Controls Development

Till now we have been consuming the Controls that are provided by Microsoft and some times the available controls may not suit our requirements so in that situations ASP.Net provides an option of developing our own controls as per our requirements and we call them as User Controls or Custom Controls.

People working with Controls are categorized as Component Developers and Application Developers i.e. the one who develops controls are known as Component Developers and the one who develops applications by making use of the available controls are known as application developers, and we are application developers because we are developing applications with the help of controls provided by Microsoft.

Note: Sometimes an application developer can also develop controls on his own either to customize the existing controls or when the existing controls are not as per the requirements, and become a Component Developer.

To develop Controls in Asp.Net Webforms, it provides us 2 options:

1. Web Forms User Controls (User Controls)
2. Web Forms Server Controls (Custom Controls)

User Controls	**Custom Controls**
Designed for single-application scenarios. Deployed in the source form (.ascx) along with the source code of the application. If the same control needs to be used in more than one application, it introduces redundancy and maintenance problems.	Designed so that it can be used by more than one application. Deployed either in the application's Bin directory or in the GAC. Distributed easily and without problems associated with redundancy and maintenance.
Development is similar to the way Web Forms are developed; well-suited for rapid application development (RAD).	Development involves lots of code because there is no designer support.
A much better choice when you need static content within a fixed layout, for example, when you make headers and footers.	More suitable when an application requires dynamic content to be displayed; can be reused across an application, for example, for a data bound table control with dynamic rows.
Development doesn't require much application designing because they are authored at design time and mostly contain static data.	Development is done from the scratch, requires a good understanding of the control's life cycle and the order in which events execute, which is normally not taken care in user controls.

Developing a User Control:

To develop a User Control we are provided with an item template in the "Add New Item" window with the name "Web Forms User Control", which adds a new item in the project with ".ascx" extension and here also we find "Source View", "Design View" and "Code View" in ".ascx.cs" file, but this class will inherit from UserControl class which is present in System.Web.UI namespace only.

Developing a Customized Calendar Control as User Control:

Step 1: Open the "Add New Item", select "Web Forms" in LHS and now on the RHS we find "Web Forms User Control", select it, name it as "CalendarUserControl.ascx" and click on "Add" button. By default the .ascx file does not contain any html code in it and also in the place of "Page Directive" we find "Control Directive" but all the attributes will be same as attributes of "Page Directive".

Step 2: now write the following code in ".ascx" file below the "Control Directive":

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <asp:TextBox ID="txtDate" runat="server" />
    <asp:ImageButton ID="imgDate" runat="server" ImageUrl="~/Icons/calendar.png" />
    <asp:Calendar ID="cldDate" runat="server" Visible="false" NextMonthText="" FirstDayOfWeek="Monday" />
  </ContentTemplate>
</asp:UpdatePanel>
```

Step 3: now go to "ascx.cs" file and there we find a class with the name "CalendarUserControl" inheriting from the class "UserControl", write the following code in it:

Code under Image Button Click Event:

```
if (cldDate.Visible) { cldDate.Visible = false; }
else { cldDate.Visible = true; }
```

Code under Calendar SelectionChange Event:

```
txtDate.Text = cldDate.SelectedDate.ToShortDateString(); cldDate.Visible = false;
```

Code under Calendar DayRender Event:

```
if ((e.Day.Date.DayOfYear > DateTime.Now.DayOfYear && e.Day.Date.Year == DateTime.Now.Year)
                    || e.Day.Date.Year > DateTime.Now.Year) {
  e.Cell.ForeColor = Color.Gray; e.Day.IsSelectable = false; e.Cell.ToolTip = "Out of Range";
}
```

Code under Calendar VisibleMonthChanged Event:

```
if (e.NewDate.Month == DateTime.Now.Month && e.NewDate.Year == DateTime.Now.Year) {
  cldDate.NextMonthText = "";
}
else {
  cldDate.NextMonthText = "&gt;";
}
```

```
//Defining a property for setting a Date or getting the SelectedDate from CalendarUserControl
public = SelectedDate {
  get { return cldDate.SelectedDate; }
  set { cldDate.SelectedDate = value; }
}
```

```
//Defining a method to clear the Text of TextBox and reset the Date to current date
public void Reset() {
  txtDate.Text = ""; cldDate.SelectedDate = DateTime.Today; cldDate.VisibleDate = DateTime.Now;
}
```

Step 4: Now we can consume the CalendarUserControl in any WebForm of our project and to do that we need to register the control in our project which can be done in 2 different levels i.e Page Level and Application Level.

<u>Registering the User Control Page Level:</u>

In this approach we register the User Control in the WebForm where we want to consume it, but the drawback is if we want to consume the control in multiple WebForms, we need to register the control in every WebForm we want to consume it.

<%@ Register Src="<name of the .ascx file>" TagName="<a name with what we want to refer the control>" TagPrefix="<some name which we want to use as a prefix>" %>

- Src attribute is to specify the path of our ".ascx" file under which we developed the control.
- TagName is to specify the name with what we want to refer to the control and general we use our original class name only as TagName but can be changed to any value.
- TagPrefix is to specify a prefix value we want to use for our User Control just like we use "asp" as a TagPrefix for all our pre-defined Asp.Net controls.

To test this, add a new WebForm under the project naming it as "TestCalendarUserControl.aspx" and write the following code below Page Directive:

<%@ Register Src="~/CalendarUserControl.ascx" TagName="UserCalendar" TagPrefix="NIT1" %>

<u>Now write the following code under <div> tag:</u>

```
<h2 style="background-color:yellow;color:red;text-align:center">Application for Date of Birth Certificate</h2>
<table align="center">
 <caption>Applicant Details</caption>
 <tr> <td>Name:</td> <td><asp:TextBox ID="txtName" runat="server" /></td> </tr>
 <tr> <td>Gender:</td>
  <td>
   <asp:RadioButton ID="rbMale" runat="server" Text="Male" GroupName="Gender" />
   <asp:RadioButton ID="rbFemale" runat="server" Text="Female" GroupName="Gender" />
  </td>
 </tr>
 <tr> <td>Date of Birth:</td> <td><NIT1:UserCalendar ID="ucDate" runat="server" /></td> </tr>
 <tr> <td>Mobile No:</td> <td><asp:TextBox ID="txtMobile" runat="server" /></td> </tr>
 <tr> <td>Address:</td> <td> <asp:TextBox ID="txtAddress" runat="server" TextMode="MultiLine" /> </td> </tr>
 <tr> <td colspan="2" align="center">
   <asp:Button ID="btnSubmit" runat="server" Text="Submit Data" Width="100" />
   <asp:Button ID="btnClear" runat="server" Text="Reset Form" Width="100" />
  </td>
 </tr>
</table>
```

<u>Now goto TestCalendarUserControl.aspx.cs file and write the following code</u>

<u>Code under Page_Load:</u>

```
if (!IsPostBack) { txtName.Focus(); }
```

<u>Code under Submit Data Button:</u>

```
Response.Write("<script>alert('" + txtName.Text + "')</script>");
if (rbMale.Checked) { Response.Write("<script>alert('Applicant is male')</script>"); }
else if (rbFemale.Checked) { Response.Write("<script>alert('Applicant is female')</script>"); }
Response.Write("<script>alert('" + ucDate.SelectedDate.ToShortDateString() + "')</script>");
Response.Write("<script>alert('" + txtMobile.Text + "')</script>");
Response.Write("<script>alert('" + txtAddress.Text + "')</script>");
```

Code under Reset Form Button:

txtName.Text = txtEmail.Text = txtMobile.Text = ""; rbMale.Checked = rbFemale.Checked = false; ucDate.Reset(); txtName.Focus();

Registering the User Control Application Level:

In this approach we register the User Control under the project in Web.config file and the advantage with this is we can consume the Control in all the Web Forms of the project with out registering it again and again.

Note: To register a User Control in application level we have a restriction i.e. the UserControl and WebForm can't be present in the same folder, so add a new Folder under the project naming it as "UserControls" and drag the "CalendarUserControl.ascx" into the "UserControls" folder so that the WebForm is in the root folder of our project and UserControl is under "UserControls" folder which is present under the root folder.

To test the above first delete the "Register Directive" we have added in "TestCalendarUserControl.aspx", then open the Web.config file and write the following code inside <system.web> tag:

```
<pages>
  <controls>
    <add src="~/UserControls/CalendarUserControl.ascx" tagPrefix="NIT1" tagName="UserCalendar"/>
  </controls>
</pages>
```

Developing a Custom Control:

Open a new empty Asp.Net Web Application project naming it as "AspControls", open the "Add New Item" windows, select "Web Forms" in LHS and now on the RHS we find "Web Forms Server Control" select it, name it as "CustomCalendar.cs" and click on "Add" button. This will define a class with the name "CustomCalendar" inheriting for "WebControl" class of "System.Web.UI.WebControls" namespace and we need to write all the code for designing and executing in this class only. Now if we look into the code we will find 2 attributes on top of the class "DefaultProperty" and "ToolBoxData".

- DefaultProperty specifies the default property for this control and it's "Text"
- ToolBoxData property is to specify the information about the code that should be generated when we drag and drop the control from "ToolBox" on to a WebForm and right now the data will be as following:

  [ToolboxData("<{0}:CustomCalendar runat=server></{0}:CustomCalendar>")]

In place of "{0}" the TagPrefix what we provide while registering the control will come and "CustomCalendar" in the name of Control which is nothing but our class name, which we can either leave the same or change it as per our requirements. Now delete all the existing code from the class and write our code inside the class which should finally look as following:

```
public class CustomCalendar : WebControl {
  TextBox txtDate; ImageButton imgDate; Calendar cldDate;
  protected override void CreateChildControls() {
    txtDate = new TextBox(); txtDate.ID = "txtDate"; txtDate.Width = Unit.Pixel(200);
    imgDate = new ImageButton(); imgDate.ID = "imgDate"; imgDate.Click += ImgDate_Click;
    cldDate = new Calendar(); cldDate.ID = "cldDate"; cldDate.Visible = false; cldDate.NextMonthText = "";
    cldDate.FirstDayOfWeek = FirstDayOfWeek.Monday; cldDate.SelectionChanged += CldDate_SelectionChanged;
    cldDate.DayRender += CldDate_DayRender; cldDate.VisibleMonthChanged += CldDate_VisibleMonthChanged;
    this.Controls.Add(txtDate); this.Controls.Add(imgDate); this.Controls.Add(cldDate);
  }
```

```
 private void ImgDate_Click(object sender, ImageClickEventArgs e) {
  if (cldDate.Visible) { cldDate.Visible = false; }
  else { cldDate.Visible = true; }
 }
 private void CldDate_SelectionChanged(object sender, EventArgs e) {
  txtDate.Text = cldDate.SelectedDate.ToShortDateString(); cldDate.Visible = false;
 }
 private void CldDate_DayRender(object sender, DayRenderEventArgs e) {
  if ((e.Day.Date.DayOfYear > DateTime.Now.DayOfYear && e.Day.Date.Year == DateTime.Now.Year)
            || e.Day.Date.Year > DateTime.Now.Year) {
   e.Cell.ForeColor = System.Drawing.Color.Gray;
   e.Day.IsSelectable = false; e.Cell.ToolTip = "Out Of Range";
  }
 }
 private void CldDate_VisibleMonthChanged(object sender, MonthChangedEventArgs e) {
  if (e.NewDate.Month == DateTime.Now.Month && e.NewDate.Year == DateTime.Now.Year) {
   cldDate.NextMonthText = "";
  }
  else { cldDate.NextMonthText = "&gt;"; }
 }
 public DateTime SelectedDate {
  get { EnsureChildControls(); return cldDate.SelectedDate; }
  set { EnsureChildControls(); cldDate.SelectedDate = value; }
 }
 public string ButtonImageUrl {
  get { EnsureChildControls(); return imgDate.ImageUrl; }
  set { EnsureChildControls(); imgDate.ImageUrl = value; }
 }
 public void Reset() {
  txtDate.Text = ""; cldDate.SelectedDate = DateTime.Today; cldDate.VisibleDate = DateTime.Now;
 }
 public override void RenderControl(HtmlTextWriter writer) {
  txtDate.RenderControl(writer); imgDate.RenderControl(writer); cldDate.RenderControl(writer);
 }
}
```
Note: now build the project to compile and generate an assembly which will have the name "AspControls.dll".

In CustomControls, CreateChildControls method should be used for writing the logic for creating the child controls, setting the properties and binding event handlers to events as we performed in the above code. CreateChildControls method is defined as "virtual" in "Control" class which is a parent class for all Controls, so we have overriden that method and implemented the logic for creating TextBox, ImageButton and Calendar.

EnsureChildControls method is typically used in Custom Controls which use child controls for functionality. The EnsureChildControls method is called in order to make sure that child controls have been created and are ready to process input, to perform data binding, or to perform other tasks. This method first checks whether the child controls are created or not and if not created then CreateChildControls method is called.

Every Asp.Net Server Control contains RenderControl method which is automatically called by the page during rendering, which outputs server control content to the provided HtmlTextWriter object. This method is defined as "virtual" in "Control" class which is a parent class for all Controls and Custom Control developers should override this method and implement the logic for rendering their control. We have overriden that method in our Control and called the RenderControl method of TextBox, ImageButton and Calendar because those controls already contain that method.

Consuming the Custom Control: the advantage with Custom Controls is they can be added to ToolBox and then we can "Drag and Drop" them on any WebForm and to do that go back to our regular project i.e. "ControlsDemo", open the ToolBox window, right click on it and choose the option "Add Tab" which will add a new tab, specify a name to it for example: "NIT Controls". Now right click on the new tab and select the option "Choose Items" which will open "Choose ToolBox Items" window, click on the "Browse" button on it and select "AspControls.dll" from its physical location and click "Ok" which will add "CustomCalendar" under the new tab.

Registering the Custom Control Page level: to register the Custom Control page level use the following statement on a WebForm below page directive:

```
<%@ Register Assembly="AspControls" Namespace="AspControls" TagPrefix="NIT2" %>
```

Registering the Custom Control Application level: to register the Custom Control application level open the Web.config file and write code under <system.web> tag where we registered "CalendarUserControl" which should now look as following:

```
<pages>
  <controls>
    <add assembly="AspControls" namespace="AspControls" tagPrefix="NIT2"/>
  </controls>
</pages>
```

Now add a new WebForm naming it as "TestCustomCalendar.aspx" and write the following code under <div> tag:

```
<h2 style="background-color:yellow;color:red;text-align:center">Application for Date of Birth Certificate</h2>
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<table align="center">
  <caption>Applicant Details</caption>
  <tr> <td>Name:</td> <td><asp:TextBox ID="txtName" runat="server" /></td> </tr>
  <tr>  <td>Gender:</td>
    <td>
      <asp:RadioButton ID="rbMale" runat="server" Text="Male" GroupName="Gender" />
      <asp:RadioButton ID="rbFemale" runat="server" Text="Female" GroupName="Gender" />
    </td>
  </tr>
  <tr> <td>Date of Birth:</td>
    <td>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server"> <ContentTemplate>
        <NIT2:CustomCalendar ID="ccDate" runat="server" />
      </ContentTemplate> </asp:UpdatePanel>
    </td>
  </tr>
  <tr> <td>Mobile No:</td> <td><asp:TextBox ID="txtMobile" runat="server" /></td> </tr>
  <tr> <td>Address:</td> <td> <asp:TextBox ID="txtAddress" runat="server" TextMode="MultiLine" /> </td> </tr>
  <tr> <td colspan="2" align="center">
      <asp:Button ID="btnSubmit" runat="server" Text="Submit Data" Width="100" />
      <asp:Button ID="btnClear" runat="server" Text="Reset Form" Width="100" />
    </td>
  </tr>
</table>
```

Now goto TestCustomCalendar.aspx.cs file and write the following code

Code under Page_Load:

```
if (!IsPostBack) { txtName.Focus(); ccDate.SelectedDate = DateTime.Today; }
```

Code under Submit Data Button:

```
Response.Write("<script>alert('" + txtName.Text + "')</script>");
if (rbMale.Checked) { Response.Write("<script>alert('Applicant is male')</script>"); }
else if (rbFemale.Checked) { Response.Write("<script>alert('Applicant is female')</script>"); }
Response.Write("<script>alert('" + ccDate.SelectedDate.ToShortDateString() + "')</script>");
Response.Write("<script>alert('" + txtMobile.Text + "')</script>");
Response.Write("<script>alert('" + txtAddress.Text + "')</script>");
```

Code under Reset Form Button:

```
txtName.Text = txtEmail.Text = txtMobile.Text = ""; rbMale.Checked = rbFemale.Checked = false;
ccDate.Reset(); txtName.Focus();
```

Developing a VideoPlayer control as a Custom Control:

Most of the existing Html controls are developed by Microsoft as “Asp.Net Server Controls” and given for us to consume and all those controls will finally render Html, whereas controls introduced in “Html 5” like <audio> and <video> are not provided to us in Asp.Net as Server Controls to let us create a VideoPlayer control which will render into “Html 5 Video”.

To perform this task go to “AspControls” project under which we have developed “CustomCalendar”, add a new “Web Form Server Control” in it naming it as “VideoPlayer.cs”, delete the whole content present inside the class and write our code in it which should look as following:

```
public class VideoPlayer : WebControl {
  public bool AutoPlay { get; set; } = false;
  public new bool Controls { get; set; } = false;
  public bool Loop { get; set; } = false;
  public bool Muted { get; set; } = false;
  public string Poster { get; set; } = null;
  public string Mp4Url { get; set; } = null;
  public string WebMUrl { get; set; } = null;
  public string OggUrl { get; set; } = null;
  public override void RenderControl(HtmlTextWriter writer) {
    writer.AddAttribute(HtmlTextWriterAttribute.Id, this.ID);
    writer.AddAttribute(HtmlTextWriterAttribute.Name, this.ID);
    writer.AddAttribute(HtmlTextWriterAttribute.Width, this.Width.ToString());
    writer.AddAttribute(HtmlTextWriterAttribute.Height, this.Height.ToString());
    if (AutoPlay) { writer.AddAttribute("autoplay", "autoplay"); }
    if (Controls) { writer.AddAttribute("controls", "controls"); }
    if (Loop) { writer.AddAttribute("loop", "loop"); }
    if (Muted) { writer.AddAttribute("muted", "muted"); }
    if (Poster != null) { writer.AddAttribute("poster", this.Poster); }

    writer.RenderBeginTag("video");
    if (this.Mp4Url != null) {
      writer.AddAttribute(HtmlTextWriterAttribute.Src, this.Mp4Url);
      writer.AddAttribute(HtmlTextWriterAttribute.Type, "video/mp4");
      writer.RenderBeginTag("source");
      writer.RenderEndTag();
    }
    if (this.WebMUrl != null) {
      writer.AddAttribute(HtmlTextWriterAttribute.Src, this.WebMUrl);
      writer.AddAttribute(HtmlTextWriterAttribute.Type, "video/webm");
      writer.RenderBeginTag("source");
      writer.RenderEndTag();
    }
    if (this.OggUrl != null) {
      writer.AddAttribute(HtmlTextWriterAttribute.Src, this.OggUrl);
      writer.AddAttribute(HtmlTextWriterAttribute.Type, "video/ogg");
      writer.RenderBeginTag("source");
      writer.RenderEndTag();
    }
    writer.RenderEndTag();
  }
}
```

Consuming the VideoPlayer Control: to consume the VideoPlayer we have developed; first compile the project so that the VideoPlayer control is added to "ASPControls.dll", now go back to "ControlDemo" project, open the ToolBox window, right click on "NIT Controls" tab we have added earlier, select "Choose Items", click "Browse" button and select "ASPControls.dll" from its physical location which will add "VideoPlayer" control below the "CustomCalendar". Now Add 2 new folders under the project naming them as "Images" and "Videos" and then add 1 image into "Images" folder and 1 video into "Vidoes" folder, then add a new WebForm in the project naming it as "TestVideoPlayer.aspx" and write the following code in its <div> tag:

```
<NIT2:VideoPlayer ID="Video1" runat="server" Poster="/Images/Mumbai.jpg" Mp4Url="/Videos/Mumbai.mp4"
Controls="true" Muted="true" Loop="true" Width="400" Height="400" />
```

Master Pages

One attribute of a well-designed website is a consistent site-wide page layout. Take the www.nareshit.com website, for example, every page has the same content at the top and bottom of the page. At the very top of each page displays a light gray bar with a list of locations where the branches are located. Beneath that are the company logo, and the core sections: Home, All Courses, Software Training, Services, and so forth. Likewise, the bottom of the page includes information about clients, address, contact details, quick links etc.

Another attribute of a well-designed site is the ease with which the site's appearance can be changed. For instance, the look and feel can be changed in the future, perhaps the menu items along the top will expand to include a new section or may be a radically new design with different colors, fonts, and layout will be unveiled. Applying such changes to the entire site should be a fast and simple process that does not require modifying the thousands of web pages that make up the site.

Creating a site-wide page template in ASP.NET is possible through the use of master pages. In a nutshell, a master page is a special type of ASP.NET page that defines the markup that is common among all content pages (a content page is an ASP.NET page that is bound to the master page). Whenever a master page's layout or formatting is changed, all of its content pages output is immediately updated, which makes applying site-wide appearance and changes as easy by updating and deploying a single file i.e. master page.

<u>Understanding How Master Pages Work:</u>

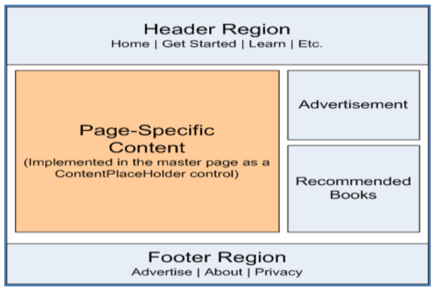
Building a website with a consistent site-wide page layout requires that each web page emit common formatting markup for example, while each page on www.nareshit.com have their own unique content, each of these pages also render a series of common <div> elements that display the top-level section links: Home, All Courses, Software Training, Services, and so on.

There are a variety of techniques for creating web pages with a consistent look and feel. A naive approach is to simply copy and paste the common layout markup into all web pages, but this approach has a number of downsides. For starters, every time a new page is created, you must remember to copy and paste the shared content into the page. Such copying and pasting operations are ripe for error as you may accidentally copy only a subset of the shared markup into a new page. And to top it off, this approach makes replacing the existing site-wide appearance with a new one a real pain because every single page in the site must be edited in order to use the new look and feel.

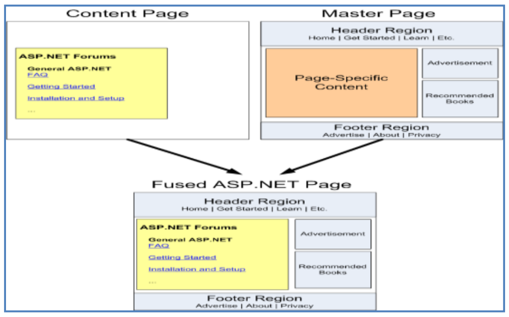
Prior to ASP.NET version 2.0, page developers often placed common markup in User Controls and then added these User Controls to each and every page. This approach require the page developer to remember that they need to manually add the User Controls to every new page, and this allowed for easier site-wide modifications because when updating the common markup only the User Controls needed to be modified. Unfortunately, Visual Studio.NET 2002 and 2003 which used to create ASP.NET 1.x applications - rendered User Controls in the Design view as gray boxes. Consequently, page developers using this approach did not enjoy a WYSIWYG design-time environment.

The shortcomings of using User Controls were addressed in ASP.NET version 2.0 and Visual Studio 2005 with the introduction of master pages. A master page is a special type of ASP.NET page that defines the site-wide markup and the regions where associated content pages can define their custom markup and those regions are defined by "ContentPlaceHolder" controls. The "ContentPlaceHolder" control simply denotes a position in the master page's control hierarchy where custom content can be injected by a content page.

Below figure shows how a master page might look like. Note that the master page defines the common site-wide layout - the markup at the top, bottom, and right of every page - as well as a "ContentPlaceHolder" in the middle-left where the unique content of each individual web page has to come and sit.


Header Region
Home | Get Started | Learn | Etc.
Page-Specific Content
(Implemented in the master page as a ContentPlaceHolder control)
Advertisement
Recommended Books
Footer Region
Advertise | About | Privacy

Once a master page has been defined it can be bound to a new ASP.NET page and those ASP.NET pages (content pages), include a Content control for each of the master page's ContentPlaceHolder controls. When the content page is visited through a browser the ASP.NET engine creates the master page's control hierarchy and injects the content page's control hierarchy into the appropriate places and the combined control hierarchy is rendered and the resulting HTML is returned to the end user's browser. Below figure illustrates this concept:



Creating a Master Page:

To add a Master Page in our site open the “Add New Item” window and select the option “WebForms Master Page” and name it as “Site.master” (.master is the extension of a Master Page). The first line in the declarative markup is the @Master directive and this is similar to the @Page directive that appears in our ASP.NET pages and all attributes will be same as @Page directive attributes only.

Now if we look into the code of the file we find a “ContentPlaceHolder” control named head and this control appears within the <head> section and can be used to declaratively add content in to the <head> element like style and script code. We also find another “ContentPlaceHolder” control named “ContentPlaceHolder1” and this control appears within the Web Form and serves as the region for the content page's user interface.

Now delete the code present inside the <form> tag along with the “ContentPlaceHeader” and write the below code in it:

```
<div id="divHeader" style="background-color: lightpink; color: aqua; text-align: center; font-size: xx-large">
  Naresh I Technologies
</div>
<div id="divMenu" style="background-color: beige; text-align: center">
  <asp:HyperLink ID="hlHome" runat="server" NavigateUrl="~/Home.aspx" Text="Home" />&nbsp;
  <asp:HyperLink ID="hlMission" runat="server" NavigateUrl="~/Mission.aspx" Text="Mission" />&nbsp;
  <asp:HyperLink ID="hlAbout" runat="server" NavigateUrl="~/About.aspx" Text="About" />
</div>
<div id="divContent"> <asp:ContentPlaceHolder ID="body" runat="server" /> </div>
<div id="divFooter" style="background-color: azure">
  <fieldset style="border:dotted blue;color:brown">
    <legend>Contact Us</legend>
    Address: Opp. Satyam Theatre, Ameerpet, Hyderabad – 16 <br />
    Phone: 2374 6666 <br /> Email: info@nareshit.com <br /> Website: www.nareshitc.com
  </fieldset>
</div>
```

Creating Associated Content Pages: with the above master page created, we are now ready to start creating ASP.NET pages that are bound to the master page. Such pages are referred to as content pages and we need to create 3 content pages Home.aspx, Mission.aspx and About.aspx.

Add a new ASP.NET page to the project and bind it to the "Site.master" master page and to do that right-click on the project name in Solution Explorer and choose "Add New Item" option, select the "Web Form with Master Page" template, enter the name as "Home.aspx" and click "Add", which opens "Select a Master Page" dialog box from where you can choose the master page to use. This action will add a content page which contains a @Page directive that points back to its master page and a "Content" control for each of the Master Page's "ContentPlaceHolder" controls. Currently we have 2 "ContentPlaceHolder" controls on Master Page so we find 2 "Content" controls on this page and the code will be as following in "Home.aspx":

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
                CodeBehind="Home.aspx.cs" Inherits="ControlsDemo.Home" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server"></asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="body" runat="server"></asp:Content>
```

Title attribute is to set a title for our page, give a value to title as "Home Page" and then write the following code under the 2 "Content" controls:

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style>
    p { text-align: justify; color: palevioletred; font-family: Arial; text-indent: 50px; }
    ol { color: cadetblue; background-color: burlywood }
  </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="body" runat="server">
  <p> We have set the pace with online learning. Learn what you want, when you want, and practice with the
        instructor-led training sessions, on-demand video tutorials which you can watch and listen. </p>
  <ol>
    <li>150+ Online Courses</li>
    <li>UNLIMITED ACCESS</li>
    <li>EXPERT TRAINERS</li>
    <li>VARIETY OF INSTRUCTION</li>
    <li>ON-THE-GO-LEARNING</li>
    <li>PASSIONATE TEAM</li>
    <li>TRAINING INSTITUTE OF CHOICE</li>
  </ol>
</asp:Content>
```

Same as the above add 2 more content pages naming them as "Mission.aspx" and "About.aspx", set the title as "Mission Page" for "Mission.aspx" and "About Page" for "About.aspx", and write below code under them:

Mission.aspx:

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style>
    p { text-align: justify; color: blue; font-family: Calibri; text-indent: 50px; }
  </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="body" runat="server">
```

```
 <p> We appreciate you taking the time today to visit our web site. Our goal is to give you an interactive tour of
our new and used inventory, as well as allow you to conveniently get a quote, schedule a service appointment, or
apply for financing. The search for a luxury car is filled with high expectations. Undoubtedly, that has a lot to do
with the vehicles you are considering, but at Motors, we think you should also have pretty high expectations for
your dealership. </p>
</asp:Content>
```

About.aspx:

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style>
    p { text-align: justify; color: green; font-family: 'Agency FB'; text-transform: capitalize; text-indent: 50px; }
  </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="body" runat="server">
  <p> NareshIT (Naresh i Technologies) is a leading Software Training Institute and provides Job Guarantee Program
through Nacre in Hyderabad, Chennai, Bangalore, Vijayawada and across the world with Online Training services.
Managed and Lead by IT Professionals with more than a decade experience in leading MNC companies. We are
most popular for our training approach that enables students to gain real-time exposure on cutting-edge
technologies. </p>
</asp:Content>
```

Now run any of the above 3 content pages and we notice that they merge with master page we have created above and launches, click on any menu in the top will launch other pages in the site.

Creating another Master Page:

Add a new Master Page in our site naming it as “NITSite.master”. Add a new folder under the project naming it as “Images” and copy 2 images into it which we can use them as header and footer of our new Master Page. Delete the code present under the <form> tag of “MasterPage” and write the below code there:

```
<table style="width:100%">
  <tr> <td colspan="2"><asp:Image ID="imgHeader" runat="server" Width="100%" Height="100"
        ImageUrl="~/Images/Header.png" /> </td> </tr>
  <tr>
    <td style="background-color: bisque; width: 25%; vertical-align: top">
      <b>Courses Offered:</b> <br />
      <center>
        <asp:HyperLink ID="hlCS" runat="server" Text="CSharp" NavigateUrl="~/CSharp.aspx" /> <br />
        <asp:HyperLink ID="hlAsp" runat="server" Text="Asp.Net WebForms" NavigateUrl="~/Asp.aspx" /> <br />
        <asp:HyperLink ID="hlMvc" runat="server" Text="Asp.Net MVC" NavigateUrl="~/Mvc.aspx" /> <br />
        <asp:HyperLink ID="hlSql" runat="server" Text="Sql Server" NavigateUrl="~/Sql.aspx" />
      </center> <br /><br />
      <b>Start Date: </b> <asp:ContentPlaceHolder ID="cphDate" runat="server" /> <br />
      <b>Batch Time: </b> <asp:ContentPlaceHolder ID="cphTime" runat="server" /> <br />
      <b>Course Fees: </b> <asp:ContentPlaceHolder ID="cphFees" runat="server" /> <br />
      <b>Faculty: </b> <asp:Label ID="lblFaculty" runat="server" />
    </td>
    <td style="vertical-align: top; width: 75%; background-color: azure">
```

```
      <asp:ContentPlaceHolder ID="cphBody" runat="server" />
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <asp:Image ID="imgFooter" runat="server" Width="100%" Height="50" ImageUrl="~/Images/Footer.png" />
    </td>
  </tr>
</table>
```

Add a new “Web Form with Master Page” under the project naming it as “CSharp.aspx” and write the below code in it under the 5 content controls and also set the Title as “CSharp”.

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style> p { text-align: justify; font-family: Arial; text-indent: 50px; color: dodgerblue; } </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphDate" runat="server">06/01/2020</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="cphTime" runat="server">11:00 A.M.</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="cphFees" runat="server">&#8377; 1500/-</asp:Content>
<asp:Content ID="Content5" ContentPlaceHolderID="cphBody" runat="server">
  <p>It is an Object-Oriented and Platform Independent programming language developed by Microsoft as part of the .NET initiative and approved as a standard by ECMA and ISO.</p>
  <p>Anders Hejlsberg leads development of the language, which has procedural, object-oriented syntax based on C++ and includes influences from several programming languages most importantly Delphi and Java with a particular emphasis on simplification.</p>
  <p>CSharp’s principal designer and lead architect Anders Hejlsberg, has previously involved with the design of Visual J++, Borland Delphi, and Turbo Pascal languages also. In interviews and technical papers he has stated that flaws in most major programming languages like C++, Java, Delphi, and Smalltalk drove the design of CSharp programming language.</p>
</asp:Content>
```

Add another “Web Form with Master Page” under the project naming it as “Asp.aspx” and write the below code in it under the 5 content controls and also set the Title as “Asp.Net Web Forms”.

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style> p { text-align: justify; font-family: Arial; text-indent: 50px; color: crimson; } </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphDate" runat="server">06/01/2020</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="cphTime" runat="server">7:00 A.M.</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="cphFees" runat="server">&#8377; 2000/-</asp:Content>
<asp:Content ID="Content5" ContentPlaceHolderID="cphBody" runat="server">
  <p>It is an open-source server-side web application framework designed by Microsoft for web development to produce dynamic web pages. It is designed to allow programmers to build Web Sites, Web Applications and Web Services.</p>
  <p>It was first released in January 5, 2002 with 1.0 version of .NET Framework, and is a successor to Microsoft's Active Server Pages (ASP) technology. The current version of ASP.Net is 4.7 released on April 5, 2017 which will be the last version and it is succeeded by ASP.Net Core. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code by using any supported .NET language and while coding we have
```

```
access to all the BCL in .NET Framework, which enable us to benefit from the CLR, type safety, inheritance, polymorphism and so on.</p>
</asp:Content>
```

Add another "Web Form with Master Page" under the project naming it as "Mvc.aspx" and write the below code in it under the 5 content controls and also set the Title as "Asp.Net MVC".

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style>
    p { text-align: justify; font-family: Arial; text-indent: 50px; color:coral }
  </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphDate" runat="server">06/01/2020</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="cphTime" runat="server">11:00 A.M.</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="cphFees" runat="server">&#8377; 1000/-</asp:Content>
<asp:Content ID="Content5" ContentPlaceHolderID="cphBody" runat="server">
  <p>The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data if requires.</p>
  <p>MVC traditionally used for desktop graphical user interfaces (GUIs), this architecture has become popular for designing web applications and even mobile, desktop and other clients. Popular programming languages like Java, C#, Ruby, PHP and others have their own MVC frameworks that are currently being used in web application development.</p>
  <p>Trygve Reenskaug introduced MVC into Smalltalk-76 while visiting the Xerox Palo Alto Research Center in the 1970s. In the 1980s, Jim Althoff and others implemented a version of MVC for the Smalltalk-80 class library.</p>
</asp:Content>
```

Add another "Web Form with Master Page" under the project naming it as "Sql.aspx" and write the below code in it under the 5 content controls and also set the Title as "Sql Server".

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
  <style> p { text-align: justify; font-family: Arial; text-indent: 50px; color:darkorchid } </style>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphDate" runat="server">06/01/2020</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="cphTime" runat="server">2:00 P.M.</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="cphFees" runat="server">&#8377; 1000/-</asp:Content>
<asp:Content ID="Content5" ContentPlaceHolderID="cphBody" runat="server">
  <p>Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications--which may run either on the same computer or on another computer across a network.</p>
  <p>Microsoft markets at least a dozen different editions of Microsoft SQL Server, aimed at different audiences ranging from small single-machine applications to large Internet applications with many concurrent users.</p>
  <p> Microsoft and Sybase released version 1.0 in 1989. However, the partnership between these two ended in the early 1990s. Microsoft maintained ownership rights to the name SQL Server. Since the 1990s, subsequent versions of SQL Server have been released including SQL Server 2000, 2005, 2008, 2012, 2014, 2016 and 2017.</p>
</asp:Content>
```

Now run any of the 4 content pages and watch the output, where we notice the Content Page launched by merging with Master Page and also we find the links to other pages on the LHS of the page and by clicking on the appropriate link the corresponding page gets loaded.

Passing values to Controls present on a Master Page, from Content Pages: If we want to pass any value to a control that is present on a Master Page, from Content Page; 1st we need to give access to that Control to Content Page which can be performed in 4 different ways.

Option 1: By using the "Master" property in a Content Page we can get a reference to the Master Page of that Content Page and with that reference we can get access to the Controls of Master Page by calling "FindControl" method on that reference and to test that, go to "CSharp.aspx.cs" file and write the below code under "Page_Load" method:

```
MasterPage mp = Master;
Control ctrl = mp.FindControl("lblFaculty");
Label lblFaculty = (Label)ctrl;
lblFaculty.Text = "Bangarraju";
                Or
Control ctrl = Master.FindControl("lblFaculty");
Label lblFaculty = (Label)ctrl;
lblFaculty.Text = "Bangarraju";
                Or
Label lblFaculty = (Label)Master.FindControl("lblFaculty");
lblFaculty.Text = "Bangarraju";
                Or
((Label)this.Master.FindControl("lblFaculty")).Text = "Bangarraju";
```

Note: NITSite is a Master Page and the parent class for every Master Page is "MasterPage" class which is defined in "System.Web.UI" Namespace and this is very similar to a normal Web Page inheriting from "Page" class.

Option 2: By defining a property in the Master Page we can expose the Control to Content Pages, so that Content Pages can access the Control and assign values to them and to test that, go to "NITSite.master.cs" file and write the below code in the class:

```
public Label Faculty {
 get { return lblFaculty; }
}
```

Now in the Content Page we can directly access the Control without calling "FindControl" method and to test that process go to "Asp.aspx.cs" file and write the below code in "Page_Load" Method:

```
MasterPage mp = Master;
NITSite obj = (NITSite)mp;
obj.Faculty.Text = "Bangarraju";
          Or
NITSite obj = (NITSite)Master;
obj.Faculty.Text = "Bangarraju";
          Or
((NITSite)Master).Faculty.Text = "Bangarraju";
```

Note: In the above case “Master” property will provide a reference to the Master Page of current Content Page but in the form of Parent i.e. “MasterPage” class but not as “NITSite” class, so with that reference we can’t call “Faculty” property because “Parent can hold the reference of Child but can’t access members of Child (Rule No. 3 of Inheritance)” and that is the reason why we have converted that reference from Parent Type (MasterPage) back into Child Type (NITSite) for accessing the “Faculty” property.

Option 3: As discussed above the return type of “Master” property in any Content Page is “MasterPage”, and that property is defined in “Page” class which is the parent class of Content Page, so we can re-implement that property in Content Page and change the return type of that property to our actual Master Page i.e. “NITSite” so that we can directly access the members of that class in Content Page and to test that process go to “Mvc.aspx.cs” file and write the below property in the class:

```
public new NITSite Master {
 get {
  return (NITSite)base.Master;
 }
}
```

Note: in the above case to re-implement the property we have used “Hiding or Shadowing” because that property is not declared as “Virtual” in Page class.

Now we can directly get access to “NITSite” class in Content Page by using the Master property we have defined above and to test that, write the following code in Page_Load method:

Master.Faculty.Text = "Sudhakar Sharma";

Option 4: Without again defining a “Master” property explicitly in the Content Pages we are provided with “MasterType” directive which if used in Content Pages, will automatically define the “Master” property same as we defined above and to test that, go to “Sql.aspx” file and write the following statement below “Page Directive”:

<%@ MasterType VirtualPath="~/NITSite.Master" %>

This action will immediately define a Master property under the current class and to check that, open Solution Explorer, expand the WebForm “Sql.aspx”, and below that we find “Sql.aspx.designer.cs” file, open it and there we find “Master” property same as we defined in “Mvc.aspx.cs” file above, so now we can directly use that “Master” property and access the “Faculty” property, and to test that go to “Sql.aspx.cs” file and write the below code in “Page_Load” method:

Master.Faculty.Text = "Sudhakar Ladda";

ViewState

Web applications are stateless i.e. these applications do not save data of 1 request, for using it in the next request. When an application is stateless, the server does not store any state about the client session. To test this, add a new WebForm naming it as “HitCount.aspx” and write the following code in it’s <div> tag:

<asp:Button ID="btnCount" runat="server" Text="Hit Count" /> <br />
<asp:Label ID="lblCount" runat="server" ForeColor="Red" />

Now goto “HitCount.aspx.cs” file and write the following code in it:

Declarations: int Count = 0;

Code under Hit Count Button Click:

Count += 1; lblCount.Text = "Hit Count: " + Count;

Now run the WebForm and click on the button for any no. of times still it will display the output as 1 only because every time we click on the button, server will create the instance of our page class, initializes “Count” with “0” and increments the value of it to “1”, destroys the page instance and then sends the output to client machine, and this process repeats every time we click on the button so every time the value of Count is “1” only and this is called as “Stateless” behavior. To overcome the above problem we use ViewState which is a technique used to store user data on page at the time of post back of a web page.

Syntax of storing a value into ViewState: ViewState[string name] = value (Object);

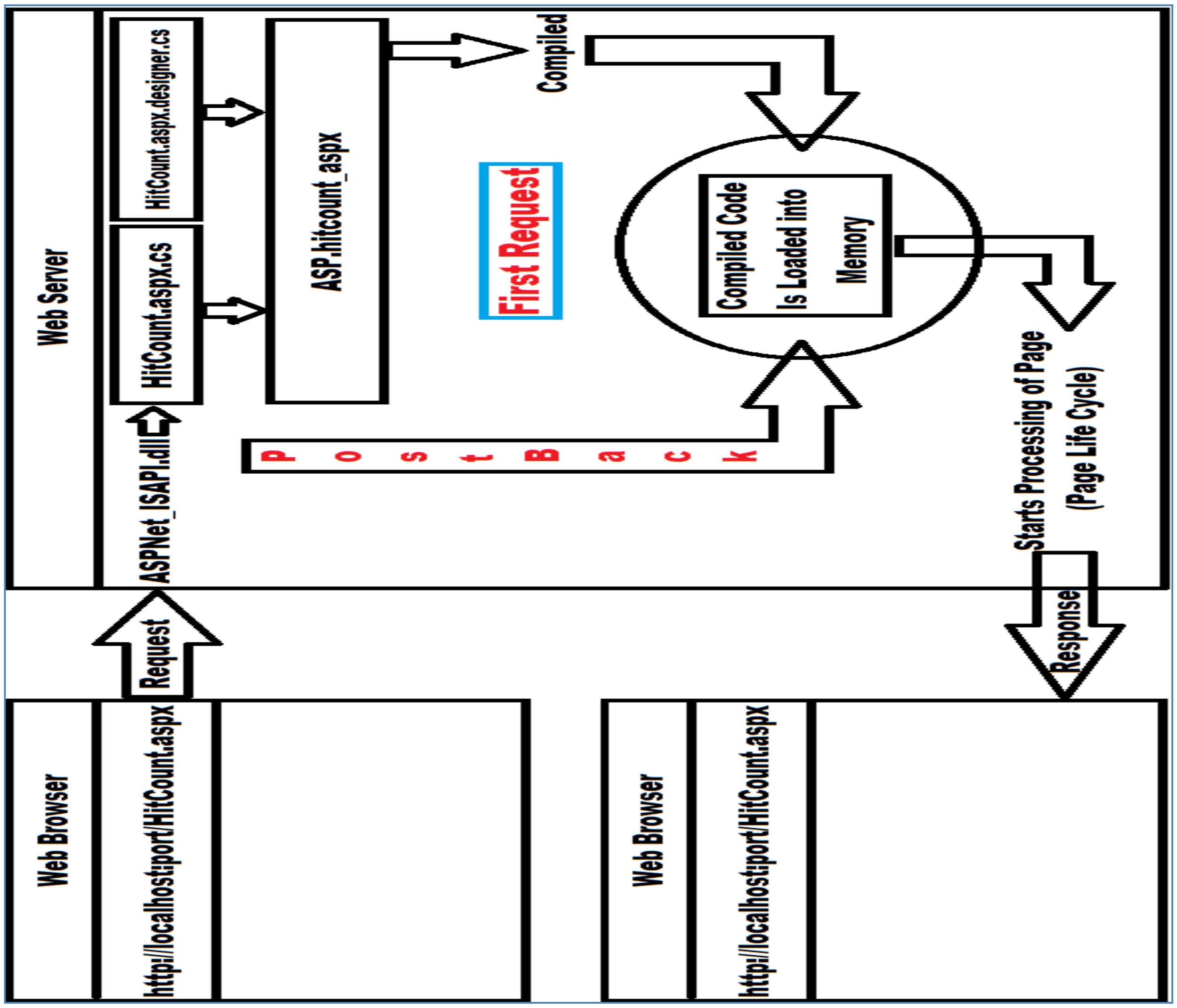
Syntax of accessing a value from ViewState: Object value = ViewState[string name];

To resolve the problem in our previous page rewrite the code under click of the button as following:

```
if (ViewState["Counter"] == null) { Count = 1; }
else { Count = (int)ViewState["Counter"] + 1; ViewState["Counter"] = Count; lblCount.Text = "Hit Count: " + Count; }
```

Note: Same as the above every Control will also maintain the state of its values and we call it as ControlState which will hold the values of controls and restores the value to control after page post back.

Processing a Page on the Server


Web Server
HitCount.aspx.designer.cs
HitCount.aspx.cs
ASP.hitcount_aspx
Compiled
First Request
Compiled Code Is Loaded into Memory
Starts Processing of Page (Page Life Cycle)
PostBack
ASPNet_ISAPI.dll
Request
Response
Web Browser
http://localhostport/HitCount.aspx
Web Browser
http://localhostport/HitCount.aspx

When the request comes from a client for the first time to a page, ASPNet_ISAPI.dll on the Web Server will take the request, creates a new class "ASP.hitcount_aspx" inheriting from the class HitCount which is defined on "HitCount.aspx.cs" and "HitCount.aspx.designer.cs" files, compiles the new class and loads the compiled code into memory, and if at all the request comes for the second time (PostBack) for the page which is already compiled and loaded into memory, then with out recompiling the page again, the compiled code in the memory will be used for starting the life cycle of page.

Page Life Cycle

Life cycle of a page describes how a Web Page gets processed on the server, when browser sends a request for that page.

Page Request: The page request occurs before the page life cycle begins, i.e. when the page is requested by a user, server determines whether the page needs to be parsed and compiled (therefore beginning the life cycle of a page) or whether a cached version of the page can be sent in response without running the page.

Various stages in the life cycle of a page will be as following:

Stage1: Start

In this stage the page properties such as request and response are created and set, and at this stage the page also determines whether the request is a postback or a new request and sets the IsPostBack property, which will be true if it's a PostBack request or else false if it's a first request.

Stage2: Initialization

During this stage all the controls on the page will be created and each controls unique Id property will be set.

Stage3: Load

In this stage if the current request is a postback, control properties are loaded with information retrieved from View State (Control State).

Stage4: Validation

In this stage any server side validations that are required will be performed and sets the IsValid property of individual validator controls and then of the page.

Stage5: Event Handling

If the request is a posback then all the controls corresponding event handlers are called (which is required only) along with any cached events also.

Stage6: Rendering

In this stage the page is rendered i.e. converts into Html and before rendering view state and control state is saved for the page and all controls.

Note: During the rendering stage, page calls the render method for each control and writes its output to the OutputStream object of Page's Response property.

Stage7: UnLoad:

After the page has been rendered and the output is sent to client, it's ready to discard and Unload event is raised, and at this stage the Request and Response properties are destroyed and cleanup is performed.

Page Events: Within each stage of page life cycle, page raises certain events which can be used for performing certain actions and these are the List of page Life Cycle events that we will use most frequently.

1. PreInit
2. Init
3. InitComplete
4. PreLoad
5. Load

-Control Events (All Postbackand Cached Events)

6. LoadComplete
7. PreRender
8. PreRenderComplete
9. SaveStateComplete

-RenderMethod Calling

10. UnLoad

1. **PreInit**: Raises after the start stage is completed and before the initialization stage begins. We use this event forperforming the following actions:

I. Check the IsPostBack property to determine whether this is the first time page is being processed.
II. The IsCallback and IsCrossPagePostBack properties are also set at this time.
III. Create or re-create Custom Controls.
IV. Set a Master Page dynamically.
V. Set the Theme property dynamically.

2. **Init**: Raises after all controls have been initialized and any skin settings have been applied. The Init event of individual controls occurs before the Init event of page. Use this event to read or initialize control properties.

3. **InitComplete**: Raised at the end of page's initialization stage. Only one operation takes place between Init and InitComplete events i.e. tracking of view state is turned on. View state tracking enables controls to persist any values that are programmatically added to the ViewState collection. Until view state tracking is turned on, any values added to view state are lost across postbacks. Controls typically turn on view state tracking immediately after they raise their Init event. Use this event to make any changes to view state that we want to make sure are persisted after the next postback.

4. **PreLoad**: Raised after the page loads view state for itself and all controls, and after it starts processing postback data that is included with the Request object.

5. **Load**: Page object calls the OnLoad method on Page, and then recursively does the same for each child control until all controls are loaded. The Load event of individual controls occurs after the Load event of page. Use this event to get or set properties of controls and to establish database connections.

Control events: Use these events to handle specific control events such as a Button control's Click event or a TextBox control's TextChanged event. In a postback request, if the page contains validation controls, check the IsValid property of Page before performing any processing.

6. **LoadComplete**: Raised at the end of event-handling stage and use this event for performing any tasks that require for all the controls on page that have been loaded.

7. **PreRender**: Raises after Page object has created all controls that are required in order to render the page, including child controls of Custom Controls. Page object raises this PreRender event on itself and then recursively does the same for each child control. Use this event to make any final changes to contents of page or its controls because hereafter rendering stage begins.

8. **PreRenderComplete**: Raises after all controls on page are ready for render.

9. **SaveStateComplete**: Raises after view state and control state have been saved for page and for all controls. Any changes to the page or controls at this point that affects rendering will not be retrieved on the next postback.

Render: This is not an event but a method and at this stage of processing, Page calls this method on each control.

Note: All ASP.NET Web Server Controls has a RenderControl method which writes the control's markup to “Output Stream” which has to be sent to the browser as output.

10. **Unload**: Raised for each control and then for page. Use this event to do any final cleanup such as closing control-specific database connections.

Note: During unload stage - page and its controls rendering is completed, so you cannot make any further changes to the response stream and if you attempt to call a method such as Response.Write will throw an exception.

To test and understand about PageEvents add a new WebForm under the project naming it as “PageEvents.aspx”, place a button on it and set the Text as “Post Back”. Now go to “aspx.cs” file, copy the method “Page_Load” present in the class and paste it for 9 more times in the same class and rename the 9 methods as following: “Page_PreInit”, “Page_Init”, “Page_InitComplete”, “Page_PreLoad”, “Page_LoadComplete”, “Page_PreRender”, “Page_PreRenderComplete”, “Page_SaveStateComplete” and “Page_Unload”, and write a statement in each method describing about the event as following:

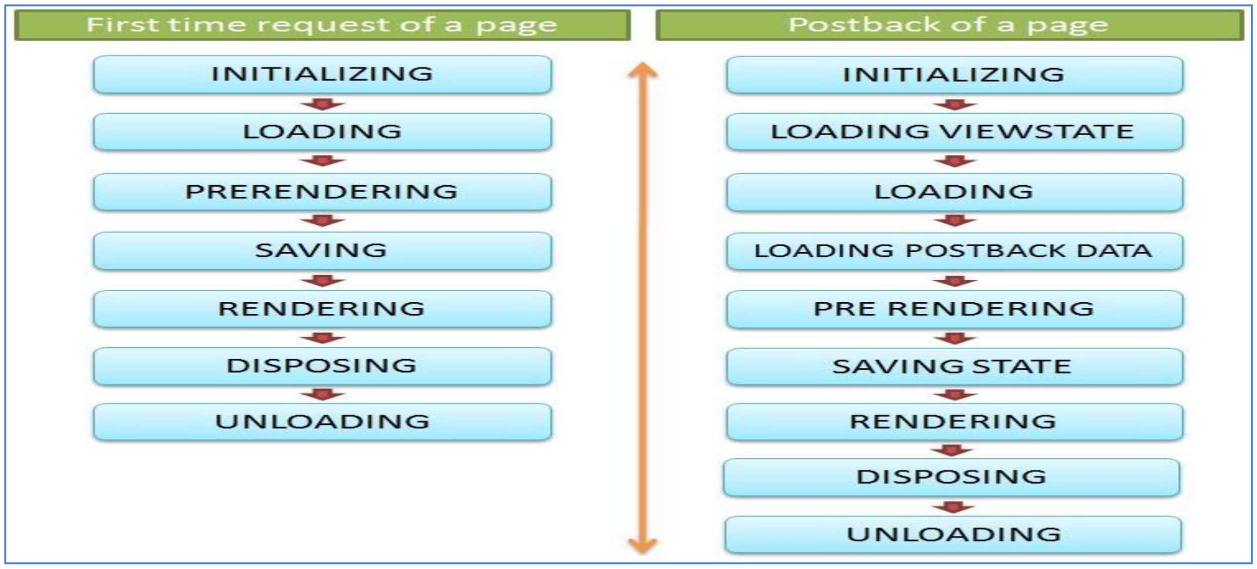
Code under Page_PreInit: Response.Write("Pre-Init Event Fired. <br />");
Code under Page_Init: Response.Write("Init Event Fired. <br />");
Code under Page_InitComplete: Response.Write("InitComplete Event Fired. <br />");
Code under Page_PreLoad: Response.Write("PreLoad Event Fired. <br />");
Code under Page_Load: Response.Write("Load Event Fired. <br />");
Code under Page_LoadComplete: Response.Write("LoadComplete Event Fired. <br />");
Code under Page_PreRender: Response.Write("PreRender Event Fired. <br />");
Code under Page_PreRenderComplete: Response.Write("PreRenderComplete Event Fired. <br />");
Code under Page_SaveStateComplete: Response.Write("SaveStateComplete Event Fired. <br />");
Code under Page_Unload: //Response.Write("Unload Event Fired. <br />");

Now generate a Click Event Handler for Button and write the following code in it:

```
Response.Write("<font color='red'>");
Response.Write("Button Click Event Fired. <br />");
Response.Write("</font>");
```

Now run the WebForm and watch the output, we notice all the events getting fired in a sequence, now click on the button to perform a Postback which will also fire Click Event along with all the other Events whereas click event will fire after the Load event and before the LoadComplete event.

Series of actions that are performed between the first request and postback request of a page



AutoEventWireup attribute of Page Directive: ASP.NET supports an automatic way to associate Page Events to its Event Handlers if the AutoEventWireup attribute of the Page directive is set to true (default is also true only), provided Event Handler names follow a naming convention i.e "Page_<Event Name>" for example "Page_Init", "Page_Load", whereas if we set the attribute value as false or we don't follow the naming convention then Event Handlers should be explicitly bound to their corresponding events under Constructor of the page as following:

```
public PageEvents() {
  this.PreInit += Page_PreInit;
  this.Init += Page_Init;
  this.InitComplete += Page_InitComplete;
  this.PreLoad += Page_PreLoad;
  this.Load += Page_Load;
  this.LoadComplete += Page_LoadComplete;
  this.PreRender += Page_PreRender;
  this.PreRenderComplete += Page_PreRenderComplete;
  this.SaveStateComplete += Page_SaveStateComplete;
  this.Unload += Page_Unload;
}
```

Tracing

ASP.NET tracing enables you to view diagnostic information about a single request for an ASP.NET page. ASP.NET tracing enables you to follow a page's execution path, display diagnostic information at run time, and debug your application. We can enable tracing in an ASP.Net application in 2 different levels:

1. Page Level Tracing
2. Application Level Tracing

Page Level Tracing:

If tracing is enabled page level, whenever a page is requested, ASP.NET appends to the page a series of tables containing execution details about the page request. Tracing is disabled by default in an ASP.NET application and to enable it add Trace attribute to Page directive with the value true as following:

<%@ Page Trace="true" Language="C#" ... %>

To test this, add the Trace attribute to "PageEvents.aspx" and run the page, and we will see the trace data at bottom of the page.

Application Level Tracing:

Instead of enabling tracing for individual pages, you can enable it for your entire application and in that case every page in our application will perform tracing. Application tracing is useful in realtime environments because you can easily enable it and disable it without editing individual pages. When tracing is enabled application level it will not display trace information on the page but we need to view trace information with the help of a trace viewer. To enable Application Level Tracing first delete your Page Level Tracing, now open the Web.config file and add the following information to it under <system.web> tag:

```
<trace enabled="true" />
```

Now run the application and notice in the page we will not see any trace data at bottom of the page and to view trace data we need to use the trace viewer i.e. "Trace.axd" and to do that copy the URL of your current executing page, open a new tab in the browser, paste the URL in the new tab and there delete the ".aspx" file name and write trace.axd, for example:

http://localhost:port/PageEvents.aspx => http://localhost:port/trace.axd

Attributes of <trace> element in Web.config:

1. **Enabled:** set it true to enable tracing for the application becase default is false.

   ```
   <trace enabled="true" />
   ```

2. **PageOutput:** set it true to display trace data both in the page and in the trace viewer also, default is false.

   ```
   <trace enabled="true" pageOutput="true" />
   ```

3. **RequestLimit:** when you enable tracing for an application, ASP.NET collects trace information for each request to that application up to the maximum number of requests you specify and the default number of requests is 10.

   ```
   <trace enabled="true" pageOutput="true" requestLimit="20" />
   ```

4. **MostRecent:** by default, when the trace viewer reaches its request limit, the application stops recording trace requests. However, you can configure the MostRecent attribute to true (default is false) which will discard the old data when the maximum number of requests are reached and adds new entries.

   ```
   <trace enabled="true" pageOutput="true" requestLimit="20" mostRecent="true" />
   ```

5. **LocalOnly:** set it true to make the trace viewer (trace.axd) available only on local Web server and default is also true, where as if set as false we can access trace data from remote or client machines also.

   ```
   <trace enabled="true" pageOutput="true" requestLimit="20" mostRecent="true" localOnly="false" />
   ```

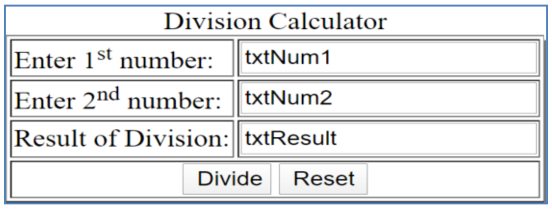
Writing Custom Message into Trace: You can append custom trace information to trace in an ASP.NET page or to the trace log. You can write trace information by using the TraceContext class's Warn or Write methods and the difference between the two methods is that a message written with the Warn method appears in red color and Write method in black color. To test this write the following statements in the button click Event Handler of PageEvents.aspx.cs file:

```
Trace.Write("Writing data into trace using Write method.");
Trace.Warn("Writing data into trace using Warn method.");
```

Exception Handling in Web Applications

Whenever a runtime error occurs in a program, CLR will internally create an instance of Exception class which is matching with the Exception that got occurred and throws that instance, which will now cause abnormal termination of the program and displays an error message describing the reason for termination.

When any runtime error occurs in a WebPage immediately details of that error will be show to end users on the browser screen and to check that add a new WebForm under project naming it as "ExceptionHandling.aspx" and design it as following:



Division Calculator	
Enter 1st number:	txtNum1
Enter 2nd number:	txtNum2
Result of Division:	txtResult
Divide Reset	

Now go to “aspx.cs” file and write the following code:

<u>Code under Page_Load:</u>

```
if (!IsPostBack) {
  txtNum1.Focus();
}
```

<u>Code under Divide Button Click:</u>

```
int num1 = int.Parse(txtNum1.Text);
int num2 = int.Parse(txtNum2.Text);
int result = num1 / num2;
txtResult.Text = result.ToString();
```

<u>Code under Reset Button Click:</u>

```
txtNum1.Text = txtNum2.Text = txtResult.Text = "";
txtNum1.Focus();
```

Now run the Web Page with out debugging i.e. Ctrl + F5 and check the output and here we have a chance of getting 3 types of Exceptions like FormatException, DivideByZeroException and OverflowException when we enter wrong values in the TextBox’s and when ever we get the Exception it will abnormally terminate the program and shows an “Yellow Screen” displaying the details of Exception. The “Yellow Screen” which is shown has a problem i.e. it will display the source code of the line where we got error and along with that it also displays 2 lines of code before and 2 lines of code after the error line and the problem here is if those lines contain any complex logic, end users can view that logic and to avoid displaying that “Yellow Screen” to users we are provided with 3 Error Handling options in ASP.Net, those are:

1. Block Level Error Handling
2. Page Level Error Handling
3. Application Level Error Handling

<u>Block Level Error Handling:</u> this is performed by using the very traditional structured error handling technique i.e. try and catch blocks and when code is enclosed under these blocks exceptions get handled, stopping abnormal termination. To handle error we need to use try and catch blocks as following:

```
try {
  -Statements which will cause any runtime errors.
  -Statements which should not execute when the error got occurred.
}
catch(<exception class name><var>) {
  -Statements which should execute only when the error got occurred.
}
```

To test "block level error handling" re-write the code under the divide button of our previous page as following:

```
try {
  int num1 = int.Parse(txtNum1.Text);
  int num2 = int.Parse(txtNum2.Text);
  int result = num1 / num2;
  txtResult.Text = result.ToString();
}
catch(Exception ex) {
  Response.Redirect("ErrorPage.aspx?ErrorMessage=" + ex.Message);
}
```

Now add a new WebForm under the project naming it as "ErrorPage.aspx" and write the following code under its <div> tag:

```
<h1 style="background-color:yellowgreen;color:orangered;text-align:center">Error Page</h1>
<font size="4">There is an error in the application and the details of the error are:
<asp:Label ID="lblMsg" runat="server" ForeColor="Red" />
</font>
```

Now go to "ErrorPage.aspx.cs" file and write the following code under Page_Load method:

```
lblMsg.Text = Request.QueryString["ErrorMessage"];
```

After doing all the above, run "ExceptionDemo.aspx" page again and now when ever any Exception occurs in the page with out displaying the "Yellow Screen" it will redirect to "ErrorPage.aspx" and displays the error details on that page.

Note: The drawback of "Block Level Error Handling" is we need to add these try and catch blocks under each and every method where there is a chance of getting any exception and it is a tedious and lengthy process, so to overcome this problem use "Page Level Error Handling".

Page Level Error Handling: this is a process of handling errors that occur in the whole page by using an Event Handler method i.e. "Page_Error" and by implementing this method under the class any error that occur in the page can be handled. When this method is implemented under our page class, where ever the error occurs in the page, control will directly transfer to this method, so we need to implement logic for handling errors under this method.

To test using the above process, first delete try and catch blocks we have added to the logic under Divide button click event handler and add the following event handler method to the class:

```
protected void Page_Error(object sender, EventArgs e) {
  Exception ex = Server.GetLastError();
  Server.ClearError();
  Response.Redirect("ErrorPage.aspx?ErrorMessage=" + ex.Message);
}
```

Now run the Web Page again and watch the output which will be same as previous but the only difference is here we can handle all errors of a page with single method with out writing multiple try and catch blocks.

Application Level Error Handling: in page level error handling we need to implement a separate Page_Error method for each Web Page, so when we have multiple Web Pages the process becomes tedious and lengthy. To avoid this problem we are provided with Application Level Error Handling which can be performed by implementing an Event Handler method known as Application_Error.

Application_Error Event Handler is capable of handling exceptions that occur under any page of the site and this method should be implemented under a special class known as "Global". We find this class under our project in a file "Global.asax" and this file will be present under every Web Application project by default. To view the Global class open the Global.asax file present under our Web Application project and there we find the Global class inheriting from a pre-defined class "HttpApplication" of System.Web namespace and by default the class contains an Event Handler method with the name Application_Start.

To test the process of application level error handling, first comment the "Page_Error" Event Handler method we defined earlier, now open Global.asax file, rename the method "Application_Start" present in the class as "Application_Error" and write the following code in it:

```
Exception ex = Server.GetLastError(); Server.ClearError();
if(ex.InnerException != null) {
  Response.Redirect("ErrorPage.aspx?ErrorMessage=" + ex.InnerException.Message);
}
else {
  Response.Redirect("ErrorPage.aspx?ErrorMessage=" + ex.Message);
}
```

In the above case when an error occurs in a Web Page and was not handled over there, server will raise another Exception there i.e. "HttpUnhandledException" and then control is transferred to Application_Error event handler, so here "ex.Message" returns the error message associated with HttpUnhandledException class but not of the actual fired Exception, so to get the error message of the actual fired Exception we need to use the statement "ex.InnerException.Message".

Sending mails to "Customer Support" team when an Exception occurs: in the above Web Page when any Exception occured we are redirecting to "ErrorPage" and displaying the error message over there, but some errors can't be fixed by the users directly and may require the "Technical Team" to fix those errors and in such cases our application should report those errors to the "Customer Support" of the site or directly to "Technical Team" and to do that an mail should be generated by the Web Page with all the error details and send to them.

Displaying data present under Products.xml of our project in a GridView control on a Web Page:

Add a new WebForm naming it as "DisplayProducts.aspx" and the write the following code under <div> tag:

```
<asp:GridView ID="GridView1" runat="server" />
```

Now go to “DisplayProducts.aspx.cs” file and write the following code under Page_Load Method by importing System.Data namespace.

```
string Physicalpath = Server.MapPath("~/Products.xml");
DataSet ds = new DataSet(); ds.ReadXml(Physicalpath);
GridView1.DataSource = ds; GridView1.DataBind();
```

Now run the Web Page and watch the output, which displays the Products data in a table format. To do that DataSet will first load data from the specified Xml File but in this case if it could not locate the Xml File in the specified location then “FileNotFoundException” will occur and jumps to our “Application_Error” event handler method we have implemented earlier and from there it jumps to “ErrorPage.aspx” to display the “Error Message” to end users. To test this process go to the physical location where “Products.xml” is present, move the file to another location and run the Web Form again and watch the output. The drawback in this approach is “Error Page” will display all the details of error to end users along with the path of “Xml File” which is missing and this information is no way required to end users as he can’t fix the problem. To overcome this problem the information has to be sent to “Customer Support” or “Techinal Team” so that they can resolve the problem and we can do that by generating a mail and send them.

To send a mail lets write the code under a class and use it where ever it is required and to do that add a new class under the project naming it as MailSystem.cs and write the following code in it:

```
using System.Net; using System.Text; using System.Net.Mail;
public class MailSystem {
  public static void SendMail(Exception ex, string PageName) {
    StringBuilder mailBody = new StringBuilder();
    mailBody.Append("Hello Team,<br /><br />");
    mailBody.Append("Client is facing an error in output of the page: " + PageName + ", and details of error are:");
    mailBody.Append("<br /><br /><font color='red' size='4'>");
    if(ex.InnerException != null) {
      msgBody.Append(ex.InnerException.GetType() + ": " + ex.InnerException.Message);
    }
    else {
      msgBody.Append(ex.GetType() + ": " + ex.Message);
    }
    mailBody.Append("</font>");
    mailBody.Append("<br /><br />Please look into the problem and try to resolve it ASAP.<br /><br />");
    mailBody.Append("Regards<br ><br />Customer Support.");

    MailMessage mailMsg = new MailMessage("<write sender mail id here>", "<write receiver mail id here>");
    mailMsg.IsBodyHtml = true; mailMsg.Subject = "Error Mail"; mailMsg.Body = mailBody.ToString();

    SmtpClient mailSender = new SmtpClient("smtp.gmail.com", 587);
    mailSender.Credentials = new NetworkCredential("<write sender mail id here>", "<write sender pwd here>");
    mailSender.EnableSsl = true; mailSender.Send(mailMsg);
  }
}
```

Now add a new WebForm naming it as “ErrorPageWithMail.aspx” and write the following code under its <div> tag:

```
<h1 style="background-color: yellowgreen; color: orangered; text-align: center">Error Page</h1>
<p>There is an error in the application and our technical team is looking into the error and we will fix it as early as
        possible, so please try after some time.</p>
<h4>Any queries please feel free to contact our customer support.</h4>
```

Now go to DisplayProducts.aspx.cs file and write the following code under the class:

```
protected void Page_Error(object sender, EventArgs e) {
  Exception ex = Server.GetLastError(); Server.ClearError();
  MailSystem.SendMail(ex, "DisplayProducts.aspx"); Response.Redirect("ErrorPageWithMail.aspx");
}
```

After doing all the above when ever an exception occurs in the page, control jumps to Page_Error Event Handler, sends an error mail to Customer Support and redirects to “ErrorPageWithMail.aspx”.

Storing Mail Settings in Web.config file: in the above case to send a mail we have entered all the details for sending the mail like Sender Id, Password, Smtp Host and Port No. in our class, but if in any case if those details change there will not be any chance of modification after hosting the site because source code will not be available, so to overcome that problem we need to store all those values in Web.config file so that SmtpClient class will read those details and sends the mail. To test this process open Web.config file and write the following code under the tag <configuration></configuration> as following:

```
<system.net>
  <mailSettings>
    <smtp deliveryMethod="Network">
      <network host="smtp.gmail.com" port="587" enableSsl="true" userName="enter your mail id here"
            password="enter your pwd here" />
    </smtp>
  </mailSettings>
</system.net>
```

Now go to MailSystem class and replace the last 4 lines of code in SendMail method with the below 2 lines:

```
SmtpClient mailSender = new SmtpClient();
mailSender.Send(mailMsg);
```

Note: to send mails from your gmail account first you need to go to “Manager your Account Settings” => Select Security Tab => scroll down to “Signing in to Google” => under that set the “2-Step Verification” value to “Off”, scroll down to “Allow less secure apps” and set it as “ON”.

Custom Errors:

This is another process of handling errors in an Asp.Net Web Page apart from the 3 options we have learnt above. In this process we handle errors in the Web Page thru their “Http Status Codes” using Web.config file. Every action we perform in a Web Application returns some result, which is informed to the client thru a code known as Http Status Codes.

Status Code	***Description***
200	Ok => Success
400	Bad Request => The request cannot be fulfilled due to bad syntax
403	Forbidden => The request was a legal request, but the server is refusing to respond to it
404	Not Found => Page Not Found
405	Method not allowed => Request made to page using a request method not supported by page
408	Timeout => Server time out
500	Internal Server Error => Exception

Note: Custom Errors work only when we don't have block level or page level or application level error handling.

To test Custom Errors add a new Web Page under the project naming it as "CustomErrors.aspx" and write the following code under <div> tag:

```
<h1 style="background-color:yellowgreen;color:orangered;text-align:center">Custom Error Page</h1>
<p style="text-align:justify;text-indent:50px">There is an un-fortunate error in the application and we are aware of
that problem. Our technical team is looking into the problem, so please try after some time.</p>
<h4>Any queries please feel free to contact our customer support.</h4>
```

Now open Web.config file and write the following code under <system.web> tag:

```
<customErrors mode="On" defaultRedirect="CustomErrors.aspx" />
```

Now if we get any error in the application of any "Http Status Code" then it will redirect to "CustomErrors.aspx" page whereas if we want to redirect to a different page based on the status codes i.e. if any Internal Server Error with Status Code 500 then redirecting to 1 page, Page Not Found with Status Code 404 then redirecting to 1 page can also be performed by creating different Error Pages and specifying that information in Web.config file.

To test the process, add 2 new WebForms under the Project naming them as PageNotFound.aspx and InternalServerError.aspx and write the following code under their <div> tag:

PageNotFound.aspx:

```
<h1 style="background-color:yellowgreen;color:orangered;text-align:center">Page Not Found Error</h1>
<p style="text-align:justify">Either the requested page is not available or moved to a different location.</p>
```

InternalServerError.aspx

```
<h1 style="background-color:yellowgreen;color:orangered;text-align:center">Internal Server Error</h1>
<p style="text-align:justify">There is an internal server error (Exception) occured in the requested page, so please
contact customer support team for any additional help.</p>
```

Now re-open Web.config file and re-write the *<customErrors>* elements under <system.web> tag as following:

```
<customErrors mode="On" defaultRedirect="ErrorPage.aspx">
  <error statusCode="404" redirect="PageNotFound.aspx"/>
  <error statusCode="500" redirect="InternalServerError.aspx"/>
</customErrors>
```

Now for all Internal Server Errors (Exception) it will redirect to “InternalServerError.aspx”, in case requested Page is not found it will redirect to “PageNotFound.aspx” and for rest of all other errors it will redirect to “ErrorPage.aspx”.

Mode Attribute of CustomErrors element: mode attribute of CustomErrors element can be set with 3 different values: On, Off and RemoteOnly (default value).
On => users will receive custom error pages with out displaying detailed error details i.e. “Yellow screen of death”.
Off => Detailed error details will be shown to the users i.e. “Yellow screen of death”, but not custom error pages.
RemoteOnly => Detailed errors are shown to local users and remote users will receive custom error pages.

Globalization and Localization

Globalization is an approach of developing an application neutral of a culture and Localization is an approach of translating a globalized application to any particular culture.

Culture: It's a set of standards followed by a country regarding Date and Time Format, Calendar System, Currency, Number System, Measuring System, Sorting Order and Language. Every country follows or adopts a different culture and representing the culture of a country there was a culture code or culture name which will be as following: en-US, en-GB, fr-FR, ja-JP, hi-IN, te-IN, and ta-IN.

Note: in culture name the first 2 characters in lower case represents the language and the next 2 characters in upper case represents the country.

By default every applications runs in American culture i.e. "en-US" and to test this add a new WebForm naming it as "TestCulture.aspx" and write the following code under <div> tag:

```
<asp:Label ID="lblDate" runat="server" Text="Select Date: "/>
<asp:TextBox ID="txtDate" runat="server" />
<asp:ImageButton ID="imgDate" runat="server" ImageUrl="~/Icons/calendar.png" />
<asp:Calendar ID="cldDate" runat="server" Visible="false" />
```

Now go to aspx.cs file and write the following code in it:

Code under Image Button Click:
```
if (cldDate.Visible) { cldDate.Visible = false; }
else { cldDate.Visible = true; }
```

Code Under Calender Selection Changed:
```
txtDate.Text = cldDate.SelectedDate.ToShortDateString(); cldDate.Visible = false;
```

Now when we run the application and select a date from Calendar then selected date is displayed in TextBox, but if we notice the format of date is "mm/dd/yyyy" and also calendar control displays month names and week names in English which proves that our application is running in "American-English" culture.

We can change the culture of our Web Application by setting Culture attribute to the Page directive with the Culture Name we want as following:

```
<%@ page Culture="hi-IN" .... %>
```

Now when we run the application selected date will be in dd/mm/yyyy format and also calendar control displays month names and week names in hindi because we set the culture as “India-Hindi”, but the draw back in this approach is, the culture we set right now applies to all users accessing the application from any where, whereas if we want to set the culture differently for each user i.e. based on his browser preferred language settings, then set the culture in page directive as following:

```
<%@ page Culture="Auto" ... %>
```

In the above case based on browser preferred language settings culture will vary for each user and to test this change the preferred language settings of your browser as following:

Chrome: go to Customize option => select Settings => scroll down and click on Advanced => scroll down to Languages and click on Language => click on “Add Languages” button and choose a language from the list of languages and click Add => now click on “More Actions” buttons beside the language and select the option “Move to the top” so, that language will become the preferred language of browser.

How the Culture of an application changes based on Culture attribute of Page Directive?
Ans: In our FCL under the System.Globalization namespace we are provided with a class known as CultureInfo and this class contains the information of all the available cultures in the industry and their culture standards. When the instance of CultureInfo class is created by passing CultureName as "Constructor Parameter" will take the responsibility of changing the applications culture.

When we specify the culture attribute for page directive with any static value like “hi-IN” or “fr-FR” or “de-DE” then internally instance of CultureInfo class is created by taking the culture name as a parameter to its constructor, so that culture of our Thread changes, whereas when the culture attribute value is set as “Auto” then server will first read default language settings of browser and creates the instance of CultureInfo class with the Culture Name it has read and changes the Thread culture.

Thread: this is a unit of execution which is responsible to serve a client request i.e. whenever a client sends his request to server, server will create an instance of Thread class and gives that Thread to client for serving his requests. So if we have “n” users connected to the server there will also be “n” Threads created.

Note: CultureInfo class is responsible in changing all the attributes of culture like Date and Time Format, Calendar System, Number System, Currency, Measuring System and Sorting Order but not Language and this is the reason why in our page Calendar is changing it's culture, but the text “Select Date” we have used for “Label” is not changing i.e. it always showsText in English only, because language translations are never performed implicitly because we are not provided with any proper language translation tools in the industry, so as a developer it is purely our responsibility to perform language translations for each and every language we wanted to support.

Developing a Multi-Lang Registration Form: Add a new WebForm naming it as “MultiLangRegistrationForm.aspx” and write the following code in <div> tag:

```
<table align="center">
  <tr>
    <td colspan="2" align="center"><asp:Label ID="lblTitle" runat="server" Text="Registration Form" /></td>
  </tr>
  <tr>
    <td colspan="2" align="center">
    <asp:DropDownList ID="ddlLang" runat="server" AutoPostBack="True">
      <asp:ListItem Text="English" Value="en" />
      <asp:ListItem Text="français" Value="fr" />
      <asp:ListItem Text="हिंदी" Value="hi" />
      <asp:ListItem Text="తెలుగు" Value="te" />
    </asp:DropDownList>
   </td>
  </tr>
  <tr>
   <td><asp:Label ID="lblUser" runat="server" Text="User Id:" /></td>
   <td><asp:TextBox ID="txtUser" runat="server" /></td>
  </tr>
  <tr>
   <td><asp:Label ID="lblPwd" runat="server" Text="Password:" /></td>
   <td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" /></td>
  </tr>
  <tr>
   <td><asp:Label ID="lblName" runat="server" Text="Name:" /></td>
   <td><asp:TextBox ID="txtName" runat="server" /></td>
  </tr>
  <tr>
   <td><asp:Label ID="lblPhone" runat="server" Text="Phone No.:" /></td>
   <td><asp:TextBox ID="txtPhone" runat="server" /></td>
  </tr>
  <tr>
   <td><asp:Label ID="lblEmail" runat="server" Text="Email Id:" /></td>
   <td><asp:TextBox ID="txtEmail" runat="server" TextMode="Email" /></td>
  </tr>
  <tr>
   <td><asp:Label ID="lblAddress" runat="server" Text="Address:" /></td>
   <td><asp:TextBox ID="txtAddress" runat="server" TextMode="MultiLine"/></td>
  </tr>
  <tr>
   <td align="center" colspan="2"><asp:Button ID="btnRegister" runat="server" Text="Register"  />
                 <asp:Button ID="btnReset" runat="server" Text="Reset" /></td>
  </tr>
</table>
```

In the above case we wanted to support 4 different languages for our registration form and to do that we need to enter "Text" for labels and buttons in 4 languages explicitly and that should be done under a special file known as "Resource File".

Resource files are used for storing information specific to each and every language we wanted to support in a "Name/Value" pair. We need to create these resource files seperate for each language that we wanted to support and store information specific to that language. Resource files are saved with ".resx" extension and while naming resource files we need to follow a pattern i.e. "<Base Name>.<Culture>.resx" where "<Base Name>" should be same for all languages and "<Culture>" varies for each language.

By default Visual Studio provides support for generating a resource file for English language with the existing “Text” of controls, and based on that we can generate resource files for remaining other languages also.To generate resource file for English go to design view of the “.aspx” page, go to Tools Menu and select the option “Generate Local Resource” and this action will perform the following:

1. Adds a special Asp.Net Folder with the name “App_LocalResources” under the project and under this folder we find a resource file that is created for english language with the name “MultLangRegistrationForm.aspx.resx” and here “.resx” in the file extension, “MultiLangRegistrationForm.aspx” is the "<Base Name>" and for english language “<Culture>” is optional.

2. For every aspx controls and page directive it adds a key for referring to that element in the resource file which will be in the following pattern:

```
meta:resourcekey="PageResource1"        => For Page Directive
meta:resourcekey="lblTitleResource1"    => For Title Label
meta:resourcekey="ddlLangResource1"     => For DropDownList
```

Note: in the above case PageResource1, lblTitleResource1 and ddlLangResource1 are the keys that are used in the resource file for referring to that control.

3. For Page directive we will also find “Culture & UICulture” attributes with a value “Auto” for changing the Culture and UICulture of the page based on user’s preffered language settings.

4. Now if we open the Resource File and observe we notice keys referring to controls and with that key it will refer to Text and ToolTip of that control and for Page it will refer to Title.

Note: Currently resouce file is having keys referring to all controls so delete entries for DropDownList, ListItems and TextBox controls also (both Text and ToolTip) because we require keys for Labels and Buttons only but those keys refers to Text and ToolTip of each control, but we require only Text entries so delete all ToolTip entries and finally we need only 10 entries to be present i.e. Text of 7 Labels, 2 buttons and Title of Page.

Creating Resource Files for other Languages: open solution explorer, right click on the existing resource file, select copy, right click on “App_LocalResources” folder and select paste; repeat the same process for 2 more times to generate 4 resource files totally under the folder and rename the 3 new resource files as following:

- MultiLangRegistrationForm.aspx.fr.resx
- MultiLangRegistrationForm.aspx.hi.resx
- MultiLangRegistrationForm.aspx.te.resx

Now open each resource file, copy content under “Value” which is present in English, use google translator, translate values from English into French, Hindi and Telugu languages repectively, and paste those translated values in place of English values under the corresponding resource files so that each resource file should be as following:

MultiLangRegistrationForm.aspx.resx

Name	***Value***
lblTitleResource1	*Registration Form*
lblUserResource1	*User Id:*
lblPwdResource1	*Password:*

lblNameResource1	*Name:*
lblPhoneResource1	*Phone No:*
lblEmailResource1	*Email Id:*
lblAddressResource1	*Address:*
btnRegisterResource1	*Register*
btnResetResource1	*Reset*
PageTitleResource1	*Registration Form - English*

MultiLangRegistrationForm.aspx.fr.resx

Name	***Value***
lblTitleResource1	Formulaire d'inscription
lblUserResource1	Identifiant d'utilisateur:
lblPwdResource1	Mot de passe:
lblNameResource1	Prénom:
lblPhoneResource1	Pas de téléphone:
lblEmailResource1	Identifiant de messagerie:
lblAddressResource1	Adresse:
btnRegisterResource1	registre
btnResetResource1	Réinitialiser
PageTitleResource1	Formulaire d'inscription – français

MultiLangRegistrationForm.aspx.hi.resx

Name	***Value***
lblTitleResource1	पंजीकरणफॉर्म
lblUserResource1	यूज़रआईडी:
lblPwdResource1	पारणशब्द:
lblNameResource1	नाम:
lblPhoneResource1	फोननंबर:
lblEmailResource1	ईमेलआईडी:
lblAddressResource1	पता:
btnRegisterResource1	रजिस्टर
btnResetResource1	रीसेट
PageTitleResource1	पंजीकरणफॉर्म – हिंदी

MultiLangRegistrationForm.aspx.te.resx

Name	***Value***
lblTitleResource1	సభ్యత్వనమోదుపత్రం
lblUserResource1	వినియోగదారునిగుర్తింపు:
lblPwdResource1	పాస్వర్డ్:
lblNameResource1	పేరు:
lblPhoneResource1	చరవాణి:
lblEmailResource1	ఇమెయిల్:
lblAddressResource1	చిరునామా:
btnRegisterResource1	నమోదు
btnResetResource1	రీసెట్
PageTitleResource1	సభ్యత్వనమోదుపత్రం – తెలుగు

Note: once we do the above automatically our page language changes based on browser preferred language settings and to test that run the WebPage and watch the output by changing the preferred language under browser.

Even if the text of Labels and Buttons changes based on browser's preferred language settings, the language in DropDownList will be still showing the first item in the list i.e. English only and if we want this to be set according to the browser preferred language, then write the following code under "Page_Load" Event Handler by reading "Http_Accept_Language" from Server VariablesCollection as following:

```
if (!IsPostBack) {
  string PreferredLang = Request.ServerVariables["Http_Accept_Language"].Split(',')[0];
  if (PreferredLang.Length > 2) { ddlLang.SelectedValue = PreferredLang.Substring(0, 2); }
  else { ddlLang.SelectedValue = PreferredLang; }
}
```

Now if we want to change the "Text" based on DropDownList selection then write the following code in "aspx.cs" file:

```
using System.Threading; using System.Globalization;
protected override void InitializeCulture() {
  string CultureName = Request.Form["ddlLang"];
  if (CultureName != null) {
    CultureInfo ci = new CultureInfo(CultureName);
    Thread.CurrentThread.CurrentCulture = ci;
    Thread.CurrentThread.CurrentUICulture = ci;
  }
}
```

The InitializeCulture method of Page class contains no logic in it and developers extending the functionality of the Page class can override the InitializeCulture method to set the Culture and UICulture information for a Thread associated with the client.

CurrentThread is a static property of Thread class which returns the reference of Thread associated with current client because every client request is handled by a seperate Thread. CurrentCulture property of Thread class gets or sets the culture for the current thread and CurrentUICulture property gets or sets the current culture used by the ResourceManager to look up culture-specific resources at run time.

State Management

Web Applications are stateless i.e. they will never maintain the values of 1 request processing under the next request processing of the same page or other pages, but sometimes we need the values of 1 request processing under the next request processing of same page or other pages also and to overcome the above problem and maintain the state of values between multiple requests of the same page or between different pages we are provided with the concept of State Management. In ASP.Net to maintain the state of values we are provided with various techniques like:

1. View State
2. Query Strings
3. Hidden Fields
4. Cookies
5. Session
6. Application

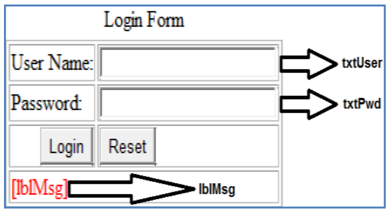
To understand State Management first open a new ASP.NET Web Application project, name it as "ASPStateMgmt", specify the location to save as our personal folder, click ok and in the window opened choose "Empty" Project Template, check Web Forms "CheckBox" and click "Ok". Now let's host the project under Local IIS and to do that open Solution Explorer and under the project - double click on Properties node which opens the Project Property window, click on the "Web" in LHS and now on the right, under "Servers" choose "Local IIS" option in the "DropDownList", click on "Create Virtual Directory" button and click on "Save" icon in VS Toolbar.

Option 1: ViewState

View State is a mechanism to preserve the values of the Page and Controls between Postback's. It is a Page-Level State Management technique i.e. it maintains the state of values for a single user between multiple requests of 1 page only (Single User Local Data).

Creating a Login Page with 3 failure attempts:

Add 3 WebForms underproject naming them as "LoginWithFailureCount.aspx", "SuccessWithFailureCount.aspx" and "FailureWithFailureCount.aspx", and design "LoginWithFailureCount.aspx" as following:



Now goto LoginWithFailureCount.aspx.cs file and write the following code:

Code under Page_Load:

```
if(!IsPostBack){
  txtUser.Focus();
  ViewState["FailureCount"] = 0;
}
```

Code under Login Button Click:

```
if (txtUser.Text == "admin" && txtPwd.Text == "admin") {
  Response.Redirect("SuccessWithFailureCount.aspx?Name=" + txtUser.Text);
}
else {
  int Count = (int)ViewState["FailureCount"] + 1;
  if (Count == 3) {
    Response.Redirect("FailureWithFailureCount.aspx?Name=" + txtUser.Text + "&Count=" + Count);
  }
  ViewState["FailureCount"] = Count;
  lblMsg.Text = Count + " attempt(s) failed, maximum are 3.";
```

Code under Reset Button Click:

```
txtUser.Text = txtPwd.Text = ""; txtUser.Focus();
```

Now go to SuccessWithFailureCount.aspx.cs file and write the following code:

Code under Page_Load:

```
string Name = Request["Name"];
Response.Write("Hello " + Name + ", welcome to the site.");
```

Now go to FailureWithFailureCount.aspx.cs file and write the following code:

Code under Page_Load:

```
string Name = Request["Name"];
int Count = int.Parse(Request["Count"]);
Response.Write("Hello " + Name + ", you have failed all the" + Count + " attempts to login.");
```

Now run the LoginForm, enter valid credentials and click on Login button which will redirect to Success page and if the credentials are wrong it will give a chance for another 2 times to correct the values and if at all the 3 attempts fail then it will redirect to Failure Page.

In our above code to maintain the FailureCount we have used ViewState because Web Applications are Stateless and ViewState is used for maintaining the state of values, which holds values of a page between multiple requests of that page for a single user i.e. "Single User Local Data".

Where does server store ViewState Values?

Server stores ViewState values on the same page only i.e. while rendering the output of a page it writes ViewState values into that rendered output and sends them to the browser, so we can view those values using "View Page Source" option of the browser and there we find a "Hidden Field" with the name "__ViewState" that contains ViewState values stored in "Base64 Encrypted String" format.

Drawbacks of ViewState:

1. ViewState values are transported between Browser and Server every time we perform a PostBack, so in case if we store huge volumes of data in ViewState then transporting all that data between Browser and Server will increase network traffic.

2. As discussed above ViewState values are accessible only with the Same Page and that is the reason why we have explicitly transported "Failure Count" value to "FailureWithFailureCount.aspx" as a query string, because in the second page we can't access first page ViewState values.

Disabling ViewState in a WebPage:

It's possible to disable ViewState either for a particular Webpage or for the whole Application also which can be done in 2 different ways:

Page Level Disabling: by setting "EnableViewState" attribute of "Page Directive" value as "false" we can disable ViewState values for a particular page. To test this goto LoginWithFailureCount.aspx and add "EnableViewState" attribute to Page Directive as following:

<%@ Page EnableViewState="false" ... %>

Note: now if we run the WebForm we get "NullReferenceException" because ViewState is disabled.

Application Level Disabling: We can also disable ViewState application level i.e. under the Web.config file so that ViewState will not work in any page and to do that write the following code under <system.web> tag of Web.config file.

<pages enableViewState="false" />

Note: ControlState is turned "On" by default for every control on the page regardless of whether it is actually used during a post-back or not and serializes the data, and we can't turn off the ControlState of controls.

Option 2: Query Strings

This is another option by using which we can maintain the state of required values by concatenating those values to page URL as we have performed in our previous example.

Syntax: PageUrl?Key=Value&Key=Value&......&Key=Value

For example in our previous pages we have used query strings to transfer values to other pages as following:

```
Response.Redirect("SucessWithFailureCount.aspx?Name=" + txtUser.Text);
Response.Redirect("FailureWithFailureCount.aspx?Name=" + txtUser.Text + "&Count=" + Count);
```

Drawbacks of Query Strings:

1. By using this we can pass values only to a particular page and that to when we are transferring control either by using Server.Transfer or Response.Redirect methods.
2. We can't carry more volumes of data by using a Query String because it supports only 2048 bytes of data.
3. Passing values by using Query Strings is not secured because all those values are visible in the address bar of browser.

Option 3: Hidden Fields

By using Hidden Field control also we can maintain the state of values with in a particular page and more over as discussed earlier ViewState maintains its values thru hidden fields only. So without using ViewState we can use hidden fields and maintain the state of values on the page.

Hidden Fields Vs ViewState:

1. In case of ViewState, data is secured because it's stored in a "Base64 Encrypted String" format where as hidden field will not encrypt the data and if encryption is required we need to explicitly perform it by implementing our own logic.
2. ViewState values are not accessible to other pages whereas we can read Hidden Field values on the page where we are submitting, by using Request object.

Add 2 new WebForms in the project naming them as "HitCountWithHiddenField.aspx" and "NewForm.aspx" and write the following code under <div> tag of 1st form:

```
<asp:Button ID="btnCount" runat="server" Text="Hit Count" />
<asp:HiddenField ID="hfCount" runat="server" Value="0" /><br />
<asp:Button ID="btnNew" runat="server" Text="Launch New Form" PostBackUrl="~/NewForm.aspx" />
```

Now write following code under "Hit Count" button of 1st form:

```
int Count = int.Parse(hfCount.Value) + 1;
hfCount.Value = Count.ToString();
Response.Write("Hit Count: " + Count);
```

Now write following code under Page_Load of NewForm.aspx:

```
string Value = Request["hfCount"];
Response.Write("Hit counter of previous page is: " + Value);
```

Now run the 1st form, click on the Hit Count button and allow the Count value to increment and at any point of time go to "View Page Source" option on browser and you can see the "Hidden Field" with the same value as "Hit Count". Now click on "Launch New Form" button which will display the "Count" value of previous page on the new page.

Option 4: Cookies

A Cookie is a small piece of text that is used to store user-specific information and that information can be read by the web application when ever user visits the site. When a user requests for a web page, web server sends not just a page, but also a cookie containing the date and time. Cookies are stored in a folder on the user's hard disk and when the user requests for the web page again, browser looks on the hard disk for cookies associated with the web page and sends them to the Server. Browser stores cookies separately for each different site visited.

By using cookies also we can maintain the state of values between multiple pages for a single user. Asp.Net provides us options for both reading and writing cookies on client machines.

Writing Cookies on client machine: we can write cookies on client machine in 2 different ways, those are:

1. Create the instance of HttpCookie class; store values in to it as an array and then write that cookie to client machine by using Response object.
   ```
   HttpCookie cookie = new HttpCookie("LoginCookie");
   cookie["user"] = "Raju"; cookie["pwd"] = "admin";
   Response.Cookies.Add(cookie);
   ```
2. Write each value to browser as an individual cookie using Response object in a name/value combination.
   ```
   Response.Cookies["user"].Value = "Raju";
   Response.Cookies["pwd"].Value = "admin";
   ```

Reading Cookies back on our WebPages: we can read Cookies present on the client machine in any WebPage of our application by using Request object.

1. If we write the cookie to browser by using the 1st approach, we need to read it as following:
   ```
   HttpCookie cookie = Request.Cookies["LoginCookie"];
   string User = cookie["user"]; string Pwd = cookie["pwd"];
   ```
2. If we write the cookie to browser by using the 2nd approach, we need to read it as following:
   ```
   string User = Request.Cookies["user"].Value;
   string Pwd = Request.Cookies["pwd"].Value;
   ```

Cookies are or 2 types:

I. In-Memory Cookies
II. Persistent Cookies

In-Memory cookies are stored in browser's memory so once the browser is closed, immediately all the cookies that are associated with that browser window will be destroyed, and by default every cookie is In-Memory only. Persistent Cookies are stored on Hard Disk of the client machines, so even after closing the browser window also they will be persisting and can be accessed next time we visit the site. To make a cookie as persistent we need to set “Expires” property of Cookie with a “DateTime” value.

Setting expires property of Cookie:

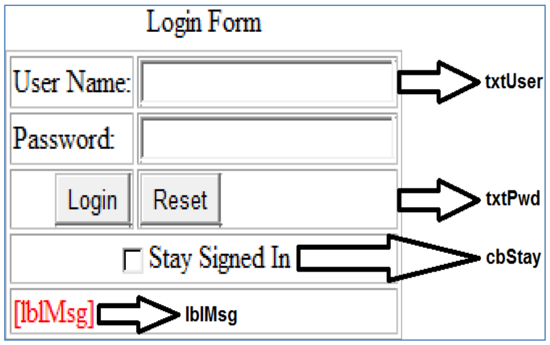
1. If we write the cookie to browser by using 1st approach, we need to set the expires property as following:
   ```
   cookie.Expires = <DateTime>;
   ```

2. If we write the cookie to browser by using 2nd approach, we need to set the expires property as following:
   ```
   Response.Cookies["user"].Expires = <DateTime>;
   Response.Cookies["pwd"].Expires = <DateTime>;
   ```

To work with Cookies first add 3 new WebForms under the project naming them as:

1. LoginWithStaySignedIn.aspx
2. SuccessWithStaySignedIn.aspx
3. FailureWithStaySignedIn.aspx

Design LoginWithStaySignedIn.aspx as following:



Now go to LoginWithStaySignedIn.aspx.cs file and write the following code:

Code under Page_Load:

```
HttpCookie cookie = Request.Cookies["LoginCookie"];
if (cookie != null) {
  Response.Redirect("SuccessWithStaySignedIn.aspx");
}
if(!IsPostBack) {
  ViewState["FailureCount"] = 0; txtUser.Focus();
}
```

Code under Login Button Click:

```
if(txtUser.Text == "admin" && txtPwd.Text == "admin") {
  HttpCookie cookie = new HttpCookie("LoginCookie");
  cookie["User"] = txtUser.Text; cookie["Pwd"] = txtPwd.Text;
  if (cbStay.Checked) {
    cookie.Expires = DateTime.Now.AddMonths(1);
  }
  Response.Cookies.Add(cookie);
  Response.Redirect("SuccessWithStaySignedIn.aspx");
}
else {
  int Count = (int)ViewState["FailureCount"] + 1;
  if (Count == 3) {
    Response.Cookies["User"].Value = txtUser.Text;
    Response.Cookies["Count"].Value = Count.ToString();
    Response.Redirect("FailureWithStaySignedIn.aspx");
}
  ViewState["FailureCount"] = Count;
  lblMsg.Text = Count + " attempt(s) failed, maximum are 3.";
}
```

Now go to SuccessWithStaySignedIn.aspx.cs file and write the following code:

Code under Page_Load:

```
HttpCookie cookie = Request.Cookies["LoginCookie"];
if (cookie == null) {
  Response.Redirect("LoginWithStaySignedIn.aspx");
}
string Name = cookie["User"];
Response.Write("<h3>Hello " + Name + ", welcome to the site.</h3>");
```

Now go to FailureWithStaySignedIn.aspx.cs file and write the following code:

Code under Page_Load:

```
if (Request.Cookies["User"] == null || Request.Cookies["Count"] == null) {
  Response.Redirect("LoginWithStaySignedIn.aspx");
}
string Name = Request.Cookies["User"].Value;
string Count = Request.Cookies["Count"].Value;
Response.Write("<h3>Hello " + Name + ", you have failed all the " + Count + " attempts to login.</h3>");
```

Now run "Login Page" and if we enter valid credentials it will take you to "SuccessPage" and if we enter wrong credentials it will check for 3 times and will take you to "Failure Page". When valid credentials are entered with "StaySignedIn" option checked, then it will not ask for credentials when we open the "Login Page" for next time and directly takes you to "Success Page" and in this case we can directly open "Success Page" from next time whereas if the credentials or not supplied, then if we try to open "Success Page" or "Failure Page" it will redirect us back to "Login Page" because unless credentials are provided we can't open any page of the site.

Drawbacks of Cookies:

1. We can create only 50 cookies for each website, so every new cookie from the site will override the old cookie once after reaching the limit.
2. A cookie can store only 4 K.B. of data that too of type string only.
3. Cookies are not secured because they are stored in text format on client machines.
4. Because cookies are stored on client machines there is a problem like clients can either delete the cookies or even disable cookies.

Disabling cookies on brower: Click on customize option beside address bar in chrome browser => select settings and in the window opened scroll down to "Advanced" => click on it, and under that click on "Content Settings/Site Settings" and in that click on "Cookies and site data" option and there switch off the option "Allow sites to save and read cookie data", which will change to "Blocked".

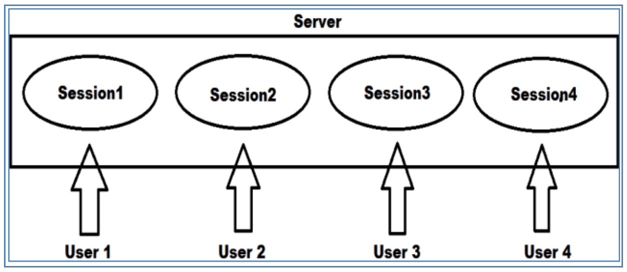
Delete Cookies on Browser: Click on customize option beside address bar in chrome browser => select settings and in the window opened scroll down to "Advanced" => click on it, and under that click on "Clear Browsing Data" and in that click on "Clear Data" button.

Option 5: Session

In ASP.NET session is a state that is used to store and retrieve values of a user across all the pages of site i.e. it is "Single User Global Data". It helps to identify requests from the same browser during a time period (session). It is used to store value for the particular time session. By default, ASP.NET session state is enabled for all

ASP.NET applications. Each created session is stored in "SessionStateItemCollection" object. We can get current session value by using Page object's Session property which in of type "HttpSessionState".

When ever a user sends his first request to server, server will provide a session for that user to store data that is associated with that user, providing the option of accessing that data across all the pages of site. Session doesn't have any size limitation and more over they can store any type of data like Simple Types and Complex Types also. Sessions are unique i.e. the session associated with one user is never accessible to other users.



Storing values into the Session:

Session[string key] = value (object)

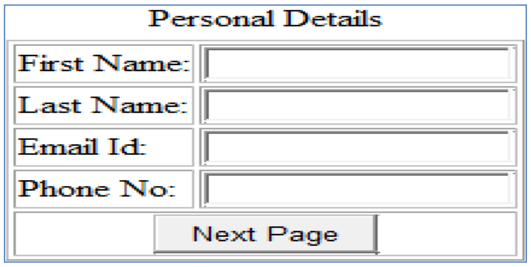
Accessing values from Session:

object value = Session[string key]

To test Sessions add 3 new WebForms under the project naming them as:

1. PersonalDetails.aspx
2. FamilyDetails.aspx
3. DisplayDetails.aspx

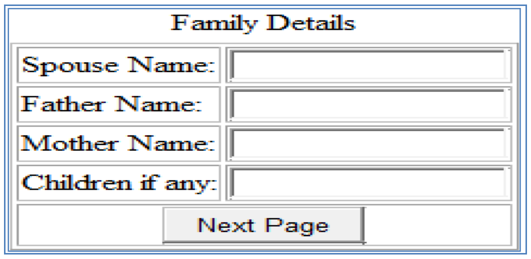
Step 1: Design PersonalDetails.aspx as following:



Now go to PersonalDetails.aspx.cs and write the following code under Next Page Button:

```
Dictionary<string, string> Details = new Dictionary<string, string>();
Details.Add("First Name", TextBox1.Text);     Details.Add("Last Name", TextBox2.Text);
Details.Add("Email", TextBox3.Text);          Details.Add("Phone", TextBox4.Text);
Session["UserDetails"] = Details;             Response.Redirect("FamilyDetails.aspx");
```

Step 2: Design FamilyDetails.aspx as following:



Now go to FamilyDetails.aspx.cs file and write following code in it:

Under Page_Load:

```
if (Session["UserDetails"] == null) { Response.Redirect("PersonalDetails.aspx"); }
if (!IsPostBack) { textBox1.Focus(); }
```

Under Next Page Button Click:

```
if (Session["UserDetails"] != null){
Dictionary<string, string> Details = (Dictionary<string, string>)Session["UserDetails"];
Details.Add("Spouse Name", TextBox1.Text);            Details.Add("Father Name", TextBox2.Text);
Details.Add("Mother Name", TextBox3.Text);            Details.Add("Children Count", TextBox4.Text);
Session["UserDetails"] = Details;                     Response.Redirect("DisplayDetails.aspx");
}
else { Response.Redirect("PersonalDetails.aspx"); }
```

Step 3: Go to DisplayDetails.aspx.cs file and write thefollowing code under Page_Load:

```
if (Session["UserDetails"] != null) {
  Dictionary<string, string> Details = (Dictionary<string, string>)Session["UserDetails"];
  foreach (string Key in Details.Keys){Response.Write(Key + ": " + Details[Key] + "<br />"); }
}
else { Response.Redirect("PersonalDetails.aspx"); }
```

Note: In the above example 1st we are storing data of “PersonalDetails” page in to a Dictionary, putting that Dictionary into Session and redirecting to “FamilyDetails” page, and in that page we are picking the Dictionary from Session which is stored by “PersonalDetails” page, storing “FamilyDetails” data also into the Dictionary and again putting that Dictionary into the Session which is finally captured in “DisplayDetails” page to display them in the form of name/value pairs.

Run “PersonalDetails.aspx” page, fill in the details, and click on the "Next Page" button which will launch “FamilyDetails.aspx” and here also fill in the details and click on the "Next Page" button to launch "DisplayDetails.aspx", which will display all the values that are captured from the first 2 pages. Now copy the URL of the “DisplayDetails.aspx” page in address bar, open a new tab in the same browser window, paste the URL in address bar to execute, and here also we see the values that are captured in 1st and 2nd pages, whereas if we open a new instance of a new browser and paste the URL it will not display the values but takes you to the PersonalDetails.aspx because Session values of 1 user are not shared with another user.

How a session is identified to which user it belongs to?

Ans: Whenever a Session is created for a user it is given with a unique Id known as “SessionId” and this “SessionId” is written to clients browser in the form of a “In-Memory Cookie”, so whenever the client comes back to the server in a next request, server will read the Cookie, picks the “SessionId”and associates user with his exact Session.

Note: because SessionId is stored on client's browser in the form of an “In-Memory Cookie”, other tabs under the browsers instance can also access that session, where as a new instance of a new browser can't access the Session.

How SessionId is maintianed if the browser disables cookies?

Ans: if the browser disables Cookies then Sessions are not maintained for that browser or user, and to test this disable Cookies in Chrome Browser and run the “PersonalDetails.aspx” page, fill in the details and click on “Next Page” button but still we will be on the same page.

To resolve the above problem we need to configure <sessionState> element in the Web.config file. <sessionState> element has an attribute known as cookieLess and it can be set with different values like false(d), true and AutoDetect.

1. Default value is false which indicates that sessions can't be maintained without a cookie.
2. If set as true, then without storing the “SessionId” in the form of a cookie, server will store “SessionId” in the URL of Page, so in this case even if the browser supports Cookies also, without using them, “SessionId” is storedin URL. The problem in this approach is if we copy the URL of one instance of browser and use it under another instance of a new browser, we can access the Session values that are associated with the “SessionId” present in the URL.
3. If set as “AutoDetect” then server will first verify, whether browser supports cookies or not and if supported “SessionId” is stored as Cookie and if not supported “SessionId” is stored in Page URL and this is the most recommended option to use.

To try this go to Web.config file and write the following code under <system.web> tag:

<sessionState cookieless="AutoDetect" />

Note: <sessionState> element in Web.config is used for making settings to Session and its behavior.

Now run “PersonalDetails.aspx” page again and notice the URL in address bar which will contain “SessionId” if cookies are disabled in browser, or else we will not find “SessionId” in Page URL.

What happens to Sessions associated with clients if the client closes the browser?

Ans: Every Session will be having a “time-out” period of “20 Minutes (default)” from the last request (Sliding Expiration), so within 20 Minutes if the Session is not used by the User, Server will destroy that Session.

Note: we can change the default time-out period of “20 Mins” to our required value thru “Web.config” file by setting “timeout” attribute value of “sessionState” element as following:

<sessionState timeout="10" />

Can we explicitly destroy a Session associated with a User?

Ans: Yes this can be performed by calling “Abandon()” method on the Session and this is what we generally do under “Sign Out” or “Log Out” options in a Website or Web Application.

E.g.: Session.Abandon();

Where will Web Server store Session values?

Ans: Web Server can store Session values in 3 different locations:

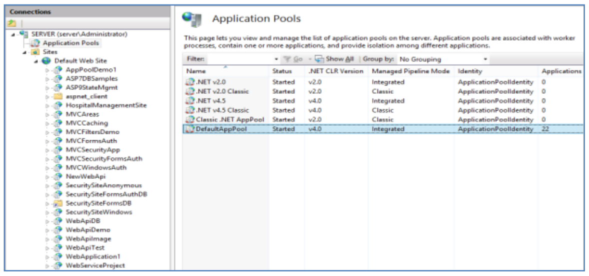
I. In-Proc [d]
II. State Server
III. Sql Server

In-Proc: this is the default option used for storing Session values and in this case Session values are stored under the memory of "IIS Worker Process".

What is IIS Worker Process?

Ans: Under IIS, Web Application run’s inside of a container known as Application Pool and an Application Pool is a logical container of Web Applications that will execute under a single or multiple worker processes. Application

Pool is the heart of a Web Application and by default under IIS all Web Applications runs under the same Application Pool which is created when IIS is installed i.e. "Default App Pool". To check that open "IIS Manager" and in the LHS under "Connections Panel" we will find "Applications Pools" and "Sites" options, select "Appliation Pools" which displays "DefaultAppPool" on the RHS.



This page lets you view and manage the list of application pools on the server. Application pools are associated with worker processes, contain one or more applications, and provide isolation among different applications.

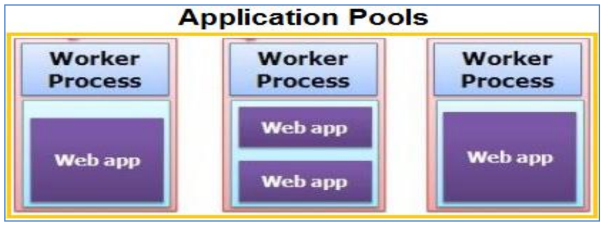
Name	Status	.NET CLR Version	Managed Pipeline Mode	Identity	Applications
.NET v2.0	Started	v2.0	Integrated	ApplicationPoolIdentity	0
.NET v2.0 Classic	Started	v2.0	Classic	ApplicationPoolIdentity	0
.NET v4.5	Started	v4.0	Integrated	ApplicationPoolIdentity	0
.NET v4.5 Classic	Started	v4.0	Classic	ApplicationPoolIdentity	0
Classic .NET AppPool	Started	v2.0	Classic	ApplicationPoolIdentity	0
DefaultAppPool	Started	v4.0	Integrated	ApplicationPoolIdentity	22

If you notice the above there are 22 applications on the server running under "DefaultAppPool" and it is still possible to run each Web Application under a separate Application Pool which enables us to isolate our Web Application for better Security, Reliability and Availability; therefore, problems in one application pool do not affect Web Sites or Applications in other Application Pools.

Note: when ever a new Site is created under IIS, it will also create an Application Pool under which the Site runs and name of that Application Pool will be the same name of Site.

We can also create our own Application Pools under IIS and to do that right click on Application Pools node under Connection Panel and select "Add Application Pool" which opens a window asking for a name, enter name as "MyPool" and click "Ok". Currently the new Application Pool i.e. "MyPool" doesn't have any applications running under it and if we want our "ASPStateMgmt" application to run under "MyPool", then right click on "ASPStateMgmt" under "Default Web Site" and choose Manage Applications => Advanced Settings, which opens a window and in that select "Application Pool" and Click on the button beside it which opens another window listing all the Pools that are available in a DropDownList, select "MyPool" and Click Ok and Ok again.

The IIS Worker Process is a Windows Process (w3wp.exe) which runs Web applications, and is responsible for handling requests sent to a Web Server for a specific Application Pool. Each application pool creates at least one instance of w3wp.exe and that is what actually processes requests in your application, responsible for processing Asp.net application request and sending back response to the client. All ASP.NET functionalities run within the scope of this Worker Process.



We can view the "IIS Worker Process" that is associated with each Application Pool under "Task Manager" which displays a separate "IIS Worker Process" for each and every Application Pool. To get the exact details go to "Details Section" in the Task Manager and there we find "w3wp.exe" and beside that it will display the Application Pool to which the Worker Process is associated.

Note: in In-Proc session mode Session Values are stored under the Memory that is associated with IIS Worker Process, who runs our Web Application. So in this case if we recycle the IIS Worker Process all the Session Values that are stored under this will be destroyed and to test this, open “Task Manager” identify the “IIS Worker Process” under which our Web Application is running, select it and click on “End Task” button which will destroy all the session values associated with that application.

To test the above add a new WebForm under Project naming it as "HitCounter.aspx", place a button on it and set the text as "Hit Count" and write the following code under its Click Event Handler:

```
int Count = 0;
if (Session["HitCounter"] == null) { Count = 1; }
else { Count = (int)Session["HitCounter"] + 1; }
Session["HitCounter"] = Count;
Response.Write("Hit Count: " + Count);
```

Now run the WebPage, click on the “Hit Count” button for multiple times to increment the “Count” value and at any point of time (without closing the browser) open the “Task Manager” and recycle the IIS Worker Process which is associated with our Application, now go back to the browser, click on the “Hit Count” button and notice its value will be 1 again because the old Session value is destroyed.

State Server: this is seperate software for storing “Session Values” which is installed when we install .Net Framework on any machine and it can be found in the “Services Window”. To see that go to Control Panel => Administrative Tools => Services and in that we find “Asp.Net State Server” Service.

To use this first we need to set “mode” attribute of <sessionState> element in the “Web.config” file and “mode” attribute accepts any of the following values like: Off, In-Proc [d], Sql Server and State Server.
Note: default is In-Proc i.e. Sessions are stored under IIS Worker Process Memory, if set as off application will not maintain Sessions at all and if set as State Server then Session values are stored under Asp.Net State Service.

To use State Service for storing Session values we need to do the following:
Step 1: Open Windows Services console, right click on Asp.Net State Service and select Start to start the service.
Step 2: Now open Web.config file and write the sessionState tag as following:

```
<sessionState mode="StateServer" stateConnectionString="tcpip=localhost:42424" />
```

Note: “localhost” refers to the machine name and if Asp.Net State Server software is running on a remote machine then write that machine name in place of localhost and 42424 is the Port No. on which Asp.Net State Server software will be running.

Now run “HitCounter.aspx” page again and click on “Hit Count” button to increment the “Count” value and at any point of time, recycle the “IIS Worker Process” in “Task Manager”, go back to the browser and click on the “Hit count” button and notice that “Count” value will not be reset to 1 because now Sessions are not stored under “IIS Worker Process Memory” but stored on “Asp.Net State Server” and if the “ount” value has to be reset we need to “Restart” the “State Server” service under “Services Console”.

SQL Server: if “Session Mode” is set as Sql Server then Session values are stored under Sql Server DB and to use that we need to do the following:

Step 1: run the “aspnet_regsql” tool at “VS Developer Command Prompt”, so that the required DB, Tables and Stored Procedure for maintaining Sessions will be created under Sql Server.

aspnet_regsql -? => Help
aspnet_regsql -S <Server Name> -U <User Id> -P <Password> -E < In-case of Windows Auth> -ssadd -sstype t|p|c

-S: to specify Sql Server name.
-U: to specify User Id in case of Sql Authentication.
-P: to specify password in case of Sql Authentication.
-E: this has to be used in case of Windows Authentication and in such case don't use -U and -P option again.
-ssadd: is to enable support for Sql Server Session State and this will create a DB on the server.
Note: If we want to remove the support for session state we need to use -ssremove in place -ssadd.
-sstype: is to specify the type of tables we want to use where "t" indicates temporary tables, "p" indicates persisted tables and "c" indicates custom tables - but in this case we need to create our own DB to store Session data, and to specify that database name we need to use "-d <Database Name>" option in the last.

Testing the processes of using "aspnet_regsql":
For Sql Server Authentication: *aspnet_regsql -S Server -U Sa -P 123 -ssadd -sstype t*
For Windows Authentication: *aspnet_regsql -S Server -E -ssadd -sstype t*

Note: in the above case we are using temporary tables for storing Session State values, so a new Database is created on the Server with the name "ASPState" and under the DB it creates a set of Stored Procedures for managing the data and because we have asked for temporary tables, all the required tables gets created on "TempDB - System Database" and this Database will be re-created every time we re-start Sql Server, whereas if we ask for persisted tables then all the required tables also gets created on "ASPState" database only, so even if we re-start Sql Server, then also tables and their values will be persisting.

Step 2: Now open Web.config file and re-write the <sessionState> tag as following:

Sql Server Authentication:

```
<sessionState mode="SQLServer" sqlConnectionString="Data Source=Server;User Id=Sa;Password=123" />
```

Windows Authentication:

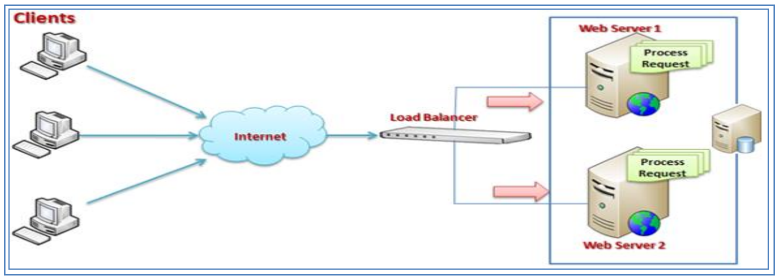
```
<sessionState mode="SQLServer" sqlConnectionString="Data Source=Server;Integrated Security=SSPI" />
```

Now run "HitCounter.aspx" page again, click on "Hit Count" button to increment "Count" value and now either recycle the "IIS Worker Process" or restart "State Server" but still the "Count" value will not be reset and if we want the value to be reset we need to restart Sql Server under "Services Console".

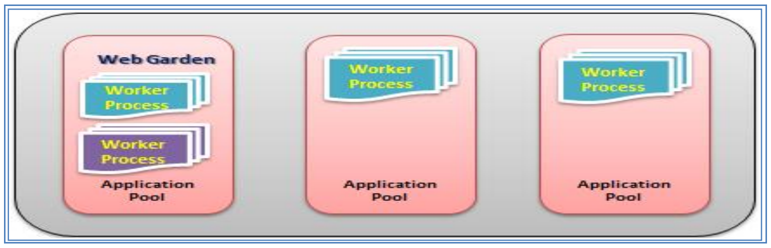
Removing support for Sql Server Session State: *aspnet_regsql -S Server -U sa -P 123 -ssremove*

When to use State Server and Sql Server Session Modes over In-Proc Session Mode?
Ans: State Server and Sql Server Session Mode options are used in scenarios where a Web Application is running in "Web Farm" or "Web Garden" architectures.

Web Farm: it's an approach of hosting a Web Application on multiple Web Servers for balancing the load, so that client request first comes to Load Balancer and Load Balancer will in-turn redirect the client to an appropriate Web Server which is free currently. In case of Web Farm architecture "In-Proc" Session Mode can't be used because if each request from the same client is re-directed to a different Web Servers then Session Data of 1 Web Server is not accessible to other Web Servers, so to overcome this problem we install "State Server" or "Sql Server" Software on a remote system and configure all the Web Servers to that machine as following:



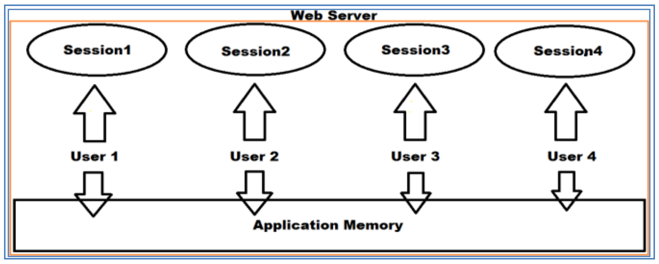
Web Garden: as discussed earlier every Application Pool under which our Web Application executes will be having 1 Worker Process to execute that Application, but we can create more than 1 Worker Process also in an Application Pool and if we do that we call that as Web Garden.



To add Worker Processes to an Application Pool right click on the Application Pool in "IIS Manager", select "Advanced Settings" which opens a window and in that window, under the "Process Model" settings we find "Maximum Worker Processes" with the value 1, change it to any value other than 1 and click Ok.

Option 6: Application State

Application-State is also a state management technique and it is a global storage mechanism that is used to store data on the server which is shared between all users i.e. data stored in Application-State is accessible to all users and any where in the application (Multi-User Global Data). Application-State is stored in the memory of the Web Server and is faster than storing and retrieving information from a database. Application-State is used in the same way as Session-State, but Session-State is specific for a single user, where as Application-state is common for all users of the application.

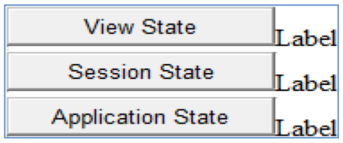


Application-State does not have any default expiration period like Session-State, so when we recycle the worker process only the application object will be lost. Data is stored into Application-State in the form of key/value pairs only like we store in Session-State and ViewState. We store data into Application-State by using "Application" object of our parent class "Page" and that Application object is of type "HTTPApplicationState" class.

Storing values into Application-State:	**Accessing values from Application-State:**
Application[string key] = value (object)	object value = Application[string key]

Note: Application Memory is not Thread-Safe, so to overcome the problem whenever we are dealing with Application-State data we need to call Lock() and UnLock() methods on Application object.

To compare the difference between “ViewState”, “Session-State” and “Application-State” memory add a new WebForm under the project naming it as “CompareStates.aspx” and design it as following:



Now goto CompareStates.aspx.cs file and write the following code:

<u>Code under View State Button:</u>

```
int Count = 0;
if (ViewState["Counter"] == null) { Count += 1; }
else { Count = (int)ViewState["Counter"] + 1; }
ViewState["Counter"] = Count;
Label1.Text = "View State: " + Count;
```

<u>Code under Session State Button:</u>

```
int Count = 0;
if (Session["Counter"] == null) { Count += 1; }
else { Count = (int)Session["Counter"] + 1; }
Session["Counter"] = Count;Label2.Text = "Session State: " + Count;
```

<u>Code under Application State Button:</u>

```
Application.Lock();
int Count = 0;
if (Application["Counter"] == null) { Count += 1; }
else { Count = (int)Application["Counter"] + 1; }
Application["Counter"] = Count;
Label3.Text = "Application State: " + Count;
Application.UnLock();
```

Now run the above page and click on the 3 buttons to increment their count up to 5, then copy the URL of page, open a new tab and call the page by using copied URL. When we click on “ViewState” button value will be 1 because it is “Single User Local Data”. When we click on “Session State” button the value will be 6 because it is “Single User Global Data” and the “Session-Id” is accessible between tabs of the browser, then open another instance of a new browser, open the page using copied URL and in this case ViewState value will 1, Session value also will be 1 because “Session-Id” of the old browser is not carried to new instance of new browser but application value will be “old + 1” as this is “Multi User Global Data”.

<u>Global.asax:</u> This file is also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events raised by ASP.NET. The file contains a class with the name “Global” that extends the HttpApplication class. There can be only one “Global.asax” file per application and it should be located in the application's root directory only. Every ASP.Net Web Application project contains this file by default. Global class contains methods for handling application events like:

1. Application_Start
2. Application_End
3. Application_Error
4. Session_Start
5. Session_End

Application_Start: This method is invoked when the application first starts i.e. whenever the first request comes to the application and executes 1 and only 1 time in the life cycle of an application. This event handler is a useful place to provide application-wide initialization code.
Application_End: This method is invoked just before an application ends. The end of an application can occur because IIS is being restarted or because the application is transitioning to a new application domain in response to updated files or the worker process recycling and it typically contains application cleanup logic.
Application_Error: This method is invoked whenever an unhandled exception occurs in the application and we implement all the error handling code in this.
Session_Start: This method is invoked each time a new session begins i.e. when the first request comes from a user. This is often used to initialize user-specific information.
Session_End: This method is invoked whenever the user's session ends. A session ends when your code explicitly releases it or when it's time is out after there have been no more requests received within a given timeout period (typically 20 minutes) and this contains logic for cleanup of user specific data.

Note: Global.asax file is never called directly by the user, rather they are called automatically in response to application events. When a Global.asax file changes, the framework reboots the application and the Application_OnStart event is fired once again when the next request comes in. Note that the Global.asax file does not need recompilation if no changes have been made to it.

Writing a program to find out the "Total No. of Users" and "Total No. of Online Users" for a site by using "Global.asax".

Step 1: Open Global.asax file and write the below code under the class Global.

Code under Application_Start method:
Application["TNU"] = 0; Application["TOU"] = 0;

Code under Session_Start method:
Application.Lock(); Application["TNU"] = (int)Application["TNU"] + 1;
Application["TOU"] = (int)Application["TOU"] + 1; Application.Unlock();

Code under Session_End method:
Application.Lock(); Application["TOU"] = (int)Application["TOU"] - 1; Application.Unlock();

Step 2: Add a new WebForm under project, name it as UsersCount.aspx and write the below code under its div tag.
Total Users: <asp:Label ID="Label1" runat="server" Text="Label" /><br />
Online Users: <asp:Label ID="Label2" runat="server" Text="Label" /><br />
<asp:Button ID="btnSignOut" runat="server" Text="Sign-Out" />

Step 3: Now goto UsersCount.aspx.cs file and write the below code.

Code under Page_Load:
Label1.Text = Application["TNU"].ToString(); Label2.Text = Application["TOU"].ToString();

Under Sign-Out Button: *Session.Abandon();*
Run the page for multiple times and notice both "TNU" and "TOU" will increment and if at all we click on Sign-Out button then we notice "TOU" value getting decremented.

ADO.Net

Pretty much every application deals with data in some manner, whether that data comes from memory, databases, XML files, text files, or something else. The location where we store the data can be called as a Data Source or Data Store where a Data Source can be a file, database, address books or indexing server etc.

Programming Languages cannot communicate with Data Sources directly because each Data Source adopts a different Protocol (set of rules) for communication, so to overcome this problem long back Microsoft has introduced intermediate technologies like Odbc and Oledb which works like bridge between the Applications and Data Sources to communicate with each other.

ODBC (Open Database Connectivity) is a standard C programming language middleware API for accessing database management systems (DBMS). ODBC accomplishes DBMS independence by using an ODBC driver as a translation layer between the application and the DBMS. The application uses ODBC functions through an ODBC driver manager with which it is linked, and the driver passes the query to the DBMS. An ODBC driver will be providing a standard set of functions for the application to use, and implementing DBMS-specific functionality. An application that can use ODBC is referred to as "ODBC-Compliant". Any ODBC-Compliant application can access any DBMS for which a driver is installed. Drivers exist for all major DBMS’s as well as for many other data sources like Microsoft Excel, and even for Text or CSV files.ODBC was originally developed by Microsoft in 1992.

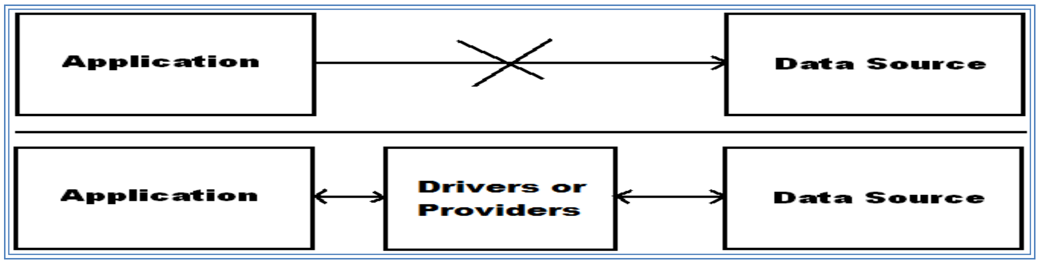
1. It's a collection of drivers, where these drivers sit between the App's and Data Source's to communicate with each other and more over we require a separate driver for each and every data source.
2. Odbc drivers comes along with your Windows O.S. and we can find them at the following location:
   Control Panel => Administrative Tools => Odbc Data Sources
3. To consume these Odbc Drivers first we need to configure them with the data source by creating a "DSN" (Data Source Name).
4. Odbc drivers are open source i.e. there is an availability of these Odbc Drivers for all the leading O.S's in the market.

<u>Drawbacks with Odbc Drivers:</u>

1. These drivers must be installed on each and every machine where the application is executing from and then the application, driver and data source should be manually configured with each other.
2. Odbc Drivers are initially designed for communication with Relational DB only.

OLE DB (Object Linking and Embedding, Database, sometimes written as OLEDB or OLE-DB), an API designed by Microsoft, allows accessing data from a variety of data sources in a uniform manner. The API provides a set of interfaces implemented using the Component Object Model (COM) and SQL. Microsoft originally intended OLE DB as a higher-level replacement for, and successor to, ODBC, extending its feature set to support a wider variety of non-relational databases, such as object databases and spreadsheets that do not necessarily implement SQL.OLE DB is conceptually divided into consumers and providers. The consumers are the applications that need access to the data, and the providers are the software components that implement the interface and therebyprovide the data to the consumer.An OLE DB provider is a software component enabling an OLE DB consumer to interact with a data source. OLE DB providers are alike to ODBC drivers. OLE DB providers can be created to access such simple data stores as a text file and spreadsheet, through to such complex databases as Oracle, Microsoft SQL Server, and many others. It can also provide access to hierarchical data stores. These OLE DB Providers are introduced by Microsoft around the year 1996.

1. It’s a collection of providers where these providers sit between the App's and Data Source to communicate with each other and we require a separate provider for each data source.
2. Oledb Providers are designed for communication with relational and non-relational data source also i.e. it provides support for communication with any Data Source.
3. Oledb Providers sits on server machine so they are already configured with data source and when we connect with any data source they will help in the process of communication.
4. Oledb Providers are developed by using COM and Sql Languages, so they are also un-managed.
5. Microsoft introduced OLEDB as a replacement for ODBC for its Windows Systems.
6. Oledb is a pure Microsoft technology which works only on Windows Platform.



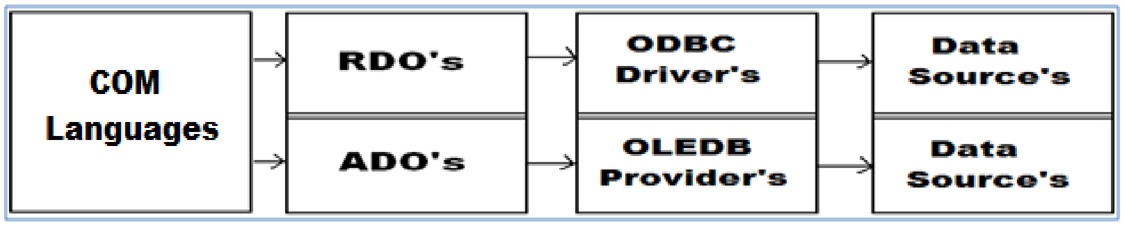
Things to remember while working with Odbc and Oledb:

1. Odbc and Oledb are un-managed or platform dependent.
2. Odbc and Oledb are not designed targeting any particular language i.e. they can be consumed by any language like: C, CPP, Visual Basic, Visual CPP, Java, CSharp etc.

Note: If any language wants to consume Odbc Drivers or OledbProviders they must use some built-in libraries of the language in which we are developing the application without writing complex coding.

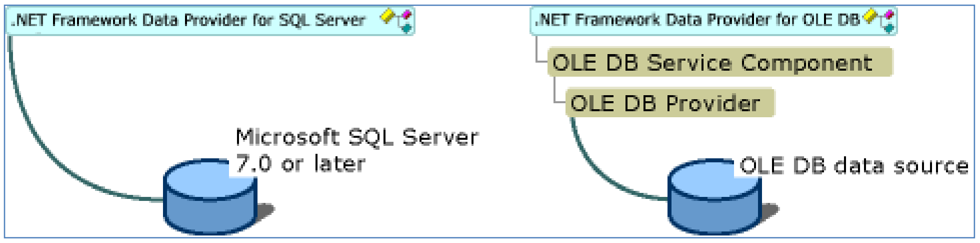
RDO’s and ADO’s in Visual Basic Language:

Visual Basic Language used RDO’s (Remote Data Objects) and ADO’s (ActiveX Data Objects) for data source communication without having to deal with the comparatively complex ODBC or OLEDB API.



.NET Framework Providers:

The .NET Framework Data Provider for SQL Server uses its own protocol to communicate with SQL Server. It is lightweight and performs well because it is optimized to access a SQL Server directly without adding an OLE DB or ODBC layer and it supports SQL Server software version 7.0 or later. The .NET Framework Data Provider for Oracle (OracleClient) enables data access to Oracle data sources through Oracle Client connectivity software. The data provider supports Oracle client software version 8.1.7 or a later.


.NET Framework Data Provider for SQL Server
Microsoft SQL Server 7.0 or later
.NET Framework Data Provider for OLE DB
OLE DB Service Component
OLE DB Provider
OLE DB data source

<u>ADO.Net:</u>

It is a set of types that expose data access services to the .NET programmer. ADO.NET provides functionality to developers writing managed code similar to the functionality provided to native COM developers by ADO. ADO.NET provides consistent access to data sources such as Microsoft SQL Server, as well as data sources exposed through OLE DB and XML. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data. It is an integral part of the .NET Framework, providing access to relational data, XML, and application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications or Internet browsers.

ADO.Net provides libraries for Data Source communicationunder the following namespaces:

- System.Data
- System.Data.Odbc
- System.Data.Oledb
- System.Data.SqlClient
- System.Data.OracleClient

<u>Note:</u> System.Data, System.Data.Odbc, System.Data.Oledb and System.Data.SqlClient namespaces are under the assembly System.Data.dll whereas System.Data.OracleClient is under System.Data.OracleClient.dll assembly.

<u>System.Data:</u> types of this namespace are used for holding and managing of data on client machines. This namespace contains following set of classes in it: **DataSet, DataTable, DataRow, DataColumn, DataView, DataRelation**, etc.

<u>System.Data.Odbc:</u> types of this namespace can communicate with any Relational DataSource using Un-Managed Odbc Drivers.

<u>System.Data.Oledb:</u> types of this namespace can communicate with any Data Source using Oledb Providers (Un-Managed COM Providers).

<u>System.Data.SqlClient:</u> types of this namespace can purely communicate with Sql Server database only using SqlClient Provider (Managed .Net Framework Provider).

<u>System.Data.OracleClient:</u> types of this namespace can purely communicate with Oracle database only using OracleClient Provider (Managed .Net Framework Provider).

All the above 4 namespaces contains same set of types as following: **Connection, Command, DataReader, DataAdapter, Parameter and CommandBuilder etc**, but here each class is referred by prefixing with theirnamespace before the class name to discriminate between each other as following:

OdbcConnection	OdbcCommand	OdbcDataReader	OdbcDataAdapter	OdbcCommandBuilder	OdbcParameter
OledbConnection	OledbCommand	OledbDataReader	OledbDataAdapter	OledbCommandBuilder	OledbParameter
SqlConnection	SqlCommand	SqlDataReader	SqlDataAdapter	SqlCommandBuilder	SqlParameter
OracleConnection	OracleCommand	OracleDataReader	OracleDataAdapter	OracleCommandBuilder	OracleParameter

Performing operations on a DataSource: the operations we perform on a Data Source will be Select, Insert, Update and Delete, and each and every operation we perform on a Data Source involves in 3 steps, like:

- Establishing a connection with the data source.
- Sending a request to data source in the form of an SQL Statements.
- Capturing the results given by the data source.

Establishing a Connection with Data Source:

It's a process of opening a channelfor communication between Application and Data Source that ispresent either on a local or remote machine to perform DBOperations and to open the channel for communication we use Connection class.

Constructors of the Class:

Connection()	Connection(string ConnectionString)

Note: ConnectionString is a collection of attributes that are required for connecting with a DataSource, those are:

- DSN
- Provider
- Data Source
- User Id and Password
- Integrated Security
- Database or Initial Catalog
- Extended Properties

DSN: this is the only attribute that is required if we want to connect with a data source by using Odbc Drivers and by using this we need to specify the DSN Name.

Provider: this attribute is required when we want to connect to the data source by using Oledb Providers. So by using this attribute we need to specify the provider name based on the data source we want to connect with.

Oracle:	Msdaora or ORAOLEDB.ORACLE	Sql Server:	SqlOledb
MS-Access or MS-Excel:	Microsoft.Jet.Oledb.4.0	MS-Indexing Server:	Msidxs

Data Source: this attribute is required to specify the server name if the data source is a database or else if the data source is a file we need to specify path of the file and this attribute is required in case of any provider communication.

User Id and Password: This attribute is required to specify the credentials for connection with a database and this attribute is required in case of any provider communication.

Oracle: Scott/tiger	Sql Server: Sa/123

Integrated Security: this attribute is used while connecting with Sql Server Database only to specifythat we want to connect with the Server by using Windows Authentication and in this case weshould not use User Id and Password attributes and this attribute is required in case of any provider communication.

Database or Initial Catalog: these attributes are used while connecting with Sql Server Database to specify the name of database we want to connect with and this attribute is required in case of any provider communication.

Extended Properties: this attribute is required only while connecting with MS-Excel using Oledb Provider.

List of attributes which are required in case of Odbc Drivers, Oledb and .Net Framework Providers:

Attribute	**ODBC Driver**	**OLEDB Provider**	**.Net Framework Provider**
DSN	Yes	No	No
Provider	No	Yes	No
Data Source	No	Yes	Yes
User Id and Password	No	Yes	Yes
Integrated Security*	No	Yes	Yes
Database or Initial Catalog*	No	Yes	Yes
Extended Properties**	No	Yes	-

*Only for Sql Server **Only for Microsoft Excel

Connection String for SqlServer to connect by using different options:

OdbcConnection con = new OdbcConnection("Dsn=<Dsn Name>");
OledbConnection con = new OledbConnection("Provider=SqlOledb;Data Source=<Server Name>;
Database=<DB Name>;User Id=<User Name>;Password=<Pwd>");
SqlConnection con = new SqlConnection("Data Source=<Server Name>;Database=<DB Name>;
User Id=<User Name>;Password=<Pwd>");

Note: in case of Windows Authentication in place of User Id and Password attributes we need to use Integrated Security = SSPI (Security Support Provider Interface).

Connection String for Oracle to connect by using different options:

OdbcConnection con = new OdbcConnection("Dsn=<Dsn Name>");
OledbConnection con = new OledbConnection("Provider=Msdaora;Data Source=<Server Name>;
User Id=<User Name>;Password=<Pwd>");
OracleConnection con = new OracleConnection("Data Source=<Server Name>;
User Id=<User Name>;Password=<Pwd>");

Connection String for MS-Excel to connect by using different options:

OdbcConnection con = new OdbcConnection("Dsn=<Dsn Name>");
OledbConnection con = new OledbConnection("Provider=Microsoft.Jet.Oledb.4.0;
Data Source=<Path of Excel Document>;Extended Properties=Excel 8.0");

Members of Connection class:

1. **Open():** a method which opens a connection with data source.
2. **Close():** a method which closes the connection that is open.
3. **State:** an enumerated property which is used to get the status of connection.
4. **ConnectionString:** a property which is used to get or set a connection string that is associated with the connection object.

Object of class Connection can be created in any of the following ways:

Connection con = new Connection();
con.ConnectionString = "<Connection String>";
or
Connection con = new Connection("<Connection String>");

Testing the process of establishing a connection: open a new project of type "ASP.Net Web Application" and name it as DBExamples. Now add a new WebForm naming it as "TestConnection.aspx" and write the following code under <div> tag:

```
<asp:Button ID="B1" runat="server" Text="Connect with Oracle using Oledb" OnClick="B1_Click" />
<br />
<asp:Button ID="B2" runat="server" Text="Connect with Sql Server using Oledb" OnClick="B2_Click" />
<br />
<asp:Button ID="B3" runat="server" Text="Connect with Oracle using OracleClient" OnClick="B3_Click" />
<br />
<asp:Button ID="B4" runat="server" Text="Connect with Sql Server using SqlClient" OnClick="B4_Click" />
<br />
<asp:Button ID="B5" runat="server" Text="Connect with Oracle using Odbc" OnClick="B5_Click"/>
<br />
<asp:Button ID="B6" runat="server" Text="Connect with Sql Server using Odbc" OnClick="B6_Click" />
```

Now create 2 DSN's for connecting with Oracle and Sql Server and to do that go to Control Panel => Administrative Tools => ODBC Data Sources => Click On Add Button => Choose a Driver for Oracle => Click Finish => Enter Data Source Name as: OraDsn, Description as: Connects with Oracle DB, TNS Service Name as: <Your Oracle Server Name>, User Id as: <User Name>/<Password> and Click Ok which Creates a New DSN for Oracle.

Same as the above again Click on Add Button => Choose a Driver for Sql Server => Click Finish => Enter Name as: SqlDsn, Description as: Connects with Sql Server DB, Server as: <Server Name>, Click Next, choose the Authentication Mode as Windows or Sql & if it is Sql then enter User Id & Password below, Click Next for choosing the DB, default will be Master, leave the same, Click Next and Click Finish which Creates a New DSN for Sql Server.

Now go back to your project, add reference to System.Data.OracleClient.dll and write the following code under "TestConnection.aspx.cs" file:

using System.Data.Odbc; using System.Data.OleDb; using System.Data.SqlClient; using System.Data.OracleClient;

Code under B1_Click:

```
OleDbConnection con = new OleDbConnection(
        "Provider=Msdaora;Data Source=Server Name;User Id=Scott;Password=tiger");
con.Open(); Response.Write("Connection is " + con.State + " with Oracle using Oledb. <br />");
con.Close(); Response.Write("Connection is " + con.State + " with Oracle using Oledb. <br />");
```

Code under B2_Click:

```
OleDbConnection con = new OleDbConnection();
con.ConnectionString = "Provider=SqlOledb;Data Source=Server;User Id=sa;Password=123;Database=Master";
//con.ConnectionString = "Provider=SqlOledb;Data Source=Server;Integrated Security=SSPI;Database=Master";
con.Open(); Response.Write("Connection is " + con.State + " with Sql Server using Oledb. <br />");
con.Close(); Response.Write("Connection is " + con.State + " with Sql Server using Oledb. <br />");
```

Code under B3_Click:

```
OracleConnection con = new OracleConnection("Data Source=Server;User Id=Scott;Password=tiger");
con.Open(); Response.Write("Connection is " + con.State + " with Oracle using OracleClient. <br />");
con.Close(); Response.Write("Connection is " + con.State + " with Oracle using OracleClient. <br />");
```

<u>Code under B4_Click:</u>
SqlConnection con = new SqlConnection();
con.ConnectionString = "Data Source=Server;User Id=Sa;Password=123;Database=Master";
//con.ConnectionString = "Data Source=Server;Integrated Security=SSPI;Database=Master";
con.Open();Response.Write("Connection is " + con.State + " with Sql Server using SqlClient. <br />");
con.Close();Response.Write("Connection is " + con.State + " with Sql Server using SqlClient. <br />");

<u>Code under B5_Click:</u>
OdbcConnection con = new OdbcConnection("Dsn=OraDsn");
con.Open();Response.Write("Connection is " + con.State + " with Oracle using Odbc. <br />");
con.Close();Response.Write("Connection is " + con.State + " with Oracle using Odbc. <br />");

<u>Code under B6_Click:</u>
OdbcConnection con = new OdbcConnection(); con.ConnectionString = "Dsn=OraDsn";
con.Open();Response.Write("Connection is " + con.State + " with Sql Server using Odbc. <br />");
con.Close();Response.Write("Connection is " + con.State + " with Sql Server using Odbc. <br />");

<u>Sending request to Data Source as Sql Statement:</u>In this process we send Sql Stmt's like Select, Insert, Update and Delete to the Data Source for execution or Call SP's in DB for execution, and to do that we use the class Command.

<u>Constructors of the class:</u> Command() Command(string CommandText, Connection con)

- CommandText means it can be any Sql Stmt like Select or Insert or Update or Delete or SP Name.
- Connection means the instance of the connection class which we created in the 1st step.

<u>Properties of Command Class:</u>

1. Connection: sets or gets the connection object associated with command object.
2. CommandText: sets or gets the Sql statement or SP name associated with command object.
3. CommandType: gets or sets whether command has to execute a Sql Statement [d] or Stored Procedure.

<u>The object of class Command can be created in any of the following ways:</u>
Command cmd = new Command();cmd.Connection = <con>; cmd.CommandText = "<Sql Stmt or SP Name>";
or
Command cmd = new Command("<Sql Stmt or SP Name>", <con>);

<u>Methods of Command class:</u>

1.	ExecuteReader()	->	DataReader
2.	ExecuteScalar()	->	object
3.	ExecuteNonQuery()	->	int

<u>Note:</u> after creating object of Command class we need to callany of the execute methods to execute the stmt's.

Use **ExecuteReader()** method when we want to execute a SelectStatement that returns data as rows and columns. The method returnsan object of class DataReader which holds data that isretrieved from data source in the form of rows and columns.

Use **ExecuteScalar()** method when we want to execute a SelectStatement that returns a single value result. The method returnsresult of the query in the form of an object.

Use **ExecuteNonQuery()** method when we want to execute any SQL statement other than select, like Insert or Update or Delete etc. The method returns an integerwhich tells the no. of rows affected by the Stmt.

<u>Note:</u> The above process of calling a suitable method to capture the results is our third step i.e. capturing the results given by data source.

Accessing data from a DataReader: DataReader is a class which can hold data in the form of rows and columns, and to access data from DataReader it provides the following members:

1. **GetName(int ColumnIndex) => string**

This method returns name of the column for given index position.

2. **Read() => bool**

This method moves the record pointer from the current location to next row & returns a Boolean value which tells whether the row to which it moved contains data or not, which will be true if data is present or false if not present.

3. **FieldCount => int**

This is a read-only property which returns the no. of columns DataReader contains or retrieved from the table.

4. **NextResult() => bool**

This method moves the record pointer from the current table to next table & returns a boolean value which tells whether the location to which it moved contains a table or not, which will be true if present or false if not present.

5. **GetValue(int ColoumnIndex) => object**

This method is used for retrieving column values from the row to which pointer was pointing by specifying the column index position. We can also access the row pointed by pointer by using an Indexer defined in the class either by specifying column index position or name, as following:

<DataReader>[int ColumnIndex]	**=>**	**object**
<DataReader>[string ColumnName]	**=>**	**object**

Note: To test the examples in this document first create Northwind DB under Sql Server which can be downloaded from following location: https://www.microsoft.com/en-us/download/details.aspx?id=23654, after downloading run the setup file which installs files into a folder "SQL Server 2000 Sample Databases" on "C Drive", open the folder and double click on the file "instnwnd.sql" which opens in "Sql Server Management Studio", execute it which creates Northwind DB on your Sql Server.

Now add a WebForm naming it as "Northwind_Customers_Select.aspx" & write the following code in "aspx.cs" file:

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

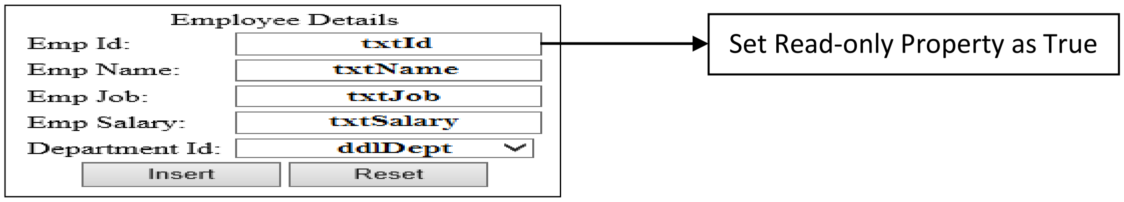
Under Page_Load:

```
SqlConnection con = new SqlConnection("Data Source=Server;Database=Northwind;User Id=Sa;Password=123");
SqlCommand cmd = new SqlCommand(
        "Select CustomerID, ContactName, City, Country, PostalCode, Phone From Customers", con);
con.Open();SqlDataReader dr = cmd.ExecuteReader();
StringBuilder sb = new StringBuilder();
sb.Append("<table border='1'><caption>Customer Details</caption><tr>");
for (int i = 0; i < dr.FieldCount;i++){
sb.Append("<th>" + dr.GetName(i) + "</th>");
}
sb.Append("</tr>");
while (dr.Read()){
sb.Append("<tr>");
 for (int i = 0; i < dr.FieldCount; i++){
sb.Append("<td>" + dr.GetValue(i)) + "</td>");
}
 sb.Append("</tr>");
}
sb.Append("</table>");con.Close(); Response.Write(sb);
```

Creating our own database on Sql Server to work with:

- Open Sql Server Management Studio and Create a new Database in it with the name *"***ASPDB**".
- Now under the DB create 2 new tables with the names Department and Employee as following:
  Create Table Department(Did Int Constraint Did_PK Primary Key Identity(10, 10), Dname Varchar(50), Location Varchar(50));
  Create Table Employee(Eid Int Constraint Eid_PK Primary Key Identity(101, 1), Ename Varchar(50), Job Varchar(50), Salary Money, Did Int Constraint Did_FK References Department(Did));
- Now insert few records into the Department table as following:
  Insert Into Department Values ('Marketing', 'Mumbai');
  Insert Into Department Values ('Sales', 'Chennai');
  Insert Into Department Values ('Accounting', 'Hyderabad');
  Insert Into Department Values ('Finance', 'Delhi');

Now add a new WebForm in the project naming it as "**ASPDB_Employee_Insert.aspx**" and design it as following:



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

Declarations:

```
SqlCommand cmd; SqlConnection con;
```

Under Page_Load:

```
con = new SqlConnection("User Id=Sa;Password=123;Database=ASPDB;Data Source=Server");
cmd = new SqlCommand(); cmd.Connection = con;
if(!IsPostBack) { LoadData();  txtName.Focus(); }
```

```
private void LoadData() {
  cmd.CommandText = "Select Did, Dname From Department Order By Did";
  con.Open(); SqlDataReader dr = cmd.ExecuteReader(); ddlDept.DataSource = dr;
  ddlDept.DataTextField = "Dname"; ddlDept.DataValueField = "Did";
  ddlDept.DataBind(); ddlDept.Items.Insert(0, "-Select Department-"); con.Close();
}
```

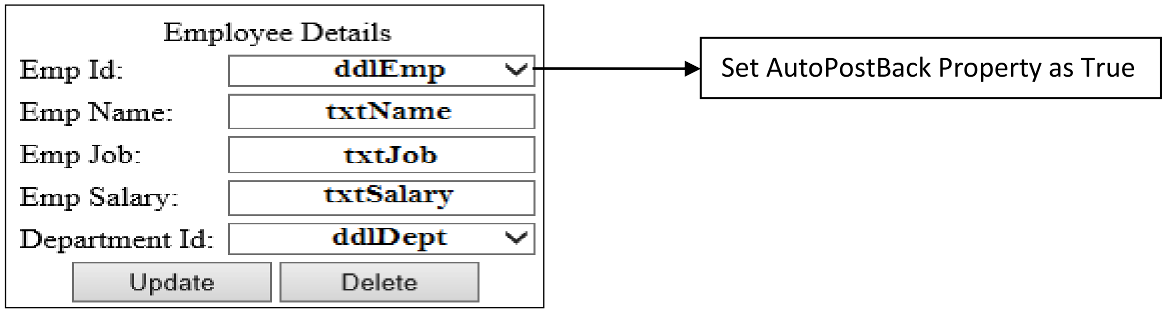
Under Insert Button Click:

```
if (ddlDept.SelectedIndex > 0){
  cmd.CommandText = $"Insert Into Employee (Ename, Job, Salary, Did) Values ('{txtName.Text}', '{txtJob.Text}',
          {txtSalary.Text}, {ddlDept.SelectedValue})";
  con.Open();
  if (cmd.ExecuteNonQuery()> 0){
    cmd.CommandText = "Select Max(Eid) From Employee"; txtId.Text = cmd.ExecuteScalar().ToString();
  }
  else { Response.Write("<script>alert('Failed inserting the record')</script>"); } con.Close();
}
else {  Response.Write("<script>alert('Please choose a department to insert.')</script>"); }
```

Under Reset Button Click:

```
txtId.Text = txtName.Text = txtJob.Text = txtSalary.Text = ""; ddlDept.SelectedIndex = 0; txtName.Focus();
```

Now add a new WebForm in the project naming it as "**ASPDB_Employee_SelectUpdateDelete.aspx**" and design it as following:



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

Declarations:

```
SqlConnection con; SqlCommand cmd; SqlDataReader dr;
```

Under Page_Load:

```
con = new SqlConnection("User Id=Sa;Password=123;Database=ASPDB;Data Source=Server");
cmd = new SqlCommand(); cmd.Connection = con;
if(!IsPostBack) { LoadEmp(); LoadDept(); }
```

```
private void LoadEmp() {
 cmd.CommandText = "Select Eid From Employee Order By Eid";
 if(con.State != ConnectionState.Open) { con.Open(); }
 dr = cmd.ExecuteReader(); ddlEmp.DataSource = dr; ddlEmp.DataTextField = "Eid";
 ddlEmp.DataValueField = "Eid"; ddlEmp.DataBind(); ddlEmp.Items.Insert(0, "-Select Employee-"); con.Close();
}
```

```
private void LoadDept() {
cmd.CommandText = "Select Did, Dname From Department Order By Did"; con.Open();
dr = cmd.ExecuteReader(); ddlDept.DataSource = dr; ddlDept.DataTextField = "Dname";
ddlDept.DataValueField = "Did"; ddlDept.DataBind(); ddlDept.Items.Insert(0, "---Select---"); con.Close();
}
```

Under ddlEmp SelectedIndexChanged Event:

```
if (ddlEmp.SelectedIndex > 0) {
 cmd.CommandText = "Select Ename, Job, Salary, Did From Employee Where Eid=" + ddlEmp.SelectedValue;
 con.Open(); dr = cmd.ExecuteReader();
 if (dr.Read()) {
  txtName.Text = dr["Ename"].ToString();txtJob.Text = dr["Job"].ToString();
  txtSalary.Text = dr["Salary"].ToString();ddlDept.SelectedValue = dr["Did"].ToString();
 }
 con.Close();
}
else { txtName.Text = txtJob.Text = txtSalary.Text = ""; ddlDept.SelectedIndex = 0; }
```

Under Update Button:

```
if(ddlEmp.SelectedIndex > 0) {
 if (ddlDept.SelectedIndex > 0) {
  cmd.CommandText = $"Update Employee Set Ename='{txtName.Text}', Job='{txtJob.Text}',
              Salary={txtSalary.Text}, Did={ddlDept.SelectedValue} Where Eid={ddlEmp.SelectedValue}";
  con.Open();
```

```
    if(cmd.ExecuteNonQuery() > 0) { Response.Write("<script>alert('Record updated successfully')</script>"); }
    else { Response.Write("<script>alert('Failed updating the record')</script>"); }
    con.Close();
  }
  else { Response.Write("<script>alert('Please select a Department for update.')</script>"); }
}
else { Response.Write("<script>alert('Please select a Employee for update.')</script>"); }
```

Under Delete Button:

```
if(ddlEmp.SelectedIndex > 0) {
  cmd.CommandText = "Delete From Employee Where Eid=" + ddlEmp.SelectedValue; con.Open();
  if (cmd.ExecuteNonQuery() > 0) {
    Response.Write("<script>alert('Record delete successfully')</script>");
    txtName.Text = txtJob.Text = txtSalary.Text = ""; ddlDept.SelectedIndex = 0; LoadEmp();
  }
  else{ Response.Write("<script>alert('Failed deleting the record')</script>"); con.Close(); }
}
else { Response.Write("<script>alert('Please select a Employee for delete')</script>"); }
```

Loading multipletables into DataReader: it's possible to Load more than 1 table into a DataReader by passing multiple select statements as parameters to Command class, separated by a semicolon. We need to use the NextResult() method of the DataReader class to navigate to the next tables after processing each table. To test this add a new WebForm under the project naming it as "Northwind_MultipleTables.aspx" and place 2 GridView controls on it and write the following code under "aspx.cs" file:

using System.Data.SqlClient;

Code under Page_Load:

```
SqlConnection con = new SqlConnection("Data Source=Server;User Id=Sa;Password=123;Database=Northwind");
SqlCommand cmd = new SqlCommand("Select CustomerId, ContactName, Address, Country, Phone From
        Customers;Select EmployeeId, LastName, FirstName, BirthDate, HireDate From Employees", con);
con.Open(); SqlDataReader dr = cmd.ExecuteReader();
GridView1.DataSource = dr; GridView1.DataBind(); dr.NextResult();
GridView2.DataSource = dr; GridView2.DataBind(); con.Close();
```

Note: GridView is a control which is used for displaying data in a table format without manually creating a table as we have done in our first example. We need to bind the table that has to be displayed in the GridView by using its DataSource property.

DataReader: it's a class designed for holding the data on client machines in the form of Rows and Columns.

Features of DataReader:

1. Faster access to data from the data source as it is connection oriented.
2. Capable of holding multiple tables in it at a time.

Drawbacks of DataReader:

1. It is connection oriented so once the connection is closed between application and data source all the data is lost, so State Mgmt is not possible with a DataReader because web application are stateless.
2. It is forward-only which allows going either to next record or table but not to previous record or table.
3. It is read-only which will not allow any changes to data that is present in it.

DataSet: it's a class which is present under System.Data namespace capable of holding Data in the form of a Table just like a DataReader.

Differences between DataReader and DataSet:

1. DataReader's work in connection oriented architecture where as DataSet's work in disconnected architecture i.e. to hold data in DataReader we need to keep the connection open whereas in case of DataSet even after closing the connection also data will be persisting so state management is possible.

2. DataReader's provide forward only access to the data whereas DataSet's provides scrollable navigation to data which allows us to move in any direction i.e. either top to bottom or bottom to top.

3. DataReader's are non-updatable where as DataSet's are updatable i.e. changes can be made to data present in DataSet and finally those changes can be sent back to Database for saving.

4. DataReader and DataSet are capable of holding multiple tables whereas in case of DataReader all those tables must be loaded only from a single Data Source but in case of DataSet each and every table can be loaded from Different Data Sources.

ADO.Net supports 2 different architectures for Data Source communication, like:

1. Connection Oriented Architecture
2. Disconnected Architecture

In the first case we require the connection to be kept open between the Web Server and Data Source for accessing the data whereas in the second case we don't require the connection to be kept open continuously between the Web Server and Data Source i.e. we need the connection to be opened only for loading the Data from Data Source but not for accessing the data.

Working with DataSet's: The class which is responsible for loading data into DataReader from a DataSource is Command and in the same way DataAdapter class is used for communication between DataSource and DataSet.

DataReader <= Command => DataSource
DataSet <=> DataAdapter <=> DataSource

Note: DataAdapter is internally a collection of 4 commands like "SelectCommand", "InsertCommand", "UpdateCommand" and "DeleteCommand" where each command is an instance of Command class, and by using these Commands DataAdapter will perform Select, Insert, Update and Delete operations on a DB Table.

Constructors of DataAdapter class:

DataAdapter()
DataAdapter(Command SelectCmd)
DataAdapter(string SelectCmdText, Connection con)
DataAdapter(string SelectCmdText, string ConnectionString)

Note: "Select Command Text" means it can be a Select Statement or a Stored Procedure which contains a Select Statement.

Instance of DataAdapter class can be created in any of the following ways:

```
Connection con = new Connection("<Connection String>");
Command cmd = new Command("<Select Stmt or SP Name>", con);
DataAdapter da = new DataAdapter();
da.SelectCommand = cmd;
```

Or

```
Connection con = new Connection("<Connection String>");
Command cmd = new Command("<Select Stmt or SP Name>", con);
DataAdapter da = new DataAdapter(cmd);
```

Or

```
Connection con = new Connection("<Connection String>");
DataAdapter da = new DataAdapter("<Select Stmt or SPName>", con);
```

or

```
DataAdapter da = new DataAdapter("<Select Stmt or SPName>", "<Connection String>");
```

Properties of DataAdapter class:

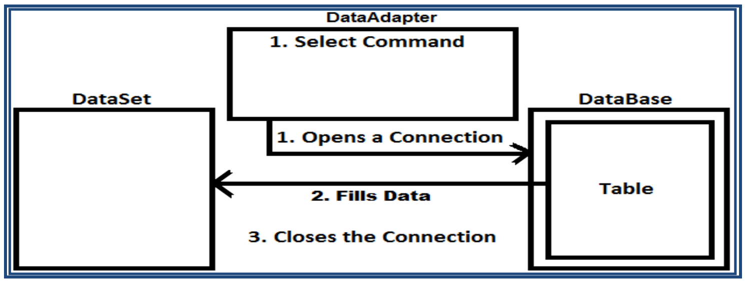
1. SelectCommand
2. InsertCommand
3. UpdateCommand
4. DeleteCommand

Methods of DataAdapter class:

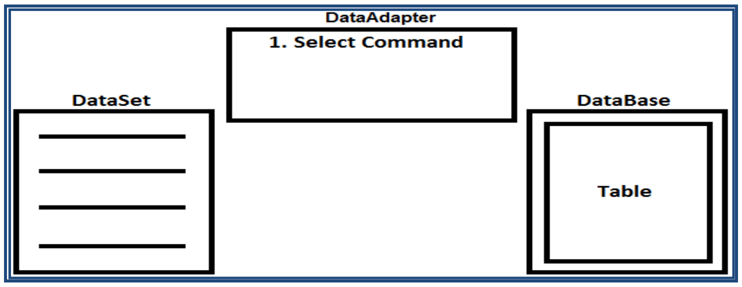
1.	Fill(DataSet ds, string tableName)	Data Source => DataAdapter => DataSet
2.	Update(DataSet ds, string tableName)	Data Source <= DataAdapter <= DataSet

When we call Fill method on DataAdapter following actions takes place internally:

1. Opens a connection with the Data Source.
2. Executes the SelectCommand present in it on DataSource and loads data from Data Source to DataSet.
3. Closes the connection with Data Source.

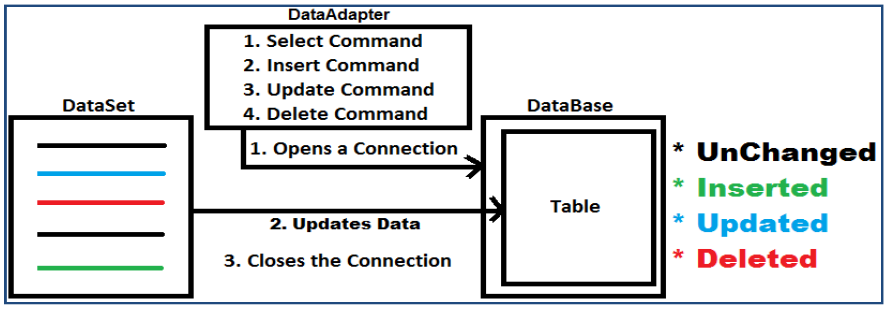


Once the execution of Fill method is completed data gets loaded into the DataSet as below:

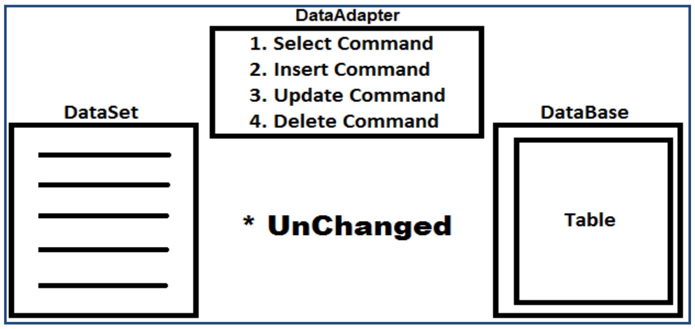


As we are discussing DataSet is updatable i.e. we can make changes to the data that is loaded into it like adding, modifying and deleting of records, and after making changes to data in DataSet if we want to send those changes back to DataSource we need to call Update method on DataAdapter, which performs the following:

1. Re-opens a connection with the Data Source.
2. Changes that are made to data present in DataSet will be sent back to corresponding Table and in this process it will make use of Insert, Update and Delete commands that are present in it.
3. Closes the connection with Data Source.



Once Update method execution is completed data gets re-loaded into DataSet as below with all unchanged rows:



<u>Accessing data from DataSet:</u> DataReader's provides pointer based access to the data, so we can get data only in a sequential order whereas DataSet provides index based access to the data, so we can get data from any location randomly.DataSet is a collection of tables where each table is represented as a class DataTable and identified by its index position or name.Every DataTable is again collection of Rows and collection of Columns where each row is represented as a class DataRow and identified by its index position and each column is represented as a class DataColumn and identified by its index position or name.

- Accessing a DataTable from DataSet: <dataset>.Tables[index] or <dataset>.Tables[name]
  E.g.: ds.Tables[0] or ds.Tables["Customer"]

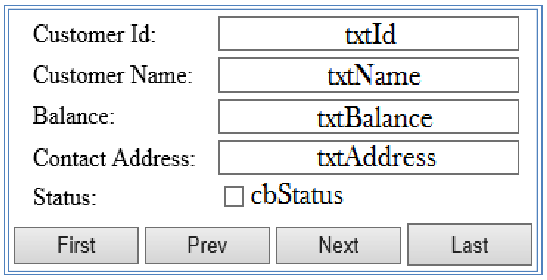
- Accessing a DataRow from DataTable: <datatable>.Rows[index]
  E.g.: ds.Tables[0].Rows[0]

- Accessing a DataColumn from DataTable: <datatable>.Columns[index] or <datatable>.Columns[name]
  E.g.: ds.Tables[0].Columns[0] or ds.Tables[0].Columns["Custid"]

- Accessing a Cell from DataTable: <datatable>.Rows[row][col]
  E.g.: ds.Tables[0].Rows[0][0] or ds.Tables[0].Rows[0]["Custid"]

To work with DataSet 1st create a table in our ASPDB with the name Customer and also insert some records into it.
Create Table Customer(Custid Int Constraint Custid_PK Primary Key, Name Varchar(50), Balance Money, Address Varchar(100), Status Bit Default 1)

Now add a new WebForm in the project naming it as "ASPDB_Customer_Select.aspx" and design it as following:



Customer Id:	txtId
Customer Name:	txtName
Balance:	txtBalance
Contact Address:	txtAddress
Status:	☐ cbStatus

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

Declarations:

```
DataSet ds; int RowIndex;
```

Under Page_Load:

```
if(Session["CDS"] == null) {
  SqlConnection con = new SqlConnection("Data Source=Server;User Id=Sa;Password=123;Database=ASPDB");
  SqlDataAdapter da = new SqlDataAdapter("Select Custid, Name, Balance, Address, Status From Customer", con);
  ds = new DataSet(); da.Fill(ds, "Customer"); ShowData(); Session["CDS"] = ds; Session["RowIndex"] = RowIndex;
}
else {
  ds = (DataSet)Session["CDS"]; RowIndex = (int)Session["RowIndex"];
}
```

```
private void ShowData() {
  txtId.Text = ds.Tables[0].Rows[RowIndex]["Custid"].ToString();
  txtName.Text = ds.Tables[0].Rows[RowIndex]["Name"].ToString();
  txtBalance.Text = ds.Tables[0].Rows[RowIndex]["Balance"].ToString();
  txtAddress.Text = ds.Tables[0].Rows[RowIndex]["Address"].ToString();
  cbStatus.Checked = Convert.ToBoolean(ds.Tables[0].Rows[RowIndex]["Status"]);
}
```

Under First Button:

```
RowIndex = 0; ShowData(); Session["RowIndex"] = RowIndex;
```

Under Previous Button:

```
if (RowIndex > 0) {
  RowIndex -= 1; ShowData(); Session["RowIndex"] = RowIndex;
}
else { Response.Write("<script>alert('First record of the table.')</script>"); }
```

Under Next Button:

```
if (RowIndex < ds.Tables[0].Rows.Count - 1) {
  RowIndex += 1; ShowData(); Session["RowIndex"] = RowIndex;
}
else { Response.Write("<script>alert('Last record of the table.')</script>"); }
```

Under Last Button:

```
RowIndex = ds.Tables[0].Rows.Count - 1; ShowData(); Session["RowIndex"] = RowIndex;
```

Storing Connection Strings in Web.config file:

Right now in our earlier examples whenever we are connecting to the database we are writing connection string, but writing the connection string each and every time is time consuming and moreover if any changes occur in the future to connection string then we need to change it in each and every Web Page.

To overcome the above problems, without writing connection string in each WebForm we need to store the connection string values in Web.config file, so that we don't require specifying the connection strings in each and every WebForm and whenever there is a change we can change it directly under the Web.config file. To store connection strings in the Web.config file we are provided with a tag <connectionStrings>which should be used under <configuration> tag.

Now open Web.config file and write the following code under the <configuration> tag:

```
<connectionStrings>
  <add name="ASPDBCS" connectionString="Data Source=Server;Database=ASPDB;User Id=Sa; Password=123"
                       providerName="System.Data.SqlClient" />
  <add name="NorthwindCS" connectionString="Data Source=Server;Database=Northwind;User Id=Sa;
                       Password=123" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Reading values from Web.config file into our application:

To read values of Web.config file in our application we need to take the help of ConfigurationManager class which is present in System.Configuration namespace.

```
string ASPDBConStr = ConfigurationManager.ConnectionStrings["ASPDBCS"].ConnectionString;
string NorthwindConStr = ConfigurationManager.ConnectionStrings["NorthwindCS"].ConnectionString;
```

From now without writing ConnectionString in our WebForms again and again we can read Connection String value from Web.config by using the above statement and use it in out code as following:

```
SqlConnection AspCon = new SqlConnection(ASPDBConStr);
SqlConnection NWCon = new SqlConnection(NorthwindConStr);
```

To still simply the process of reading connection string values into our WebForm without writing lengthy code every time, add a new class into Project naming it as ReadCS.cs and write the following code under the class:

```
using System.Configruation;
public static class ReadCS {
  public static string ASPDB {
    get { return ConfigurationManager.ConnectionStrings["ASPDBCS"].ConnectionString; }
  }
  public static string Northwind {
    get { return ConfigurationManager.ConnectionStrings["NorthwindCS"].ConnectionString; }
  }
}
```

Now in our WebForms we can read the Connection String directly by calling the properties we have defined as following:

```
SqlConnection AspCon = new SqlConnection(ReadCS.ASPDB);
SqlConnection NwCon = new SqlConnection(ReadCS.Northwind);
```

Working with GridView Control:

This control is designed to display the data on a WebForm in a table structure and this control has various features like Paging, Sorting, Customizing, Editing, etc.

Paging with GridView:

This is a mechanism of displaying data on a WebForm as “Blocks of Records”, generally used when we have huge volumes of data. To perform paging with a GridView control first we need to set “AllowPaging” property as true (default is false) and then we need to specify the no. of records that has to be displayed within each page by setting “PageSize” Property (default is 10).

PagerSettings is a property of GridView using which we need to specify the type of paging UI has to use with the help of “Mode” attribute which can be set as “NextPrevious or Numeric or NextPreviousFirstLast or NumericFirstLast” as well as here we can also specify the Position where navigation options should be present and the Text or Image for Next, Previous, First and Last buttons.

PageIndexChanging is an event under which we need to implement the logic for navigating to next pages by setting the “PageIndex” property which is “0” by default. To identify Index of the page where the user wants to navigate in runtime we can use “NewPageIndex” attribute of “PageIndexChanging” event.

To test paging with GridView add a new WebForm naming it as “Northwind_Customers_GridView_Paging.aspx”, place a GridView Control on it, set the “AllowPaging” property as true and write the below code in “aspx.cs” file:

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

Declarations: *DataSet ds;*

Under Page_Load:

```
if (Session["CustomerDS"] == null) {
  SqlDataAdapter da = new SqlDataAdapter("Select CustomerId, ContactName, ContactTitle, Country, Phone From
        Customers", ReadCS.Northwind);
  ds = new DataSet(); da.Fill(ds, "Customers"); Session["CustomerDS"] = ds; LoadData();
}
else { ds = (DataSet)Session["CustomerDS"]; }
```

```
private void LoadData() {
  GridView1.DataSource = ds; GridView1.DataBind();
}
```

Under GridView PageIndexChanging:

```
GridView1.PageIndex = e.NewPageIndex;
LoadData();
```

Sorting with GridView:

This is a mechanism of arranging the data on a WebForm in runtime ascending or descending order based on any particular column of table. To perform Sorting with GridView Control, first we need to set “AllowSorting” property as true (default is false) so that, “Column Names” in GridView looks like “Hyper Links” providing an option to click on them.

“Sorting” is an event under which we need to implement the logic that is required for sorting the data based on Selected Column which can be identified in our code by using “SortExpression” property of the event and this event fires when the Column Names (Hyper Links) are clicked by the end user.

To test sorting with GridView add a new WebForm naming it as “Northwind_Employees_GridView_Sorting.aspx”, place a GridView Control on it, set the “AllowSorting” property as true and write the below code in “aspx.cs” file:

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

Declarations: *DataSet ds;*

Under Page_Load:

```
if(Session["EmpDS"] == null) {
  SqlDataAdapter da = new SqlDataAdapter("Select EmployeeId, LastName, FirstName, BirthDate, City, Country,
                        HomePhone From Employees Order By EmployeeId", ReadCS.Northwind);
  ds = new DataSet(); da.Fill(ds, "Employees");
  Session["EmpDS"] = ds; Session["SortOrder"] = "EmployeeId Asc";
  GridView1.DataSource = ds; GridView1.DataBind();
}
else {
  ds = (DataSet)Session["EmpDS"];
}
```

Under GridView Sorting Event:

```
string[] sarr = Session["SortOrder"].ToString().Split(' ');
if (sarr[0] == e.SortExpression){
  if (sarr[1] == "Asc") {
    Session["SortOrder"] = e.SortExpression + " Desc";
  }
  else {
    Session["SortOrder"] = e.SortExpression + " Asc";
  }
}
else {
  Session["SortOrder"] = e.SortExpression + " Asc";
}
DataView dv = ds.Tables["Employees"].DefaultView;
dv.Sort = Session["SortOrder"].ToString();
GridView1.DataSource = dv; GridView1.DataBind();
```

Customizing a GridView:

Generally when we bind a DataSet to the GridView control it will automatically display all the columns that are present in the DataTable under GridView because GridView control has a mechanism of auto-generating the columns, so that column names will be displayed as header text and column values will display in the body.

If required there is a chance of customizing the GridView so that we can display different Header Text values apart from column names, define styles for Header Text, Item Text as well as we can use different controls for displaying data and even we can perform “Editing” of the data, but to do these all first we need to set “AutoGenerateColumns” property of the control as false (default is true).

Once we set AutoGenerateColumns property as false it is our responsibility to add each and every column individually to the GridView control with the help of some special fields like BoundField, ButtonField, CheckBoxField, CommandField, HyperLinkField, ImageField and TemplateField.

- **BoundField:** Displays the value of a column in a data source. This is the default column type of the GridView control.
- **ButtonField:** Displays a command button for each item in the GridView control. This enables you to create a column of custom button controls, such as the Add or the Remove button.
- **CheckBoxField:** Displays a check box for each item in the GridView control. This column field type is commonly used to display fields with a Boolean value.
- **CommandField:** Displays pre-defined command buttons to perform select, edit, or delete operations.
- **HyperLinkField:** Displays the value of a field in a data source as a hyperlink. This column field type enables you to bind a second field to the hyperlink's URL.
- **ImageField:** Displays an image for each item in the GridView control.
- **TemplateField:** Displays user-defined content for each item in the GridView control, according to a specified template. This column field type enables you to create a custom column field.

To test these add a new WebForm naming it as “ASPDB_Customer_GridView_Customize.aspx” and write the following code under <div> tag:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="false"HorizontalAlign="Center"
        Caption="Customer Details">
  <HeaderStyle BackColor="YellowGreen" ForeColor="Red" Font-Size="Larger" Font-Italic="true" />
  <RowStyle ForeColor="Wheat" BackColor="SlateGray" />
  <AlternatingRowStyle ForeColor="SlateGray" BackColor="Wheat" />
  <Columns>
   <asp:BoundField HeaderText="Customer Id" DataField="Custid" ItemStyle-HorizontalAlign="Center" />
   <asp:BoundField HeaderText="Name" DataField="Name" />
   <asp:BoundField HeaderText="Balance" DataField="Balance" ItemStyle-HorizontalAlign="Right"/>
   <asp:BoundField HeaderText="City" DataField="Address" />
   <asp:CheckBoxField HeaderText="Is-Active" DataField="Status" ItemStyle-HorizontalAlign="Center"/>
                                        or
   <asp:TemplateField HeaderText="Is-Active" ItemStyle-HorizontalAlign="Center" >
    <ItemTemplate>
     <asp:RadioButton ID="rbStatus" runat="server"Enabled="false" Checked='<%# Eval("Status") %>' />
    </ItemTemplate>
  </asp:TemplateField>
 </Columns>
</asp:GridView>
```

Now go to “aspx.cs” file and write the following code:

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

Under Page_Load:

```
SqlConnection con = new SqlConnection(ReadCS.ASPDB);
SqlCommand cmd = new SqlCommand("Select Custid, Name, Balance, Address, Status From Customer", con);
con.Open();
SqlDataReader dr = cmd.ExecuteReader();
GridView1.DataSource = dr;
GridView1.DataBind();
con.Close();
```

Data editing with GridView:

We can edit the data that is present in the GridView control and update those changes back to the corresponding table in Database. To edit the data first GridView requires an Edit and Delete buttons, and to generate them we have 3 options:

1. Set the properties AutoGenerateDeleteButton and AutoGenerateEditButton as true.
2. Add Edit & Delete options using "CommandField", by manually generating the columns as following:
3. Manual generation of those buttons by using Template Field.

