Web.Config

```
<connectionStrings>
    <add name="ConStr" connectionString="Data Source=(LocalDB)\MSSQLLocalDB;Atta
</connectionStrings>
```

name: This is a unique name given to the connection string. Here, it's "myConnectionString." We will use this name in your code to reference this connection string.

connectionString: This attribute contains the actual connection information. Let's break down the components:

Data Source=(LocalDB)\MSSQLLocalDB: Specifies the data source, which is the LocalDB instance named "MSSQLLocalDB."
AttachDbFilename=D:\College\CSharp\demo_college_2023\App_Data\db Demo_A.mdf: Specifies the path to the database file (dbDemo_A.mdf).

The AttachDbFilename attribute is used when you want to attach a database file directly. Integrated Security=True: Indicates that Windows authentication (integrated security) should be used for the connection.

```
string conStr =
ConfigurationManager.ConnectionStrings["myConnectionString"].Connecti
onString;
```

The line of code you provided retrieves the connection string from the configuration file using the ConfigurationManager class.

ConfigurationManager.ConnectionStrings: This property provides access to the collection of connection strings defined in the application's configuration file (app.config or web.config).

["myConnectionString"]: This is an index into the collection, retrieving the connection string with the specified name, in this case, "myConnectionString."

.ConnectionString: Once you have the connection string object, the ConnectionString property is used to get the actual connection string as a string value.

So, the overall purpose of this line is to retrieve the connection string named "myConnectionString" from the configuration file and store it in the variable conStr for later use in creating a SqlConnection object.

Get Connection String from Web.Config:

- It retrieves the connection string named "myConnectionString" from the Web.Config **file using** ConfigurationManager.ConnectionStrings["myConnectionString"].ConnectionString**.**

Create SqlConnection:

- **It creates a new instance of** SqlConnection **(**conn**) using the retrieved connection string.**

Open Connection if Closed:

- **It checks if the connection state is not open (**if (conn.State != ConnectionState.Open)**).**
- **If the connection is not open, it opens the connection using** conn.Open()**.**

Exception Handling:

- **It includes a try-catch block to handle any exceptions that may occur during the database connection process.**

- **If an exception occurs, it writes the exception details to the response using** Response.Write(ex.ToString())

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

```
0 references
public void fnConnectDB()
{

    //1. get connection from Web.Config into StrCon
    //2. Pass StrCon into SQLCOnnection to connect to  DB
    //3. Open the connection STate if closed.
    //============================================================
    try
    {
        string conStr = ConfigurationManager.ConnectionStrings["myConnectionString"].ConnectionString;
        conn = new SqlConnection(conStr);
        if (conn.State != ConnectionState.Open)
        {
            conn.Open();
        }
    }
    catch (Exception ex)
    {
        Response.Write(ex.ToString());
    }
}
```

Call function on **Page_load**

Here's a brief explanation of each namespace:

System.Configuration: This namespace provides classes for accessing configuration settings, including reading settings from configuration files. In your code, you use it to retrieve the connection string from the configuration file.

System.Data: This namespace provides the base classes for ADO.NET. It includes classes such as DataTable, DataRow, and other core classes used in working with data.

System.Data.SqlClient: This namespace specifically provides classes for working with Microsoft SQL Server databases using ADO.NET. The SqlConnection class, which you use in your code, is part of this namespace.

When working with databases in C# using ADO.NET, these namespaces are commonly used to manage database connections, execute SQL commands, and work with the resulting data.

Gridview

```
//Bind data to gridview
//============================================================
1 reference
public void fnBindGrid()
{
    try
    {
        fnConnectDB();
        string qry = "SELECT * FROM tbl_device";
        cmd = new SqlCommand(qry, conn);
        sda = new SqlDataAdapter(cmd);
        DataSet ds = new DataSet();
        sda.Fill(ds);//Fill Dataset
        dgv_Users.DataSource = ds;  //Bind to Grid
        dgv_Users.DataBind();
        conn.Close();
    }
    catch (Exception ex)
    {
        Response.Write(ex.ToString());
    }
}
```

