Agenda

1. *Databinding traditional way*
2. *SqlDataSource*
3. *GridView*
4. *DetailsView*
5. *FormView*
6. *DataList*
7. *Repeater*
8. *ListView*
9. *DataPager*

Data Bound Controls

Introduction:

These controls bind to a data source and make it easy to display and edit data from database.

Example to fetch data from database and displaying the same:

Drag and Drop two Literal Controls

1. lcTableDataSet
2. lcTableDataReader

In web.config add connection string:

web.config

```
<connectionStrings>
  <add name="DemoDb" connectionString=".\sqlexpress;database=DotNetDb;Integrated Security=True"/>
</connectionStrings>
```

Method 1:

```
protected void Page_Load(object sender, EventArgs e)
{
    string sql = "Select * from Emp";
    string cs = ConfigurationManager.ConnectionStrings["DemoDb"].ConnectionString;
    SqlConnection con = new SqlConnection(cs);
    SqlCommand cmd = new SqlCommand(sql, con);
    Page.Trace.Warn("DataSet Rendering Starts");
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataSet ds = new DataSet();
    da.Fill(ds, "Emp");
    string str = "<table border=1>";
    foreach (DataRow dr in ds.Tables[0].Rows)
    {
        str += "<tr>";
        str += "<td>" + dr["EmpId"] + "</td>";
        str += "<td>" + dr["EmpName"] + "</td>";
        str += "<td>" + dr["EmpSalary"] + "</td>";
        str += "</tr>";
    }
    str += "</table>";
    Page.Trace.Warn("DataSet Rendering Ends");
    lcTableDataSet.Text = str;
}
```

Method 2:

```
protected void Page_Load(object sender, EventArgs e)
{
   string sql = "Select * from Emp";
   string cs = ConfigurationManager.ConnectionStrings["DemoDb"].ConnectionString;
   SqlConnection con = new SqlConnection(cs);
   SqlCommand cmd = new SqlCommand(sql, con);
   Page.Trace.Warn("DataReader Rendering Starts");
   con.Open();
   SqlDataReader dr1 = cmd.ExecuteReader();
   str = "<table border=1>";
   while (dr1.Read())
   {
      str += "<tr>";
      str += "<td>" + dr1["EmpId"] + "</td>";
      str += "<td>" + dr1["EmpName"] + "</td>";
      str += "<td>" + dr1["EmpSalary"] + "</td>";
      str += "</tr>";
   }
   str += "</table>";
   dr1.Close();
   con.Close();
   Page.Trace.Warn("DataReader Rendering Ends");
   lcTableDataReader.Text = str;
}
```

GridView

GridView Properties:

AllowPaging, **AllowSorting**, AlternatingRowStyle, AutoGenerateColumns, AutoGenerateDeleteButton, AutoGenerateEditButton, AutoGenerateSelectButton, BottomPagerRow, **Columns**, **DataKeyNames**, **DataKeys**, **DataMember**, **DataSource**, **DataSourceID**, **EditIndex**, EditRowStyle, EmptyDataRowStyle, EmptyDataTemplate, **EmptyDataText**, FooterRow, FooterStyle, HeaderRow, PageCount, PageIndex, PagerSettings, PagerStyle, PagerTemplate, PageSize, RowHeaderColumn, **Rows**, RowStyle, **SelectedDataKey**, **SelectedIndex**, **SelectedRow**, SelectedRowStyle, **SelectedValue**, ShowFooter, ShowHeader, SortDirection, SortExpression, TopPagerRow

GridView Events:

RowCommand, SelectedIndexChanged, SelectedIndexChanging, Sorting, Sorted, RowUpdating, RowUpdated, RowDeleted, RowDeleting, RowCreated, RowDataBound, DataBinding, DataBound, PageIndexChanging, PageIndexChanged.

Types of Columns/Fields in GridView / DetailsView / FormsView:

1. **DataControlField**: Its parent of all other Fields and is abstract class.

   Properties:

   ControlStyle, FooterStyle, FooterText, HeaderText, ItemStyle, ShowHeader, SortExpression, Visible.

2. **BoundField**: It renders Literal Control within the cell.