```
<asp:CommandField ShowEditButton="true" ShowDeleteButton="true" />
```

To perform editing operations we need to write logic under 4 events of the GridView those are: RowEditing, RowCancelingEdit, RowUpdating and RowDeleting.

- **RowEditing:**Occurs when a row's Edit button is clicked which changes Edit button to Update and Cancel buttons and here we need to set the GridViews EditIndex to selected rows Index.
- **RowCancelingEdit:** Occurs when the Cancel button of a row in edit mode is clicked which changes back to Edit button and here we need to set the GridViews EditIndex to -1.
- **RowUpdating:**Occurs when a row's Update button is clicked and here we need to implement the logic for updating the row and finally set the GridViews EditIndex to -1.
- **RowDeleting:**Occurs when a row's Delete button is clicked and here we need to implement the logic for deleting the row.

Now to test editing with GridView add a new WebForm naming it as "ASPDB_Customer_GridView_Editing.aspx" and write the following code under <div> tag:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="false" Caption="Customer Editing"
        HorizontalAlign="Center" >
 <HeaderStyle BackColor="Yellow" ForeColor="Red" />
 <RowStyle ForeColor="Tomato" BackColor="YellowGreen" />
 <AlternatingRowStyle ForeColor="SlateGray" BackColor="Wheat" />
 <EditRowStyle BackColor="LightCoral" ForeColor="WindowFrame" />
 <Columns>
  <asp:BoundField HeaderText="Customer Id" DataField="Custid" ReadOnly="true"
        ItemStyle-HorizontalAlign="Center" />
  <asp:BoundField HeaderText="Name" DataField="Name" />
  <asp:BoundField HeaderText="Balance" DataField="Balance" ItemStyle-HorizontalAlign="Right" />
  <asp:BoundField HeaderText="City" DataField="Address" />
  <asp:CheckBoxField HeaderText="Is-Active" DataField="Status" ItemStyle-HorizontalAlign="Center”
        ReadOnly="true" />
  <asp:TemplateField HeaderText="Actions" ItemStyle-BackColor="White">
   <ItemTemplate>
    <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
    <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
        OnClientClick="return confirm('Are you sure of deleting the current record?')" />
   </ItemTemplate>
```

```
    <EditItemTemplate>
      <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
      <asp:LinkButton ID="LinkCancel" runat="server" Text="Cancel" CommandName="Cancel" />
    </EditItemTemplate>
  </asp:TemplateField>
 </Columns>
</asp:GridView>
```

Now go to Design view of the WebForm, select GridView, go to it events, double click on RowEditing, RowCancelingEdit, RowUpdating and RowDeleting events to generate associated Event Handlers (Methods) for implementing the logic and write the following code under "aspx.cs" file:

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

Declarations:

```
SqlConnection con; SqlCommand cmd;
```

Under Page_Load:

```
con = new SqlConnection(ReadCS.ASPDB);
cmd = new SqlCommand(); cmd.Connection = con;
if (!IsPostBack) { LoadData(); }
```

```
private void LoadData() {
  cmd.CommandText = "Select Custid, Name, Balance, Address, Status From Customer Where Status=1 order by
        Custid";
  if (con.State != ConnectionState.Open) { con.Open(); } SqlDataReader dr = cmd.ExecuteReader();
  GridView1.DataSource = dr;GridView1.DataBind();con.Close();
}
```

Under GridView RowEditing:

```
GridView1.EditIndex = e.NewEditIndex; LoadData();
```

Under GridView RowCancelingEdit:

```
GridView1.EditIndex = -1; LoadData();
```

Under GridView RowUpdating:

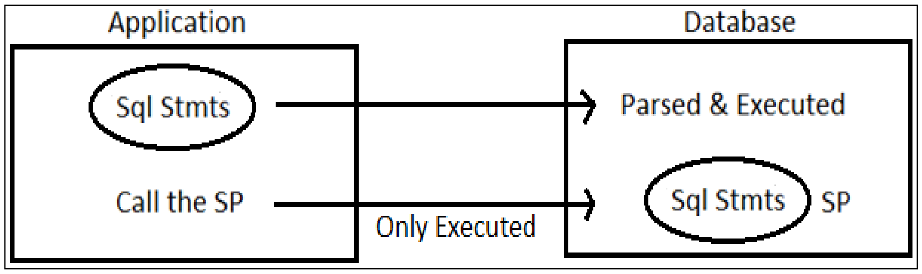
```
try {
  int Custid = int.Parse(GridView1.Rows[e.RowIndex].Cells[0].Text);
  string Name = ((TextBox)GridView1.Rows[e.RowIndex].Cells[1].Controls[0]).Text;
  decimal Balance = decimal.Parse(((TextBox)GridView1.Rows[e.RowIndex].Cells[2].Controls[0]).Text);
  string Address = ((TextBox)GridView1.Rows[e.RowIndex].Cells[3].Controls[0]).Text;
  cmd.CommandText = $"Update Customer Set Name='{Name}', Balance={Balance}, Address='{Address}' }
        Where Custid={Custid}";
  con.Open();
  if (cmd.ExecuteNonQuery() > 0) {
    GridView1.EditIndex = -1; LoadData();
  }
}
catch(Exception ex)
{ Response.Write("<script>alert('" + ex.Message + "')</script>"); }
finally { con.Close(); }
```

Under GridView RowDeleting:

```
try {
  int Custid = int.Parse(GridView1.Rows[e.RowIndex].Cells[0].Text);
  cmd.CommandText = $"Update Customer Set Status=0 Where Custid={Custid}";
  con.Open();
  if (cmd.ExecuteNonQuery() > 0) {
    LoadData();
  }
}
catch (Exception ex)
{ Response.Write("<script>alert('" + ex.Message + "')</script>"); }
finally { con.Close(); }
```

Stored Procedures

Whenever we want to interact with a database from an application we use Sql stmts. When we use Sql statements within the application we have a problem i.e. when the application runs Sql Statements will be sent to db for execution where the statements will be parsed (compiled) and then executed. The process of parsing takes place each time we run the application, because of this performance of our application decreases. To overcome the above drawback write Sql statements directly under db only, with in an object known as Stored Procedure and call them for execution. As a SP is a pre-compiled block of code that is ready for execution will directly execute the statements without parsing each time.



Syntax to define a Stored Procedure:

```
Create Procedure <Name> [ (<Parameter List>) ]
As
Begin
  <Stmts>
End;
```

- SP's are similar to a method's in our language.

  public void Test() ***//Method***

  Create Procedure Test() ***//Stored Procedure***

- If required we can also define parameters but only optional. If we want to pass parameters to a Sql Server SP prefix the special character "@" before parameter name.

  public void Test(int x) ***//CSharp***

  Create Procedure Test(@x int) ***//Sql Server***

- A SP can also return values, to return a value we use Out or Output keyword in Sql Server.

  public void Test(int x, ref|out int y) ***//CSharp***

  Create Procedure Test(@x int, @y int out|output) ***//Sql Server***

Creating a Stored Procedure:

We can create a SP in SQL Server either by using SQL Server Management Studio or Visual Studio also. To create a SP using Visual Studio open the "Server Explorer" window, this will display "ASPDB" and "Northwind" Database there and those Databases are configure based on the "Connection String" value we have used in "Web.config" file. Expand our "ASPDB" database, right click on the node Stored Procedures, select "Add New Stored Procedure" which opens a window write code in it for creating a Procedure and finally right click on the document window and select "Execute" which will create the Procedure on DB Server. Now let's create the following Stored Procedures under "ASPDB" database.

```
Create Procedure Customer_Select(@Custid Int=Null, @Status Bit=Null)
As
Begin
 If @Custid Is Null And @Status Is Null                --Fetches all records from the table
  Select Custid, Name, Balance, Address, Status From Customer Order By Custid;
 Else If @Custid Is Null And @Status Is Not Null       --Fetches records from the table based of given status
  Select Custid, Name, Balance, Address, Status From Customer Where Status=@Status Order By Custid;
 Else If @Custid Is Not Null And @Status Is Not Null   --Fetches record from the table based on given Id & Status
  Select Custid, Name, Balance, Address, Status From Customer Where Custid=@Custid And Status=@Status;
End;
```

```
Create Procedure Customer_Insert(@Name Varchar(50), @Balance Money, @Address Varchar(100), @Status Bit,
        @Custid Int Out)
As
Begin
 Begin Transaction
  Select @Custid = IsNull(Max(Custid), 100) + 1 From Customer
  Insert Into Customer (Custid, Name, Balance, Address, Status) Values (@Custid, @Name, @Balance, @Address,
        @Status);
 Commit Transaction;
End;
```

```
Create Procedure Customer_Update(@Custid Int, @Name Varchar(50), @Balance Money, @Address
Varchar(100))
As
 Update Customer Set Name=@Name, Balance=@Balance, Address=@Address Where Custid=@Custid;
```

```
Create Procedure Customer_Delete(@Custid Int)
As
  Update Customer Set Status=0 Where Custid=@Custid;
```

Calling a SP from .Net application:

To call a SP from .net application we use Command class and the process of calling will be as following:

1. Create instance of class Command by specifying SP Name as CommandText.
2. Change the CommandType property of Command as StoredProcedure because by default CommandType property is set as Text which executes Sql Stmt's and after changing that property Command can call SP's.

   cmd.CommandType = CommandType.StoredProcedure;
3. If the SP has any parameters we need to add those parameters to the Command i.e. if the parameter is input we need to add input parameters by calling "AddWithValue" method and incase if the parameter is output we need to add them by calling "Add" method of Command class.

4. If the procedure is a Query Procedure then call ExecuteReader() method of Command class which executes the procedureand loads data into DataReader, whereas if we want to load data into DataSet then create DataAdapter class instance passing Command as a parameter and call Fill Method. If the SP contains any Non-Query operations then call ExecuteNonQuery method ofCommandto execute the SP.

Adding Parameter values to Command:

SP can be defined with parameters either to send values for execution or receive values after execution. While calling a SP with parameters from .net application for each parameter of the SP we need to add a matching parameter under Command i.e. for input parameter matching input parameter has to be added and for output parameter a matching output parameter has to be added. Every parameter has 5 attributes to it like Name, Value, DbType, Size and Direction which can be Input (d) or Output.

- Name refers to name of the parameter that is defined in SP.
- Value refers to value being assigned in case of input or value we are expecting in case of output.
- DbType refers to data type of the parameter in terms of the DB where the SP exists.
- Size refers to size of data.
- Direction specifies whether parameter is Input or Output.

Based on SP's Input or Output parameters we need to add them under Command by specify following attributes:

	Input	**Output**
Name	Yes	Yes
Value	Yes	No
DbType	**Yes [*]**	Yes
Size	No	**Yes [**]**
Direction	No	Yes

*Required only if value supplied is null.
**Only in case of variable length types.

Adding input parameters under Command:
cmd.Parameters.AddWithValue(string <pname>, object <pvalue>)

Adding output parameters under Command:
cmd.Parameters.Add(string <pname>, DbType <type>[, int <size>]).Direction = ParameterDirection.Output;

Capturing output parameter values after execution of SP:
Object obj = cmd.Parameters[string <pname>].Value;
Note: ParameterDirection is an enum which contains all the directions that are supported like Input, Output, InputOutput and Returnvalue.

Now to call the above Stored Procedures in our WebForms add a new class naming it as Customer.cs and write methods with all the necessary logic for calling the SP's, so that we can call those methods directly in our WebForms without writing the logic each and every time we want to call the SP.

```
using System.Data; using System.Data.SqlClient;
public class Customer {
 SqlConnection con; SqlCommand cmd;
 public Customer() {
  con = new SqlConnection(ReadCS.ASPDB); cmd = new SqlCommand();
  cmd.Connection = con;   cmd.CommandType = CommandType.StoredProcedure;
 }
 public DataSet Customer_Select(int? Custid, bool? Status) {
  DataSet ds = new DataSet();
  try {
   cmd.CommandText = "Customer_Select"; cmd.Parameters.Clear();
   if (Custid == null && Status != null)
    cmd.Parameters.AddWithValue("@Status", Convert.ToInt32(Status));
   else if (Custid != null && Status != null) {
    cmd.Parameters.AddWithValue("@Custid", Custid);
    cmd.Parameters.AddWithValue("@Status", Convert.ToInt32(Status));
   }
   SqlDataAdapter da = new SqlDataAdapter(cmd); da.Fill(ds, "Customer");
  }
  catch(Exception ex) { throw ex; } return ds;
 }
public int Customer_Insert(string Name, decimal? Balance, string Address, bool? Status, ref int? Custid) {
  int Count = 0;
  try {
   cmd.CommandText = "Customer_Insert"; cmd.Parameters.Clear();
   cmd.Parameters.AddWithValue("@Name", Name); cmd.Parameters.AddWithValue("@Balance", Balance);
   cmd.Parameters.AddWithValue("@Address", Address); cmd.Parameters.AddWithValue("@Status", Status);
   cmd.Parameters.Add("@Custid", SqlDbType.Int).Direction = ParameterDirection.Output;
   con.Open(); Count = cmd.ExecuteNonQuery(); Custid = (int)cmd.Parameters["@Custid"].Value;
  }
  catch (Exception ex) { throw ex; } finally { con.Close(); } return Count;
 }
 public int Customer_Update(int? Custid, string Name, decimal? Balance, string Address) {
  int Count = 0;
  try {
   cmd.CommandText = "Customer_Update"; cmd.Parameters.Clear();
   cmd.Parameters.AddWithValue("@Custid", Custid); cmd.Parameters.AddWithValue("@Name", Name);
   cmd.Parameters.AddWithValue("@Balance", Balance); cmd.Parameters.AddWithValue("@Address", Address);
   con.Open(); Count = cmd.ExecuteNonQuery();
  }
  catch (Exception ex) { throw ex; } finally { con.Close(); } return Count;
 }
 public int Customer_Delete(int? Custid) {
  int Count = 0;
  try {
   cmd.CommandText = "Customer_Delete"; cmd.Parameters.Clear();
   cmd.Parameters.AddWithValue("@Custid", Custid); con.Open(); Count = cmd.ExecuteNonQuery();
  }
  catch(Exception ex) { throw ex; } finally { con.Close(); } return Count;
 }
}
```

Now we can consume the above methods we defined under Customer class to perform select, insert, update and delete operations and to test this add a new WebForm naming it as "ASPDB_Customer_GridView_Editing_SP.aspx" and design it same as "ASPDB_Customer_GridView_Editing.aspx" and write the following code in "aspx.cs" file:

Declarations:

```
Customer obj;
```

Under Page_Load:

```
obj = new Customer();
if (!IsPostBack) { LoadData(); }
```

```
private void LoadData() {
  GridView1.DataSource = obj.Customer_Select(null, true); GridView1.DataBind();
}
```

Under Page_Error:

```
Exception ex = Server.GetLastError(); Server.ClearError();
Response.Write("<script>alert('" + ex.Message + "')</script>");
```

Under GridView RowEditing:

```
GridView1.EditIndex = e.NewEditIndex; LoadData();
```

Under GridView RowCancelingEdit:

```
GridView1.EditIndex = -1; LoadData();
```

Under GridView RowUpdating:

```
int Custid = int.Parse(GridView1.Rows[e.RowIndex].Cells[0].Text);
string Name = ((TextBox)GridView1.Rows[e.RowIndex].Cells[1].Controls[0]).Text;
decimal Balance = decimal.Parse(((TextBox)GridView1.Rows[e.RowIndex].Cells[2].Controls[0]).Text);
string Address = ((TextBox)GridView1.Rows[e.RowIndex].Cells[3].Controls[0]).Text;

if (obj.Customer_Update(Custid, Name, Balance, Address) > 0) {
  GridView1.EditIndex = -1; LoadData();
}
else { Response.Write("<script>alert('Failed updating the record.')</script>"); }
```

Under GridView RowDeleting:

```
int Custid = int.Parse(GridView1.Rows[e.RowIndex].Cells[0].Text);
if (obj.Customer_Delete(Custid) > 0) { LoadData(); }
else { Response.Write("<script>alert('Failed deleting the record.')</script>"); }
```

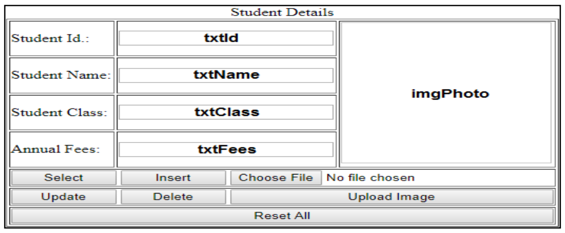
Storing Images in Database:

We can store images, audio and video data into database and while saving them we can either save path of those files in a column or content of those files in database by converting them into byte[] or binary format based on the type of application. To store byte[] or binary data, column should use Image or Varbinary data type.

Step 1: Create a table under our ASPDB database representing a Student entity as following:

Create Table Student (Sid Int Constraint Sid_Pk Primary Key, Name Varchar(50), Class Int, Fees Money, PhotoName Varchar(50), PhotoBinary VarBinary(Max), Status Bit Not Null Default 1);

Step 2: Add a WebForm in the project naming it as "ASPDB_Student_DataMgmt.aspx" and design it as following:



Step 3: Write the following code under "aspx.cs" file:
using System.IO; using System.Data; using System.Data.SqlClient;

Declarations: SqlCommand cmd; SqlConnection con;

Under Page_Load:

```
con = new SqlConnection(ReadCS.ASPDB); cmd = new SqlCommand(); cmd.Connection = con;
```

Under Upload Image Button:

```
if (FileUpload1.HasFiles) {
  HttpPostedFile selectedFile = FileUpload1.PostedFile; string fileExt = Path.GetExtension(selectedFile.FileName);
  if (fileExt == ".jpg" || fileExt == ".bmp" || fileExt == ".gif" || fileExt == ".png") {
    string imgName = selectedFile.FileName; string folderPath = Server.MapPath("~/Images/");
    if (Directory.Exists(folderPath) == false) { Directory.CreateDirectory(folderPath); }
    selectedFile.SaveAs(FolderPath + ImgName); imgPhoto.ImageUrl = "~/Images/" + ImgName;
    BinaryReader br = new BinaryReader(selectedFile.InputStream);
    byte[] imgData = br.ReadBytes(selectedFile.ContentLength);
    Session["PhotoBinary"] = imgData; Session["PhotoName"] = imgName;
  }
  else { Response.Write("<script>alert('Selected file format is not supported.')</script>"); }
}
else { Response.Write("<script>alert('Please select an image file to upload.')</script>"); }
```

```
private void ClearData(){
  txtId.Text = txtName.Text = txtClass.Text = txtFees.Text = "";
  imgPhoto.ImageUrl = ""; Session["PhotoName"] = Session["PhotoBinary"] = null; txtId.Focus();
}
```

Under Reset All Button: ClearData();

Under Select Button:

```
cmd.CommandText = "Select Sid, Name, Class, Fees, PhotoName, PhotoBinary From Student Where Status=1 And
Sid=" +   txtId.Text;
con.Open(); SqlDataReader dr = cmd.ExecuteReader();
if (dr.Read()){
  txtId.Text = dr["Sid"].ToString(); txtName.Text = dr["Name"].ToString();
  txtClass.Text = dr["Class"].ToString(); txtFees.Text = dr["Fees"].ToString();
  if (dr["PhotoName"] != DBNull.Value) {
    Session["PhotoName"] = dr["PhotoName"].ToString(); imgPhoto.ImageUrl = "~/Images/" + dr["PhotoName"];
  }
  else{ Session["PhotoName"] = null; imgPhoto.ImageUrl = ""; }
  if (dr["PhotoBinary"] != DBNull.Value) { Session["PhotoBinary"] = (byte[])dr["PhotoBinary"]; }
  else { Session["PhotoBinary"] = null; }
}
else { Response.Write("<script>alert('No student exists with given ID.')</script>"); ClearData(); } con.Close();
```

```
private void AddParameters() {
  cmd.Parameters.AddWithValue("@Sid", txtId.Text); cmd.Parameters.AddWithValue("@Name", txtName.Text);
  cmd.Parameters.AddWithValue("@Class", txtClass.Text); cmd.Parameters.AddWithValue("@Fees", txtFees.Text);
  if (Session["PhotoName"] != null) {
    cmd.Parameters.AddWithValue("@PhotoName", Session["PhotoName"].ToString);
  }
  else {
    cmd.Parameters.AddWithValue("@PhotoName", DBNull.Value);
    cmd.Parameters["@PhotoName"].SqlDbType = SqlDbType.VarChar;
  }
  if (Session["PhotoBinary"] != null){
    cmd.Parameters.AddWithValue("@PhotoBinary", (byte[])Session["PhotoBinary"]);
  }
  else {
    cmd.Parameters.AddWithValue("@PhotoBinary", DBNull.Value);
    cmd.Parameters["@PhotoBinary"].SqlDbType = SqlDbType.Image;
  }
}
```

Under Insert Button:

```
try {
  cmd.CommandText = "Insert Into Student(Sid, Name, Class, Fees, PhotoName, PhotoBinary) Values (@Sid,
              @Name, @Class, @Fees, @PhotoName, @PhotoBinary)";
  AddParameters(); con.Open(); cmd.ExecuteNonQuery();
  Response.Write("<script>alert('Record inserted into the table.')</script>");
}
catch(Exception ex) { Response.Write("<script>alert('" + ex.Message + "')</script>"); }
finally { con.Close(); }
```

Under Update Button:

```
try {
  cmd.CommandText = "Update Student Set Name=@Name, Class=@Class, Fees=@Fees,
                              PhotoName=@PhotoName, PhotoBinary=@PhotoBinary Where Sid=@Sid";
  AddParameters(); con.Open(); cmd.ExecuteNonQuery();
  Response.Write("<script>alert('Record updated in the table.')</script>");
}
catch (Exception ex) { Response.Write("<script>alert('" + ex.Message + "')</script>"); }
finally { con.Close(); }
```

Under Delete Button:

```
try{
  cmd.CommandText = "Update Student Set Status=0 Where Sid=" + txtId.Text;
  con.Open(); cmd.ExecuteNonQuery();
  ClearData();
  Response.Write("<script>alert('Record deleted from the table.')</script>");
}
catch (Exception ex) { Response.Write("<script>alert('" + ex.Message + "')</script>"); }
finally { con.Close(); }
```

Repeater Control:

This control is a container control that allows you to create custom lists out of any data that is available to the page which can be loaded from DB, XML File or any other source of data. The Repeater control does not have a built-in rendering of its own i.e. doesn't have any generation of columns like a GridView, which means that you must provide the layout for the Repeater control by using templates. When the page runs the Repeater control loops through the records in the data source and renders an item for each record. To use the Repeater control, we must create templates that define the layout of the control's content. Templates can contain any combination of markup and controls. If no templates are defined, or if none of the templates contain elements, the control does not appear on the page when the application is run. Repeater control supports the following templates:

1. **HeaderTemplate:** contains the text and controls to render at the beginning of the list.
2. **ItemTemplate:** contains the HTML elements and controls to render once for each data item loaded from data source.
3. **AlternatingItemTemplate:** contains the HTML elements and controls to render once for every other data item loaded from data source. Typically, this template is used to create a different look for the alternating items, such as a different background color than the color that is specified in the ItemTemplate.
4. **SeparatorTemplate:** contains the elements to render between each item. A typical example might be a line or break (using an <hr /> or <br /> elements).
5. **FooterTemplate:** contains the text and controls to render at the end of the list.

To work with Repeater control, add a new WebForm naming it as "ASPDB_Customer_Repeater_Demo1.aspx" and write the following code under <div> tag:

```
<asp:Repeater ID="Repeater1" runat="server">
  <HeaderTemplate>
    <table border="1" align="center"><caption>Customer Details</caption>
      <tr>
        <th>Customer Id</th><th>Name</th><th>Balance</th><th>Address</th><th>Status</th>
      </tr>
  </HeaderTemplate>
  <ItemTemplate>
    <tr style="background-color:aqua">
      <td align="center"><%# Eval("Custid") %></td>
      <td><%# Eval("Name") %></td><td><%# Eval("Balance") %></td><td><%# Eval("Address") %></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Checked='<%# Eval("Status")%>'
                          Enabled="false" /></td>
    </tr>
  </ItemTemplate>
  <AlternatingItemTemplate>
    <tr style="background-color:bisque">
      <td align="center"><%# Eval("Custid") %></td>
      <td><%# Eval("Name") %></td><td><%# Eval("Balance") %></td><td><%# Eval("Address") %></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Checked='<%# Eval("Status")%>'
                          Enabled="false" /></td>
    </tr>
  </AlternatingItemTemplate>
  <FooterTemplate></table></FooterTemplate>
</asp:Repeater>
```

Now go to "aspx.cs" file and write the following code in Page_Load method:

```
Customer obj = new Customer(); Repeater1.DataSource = obj.Customer_Select(null, null); Repeater1.DataBind();
```

Now add another WebForm naming it as "ASPDB_Customer_Repeater_Demo2.aspx" and write the following code under <div> tag which will now display the data in a List format:

```
<asp:Repeater ID="Repeater1" runat="server">
  <HeaderTemplate>
    <h1 style="background-color:gray;color:red;text-decoration:underline;text-align:center">Customer Details</h1>
  </HeaderTemplate>
  <ItemTemplate>
    Customer <%# Eval("Custid") %>
    <ul style="background-color:aliceblue">
      <li>Name: <%# Eval("Name") %></li>
      <li>Balance: <%# Eval("Balance") %></li>
      <li>City: <%# Eval("Address") %></li>
      <li>Is-Active: <%# Eval("Status") %></li>
    </ul>
  </ItemTemplate>
  <AlternatingItemTemplate>
    Customer <%# Eval("Custid") %>
    <ul style="background-color:bisque">
      <li>Name: <%# Eval("Name") %></li>
      <li>Balance: <%# Eval("Balance") %></li>
      <li>City: <%# Eval("Address") %></li>
      <li>Is-Active: <%# Eval("Status") %></li>
    </ul>
  </AlternatingItemTemplate>
  <SeparatorTemplate><hr /></SeparatorTemplate>
  <FooterTemplate>
    <h3 style="background-color:yellow;color:red;text-align:center">End of Customer Data</h3>
  </FooterTemplate>
</asp:Repeater>
```

Now go to "aspx.cs" file and write the following code in Page_Load method:

```
Customer obj = new Customer();
Repeater1.DataSource = obj.Customer_Select(null, null); Repeater1.DataBind();
```

Displaying data in MasterDetail format using Nested Repeaters:

We can display data in master/detail format with the help of Repeater's, by nesting them. To do this let us use Categories and Products tables of Northwind Database. Add a new WebForm naming it as "Northwind_Categories_Products_Repeater.aspx" and write the following code under <div> tag:

```
<asp:Repeater ID="rptCategories" runat="server">
  <HeaderTemplate>
    <h1 style="background-color: yellow; color: red; text-decoration: underline; text-align: center">
      Categories-Product Details</h1>
  </HeaderTemplate>
```

```
<ItemTemplate>
  <div style="background-color: aquamarine">
    <b>Category Id:</b> <asp:Label ID="lblCatId" runat="server" Text='<%# Eval("CategoryID") %>' /> <br />
    <b>Category Name:</b> <%# Eval("CategoryName") %> <br />
    <b>Description:</b> <%# Eval("Description") %>
  </div>
  <asp:Repeater ID="rptProducts" runat="server">
    <HeaderTemplate>
      <table border="1">
        <tr>
          <th>Product-ID</th>               <th>Product Name</th>                    <th>Supplier-ID</th>
          <th>Quantity Per Unit</th>        <th>Unit Price</th>                      <th>Units In Stock</th>
        </tr>
    </HeaderTemplate>
    <ItemTemplate>
      <tr style="background-color: aliceblue">
        <td><%# Eval("ProductId") %></td>                <td><%# Eval("ProductName") %></td>
        <td><%# Eval("SupplierId") %></td>               <td><%# Eval("QuantityPerUnit") %></td>
        <td><%# Eval("UnitPrice") %></td>                <td><%# Eval("UnitsInStock") %></td>
      </tr>
    </ItemTemplate>
    <AlternatingItemTemplate>
      <tr style="background-color: bisque">
        <td><%# Eval("ProductId") %></td>                <td><%# Eval("ProductName") %></td>
        <td><%# Eval("SupplierId") %></td>               <td><%# Eval("QuantityPerUnit") %></td>
        <td><%# Eval("UnitPrice") %></td>                <td><%# Eval("UnitsInStock") %></td>
      </tr>
    </AlternatingItemTemplate>
    <FooterTemplate></table></FooterTemplate>
  </asp:Repeater>
</ItemTemplate>
<SeparatorTemplate><hr /></SeparatorTemplate>
<FooterTemplate>
  <h1 style="background-color: yellow; color: red; text-align: center">End of Categories-Product data</h1>
</FooterTemplate>
</asp:Repeater>
```

Now write the following code in “aspx.cs” file:

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

<u>Declarations:</u>

DataSet ds; SqlDataAdapter da;

<u>Under Page_Load:</u>

da = new SqlDataAdapter("Select CategoryId, CategoryName, Description From Categories Order By CategoryId", ReadCS.Northwind);
ds = new DataSet(); da.Fill(ds, "Categories"); rptCategories.DataSource = ds; rptCategories.DataBind();

Under rptCategories ItemDataBound Event:

```
if (e.Item.ItemType != ListItemType.Header && e.Item.ItemType != ListItemType.Separator
                        && e.Item.ItemType != ListItemType.Footer)
{
 Label lblCatId = (Label)e.Item.FindControl("lblCatId");
 da.SelectCommand.CommandText = "Select ProductID, ProductName, SupplierId, QuantityPerUnit, UnitPrice,
            UnitsInStock From Products Where CategoryID=" + lblCatId.Text;
 ds = new DataSet(); da.Fill(ds, "Products");
 Repeater rptProducts = (Repeater)e.Item.FindControl("rptProducts");
 rptProducts.DataSource = ds; rptProducts.DataBind();
}
```

DataList Control:

The DataList control displays data in a format that you can define using templates and styles like a Repeater control. The DataList control is useful for displaying data in any repeating structure, such as a table. The DataList control can display rows in different layouts, such as ordering them in columns or rows or rows & columns. Optionally, you can configure the DataList control to allow users to edit or delete information. You can also customize the control to support other functionality, such as selecting rows. DataList control support the following Templates and Styles for displaying data as following:

Templates	Styles
HeaderTemplate	HeaderStyle
FooterTemplate	FooterStyle
ItemTemplate	ItemStyle
AlternatingItemTemplate	AlternatingItemStyle
SelectedItemTemplate	SelectedItemStyle
EditItemTemplate	EditItemStyle
SeparatorTemplate	SeparatorStyle

SelectedItemTemplate: Used to render the selected item; the selected item is the item whose index corresponds to the DataList's SelectedIndex property.

EditItem Template: Used to render the item being edited and provide editable controls for editing the row.

Add a WebForm naming it as "ASPDB_Student_DataList_Demo.aspx" and write the following code under <div> tag:

```
<asp:DataList ID="DataList1" runat="server" RepeatDirection="Vertical" RepeatColumns="2" GridLines="Both"
        Width="100%">
 <HeaderStyle BackColor="YellowGreen" ForeColor="Red" Font-Size="XX-Large" HorizontalAlign="Center" />
 <ItemStyle BackColor="Wheat" /><AlternatingItemStyle BackColor="SteelBlue" />
 <HeaderTemplate>Student Details</HeaderTemplate>
 <ItemTemplate>
  <table width="100%" border="1">
   <tr>
    <td rowspan="5" width ="200px"><asp:Image ID="imgPhoto" runat="server" Width="200" Height="200"
        ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' /></td>
    <td><b>Sno:</b><%# Eval("Sid") %></td>
   </tr>
   <tr><td><b>Sname:</b><%# Eval("Name") %></td></tr>
   <tr><td><b>Class:</b><%# Eval("Class") %></td></tr>
   <tr><td><b>Fees:</b><%# Eval("Fees") %></td></tr>
```

```
    <tr><td><b>Status:</b><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
            Checked='<%# Eval("Status") %>' /></td></tr>
  </table>
 </ItemTemplate>
</asp:DataList>
```

Note: In the above case we are displaying photo in Image Control with the help of its path that's stored in DB where as if we want to display image based on the byte[] stored in DB table we need to write code as following:

```
<asp:Image ID="imgPhoto" runat="server" Width="200" Height="200"
        ImageUrl='<%# "data:image/jpeg;base64," + Convert.ToBase64String((byte[])Eval("PhotoBinary"))%>' />
```

Now write the following code in "aspx.cs" file:

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

Under Page_Load:

```
SqlConnection con = new SqlConnection(ReadCS.ASP7DB);
SqlCommand cmd = new SqlCommand("Select Sid, Name, Class, Fees, PhotoName, PhotoBinary, Status From
        Student Order By Sid", con);con.Open();
SqlDataReader dr = cmd.ExecuteReader();DataList1.DataSource = dr;DataList1.DataBind(); con.Close();
```

Performing Editing operations by using DataList Control: Add a new WebForm naming it as "ASPDB_Customer_DataList_Editing.aspx" and write the following code under its <div> tag:

```
<asp:DataList ID="DataList1" runat="server" RepeatDirection="Horizontal" RepeatColumns="3" GridLines="Both"
        Width="100%" DataKeyField="Custid" >
 <HeaderStyle BackColor="YellowGreen" ForeColor="Red" Font-Size="XX-Large" HorizontalAlign="Center" />
 <ItemStyle BackColor="Azure" /><AlternatingItemStyle BackColor="Tan"/>
 <EditItemStyle BackColor="Turquoise"/><HeaderTemplate>Customer Data Editing</HeaderTemplate>
 <ItemTemplate>
  <table border="1" width="100%">
    <tr><td>Custid:</td><td><%# Eval("Custid") %></td></tr>
    <tr><td>Name:</td><td><%# Eval("Name") %></td></tr>
    <tr><td>Balance:</td><td><%# Eval("Balance") %></td></tr>
    <tr><td>Address:</td><td><%# Eval("Address") %></td></tr>
    <tr><td>Status:</td>
     <td><asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status") %>' /></td>
    </tr>
    <tr><td colspan="2" align="center" style="background-color:white">
     <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
     <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
        OnClientClick="return confirm('Are you sure of deleting the current record?')" /></td></tr>
  </table>
 </ItemTemplate>
 <EditItemTemplate>
  <table border="1" width="100%">
    <tr><td>Custid:</td><td><%# Eval("Custid") %></td></tr>
    <tr><td>Name:</td><td><asp:TextBox ID="txtName" runat="server" Text='<%# Eval("Name") %>' /></td></tr>
    <tr><td>Balance:</td><td><asp:TextBox ID="txtBalance" runat="server" Text='<%# Eval("Balance") %>' /></td>
    </tr>
    <tr><td>Address:</td><td><asp:TextBox ID="txtAddress" runat="server" Text='<%# Eval("Address") %>' /></td>
    </tr>
    <tr><td>Status:</th>
        <td><asp:CheckBox ID="cbStatus" runat="server" Enabled="false" Checked='<%# Eval("Status") %>' /></td>
    </tr>
    <tr><td colspan="2" align="center" style="background-color:white">
     <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
     <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" /></td></tr>
   </table>
  </EditItemTemplate>
</asp:DataList>
```

Note: DataKeyField property is used for specifying Key Column of the table which is used in update and delete.

To perform updations with DataList control we need to write code under the following events:

1. **EditCommand:** Occurs when the Edit button is clicked for an item in the DataList control.
2. **CancelCommand:** Occurs when the Cancel button is clicked for an item in the DataList control.
3. **UpdateCommand:** Occurs when the Update button is clicked for an item in the DataList control.
4. **DeleteCommand:** Occurs when the Delete button is clicked for an item in the DataList control.

Now go to "aspx.cs" file and write the following code:

Declarations: Customer obj;

Under Page_Load:

```
obj = new Customer(); if (!IsPostBack) { LoadData(); }
```

private void LoadData() *{*

```
DataList1.DataSource = obj.Customer_Select(null, true); DataList1.DataBind();
}
```

Under DataList EditCommand: *DataList1.EditItemIndex = e.Item.ItemIndex; LoadData();*

Under DataList CancelCommand: *DataList1.EditItemIndex = -1; LoadData();*

Under DataList UpdateCommand:

```
int Custid = Convert.ToInt32(DataList1.DataKeys[e.Item.ItemIndex]);
string Name = ((TextBox)e.Item.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)e.Item.FindControl("txtBalance")).Text);
string Address = ((TextBox)e.Item.FindControl("txtAddress")).Text;
if (obj.Customer_Update(Custid, Name, Balance, Address) > 0) {
  DataList1.EditItemIndex = -1; LoadData();
}
else { Response.Write("<script>alert('Failed updating record in the table.')</script>"); }
```

Under DataList DeleteCommand:

```
int Custid = Convert.ToInt32(DataList1.DataKeys[e.Item.ItemIndex]);
if(obj.Customer_Delete(Custid) > 0) {
  Response.Write("<script>alert('Record deleted from the table.')</script>"); LoadData();
}
else { Response.Write("<script>alert('Failed deleting record from the table.')</script>"); }
```

DetailsView Control:

This control displays a single record from a data source, where each data row represents a field in the record. It is often used in combination with a GridView control for master/detail scenarios. This control gives you the ability to display, edit, insert, or delete a single record at a time from its associated data source. This control does not support sorting.

By default the controls AutoGenerateRows property is set as true so when we bind a DataTable to the control it will by default display first record of the table. The control provides user interface for navigating between data records and to enable paging behavior, set the AllowPaging property to true and then write code under "PageIndexChanging" event for navigating to next records.

To test working with DetailView control add a new WebForm in the project naming it as "ASPDB_Student_DetailsView_Demo.aspx", place a DetailsView control on it and write the following code under "aspx.cs" file:

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

Under Page_Load: *if (!IsPostBack) { LoadData(); }*

```
private void LoadData() {
  SqlDataAdapter da = new SqlDataAdapter("Select Sid, Name, Class, Fees, PhotoName, Status From Student Order
        By Sid", ReadCS.ASPDB);
  DataSet ds = new DataSet(); da.Fill(ds, "Student"); DetailsView1.DataSource = ds; DetailsView1.DataBind();
}
```

Now when we run the WebForm by default it displays the first record of the table only without any paging option, so to enable paging go to design view and set the AllowPaging property of DetailsView control as true which displays pager below the control. We can use PagerSettings property to specify the type of paging we want. After enabling the paging, go to events of the DetailsView control double click on PageIndexChanging event and write the following code:

```
DetailsView1.PageIndex = e.NewPageIndex; LoadData();
```

Right now in our above example we will not see the Photograph of the student but we see the name of the image because in our table it is a Varchar column and treated same like other columns (Sid, Name, etc.). So if we want the image to be displayed over there we need to explicitly load the image by placing Image Control and to do that we need to generate rows manually without using auto generation of rows. To test the process set the AutoGenerateRows property of the control as false and write the following code under DetailsView control:

```
<Fields>
  <asp:BoundField HeaderText="Sid:" DataField="Sid" />
  <asp:BoundField HeaderText="Name:" DataField="Name" />
  <asp:BoundField HeaderText="Class:" DataField="Class" />
  <asp:BoundField HeaderText="Fees:" DataField="Fees" />
  <asp:CheckBoxField HeaderText="Status:" DataField="Status" />
  <asp:TemplateField HeaderText="Student Photo:">
    <ItemTemplate>
     <asp:Image ID="imgPhoto" runat="server" Width="200" Height="200"
          ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' />
    </ItemTemplate>
  </asp:TemplateField>
</Fields>
```

Displaying data in Master/Detail format by using GridView and DetailsView Controls: Add a new WebForm under the project naming it as "ASPDB_Dept_Emp_Grid_Details.aspx" and write the following code under <div> tag:

```
<table>
 <tr>
  <td>
   <asp:GridView ID="GridView1" runat="server" DataKeyNames="Did" AutoGenerateColumns="false">
    <Columns>
     <asp:BoundField HeaderText="Did" DataField="Did" />
     <asp:BoundField HeaderText="Name" DataField="Dname" />
     <asp:BoundField HeaderText="Location" DataField="Location" />
     <asp:CommandField ShowSelectButton="true" SelectText="View Details" />
    </Columns>
   </asp:GridView>
  </td>
  <td>
   <asp:DetailsView ID="DetailsView1" runat="server" AllowPaging="true" AutoGenerateRows ="false" >
    <Fields>
     <asp:BoundField HeaderText="Eid" DataField="Eid" />
     <asp:BoundField HeaderText="Name" DataField="Ename" />
     <asp:BoundField HeaderText="Job" DataField="Job" />
     <asp:BoundField HeaderText="Salary" DataField="Salary" />
     <asp:BoundField HeaderText="Did" DataField="Did" />
    </Fields>
   </asp:DetailsView>
  </td>
 </tr>
</table>
```

When we run the above form GridView displays details of Departments and every row in the GridView contains a Select Button with the caption as "View Details" and when we click on that button it raises an event "SelectedIndexChanging", so under that event we can identify the row that has been selected and access the key value i.e. Did by using DataKey property of GridView and display details of Employees working in that department under DetailsView control.

Now go to Design View of the WebForm, double click on SelectedIndexChanging event of GridView and also on PageIndexChanging event of DetailsView control and write the following code under "aspx.cs" file:

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

Declarations: *DataSet ds; SqlDataAdapter da; SqlConnection con;*

Under Page_Load:

```
con = new SqlConnection(ReadCS.ASPDB);
if (!IsPostBack) { LoadDept(); LoadEmp(0); ViewState["Did"] = 0;}
```

```
private void LoadDept() {
 da = new SqlDataAdapter("Select Did, Dname, Location From Department Order By Did", con);
 ds = new DataSet(); da.Fill(ds, "Department"); GridView1.DataSource = ds; GridView1.DataBind();
}
```

```
private void LoadEmp(int Did) {
  if (Did == 0)
    { da = new SqlDataAdapter("Select Eid, Ename, Job, Salary,Did From Employee", con); }
  else
    { da = new SqlDataAdapter("Select Eid, Ename, Job, Salary,Did From Employee Where Did=" + Did, con); }
  ds = new DataSet(); da.Fill(ds, "Employee"); DetailsView1.DataSource = ds; DetailsView1.DataBind();
}
```

<u>Under GridView SelectedIndexChanging:</u>

```
ViewState["Did"] = GridView1.DataKeys[e.NewSelectedIndex].Value;
DetailsView1.PageIndex = 0; LoadEmp((int)ViewState["Did"]);
```

<u>Under DetailsView PageIndexChanging:</u>

```
DetailsView1.PageIndex = e.NewPageIndex; LoadEmp((int)ViewState["Did"]);
```

<u>Performing editing operations using DetailsView control:</u>

To perform editing operations by using DetailsView first we need to provide Insert, Update and Delete buttons for the control either by setting the properties "ShowInsertButton", "ShowUpdateButton" and "ShowDeleteButton" as true in property window or we can do that manually by using "CommandFields" or "TemplateFields".

To do these add a new WebForm under the project naming it as "ASPDB_Customer_DetailsView_Editing.aspx" and write the following code under <div> tag:

```
<asp:DetailsView ID="DetailsView1" runat="server" AllowPaging="true" AutoGenerateRows="false"
        Caption="Customer Details" HorizontalAlign="Center" DataKeyNames="Custid" >
  <Fields>
    <asp:TemplateField HeaderText="Custid">
      <ItemTemplate>
        <asp:Label ID="lblCustid" runat="server" Text='<%# Eval("Custid") %>' />
      </ItemTemplate>
    </asp:TemplateField>
    <asp:BoundField HeaderText="Name" DataField="Name" />
    <asp:BoundField HeaderText="Balance" DataField="Balance" />
    <asp:BoundField HeaderText="Address" DataField="Address" />
    <asp:CheckBoxField HeaderText="Status" DataField="Status" ReadOnly="true" />
    <asp:CommandField ShowInsertButton="true" ShowEditButton="true" />
    <asp:TemplateField>
      <ItemTemplate>
        <asp:LinkButton ID="bthDelete" runat="server" CommandName="Delete" Text="Delete"
          OnClientClick="return confirm('Are you sure of deleting the current record?')" />
      </ItemTemplate>
    </asp:TemplateField>
  </Fields>
</asp:DetailsView>
```

Now we need to write the logic under the following events of the control:

1. **ModeChanging:** fires when we click on the "New, Edit and Cancel" buttons and under this we need to change Mode of the control to "Insert Mode", "Edit Mode" and "ReadOnly Mode" respectively and all these 3 modes are defined under an enum "DetailsViewMode".

2. **ItemInserting:** fires when we click on Insert button and here we need to implement logic for insert.
3. **ItemUpdating:** fires when we click on Update button and here we need to implement logic for update.
4. **ItemDeleting:** fires when we click on Delete button and here we need to implement logic for delete.

Now go to design view of the WebForm and double click on PageIndexChanging, ModeChanging, ItemInserting, ItemUpdating and ItemDeleting events of DetailsView control to generate methods for implementing the logic and write the following code under "aspx.cs" file:

Declarations: *Customer obj;*

Under Page_Load: *obj = new Customer(); if (!IsPostBack) { LoadData(); }*

```
private void LoadData() {
DetailsView1.DataSource = obj.Customer_Select(null, true); DetailsView1.DataBind();
}
```

Under DetailsView PageIndexChanging:

```
DetailsView1.PageIndex = e.NewPageIndex; LoadData();
```

Under DetailsView ModeChanging:

```
if (e.NewMode == DetailsViewMode.Insert) { DetailsView1.ChangeMode(DetailsViewMode.Insert); }
else if (e.NewMode == DetailsViewMode.Edit) { DetailsView1.ChangeMode(DetailsViewMode.Edit); LoadData(); }
else { DetailsView1.ChangeMode(DetailsViewMode.ReadOnly); LoadData();  }
```

Under DetailsView ItemInserting:

```
string Name = ((TextBox)DetailsView1.Rows[1].Cells[1].Controls[0]).Text;
decimal Balance = decimal.Parse(((TextBox)DetailsView1.Rows[2].Cells[1].Controls[0]).Text);
string Address = ((TextBox)DetailsView1.Rows[3].Cells[1].Controls[0]).Text;
bool Status = ((CheckBox)DetailsView1.Rows[4].Cells[1].Controls[0]).Checked;

int? Custid = null;
if (obj.Customer_Insert(Name, Balance, Address, Status, ref Custid) > 0) {
  DetailsView1.ChangeMode(DetailsViewMode.ReadOnly); LoadData();
}
else { Response.Write("<script>alert('Failed inserting record into the table.')</script>"); }
```

Under DetailsView ItemUpdating:

```
int Custid = DetailsView1.DataKey.Value;
string Name = ((TextBox)DetailsView1.Rows[1].Cells[1].Controls[0]).Text;
decimal Balance = decimal.Parse(((TextBox)DetailsView1.Rows[2].Cells[1].Controls[0]).Text);
string Address = ((TextBox)DetailsView1.Rows[3].Cells[1].Controls[0]).Text;

if (obj.Customer_Update(Custid, Name, Balance, Address) > 0) {
  DetailsView1.ChangeMode(DetailsViewMode.ReadOnly); LoadData();
}
else { Response.Write("<script>alert('Failed updating record in the table.')</script>"); }
```

Under DetailsView ItemDeleting:

```
int Custid = DetailsView1.DataKey.Value;
if(obj.Customer_Delete(Custid) > 0) {
  LoadData();
}
else { Response.Write("<script>alert('Failed deleting record from the table.')</script>"); }
```

FormView Control:

The FormView control lets you work with a single record from a data source, similar to the DetailsView control. The difference between the FormView and the DetailsView controls is that the DetailsView control uses a tabular layout where each field of the record is displayed as a row of its own. In contrast, the FormView control does not specify a pre-defined layout for displaying the record. Instead, you create a template that contains controls to display individual fields from the record.