DropDown

```
1 reference
public void fnBindBrand()
{
    DataSet ds = new DataSet();
   try
    {
        fnConnectDB();
        //  string qry = "SELECT * FROM\r\ntbl_brand\r\nWHERE\r\ntype='" + ddlType.SelectedValue +"'
        //   string qry = "SELECT * FROM\r\ntbl_brand\r\nWHERE\r\ntype=@b_type";
        string qry = "SELECT FROM tbl_brand WHERE type=@b_type";
        cmd = new SqlCommand(qry);
        cmd.Connection = conn;
        cmd.Parameters.AddWithValue("b_type",ddlType.SelectedValue);
        sda = new SqlDataAdapter(cmd);
        sda.Fill(ds);
        ddlBrand.DataSource = ds;
        ddlBrand.DataTextField = "brand";
        ddlBrand.DataValueField = "b_id";
        ddlBrand.DataBind();
        ddlBrand.Items.Insert(0, new ListItem("--select--"));
        conn.Close();
    }
    catch (Exception ex)
    {

        throw ex;
    }
```

Insert

```
    try
    {
        //id is autoincremented hence not included in the query.
        fnConnectDB();
        string qry = "INSERT INTO tblUser(name,city_id,gender,s_media,blood_grp,email )VALUES(@name,@city_id,@gen, @sm,@bg,@e
        cmd = new SqlCommand(qry,conn);
        cmd.Parameters.AddWithValue("@name", txtname.Text);  //textbox
        cmd.Parameters.AddWithValue("@city_id",ddlCity.SelectedValue); //dropdown
        cmd.Parameters.AddWithValue("@gen", rbl_gender.SelectedValue); //radiobutton
        cmd.Parameters.AddWithValue("@sm", social_media);//checkbox
        cmd.Parameters.AddWithValue("@bg", ddlBlood.SelectedValue);
        cmd.Parameters.AddWithValue("@eml", txtemail.Text);
        int res = cmd.ExecuteNonQuery();
        if (res > 0)
            Response.Write("Data Inserted!");
        else
            Response.Write("Insert operation Fail");
        conn.Close();
        fnBindGrid();

    }
    catch (Exception ex)
```

SqlCommand (cmd):

- cmd **is an instance of the** SqlCommand **class in ADO.NET. It represents a SQL statement or stored procedure that is executed against a SQL Server database.**

Parameters Collection:

- cmd.Parameters **is a collection that stores the parameters associated with the SQL command. Parameters are used to pass values into the SQL query in a secure and parameterized manner.**

AddWithValue Method:

- **The** AddWithValue **method is a convenient way to add parameters to the** Parameters **collection.**
- **It takes two arguments: the parameter name and the value.**

Parameter Name (@name):

- @name **is the parameter name used in the SQL command. It is a placeholder that will be replaced by the actual value during the execution of the SQL query.**

Parameter Value (txtname.Text):

- txtname.Text **is an example of the value that will be associated with the parameter. This value typically comes from a user interface element, in this case, a textbox named** txtname

Execute Nonquery

Page 291:

https://theswissbay.ch/pdf/Gentoomen%20Library/Programming/CSharp/Pro%20ASP.Net%20in%20C%23%202010.pdf

In ADO.NET, ExecuteNonQuery **is a method provided by the** SqlCommand **class that is used to execute SQL commands that don't return any result sets. This method is typically employed for statements that modify data in the database, such as INSERT, UPDATE, DELETE, and some Data Definition Language (DDL) statements.**

Here's a brief explanation of ExecuteNonQuery**:**

- **ExecuteNonQuery Method:**
  - **Definition:** int ExecuteNonQuery();
  - Purpose: It is used to execute SQL commands that do not return any data (e.g., rows).
  - Return Type: The method returns an integer representing the number of rows affected by the SQL command.
  - Common Usage: Typically used with INSERT, UPDATE, DELETE, and certain DDL statements.