   Properties:

   DataField, DataFormatString, ApplyFormatInEditMode, HtmlEncode. InsertVisible. NullDisplayText, ReadOnly

3. **Command Field**: It's used only for the commands - (Select, Delete, Update, Cancel, Edit, New and Insert)

   Properties:

   ShowCancelButton, ShowDeleteButton, ShowEditButton, ShowInsertButton, ShowSelectButton, ButtonType, CancelImageUrl, CancelText, CausesValidation, DeleteText, DeleteImageUrl, EditImageUrl, EditText, HeaderImageUrl, InsertImageUrl, InsertText, InsertVisible, NewImageUrl, NewText, SelectImageUrl, SelectText, UpdateImageUrl, UpdateText, ValidationGroup

4. **CheckboxField:** It renders Checkbox with the cell

   Properties:

   ApplyFormatInEditMode, ControlStyle, ConvertEmptyStringToNull, DataField, DataFormatString, HtmlEncode, InsertVisible, NullDisplayText, ReadOnly, Text

5. **ButtonField**: It's used for Custom commands

   Properties:

   ButtonType, CausesValidation, CommandName, DataTextField, DataTextFormatString, ImageUrl, InsertVisible, Text, ValidationGroup

6. **HyperlinkField**: It is used to Navigate to another page

   Properties:

   DataNavigateUrlFields, DataNavigateUrlFormatString, DataTextField, DataTextFormatString, Target

7. **ImageField:** It is used to display image of the field in the cell.

   Properties:

   AlternateText, ConvertEmptyStringToNull, DataAlternateTextField, DataAlternateTextFormatString, DataImageUrlField, DataImageUrlFormatString, InsertVisible, NullDisplayText, NullImageUrl, ReadOnly.

8. **TemplateField**: It is used if any of the field needs some kind of customization

   Properties:

   ItemTemplate, EditItemTemplate, AlternatingItemTemplate, EditItemTemplate, ConvertEmptyStringToNull

GridView Demo

Example to use GridView:

1. Create a Table **Emp**(EmpId,EmpName,EmpSalary,IsActive)
   a. EmpId (int) - Primary Key and AutoIncrement
   b. EmpName (varchar(50)) - Null Allowed = false
   c. EmpSalary (money) - Null Allowed = true
   d. IsActive (bit) – **Null Allowed = true**

2. Drag and Drop SqlDataSource, right click and goto Configure DataSource and provide valid ConnectionString click on Next, Select TableName and its Columns, Click on Advanced Button and Check Generate Insert, Delete, Update Statements
3. As this sql datasource is going to be used with GridView and it is not used for adding a row we can remove InsertCommand from Sql Datasource.

SqlDataSource

```
<asp:SqlDataSource ID="sdsEmp" runat="server"
ConnectionString="<%$ ConnectionStrings:MSNETConnectionString1 %>"
SelectCommand="SELECT [EmpId], [EmpName], [EmpSalary], [IsActive] FROM [Emp]"
UpdateCommand="UPDATE [Emp] SET [EmpName] = @EmpName, [EmpSalary] = @EmpSalary, [IsActive] = @IsActive
WHERE [EmpId] = @EmpId"
DeleteCommand="DELETE FROM [Emp] WHERE [EmpId] = @EmpId ">
  <DeleteParameters>
    <asp:Parameter Name="EmpId" Type="Int32" />
  </DeleteParameters>
  <UpdateParameters>
    <asp:Parameter Name="EmpName" Type="String" />
    <asp:Parameter Name="EmpSalary" Type="Decimal" />
    <asp:Parameter Name="IsActive" Type="Boolean" />
    <asp:Parameter Name="EmpId" Type="Int32" />
  </UpdateParameters>
</asp:SqlDataSource>
```

4. Drag and Drop GridView (ID="gvEmp")on the Form and set its **DataSourceID** = “sdsEmp”.
5. In Smart Tag Click on Refresh Schema and Check Enable Editing, Enable Selection, Enable Deleting, Enable Paging and Enable Sorting. Also Click on AutoFormat and select any one scheme.

