Unit 2: Database Access and Client server communication

Data access is a core part of application because most web apps need to retrieve, display, and manipulate data stored in a database or other storage system. This is done using data access code, either through ADO.NET which will discuss in this chapter. Client-server communication is a foundational model in networked applications where: A client sends requests. A server processes the request and sends back a response. Also discussed in this chapter.

2.1 Introduction about Ado.Net

- **ADO.NET (ActiveX Data Objects for .NET)** is a **data access technology** from the .NET Framework that provides a bridge between front-end controls and a back-end database.
- It is used to connect to databases like **SQL Server**, **MySQL**, **Oracle**, etc., and perform operations like **Insert**, **Select**, **Update**, and **Delete**.
- ADO.NET consists of two main components:
- **Connected Architecture**
    - It uses SqlConnection, SqlCommand, SqlDataReader
    - Data is directly accessible as long as the connection is open.
    - It is faster, lightweight, read-only
- **Disconnected Architecture**
    - It uses DataSet, DataTable, DataAdapter
    - It works without a live connection to the database
    - It is suitable for caching and manipulating data offline
- **Common ADO.NET Classes (System.Data.SqlClient) are listed below.**

Class	**Description**
SqlConnection	Opens a connection to SQL Server
SqlCommand	Executes SQL queries and stored procedures
SqlDataReader	Reads data in a forward-only, read-only manner
SqlDataAdapter	Acts as a bridge between DataSet and database
DataSet	In-memory representation of data (disconnected)
DataTable	Represents a single table in memory

SqlParameter	Used for parameterized queries to avoid SQL injection

2.2 Introduction about Provider, Adapter, Reader, Command Builder

Provider

- It is a set of classes used to connect to a specific type of data source (like SQL Server, Oracle, MySQL).

- Some common provider names are listed below:

Provider	**Namespace**	**Database**
SqlClient	System.Data.SqlClient	Microsoft SQL Server
OleDb	System.Data.OleDb	Access, Excel
Odbc	System.Data.Odbc	Any ODBC-supported DB
OracleClient	System.Data.OracleClient	Oracle (deprecated)

- **Each provider has the following classes listed below**:
  - Connection (SqlConnection)
  - Command (SqlCommand)
  - DataReader (SqlDataReader)
  - DataAdapter (SqlDataAdapter)

- **Each class will be explained below.**

1. **Connection Object:**

- This is the object that allows you to **establish a connection** with the data source.
- The connection object helps to identify the **database server**, the **database name**, **user name, Password** and other parameters those are required for connection to the database.

Properties

ConnectionString	It stores the connection string that is passed to the connection object at the time of creating its object
Database	It stores the name of the database to which you need to connect

State	It returns the state of the connection i.e **IsClose or IsOpen**
ConnectionTimeOut	Gets the time to wait while trying to establish a connection before terminating the attempt and generating an error.

Methods

Open	It opens the connection
Close	It closes the connection
Dispose	Releases resources.

- **Example**
- Define the connection string in the Web.config file.

```
<connectionStrings>
 <add name="constring" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\dbemployee.mdf;Integrated
Security=True;User Instance=True"
  providerName="System.Data.SqlClient" />
  </connectionStrings>
```

- Use the above connection string object in the code file.

```
Imports System.Data.SqlClient

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
    Try
        con.ConnectionString = constring
        cmd.Connection = con
    Catch ex As Exception
        lblmsg.Text = ex.Message
    End Try
 End Sub
```

2. **Command Object:**

- It uses the connection object to execute SQL queries.
- The queries can be in the Form of Inline text, Stored Procedures or direct Table access.If a select query is issued, the result set it returns is usually stored in either a DataSet or a DataReader object.

Properties:

Property	**Description**
Connection	To set connection object
CommandText	It specifies **SQL Statement** or the name of the Stored Procedure.
CommandType	This property indicates how the CommandText property should be interpreted. The possible values are: • Text (T-SQL Statement) • StoredProcedure (Stored Procedure Name) • TableDirect

Methods

Method	**Description**
ExecuteNonQuery	Used to execute an SQL statement and **returns the no. of rows affected** by the given statement. Return type of this method is **int**
ExecuteScalar	Used to execute an SQL statement and **return a single value**.
ExecuteReader	Used to execute a select a statement and return the rows returned by the select statement as a **DataReader**.

