can model our application in separate layers, which is what ADO.NET is built around.
3. A programming style in which words are used to construct assertions or evaluate expressions is called word-based programming. The following code fragment illustrates how to select the “Ranks” column from “Scaler” in the “Student” table using word-based programming:
   DataSet.Student(“Scaler”).Ranks;
4. The data architecture is simple to scale as it involves only disconnected data on the server. Because everything is handled on the client-side, performance is improved.
5. The growing number of clients requiring degraded performance as it uses disconnected data access is accommodated by the application’s use of lock connections that last longer. In addition, the application can afford to make the programmers conserve resources and allow users to access data simultaneously.

Advantages of ADO.Net Architecture

- A Data component in Visual Studio environment works to establish data access in a variety of ways to make it simple and safe to develop applications. ADO.NET data components in Visual Studio environment encapsulate data access functionality in various ways that make it simpler and safer to develop applications.
- As a result of its Disconnected Architecture, ADO.Net provides remarkable performance advantages by eliminating all data connection dependencies. DataSet functions in ADO.Net are completely disconnected, making it possible to plug an unlimited number of supported data sources into code without any difficulty in the future.
- Extensible Markup Language (XML) is the standard format for exchanging data across a network. Any component that can read XML can process data, so XML is a perfectly viable format for transmitting datasets. Although various ADO.NET data types like the DataSet are so intertwined with XML that they cannot exist or function without it, XML is not required.
- The advantage of using the Advantage ADO.NET Data Provider, is that it enables you to directly modify data using SQL. In order to provide interaction with SQL Server, the SQL Server Data Provider that is included with ADO.NET is highly optimized. It is utilized in conjunction with the tabular data stream (TDS) format that is employed by SQL Server. The advantage of using the Advantage ADO.NET Data Provider is that it is expertly crafted to facilitate data modification.
- The ADO.NET object model is another big advantage. ADO.NET is a complicated object model that is built using class inheritance and interface implementation. Once you look for things you need in this scope, you will realize that the logical base classes and features derived from the entire system are simple to use.

Grid View control :

The GridView control is used to display the values of a data source in a table. Each column represents a field, while each row represents a record. The GridView control supports the following features:

- Binding to data source controls, such as SqlDataSource.
- Built-in sort capabilities.
- Built-in update and delete capabilities.
- Built-in paging capabilities.
- Built-in row selection capabilities.
- Programmatic access to the GridView object model to dynamically set properties, handle events, and so on.
- Multiple key fields.
- Multiple data fields for the hyperlink columns.
- Customizable appearance through themes and styles.

Properties :

AccessKey	Gets or sets the access key that allows you to quickly navigate to the Web server control.
Adapte	Gets the browser-specific adapter for the control.
AllowCustomPaging	Gets or sets a value that indicates whether custom paging is enabled.

AllowPaging	Gets or sets a value indicating whether the paging feature is enabled.
AllowSorting	Gets or sets a value indicating whether the sorting feature is enabled.
AlternatingRowStyle	Gets a reference to the TableItemStyle object that enables you to set the appearance of alternating data rows in a GridView control.
AppRelativeTemplateSourceDirectory	Gets or sets the application-relative virtual directory of the Page or UserControl object that contains this control. (Inherited from Control)
Attributes	Gets the collection of arbitrary attributes (for rendering only) that do not correspond to properties on the control. (Inherited from WebControl)
AutoGenerateColumns	Gets or sets a value indicating whether bound fields are automatically created for each field in the data source.

AutoGenerateDeleteButton	Gets or sets a value indicating whether a CommandField field column with a Delete button for each data row is automatically added to a GridView control.
AutoGenerateEditButton	Gets or sets a value indicating whether a CommandField field column with an Edit button for each data row is automatically added to a GridView control.
AutoGenerateSelectButton	Gets or sets a value indicating whether a CommandField field column with a Select button for each data row is automatically added to a GridView control.
BackColor	Gets or sets the background color of the Web server control. (Inherited from WebControl)
BackImageUrl	Gets or sets the URL to an image to display in the background of a GridView control.
BindingContainer	Gets the control that contains this control's data binding. (Inherited from Control)