Using GridView

```
<asp:GridView ID="gvEmp" runat="server" AutoGenerateColumns="False" DataKeyNames="EmpId"
AllowSorting="True"  DataSourceID="sdsEmp" EmptyDataText="There are no data records to display."
AllowPaging="True" >
  <Columns>
    <asp:CommandField ShowDeleteButton="True" ShowEditButton="True" ShowSelectButton="True" />
    <asp:BoundField DataField="EmpId" HeaderText="EmpId" ReadOnly="True" SortExpression="EmpId" />
    <asp:BoundField DataField="EmpName" HeaderText="EmpName" SortExpression="EmpName" />
    <asp:BoundField DataField="EmpSalary" HeaderText="EmpSalary" SortExpression="EmpSalary" />
    <asp:CheckBoxField DataField="IsActive" HeaderText="IsActive" SortExpression="IsActive" />
  </Columns>
</asp:GridView>
```

6. Run the WebForm and see the output. Test Sorting, Paging, Editing, Deleting, Selecting features.

Notes: How is data fetched from database and rendered as <table> to the browser

a) SqlDataSource encapsulates the functionality of DataAdapter i.e. using all its properties it constructs a DataAdapter using the same it creates a DataSet with a DataTable.

b) The DefaultView (DataView) of the DataTable is Binded to GridView.

c) The **Columns** in the GridView are created based on "<**Columns**>" sub section in "<asp:GridView>".

d) For every DataRowView in DefaultView of the DataTable, a GridViewRow is added to the **Rows** collection in GridView

7. Handle **SelectedIndexChanged** event of GridView (DoubleClick on design view of GridView).

7.1 'For GridView set the property **DataKeyNames** = "EmpId"

7.2 Response.Write(gvEmp.SelectedValue) //Gives the value of column mentioned in DataKeyNames

7.3 Response.Write(gvEmp.**SelectedDataKey**.**Value**) //Same as above

7.4 'Run and click on Select of any row in GridView

7.5 'Change DataKeyName="EmpName" and Repeat 7.4

To get the content of a given cell in a given row of the grid.

7.6 Dim gvr As **GridViewRow** = gvEmp.**SelectedRow**

7.7 Response.Write(gvr.**Cells**[2].Text) 'Gets the Text of the third cell of a given Row in GridView.

7.8 Dim chk As CheckBox = DirectCast(**gvr.Cells([4].Controls[0]**, CheckBox)

7.9 Response.Write(chk.Checked.ToString())

Note: If the type of column is BoundField, we can use Text property of the cell to get the value in it.
If the type of column is CheckBoxField, we have to get the reference to the checkbox in that cell using Controls collection (as in 7.8) and use checked property of checkbox to find if its checked or not**.**

About CommandField

Its used to add following functionality for every row in the grid.

- **Select**: Sets the "SelectedIndex" property of GridView to the index of current row
- **Edit**: Sets the "EditIndex" property of GridView to the index of current row
- **Cancel**: Sets the "EditIndex" property of GridView to "-1"
- **Update**: Takes the data from the controls of the edited row and assigns it to parameters in the UpdateCommand of the SqlDataSource and executes the UpdateCommand.
- **Delete:** Executes the DeleteCommand of the SqlDataSource.

Note: For a command field we can set ButtonType = "Image/Button/Link".

We can add custom commands to the gridview by adding **ButtonField** to the Columns collection.

Steps for adding Custom Commands to the GridView:

- Add a ButtonField to the GridView. (Step 8)
- Set its CommandName and Text Properties.
- Handle the RowCommand event of GridView

8. Add the following line in between <Columns> tag in GridView

Adding a control to "Columns" in GridView

```
<asp:ButtonField ButtonType="Button" CommandName="Msg" HeaderText="Custom" Text="Send Msg" />
```

9. Handle **RowCommand** event of GridView.

RowCommand event handler

```
protected void gvEmp_RowCommand(object sender, GridViewCommandEventArgs e)
{
   if ((e.CommandName == "Msg"))
   {
      int indRow = Convert.ToInt32(e.CommandArgument);
      //e.CommandArgument gives the index of the row.
      GridViewRow gvr = gvEmp.Rows[indRow];
      Response.Write(gvr.Cells[1].Text);
   }
}
```

Note: The RowCommand Event of GridView is raised for all commands including the predefined and custom commands

10. Run the application

Using **HyperlinkField** we can **Navigate** to a new page, whereas using Button field we do **Postback** to the same page.

11. Adding **HyperlinkField** to the Grid.

HyperLink control

```
<asp:HyperLinkField DataTextField="EmpName" DataNavigateUrlFields="EmpId, EmpName"
DataNavigateUrlFormatString="EmpDetails.aspx?EmpId={0}&EmpName={1}" Target="_blank" />
```

Note: Hyperlink field we can set either Text (="Details") or DataTextField (="EmpName") property. If Text is set then for every row same text is rendered but if DataTextField is set then the specified field/column value is rendered for every row.