- Example: Write down the following code in button click event.

```
Protected Sub btnsave_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles btnsave.Click
        Try
            cmd.CommandText = "insert into tblusermst values('" & txtusername.Text & "','"
& txtsurname.Text & "','" & txtcity.Text & "')"
            con.Open()
            cmd.ExecuteNonQuery()
            con.Close()
            lblmsg.Text = "Record is inserted"
        Catch ex As Exception
            lblmsg.Text = ex.Message()
        End Try
    End Sub
```

3. DataAdapter:

- A DataAdapter acts as a bridge between the database and a DataSet. It fills a DataSet with data and can update the database using the changes made in the DataSet.
- **Properties:**

Property	**Description**
SelectCommand	Used to hold a command that retrieve data from the data source.
InsertCommand	Used to hold a command that insert data into the data source.
UpdateCommand	Used to hold a command that updates data to the data source.
DeleteCommand	Used to hold a command that delete data from the data source.
CommandType	This property indicates how the CommandText property should be interpreted. The possible values are: Text (T-SQL Statement) StoredProcedure (Stored Procedure Name) TableDirect

- **Methods:**

Method	**Description**
Fill	Loads data from the database.
Update	Saves changes back to the database.

- Example: Source code to retrieve data from a database and display it using a GridView control.

```
Sub fillgrid()
        Try
            cmd.CommandText = "select * from tblusermst order by id"
            da = New SqlDataAdapter(cmd)
            dt = New DataTable
            da.Fill(dt)
            grvuser.DataSource = dt
            grvuser.DataBind()
        Catch ex As Exception
            lblmsg.Text = ex.Message()
        End Try
    End Sub
```

- In the Page Load event, call the fillgrid method to populate the GridView.

```
  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
        If IsPostBack = False Then
            fillgrid()
        End If
    End Sub
```

4. DataReader

- It is a forward-only, read-only object used to read data from a SQL Server database row by row**.**It is working on **connected mode**. It is an alternative to the **Dataset and DataAdapter combination**.
- It is faster than DataSet
- **Properties:**

Property	**Description**
HasRows	Checks if the result set has rows.
FieldCount	Number of columns in the current row.
Item	Accesses column values by index or name.

- **Methods:**

Method	**Description**
Read()	Moves and read to the next record.
Close()	It is used to closes the SqlDataReader object
GetValue(), GetString(), GetInt32(), etc.:	Gets column values.

- Examlpe: Source code for a login form.

```
Protected Sub btnsignin_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnsignin.Click
    Try
        cmd.CommandText = "select * from tblusermaster where username='" & txtuname.Text & "' and pwd='" & txtpwd.Text & "'"
        con.Open()
        dr = cmd.ExecuteReader()
        If dr.HasRows = True Then
            Session("username") = txtuname.Text
            Response.Redirect("~/2024/frmdashboard.aspx")
```

```
        Else
            lblmsg.Text = "user name and password does not match."
        End If
    Catch ex As Exception
        lblmsg.Text = ex.Message
    End Try
    con.Close()
End Sub
```

5. Dataset

- It is an in-memory representation of data. It can be considered as **a local copy of the portions of the database. It is persisted in memory** and it can be manipulated and updated independent of the database.
- When the use of this dataset is finished, changes can be made back to the central the database for updating.
- It can hold multiple **DataTables**, each with rows, columns, constraints, and relationships — making it a **disconnected** data source (meaning it doesn't require an active database connection to manipulate data).
- **Properties**

Property	**Description**
Tables	Collection of DataTables in the DataSet.
Relations	Defines relationships between tables.
DataSetName	Name of the DataSet.
HasChanges()	Checks if the DataSet has changes.

- **Methods**

Method	**Description**
ReadXml()	Loads data from an XML file/string.
WriteXml()	Saves data to XML format.
GetChanges()	Returns a copy with all changes made.