Update Fetch values from the Gridview in the controls. Make sure, the variable for id is declared globally in a public partial class.

```
2 references
public partial class _2_CRUD_registration : System.Web.UI.Page
{

    //(LocalDB)\MSSQLLocalDB
    //SQL command, connection and data adapter objects must be declared globally as shown here:
    SqlDataAdapter sda;
    SqlCommand cmd;  //Command object
    SqlConnection conn;  //connection object
    public static int up_id; //For update
0 references
    protected void Page_Load(object sender, EventArgs e)
    {
```

Write the code on the Edit button and click the event

```
    //Fetching values in the control from GridView
0 references
    protected void dtgUser_SelectedIndexChanged(object sender, EventArgs e)
    {

        GridViewRow rw = dtgUser.SelectedRow; //Selected row
        txtname.Text = rw.Cells[2].Text;
        txtemail.Text = rw.Cells[7].Text;
        up_id= Convert.ToInt32(rw.Cells[1].Text);

        for (int i = 0; i < ddlCity.Items.Count; i++) //dropdown
        {
            if (ddlCity.Items[i].Text == rw.Cells[3].Text)
            {
                ddlCity.SelectedIndex = i;
            }
        }
        ////radiobutton
        for (int i = 0; i < rbl_gender.Items.Count; i++)
        {
            if (rbl_gender.Items[i].Text == rw.Cells[4].Text.Trim())
            {
                rbl_gender.Items[i].Selected = true;
                break; // You want to select only the first match
            }
        }
    }
```

global variable

Delete

Write the code on Gridview RowDeleting event

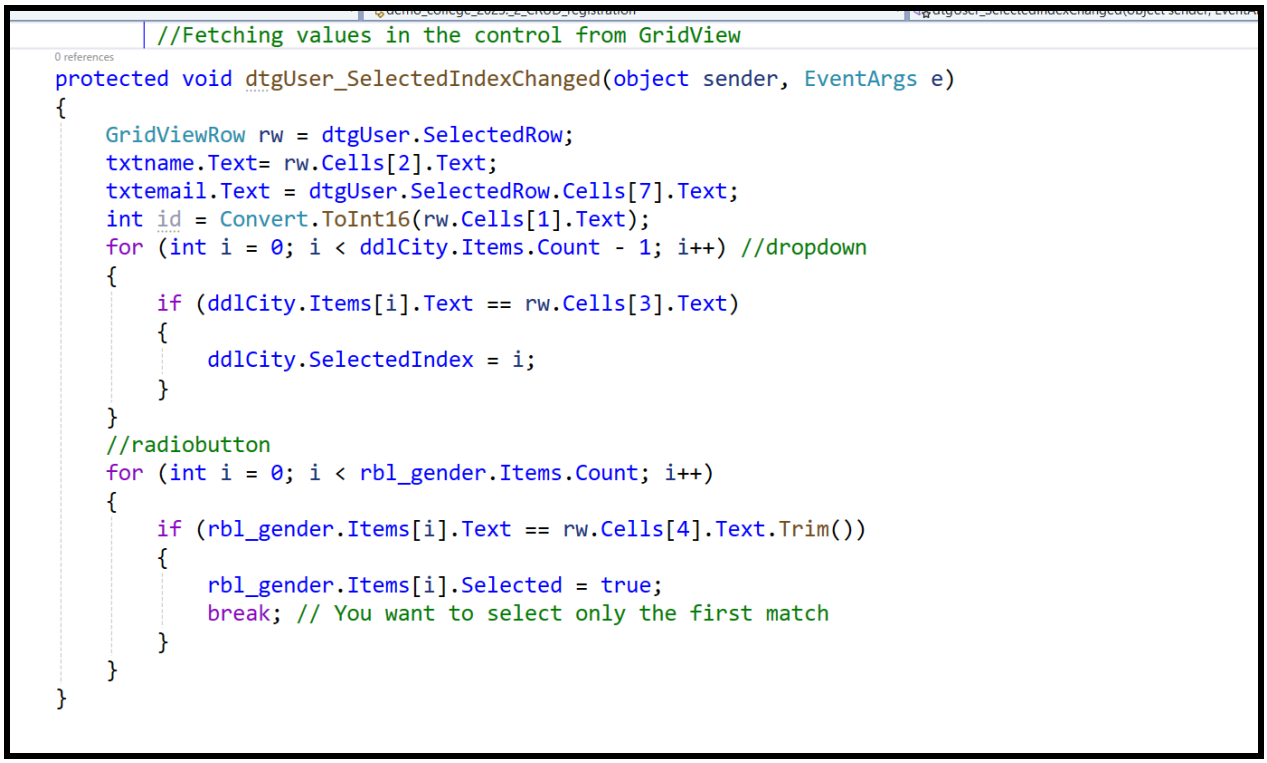
```
0 references
protected void dtgUser_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    //e.RowIndex is a property commonly used in ASP.NET
    //GridView events to retrieve the index of the row involved
    //in the event. .
    GridViewRow rw = dtgUser.Rows[e.RowIndex];
    int del_id = Convert.ToInt32(rw.Cells[1].Text);
    try
    {
        fnConnectDB();
        string qry = "DELETE FROM tblUser WHERE Id=@d_id";
        cmd = new SqlCommand(qry, conn);
        cmd.Parameters.AddWithValue("d_id", del_id);
        int res = cmd.ExecuteNonQuery();
        if (res > 0)
            Response.Write("Data removed!");
        else
            Response.Write("delete failed");

        conn.Close();
        fnBindGrid();
    }
    catch (Exception ex)
    {
        Response.Write("<script>alert('Delete operation failed! " + ex.ToString() + "');</script>");
    }
```