BorderColor	Gets or sets the border color of the Web control. (Inherited from WebControl)
BorderStyle	Gets or sets the border style of the Web server control. (Inherited from WebControl)
BorderWidth	Gets or sets the border width of the Web server control. (Inherited from WebControl)
BottomPagerRow	Gets a GridViewRow object that represents the bottom pager row in a GridView control.
Caption	Gets or sets the text to render in an HTML caption element in a GridView control. This property is provided to make the control more accessible to users of assistive technology devices.
CaptionAlign	Gets or sets the horizontal or vertical position of the HTML caption element in a GridView control. This property is provided to make the control more accessible to users of assistive technology devices.

CellPadding	Gets or sets the amount of space between the contents of a cell and the cell's border.
CellSpacing	Gets or sets the amount of space between cells.
ChildControlsCreated	Gets a value that indicates whether the server control's child controls have been created. (Inherited from Control)
ClientID	Gets the control ID for HTML markup that is generated by ASP.NET. (Inherited from Control)
ClientIDMode	Gets or sets the algorithm that is used to generate the value of the ClientID property. (Inherited from Control)
ClientIDRowSuffix	Gets or sets the names of the data fields whose values are appended to the ClientID property value to uniquely identify each instance of a data-bound control.

ClientIDRowSuffixDataKeys	Gets the data values that are used to uniquely identify each instance of a data-bound control when ASP.NET generates the ClientID value.
ClientIDSeparator	Gets a character value representing the separator character used in the ClientID property. (Inherited from Control)
Columns	Gets a collection of DataControlField objects that represent the column fields in a GridView control.
ColumnsGenerator	Gets or sets the control that will automatically generate the columns for a GridView control that uses ASP.NET Dynamic Data features.
Context	Gets the HttpContext object associated with the server control for the current Web request. (Inherited from Control)
Controls	Gets a collection of the child controls within the composite data-bound control. (Inherited from CompositeDataBoundControl)

ControlStyle	Gets the style of the Web server control. This property is used primarily by control developers. (Inherited from WebControl)
ControlStyleCreated	Gets a value indicating whether a Style object has been created for the ControlStyle property. This property is primarily used by control developers. (Inherited from WebControl)
CssClass	Gets or sets the Cascading Style Sheet (CSS) class rendered by the Web server control on the client. (Inherited from WebControl)
DataItemContainer	Gets a reference to the naming container if the naming container implements IDataItemContainer. (Inherited from Control)
DataKeyNames	Gets or sets an array that contains the names of the primary key fields for the items displayed in a GridView control.

DataKeys	Gets a collection of DataKey objects that represent the data key value of each row in a GridView control.
DataKeysContainer	Gets a reference to the naming container if the naming container implements IDataKeysControl. (Inherited from Control)
DataMember	Gets or sets the name of the list of data that the data-bound control binds to, in cases where the data source contains more than one distinct list of data items. (Inherited from DataBoundControl)
DataSource	Gets or sets the object from which the data-bound control retrieves its list of data items. (Inherited from BaseDataBoundControl)
DataSourceID	Gets or sets the ID of the control from which the data-bound control retrieves its list of data items. (Inherited from DataBoundControl)

DataSourceObject	Gets an object that implements the IDataSource interface, which provides access to the object's data content. (Inherited from DataBoundControl)
DeleteMethod	Gets or sets the name of the method to call in order to delete data.
DesignMode	Gets a value indicating whether a control is being used on a design surface. (Inherited from Control)
EditIndex	Gets or sets the index of the row to edit.
EditRowStyle	Gets a reference to the TableItemStyle object that enables you to set the appearance of the row selected for editing in a GridView control.
EmptyDataRowStyle	Gets a reference to the TableItemStyle object that enables you to set the appearance of the empty data row rendered when a GridView control is bound to a data source that does not contain any records.