To work with FormView control, add a new WebForm naming it as "ASPDB_Student_FormView_Demo.aspx" and write the following code under <div> tag:1

```
<asp:FormView ID="FormView1" runat="server" AllowPaging="True" HorizontalAlign="Center"
        HeaderStyle-HorizontalAlign="Center" PagerStyle-HorizontalAlign="Center">
 <HeaderTemplate>Student Details</HeaderTemplate>
 <HeaderStyle BackColor="Yellow" ForeColor="Red" Font-Size="XX-Large" />
 <FooterStyle BackColor="SlateBlue" ForeColor="Tan" Font-Size="Large" />
 <ItemTemplate>
  <table border="1">
   <tr>
    <th>Sid</th><th>Name</th><th>Class</th><th>Fees</th><th>Status</th><th>Photo</th>
   </tr>
   <tr>
    <td><%# Eval("Sid") %></td><td><%# Eval("Name") %></td>
    <td><%# Eval("Class") %></td><td><%# Eval("Fees") %></td>
    <td align="Center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                Checked='<%# Eval("Status") %>' /></td>
    <td><asp:Image ID="imgPhoto" runat="server" Width="200" Height="200"
                ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' /></td>
   </tr>
  </table>
 </ItemTemplate>
 <FooterTemplate>Click to navigate:</FooterTemplate>
</asp:FormView>
```

Note: now go to design view of the WebForm and double click on PageIndexChanging event to generate an associated method to implement the logic for navigating to next and previous records.

Now go to "aspx.cs" file and write the following code:

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

Under Page_Load:

```
if (!IsPostBack) { LoadData(); }
```

```
private void LoadData() {
 SqlDataAdapter da = new SqlDataAdapter(
        "Select Sid, Name, Class, Fees, PhotoName, Status From Student Order By Sid", ReadCS.ASPDB);
 DataSet ds = new DataSet(); da.Fill(ds, "Student"); FormView1.DataSource = ds; FormView1.DataBind();
}
```

Under FormView PageIndexChanging:

```
FormView1.PageIndex = e.NewPageIndex; LoadData();
```

<u>Performing editing operations using FormView control:</u>

To perform editing operations by using FormView first we need to explicitly generate the text and controls that has to be rendered in case of Insert and Edit by using ItemTemplate, InsertItemTemplate and EditItemTemplates. To test this process add a WebForm under the project naming it as “ASPDB_Customer_FormView_Editing.aspx” and write the following under <div> tag:

```
<asp:FormView ID="FormView1" runat="server" DataKeyNames="Custid" AllowPaging="true">
 <HeaderStyle BackColor="Yellow" ForeColor="Red" HorizontalAlign="Center" Font-Size="XX-Large" />
 <FooterStyle BackColor="SlateBlue" ForeColor="Tan" Font-Size="Large" />
 <HeaderTemplate>Customer Details</HeaderTemplate>
 <ItemTemplate>
  <table border="1">
   <tr><th>Custid</th><th>Name</th><th>Balance</th><th>Address</th><th>Status</th></tr>
   <tr>
    <td><asp:Label ID="lblCustid" runat="server" Text='<%# Eval("Custid") %>' /></td>
    <td><%# Eval("Name") %></td><td><%# Eval("Balance") %></td><td><%# Eval("Address") %></td>
    <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                  Checked='<%# Eval("Status") %>' /></td>
   </tr>
   <tr>
    <td colspan="5" align="center">
     <asp:Button ID="btnNew" runat="server" Text="New" CommandName="New" />
     <asp:Button ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
     <asp:Button ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
                  OnClientClick="return confirm('Are you sure of deleting the selected record?')"  />
    </td>
   </tr>
  </table>
 </ItemTemplate>
 <InsertItemTemplate>
  <table border="1">
   <tr><th>Name</th><th>Balance</th><th>Address</th><th>Status</th></tr>
   <tr>
    <td><asp:TextBox ID="txtName" runat="server" /></td>
    <td><asp:TextBox ID="txtBalance" runat="server" /></td>
    <td><asp:TextBox ID="txtAddress" runat="server" /></td>
    <td align="center"><asp:CheckBox ID="cbStatus" runat="server" /></td>
   </tr>
   <tr>
    <td colspan="4" align="center">
     <asp:Button ID="btnInsert" runat="server" Text="Insert" CommandName="Insert" />
     <asp:Button ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
    </td>
   </tr>
  </table>
 </InsertItemTemplate>
 <EditItemTemplate>
  <table border="1">
   <tr><th>Custid</th><th>Name</th><th>Balance</th><th>Address</th><th>Status</th></tr>
   <tr><td><%# Eval("Custid") %></td>
    <td><asp:TextBox ID="txtName" runat="server" Text='<%# Eval("Name") %>' /></td>
    <td><asp:TextBox ID="txtBalance" runat="server" Text='<%# Eval("Balance") %>' /></td>
    <td><asp:TextBox ID="txtAddress" runat="server" Text='<%# Eval("Address") %>' /></td>
    <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                    Checked='<%# Eval("Status") %>' /></td>
   </tr>
   <tr>
    <td colspan="5" align="center">
     <asp:Button ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
     <asp:Button ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
    </td>
   </tr>
  </table>
 </EditItemTemplate>
 <FooterTemplate>Click to navigate:</FooterTemplate>
</asp:FormView>
```

Now go to design view of the WebForm and double click on PageIndexChanging, ModeChanging, ItemInserting, ItemUpdating and ItemDeleting events of FormView control to generate methods for implementing the logic and write the following code under "aspx.cs" file:

Declarations: Customer obj;

Under Page_Load: obj = new Customer(); if (!IsPostBack) { LoadData(); }

```
private void LoadData() {
 FormView1.DataSource = obj.Customer_Select(null, true); FormView1.DataBind();
}
```

Under FormView PageIndexChanging:

```
FormView1.PageIndex = e.NewPageIndex; LoadData();
```

Under FormView ModeChanging:

```
if (e.NewMode == FormViewMode.Insert) { FormView1.ChangeMode(FormViewMode.Insert); }
else if(e.NewMode == FormViewMode.Edit) { FormView1.ChangeMode(FormViewMode.Edit);LoadData(); }
else { FormView1.ChangeMode(FormViewMode.ReadOnly);LoadData(); }
```

Under FormView ItemInserting:

```
string Name = ((TextBox)FormView1.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)FormView1.FindControl("txtBalance")).Text);
string Address = ((TextBox)FormView1.FindControl("txtAddress")).Text;
bool Status = ((CheckBox)FormView1.FindControl("cbStatus")).Checked;
int? Custid = null;
if (obj.Customer_Insert(Name, Balance, Address, Status, ref Custid) > 0) {
 FormView1.ChangeMode(FormViewMode.ReadOnly); LoadData();
}
else { Response.Write("<script>alert('Record insertion failed.')</script>"); }
```

Under FormView ItemUpdating:

```
int Custid = (int)e.Keys["Custid"];
string Name = ((TextBox)FormView1.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)FormView1.FindControl("txtBalance")).Text);
string Address = ((TextBox)FormView1.FindControl("txtAddress")).Text;

if (obj.Customer_Update(Custid, Name, Balance, Address) > 0) {
  FormView1.ChangeMode(FormViewMode.ReadOnly); LoadData();
}
else { Response.Write("<script>alert('Record updation failed.')</script>"); }
```

Under FormView ItemDeleting:

```
int Custid = int.Parse(((Label)FormView1.FindControl("lblCustid")).Text);
if (obj.Customer_Delete(Custid) > 0) {
  Response.Write("<script>alert('Record deleted from the table.')</script>");
}
else { Response.Write("<script>alert('Record deletion failed.')</script>"); }
LoadData();
```

ListView Control:

The ASP.NET ListView control enables you to bind data items that are returned from a data source and display them either in the form of pages, individual records, or we can even group them.The ListView control displays data in a format that you define by using templates and styles. It is useful for data in any repeating structure, similar to the DataList and Repeater controls. However, unlike those controls, with the ListView control we can enable users to edit, insert, and delete data, and to sort and page data. To work with ListView Control add a new WebForm naming it as ASPDB_Student_ListViewDemo1.aspx and write the following code under <div> tag:

```
<asp:ListView ID="ListView1" runat="server" >
<ItemTemplate>
<table border="1">
<tr><th>Sid</th><th>Name</th><th>Class</th><th>Fees</th><th>Photo</th><th>Status</th></tr>
<tr>
<td><%# Eval("Sid") %></td><td><%# Eval("Name") %></td>
<td><%# Eval("Class") %></td><td><%# Eval("Fees") %></td>
<td><asp:Image id="imgPhoto" runat="server" Width="200" Height="200"
                    ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' /></td>
<td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                    Checked='<%# Eval("Status") %>' /></td>
</tr>
</table>
</ItemTemplate>
</asp:ListView>
```

Now write the following code under “aspx.cs” file:

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

Under Page_Load:

```
if (!IsPostBack) { LoadData(); }
```

```
private void LoadData() {
  SqlDataAdapter da = new SqlDataAdapter(
            "Select Sid, Name, Class, Fees, PhotoName, Status From Student Order By Sid", ReadCS.ASPDB);
  DataSet ds = new DataSet(); da.Fill(ds, "Student"); ListView1.DataSource = ds; ListView1.DataBind();
}
```

When we run the above WebForm it displays each record in a tabular format i.e. every record will be one table whereas if we want to display the data record by record just like DetailsView or FormView controls we need to enable paging and to do that we need to use the DataPager control. To test it go to source view of the WebForm and write the following code under </asp:ListView> tag:

```
<asp:DataPager ID="DataPager1" runat="server" PagedControlID="ListView1" PageSize="1">
  <Fields><asp:NumericPagerField /></Fields>
</asp:DataPager>
```

- PagedControlId property is to specify to which control the paging has to be enabled.
- PageSize property is to specify the no. of records that has to be displayed at a time.
- Fields attribute is to specify the type of paging we want to enable which can either be NumericPagerField or NextPreviousPagerField.

Now when we run the WebForm it displays only a single record with paging option below but to navigate to next records we need to implement logic under ListView controls PagePropertiesChanging event as following:

DataPager1.SetPageProperties(e.StartRowIndex, e.MaximumRows, false); LoadData();

In above example we have displayed data row by row in the form of DetailsView and FormView controls where as if we want to display data as a single table just like a GridView control we need to take the help of a LayoutTemplate with a PlaceHolder control so that ItemTemplate comes and sits under that PlaceHolder as a row.

Layout Template (Table)

Sid	Name	Class	Fees	Photo	Status
<asp:PlaceHolder ID="ItemPlaceHolder" runat="server" />					
End of Table Data					

In the above case the Layout contains a table with 2 rows and between the 2 rows we have a PlaceHolder control and in that PlaceHolder, ItemTemplate will come and sit as a row for each record fetched from the table. To test these add a new WebForm naming it as “ASPDB_Student_ListViewDemo2.aspx” and write following code under <div> tag:

```
<asp:ListView ID="ListView1" runat="server">
  <LayoutTemplate>
    <table border="1">
      <tr><th>Sid</th><th>Name</th><th>Class</th><th>Fees</th><th>Photo</th><th>Status</th></tr>
      <asp:PlaceHolder ID="ItemPlaceHolder" runat="server" />
      <tr><td colspan="6" align="Center"> End of Student Data </td></tr>
    </table>
  </LayoutTemplate>
  <ItemTemplate>
    <tr>
      <td><%# Eval("Sid") %></td><td><%# Eval("Name") %></td>
      <td><%# Eval("Class") %></td><td><%# Eval("Fees") %></td>
      <td><asp:Image id="imgPhoto" runat="server" Width="200" Height="200"
                        ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' /></td>
    <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                        Checked='<%# Eval("Status") %>' /></td>
  </tr>
 </ItemTemplate>
</asp:ListView>
```

Now write the following code under "aspx.cs" file:

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

Under Page_Load:

```
SqlDataAdapter da = new SqlDataAdapter(
                        "Select Sid, Name, Class, Fees, PhotoName, Status FromStudent ", ReadCS.ASPDB);
DataSet ds = new DataSet(); da.Fill(ds, "Student"); ListView1.DataSource = ds; ListView1.DataBind();
```

Performing Grouping operations with ListView Control:

We use the GroupTemplate property to create a tiled layout in the ListView control. In a tiled table layout, the items are repeated horizontally in a row. The numbers of times that an item is repeated is specified by the GroupItemCount property. In our previous example ItemTemplate is included into LayoutTemplate with the help of a place holder control whereas in case of Grouping under LayoutTemplate, GroupTemplate is included and under GroupTemplate, ItemTemplate is included.

Layout Template (Table)

<asp:PlaceHolder ID="GroupPlaceHolder" runat="server" />

Group Template (TableRow)

<asp:PlaceHolder ID="ItemPlaceHolder" runat="server" />

In the above case Layout Template is a table and under it GroupTemplate is a row and under it ItemTemplate comes and sits as a column(s), where the no. of columns is based on the GroupItemCount property of ListView control. To test these add a new WebForm naming it as "ASPDB_Student_ListViewDemo3.aspx" and write the following code under <div> tag:

```
<asp:ListView ID="ListView1" runat="server"GroupItemCount="2" >
 <LayoutTemplate>
  <table border="1" width="100%">
   <asp:PlaceHolder ID="GroupPlaceHolder" runat="server" />
  </table>
 </LayoutTemplate>
 <GroupTemplate>
  <tr><asp:PlaceHolder ID="ItemPlaceHolder" runat="server" /></tr>
 </GroupTemplate>
 <ItemTemplate>
  <td>
   <table width="100%" border="1">
    <tr>
     <td rowspan ="5">
      <asp:Image id="imgPhoto" runat="server" Width="200" Height="200"
                 ImageUrl='<%# "~/Images/" + Eval("PhotoName") %>' />
     </td>
```

```
      <td><b>Sno: </b><%# Eval("Sid") %></td>
    </tr>
    <tr><td><b>Sname: </b><%# Eval("Name") %></td></tr>
    <tr><td><b>Class: </b><%# Eval("Class") %></td></tr>
    <tr><td><b>Fees: </b><%# Eval("Fees") %></td></tr>
    <tr>
      <td><b>Status: </b><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                      Checked='<%# Eval("Status") %>' /></td>
    </tr>
   </table>
  </td>
 </ItemTemplate>
</asp:ListView>
```

Now write the following code under "aspx.cs" file:

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

<u>Under Page_Load:</u>

```
SqlDataAdapter da = new SqlDataAdapter(
                              "Select Sid, Name, Class, Fees, PhotoName, Status From Student", ReadCS.ASPDB);
DataSet ds = new DataSet(); da.Fill(ds, "Student");
ListView1.DataSource = ds;
ListView1.DataBind();
```

<u>Peforming editing operations with ListView control:</u>

Add a new WebForm under the project naming it as "ASPDB_Customer_ListView_Editing.aspx" and write the following code under <div> tag:

```
<asp:ListView ID="ListView1" runat="server" DataKeyNames="Custid"  InsertItemPosition="LastItem" >
 <LayoutTemplate>
  <table border="1">
   <tr>
    <th><asp:LinkButton ID="btnCustid" runat="server" Text="Custid" CommandName="Sort"
        CommandArgument="Custid" /></th>
    <th><asp:LinkButton ID="btnName" runat="server" Text="Name" CommandName="Sort"
        CommandArgument="Name" /></th>
    <th><asp:LinkButton ID="btnBalance" runat="server" Text="Balance" CommandName="Sort"
        CommandArgument="Balance" /></th>
    <th><asp:LinkButton ID="btnAddress" runat="server" Text="City" CommandName="Sort"
        CommandArgument="Address" /></th>
    <th><asp:LinkButton ID="btnStatus" runat="server" Text="Is-Active" CommandName="Sort"
        CommandArgument="Status" /></th>
    <th>Actions</th>
   </tr>
   <asp:PlaceHolder id="ItemPlaceHolder" runat="server" />
  </table>
 </LayoutTemplate>
```

```
  <ItemTemplate>
    <tr>
      <td><%# Eval("Custid") %></td><td><%# Eval("Name") %></td>
      <td><%# Eval("Balance") %></td><td><%# Eval("Address") %></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
            Checked='<%# Eval("Status") %>' /></td>
      <td>
        <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
        <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
            OnClientClick="return confirm('Are you sure of deleting the record?')" />
      </td>
    </tr>
  </ItemTemplate>
  <InsertItemTemplate>
    <tr>
      <td></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtAddress" runat="server" Width="150" /></td>
      <td align="Center"><asp:CheckBox ID="cbStatus" runat="server" /></td>
      <td><asp:LinkButton ID="btnInsert" runat="server" Text="Insert" CommandName="Insert" /></td>
    </tr>
  </InsertItemTemplate>
  <EditItemTemplate>
    <tr>
      <td><%# Eval("Custid")%></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" Text='<%# Eval("Name")%>' /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" Text='<%# Eval("Balance")%>' /></td>
      <td><asp:TextBox ID="txtAddress" runat="server" Width="150" Text='<%# Eval("Address")%>' /></td>
      <td align="Center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                  Checked='<%# Eval("Status")%>' /></td>
      <td>
        <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
        <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
      </td>
    </tr>
  </EditItemTemplate>
</asp:ListView>
```

Now go to desing view of the WebForm, double click on ItemEditing, ItemCanceling, ItemInserting, ItemUpdating, ItemDeleting and Sorting events of ListView control and write following code under “aspx.cs” file:

```
using System.Data;
```

Declarations: *Customer obj;*

Under Page_Load:

```
obj = new Customer(); if (!IsPostBack) { LoadData(); ViewState["SortAction"] = "Custid Asc"; }
```

```
private void LoadData() {
  ListView1.DataSource = obj.Customer_Select(null, true);
  ListView1.DataBind();
}
```

Under ListView ItemEditing:

```
ListView1.EditIndex = e.NewEditIndex; LoadData();
```

Under ListView ItemCanceling:

```
ListView1.EditIndex = -1; LoadData();
```

Under ListView ItemInserting:

```
string Name = ((TextBox)e.Item.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)e.Item.FindControl("txtBalance")).Text);
string  Address = ((TextBox)e.Item.FindControl("txtAddress")).Text;
bool Status = ((CheckBox)e.Item.FindControl("cbStatus")).Checked;

int? Custid = null;
if (obj.Customer_Insert(Name, Balance, Address, Status, ref Custid) > 0) { LoadData(); }
else { Response.Write("<script>alert('Record insertion failed.')</script>"); }
```

Under ListView ItemUpdating:

```
int Custid = (int)e.Keys["Custid"];
string Name = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtBalance")).Text);
string Address = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtAddress")).Text;

if (obj.Customer_Update(Custid, Name, Balance, Address) > 0) {
  ListView1.EditIndex = -1;
  LoadData();
}
else { Response.Write("<script>alert('Record updation failed.')</script>"); }
```

Under ListView ItemDeleting:

```
int Custid = (int)e.Keys["Custid"];

if(obj.Customer_Delete(Custid) > 0) { LoadData(); }
else { Response.Write("<script>alert('Record deletion failed.')</script>"); }
```

Under ListView Sorting:

```
string[] SortAction = ViewState["SortAction"].ToString().Trim().Split(' ');
if (e.SortExpression == SortAction[0]) {
  if (SortAction[1] == "Asc") { ViewState["SortAction"] = e.SortExpression + " Desc"; }
  else { ViewState["SortAction"] = e.SortExpression + " Asc"; }
}
else { ViewState["SortAction"] = e.SortExpression + " Asc"; }

DataView dv = obj.Customer_Select(null, true).Tables[0].DefaultView;
dv.Sort = (string)ViewState["SortAction"];
ListView1.DataSource = dv;
ListView1.DataBind();
```

SqlDataSource Control:

The SqlDataSource control uses ADO.NET classes to interact with any database supported by ADO.NET. This includes Microsoft SQL Server (using the System.Data.SqlClient provider), System.Data.OleDb, System.Data.Odbc, and Oracle (using the System.Data.OracleClient provider). Using a SqlDataSource control allows you to access and manipulate data in an ASP.NET page without using ADO.NET classes directly. You provide a connection string to connect to your database and define the SQL statements or stored procedures that work with your data. At run time, the SqlDataSource control automatically opens the database connection, executes the SQL statement or stored procedure, returns the selected data (if any), and then closes the connection. It also supports performing insert, update and delete operations also.

To work with SqlDataSource control add a new WebForm naming it as "ASPDB_Customer_DataSource_Demo.aspx" and place a SqlDataSource control on it, now we need to configure the control with required data source and to do that click on the Configure Button present at the top right corner and select the option Configure Data Source, which opens a window and click New Connection button on it, which opens a new window choose the Data Source as Microsoft Sql Server and click ok on it, which opens a new window to enter connection details, enter the connection details and choose the Database as ASPDB and click ok, now click on the next button which asks for saving the connection string into config file, select yes and click next, which takes to another window asking for writing a custom Sql Statement or choosing a Stored Procedure or to specify columns from any particular table or view => choose this option and below that we find all the tables in our database, choose "Customer" and select the required columns and beside that we find buttons to add where clause, order by clause and a button "Advanced", when we click on this button it will ask for generating Insert, Update and Delete statements, select the CheckBox and click ok, now click on the Next button where we can test our Query and click finish to complete the configuration.

Now on the WebForm place a GridView conrol and click on the Configure button present at top right corner which opens a window, in that under the Choose Data Source: select => SqlDataSource1 we configured right now and in the below it will ask for "Enable Paging", "Enable Sorting", "Enable Editing" and "Enable Deleting", choose the required options by selecting the CheckBox's and run the WebForm which display the data by providing all options we have choosen. Now if we go to the Source View of the WebForm we will find all the code that is generated by the SqlDataSource control.

Chart Control:

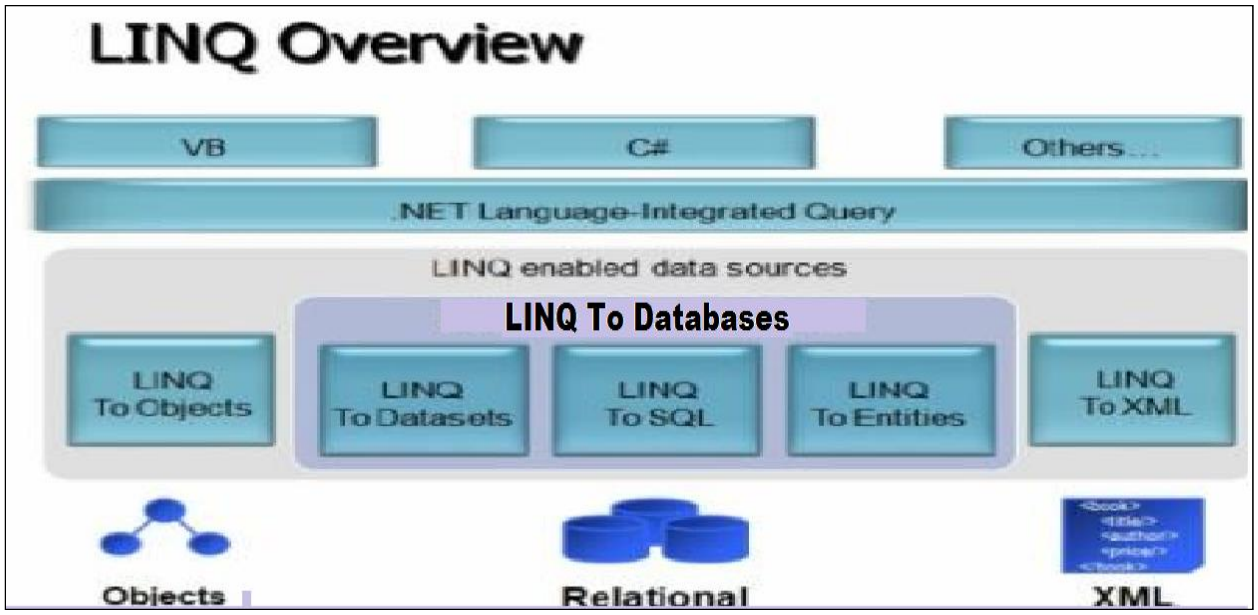
The Chart controls enable you to create ASP.NET pages or Windows Forms applications with simple, intuitive, and visually compelling charts for complex statistical or financial analysis.

To work with Chart control add a new WebForm naming it as "ASPDB_Employee_Chart_Demo.aspx", place a Chart Control on the WebForm and click on the button present at top right corner which opens a window, in that under Choose Data Source: select "New Data Source" Option which opens a window and in that select "Sql – Database" option and in the below specify an Id for the data source as "EmployeeDS" and click ok which opens a window asking for choosing Data Source, below that we find a ComboBox showing the list of databases configured under Server Explorer and if our database is in that list select it or click on the "New Connection" button to configure with a new data source and click Next button, and in that window choose the option "Specify a custom SQL Statement or Stored Procedure radio button and click next, in that under the Sql Statements option write the following query => "Select Did, TotalSalary = Sum(Salary) From Employee Group By Did Order By Did" and click next and click finish, so that chart control uses "EmployeeDS" we have created now and below that choose the chart type we want like column, pie, line etc, and below that under "X Value Member" choose "Did" column and under "Y Value Member" choose "TotalSalary" and run the WebForm to watch the Chart.

LINQ (Language Integrated Query)

In C# 3.0 .Net has introduced a new language known as "LINQ" much like SQL (which we use universally with relational databases to perform queries). LINQ allows you to write query expressions (similar to SQL queries) that can retrieve information from a wide variety of data sources like Objects, Databases and XML.

LINQ stands for Language Integrated Query. LINQ is a data querying methodology which provides querying capabilities to .NET languages with syntax similar to a SQL Query. LINQ has a great power of querying on any source of data, where the data source could be collections of objects, database or XML files.



LINQ to Objects: used to perform queries against the in-memory data like an array or collection.

LINQ to Databases:

- LINQ to DataSet is used to perform queries against ADO.NET Data Table's.
- LINQ to SQL is used to perform queries against the relation database, but only Microsoft SQL Server.
- LINQ to Entities is used to perform queries against any relation database like SQL Server, Oracle, etc.

LINQ to XML (XLinq): used to perform queries against the XML source.

Advantages of LINQ:

i. LINQ offers an object-based, language-integrated way to query over data, no matter where that data came from. So through LINQ we can query database, XML as well as collections.

ii. Compile time syntax checking.

iii. It allows you to query collections, arrays, and classes etc. in the native language of your application like VB or CSharp.

LINQ to Objects

Using this we can perform queries against the in-memory data like an array or collection and filter or sort the information under them. Syntax of the query we want to use on objects will be as following:

from <alias> in <array_name | collection_name> [<clauses>] select <alias> | new { <list of columns> }

- Linq queries start with from keyword and ends with select keyword.
- While writing conditions we need to use the alias name we have specified if column names are not present, just like we use column names in case of SQL.
- Clauses can be like where, groupby and orderby.
- To use LINQ in your application first we need to import System.Linq namespace.

To work with LINQ, open a new ASP.Net Web Application Project naming it as “LinqExamples”, add a new WebForm in it naming it as “LinqWithArray.aspx” and write the following code under <div> tag:

```
<asp:Button ID="Button1" runat="server" Text="Get values > 40 using Imperative Code" /><br />
<asp:Label ID="Label1" runat="server" /><br />
<asp:Button ID="Button2" runat="server" Text="Get values > 40 using Declarative Code" /><br />
<asp:Label ID="Label2" runat="server" />
```

Now write the following code “aspx.cs” file:

Declarations:*int[] arr = { 13, 56, 29, 78, 31, 45, 91, 7, 34, 86, 24, 62, 19, 3, 98, 16, 36, 41, 52, 83, 38, 79, 67, 27, 5 };*

Under Button1 Click:

```
int Count = 0, Index = 0;
foreach(int i in arr) {
  if (i > 40) { Count += 1; }
}
int[] brr = new int[Count];
foreach (int i in arr) {
  if (i > 40) {
    brr[Index] = i; Index += 1;
  }
}
Array.Sort(brr); Array.Reverse(brr); Label1.Text = String.Join(" ", brr);
```

Under Button2 Click:

```
var brr = from i in arr where i > 40 orderby i descending select i; Label2.Text = String.Join(" ", brr);
```

Now to test Linq with Collections add a new WebForm naming it as “LinqWithCollections.aspx” and write the following code under “aspx.cs” file:

Under Page_Load:

```
List<string> Cities = new List<string>() { "Amaravati", "Itanagar", "Dispur", "Patna", "Panaji", "Gandhinagar",
        "Shimla", "Srinagar", "Bangalooru", "Thiruvananthapuram", "Bhopal", "Mumbai", "Imphal","Shillong",
        "Aizawl", "Kohima", "Bhubaneswar", "Chandigarh", "Jaipur", "Gangtok", "Chennai", "Agartala","Lucknow",
        "Kolkata", "Raipur", "Dehradun", "Ranchi", "Hyderabad", "Delhi" };

//Query to fetch all the Cities as it
var Coll1 = from s in Cities select s;                 Response.Write(String.Join(" ", Coll1) + "<hr />");

//Query to fetch all the Cities in Ascending order
var Coll2 = from s in Cities orderby s select s;       Response.Write(String.Join(" ", Coll2) + "<hr />");

//Query to fetch all the Cities in Descending order
var Coll3 = from s in Cities orderby s descending select s;
Response.Write(String.Join(" ", Coll3) + "<hr />");

//Query to fetch all the Cities with a length of 6 characters
var Coll4 = from s in Cities where s.Length == 6 select s;
Response.Write(String.Join(" ", Coll4) + "<hr />");
```

```
//Query to fetch cities starting with a particular character
var Coll5 = from s in Cities where s.IndexOf('A') == 0 select s; Response.Write(String.Join(" ", Coll5) + "<hr />");
var Coll6 = from s in Cities where s.Substring(0, 1) == "B" select s; Response.Write(String.Join(" ", Coll6) + "<hr />");
var Coll7 = from s in Cities where s.StartsWith("C") select s; Response.Write(String.Join(" ", Coll7) + "<hr />");
var Coll8 = from s in Cities where s[0] == 'D' select s; Response.Write(String.Join(" ", Coll8) + "<hr />");

//Query to fetch cities ending with a particular character
var Coll9 = from s in Cities where s.IndexOf('i') == s.Length - 1 select s;
Response.Write(String.Join(" ", Coll9) + "<hr />");
var Coll10 = from s in Cities where s.Substring(s.Length - 1) == "r" select s;
Response.Write(String.Join(" ", Coll10) + "<hr />");
var Coll11 = from s in Cities where s.EndsWith("a") select s; Response.Write(String.Join(" ", Coll11) + "<hr />");
var Coll12 = from s in Cities where s[s.Length - 1] == 'h' select s; Response.Write(String.Join(" ", Coll12) + "<hr />");

//Query to fetch cities containing "a" in third place
var Coll13 = from s in Cities where s.IndexOf('a') == 2 select s; Response.Write(String.Join(" ", Coll13) + "<hr />");
var Coll14 = from s in Cities where s.Substring(2, 1) == "a" select s;
Response.Write(String.Join(" ", Coll14) + "<hr />");
var Coll15 = from s in Cities where s[2] == 'a' select s; Response.Write(String.Join(" ", Coll15) + "<hr />");

//Query to fetch cities containing character "h" or "H" under them
var Coll16 = from s in Cities where s.IndexOf('h') >= 0 || s.IndexOf('H') >= 0 select s;
Response.Write(String.Join(" ", Coll16) + "<hr />");
var Coll17 = from s in Cities where s.ToLower().IndexOf('h') >= 0 select s;
Response.Write(String.Join(" ", Coll17) + "<hr />");
var Coll18 = from s in Cities where s.Contains('h') || s.Contains('H') select s;
Response.Write(String.Join(" ", Coll18) + "<hr />");
var Coll19 = from s in Cities where s.ToUpper().Contains('H') select s;
Response.Write(String.Join(" ", Coll19) + "<hr />");

//Query to fetch cities not containing character "h" or "H" under them
var Coll20 = from s in Cities where s.IndexOf('h') == -1 && s.IndexOf('H') == -1 select s;
Response.Write(String.Join(" ", Coll20) + "<hr />");
var Coll21 = from s in Cities where s.ToLower().IndexOf('h') == -1 select s;
Response.Write(String.Join(" ", Coll21) + "<hr />");
var Coll22 = from s in Cities where s.Contains('h') == false && s.Contains('H') == false select s;
Response.Write(String.Join(" ", Coll22) + "<hr />");
var Coll23 = from s in Cities where s.ToUpper().Contains('H') == false select s;
Response.Write(String.Join(" ", Coll23) + "<hr />");
```

Note: the values that are returned by LINQ queries can be captured by using implicitly typed local variables, so in the above programs'brr' & 'coll'are implicitly declared array/collection which stores values retrieved by the query.

In traditional process of filtering data of an array or collection we have repetition statements that filter arrays focusing on the process of getting the results i.e. iterating through the elements and checking whether they satisfy the desired criteria, whereas LINQ specifies, not the steps necessary to get the results, but rather the conditions that selected elements must satisfy and this is known as ***declarative programming***- as opposed to ***imperative***

programming (which we've been using so far) in which we specify the actual steps to perform a task. Procedural & Object Oriented Languages are a subset of imperative. The queries we have used above specifies that the result should consist of all the integers in the List that are greater than 40, but it does not specify how to obtain the result, the C# compiler generates all the necessary code automatically, which is one of the great strengths of LINQ.

LINQ Providers: The syntax of LINQ is built into the language, but LINQ queries may be used in many different contexts because of libraries known as providers. A LINQ provider is a set of classes that implement LINQ operations and enable programs to interact with data sources to perform tasks such as sorting, grouping and filtering elements. When we import the "System.Linq" namespace it contains the LINQ to Objects provider, without importing it the compiler cannot locate a provider for the LINQ queries and issues errors on LINQ queries.

Storing user-defined type values in Collection:

The type of values being stored in a Generic Collection can be of user-defined types also like a class type or structure type that is defined to represent any entity. To test this, add a new class under the project naming it as "Employee.cs" and write the following code in it:

```
public class Employee {
  public int Id { get; set; }
  public string Name { get; set; }
  public string Job { get; set; }
  public double Salary { get; set; }
  public bool Status { get; set; }
}
```

Now we can store instances of Employee type under the List just like we stored String type instances in our previous example but the difference is string is a Simple type and it's instance represents a single value whereas Employee is a complex type and it's instance is internally a collection of different attributes like Id, Name, Job, Salary and Status, and we can also write queries on this list and access data based on the Employee attributes. To test this process goto "Global.asax" file and write the following code:

Under "Application_Start" method:

```
Employee e1 = new Employee { Id = 1001, Name = "Naresh", Job = "President", Salary = 12000.00, Status = true };
Employee e2 = new Employee { Id = 1002, Name = "Raju", Job = "Manager", Salary = 10000.00, Status = true };
Employee e3 = new Employee { Id = 1003, Name = "Ajay", Job = "Salesman", Salary = 6000.00, Status = true };
Employee e4 = new Employee { Id = 1004, Name = "James", Job = "Salesman", Salary = 6000.00, Status = true };
Employee e5 = new Employee { Id = 1005, Name = "Jones", Job = "Clerk", Salary = 3000.00, Status = true };
Employee e6 = new Employee { Id = 1006, Name = "Scott", Job = "Manager", Salary = 10000.00, Status = true };
Employee e7 = new Employee { Id = 1007, Name = "Smith", Job = "Analyst", Salary = 8000.00, Status = true };
Employee e8 = new Employee { Id = 1008, Name = "Vijay", Job = "Analyst", Salary = 8000.00, Status = true };
Employee e9 = new Employee { Id = 1009, Name = "Venkat", Job = "Clerk", Salary = 3000.00, Status = true };
Employee e10 = new Employee { Id = 1010, Name = "Satish", Job = "Admin", Salary = 5000.00, Status = true };
List<Employee> Employees = new List<Employee>() { e1, e2, e3, e4, e5, e6, e7, e8, e9, e10 };
Application["Emps"] = Employees;
```

Now add a new WebForm under the project naming it as "LinqWithComplexTypes.aspx" and write the following code under <div> tag:

```
        <table align="center"><tr><td align="center">
        <asp:DropDownList ID="ddlJobs" runat="server" AutoPostBack="true" /></td></tr>
        <tr><td><asp:GridView ID="gvEmp" runat="server" /></td></tr></table>
```

Now write the following code under "aspx.cs" file:

Declarations: *List<Employee> Employees;*

Under Page_Load:

```
Employees = (List<Employee>)Application["Emps"];
if(!IsPostBack) {
  ddlJob.DataSource = (from E in Employees select new { E.Job }).Distinct();
  ddlJob.DataTextField = "Job"; ddlJob.DataValueField = "Job";
  ddlJob.DataBind(); ddlJob.Items.Insert(0, "-Select Job-");
  gvEmp.DataSource = from E in Employees select E; gvEmp.DataBind();
}
```

Under DropDownList SelectedIndexChanged:

```
if (ddlJob.SelectedIndex == 0) { gvEmp.DataSource = from E in Employees select E; }
else { gvEmp.DataSource = from E in Employees where E.Job == ddlJob.SelectedValue select E; }
gvEmp.DataBind();
```

LINQ to XML (XLINQ)

Using this we can write queries on XML Source or XML Data just like we have written queries on Arrays and Collections. To work with LINQ to XML first we need to import the namespace "System.Xml.Linq" which provides all the classes for writing queries on XML Source. In this namespace we are provided with a class known as "XElement" which is actually responsible for loading an XML Document into our application and then we need to write the query on loaded document. To test this first add a XML File under the project naming it as "Travel.xml" and write the following code in it:

```
<Cities>
  <City><Name>Bangkok</Name><Continent>Asia</Continent><Image>Bangkok.jpg</Image>
    <Video>Bangkok.mp4</Video><Price>25000</Price></City>
  <City><Name>Dubai</Name><Continent>Asia</Continent><Image>Dubai.jpg</Image>
    <Video>Dubai.mp4</Video><Price>22000</Price></City>
  <City><Name>Hong Kong</Name><Continent>Asia</Continent><Image>HongKong.jpg</Image>
    <Video>HongKong.mp4</Video><Price>35000</Price></City>
  <City><Name>London</Name><Continent>Europe</Continent><Image>London.jpg</Image>
    <Video>London.mp4</Video><Price>50000</Price></City>
  <City><Name>Mumbai</Name><Continent>Asia</Continent><Image>Mumbai.jpg</Image>
    <Video>Mumbai.mp4</Video><Price>15000</Price></City>
  <City><Name>New York</Name><Continent>North America</Continent><Image>NewYork.jpg</Image>
    <Video>NewYork.mp4</Video><Price>60000</Price></City>
  <City><Name>Paris</Name><Continent>Europe</Continent><Image>Paris.jpg</Image>
    <Video>Paris.mp4</Video><Price>45000</Price></City>
  <City><Name>Rio de Janeiro</Name><Continent>South America</Continent>
    <Image>RiodeJaneiro.jpg</Image><Video>RiodeJaneiro.mp4</Video><Price>55000</Price></City>
  <City><Name>Singapore</Name><Continent>Asia</Continent><Image>Singapore.jpg</Image>
    <Video>Singapore.mp4</Video><Price>30000</Price></City>
  <City><Name>Sydney</Name><Continent>Australia</Continent><Image>Sydney.jpg</Image>
    <Video>Sydney.mp4</Video><Price>75000</Price></City>
</Cities>
```

Now create 2 folders under the project naming them as Images and Videos, and copy an image of each city into Images folder and a video of each city into videos folder, then add a new WebForm under the project naming it as “Travel.aspx” and write the following code under <div> tag:

```
<h1 style="background-color:yellow;color:red;text-align:center">Travel Analog</h1>
<table align="center">
  <tr><td align="center"><asp:DropDownList ID="ddlContinents" runat="server" AutoPostBack="true" /></td></tr>
  <tr><td>
    <asp:Repeater ID="rptCities" runat="server">
      <HeaderTemplate><table border="1" align="center">
        <tr><th>City Name</th><th>Video</th><th>Image</th><th>Price</th></tr>
      </HeaderTemplate>
      <ItemTemplate>
        <tr>
          <td><%# Eval("City") %></td>
          <td><NIT:VideoPlayer ID="VideoPlayer1" runat="server" Width="200" Height="200" Controls="true"
                    Mp4Url='<%# "/Videos/" +  Eval("Video") %>' /></td>
          <td><asp:Image ID="Image1" runat="server" Width="150" Height="200"
                    ImageUrl='<%# "~/Images/" + Eval("Photo") %>' /></td>
          <td><%# Eval("Price") %></td>
        </tr>
      </ItemTemplate>
      <FooterTemplate></table></FooterTemplate>
    </asp:Repeater></td>
  </tr>
</table>
```

Now write the following code under “aspx.cs” file:

using System.Xml.Linq;

<u>Under Page_Load:</u>

```
if (!IsPostBack) {
  var doc = XElement.Load(Server.MapPath("Travel.xml")); var Cities = doc.Elements("City");
  ddlContinents.DataSource = (from C in Cities select new { Continent = C.Element("Continent").Value}).Distinct();
  ddlContinents.DataTextField = "Continent"; ddlContinents.DataValueField = "Continent";
  ddlContinents.DataBind(); ddlContinents.Items.Insert(0, "-Select Continent-");
  rptCities.DataSource = from C in Cities select new { City = C.Element("Name").Value,
        Video = C.Element("Video").Value, Photo = C.Element("Image").Value, Price = C.Element("Price").Value };
  rptCities.DataBind();
}
```

<u>Under DropDownList SelectedIndexChanged:</u>

```
var doc = XElement.Load(Server.MapPath("Travel.xml")); var table = doc.Elements("City");
var coll = from s in table where s.Element("Continent").Value == ddlContinents.SelectedItem.Text select new
        { Name = s.Element("Name").Value, Image = s.Element("Image").Value, Video = s.Element("Video").Value,
                Price = s.Element("Price").Value };
dlCities.DataSource = coll; dlCities.DataBind();
```

LINQ to Databases

This is designed for working with relational databases like Sql Server, Oracle, etc and this is the biggest and most exciting addition to the .Net Framework 3.5. Basically, what LINQ provides is a lightweight interface over programmatic data integration. This is such a big deal because **Data is King**. Pretty much every application deals with data in some manner, whether that data comes from memory, databases, XML files, text files, or something else. Many developers find it very difficult to move from the strongly typed object-oriented world of .Net languages to the data tier where objects are second-class citizens. The transition from the one world to the next was a kludge at best and was full of error-prone actions. In .Net, programming with objects means a wonderful strongly typed ability to work with code. You can navigate very easily through the namespaces; work with a debugger in the Visual Studio IDE, and more. However, when you have to access data, you will notice that things are dramatically different. You end up in a world that is not strongly typed, where debugging is a pain or even non-existent, and you end up spending most of the time sending strings to the database as commands. As a developer, you also have to be aware of the underlying data and how it is.

Advantages of LINQ:

- LINQ is a part of .Net Framework so we can use any .Net Language for writing LINQ statements, so that we can write all the queries in the syntax of the .Net language we are working with.
- It is type safe i.e. we get data values from database in the similar type of the columns "data type" whereas in ADO.Net we receive all data values as "object type" only.
- Full support of intellisense while writing the queries and also supports debugging.
- Compile-time syntax checking rather than runtime.
- It is a pure object oriented way of communication with data sources where as in ADO.Net we used SQL for communication with databases which is a blend of relational and object oriented code i.e. in LINQ, Tables (Entities) are Classes, Columns (Attributes) are Properties of Classes, Records or Rows are Instances of Classes and Stored Procedures are Methods.

LINQ to Databases is again divided into 2 parts:

1. **LINQ to SQL**
2. **LINQ to Entities or Entity Framework**

LINQ to SQL:

LINQ to SQL in particular is a means to have a strongly typed interface against a SQL Server Database. You will find the approach that LINQ to SQL provides is by far the easiest approach for querying SQL Server available at the moment. It is important to remember that LINQ to SQL is not only about querying data, but you can also perform Insert/Update/Delete operations that you need to perform which are known as CRUD operations (Create(Insert)/Read(Select)/Update/Delete).

To work with "Linq to Sql" first we need to convert relational objects in DB into object oriented types under the language and the process of this conversion is known as ORM (Object Relational Mapping) and to do this we are provided with 2 tools under Visual Studio which does an outstanding job of making it as easy as possible.

1. **OR (Object-Relational) Designer**
2. **EDM (Entity Data Model)**

Note: OR-Designer is used in LINQ to SQL and EDM is used in LINQ to Entities.