12. Run the application
13. **Image Field:** It is used to display the image for every row in the specified column.
14. Add to the project a folder by name "EmpImages" and based on EmpId add jpg files to the folder for ex: 1.jpg, 2.jpg etc...
15. Add the following to the <Columns> section of GridView.

Adding to "Columns" section of GridView

```
<asp:ImageField DataImageUrlField="EmpId" DataImageUrlFormatString="EmpImages\{0}.jpg"
   DataAlternateTextField="EmpName" DataAlternateTextFormatString="Name: {0}"  ReadOnly = "true"
ControlStyle-Width="50" ControlStyle-Height="50" />
```

Note: Based on DataImageUrlFormatString, Src attribute of the <Img> is rendered.

16. Run the application.

DetailsView Demos

Used for showing one row at a time

Requirement: For the selected row in GridView, all the fields of Emp table should be shown in DetailsView

17. Add another SqlDataSource to the WebForm. This time set a where condition. In "Add Where Clause" dialog set Column as EmpId, Operator as =, Source as Control, Control Id as gvEmp and and Click on Add Button

Adding another SqlDataSource

```
<asp:SqlDataSource ID="dsEmpFiltered" runat="server" ConnectionString="<%$ ConnectionStrings:csMSNET %>"
InsertCommand="INSERT INTO [Emp] ([EmpId], [EmpName], [EmpSalary]) VALUES (@EmpId,@EmpName,
@EmpSalary)"
SelectCommand="SELECT [EmpId], [EmpName], [EmpSalary] FROM [Emp] WHERE ([EmpId] = @EmpId)"
UpdateCommand="UPDATE [Emp] SET [EmpName] = @EmpName, [EmpSalary] = @EmpSalary where
[EmpId]="@EmpId">
        <SelectParameters>
          <asp:ControlParameter ControlID="gvEmp" DefaultValue="-1"
Name="EmpId" PropertyName="SelectedValue" Type="Int32" />
        </SelectParameters>
        <UpdateParameters>
          <asp:Parameter Name="EmpName" Type="String" />
          <asp:Parameter Name="EmpSalary" Type="Decimal" />
          <asp:Parameter Name="EmpId" Type="Int32" />
        </UpdateParameters>
        <InsertParameters>
          <asp:Parameter Name="EmpId" Type="Int32" />
          <asp:Parameter Name="EmpName" Type="String" />
          <asp:Parameter Name="EmpSalary" Type="Decimal" />
        </InsertParameters>
 </asp:SqlDataSource>
```

Note: DeleteCommand has been removed from the above SqlDataSource because this datasource will be used with DetailsView and in details view we don't want to provide delete functionality

18. Add DetailsView to the WebForm and set its DataSourceId = "dsEmpFiltered".
    Also Enable Inserting and Editing for DetailsView (Smart Tag)

Adding DetailsView

```
<asp:DetailsView ID="dvEmp" runat="server" AutoGenerateRows="False" CellPadding="4"
   DataKeyNames="EmpId" DataSourceID="dsEmpFiltered" GridLines="None">
   <Fields>
```

```
    <asp:BoundField DataField="EmpId" HeaderText="EmpId" ReadOnly="True" SortExpression="EmpId" />
    <asp:BoundField DataField="EmpName" HeaderText="EmpName" SortExpression="EmpName" />
    <asp:BoundField DataField="EmpSalary" HeaderText="EmpSalary" SortExpression="EmpSalary" />
    <asp:CommandField ShowEditButton="True" ShowInsertButton="True" />
  </Fields>
</asp:DetailsView>
```

19. Run the webform, Click on Select command in the grid, The selected row of is shown in the DetailsView.
    Requirement: If DetailsView is either in Insert or Edit mode, we should not able to perform any operations on GridView and hence it should be disabled
20. Handle ModeChanged event of DetailsView

ModeChanged event of DetailsView

```
protected void dvEmp_ModeChanged(Object sender, EventArgs e)
{
   if(dvEmp.CurrentMode == DetailsViewMode.ReadOnly)
      gvEmp.Enabled = true;
   else
      gvEmp.Enabled = false;
}
```

Requirement:

Any changes (Edit / New) made to the Database using DetailsView are by default not reflected in GridView because the GridView is not refreshed.