EmptyDataTemplate	Gets or sets the user-defined content for the empty data row rendered when a GridView control is bound to a data source that does not contain any records.
EmptyDataText	Gets or sets the text to display in the empty data row rendered when a GridView control is bound to a data source that does not contain any records.
Enabled	Gets or sets a value indicating whether the Web server control is enabled. (Inherited from WebControl)
EnableModelValidation	Gets or sets a value that indicates whether a validator control will handle exceptions that occur during insert or update operations.
EnablePersistedSelection	Gets or sets a value that indicates whether the selection of a row is based on index or on data-key values.
EnableSortingAndPagingCallbacks	Gets or sets a value indicating whether client-side callbacks are used for sorting and paging operations.
EnableViewState	Gets or sets a value indicating whether the server control persists its view state, and the view state of any child controls it contains, to the requesting client.

Events	Gets a list of event handler delegates for the control. This property is read-only. (Inherited from Control)
Font	Gets the font properties associated with the Web server control. (Inherited from WebControl)
FooterRow	Gets a GridViewRow object that represents the footer row in a GridView control.
FooterStyle	Gets a reference to the TableItemStyle object that enables you to set the appearance of the footer row in a GridView control.
ForeColor	Gets or sets the foreground color (typically the color of the text) of the Web server control. (Inherited from WebControl)
GridLines	Gets or sets the gridline style for a GridView control.

HasAttributes	Gets a value indicating whether the control has attributes set. (Inherited from WebControl)
HasChildViewState	Gets a value indicating whether the current server control's child controls have any saved view-state settings. (Inherited from Control)
HeaderRow	Gets a GridViewRow object that represents the header row in a GridView control.
HeaderStyle	Gets a reference to the TableItemStyle object that enables you to set the appearance of the header row in a GridView control.
Height	Gets or sets the height of the Web server control. (Inherited from WebControl)
HorizontalAlign	Gets or sets the horizontal alignment of a GridView control on the page.

ID	Gets or sets the programmatic identifier assigned to the server control. (Inherited from Control)
IdSeparator	Gets the character used to separate control identifiers. (Inherited from Control)
InsertMethod	Gets or sets the name of the method to call in order to insert data. (Inherited from CompositeDataBoundControl)
IsBoundUsingDataSourceID	Gets a value indicating whether the DataSourceID property is set. (Inherited from BaseDataBoundControl)
IsChildontrolStateCleared	Gets a value indicating whether controls contained within this control have control state. (Inherited from Control)

IsDataBindingAutomatic	Gets a value that indicates whether data binding is automatic. (Inherited from BaseDataBoundControl)
IsTrackingViewState	Gets a value that indicates whether the server control is saving changes to its view state. (Inherited from Control)
IsUsingModelBinders	Gets a value that indicates whether model binding is in use. (Inherited from CompositeDataBoundControl)
IsViewStateEnabled	Gets a value indicating whether view state is enabled for this control. (Inherited from Control)
ItemType	Gets or sets the name of the data item type for strongly typed data binding. (Inherited from DataBoundControl)

Page	Gets a reference to the Page instance that contains the server control. (Inherited from Control)
PageCount	Gets the number of pages required to display the records of the data source in a GridView control.
PageIndex	Gets or sets the index of the currently displayed page.
PagerSettings	Gets a reference to the PagerSettings object that enables you to set the properties of the pager buttons in a GridView control.
PagerStyle	Gets a reference to the TableItemStyle object that enables you to set the appearance of the pager row in a GridView control.
PagerTemplate	Gets or sets the custom content for the pager row in a GridView control.
PageSize	Gets or sets the number of records to display on a page in a GridView control.

Parent	Gets a reference to the server control's parent control in the page control hierarchy. (Inherited from Control)
RenderingCompatibility	Gets a value that specifies the ASP.NET version that rendered HTML will be compatible with. (Inherited from Control)
RequiresDataBinding	Gets or sets a value indicating whether the DataBind() method should be called. (Inherited from BaseDataBoundControl)
RowHeaderColumn	Gets or sets the name of the column to use as the column header for the GridView control. This property is provided to make the control more accessible to users of assistive technology devices.
Rows	Gets a collection of GridViewRow objects that represent the data rows in a GridView control.
RowStyle	Gets a reference to the TableItemStyle object that enables you to set the appearance of the data rows in a GridView control.