Introducing the O/R Designer:

To start working with "LINQ to SQL" open a new ASP.Net Web Application Project naming it as "LinqToSqlProject", then open the "Add New Item" Window for adding OR-Designer into the project which is found with the name "LINQ to SQL Classes" that adds a new item with the extension .dbml (Database Markup Language).

We can give any name to the .dbml item but it is always suggested to use our Database name as a name to this, so let us name this as ASPDB.dbml as our database name is ASPDB and click “Add” button which will do the following:

1. Adds a reference to System.Data.Linq assembly which is required to work with “LINQ to Sql”.
2. Under Solution Explorer we will find “ASPDB.dbml” and under it we will find 2 sub-items “ASPDB.dbml.layout” and “ASPDB.Designer.cs” and under this file only OR-Designer writes all the ORM code converting Relational Objects into Object Oriented Types.
3. The O/R Designer is added in the studio which will appear as a tab within the document window directly in the IDE and this is made up of two parts. The first part on the left is for Data Classes, which map to Tables, Views, etc, dragging such items on this surface will give us a visual representation of those objects that can be worked with. The second part on the right is for Methods, which map to the Stored Procedures within the DB.

If we look into the code of ASPDB.designer.cs file we will find a class ASPDBDataContent inheriting from DataContext class; we can view this class something that maps to a Connection class binding with the DB. This class works with the connection string and connects to the database for any required operations when we create instance of the class. This class also provides methods like CreateDatabase, DeleteDatabase, GetTable, ExecuteCommand, ExecuteQuery, SubmitChanges etc inherited from its parent, using which we can perform action directly on the DB. Currently in this class we find 4 parameterized constructors for creating instance of the class.

Creating the Customer Class to work with Customer Table:

For this example, let’s work with the Customer Table (Entity) from our ASPDB database, which means that we are going to create a Customer Class (Entity) that will create a map to Customer table. To accomplish this task simply open the “Server Explorer” within Visual Studio, configure our ASPDB Database under it, and drag and drop the Customer table onto the design surface of O/R Designer in LHS which will add a bunch of code in to ASPDB.designer.cs file on our behalf with a Customer Class in it, and this class will give us a strongly typed access to Customer Table (Entity). When we drag and drop a table on OR Designer the following actions gets performed internally:

1. Defines a class representing the table (Entity) we have dragged and dropped on the OR Designer where the name of the class will be same as the table name, as we dropped the Customer table on OR Designer Customer class gets defined.
2. Defines properties under the class defined representing the table (Entity), where each property represents each column of the table.
3. Defines a property under the ASPDBDataContext class for referring to the table we are working with and the type of the property will be Table<Entity>, for example because we are working with Customer Entity the property name will be Customers and the type of the property will be Table<Customer>.

Note: Table<Entity> is a generic class under System.Data.Linq namespace which contains a set of methods DeleteOnSubmit, InsertOnSubmit, SingleOrDefault etc. for performing the CRUD operations.

Apart from the above 3 when we place the first object on the OR Designer it will also perform these activities also:

1. Writes the Connection String in the Web.config file targeting to the database we are working with.
2. Defines a new parameter less constructor under the ASPDBDataContext class and we can use this for creating the instance for connecting to DB and it only will read the connection string from Web.config.

Now add a new WebForm in the project naming it as "DisplayData.aspx", place a GridView control on it and write the following code in "aspx.cs" file:

```
using System.Data.Linq;
```

Under Page_Load:

```
ASPDBDataContext dc = new ASPDBDataContext();
Table<Customer> tab = dc.Customers;  GridView1.DataSource = tab; GridView1.DataBind();
```

Note: ASPDBDataConext class instance when created will read the connection string from the Web.config file and connects to our ASPDB database and Customers is the property under this class for referring to the whole table whose type is Table<Customer> under System.Data.Linq namespace which we have bound to GridView Control. In the above code "*Table<Customer> tab = dc.Customers; GridView1.DataSource = tab;*" can be directly implemented as "*GridView1.DataSource = dc.Customers;*" without again capturing data in to "tab".

Performing Query Operations using LINQ:

We can perform query operations by using LINQ and fetch required data from the table just like we have implemented queries on Arrays, Collections and XML Source, but here the syntax of the query will be as following:

from <alias> in <table> [clauses] select <alias> | new { <list of columns> }

Note: "select <alias>" means "select *" i.e. all columns, whereas if we want selected columns then it should be "select new { <specify column names here prefixing with the alias name> }"

To test the above add a new WebForm in the project naming it as "LinqQuery.aspx" and write the following code under the <div> tag:

```
<table align="center">
<tr><td align="center"><asp:DropDownList ID="ddlCities" runat="server" AutoPostBack="true" /></td></tr>
<tr><td><asp:GridView ID="gvCustomers" runat="server" /></td></tr>
</table>
```

Now write the following code under "aspx.cs" file:

Declarations: *ASPDBDataContext dc;*

Under Page_Load:

```
dc = new ASPDBDataContext();
if (!IsPostBack) {
  ddlCities.DataSource = (from C in dc.Customers select new { C.Address }).Distinct();
  ddlCities.DataTextField = "Address"; ddlCities.DataValueField = "Address";
  ddlCities.DataBind(); ddlCities.Items.Insert(0, "-Select City-");
  gvCustomers.DataSource = dc.Customers; gvCustomers.DataBind();
}
```

Under DropDownList SelectedIndexChanged:

```
if (ddlCities.SelectedIndex > 0)
  gvCustomers.DataSource = from C in dc.Customers where C.Address == ddlCities.SelectedValue select C;
else
  gvCustomers.DataSource = from C in dc.Customers select C;
gvCustomers.DataBind();
```

Performing CRUD operations using LINQ:

To perform CRUD operations by using LINQ, we need to follow the below steps for each operation.

Steps for Inserting:

1. Create a instance of Customer class, which is defined representing the Customer Entity because each instance is a record and assign values to the properties of that instance, because those properties represents columns.
2. Call InsertOnSubmit method on the table i.e. Customers which adds the record into the table (local copy) in a pending state.
3. Call SubmitChanges method on DataContext object for saving the changes to DB Server.

Steps for Updating:

1. Create a reference of the Customer Entity (class) that has to be updated by calling Single or First or SingleOrDefault or FirstOrDefault methods on the table i.e. Customers.
2. Re-assign values to properties of the reference so that old values gets changed to new values.
3. Call SubmitChanges method on DataContext object for saving the changes to DB server.

Steps for Deleting:

1. Create a reference of the Customer Entity (class) that has to be deleted by calling Single or First or SingleOrDefault or FirstOrDefault method on the table i.e. Customers.
2. Call DeleteOnSubmit method on the table i.e. Customers that deletes the record from table (local copy) in a pending state.
3. Call SubmitChanges method on DataContext object for saving the changes to DB server.

To perform CRUD operations, add a new WebForm in the project naming it as “CRUDWithLinq.aspx” and write the following code under <div> tag:

```
<asp:ListView ID="ListView1" runat="server" DataKeyNames="Custid"  InsertItemPosition="LastItem" >
 <LayoutTemplate>
  <table border="1">
   <tr>
    <th>Custid</th><th>Name</th><th>Balance</th><th>Address</th><th>Status</th><th>Action</th>
   </tr>
   <asp:PlaceHolder id="ItemPlaceHolder" runat="server" />
  </table>
 </LayoutTemplate>
 <ItemTemplate>
  <tr>
   <td><%# Eval("Custid") %></td><td><%# Eval("Name") %></td>
   <td><%# Eval("Balance") %></td><td><%# Eval("Address") %></td>
   <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
        Checked='<%# Eval("Status") %>' /></td>
   <td align="center">
    <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
    <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
        OnClientClick="return confirm('Are you sure of deleting the record?')" />
   </td>
  </tr>
 </ItemTemplate>
```

```
  <InsertItemTemplate>
    <tr>
      <td><asp:TextBox ID="txtCustid" runat="server" Width="50" /></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtAddress" runat="server" Width="150" /></td>
      <td align="Center"><asp:CheckBox ID="cbStatus" runat="server" /></td>
      <td align="Center">
        <asp:LinkButton ID="btnInsert" runat="server" Text="Insert" CommandName="Insert" /></td>
    </tr>
  </InsertItemTemplate>
  <EditItemTemplate>
    <tr>
      <td><%# Eval("Custid")%></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" Text='<%# Eval("Name")%>' /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" Text='<%# Eval("Balance")%>' /></td>
      <td><asp:TextBox ID="txtAddress" runat="server" Width="150" Text='<%# Eval("Address")%>' /></td>
      <td align="Center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                          Checked='<%# Eval("Status")%>' /></td>
      <td align="center">
        <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
        <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
      </td>
    </tr>
  </EditItemTemplate>
</asp:ListView>
```

Now go to design view of the WebForm, double click on ItemEditing, ItemCanceling, ItemInserting, ItemUpdating, and ItemDeleting events of ListView control and write following code under "aspx.cs" file:

<u>**Declarations:**</u> ASPDBDataContext dc;

<u>**Under Page_Load:**</u>
dc = new ASPDBDataContext(); if (!IsPostBack) { LoadData(); }

private void LoadData() {
  ListView1.DataSource = from C in dc.Customers where C.Status == true select C; ListView1.DataBind();
}

<u>**Under ListView ItemInserting:**</u>

```
int Id = int.Parse(((TextBox)e.Item.FindControl("txtCustid")).Text);
string Name = ((TextBox)e.Item.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)e.Item.FindControl("txtBalance")).Text);
string Address = ((TextBox)e.Item.FindControl("txtAddress")).Text;
bool Status = ((CheckBox)e.Item.FindControl("cbStatus")).Checked;

Customer obj = new Customer {Custid = Id, Name = Name, Balance = Balance, Address = Address, Status = Status};
dc.Customers.InsertOnSubmit(obj); dc.SubmitChanges(); LoadData();
```

<u>**Under ListView ItemEditing:**</u> ListView1.EditIndex = e.NewEditIndex; LoadData();

<u>Under ListView ItemCanceling:</u> ListView1.EditIndex = -1; LoadData();

<u>Under ListView ItemUpdating:</u>

```
int Id = (int)e.Keys["Custid"];
string Name = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtBalance")).Text);
string Address = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtAddress")).Text;

Customer obj = dc.Customers.SingleOrDefault(C => C.Custid == Id);
obj.Name = Name; obj.Balance = Balance; obj.Address = Address;
dc.SubmitChanges(); ListView1.EditIndex = -1; LoadData();
```

<u>Under ListView ItemDeleting:</u>

```
int Id = (int)e.Keys["Custid"];

Customer obj = dc.Customers.SingleOrDefault(C => C.Custid == Id);
(dc.Customers.DeleteOnSubmit(obj); or obj.Status = false;) dc.SubmitChanges(); LoadData();
```

<u>Calling Stored Procedures using LINQ</u>:

If we want to call any SP of Sql Server DB using LINQ we need to first drag and drop the SP on RHS panel of OR-designer, so that it gets converted into a method under ASPDBDataContext class with same name of the SP. If the SP has any parameters those parameters will be defined for the method also, where input parameters of procedure becomes input parameters and output parameters of procedure becomes ref parameters of the method. For example if the below SP was dropped on RHS panel of OR-designer:

Create Procedure Add(@x int, @y int, @z int out)

The method gets created as following:

public int Add(int? x, int? y, ref int? z)

If the SP contains any non-query operations in it, in such cases the return type of method will be int, where as if the SP has any select statements in it that returns table results then the return type of the method will be ISingleResult<T>, where T represents a class that is newly defined when we drag and drop the select SP whose name will be SP Name suffixed with "Result" i.e. for example if the procedure name is Customer_Select then the class name will be Customer_SelectResult.

To call Stored Procedures using LINQ first drag and drop all our 4 Stored Procedures we defined earlier Customer_Select, Customer_Insert, Customer_Update and Customer_Delete on the RHS panel of the OR-Designer from Server Explorer so that all the methods gets generated under ASPDBDataContext class.

Now add a new WebForm in the project naming it as "CRUDWithSP.aspx" and write the following code under <div> tag:

```
<asp:ListView ID="ListView1" runat="server" DataKeyNames="Custid"InsertItemPosition="LastItem" >
 <LayoutTemplate>
  <table border="1">
   <tr>
    <th>Custid</th><th>Name</th><th>Balance</th><th>Address</th><th>Status</th><th>Action</th>
   </tr>
   <asp:PlaceHolder id="ItemPlaceHolder" runat="server" />
  </table>
 </LayoutTemplate>
```

```
  <ItemTemplate>
    <tr>
      <td><%# Eval("Custid") %></td><td><%# Eval("Name") %></td>
      <td><%# Eval("Balance") %></td><td><%# Eval("Address") %></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                Checked='<%# Eval("Status") %>' /></td>
      <td align="center"> <asp:LinkButton ID="btnEdit" runat="server" Text="Edit" CommandName="Edit" />
        <asp:LinkButton ID="btnDelete" runat="server" Text="Delete" CommandName="Delete"
                OnClientClick="return confirm('Are you sure of deleting the record?')" />
      </td>
    </tr>
  </ItemTemplate>
  <InsertItemTemplate>
    <tr>
      <td></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" /></td>
      <td><asp:TextBox ID="txtAddress" runat="server" Width="150" /></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" /></td>
      <td align="center">
        <asp:LinkButton ID="btnInsert" runat="server" Text="Insert" CommandName="Insert" /></td>
    </tr>
  </InsertItemTemplate>
  <EditItemTemplate>
    <tr>
      <td><%# Eval("Custid")%></td>
      <td><asp:TextBox ID="txtName" runat="server" Width="150" Text='<%# Eval("Name")%>' /></td>
      <td><asp:TextBox ID="txtBalance" runat="server" Width="150" Text='<%# Eval("Balance")%>' /></td>
      <td><asp:TextBox ID="txtAddress" runat="server" Width="150" Text='<%# Eval("Address")%>' /></td>
      <td align="center"><asp:CheckBox ID="cbStatus" runat="server" Enabled="false"
                        Checked='<%# Eval("Status")%>' /></td>
      <td align="center">
        <asp:LinkButton ID="btnUpdate" runat="server" Text="Update" CommandName="Update" />
        <asp:LinkButton ID="btnCancel" runat="server" Text="Cancel" CommandName="Cancel" />
      </td>
    </tr>
  </EditItemTemplate>
</asp:ListView>
```

Now go to design view of the WebForm, double click on ItemEditing, ItemCanceling, ItemInserting, ItemUpdating, and ItemDeleting events of ListView control and write following code under "aspx.cs" file:

Declarations: *ASPDBDataContext dc;*

Under Page_Load: *dc = new ASPDBDataContext(); if (!IsPostBack) { LoadData(); }*

```
private void LoadData() {
  ListView1.DataSource = dc.Customer_Select(null, true); ListView1.DataBind();
}
```

Under ListView ItemInserting:

```
string Name = ((TextBox)e.Item.FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)e.Item.FindControl("txtBalance")).Text);
string Address = ((TextBox)e.Item.FindControl("txtAddress")).Text;
bool Status = ((CheckBox)e.Item.FindControl("cbStatus")).Checked;
int? Custid = null; dc.Customer_Insert(Name, Balance, Address, Status, ref Custid);
if (Custid != null) { LoadData(); }
else { Response.Write("<script>alert('Insert operation failed.')</script>"); }
```

Under ListView ItemEditing: *ListView1.EditIndex = e.NewEditIndex; LoadData();*

Under ListView ItemCanceling: *ListView1.EditIndex = -1; LoadData();*

Under ListView ItemUpdating:

```
int Custid = (int)e.Keys["Custid"];
string Name = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtName")).Text;
decimal Balance = decimal.Parse(((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtBalance")).Text);
string Address = ((TextBox)ListView1.Items[e.ItemIndex].FindControl("txtAddress")).Text;
if (dc.Customer_Update(Custid, Name, Balance, Address) == 0) {
  ListView1.EditIndex = -1; LoadData();
}
else { Response.Write("<script>alert('Update operation failed.')</script>"); }
```

Under ListView ItemDeleting:

```
int Custid = (int)e.Keys["Custid"];
if (dc.Customer_Delete(Custid) == 0) { LoadData(); }
else Response.Write("<script>alert('Delete operation failed.')</script>");
```

Implementing Complex Queries using LINQ:

To try this first drag and drop Department and Employee tables from our ASPDB database on the LHS panel of OR Designer and add a new WebForm naming it as “ComplexQueries.aspx” and place a GridView control on it. Now go to “aspx.cs” file and write the following code:

Declarations:

```
ASPDBDataContext dc;
```

Under Page_Load:

```
dc = new ASPDBDataContext(); if (!IsPostBack) { LoadData(); }
```

```
private void LoadData() {
  //var tab = from C in dc.Customers select C;
  //var tab = from C in dc.Customers select new { C.Custid, C.Name, IsActive = C.Status };
  //var tab = from C in dc.Customers where C.Address == "Hyderabad" select C;
  //var tab = from C in dc.Customers where C.Balance > 20000 select C;
  //var tab = from C in dc.Customers where C.Address == "Hyderabad"&& C.Balance > 10000 select C;
  //var tab = from C in dc.Customers where C.Address == "Hyderabad" || C.Balance > 10000 select C;
  //var tab = from C in dc.Customers orderby C.Balance select C;
  //var tab = from C in dc.Customers orderby C.Name descending select C;
  //var tab = from C in dc.Customers group C by C.Address into G select new { City = G.Key, CustCount = G.Count() };
  //var tab = from C in dc.Customers group C by C.Address into G where G.Count() > 1  select new { City = G.Key,
          CustCount = G.Count() };
```

```
 //var tab = from C in dc.Customers group C by C.Address into G where G.Count() > 1 orderby G.Key descending
        select new { City = G.Key, CustCount = G.Count() };
 //var tab = from C in dc.Customers group C by C.Address into G select new { City = G.Key,
        MaxBalance = G.Max(C => C.Balance) };
 //var tab = from E in dc.Employees group E by E.Job into G select new { Job = G.Key, EmpCount = G.Count() };
 //var tab = from E in dc.Employees group E by E.Did into G select new { Did = G.Key,
        TotalSal = G.Sum(E => E.Salary) };
 //var tab = from E in dc.Employees where E.Job = "Clerk" group E by E.Did into G where G.Count() > 1 orderby
        G.Key descending select new { Did = G.Key, ClerkCount =G.Count() };
 var tab = from E in dc.Employees join D in dc.Departments on E.Did equals D.Did select new { E.Eid, E.Ename,
        E.Job, E.Salary, E.Did, D.Dname, D.Location };

 GridView1.DataSource = tab;
 GridView1.DataBind();
}
```

LINQDataSource Control:

Just like we have SqlDataSource control, we have LinqDataSource control also using which we can perform CRUD operations using LINQ without writing any code manually.

To test this add a new WebForm under the project naming it as "LinqDS.aspx" and place a GridView control on it and also add a LinqDataSource control, go to its properties => change the ID as "EmployeeDS" and click on the top right corner button, select "Configure Data Source" option which open a window in that select our "LinqToSqlProject.ASPDBDataContext" and click Next button, now under the tables select "Employees Table<Employee>" and click finish and select the CheckBox's "Enable Delete", "Enable Insert" and "Enable Update" CheckBoxs.

Now click on the top right corner button of GridView control we placed earlier and under Choose DataSource option select "EmployeeDS" and select the CheckBoxs "Enable Paging", "Enable Sorting", "Enable Editing" and "Enable Deleting", and run the WebForm which displays the data of Employee table.

In the above case when we click on the Edit Button of a record it provides the TextBox's for editing that record but in case of Did column the value we want to change must be a value that is present under the Department table, so without a normal TextBox to edit, it will be better if a DropDownList control comes there displaying all the Department Names present in Department table so that user can choose a value from the list to update. To do this add one more LinqDataSource control, change its Id as "DepartmentDS", click on the top right corner of it and select our "LinqToSqlProject.ASPDBDataContext" and click Next button, now under the tables select "Departments Table<Department>" and click finish. Now go to source view and under the GridView columns delete the "Did" Bound Field and write the following code in that place:

```
<asp:TemplateField HeaderText="Did" SortExpression="Did">
 <ItemTemplate><%# Eval("Did")%></ItemTemplate>
 <EditItemTemplate>
  <asp:DropDownList ID="ddlDepts" runat="server" DataSourceID="DepartmentDS" DataTextField="Dname"
        DataValueField="Did" SelectedValue='<%# Bind("Did") %>' />
 </EditItemTemplate>
</asp:TemplateField>
```

Now run the WebForm again and we can see a DropDownList in EditMode under "Did" column displaying Department Names to choose.

Authentication and Authorization

Authentication is the process of obtaining identification credentials such as name and password from a user and validating those credentials against some authority. If the credentials are valid, the user that submitted the credentials is considered an authenticated user. Once the user has been authenticated, the authorization process determines whether that user has access to a resource or not.

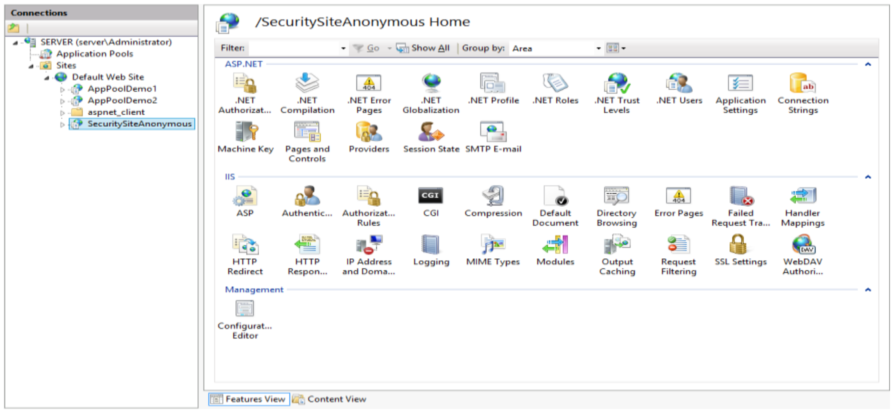
ASP.Net supports different types of authentications like:

1. Anonymous Authentication
2. Windows Authentication
3. Forms Authentication
4. Open Authentication

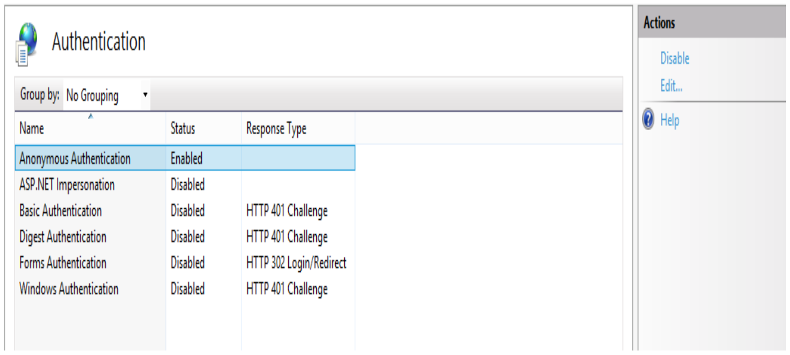
Anonymous Authentication:

This gives users access to the public areas of our Web site, Web application, or Web service without asking them for a user name or password. By default, the IUSR account, which was introduced in IIS 7.0 and replaces the IIS 6.0 IUSR_ComputerName account, is used to allow anonymous access. By default, IIS uses Anonymous authentication for every Web site, Web application, or Web service.

To test the above open a new Web Application naming it as "SecuritySiteAnonymous", enable it to run under IIS by creating a Virtual Directory and we find our App. under Connection Panel of IIS Manager as following:

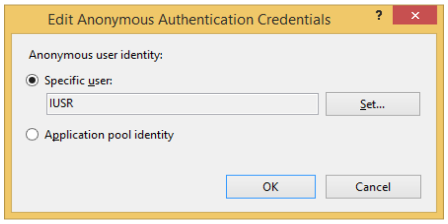


Now when we select the site, on the right side in the "Features View" we find "Authentication" option double click on it which displays Authentication details over there and we find Anonymous Authentication enabled and the other authentication modes disabled as following:



Name	Status	Response Type
Anonymous Authentication	Enabled	
ASP.NET Impersonation	Disabled	
Basic Authentication	Disabled	HTTP 401 Challenge
Digest Authentication	Disabled	HTTP 401 Challenge
Forms Authentication	Disabled	HTTP 302 Login/Redirect
Windows Authentication	Disabled	HTTP 401 Challenge

Now in the right hand side we find "Actions Panel" displaying "Edit…" option and when we click on it a window called "Edit Anonymous Authentication Credentials" get opened and under it we can find the user account IIS is using for anonymous access i.e. IUSR as following:


Edit Anonymous Authentication Credentials
?
Anonymous user identity:
Specific user:
IUSR
Set...
Application pool identity
OK
Cancel

Now add a WebForm under the application name it as "CheckAuthentication.aspx" and write the following code in "aspx.cs" file under Page_Load method:

Response.Write("Authentication Type: " + User.Identity.AuthenticationType + "<br />");
Response.Write("User Name: " + User.Identity.Name + "<br />");
Response.Write("Is Authenticated: " + User.Identity.IsAuthenticated + "<br />");

When we run the WebForm it will not display "Authentication Type" and "User Name" in case of Anonymous Authentication and "Is Authenticated" value will be false. We need to disable Anonymous Authentication for our site in case if we want to enable any other Authentication Mode and to disable Anonymous Authentication, in IIS Manager under "Authentication Panel" right click on "Anonymous Authentication" and select "Disable". Now if we run our WebForm again we get "HTTP Error 401.2 – Unauthorized" error.

Even if "Anonymous Authentication" is enabled it is still possible to restrict access to pages in our site by setting the Authorization header in our Web.config file as following under <system.web> tag:

<authorization><deny users="?"/></authorization> => ("?" indicates anonymous users)

To test the above enable Anonymous Authentication again, come back to our "SecuritySiteAnonymous" web application and write the above code in Web.config file under <system.web> tag and run the WebForm again which displays a Server Error "Access is denied".

Windows Authentication:

We use Windows authentication when our IIS server runs on a corporate network that is using Microsoft Active Directory service domain identities or other Windows accounts to identify users. Windows authentication is a secure form of authentication because the user name and password are hashed (encrypted) before being sent across the network. When you enable Windows authentication, the client browser sends a strongly hashed version of the password in a cryptographic exchange to the Web server. This authentication mode is best suitable for Intranet Applications.

To test Windows Authentication open a new Web Application naming it as "SecuritySiteWindows", enable it to run under IIS by creating a Virtual Directory. Now go to IIS Manager, select "SecuritySiteWindows", double click on Authentication option on the right side, "Disable" Anonymous Authentication and "Enable" Windows Authentication. Now add a new WebForm naming it as "CheckAuthentication.aspx" and write the following code in "aspx.cs" file under Page_Load method:

Response.Write("Authentication Type: " + User.Identity.AuthenticationType + "<br />");
Response.Write("User Name: " + User.Identity.Name + "<br />");
Response.Write("Is Authenticated: " + User.Identity.IsAuthenticated + "<br />");

Now when we run the WebForm it displays "Authentication Type: Negotiate" in case of Windows Authentication, "User Name: MachineName\UserName (Server\Administrator)" (where Server is my machine name and Administrator is the name of the user I am logged in) and "Is Authenticated: True".

In this authentication mode we can access the application by using any windows user account that is configured and we can try this by logging in to the computer by using any other user account.

Note: if no other user account is present on your machine we can create a new user account and to do this, right click on "This PC" icon on the desktop and select "Manage" which opens "Computer Management" window and in that we find "Local Users and Groups" in LHS, expand it and select Users which display users on your computer. To create a new user account right click on "Users" and select "New User" option which opens a window and in that enter "User Name", "Password" and "Confirm Password" options and click create, now use the switch user option, log in by using the new user account, open browser and enter the following URL to open the web page: "http://localhost/SecuritySiteWindows/CheckAuthentication.aspx".

It is possible to provide access to particular users only by specifying those users list in "Web.config" file under <system.web> tag as following:

```
<authorization>
  <allow users="server\Administrator"/>
  <deny users="*"/>                    =>        ("*" indicates all other authenticated users)
</authorization>
```

Note: Now it provides access only to Administrator user but not to any other user.

User Groups: under the Operating System apart from Users we also find Groups and to check this open "Computer Management" window, expand "Local Users and Groups" option which displays Users and Groups below, when we select users it displays existing users and when we select groups displays existing groups. Every group is a collection of users and every user must be members of some group, to check this, select users, double click on any user name which displays a window "Administrator Properties" and in that we find a tab "Member Of" click on it which displays the list of groups to which this user belongs to.

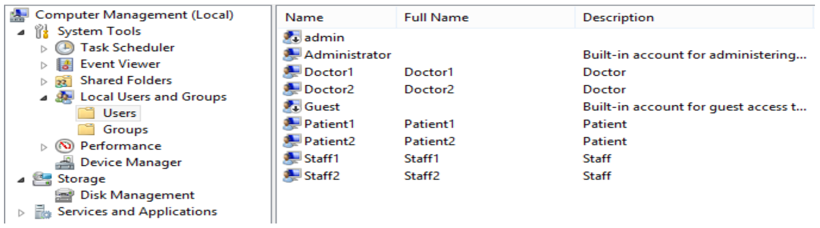
It is possible to provide access to users based on their roles so that all users under that role will be getting access to the site and to that we need to write following code in "Web.config" file in place of our previous code:

```
<authorization>
  <allow roles="Administrators"/>
  <deny users="*"/>
</authorization>
```

Note: Now it provides access to all users that belong to Administrators group.

Folder Level Authorization:

It is possible to provide access to particular pages of a Web Site or Web Application to Users or Groups with the help of Folder Level Authorization and to do this let us create a Hospital Management Site which contains 3 types of users: Doctor, Staff and Patient, where access should be given to only those pages required by that user or group. To test this first open Computer Management window and create 6 users on it as Doctor1, Doctor2, Staff1, Staff2, Patient1 and Patient2 which look as following:



Name	Full Name	Description
admin		
Administrator		Built-in account for administering...
Doctor1	Doctor1	Doctor
Doctor2	Doctor2	Doctor
Guest		Built-in account for guest access t...
Patient1	Patient1	Patient
Patient2	Patient2	Patient
Staff1	Staff1	Staff
Staff2	Staff2	Staff

Now create 3 new groups naming them as "DoctorGroup", "StaffGroup" and "PatientGroup", and to do this right click on Groups, select "New Group" which opens a Window, enter group name in it as "DoctorGroup", click on "Add" button which opens a window in that under the last TextBox enter "Doctor1" click "Check Names" button which displays "ComputerName\Doctor1" and repeat the same process to add "Doctor2" also to this group. Now follow the same process to create "StaffGroup" and "PatientGroup" also and now under the Groups we find our 3 new groups.

Now open a new ASP.Net Web Application naming it as “HospitalMgmt” and enable it to run under IIS by creating a Virtual Directory, go to IIS Manager, select “HospitalMgmtSite”, and double click on Authentication option on the right side disable Anonymous Authentication and Enable Windows Authentication. Now open Solution Explorer, add 3 new folders under the application naming them as “Doctor”, “Staff” and “Patient”, and now under all the 3 folders add a “Web.config” file and write the following code:

Doctor	Staff	Patient
<authorization> *<allow roles="DoctorGroup"/>* *<deny users="*"/>* *</authorization>*	*<authorization>* *<allow roles="StaffGroup"/>* *<deny users="*"/>* *</authorization>*	<authorization> <allow roles="PatientGroup"/> <deny users="*"/> </authorization>

Now add one WebForm in each folder naming it as “DoctorHomePage.aspx”, “StaffHomePage.aspx” and “PatientHomePage.aspx” and write the following code under the <div> tag of each page:

DoctorHomePage.aspx:

```
<h1 style="text-align: center; background-color: yellowgreen; color: orangered">Appollo Hospitals Ltd.</h1>
<h2>Welcome to Doctor Page</h2>
```

StaffHomePage.aspx:

```
<h1 style="text-align: center; background-color: yellowgreen; color: orangered">Appollo Hospitals Ltd.</h1>
<h2>Welcome to Staff Page</h2>
```

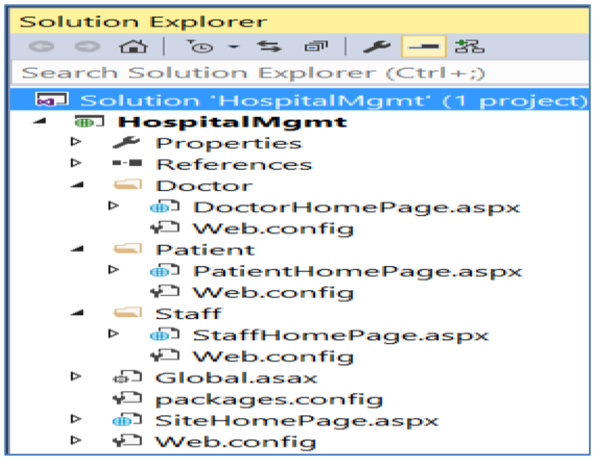
PatientHomePage.aspx:

```
<h1 style="text-align: center; background-color: yellowgreen; color: orangered">Appollo Hospitals Ltd.</h1>
<h2>Welcome to Patient Page</h2>
```

Now add a new WebForm under the application root folder with the name “SiteHomePage.aspx” and write the following code under <div> tag:

```
<h1 style="text-align: center;background-color: yellowgreen; color: orangered">Appollo Hospitals Ltd.</h1>
<center>
  <font size="5">Log in as:</font><br />
  <asp:HyperLink ID="hlDoctor" runat="server" NavigateUrl="~/Doctor/DoctorHomePage.aspx" Text="Doctor" />
  <asp:HyperLink ID="hlStaff" runat="server" NavigateUrl="~/Staff/StaffHomePage.aspx" Text="Staff" />
  <asp:HyperLink ID="hlPatient" runat="server" NavigateUrl="~/Patient/PatientHomePage.aspx" Text="Patient" />
</center>
```

Finally the application should look as following in Solution Explorer:



```
Solution 'HospitalMgmt' (1 project)
  HospitalMgmt
    Properties
    References
    Doctor
      DoctorHomePage.aspx
      Web.config
    Patient
      PatientHomePage.aspx
      Web.config
    Staff
      StaffHomePage.aspx
      Web.config
    Global.asax
    packages.config
    SiteHomePage.aspx
    Web.config
```

Forms Authentication:

Forms authentication lets us authenticate users by using our own code and then maintain an authentication token in a cookie or in the page URL. To use forms authentication, we must first create a login page that collects credentials from the user and that includes code to authenticate the credentials. Typically we configure the application to redirect requests to the login page when users try to access a protected resource, such as a page that requires authentication. If the user's credentials are valid, we can call methods of the FormsAuthentication class to redirect the request back to the originally requested resource with an appropriate authentication ticket (cookie), on subsequent requests, the user's browser passes the authentication cookie with the request, which then by passes the login page.

We configure forms authentication by using the authentication configuration element. In the simplest case, we have a login page and in the configuration file, we specify a URL to redirect un-authenticated requests to the login page. We then define valid credentials, either in the Web.config file or in a separate file or validate thru a database by our own code. This is best suitable for internet application than intranet application.

To understand the process, first open a new ASP.Net Web Application project naming it as "SecuritySiteForms" and enable it to run under IIS by creating a Virtual Directory. Now add 4 WebForms under the project naming it as "HomePage.aspx", "Page1.aspx", "Page2.aspx" and "LoginPage.aspx", and write the below code under each page:

Code under <div> tag of "HomePage.aspx":

```
<h1 style="text-align:center;background-color:greenyellow;color:orangered">Naresh I Technologies</h1>
<h3>This is Home Page of the site.</h3>
<ul>
  <li><asp:HyperLink ID="hlPage1" runat="server" Text="First Page" NavigateUrl="~/Page1.aspx" /></li>
  <li><asp:HyperLink ID="hlPage2" runat="server" Text="Second Page" NavigateUrl="~/Page2.aspx" /></li>
</ul>
```

Code under <div> tag of "Page1.aspx":

```
<h1 style="text-align:center;background-color:greenyellow;color:orangered">Naresh I Technologies</h1>
<h3>This is First Page of the site.</h3>
Back to <asp:HyperLink ID="hlBack" runat="server" Text="Home" NavigateUrl="~/HomePage.aspx" /> Page
```

Code under <div> tag of "Page2.aspx":

```
<h1 style="text-align:center;background-color:greenyellow;color:orangered">Naresh I Technologies</h1>
<h3>This is Second Page of the site.</h3>
Back to <asp:HyperLink ID="hlBack" runat="server" Text="Home" NavigateUrl="~/HomePage.aspx" /> Page
```

Code under <div> tag of "LoginPage.aspx":

```
<table align="center">
  <caption>Login Form</caption>
  <tr><td>User Id:</td><td><asp:TextBox ID="txtUid" runat="server" Width="200px" /></td></tr>
  <tr><td>Password:</td>
       <td><asp:TextBox ID="txtPwd" runat="server" Width="200px" TextMode="Password" /></td></tr>
  <tr><td><asp:CheckBox ID="cbRemember" runat="server" Text="Remember Me" /></td>
       <td><asp:Button ID="btnLogin" runat="server" Text="Login" Width="100px" />
         <asp:Button ID="btnReset" runat="server" Text="Reset" Width="100px" /></td></tr>
  <tr><td colspan="2"><asp:Label ID="lblStatus" runat="server" ForeColor="Red" /></td></tr>
</table>
```

Right now we can directly access HomePage or any other page that is present under the site without any restriction but if we want them to be accessed only after supplying the valid credentials, then we need to enable Forms Authentication thru the Web.config file, and to do that write the following code under <system.web> tag:

```
<authentication mode="Forms">
  <forms loginUrl="LoginPage.aspx">
    <credentials passwordFormat="Clear">
      <user name="Raju" password="raju1234" />
      <user name="Admin" password="admin1234" />
    </credentials>
  </forms>
</authentication>
```

Note: Password format is to specify whether password should be clear text or encrypted to SHA1 or MD5 formats.

Attributes of <forms> element:

- **loginUrl** is to specify the name of the page which has to be launched first when the site is accessed.
- **defaultUrl** is to specify the name of the default page which should be launched after logging in to the site if no other page is requested, if not specified default “defaultUrl” is “default.aspx”
- **name** is to specify name for the login cookie which will be “.ASPXAUTH” if not specified.
- **timeout** is to specify amount of time in minutes after which the cookie expires. The default value is 30.
- **slidingExpiration** is to specify whether sliding expiration is enabled and default is true. Sliding expiration value if true resets an active authentication cookie's time to expiration upon each request during a single session, whereas if it is false then the cookie expires at a set interval from the time it was originally issued.

After doing the above also we can access any page under the site directly because under IIS Anonymous Authentication is enabled for this site which must be either disabled in IIS or we can do that by adding authorization element under authentication element in Web.config file as following:

<authorization><deny users="?"/></authorization>

Now if we try to open any page under the site by default it redirects to “LoginPage.aspx” where we need to supply valid credentials to login, which will then redirect to the requested page, but to do that we need to write the following code under click event of Login button by importing the namespace System.Web.Security.

```
if (FormsAuthentication.Authenticate(txtUid.Text, txtPwd.Text))
  FormsAuthentication.RedirectFromLoginPage(txtUid.Text, cbRemember.Checked);
else
  lblStatus.Text = "Login failed, given credentials doesn't match.";
```

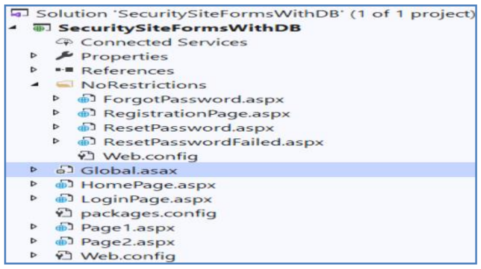
Authenticate is a static method of FormsAuthentication class which validates the given user name and password against credentials stored in configuration file and returns true if he is validated or else returns false.

RedirectFromLoginPage is a static method of FormsAuthentication class which will redirect an authenticated user back to the original requested URL or the default URL. The second parameter of the method is a Boolean value which should be true to create a persistent or durable cookie or else false for in-memory cookie.

Now if we try opening any page, first it will go to LoginPage and then after confirming the credentials it will redirect back to the page we requested, but if we directly open the LoginPage, then after entering the valid credentials also we get an error “resource not found (404)” because when we directly go to “LoginPage” then it will redirect us to “default.aspx” page which is not present in our site so to overcome this problem either we need to add “default.aspx” in our site or change this by overriding the “defaultUrl” under <forms> element of Web.config file as following: ***<forms loginUrl="LoginPage.aspx" defaultUrl="HomePage.aspx">***

Forms Authentication with Database:

Open a new ASP.Net Web Application naming it as "SecuritySiteFormsWithDB" and enable it to run under IIS by creating a Virtual Directory. Now add a new Folder under the Web Application naming it as "NoRestrictions" and add 4 WebForms under the folder naming them as "RegistrationPage.aspx", "ForgotPassword.aspx", "ResetPassword.aspx" and "ResetPasswordFailed.aspx", and also add a "Web.config" file. Now under the root application add 4 WebForms naming them as "LoginPage.aspx", "HomePage.aspx", "Page1.aspx" and "Page2.aspx". Now items under Solution Explorer should look as following:



```
Solution 'SecuritySiteFormsWithDB' (1 of 1 project)
  SecuritySiteFormsWithDB
    Connected Services
    Properties
    References
    NoRestrictions
      ForgotPassword.aspx
      RegistrationPage.aspx
      ResetPassword.aspx
      ResetPasswordFailed.aspx
      Web.config
    Global.asax
    HomePage.aspx
    LoginPage.aspx
    packages.config
    Page1.aspx
    Page2.aspx
    Web.config
```

Write the below code in Web.config file of application's root folder, under <system.web> tag :

```
<authentication mode="Forms">
  <forms loginUrl="LoginPage.aspx" defaultUrl="HomePage.aspx" />
</authentication>
<authorization><deny users="?"/></authorization>
<appSettings><add key="ValidationSettings:UnobtrusiveValidationMode" value="none"/></appSettings>
```

Write the following code in Web.config file present in NoRestrictions folder under <configuration> tag:

```
<system.web><authorization><allow users="?"/></authorization></system.web>
```

Write the following code under <div> tag of HomePage.aspx:

```
<h1 style="text-align:center;background-color:yellow;color:red">Welcome to Naresh I Technologies</h1>
<table border="1" align="center" style="text-align:center;width:20%">
<caption>Click to navigate:</caption>
<tr><td><asp:HyperLink ID="hlPage1" runat="server" Text="Page1" NavigateUrl="~/Page1.aspx" /></td></tr>
<tr><td><asp:HyperLink ID="hlPage2" runat="server" Text="Page2" NavigateUrl="~/Page2.aspx" /></td></tr>
</table>
```

Write the following code under <div> tag of Page1.aspx: `<h1>This is first page of the site.</h1>`

Write the following code under <div> tag of Page2.aspx: `<h1>This is second page of the site.</h1>`

Create the following tables under our ASPDB Database:

```
Create Table Users(UserId Varchar(20) Primary Key, Name Varchar(50), Password Varchar(200), Email
        Varchar(100), IsLocked Bit default 0, FailureCount TinyInt default 0, LockedDate Date)
Create Table ResetPasswordRequest(Guid UniqueIdentifier Primary Key, UserId Varchar(20) References
        Users(UserId), RequestDate Date)
```

Create the following Stored Procedures under our ASPDB Database:

```
Create Procedure Users_Insert(@UserId Varchar(20), @Name Varchar(50), @Password Varchar(200), @Email
Varchar(100), @Count int out)
As
Begin
 Select @Count = Count(*) From Users Where UserId = @UserId;
 If @Count = 0
   Insert Into Users (UserId, Name, Password, Email)    Values (@UserId, @Name, @Password, @Email);
End;
```

```
Create Procedure Users_Validate(@UserId Varchar(20), @Password Varchar(200), @Status Int Out)
As
Begin
 Declare @IsLocked Bit, @FailureCount TinyInt;
 If Not Exists(Select * From Users Where UserId=@UserId)
 Begin
  Set @Status = -1;
 End
 Else
 Begin
  Select @IsLocked=IsLocked From Users Where UserId=@UserId;
  If @IsLocked = 1
  Begin
   Set @Status = -2;
  End;
  Else
  Begin
   If Exists(Select * From Users Where UserId=@UserId And Password=@Password)
   Begin
    Set @Status = 0; Update Users Set FailureCount = 0 Where UserId=@UserId;
   End;
   Else
   Begin
    Select @FailureCount=FailureCount From Users Where UserId=@UserId;
    If @FailureCount = 2
    Begin
     Set @Status = -3; Update Users Set IsLocked=1, LockedDate=GetDate() Where UserId=@UserId;
    End;
    Else
    Begin
     Set @FailureCount += 1; Set @Status = @FailureCount;
     Update Users Set FailureCount=@FailureCount Where UserId=@UserId;
    End;
   End;
  End;
 End;
End;
```

```
Create Procedure ResetPasswordRequest_Insert(@UserId Varchar(20), @Name Varchar(50) Out, @Email
Varchar(100) Out, @Guid UniqueIdentifier Out)
As
Begin
 Set @Name = Null; Set @Email = Null; Set @Guid = Null;
 If Exists(Select * From Users Where UserId=@UserId)
 Begin
  If Exists(Select * From Users Where UserId=@UserId And IsLocked=0)
  Begin
```

```
    Set @Guid = NewId();Select @Name=Name, @Email=Email From Users Where UserId=@UserId;
    Insert Into ResetPasswordRequest Values(@Guid, @UserId, GetDate());
   End;
  End;
End;
```

```
Create Procedure ResetPasswordLink_IsValid(@Guid UniqueIdentifier, @Status Int Out)
As
Begin
  Declare @StartDate Date;
  If Exists(Select * From ResetPasswordRequest Where Guid=@Guid)
  Begin
    Select @StartDate = RequestDate  From ResetPasswordRequest Where Guid=@Guid;
    If DateDiff(d, @StartDate, GetDate()) > 30
    Begin
      Delete From ResetPasswordRequest Where Guid=@Guid;
      Set @Status = 0;
    End;
    Else
      Set @Status = 1
  End;
  Else
    Set @Status = -1;
End;
```

```
Create Procedure Users_ResetPassword(@Guid UniqueIdentifier, @Password Varchar(200), @Status Int Out)
As
Begin
 Declare @UserId Varchar(20);
 Select @UserId = UserId From ResetPasswordRequest Where Guid=@Guid;
 Update Users Set Password=@Password, FailureCount=0 Where UserId=@UserId;
 If @@ROWCOUNT = 1
 Begin
  Delete From ResetPasswordRequest Where Guid=@Guid;Set @Status = 1;
 End;
 Else
  Set @Status = 0;
End;
```

Add OR Designer under the project naming it as "ASPDB.dbml", configure our ASPDB database under Server Explorer of Visual Studio, drag and drop all the above 5 stored procedures on the RHS panel of the OR Designer to generate mapping methods.