Accessing a GridView row and storing the cell values in respective form input control on GridViewRow selection

GridViewRow rw = dtgUser.SelectedRow;

This line is used when you want to access the currently selected row in the GridView. It assumes that the GridView (*dtgUser *in this case) has the property



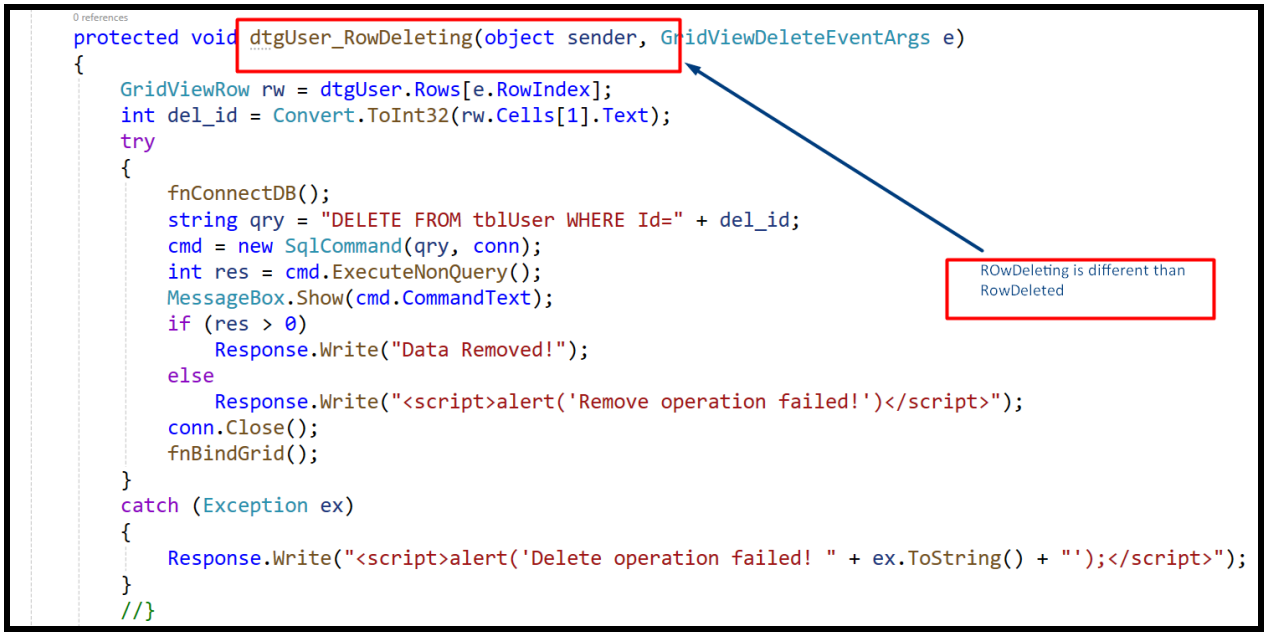
```
        //Fetching values in the control from GridView
0 references
protected void dtgUser_SelectedIndexChanged(object sender, EventArgs e)
{
    GridViewRow rw = dtgUser.SelectedRow;
    txtname.Text= rw.Cells[2].Text;
    txtemail.Text = dtgUser.SelectedRow.Cells[7].Text;
    int id = Convert.ToInt16(rw.Cells[1].Text);
    for (int i = 0; i < ddlCity.Items.Count - 1; i++) //dropdown
    {
        if (ddlCity.Items[i].Text == rw.Cells[3].Text)
        {
            ddlCity.SelectedIndex = i;
        }
    }
    //radiobutton
    for (int i = 0; i < rbl_gender.Items.Count; i++)
    {
        if (rbl_gender.Items[i].Text == rw.Cells[4].Text.Trim())
        {
            rbl_gender.Items[i].Selected = true;
            break; // You want to select only the first match
        }
    }
}
```