- Example

```
Sub fillgrid()
  Try
     cmd.CommandText = "select * from tblusermst order by id"
     da = New SqlDataAdapter(cmd)
     ds = New DataSet
     da.Fill(ds)
     GridView1.DataSource = ds.Tables(0)
     GridView1.DataBind()
   Catch ex As Exception
     lblmsg.Text = ex.Message
   End Try
End Sub
```

6. Command Builder

- The ADO.NET object model does not only allow you to define your own updating logic, but it also provides a dynamic updating, using the CommandBuilder object. If you instantiate a CommandBuilder object and associate it with a DataAdapter object, the CommandBuilder will attempt to generate updating logic based on the query contained in the DataAdapter object's SelectCommand.

- The CommandBuilder can generate updating logic if all the following are true:

    - Your query returns data from only one table
    - That table has a primary key
    - The primary key is included in the results of your query

- Example

```
Protected Sub btnupdate_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles btnupdate.Click
    Try
       cmd.CommandText = "select * from tblemployee"
```

```
        da = New SqlDataAdapter(cmd)
        cmb = New SqlCommandBuilder(da)
        ds = New DataSet
        da.Fill(ds)
        For i = 0 To ds.Tables(0).Rows.Count - 1
            ds.Tables(0).Rows(i).Item(5) = ds.Tables(0).Rows(i).Item(5) + 2000
        Next
        da.Update(ds.Tables(0))
        lblmsg.Text = "updated"
    Catch ex As Exception
        lblmsg.Text = ex.Message
    End Try
End Sub
```

2.3 Database Access using ADO.NET

- To access data from database like SQL Server**,** Oracle**,** MySQL, ADO.NET framework is used. It is used to connect to a database, retrieve data, manipulate it, and update the database.
- **Key Components of ADO.NET**
    - **Connection**: Establishes a connection to a database.
    - **Command**: Executes SQL commands.
    - **DataReader**: Reads data from the database (forward-only, read-only).
    - **DataAdapter**: Bridges the DataSet and the database for data retrieval and update.
    - **DataSet**: In-memory representation of data.

- Example:

1. Create a Connection String in web.config (for ASP.NET)

```
<connectionStrings>
 <add name="constring" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\dbemployee.mdf;Integr
ated Security=True;User Instance=True"
  providerName="System.Data.SqlClient" />
  </connectionStrings>
```

2. Add Required Namespaces
   - Imports System.Data.SqlClient
   - Imports System.Data
3. Declare objects of different classes.

```
'getting connection string from web.config
 Dim constring As String =
ConfigurationManager.ConnectionStrings("constring").ToString()

   Dim con As New SqlConnection(constring)
   Dim cmd As New SqlCommand
   Dim da As SqlDataAdapter
   Dim dt As DataTable
```

4. In the Page Load event, we retrieve the connection string and open a database connection. So write down the below code in page load event.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
    Try
        con.ConnectionString = constring
        cmd.Connection = con
    Catch ex As Exception
        lblmsg.Text = ex.Message
    End Try
  End Sub
```

5. Insert /update/delete code using ExecuteNonQuery()

```
Protected Sub btnsave_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnsave.Click
    Try
        cmd.CommandText = "insert into tblusermst values ('" & txtusername.Text &
"','" & txtsurname.Text & "','" & txtcity.Text & "')"
        con.Open()
        cmd.ExecuteNonQuery()
        lblmsg.Text = "Record inserted successfully"
    Catch ex As Exception
        lblmsg.Text = ex.Message()
    End Try
    fillgrid()
    con.Close()
  End Sub
```

6. Create a method to retrieve data from the database, call it in the Page Load event, and reuse it whenever needed.

```
"Sub fillgrid()

    Try
        cmd.CommandText = "select * from tblusermst order by id"
        da = New SqlDataAdapter(cmd)
        dt = New DataTable
        da.Fill(dt)
        grvuser.DataSource = dt
        grvuser.DataBind()
    Catch ex As Exception
        lblmsg.Text = ex.Message
    End Try
  End Sub
```

2.4 Communications with Web Browser

- Communication with a web browser," refers to how clients (web browsers) send requests to a web server, and the server responds — usually over the HTTP/HTTPS protocol.
- This is the flow when client interact with server.
    - The user types a URL into the browser or clicks a hyperlink.
    - The browser sends an HTTP request to the web server for the specified resource.
    - The server receives the request, processes it (e.g., fetches data, runs code), and sends back an HTTP response.
    - The browser receives the response, renders the HTML, and executes any included JavaScript or other resources.

2.5 Response Object

- It is used to send output to the user from the server. It is part of the **HttpResponse** class. It provides wide range of properties and methods to control the HTTP response, such as writing output, setting headers, redirecting etc.
- **Properties**

Property	Description
ContentType	Specifies the type of content (e.g., text/html, application/json)
StatusCode	Sets the HTTP status code (e.g., 200, 404)
StatusDescription	Sets a brief description of the status code
Cookies	Collection of cookies sent to the browser

- **Methods**

Method	**Description**
Write()	Writes data to the output stream
Redirect()	Redirects to another URL
Clear()	Clears the buffer
End()	Ends response execution immediately

- **Example**

1. Send Plain Text to Browser