The events **ItemInserted** and **ItemUpdated** can be used for writing code to be executed after the Sql Statements are executed in the backend. Here we can also write the code for handling exceptions if thrown during execution of statements.

21. Handle ItemInserted and ItemUpdated event of DetailsView

Handling ItemInserted and ItemUpdated events

```
protected void dvEmp_ItemInserted(object sender, DetailsViewInsertedEventArgs e)
{
   if(e.Exception != null)
   {
      Page.ClientScript.RegisterStartupScript(this.GetType(),"k1","alert(" + e.Exception.Message + ")",true);
      e.ExceptionHandled = true;
      e.KeepInInsertMode = true;
   }
   else
   {
      gvEmp.DataBind();
      gvEmp.SelectedIndex = gvEmp.Rows.Count - 1;
```

```
    }
}

protected void dvEmp_ItemUpdated(object sender, DetailsViewUpdatedEventArgs e)
{
    if(e.Exception != null)
    {
        Page.ClientScript.RegisterStartupScript(this.GetType(),"k1","alert(" + e.Exception.Message + ")",true);
        e.ExceptionHandled = true;
        e.KeepInEditMode = true;
    }
    else
    {
        gvEmp.DataBind();
        gvEmp.SelectedIndex = gvEmp.Rows.Count - 1;
    }
}
```

22. Run the Web form, Edit and Add a row in DetailsView and note that changes are also reflected in GridView.
23. Edit the record in DetailsView → For EmpSalary give a non numeric value (ABCD) → Update the row → Exception is thrown and is handled in ItemUpdated / ItemInserted event handler.

Template Field:

When ever any kind of customization is needed to a column or field, convert it into a TemplateField

TemplateField has following Templates:

a. **ItemTemplate:** The content in it is rendered for all rows which are not in edit mode
b. **EditItemTemplate:** The content in it is rendered for row which is in edit mode
c. **HeaderTemplate:** The content in it is rendered for Header of the column
d. **FooterTemplate:** The content in it is rendered for footer of the column

Bind (Two-Way Binding):

When the GridViewRow is rendered, the value of the mentioned column for a given row is assigned to the specified property of control. Also when the GridViewRow changes have to updated in database, the value of control property is assigned to the parameter of the command before it is executed.

For Example: Text='<%#Bind("EmpName") %>' - The value of EmpName Column of the current row is assigned to Text property of Control and when the row is updated the Text Property of control is assinged to the parameter @EmpName before the Update Command is executed.

Eval (One way Binding):

It fetches the data from column of a given row and is assigns to control property but change made to the property is not assigned to the parameter and hence “null” will be used for the parameter when the command is executed.

Requirement: In edit mode EmpName TextBox should not allow empty string and thus RequiredFieldValidator must be associated to it.

24. Convert EmpName (BoundField) to TemplateField and add RequiredFieldValidator to EditItemTemplate as below.

a) Smart Tag → Edit Columns → Select the Column (EmpName) → Click on the Hyperlink (Convert this field into a TemplateField)

b) Smart Tag → Edit Templates → Now in Smart Tag set Display = "EmpName – EditItemTemplate" → Drag and Drop RequiredFieldValidator against TextBox

c) For TextBox Set ID="txtEmpName"

d) For RequiredFieldValidator set ControlToValidate="txtEmpName", also set ErrorMessage and Text as required

Template Field

```
<asp:templatefield showheader="true" headertext="EmpName">
   <ItemTemplate>
      <asp:Label ID="lblEmpName" runat="server" Text='<%#Eval("EmpName") %>'/>
   </ItemTemplate>
   <EditItemTemplate>
      <asp:TextBox ID="txtEmpName" runat="server" Text='<%#Bind("EmpName") %>' />
      <asp:RequiredFieldValidator runat="server" ID="rfvEmpName" ControlToValidate="txtEmpName" Text="*"
ErrorMessage="Please provide Employee Name" />
    </EditItemTemplate>
</asp:templatefield>
```

Note: **To Disable UnObstrusive Validation** add the following to web.config

```
<appSettings>
        <add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />
</appSettings>
```

25. In GridView add a new TemplateColumn for Delete. Now delete button will prompt the user before the row is deleted

Appending TemplateColumn

```
<asp:TemplateField>
  <ItemTemplate>
    <asp:Button CommandName="delete"  runat="server" ID="btnDelete" Text="Delete" OnClientClick="return
confirm('Are you sure')" />
  </ItemTemplate>
</asp:TemplateField>
```

26. GOTO Example “EmpGridViewDemo with Department”

Requirement: We want to show “Yes” or “No” in the IsActive Column

27. Directly insert NULL value for “IsActive” Column in Database table “Emp” (using Server Explorer)
28. In GridView, convert IsActive field to Templetefield and in ItemTemplate replace Checkbox with Label as below
29. In ItemTemplate edit the label control by changing ID and Text attribute as below.