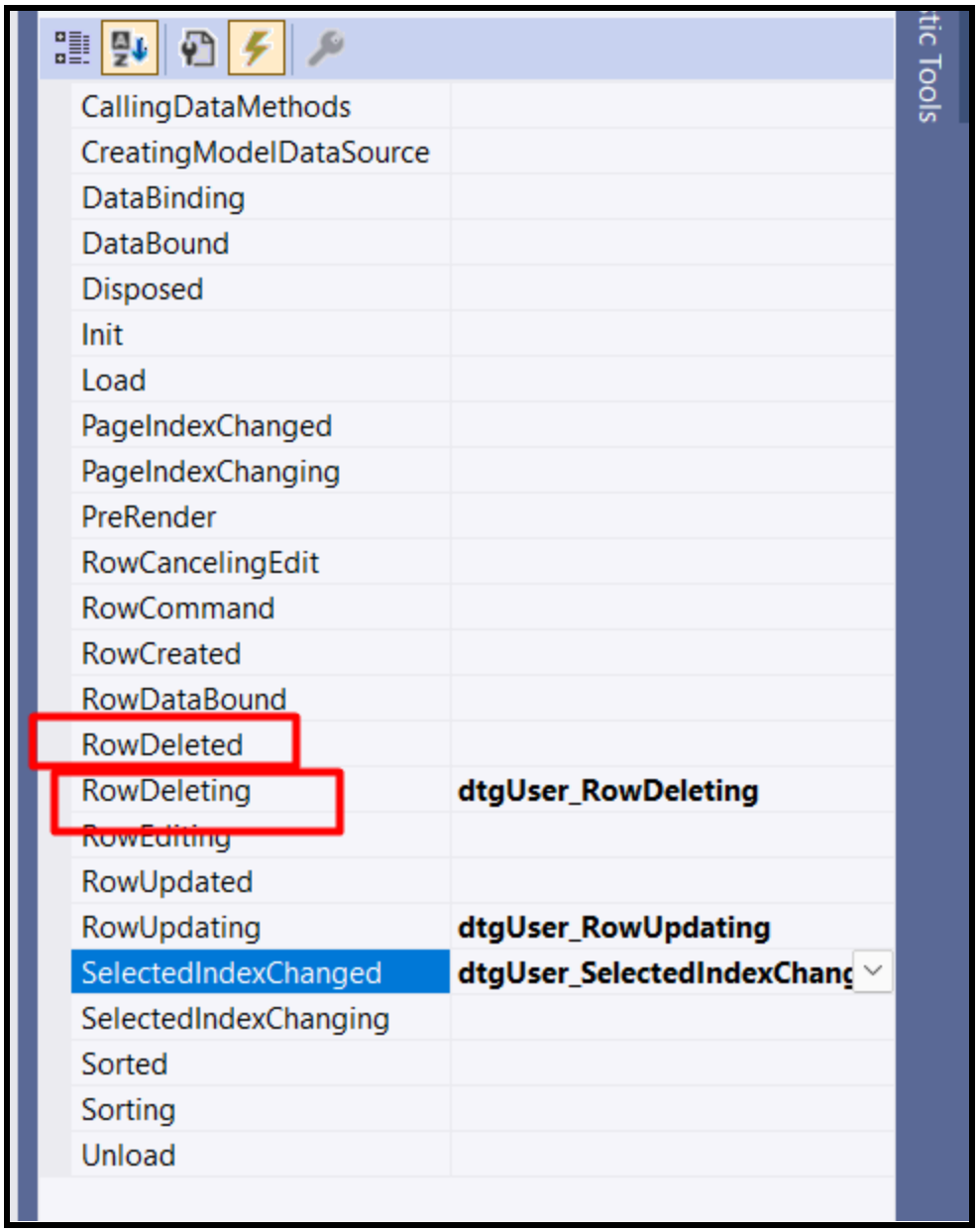
GridViewRow rw = dtgUser.Rows[e.RowIndex];