SelectArguments	Gets a DataSourceSelectArguments object that the data-bound control uses when retrieving data from a data source control. (Inherited from DataBoundControl)
SelectedDataKey	Gets the DataKey object that contains the data key value for the selected row in a GridView control.
SelectedIndex	Gets or sets the index of the selected row in a GridView control.
SelectedPersistedDataKey	Gets or sets the data-key value for the persisted selected item in a GridView control.
SelectedRow	Gets a reference to a GridViewRow object that represents the selected row in the control.
SelectedRowStyle	Gets a reference to the TableItemStyle object that enables you to set the appearance of the selected row in a GridView control.
SelectedValue	Gets the data key value of the selected row in a GridView control.

SelectMethod	The name of the method to call in order to read data. (Inherited from DataBoundControl)
ShowFooter	Gets or sets a value indicating whether the footer row is displayed in a GridView control.
ShowHeader	Gets or sets a value indicating whether the header row is displayed in a GridView control.
ShowHeaderWhenEmpty	Gets or sets a value that indicates whether the heading of a column in the GridView control is visible when the column has no data.
Site	Gets information about the container that hosts the current control when rendered on a design surface. (Inherited from Control)
SkinID	Gets or sets the skin to apply to the control. (Inherited from WebControl)
SortDirection	Gets the sort direction of the column being sorted.

SortedAscendingCellStyle	Gets or sets the CSS style for a GridView column when the column is sorted in ascending order.
SortedAscendingHeaderStyle	Gets or sets the CSS style to apply to a GridView column heading when the column is sorted in ascending order.
SortedDescendingCellStyle	Gets or sets the style of a GridView column when the column is sorted in descending order.
SortedDescendingHeaderStyle	Gets or sets the style to apply to a GridView column heading when the column is sorted in descending order.
SortExpression	Gets the sort expression associated with the column or columns being sorted.
TabIndex	Gets or sets the tab index of the Web server control. (Inherited from WebControl)
TagKey	Gets the HtmlTextWriterTag value for the GridView control.

TagName	Gets the name of the control tag. This property is used primarily by control developers. (Inherited from WebControl)
TemplateSourceDirectory	Gets the virtual directory of the Page or UserControl that contains the current server control. (Inherited from Control)
ToolTip	Gets or sets the text displayed when the mouse pointer hovers over the Web server control. (Inherited from WebControl)
TopPagerRow	Gets a GridViewRow object that represents the top pager row in a GridView control.
UniqueID	Gets the unique, hierarchically qualified identifier for the server control. (Inherited from Control)
UpdateMethod	Gets or sets the name of the method to call in order to update data.

UseAccessibleHeader	Gets or sets a value indicating whether a GridView control renders its header in an accessible format. This property is provided to make the control more accessible to users of assistive technology devices.
ValidateRequestMode	Gets or sets a value that indicates whether the control checks client input from the browser for potentially dangerous values. (Inherited from Control)
ViewState	Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page.
ViewStateIgnoresCase	Gets a value that indicates whether the StateBag object is case-insensitive. (Inherited from Control)
ViewStateMode	Gets or sets the view-state mode of this control. (Inherited from Control)
VirtualItemCount	Gets or sets the virtual number of items in the data source that the GridView control is bound to when custom paging is used.

Visible	Gets or sets a value that indicates whether a server control is rendered as UI on the page. (Inherited from Control)
Width	Gets or sets the width of the Web server control. (Inherited from WebControl)

Example:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Web.UI.WebControls;
public partial class GridViewDemo : System.Web.UI.Page
{
    SqlConnection conn;
    SqlDataAdapter adapter;
    DataSet ds;
    string str="EmpID";
    protected void Page_Load(object sender, EventArgs e)
    {
        if(!IsPostBack)
        {
            FillGridView(str);
        }
    }
    protected void FillGridView(string str)
    {
        string cs = ConfigurationManager.ConnectionStrings["conString"].ConnectionString;
        try
        {
            conn = new SqlConnection(cs);
            adapter = new SqlDataAdapter("select * from tblEmps order by "+str, conn);
            ds = new DataSet();
            adapter.Fill(ds);
```