Editing the Label control

```
<asp:Label ID="lblIsActive" runat="server" Text='<%# GetYesOrNo(Eval("IsActive")) %>'/>
```

30. In .cs file add the following method

Method to check for null values

```
protected string GetYesOrNo(object value)
{
   if ((DBNull.Value == value)) // To Handle NULL value for IsActive
      return "Unknown";
   return ( Convert.ToBoolean(value) ? "Yes" : "No" );
}
```

31. Run the application and note the values of IsActive column.
32. Edit the row which doesn't have NULL for IsActive, on update it does so successfully
33. Edit the row which has NULL for IsActive, on update it throws exception.
34. In IsActive, EditItemTemplate change the ID and Checked Property of checkbox as below

Editing the Checkbox control

```
<asp:CheckBox ID="chkIsActive" runat="server" Checked='<%# IsEmpActive(Eval("IsActive")) %>' />
```

35. Add the following function to .cs file

Adding a method to check for null

```
protected bool IsEmpActive(object active)
{
   if ((DBNull.Value == active))
      return false;
   return Convert.ToBoolean(active);
}
```

36. Run the Webform, edit the row which has NULL for "IsActive" and note that this time exception is not thrown.
37. But when the row is updated, irrespective of the CheckBox state NULL will be saved for IsActive because Two-way binding is not done.

Requirement: We have to now programatically get the CheckBox state and assign to IsActive parameter before the UpdateCommand of SqlDataSource is executed.

38. Handle RowUpdating event of GridView (as below)

Handling RowUpdating events

```
protected void gvEmp_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
   CheckBox chkIsActive = (CheckBox)(gvEmp.Rows[e.RowIndex].FindControl("chkIsActive"));
   e.NewValues("IsActive") = chkIsActive.Checked;
```

```
}
```

Note: e.NewValue collection should be used for setting values of those fields which are not updated using the DetailsView and the value has to be set programatically.

e.NewValues("IsActive") – Reference to the "IsActive" parameter in UpdateCommand of SqlDataSource

39. Run the webform, edit the row and on update this time its saved correctly.

Requirement:

40. RowDataBound event of GridView is raised for every GridViewRow when it is binded to DataRowView

Steps for Calculating TotalSalary of all employees:

e. In GridView Convert EmpSalary to Template Column

f. For GridView set **ShowFooter** = True

g. Select the Column (EmpSalary) → RightClick → EditTemplate → EmpSalary

h. In the **FooterTemplate** place a label with ID="**lblTotalSalary"**

i. Handle **RowDataBound** event of GridView

Handling RowDataBound event

```
Decimal totalSalary;
protected void gvEmp_RowDataBound(object sender, GridViewRowEventArgs e)
{
   if ((e.Row.RowType == DataControlRowType.DataRow))
   {
      DataRowView drv;
      drv = ((DataRowView)(e.Row.DataItem));
      totalSalary = (totalSalary + Decimal.Parse(drv["EmpSalary"]));
      if ((Convert.ToDecimal(drv["EmpSalary"]) > 10000))
      {
         e.Row.ForeColor = System.Drawing.Color.Red;
      }
      DataControlRowType.Footer;
      Label lbl = ((Label)(e.Row.FindControl("lblTotalSalary ")));
      lbl.Text = totalSalary.ToString();
   }
}
```

Note: The above code also changes the ForeColor of Employees whose EmpSalary is greater than 10000