This line is typically used in an event handler for the GridView (e.g., the RowUpdating**,** RowDeleting**, or** RowCommand **event). The** e.RowIndex **represents the index of the row involved in the event. This line retrieves the GridViewRow based on the index provided in the event arguments** (e)**.**

e.RowIndex is a property commonly used in ASP.NET GridView events to retrieve the index of the row involved in the event. This property is part of the event arguments (e) passed to event handlers for GridView events that deal with row operations, such as updating, deleting, or selecting a row.

```
protected void dtgUser_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    GridViewRow rw = dtgUser.Rows[e.RowIndex];
    int del_id = Convert.ToInt32(rw.Cells[1].Text);
    try
    {
        fnConnectDB();
        string qry = "DELETE FROM tblUser WHERE Id=" + del_id;
        cmd = new SqlCommand(qry, conn);
        int res = cmd.ExecuteNonQuery();
        MessageBox.Show(cmd.CommandText);
        if (res > 0)
            Response.Write("Data Removed!");
        else
            Response.Write("<script>alert('Remove operation failed!')</
              script>");
        conn.Close();
        fnBindGrid();
    }
    catch (Exception ex)
    {
        Response.Write("<script>alert('Delete operation failed! " +
          ex.ToString() + "');</script>");
    }
    //}
```


0 references
protected void dtgUser_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
GridViewRow rw = dtgUser.Rows[e.RowIndex];
int del_id = Convert.ToInt32(rw.Cells[1].Text);
try
{
fnConnectDB();
string qry = "DELETE FROM tblUser WHERE Id=" + del_id;
cmd = new SqlCommand(qry, conn);
int res = cmd.ExecuteNonQuery();
MessageBox.Show(cmd.CommandText);
if (res > 0)
Response.Write("Data Removed!");
else
Response.Write("<script>alert('Remove operation failed!')</script>");
conn.Close();
fnBindGrid();
}
catch (Exception ex)
{
Response.Write("<script>alert('Delete operation failed! " + ex.ToString() + "');</script>");
}
//}
ROwDeleting is different than RowDeleted


tic Tools
CallingDataMethods
CreatingModelDataSource
DataBinding
DataBound
Disposed
Init
Load
PageIndexChanged
PageIndexChanging
PreRender
RowCancelingEdit
RowCommand
RowCreated
RowDataBound
RowDeleted
RowDeleting
dtgUser_RowDeleting
RowEditing
RowUpdated
RowUpdating
dtgUser_RowUpdating
SelectedIndexChanged
dtgUser_SelectedIndexChang
SelectedIndexChanging
Sorted
Sorting
Unload

Login

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    //using ExecuteScalar() method for login
    //Since ExecuteScalar() returns a count (an object) we used count() function
    try
    {
        fnConnectDB();
        string qry = "SELECT COUNT(*) FROM tbl_Users WHERE email=@eml AND password=@pwd";
        cmd = new SqlCommand(qry, conn);
        cmd.Parameters.AddWithValue("eml", txtemail.Text);
        cmd.Parameters.AddWithValue("pwd", txtPass.Text);
        int cnt = (int)cmd.ExecuteScalar(); //Always convert result into int before assigning the return
          count to an integer var
        if (cnt > 0)//This is because if user exists,count will be greater than 0
        {
            Session["s_eml"] = txtemail.Text;
            Response.Redirect("~/Welcome.aspx");
        }
        else
            Response.Write("Incorrect email or Password!");
        conn.Close();
    }
    catch (Exception ex)
    {
        Response.Write(ex.ToString());
    }
}
```