```
            GridView1.DataSource = ds;
            GridView1.DataBind();
        }
        catch (Exception ex)
        {
            Label1.Text = "ERROR :: " + ex.Message;
        }
        finally
        {
            ds.Dispose();
            conn.Dispose();
        }
    }
    protected void GridView1_Sorting(object sender, GridViewSortEventArgs e)
    {
        str = e.SortExpression;
        FillGridView(str);
    }
    protected void GridView1_PageIndexChanging(object sender, GridViewPageEventArgs e)
    {
        GridView1.PageIndex= e.NewPageIndex;
        FillGridView(str);
    }
}
```

Repeater Control :

Repeater Control is a control which is used to display the repeated list of items

Repeater Control is used to display a repeated list of items that are bound to the control and it's the same as gridview and datagridview. Repeater control is lightweight and faster to display data when compared with gridview and datagrid. By using this control we can display data in custom format but it’s not possible in gridview or datagridview and it doesn’t support for paging and sorting.

The Repeater control works by looping through the records in your data source and then repeating the rendering of it’s templates called item template. Repeater control contains different types of template fields those are

ItemTemplate: ItemTemplate defines how the each item is rendered from data source collection.

AlternatingItemTemplate: AlternatingItemTemplates is used to change the background color and styles of AlternatingItems in DataSource collection

HeaderTemplate: HeaderTemplate is used to display Header text for DataSource collection and apply different styles for header text.

FooterTemplate: FooterTemplate is used to display footer element for DataSource collection

SeparatorTemplate: SeparatorTemplate will determine the separator element which separates each Item in Item collection. Usually, SeparateTemplate will be
 html element or <hr> html element.

To implement repeater control sample first design table in your database as shown below

Column Name	**Data Type**	**Allow Nulls**
Id	int(set identity property=true)	No
UserName	varchar(50)	Yes
Subject	nvarchar(MAX)	Yes
Comment	nvarchar(MAX)	Yes
PostedDate	datetime	Yes

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Repeator Control Example</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<table>
<tr>
<td>Enter Name: </td>
<td><asp:TextBox ID="txtName" runat="server"/></td>
</tr>
<tr>
<td>Enter Subject: </td>
<td><asp:TextBox ID="txtSubject" runat="server"/></td>
</tr>
<tr>
<td valign="top">Enter Comments:</td>
<td><asp:TextBox ID="txtComment" runat="server" Rows="5" Columns="20"
TextMode="MultiLine"/></td>
</tr>
<tr>
<td></td>
<td><asp:Button ID="btnSubmit" runat="server" Text="Submit" onclick="btnSubmit_Click"
/></td>
</tr>
</table>
</div>
<div>
<asp:Repeater ID="RepDetails" runat="server">
<HeaderTemplate>
<table style=" border:1px solid #df5015; width:500px" cellpadding="0">
<tr style="background-color:#df5015; color:White">
<td colspan="2">
<b>Comments</b>
</td>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr style="background-color:#EBEFF0">
<td>
<table style="background-color:#EBEFF0;border-top:1px dotted #df5015; width:500px" >
<tr>
<td>
```

```
Subject:
<asp:Label ID="lblSubject" runat="server" Text='<%#Eval("Subject") %>' Font-Bold="true"/>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
<asp:Label ID="lblComment" runat="server" Text='<%#Eval("Comment") %>'/>
</td>
</tr>
<tr>
<td>
<table style="background-color:#EBEFF0;border-top:1px dotted #df5015;border-bottom:1px
solid #df5015; width:500px" >
<tr>
<td>Post By: <asp:Label ID="lblUser" runat="server" Font-Bold="true"
Text='<%#Eval("UserName") %>'/></td>
<td>Created Date:<asp:Label ID="lblDate" runat="server" Font-Bold="true"
Text='<%#Eval("PostedDate") %>'/></td>
</tr>
</table>
</td>
</tr>
<tr>
<td colspan="2"> </td>
</tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</div>
</form>
</body>
</html>
using System;
using System.Data;
using System.Data.SqlClient;
```