```
Response.Write("Hello India")
```

2. Redirect to Another Page

```
Response.Redirect("http://www.google.com")
```

3. Set Cookies using Response

```
Dim ckstudent As New HttpCookie("studinfo")
ckstudent.Values("name") = txtname.Text
ckstudent.Values("age") = txtage.Text
Response.Cookies.Add(ckstudent)
```

2.6 State Management Techniques

- Why “State" is Important in ASP.NET?
    - Let us assume that someone is trying to access a banking website and he has to fill in a form. So the person fills in the form and submits it. After submission of the form, the person realizes he has made a mistake.
    - So he goes back to the form page and he sees that the information he submitted is lost. So he again fills in the entire form and submits it again. This is quite annoying for any user. So to avoid such problems "STATE MANAGEMENT" techniques are used
    - HTTP is stateless, meaning:
        - Every request is independent.
        - The server does not automatically remember previous interactions.
        - So, to maintain continuity (like tracking a logged-in user, cart items, or form data), ASP.NET uses state management to preserve information across requests.
- **What is State?**
    - Information stored to maintain continuity between requests in a web application is called state.

2.6.1 Types of state management techniques

2.6.1.1 Client side technique

- These techniques store data on the client (browser). They're lightweight but less secure and have storage limits. These techniques are explained below.

2.6.1.1.1 ViewState

- It stores control values across postbacks.
- Data is stored in a **hidden field** (__VIEWSTATE) on the page.

Advantages

- No server memory required.
- Maintains control state automatically.

Disadvantages

- It increases page size.
- It is visible to users so it is not secure by default.

Example

Syntax:

Store value

ViewState(“keyname”)=value

Example:

```
ViewState("name")= txtname.Text
```

Syntax:

Retrive value

Variable name/Control Name= ViewState(“keyname”)

Example:

```
Label1.Text = ViewState("name")
```

2.6.1.1.2 Hidden Filed

- It is used to store small amount of data for specific page.
- The important property of Hidden field is Value.

Advantages

- No server memory required.
- Widespread Support which means almost all browsers and client devices support forms with hidden fields.
- Simple Implementation.

Disadvantages

- Can be modified on the client side.
- Performance Considerations
    - Storing large values can cause the page to slow.
- Potential Security risks
    - Values can be viewed clearly over a network.

Example

Syntax:

To Store value

HiddenField.value=Value

Example:

HiddenField1.value =txtname.Text

Syntax:

To Retrive value

Variable name/Control Name = HiddenField1.value

Example:

Label.Text= HiddenField1.value

2.6.1.1.3 QueryString

- It is a simple way to pass some information from one page to another.
- A String which is appended to the end of the page URL.
- It is used to send small amount of data and if security is not concern.
- It supports only 255 character length of url in query string.

Advantages

- No server memory required.
- Widespread Support which means almost all browsers and client devices support forms with hidden fields.
- Simple Implementation.

Disadvantages

- It is visible in the browser address bar.
- It is not suitable for sensitive data.
- Limited length

Example:

Syntax:

Sending only one QueryString value

Sender Page

```
Response.Redirect(“PageName?KeyName=Value”)
```

Example:

```
Response.Redirect("Webform1.aspx?name=" & txtname.Text)
```

Receiver Page

```
str=Request.QueryString(“KeyName”)
```

Example:

```
Label1.Text = "your name is" & Request.QueryString("name")
```

Sending more than one QueryString value

Syntax:

Sender Page

Response.Redirect(“PageName?KeyName1=Value1& KeyName2=Value2”)

Example:

Response.Redirect("Webform1.aspx?name=" & txtname.Text & "&age=" & txtage.Text)

Receiver Page

str1=Request.QueryString(“KeyName1”)

str2=Request.QueryString(“KeyName2”)

Example:

Label1.Text = "your name is" & Request.QueryString("name")

Label2.Text = "Your Age is " & Request.QueryString("age")

2.6.1.1.4 Cookies

- Cookies are small text files stored in the user’s browser to save data like preferences, login info, or session identifiers between page visits or across sessions.
- **There are two types of cookies**
- Persistence cookies
    - Stored on the user's computer and **persists** after the browser is closed.
    - Set with an **expiration date**.
- Non Persistence cookies
    - Stored in memory and **deleted when the browser is closed**.
    - No expiration date is set.

Advantages

- No server memory required.
- Widespread Support which means almost all browsers and client devices support forms with hidden fields.
- Simple Implementation.

Disadvantages

- Can be modified on the client side.
- Performance Considerations
    - Storing large values can cause the page to slow.

- Potential Security risks
    - Values can be viewed clearly over a network.

Example

Write down the following code in frmstorecookie.aspx file

```
Protected Sub btnstore_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
btnstore.Click

 Dim ckstudent As New HttpCookie("studinfo")
      ckstudent.Values("name") = txtname.Text
      ckstudent.Values("age") = txtage.Text
      ckstudent.Expires = Date.Now.AddHours(1)
      Response.Cookies.Add(ckstudent)
      Response.Redirect("frmretrivecookie.aspx")
End Sub
```

Write down the following code in frmretrivecookie.aspx file

```
      Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
      Label1.Text = Request.Cookies("studinfo").Values("name").ToString()
      Label2.Text = Request.Cookies("studinfo").Values("age").ToString()
  End Sub
```

2.6.1.2 Server side technique

These technique stores user-specific or application-wide data on the server, which is more secure and reliable than client-side storage.

2.6.1.2.1 Session state

- Session means interaction time between user and the web application.
- It is used to store **user-specific data** on the server for the duration of a user's session.
- There are different modes of session which are listed below.
    - **InProc** (in-memory, default)
    - **StateServer** (out-of-process, separate Windows service)
    - **SQLServer** (stored in SQL database)
    - **Custom** (your own provider)
- **Properties**

Property	**Description**
key	Stores or retrieves a value by key.
SessionID	Gets the unique session identifier for the current session.
Timeout	Gets or sets the timeout period (in minutes) for the session. Default time is 20 minute.
IsNewSession	Returns true if the session was just created.
Count	Gets the number of items in the session.
Mode	Gets the current session state mode (InProc, StateServer, etc.).
IsCookieless	Indicates if session is using cookies or cookieless URLs.

- **Methods**

Method	**Description**
Remove	Removes an item from the session.
RemoveAll()/ Clear()	Clears all session items.
Abandon()	Ends the session and releases all data. Creates a new session on next request.

Advantages

- It is easy to implement.
- The session data can be stored in database so the session data is persistent.

Disadvantages

- Increases server memory usage.

Example

- Storing a value in session

  Session("username") = txtuname.Text
- Retrieving a value from session

  lblusername.Text = Session("username")
- Removing a specific session variable

  lblusername.Text = Session("username")
- Clearing all session data

  Session.Clear() // Removes all session variables

  Session.Abandon() // Ends the session completely

2.6.1.2.2 Application State

- It is a global storage mechanism that is accessible from all pages in the Web application.
- It is also known as data repository for the applications.

- **Properties**

Property	**Description**
Count	Gets the number of items in the pplication.
Item	Gets the value of the specified index.
IsCookieless	Indicates if session is using cookies or cookieless URLs.

- **Methods**

Method	**Description**
Remove	Removes an item from the application state.

RemoveAll()/ Clear()	Clears all items from the application state.
Lock()	Locks the application state to prevent simultaneous access (for writes).
UnLock()	Unlocks the application state after a lock.

Example.

Write down the following code in Global.asax file.

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Code that runs on application startup
        Application("totalvisitor") = 0
    End Sub
Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Code that runs when a new session is started
        Application.Lock()
        Application("totalvisitor") += 1
        Application.UnLock()
 End Sub
```

Reading the Value on a Page

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
            lblmsg.Text = "Total Visitors: " & Application("totalvisitor")
End Sub
```