FormsView

FormView

```
<asp:FormView ID="FormView1" runat="server" DataKeyNames="EmpId" DataSourceID="EmpDataSourceFiltered">
  <ItemTemplate>
    EmpId: <asp:Label ID="EmpIdLabel" runat="server" Text='<%# Eval("EmpId") %>'></asp:Label>
    EmpName: <asp:Label ID="EmpNameLabel" runat="server" Text='<%# Eval("EmpName") %>'></asp:Label>
    EmpSalary:<asp:Label ID="Label1" runat="server" Text='<%# Eval("EmpSalary") %>'></asp:Label><br />
    <asp:LinkButton ID="EditButton" runat="server" CausesValidation="False" CommandName="Edit" Text="Edit" />
    <asp:LinkButton ID="NewButton" runat="server" CausesValidation="False" CommandName="New" Text="New"
/>
  </ItemTemplate>
  <EditItemTemplate>
    EmpId: <asp:Label ID="EmpIdLabel1" runat="server" Text='<%# Eval("EmpId") %>'></asp:Label><br />
    EmpName: <asp:TextBox ID="EmpNameTextBox" runat="server" Text='<%# Bind("EmpName") %>' />
    EmpSalary: <asp:TextBox ID="EmpSalaryTextBox" runat="server" Text='<%# Bind("EmpSalary") %>' />
    <asp:LinkButton ID="UpdateButton" runat="server" CausesValidation="True" CommandName="Update"
Text="Update" />
    <asp:LinkButton ID="UpdateCancelButton" runat="server" CausesValidation="False" CommandName="Cancel"
Text="Cancel" />
  </EditItemTemplate>
  <InsertItemTemplate>
    EmpId: <asp:TextBox ID="EmpIdTextBox" runat="server" Text='<%# Bind("EmpId") %>' />
    EmpName: <asp:TextBox ID="EmpNameTextBox" runat="server" Text='<%# Bind("EmpName") %>' />
    EmpSalary: <asp:TextBox ID="EmpSalaryTextBox" runat="server" Text='<%# Bind("EmpSalary") %>' />
    <asp:LinkButton ID="InsertButton" runat="server" CausesValidation="True" CommandName="Insert"
Text="Insert" />
    <asp:LinkButton ID="InsertCancelButton" runat="server" CausesValidation="False" CommandName="Cancel"
Text="Cancel" />
  </InsertItemTemplate>
</asp:FormView>
```

EmpGridViewDemo with Department

1. Create **Department** and **Employee** tables in the database. Employee table should have DeptId as one of the fields
2. Drag and drop SqlDataSource (ID=sdsDept), bind it to “Department” database table.
3. Drag and drop SqlDataSource (ID=sdsEmp), bind it to “Employee” table. Also for this sqldatasource, generate “Insert, Update and Delete command. (Advanced button in wizard)
4. Drag and drop GridView, Bind it to sdsEmp → Refresh the schema.
5. Run the webform, note that for every row DeptId is rendered.

Requirement: In place of “DeptId”, we want to render “DeptName” for every Employee row.

6. Convert DeptId to Template Field (use Edit Columns dialog).
7. In ItemTemplate bind the Text property of Label to return value of the function “GetDeptName”. Pass to the function the current row DeptId **Eval("DeptId") – Change the existing and write as below**

   <asp:Label . . . Text='<%# **GetDeptName**(Eval("DeptId")) %>'/>
8. In .cs file add the code as below.

Programatically using DataView

```
DataView dv;
protected void Page_Load(object sender, EventArgs e)
{
   DataSourceSelectArguments args = new DataSourceSelectArguments();
   dv = (DataView)sdsDept.Select(args);
}
```

9. Run the webform and note that this in place of DeptId, DeptName is rendered.
10. To the Grid, add CommandField with Edit functionality.

    <asp:CommandField ShowEditButton="True" />
11. Step10: Run the webform, Note that in Edit mode for Department column, TextBox is rendered.

Requirement: Edit a row and for DeptId column, we should provide DropDownList will all Departments.

12. GridView → SmartTag → Edit Templates → Display = EditItemTemplate of DeptId Column
13. Delete TextBox and drag and drop DropDownList

Requirement: Binding DropDownList DataSource to Department table.

14. DropDownList → Smart Tag → Choose DataSource…→ DataSource="sdsDept", DisplayField="DeptName", ValueField="DeptId". → Click on OK

Requirement: To bind "SelectedValue" property of DropDownList to current employee row "DeptId"

15. DropDownList → Smart Tag → Edit DataBindings → Bindable properties="SelectedValue", BoundTo="DeptId" → OK

Following is added to "EditItemTemplate" of DeptId Column.