After add namespace write the following code

```
private SqlConnection con = new SqlConnection("Data Source=SureshDasari;Initial
Catalog=MySampleDB;Integrated Security=true");
protected void Page_Load(object sender, EventArgs e)
{
if(!IsPostBack)
{
BindRepeaterData();
}
}
// This button click event is used to insert comment details and bind data to repeater control
protected void btnSubmit_Click(object sender, EventArgs e)
{
con.Open();
SqlCommand cmd = new SqlCommand("insert into Repeater_Table
(UserName,Subject,Comment,PostedDate)
values(@userName,@subject,@comment,@postedDate)", con);
cmd.Parameters.AddWithValue("@userName", txtName.Text);
cmd.Parameters.AddWithValue("@subject", txtSubject.Text);
cmd.Parameters.AddWithValue("@comment", txtComment.Text);
cmd.Parameters.AddWithValue("@postedDate", DateTime.Now);
cmd.ExecuteNonQuery();
con.Close();
txtName.Text = string.Empty;
txtSubject.Text = string.Empty;
txtComment.Text = string.Empty;
BindRepeaterData();
}
//Bind Data to Repeater Control
protected void BindRepeaterData()
{
con.Open();
SqlCommand cmd = new SqlCommand("select * from Repeater_Table Order By PostedDate
desc", con);
DataSet ds = new DataSet();
SqlDataAdapter da = new SqlDataAdapter(cmd);
da.Fill(ds);
RepDetails.DataSource = ds;
RepDetails.DataBind();
con.Close();
}
```

DataList Control:

The DataList control supports different Repeat Layouts

Table – This is the default layout. The Table layout renders the DataList as an HTML Table and its items are rendered inside the Table Cell. The number of Cells in each row can be set using the RepeatColumns property.

Flow – The Flow layout does not render the DataList as any HTML element. All the DataList items are rendered directly just as how the Repeater control repeats its items.

OrderedList and UnorderedList – The OrderedList and UnorderedList layouts renders the DataList as HTML Ordered List and Unordered List. But in .Net 4.0 and above these are no longer supported.

DataList control and Templates

The DataList control makes use of the following templates.

HeaderTemplate – The content of this template will not be repeated and will be placed in the top most position i.e. head section of the DataList control.

ItemTemplate – The content of this template will be repeated for each record present in its DataSource.

AlternatingItemTemplate – AlternatingItemTemplate is used for adding alternate items. It is used along with ItemTemplate, generally for displaying a different design for alternating items.

SeparatorTemplate - This template is used to add a separator between two items of the DataList control.

FooterTemplate – The content of this template will not be repeated and will be placed in the bottom most position i.e. footer section of the DataList control.

Example:

```
<asp:DataList ID="dlCustomers" runat="server" DataSourceID="SqlDataSource1"
RepeatColumns = "3" CellSpacing = "3" RepeatLayout = "Table">
   <ItemTemplate>
      <table class = "table">
         <tr>
            <th colspan="2">
               <b><%# Eval("ContactName") %></b>
            </th>
         </tr>
         <tr>
            <td colspan="2">
               <%# Eval("City") %>,
               <%# Eval("PostalCode") %><br />
               <%# Eval("Country")%>
            </td>
         </tr>
         <tr>
            <td>
                Phone:
            </td>
            <td>
               <%# Eval("Phone")%>
            </td>
         </tr>
         <tr>
            <td>
                Fax:
            </td>
            <td>
               <%# Eval("Fax")%>
            </td>
         </tr>
      </table>
   </ItemTemplate>
</asp:DataList>
```