Write the following code under <div> tag of RegistrationPage.aspx:

```
<table align="center">
<caption>Registration Form</caption>
<tr>
<td>Name:</td><td><asp:TextBox ID="txtName" runat="server" Width="150px"></asp:TextBox></td>
```

```
<td><asp:RequiredFieldValidator ID="rfvName" runat="server" ControlToValidate="txtName"
        Display="Dynamic" ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
</tr>
<tr>
<td>User Id:</td><td><asp:TextBox ID="txtId" runat="server" Width="150px"></asp:TextBox></td>
<td><asp:RequiredFieldValidator ID="rfvId" runat="server" ControlToValidate="txtId" Display="Dynamic"
        ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
</tr>
<tr>
<td>Password:</td>
<td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" Width="150px"></asp:TextBox></td>
<td><asp:RequiredFieldValidator ID="rfvPwd" runat="server" ControlToValidate="txtPwd" Display="Dynamic"
        ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
</tr>
<tr>
<td>Confirm Pwd:</td>
<td><asp:TextBox ID="txtCPwd" runat="server" TextMode="Password" Width="150px"></asp:TextBox></td>
<td><asp:CompareValidator ID="cvCPwd" runat="server" ControlToCompare="txtPwd" Display="Dynamic"
    ControlToValidate="txtCPwd" ErrorMessage="Confirm pwd should match with pwd." ForeColor="Red" /></td>
</tr>
<tr>
<td>Email Id:</td>
<td><asp:TextBox ID="txtEmail" runat="server" TextMode="Email" Width="150px"></asp:TextBox></td>
<td><asp:RequiredFieldValidator ID="rfvEmail" runat="server" ControlToValidate="txtEmail" Display="Dynamic"
          ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
</tr>
<tr>
<td align="center" colspan="2"><asp:Button ID="btnRegister" runat="server" Text="Register" />
<asp:Button ID="btnReset" runat="server" Text="Reset" Width="100px" /></td>
<td>&nbsp;</td>
</tr>
<tr>
<td colspan="3"><asp:Label ID="lblStatus" runat="server" ForeColor="Red"></asp:Label></td>
</tr>
</table>
```

Write the following code under RegistrationPage.aspx.cs file:

```
using System.Web.Security;
```

Under Register Button:

```
int? Count = null;
string EncPwd = FormsAuthentication.HashPasswordForStoringInConfigFile(txtPwd.Text, "MD5");
ASPDBDataContext dc = new ASPDBDataContext();
dc.Users_Insert(txtId.Text, txtName.Text, EncPwd, txtEmail.Text, ref Count);
if (Count > 0)
  lblStatus.Text = "User Id already exists.";
else
  Response.Redirect("~/LoginForm.aspx");
```

Under Reset Button:

```
txtId.Text = txtName.Text = txtPwd.Text = txtCPwd.Text = txtEmail.Text = "";
txtId.Focus();
```

Write the following code under <div> tag of LoginPage.aspx:

```
<table align="center">
<caption>Login Form</caption>
<tr>
<td align="right">User Id:</td>
<td><asp:TextBox ID="txtUserId" runat="server" Width="150px" /></td>
</tr>
<tr>
<td align="right">Password:</td>
<td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" Width="150px" /></td>
</tr>
<tr>
<td><asp:CheckBox ID="cbRemember" runat="server" Text="Remember Me" /></td>
<td><asp:Button ID="btnLogin" runat="server" Text="Login" Width="75px" />
<asp:Button ID="btnReset" runat="server" Text="Reset" Width="75px" /></td>
</tr>
<tr>
<td colspan="2">New user&nbsp;<asp:HyperLink ID="hlRegister" runat="server"
        NavigateUrl="~/NoRestrictions/RegistrationPage.aspx">click</asp:HyperLink>&nbsp;to register</td>
</tr>
<tr>
<td colspan="2">Forgot password&nbsp;<asp:HyperLink ID="HyperLink1" runat="server"
        NavigateUrl="~/NoRestrictions/ForgotPassword.aspx">click</asp:HyperLink>&nbsp;to reset</td>
</tr>
<tr>
<td colspan="2"><asp:Label ID="lblStatus" runat="server" ForeColor="Red"></asp:Label></td>
</tr>
</table>
```

Write the following code under LoginPage.aspx.cs file:

```
using System.Web.Security;
```

Under Login Button:

```
int? Status = null;
string EncPwd = FormsAuthentication.HashPasswordForStoringInConfigFile(txtPwd.Text, "MD5");
ASPDBDataContext dc = new ASPDBDataContext();
dc.Users_Validate(txtUserId.Text, EncPwd, ref Status);
if (Status == -1) { lblStatus.Text = "No user is existing with given UserId."; }
else if (Status == -2) { lblStatus.Text = "Your account is already locked, contact the admin team."; }
else if (Status == -3) { lblStatus.Text = "Your account is locked, contact the admin team."; }
else if (Status == 0) { FormsAuthentication.RedirectFromLoginPage(txtUserId.Text, cbRemember.Checked); }
else { lblStatus.Text = "You made " + Status + " failure attempt(s), maximum is 3."; }
```

Under Reset Button:

```
txtUserId.Text = txtPwd.Text = ""; cbRemember.Checked = false;
txtUserId.Focus();
```

Write the following code under <div> tag of ForgotPassword.aspx:

```
<table align="center">
<caption>Forgot Your Password?</caption>
<tr><td>Enter your UserId to recieve a reset passwork link.</td></tr>
<tr>
<td>Enter User Id: <asp:TextBox ID="txtUserId" runat="server" Width="150px"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvUserId" runat="server" ControlToValidate="txtUserId"
            Display="Dynamic" ErrorMessage="Can't leave the field empty." ForeColor="Red" />
</td>
</tr>
<tr><td align="center"><asp:Button ID="btnSubmit" runat="server" Text="Submit" /></td></tr>
<tr><td><asp:Label ID="lblStatus" runat="server" ForeColor="Red" /></td></tr>
</table>
```

Write the following code under ForgotPassword.aspx.cs file:

```
using System.Net; using System.Text; using System.Net.Mail;
```

Under Submit Button:

```
Guid? Uid = null; string Name = null, Email = null;
ASPDBDataContext dc = new ASPDBDataContext();
dc.ResetPasswordRequest_Insert(txtUserId.Text, ref Name, ref Email, ref Uid);
if (Email != null) {
  SendRequestMail(Email, Name, Uid.ToString());
  lblStatus.Text = "Reset password link has been sent to your registered email.";
}
else { lblStatus.Text = "Given UserId is either invalid or account is already locked."; }
```

```
private void SendRequestMail(string Email, string Name, string Uid) {
  StringBuilder StrMsg = new StringBuilder("Mr./Ms./Mrs." + Name);
  StrMsg.Append("<br/><br/>Click on the below link to reset your password:<br/>");
  StrMsg.Append("http://localhost/SecuritySiteFormsAuthDB/NoRestrictions/ResetPassword.aspx?uid=" + Uid);
  StrMsg.Append("<br/><br/>Regards"); StrMsg.Append("<br/><br/>NIT Team");
  MailMessage MailMsg = new MailMessage("<your email id here>", Email);
  MailMsg.IsBodyHtml = true; MailMsg.Subject = "Reset Password Link"; MailMsg.Body = StrMsg.ToString();
  SmtpClient MailSender = new SmtpClient("smtp.gmail.com", 587);
  MailSender.Credentials = new NetworkCredential("<your email id here>", "<your email password here>");
  MailSender.EnableSsl = true; MailSender.Send(MailMsg);
}
```

Write the following code under <div> tag of ResetPassword.aspx:

```
<table align="center">
<caption>Reset Password</caption>
<tr>
```

```
<td align="right">Enter New Password:</td>
<td><asp:TextBox ID="txtPwd" runat="server" TextMode="Password" Width="150px" /></td>
<td><asp:RequiredFieldValidator ID="rfvPwd" runat="server" ControlToValidate="txtPwd" Display="Dynamic"
        ErrorMessage="Can't leave the field empty." ForeColor="Red" /></td>
</tr>
<tr>
<td>Confirm New Password:</td>
<td><asp:TextBox ID="txtCPwd" runat="server" TextMode="Password" Width="150px" /></td>
<td><asp:CompareValidator ID="cvCPwd" runat="server" Display="Dynamic" ControlToCompare="txtPwd"
        ControlToValidate="txtCPwd" ErrorMessage="Should match with password." ForeColor="Red" /></td>
</tr>
<tr>
<td colspan="3" align="center"><asp:Button ID="btnSave" runat="server" Text="Save" /></td>
</tr>
<tr>
<td colspan="3"><asp:Label ID="lblStatus" runat="server" ForeColor="Red"></asp:Label></td>
</tr>
</table>
```

Write the following code under ResetPassword.aspx.cs file:

```
using System.Web.Security;
```

Under Page_Load:

```
if (!IsPostBack) {
  int? Status = null; string Uid = Request.QueryString["uid"];
  ASPDBDataContext dc = new ASPDBDataContext(); dc.ResetPasswordLink_IsValid(Guid.Parse(Uid), ref Status);
  if (Status == -1 || Status == 0) {
    Response.Redirect("~/NoRestrictions/ResetPasswordFailed.aspx?Status=" + Status);
  }
}
```

Under Save Button:

```
int? Status = null;
string Uid = Request.QueryString["uid"];
string EncPwd = FormsAuthentication.HashPasswordForStoringInConfigFile(txtPwd.Text, "MD5");
ASPDBDataContext dc = new ASPDBDataContext(); dc.Users_ResetPassword(Guid.Parse(Uid), EncPwd, ref Status);
if (Status == 1) { lblStatus.Text = "Your password has been reset successfully."; }
else { lblStatus.Text = "Failed resetting the password, please contact the admin team."; }
```

Write the following code under ResetPasswordFailed.aspx.cs file:

Under Page_Load:

```
int Status = int.Parse(Request.QueryString["Status"]);
if (Status == 0) { Response.Write("<h1>Your reset password link is expired.</h1>"); }
else if(Status == -1) { Response.Write("<h1>Your reset password link is already used.</h1>"); }
```

Note: to send mails from your gmail account first you need to go to gmail account settings i.e. My Account => Click Sign-in & Security => scroll down to “Password & sign-in method” => under that set the “2-Step Verification” => change value to “Off”, scroll down to “Allow less secure apps” and set it as “ON”.

Caching

One of the most important factors in building high-performance, scalable Web applications is the ability to store items, whether data objects or pages, or parts of a page, in memory the initial time they are requested. You can cache, or store, these items on the Web server or other software in the request stream, such as the proxy server or browser. This allows you to avoid recreating information that satisfied a previous request, particularly information that demands significant processor time or other resources. ASP.NET caching allows you to use a number of techniques to store page output or application data across HTTP requests and reuse it.

An application can often increase performance by storing data in memory that is accessed frequently and that requires significant processing time to create. For example, if your application processes large amounts of data using complex logic and then returns the data as a report accessed frequently by users, it is efficient to avoid re-creating the report every time that a user requests it. Similarly, if your application includes a page that processes complex data but that is updated only in-frequently, it is in-efficient for the server to re-create that page on every request. To help us increase application performance in these situations, ASP.NET provides caching which is divided into 2 categories like:

1. **Output Caching**
2. **Data Caching**

Output Caching:

Output caching allows you to store dynamic page and user control responses on any HTTP cache-capable device in the output stream, from the originating server to the requesting browser. On subsequent requests, the page or user control code is not executed; the cached output is used to satisfy the request.

To enable Output Caching we need to use "OutputCache Directive" either in a WebForm or a WebUserControl as following:

<%@ OutputCache Duration="<time in seconds>" VaryByParam="<none/value>" %>

- Duration is to specify the time period output should be stored in cache memory, in seconds.
- VaryByParam is to specify the parameter based on which the cached output should vary.

Open a new ASP.Net Web Application naming it as "CachingSite", add a new WebForm in it naming it as "OutputCache.aspx" and write the following code under <div> tag:

```
<asp:GridView ID="GridView1" runat="server" />
Client Machine Date & Time: <script>document.write(Date());</script>
```

Write the following code under "aspx.cs" file of the WebForm:

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

Under Page_Load:

```
SqlConnection con = new SqlConnection("Data Source=Server;User Id=Sa;Password=123;Database=ASPDB");
SqlDataAdapter da = new SqlDataAdapter("Select * from Customer", con);
DataSet ds = new DataSet(); da.Fill(ds, "Customer"); GridView1.DataSource = ds; GridView1.DataBind();
Response.Write("Server Machine Date & Time: " + DateTime.Now.ToString());
```

Now if you run the WebForm the Server Time value and Browser Time value will be same even if we refresh the page also where as if we enable caching then the Server Time and Browser Time values will be different, to test that go to OutputCache.aspx file and write the following code below the Page Directive:

<%@ OutputCache Duration="30" VaryByParam="none" %>

<u>VaryByParam attributeof Caching:</u>

Add a new WebForm under the project naming it as “MasterDetail.aspx” and write the following code under <div> tag:

```
<table>
<tr><td align="center"><asp:DropDownList ID="ddlDept" runat="server" AutoPostBack="true" /></td></tr>
<tr><td><asp:GridView ID="gvEmp" runat="server" /></td></tr>
</table>
Browser Time: <script>document.write(Date());</script>
```

Write the following code under “aspx.cs” file:

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

<u>Declarations:</u>

```
SqlConnection con; SqlDataAdapter da; DataSet ds;
```

<u>Under Page_Load:</u>

```
con = new SqlConnection("User Id=Sa;Password=123;Database=ASPDB;Data Source=Server");
if (!IsPostBack) {
  da = new SqlDataAdapter("Select * From Department", con); ds = new DataSet(); da.Fill(ds, "Department");
  da.SelectCommand.CommandText = "Select * From Employee"; da.Fill(ds, "Employee");
  ddlDept.DataSource = ds.Tables["Department"]; ddlDept.DataTextField = "Dname";
  ddlDept.DataValueField = "Did"; ddlDept.DataBind();ddlDept.Items.Insert(0, "All");
  gvEmp.DataSource = ds.Tables["Employee"]; gvEmp.DataBind();
}
```

<u>Code under DropDownListSelectedIndexChanged:</u>

```
if (ddlDept.SelectedIndex == 0)
  da = new SqlDataAdapter("Select * From Employee", con);
else
  da = new SqlDataAdapter("Select * From Employee Where Did=" + ddlDept.SelectedValue, con);
ds = new DataSet(); da.Fill(ds, "Employee");
gvEmp.DataSource = ds.Tables["Employee"]; gvEmp.DataBind();
Response.Write("Data Cached at: " + DateTime.Now.ToLongTimeString());
```

Now turn on caching under the above WebForm by using the OutputCache directive as following:

```
<%@ OutputCache Duration="300" VaryByParam="none" %>
```

Now run the page and select Department Name from DropDownList control and data gets cached at that point of time, but when we change the selection of Name it will still display the old cached output only because the cached output duration is 300 seconds, but we require the output to be changing based on the Department Name we select and to achieve that we need to use the VaryByParam attribute of OuputCache directive which should be as following:

```
<%@ OutputCache Duration="300" VaryByParam="ddlDept" %>
```

Now run the page again and watch the difference in output where we will notice the data being cached based on the value selected under the DropDownList control.

<u>Note:</u> If in any situation if we want the output should be varying based on multiple parameters we can even specify a comma separated list of values under VaryByParam.

<u>Fragment Caching:</u>

It is an approach of caching a part of a webpage which can be achieved through User Control i.e. if at all we want to cache a particular area in a webpage the data we wanted to cache must be first in the form of a User Control with caching enabled and then that control can be placed on the web form, so that the controls output will be cached but not the remaining content of the webpage.

To test this, add a “Web Form User Control” in the project naming it as “CustomerControl.ascx” and write the following code under the Control directive:

```
<table>
<tr><td><asp:Label ID="lblHeader" runat="server" Text="Label" /></td></tr>
<tr><td><asp:GridView ID="GridView1" runat="server" /></td></tr>
<tr><td><asp:Label ID=" lblFooter " runat="server" Text="Label" /></td></tr>
</table>
```

Now write the following code under the code behind file:

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

<u>Code under Page_Load method:</u>

```
SqlConnection con = new SqlConnection("User Id=Sa;Password=123;Database=ASPDB;Data Source=Server");
SqlDataAdapter da = new SqlDataAdapter("Select * From Customer", con);
DataSet ds = new DataSet(); da.Fill(ds, "Customer");
GridView1.DataSource = ds.Tables[0]; GridView1.DataBind();
lblHeader.Text = "Control output cached at: " + DateTime.Now.ToLongTimeString();
lblFooter.Text = "Control output cached at: " + DateTime.Now.ToLongTimeString();
```

Now to consume the control add a new WebForm naming it as “CustomerForm.aspx and write the following code under Page directive:

```
<%@ Register Src="~/CustomerControl.ascx" TagPrefix="NIT" TagName="CustomerControl" %>
```

Now write following code under <div> tag:

```
<table>
<tr><td><asp:Label ID="lblHeader" runat="server" Text="Label" /></td></tr>
<tr><td><NIT:CustomerControl runat="server" ID="CustomerControl1" /></td></tr>
<tr><td><asp:Label ID="lblFooter" runat="server" Text="Label" /></td></tr>
</table>
```

Now write the following code under Page_Load method:

```
    lblHeader.Text = "Form output generated at: " + DateTime.Now.ToLongTimeString();
    lblFooter.Text = "Form output generated at: " + DateTime.Now.ToLongTimeString();
```

When we run the current form we will notice the Form Output and Control Output generated time will be same because we have not used caching, to use caching now goto CustomerControl.ascx and write the following code under control directive:

```
<%@ OutputCache Duration="300" VaryByParam="none" %>
```

Now run the webform again to the see the difference in output and we will notice the control output cached but not the page output because caching is applied to control only.

VaryByControl in Caching:

Earlier we used an attribute known as VaryByParam in caching to differentiate the output generated based on the selection in DropDownList control whereas if at all the same was present in case of a UserControl we need to use VaryByControl attribute in place of VaryByParam attribute. To check this, add a new "Web Form User Control" under the project naming it as "MasterDetailControl.ascx" and write the following code under control directive:

```
<table>
<tr><td><asp:Label ID="lblHeader" runat="server" Text="Label"></asp:Label></td></tr>
<tr><td><asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True" /></td></tr>
<tr><td><asp:GridView ID="GridView1" runat="server" /></td></tr>
<tr><td><asp:Label ID="lblFooter" runat="server" Text="Label"></asp:Label></td></tr>
</table>
```

Now under the code behind file write the below code.

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

Declarations: *SqlConnection con; SqlDataAdapter da; DataSet ds;*

Code under Page_Load Method:

```
con = new SqlConnection("User Id=Sa;Password=123;Database=ASPDB;Data Source=Server");
if (!IsPostBack) {
da = new SqlDataAdapter("Select * From Department", con); ds = new DataSet(); da.Fill(ds, "Department");
da.SelectCommand.CommandText = "Select * From Employee"; da.Fill(ds, "Employee");
DropDownList1.DataSource = ds.Tables["Department"]; DropDownList1.DataTextField = "Dname";
DropDownList1.DataValueField = "Did"; DropDownList1.DataBind();DropDownList1.Items.Insert(0, "All");
GridView1.DataSource = ds.Tables["Employee"]; GridView1.DataBind();
}
```

Code under DropDownListSelectedIndexChanged:

```
if (DropDownList1.SelectedIndex == 0) { da = new SqlDataAdapter("Select * From Employee", con); }
else{ da = new SqlDataAdapter("Select * From Employee Where Did=" + DropDownList1.SelectedValue, con); }
ds = new DataSet(); da.Fill(ds, "Employee");GridView1.DataSource = ds.Tables["Employee"]; GridView1.DataBind();
lblHeader.Text = "Control output cached at: " + DateTime.Now.ToLongTimeString();
lblFooter.Text = "Control output cached at: " + DateTime.Now.ToLongTimeString();
```

Now add a new webform under the site naming it as MasterDetailForm.aspx and write the following code under Page directive:

```
<%@ Register Src="~/MasterDetailControl.ascx" TagPrefix="NIT" TagName="MasterDetailControl" %>
```

Now write the following code under <div> tag:

```
<table>
<tr><td><asp:Label ID="lblHeader" runat="server" Text="Label" /></td></tr>
<tr><td><NIT:MasterDetailControl runat="server" ID="MasterDetailControl1" /></td></tr>
<tr><td><asp:Label ID="lblFooter" runat="server" Text="Label" /></td></tr>
</table>
```

Now write the following code under Page_Load:

```
lblHeader.Text = "Form output generated at: " + DateTime.Now.ToLongTimeString();
lblFooter.Text = "Form output generated at: " + DateTime.Now.ToLongTimeString();
```

Now run the webform we will not notice any difference in the control output time and form output time because we did not enable caching, so to enable caching go to MasterDetailControl.ascx and write the following code under Control directive:

<%@ OutputCache Duration="300" VaryByControl="DropDownList1" %>

Data Caching:

It's a process of storing an amount of data under a Web Page into cache memory, which is very similar like storing data in "View State" or "Session State" or "Application State".

Syntax of storing value into cache: Cache[string key] = value (accepts value of type object)
Syntax of accessing values from cache: Object value = Cache[string key]

Note: apart from the above we can also store the data either by using Cache.Add and Cache.Insert methods also.

Storing data in Cache is very similar to storing data in "Application State" i.e. both are "multi-user global data", whereas the difference between Application State and Cache are:

1. Application state is not Thread Safe, whereas Cache Memory is Thread Safe.
2. We can access Application State data in Global.asax file, whereas we can't access Cache Memory in Global.asax.
3. In Application State the data stays for a long time i.e. until the server restarts or the worker process is recycled, whereas data in Cache Memory will not stay for long time i.e. server can remove the values in Cache Memory at any point of time.
4. In case of Application State, data will not have any dependencies, whereas in case of Cache Memory, data can have 3 types of dependencies:
   a) Time Based Dependency.
   b) File Based Dependency.
   c) SQL Based Dependency.

Time Based Dependency:

In this case the values in the Cache Memory will be removed based on a time i.e. when the time expires the data in the Cache Memory is deleted. This is again dividedin to 2 types:

1. **Absolute Expiration:** In this case the data in the Cache Memory stays for a specified time period which is calculated from the first time it was stored in the Cache and once the time elapses data in Cache Memory is removed.
2. **Sliding Expiration:** In this case the data in the Cache Memory stays for a specified time period which is calculated from the last visit and once the time elapses data in Cache Memory is removed.

To test this add a new WebForm under the site naming it as DataCachingTimeBased.aspx, place a GridView control on it and write the following code in code behind file.

using System.Data; using System.Web.Caching; using System.Data.SqlClient;

Under Page_Load:

```
DataSet ds;
if (Cache["CustomerDS"] == null) {
  SqlConnection con = new SqlConnection("User Id=Sa;Password=123;Database=ASPDB;Data Source=Server");
  SqlDataAdapter da = new SqlDataAdapter("Select * From Customer", con);
  ds = new DataSet();da.Fill(ds, "Customer");
  Cache.Insert("CustomerDS", ds); Response.Write("Data loaded from Database.");
}
else { ds = (DataSet)Cache["CustomerDS"];Response.Write("Data loaded from Cache."); }
GridView1.DataSource = ds.Tables[0]; GridView1.DataBind();
```

In the above case we haven't applied an dependency on the cache, so the values gets stored in the Cache Memory until the Server explicitly removes it, whereas we can apply any of the 3 types of dependencies we have discussed above. Now let us implement Time Based Dependency for the above program:

Implementing Absolute Expiration:

To implement this re-write the Cache.Insert method in the above program as following:

```
Cache.Insert("CustomerDS", ds, null, DateTime.UtcNow.AddSeconds(30), Cache.NoSlidingExpiration);
```

In the above case because we have applied Absolute Expiration to Cache, data stays in the Cache Memory for 30 seconds from the point it was stored in the cache.

Implementing Sliding Expiration:

To implement this re-write the Cache.Insert method in the above program as following:

```
Cache.Insert("CustomerDS", ds, null, Cache.NoAbsoluteExpiration, TimeSpan.FromSeconds(30));
```

In the above case because we have applied Sliding Expiration to Cache, data stays in the Cache Memory for 30 seconds from the last visit to the page.

File Based Dependency:

In this case the data in the Cache is dependent on some file and whenever the data in the file changes automatically the Cache data gets flushed out. To use file based dependency we need to monitor the file present on the hard disk by using a class known as "CacheDependency" present under the namespace "System.Web.Caching". We need to create instance of this class by passing FileName as a parameter and then the class instance should be passed as a parameter to insert method, so that whenever the data in file changes immediately it gets flushed out from the Cache Memory. To test this, add a XML File under the project naming it as Products.xml and the following code in it:

```
<Products>
  <Product>
    <Pid>101</Pid>
    <Pname>DVD Player</Pname>
  </Product>
  -Add multiple products data here
</Products>
```

Now add a WebForm under the site naming it as "DataCachingFileBased.aspx", place a GridView control on it and write the below code under its code behind file.

```
using System.Data; using System.Web.Caching;
```

Under Page_Load:

```
DataSet ds;
if (Cache["ProductDS"] == null) {
  ds = new DataSet(); ds.ReadXml(Server.MapPath("~/Products.xml"));
  CacheDependency cd = new CacheDependency(Server.MapPath("~/Products.xml"));
  Cache.Insert("ProductDS", ds, cd); Response.Write("Data loaded from XML File.");
}
else { ds = (DataSet)Cache["ProductDS"]; Response.Write("Data loaded from Cache."); }
GridView1.DataSource = ds.Tables[0]; GridView1.DataBind();
```

Run the above and watch the output and the data in the Cache will be staying in Cache until the file is modified and once the file is modified it will immediately flushes the data from cache.

SQL Based Dependency:

Flushing out data from the cache memory based on the changes made in the database table is known as "SQL Based Dependency" i.e. whenever the data in the table changes automatically the data in the Cache is flushed out. To do this we need to perform 3 extra steps:

1. We need to enable SQL Cache Dependency on the database and table explicitly by using "aspnet_regsql" tool.
2. We need to specify the Cache Details under the config file.
3. We need to create the SQLCacheDependency class instance by specifying the Database Name and Table Name which should be passed as a parameter to the Insert method of Cache.

Step 1: Open Developer Command Prompt and enable Cache Dependency to the Database and Table as following:

Enabling Cache Dependency for Database and Table:

```
aspnet_regsql -S Server -U Sa -P 123 -d ASPDB -ed -t Customer -et
```

Note: We can also disable the enabled Cache Dependency on Database with the following statement:

```
aspnet_regsql -S Server -U Sa -P 123 -d ASPDB -dd
```

Step 2: Open Web.config file and specify the Connection String under <configuration> tag as following:

```
<connectionStrings>
  <add name="ConStr" connectionString="User Id=Sa;Password=123;Database=ASPDB;Data Source=Server"
        providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Now under <system.web> tag specify Cache details as following:

```
<caching>
  <sqlCacheDependency enabled="true">
    <databases><add name="ASPDB" connectionStringName="ConStr" pollTime="2000"  /></databases>
  </sqlCacheDependency>
</caching>
```

Note: pollTime is an attribute to specify when IIS has to contact the Database Engine for any change notifications, default pollTime is 500 milliseconds.

Step 3: Now add a new WebForm under the site naming it as "DataCachingSqlBased.aspx", place a GridView control on it and write the following code under code behind file:

```
using System.Data; using System.Data.SqlClient; using System.Configuration; using System.Web.Caching;
```

Under Page_Load:

```
DataSet ds;
if (Cache["CustomerDS"] == null) {
  string ConStr = ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString;
  SqlDataAdapter da = new SqlDataAdapter("Select * From Customer", ConStr);
  ds = new DataSet(); da.Fill(ds, "Customer");
  SqlCacheDependency cd = new SqlCacheDependency("ASPDB", "Customer");
  Cache.Insert("CustomerDS", ds, cd); Response.Write("Data loaded from Database");
}
else { ds = (DataSet)Cache["CustomerDS"]; Response.Write("Data loaded from Cache"); }
GridView1.DataSource = ds.Tables[0];GridView1.DataBind();
```

Now any changes that are made in database to the table will remove the data in cache and reloads when the next request comes to the page.

Note: we can explicitly remove values from Cache Memory by calling Cache.Remove method by passing the key string as a parameter to the method. E.g: **Cache.Remove(string key)**

Web Services

A web service is a web application which is basically a class consisting of methods called “Web Methods” that could be used by other applications. It means that we can create a web service in any language, such as Java or other languages and that web service can be used in a .Net based application and also a .Net web service in any other application to exchange the information. It follows code-behind architecture such as the ASP.NET web pages, although it does not have a user interface.

Web Method:

Methods in web services attached with [WebMethod] attribute are known as Web Methods which indicates that you want the method exposed as part of the XML Web service. The WebMethod attribute provides the following properties:

1. BufferResponse
2. CacheDuration
3. Description
4. EnableSession
5. MessageName

BufferResponse: This property of WebMethod attribute enables buffering of responses for an XML Web service method. When set to true [default], ASP.NET buffers the entire response before sending it down to the client. The buffering is very efficient and helps improve performance by minimizing communication between the worker process and the IIS process. When set to false, ASP.NET buffers the response in chunks of 16KB. Typically, you would set this property to false only if you did not want the entire contents of the response in memory at once. For example, you are writing back a collection that is streaming its items out of a database. Unless otherwise specified, the default value is true.

CacheDuration: This property of the WebMethod attribute enables caching of the results for an XML Web service method. ASP.NET will cache the results for each unique parameter set. The value of this property specifies how many seconds ASP.NET should cache the results. A value of zero disables the caching of results. Unless otherwise specified, the default value is zero.

Description: This property of the WebMethod attribute supplies a description for an XML Web service method that will appear on the Service help page.

EnableSession: This property of the WebMethod attribute enables session state for an XML Web service method. Once enabled, the XML Web service can access the session state collection directly, provided the WebService class inherits from System.Web.Services.WebService class. Unless otherwise specified, the default value is false.

MessageName: This property of the WebMethod attribute enables the XML Web service to uniquely identify overloaded methods using an alias. Unless otherwise specified, the default value is the method name. When specified the MessageName, the resulting SOAP messages will reflect this name instead of the actual method name.

SOAP and Web Services: SOAP (simple object access protocol) is an XML-based protocol for exchanging information between computers. In short, SOAP is the way by which method calls translate into XML format and sent via HTTP. SOAP is a standard XML based protocol that communicated over HTTP. We can think of SOAP as message format for sending messages between applications using XML. It is independent of technology, language and platform. It doesn't mean that we have to write XML and SOAP specific things ourself to facilitate this communications, ASP.NET will take care of doing the low level SOAP and XML work for us.

WSDL: It stands for Web Service Description Language, is the standard format for describing a web service by which a web service can tell clients what messages it accepts and which results it will return. WSDL contains every detail regarding using web service and its Method provided by web service and URL's from which those methods can be accessed and Data Types used. WSDL definition describes how to access a web service and what operations it will perform. Once we implement an ASP.NET Web Service Visual Studio automatically generates Web Services Description Language (WSDL) file for each Web Service on that application. When you type the URL of a Web service appended by the "?wsdl" parameter in a Web browser, the ASP.NET application returns the WSDL file, which contains the WSDL binding definition for the Web service.

UDDI - A Global Registry of Web Services: UDDI stands for Universal Description, Discovery and Integration is a public registry designed to house information about businesses and their services in a structured way. Through UDDI, one can publish and discover information about a business and its Web Services. Through a set of SOAP-based XML API calls, one can interact with UDDI at both design time and run time to discover technical data, such that those services can be invoked and used. In this way, UDDI serves as infrastructure for a software landscape based on Web Services.

Why UDDI? What is the need for such a registry? As we look toward a software landscape of thousands - perhaps millions of Web Services, a tough challenge emerges: How are Web Services discovered?

Ans: In response to this challenge, the UDDI initiative emerged. A number of companies, including Microsoft, IBM, Sun, Oracle, Compaq, Hewlett Packard, Intel, SAP, and over three hundred other companies came together to develop a specification based on open standards and non-proprietary technologies to solve these challenges. The result, initially launched in beta December 2000 and in production by May 2001, was a global business registry hosted by multiple operator nodes that users could at no cost search and publish web services.

<u>Developing a Web Service:</u>

Open a new Web Application Project naming it as "WebServiceProject" => Choose Empty, but don't select "Web Forms", click Ok and enable it to run under IIS. Now right click on the project in Solution Explorer => Select Add => New Item and choose "Web Service (ASMX) which will add an item with an extension of ".asmx" name it as "FirstService.asmx" and click ok and under the file we find a class "FirstService" inheriting from "System.Web.Services.WebService" and this class uses an attribute known as [WebService] to specify this is a Web Service which will be as following:

[WebService(Namespace = "http://tempuri.org/")]

Namespace is to identify our web services unique after publishing them, so generally we use our company name as the namespace, so let us change this as following:

[WebService(Namespace = "http://bangarraju.net/")]

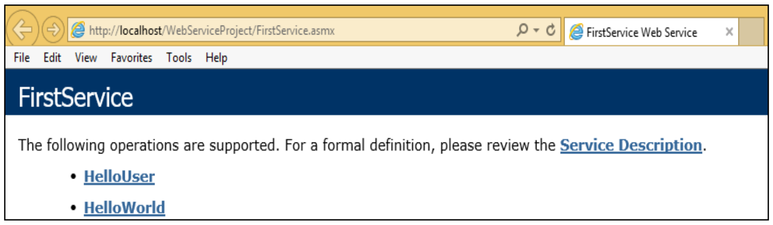
Under the class we find a method "HelloWorld" which uses an attribute [WebMethod] to specify that the method is a Web Method and if this attribute is not present the method can't be accessed by the clients and the method looks as following:

```
[WebMethod]
public string HelloWorld() {
  return "Hello World";
}
```

Under this method let us add another method "HelloUser" and that method also should have the [WebMethod] attribute as following:

```
[WebMethod]
public string HelloUser(string Name) {
  return "Hello " + Name;
}
```

Now if the run the class by hitting F5 it will display as following under the browser:


http://localhost/WebServiceProject/FirstService.asmx
FirstService Web Service
File Edit View Favorites Tools Help

FirstService

The following operations are supported. For a formal definition, please review the Service Description.

- HelloUser
- HelloWorld

In the above HelloUser and HelloWorld are displayed as HyperLinks and when we click on the appropriate method it will display a window with Invoke button and if the method requires any Input it will provide a TextBox for entering the Input values which you can enter and then click Invoke which displays the result in a new window.

<u>Note:</u> if we want to view the "wsdl" code of this service, add "?wsdl" to the url and then hit enter, as following:

http://localhost/WebServiceProject/FirstService.asmx?wsdl

<u>Consuming the Web Service:</u>

To consume the Web Service we have developed open a new Web Application project choosing "WebForms", name it as "WebServiceConsumer" and enable it to run under IIS. Open solution explorer right click on the "References" option under project and select "Add Service Reference" which opens a window and in that enter the following URL: "http://localhost/WebServiceProject/FirstService.asmx" and click go which displays our "FirstService" and below we will find a namespace option with a value "ServiceReference1", either leave the same or change to any meaningful name like "TestFS" and click ok which will generate a proxy class for consuming the Web Methods and that will be under a new folder that is add under project with the name "Service References".

Note: To watch the Proxy class, open Visual Studio => select the project "WebServiceConsumer" and on the top we find an option "Show All Files" select it => expand "Services References" node => expand "TestFS" node => expand "Reference.svcmap" node and double click on "Reference.cs" file which will display the contents of that file and in that we find a class with the name"FirstServiceSoapClient" which is our proxy class using which we can call the Web Methods.

Now add a new WebForm under the project naming it as "WebForm1.aspx" and write the following code under its <div> tag:

```
<asp:Button ID="Button1" runat="server" Text="Call Hello World" />
<asp:Label ID="Label1" runat="server" Text=""></asp:Label><br />
Enter Name: <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
<asp:Button ID="Button2" runat="server" Text="Call Hello User" />
<asp:Label ID="Label2" runat="server" Text=""></asp:Label>
```

Now write the following code under "aspx.cs" file:

Under "Call Hello World" Button:

```
TestFS.FirstServiceSoapClient obj = new TestFS.FirstServiceSoapClient(); Label1.Text = obj.HelloWorld();
```

Under "Call Hello User" Button:

```
TestFS.FirstServiceSoapClient obj = new TestFS.FirstServiceSoapClient();
Label2.Text = obj.HelloUser(txtName.Text);
```

Creating a new Web Service to understand the properties of WebMethod attribute:

Add a new Service under the "WebServiceProject" naming it as "SecondService.asmx" and change the WebService Namespace as "http://bangarraju.net/" and also below the "WebService" attribute we find another attribute "WebServiceBinding" as following: "[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]" which should be change as following: "[WebServiceBinding(ConformsTo = WsiProfiles.None)]" and then write the following code under the "SecondService" class removing the HelloWorld" method:

```
[WebMethod(MessageName = "SayHello1", Description = "This method wishes a  user.")]
public string SayHello() {
  return "Hello Mr./Ms. have a nice day.";
}
[WebMethod(MessageName = "SayHello2", Description = "This method wishes a user with his/her name.")]
public string SayHello(string name) {
  return "Hello Mr./Ms. " + name + " have a nice day.";
}
[WebMethod(CacheDuration = 30, Description = "This method adds 2 integer values and will cache output.")]
public string AddNums(int a, int b) {
  int c = a + b; return "Output generated at: " + DateTime.Now.ToLongTimeString() + " and the value is: " + c;
}
[WebMethod(EnableSession = true, Description = "This method increments a number.")]
public string IncrementValue() {
  int Count = 0;
  if (Session["Counter"] == null) { Count = 1; }
  else { Count = (int)Session["Counter"]; Count += 1; }
  Session["Counter"] = Count; return "Count value is: " + Count;
}
```

Run the class to test all the methods and now to consume those methods add a new WebForm under the "WebServiceConsumer" project naming it as "WebForm2.aspx" and write the following code under its <div> tag:

```
<asp:Button ID="Button1" runat="server" Text="Call SayHello()" Width="200px" /><br />
<asp:Label ID="Label1" runat="server" Text="" /><br />
<asp:Button ID="Button2" runat="server" Text="Call SayHello(string)" Width="200px" />
<asp:TextBox ID="txtName" runat="server" /><br />
<asp:Label ID="Label2" runat="server" Text="" /><br />
<asp:Button ID="Button3" runat="server" Text="Call AddNums(int, int)"  Width="200px" />
<asp:TextBox ID="txtNum1" runat="server" />
<asp:TextBox ID="txtNum2" runat="server" /><br />
<asp:Label ID="Label3" runat="server" Text="" /><br />
<asp:Button ID="Button4" runat="server" Text="Call IncrementValue()"  Width="200px" /><br />
<asp:Label ID="Label4" runat="server" Text="" />
```

Add "WebReference" to the "WebServiceConsumer" Project target the "SecondService" class by the using following URL "http://localhost/WebServiceProject/SecondService.asmx" and name the namespace as "TestSS" which creates a proxy class with the name "SecondServiceSoapClient". Now write the following code under "WebForm2.aspx.cs" file:

Declarations: *TestSS.SecondServiceSoapClient obj;*

Under Page_Load: *obj = new TestSS.SecondServiceSoapClient();*

Under Button1 Click: *Label1.Text = obj.SayHello();*

Under Button2 Click: *Label2.Text = obj.SayHello1(txtName.Text);*

Under Button3 Click: *Label3.Text = obj.AddNums(int.Parse(txtNum1.Text), int.Parse(txtNum2.Text));*

Under Button4 Click: *Label4.Text = obj.IncrementValue();*

Note: in the above case even if the IncrementValue method is enabled with Session still it will not increment the value because the session id is not submitted to the Web Service by our application so to resolve the problem open the Web.config file and there we find <system.serviceModel> tag and under that we find <binding name="SecondServiceSoap" /> and to this add allowCookies = "true" which should be as following:

<binding name="SecondServiceSoap" allowCookies="true" />

Creating a Web Service performing Database Operations:

Add a new Service under the "WebServiceProject" naming it as "DBService.asmx" and change the WebService Namespace as "http://bangarraju.net/" and write the following code under the class:

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

```
[WebMethod(Description = "Returns all the table names in the given Database")]
public DataSet GetTableNames(string DBName) {
  DataSet ds = new DataSet();
  SqlConnection con = new SqlConnection("User Id=Sa;Password=123;Data Source=Server;Database=" + DBName);
  SqlDataAdapter da = new SqlDataAdapter("Select Name From SysObjects Where xtype='U'", con);
  da.Fill(ds, "SysObjects"); return ds;
}
```

```
[WebMethod(BufferResponse = false, Description = "Returns data of given table.")]
public DataSet GetTableData(string TableName, string DBName) {
  DataSet ds = new DataSet();
  SqlConnection con = new SqlConnection("User Id=Sa;Password=123;Data Source=Server;Database=" + DBName);
  SqlDataAdapter da = new SqlDataAdapter("Select * From " + TableName, con);
  da.Fill(ds, TableName);
  return ds;
}
```

Run the class to test all the methods and now to consume those methods add a new WebForm under the "WebServiceConsumer" project naming it as "WebForm3.aspx" and write the following code under its <div> tag:

```
<table align="center">
  <tr>
    <td align="right">DB Name:</td>
    <td>
      <asp:TextBox ID="txtDBName" runat="server" Width="150px" />
      <asp:Button ID="Button1" runat="server" Text="Get Tables" />
    </td>
  </tr>
  <tr>
    <td align="right">Tables:</td>
    <td><asp:DropDownList ID="ddlTables" runat="server" AutoPostBack="True" Width="155px" /></td>
  </tr>
  <tr>
    <td colspan="2"><asp:GridView ID="GridView1" runat="server" /></td>
  </tr>
</table>
```

Add "WebReference" to the "WebServiceConsumer" Project target the "DBService" class by the using following URL "http://localhost/WebServiceProject/DBService.asmx" and name the namespace as "TestDBS" which creates a proxy class with the name "DBServiceSoapClient". Now write the following code under "WebForm3.aspx.cs" file:

```
using System.Data;
```

Declarations: TestDBS.DBServiceSoapClient obj;

Under Page_Load: obj = new TestDBS.DBServiceSoapClient();

Under "Get Tables" Button:

```
DataSet ds = obj.GetTableNames(txtDBName.Text);
ddlTables.DataSource = ds; ddlTables.DataTextField = "Name"; ddlTables.DataValueField = "Name";
ddlTables.DataBind(); ddlTables.Items.Insert(0, "Select Table");
```

Under DropDownList SelectedIndexChanged:

```
if (ddlTables.SelectedIndex > 0) {
  DataSet ds = obj.GetTableData(ddlTables.SelectedValue, txtDBName.Text);
  GridView1.DataSource = ds; GridView1.DataBind();
}
```

http://currencyconverter.kowabunga.net/converter.asmx