Binding DropDownList

```
<asp:DropDownList ID="DropDownList1" runat="server" DataSourceID="sdsDept"  DataTextField="DeptName"
DataValueField="DeptId" SelectedValue='<%# Bind("DeptId") %>'>
```

16. Run the webform, Edit the row and note the DropDownList in DeptId column, Change and value and update.

EmpDataList Demo (DataList)

Handling events of DataList

```
protected void dlEmp_EditCommand(Object sender, DataListCommandEventArgs e)
{
   dlEmp.EditItemIndex = e.Item.ItemIndex;
   dlEmp.DataBind();
}

 protected void dlEmp_UpdateCommand(Object sender, DataListCommandEventA
{
   int empid;
   string newempname;
   string newempsalary;
   bool newisactive;
   empid = int.Parse(dlEmp.DataKeys(e.Item.ItemIndex));
   TextBox txtName;
   txtName = ((TextBox)(e.Item.FindControl("txtEditEmpName")));
   newempname = txtName.Text;
   TextBox txtSalary;
   txtSalary = ((TextBox)(e.Item.FindControl("txtEditEmpSalary")));
   newempsalary = txtSalary.Text;
   CheckBox chkIsActive;
   chkIsActive = ((CheckBox)(e.Item.FindControl("chkEditIsActive")));
   newisactive = chkIsActive.Checked;
   Response.Write((empid + (" "
               + (newempname + (" "
               + (newempsalary + (" " + newisactive)))))));
   sdsEmp.UpdateParameters["EmpId"].DefaultValue = id.ToString();
   sdsEmp.UpdateParameters["EmpName"].DefaultValue = txtName.Text;
   sdsEmp.UpdateParameters["EmpSalary"].DefaultValue = txtSalary.Text;
   sdsEmp.Update();
   dlEmp.EditItemIndex = -1;
   dlEmp.DataBind();
}

protected void dlEmp_DeleteCommand(Object sender, DataListCommandEventArgs e)
{
   int empid;
   empid = int.Parse(dlEmp.DataKeys(e.Item.ItemIndex));
```

```
    sdsEmp.UpdateParameters["EmpId"].DefaultValue = id.ToString();
    sdsEmp.Delete();
    dlEmp.DataBind();
}
```

SqlDataSource

```
<asp:SqlDataSource ID="sdsEmp" runat="server" ConnectionString="<%$ ConnectionStrings:csMSNET %>"
  SelectCommand="SELECT [EmpId], [EmpName], [EmpSalary], [IsActive] FROM [Emp]" />
<asp:DataList ID="dlEmp" runat="server" DataKeyField="EmpId" DataSourceID="sdsEmp"
  RepeatColumns="3" RepeatDirection="Horizontal">
  <ItemTemplate>
    EmpId:
    <asp:Label ID="EmpIdLabel" runat="server" Text='<%# Eval("EmpId") %>' /><br />
    EmpName:
    <asp:Label ID="EmpNameLabel" runat="server" Text='<%# Eval("EmpName") %>/'>EmpSalary:
      <asp:Label ID="EmpSalaryLabel" runat="server" Text='<%# Eval("EmpSalary") %>' />
      IsActive:
      <asp:Label ID="IsActiveLabel" runat="server" Text='<%# Eval("IsActive") %>' />
      <asp:LinkButton ID="lnkEdit" CommandName="edit" runat="server" Text="Edit" />
      <asp:LinkButton ID="lnkDelete" CommandName="delete" runat="server" Text="Delete" />
      <asp:LinkButton ID="lnkSelect" CommandName="select" runat="server" Text="Select" />
  </ItemTemplate>
  <EditItemTemplate>
    EmpId:
    <asp:Label ID="EmpIdEditLabel" runat="server" Text='<%# Eval("EmpId") %>'></asp:Label><br />
    EmpName:
    <asp:TextBox ID="txtEditEmpName" runat="server" Text='<%# Eval("EmpName") %>' />
    EmpSalary
    <asp:TextBox ID="txtEditEmpSalary" runat="server" Text='<%# Eval("EmpSalary") %>' />
    IsActive:
    <asp:CheckBox ID="chkEditIsActive" runat="server" Checked='<%# Eval("IsActive") %>' />
    <asp:LinkButton ID="lnkUpdate" CommandName="Update" runat="server" Text="Update" />
    <asp:LinkButton ID="lnkCancel" CommandName="Cancel" runat="server" Text="Cancel" />
  </EditItemTemplate>
</asp:DataList>
```

Other Important Events of DataList: ItemCommand, ItemDataBound and ItemCreated