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%$
ConnectionStrings:constr %>"
   SelectCommand="SELECT TOP 9 * FROM Customers"></asp:SqlDataSource>
```

How to display image using datalist control :

https://www.youtube.com/watch?v=IUF_6hwuePg

Details View:

DetailView is one of the important controls that is mainly used in master-detail scenarios. You select the record from master control (as example GridView) and the selected record is displayed in DetailView control.

In general GridView control displays more than one record, but the DetailsView control displays a single record from the database table. When you execute your web page, DetailsView control renders an HTML table.The DetailsView supports both declarative and programmatic databinding.

DetailsView control supports the edit, insert, delete and paging functionality.

Example:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Web.UI.WebControls;
using System.Web.UI;
public partial class Default : System.Web.UI.Page
{
   SqlConnection conn;
   SqlDataAdapter adapter;
   DataSet ds;
   SqlCommand cmd;
   string cs = ConfigurationManager.ConnectionStrings["conString"].ConnectionString;
   protected void Page_Load(object sender, EventArgs e)
   {
```

```
        if (!IsPostBack)
        {
            PopulateDetailView();
        }
    }
    protected void PopulateDetailView()
    {
        try
        {
            conn = new SqlConnection(cs);
            adapter = new SqlDataAdapter("select * from tblEmps", conn);
            ds = new DataSet();
            adapter.Fill(ds);
            DetailsView1.DataSource = ds;
            DetailsView1.DataBind();
        }
        catch (Exception ex)
        {
           Label1.Text = "ERROR :: " + ex.Message;
        }
    }
    protected void DetailsView1_PageIndexChanging(object sender, DetailsViewPageEventArgs
e)
    {
        DetailsView1.PageIndex = e.NewPageIndex;
        PopulateDetailView();
    }
    protected void DetailsView1_ModeChanging(object sender, DetailsViewModeEventArgs e)
    {
        DetailsView1.ChangeMode(e.NewMode);
        PopulateDetailView();
    }
}
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Web.UI.WebControls;
using System.Web.UI;
public partial class Default : System.Web.UI.Page
{
    SqlConnection conn;
    SqlDataAdapter adapter;
```

```
    DataSet ds;
    SqlCommand cmd;
    string cs = ConfigurationManager.ConnectionStrings["conString"].ConnectionString;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            PopulateDetailView();
        }
    }
    protected void PopulateDetailView()
    {
        try
        {
            conn = new SqlConnection(cs);
            adapter = new SqlDataAdapter("select * from tblEmps", conn);
            ds = new DataSet();
            adapter.Fill(ds);
            DetailsView1.DataSource = ds;
            DetailsView1.DataBind();
        }
        catch (Exception ex)
        {
          Label1.Text = "ERROR :: " + ex.Message;
        }
    }
    protected void DetailsView1_PageIndexChanging(object sender,
DetailsViewPageEventArgs e)
    {
        DetailsView1.PageIndex = e.NewPageIndex;
        PopulateDetailView();
    }
    protected void DetailsView1_ModeChanging(object sender,
DetailsViewModeEventArgs e)
    {
        DetailsView1.ChangeMode(e.NewMode);
        PopulateDetailView();
    }
}
```

FormView Control :

We use the FormView control to do anything that we also can do with the DetailsView control. Just as we can with the DetailsView control, we can use the FormView control to display, page, edit, insert, and delete database records. However, unlike the DetailsView control, the FormView control is entirely template driven. I end up using the FormView control much more than the DetailsView control. The FormView control provides us with more control over the layout of a form. Furthermore, adding validation controls to a FormView is easier than adding validation controls to a DetailsView control.

Example:

```
<%@ Page Language="VB" %>


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
   <style type="text/css">
      html
        {
      background-color:silver;
        }
   #content
        {
      margin:auto;
      width:300px;
      padding:10px;
      background-color:white;
      font:14px Georgia,Serif;
        }
   </style>
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
```

```
    <div id="content">
    <asp:FormView
       id="frmBooks"
       DataSourceID="SqlDataSource1"
       Runat="server">
       <ItemTemplate>
       <b><u><%# Eval("TITLE")%></u></b>
            <br />
       <b>Serial Number:</b>
       <%# Eval("ID")%>
       <br />
       <b>Author:</b>
       <%# Eval("AUTHOR")%>
       <br />
       <b>Price:</b>
       <%# Eval("PRICE")%>
       </ItemTemplate>
    </asp:FormView>
       <asp:SqlDataSource ID="SqlDataSource1" runat="server"
          ConnectionString="<%$ ConnectionStrings:DatabaseConnectionString1 %>"
          ProviderName="<%$
ConnectionStrings:DatabaseConnectionString1.ProviderName %>"
          SelectCommand="SELECT [ID], [TITLE], [AUTHOR], [PRICE] FROM
[BOOK_LIST]">
       </asp:SqlDataSource>
            </div>
    </form>
</body>
</html>
```