WHAT IS ASP.NET?

ASP.NET is an entirely new technology for server-side scripting. It was written from the ground up and is not backward compatible with classic ASP.

ASP.NET is the major part of the Microsoft's .NET Framework.

"ASP.NET is a server side scripting technology that enables scripts (embedded in web pages) to be executed by an Internet server."

- ASP.NET is a Microsoft Technology
- ASP stands for Active Server Pages
- ASP.NET is a program that runs inside IIS
- IIS (Internet Information Services) is Microsoft's Internet server
- IIS comes as a free component with Windows servers
- IIS is also a part of Windows 2000 and XP Professional
- Files containing HTML and scripting code
- Access via HTTP requests
- Scripting code is interpreted on server side

WHAT CAN I DO WITH ASP.NET?

- Easily and quickly create simple Web applications
- Generate dynamic Web content
- Client-side scripting for validation
- Access COM components to extend functionality

WHAT IS AN ASP.NET FILE?

- An ASP.NET file is just the same as an HTML file
- An ASP.NET file can contain HTML, XML, and scripts
- Scripts in an ASP.NET file are executed on the server
- An ASP.NET file has the file extension ".aspx"

HOW DOES ASP.NET WORK?

- When a browser requests an HTML file, the server returns the file
- When a browser requests an ASP.NET file, IIS passes the request to the ASP.NET engine on the server
- The ASP.NET engine reads the file, line by line, and executes the scripts in the file
- Finally, the ASP.NET file is returned to the browser as plain HTML

SEVEN TOUCHSTONES(ADVANTAGE) OF ASP.NET DEVELOPMENT.

- ASP.NET Is Integrated with the .NET Framework

- ASP.NET Is Compiled, Not Interpreted
  - One of the major reasons for performance degradation in ASP scripts is that all ASP web-page code uses interpreted scripting languages.
  - ASP.NET applications actually go through two stages of compilation.
    - *In the first stage, the C# code you write is compiled into an intermediate language called Microsoft Intermediate Language (MSIL) code, or just IL.*
    - *This first compilation step may happen automatically when the page is first requested, or you can perform it in advance (a process known as* precompiling*)*
  - The compiled file with IL code is an *assembly*.
  - The second level of compilation happens just before the page is actually executed.
  - At this point, the IL code is compiled into low-level native machine code. This stage is known as *just-in time* (JIT) compilation

- ASP.NET Is Multilanguage
  - No matter what language you use, the code is compiled into IL.
  - IL is a stepping-stone for every managed application. (A *managed application* is any application that's written for .NET and executes inside the managed environment of the CLR.)
  - It's easy enough to look at the IL for any compiled .NET application. You simply need to run the IL Disassembler, which is installed with Visual Studio and the .NET SDK (software development kit).
  - Look for the file ildasm.exe in a directory like c:\Program Files\Visual Studio 2005\SDK\v2.0\Bin.
  - Once you've loaded the program, use the File -> Open command, and select any DLL or EXE that was created with .NET.

- ASP.NET Runs Inside the Common Language Runtime
  - *Automatic memory management and garbage collection:*
  - *Type safety:*
  - *Extensible metadata:*
  - *Structured error handling:*
  - *Multithreading*

- ASP.NET Is Object-Oriented

- ASP.NET Is Multidevice and Multibrowser

- ASP.NET Is Easy to Deploy and Configure

PAGE SYNTAX

- Directives
  - <%@ Page language="C#" [...] %>
- Code Declaration Blocks
  - <script runat="server" [...]>
  - [lines of code]
  - </script>
- Code Render Blocks
  - <%
    [inline code or expression]
  - %>
- HTML Control Syntax
  - <HTMLelement runat="server" [attribute(s)]>
  - </HTMLelement>

CUSTOM CONTROL SYNTAX

- Custom server controls
  - <ASP:TextBox id="MyTb1" runat="server">
- Server control property
  - <ASP:TextBox maxlength="80" runat="server">
- Subproperty
  - <ASP:Label font-size="14" runat="server">
- Server control event binding
  - <ASP:Button OnClick="MyClick" runat="server">
- Data Binding Expression
  - <asp:label text='<%# *databinding expression* %>'
  - *runat="server" />*

- Server-side Object Tags
  - <object id="*id*" runat="server" *identifier*="*idName*" />
- Server-side Include Directives
  - <!-- #include *pathtype* = *filename* -->
- Server-side Comments
  - <%-- *comment block* --%>

PAGE DIRECTIVES

- Defines page-specific (.aspx file) attributes used by the ASP.NET page parser and compiler.
  <%@ Page *attribute*="*value*" [*attribute*="*value*"...] %> Attributes
- E.g <%@ Page language="C#" [...] %>

DIFFERENT ATTRIBUTES :

- **AutoEventWireup**
  - Indicates whether the page's events are autowired. true if event autowiring is enabled; otherwise, false. The default is true.
- **Buffer**
  - Determines whether HTTP response buffering is enabled. true if page buffering is enabled; otherwise, false. The default is true.
- **ClassName**
  - Specifies the class name for the page that will be dynamically compiled automatically when the page is requested. This value can be any valid class name but should not include a namespace.
- **ClientTarget**
  - Indicates the target user agent for which ASP.NET server controls should render content. This value can be any valid user agent or alias.
- **CodeBehind**
  - Specifies the name of the compiled file that contains the class associated with the page.
  - value of WebForm1.aspx.vb, for Visual Basic, or WebForm1.aspx.cs, for C#.
- **Strict**
  - Indicates that the page should be compiled using the Visual Basic Option Strict mode. true if Option Strict is enabled; otherwise, false. The default is false.
- **Trace**
  - Indicates whether tracing is enabled. true if tracing is enabled; otherwise, false. The default is false.

- **Transaction**
  - Indicates whether transactions are supported on the page. Possible values are Disabled, NotSupported, Supported, Required, and RequiresNew.
  - The default is Disabled.

- **ValidateRequest**
  - Indicates whether request validation should occur. If true, request validation checks all input data against a hard-coded list of potentially dangerous values. If a match occurs, an HttpRequestValidationException Class is thrown. The default is true.
  - This feature is enabled in the machine configuration file (Machine.config). You can disable it in your application configuration file (Web.config) or on the page by setting this attribute to false.

❖ THE .NET FRAMEWORK:

Microsoft .NET is a software component that runs on the Windows operating system.

The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

The .NET Framework has two main components: the common language runtime (CLR) and the .NET Framework Base class library (BCL).

The Microsoft .NET Framework was developed to solve this problem.

- Easier and quicker programming
- Reduced amount of code
- Declarative programming model
- Richer server control hierarchy with events
- Larger class library
- Better support for development tools

THE .NET FRAMEWORK CONSISTS OF 3 MAIN PARTS:

- Programming languages:
  - C# (Pronounced C sharp)
  - Visual Basic (VB .NET)
  - J# (Pronounced J sharp)
- Server technologies and client technologies:
  - ASP .NET (Active Server Pages)
  - Windows Forms (Windows desktop solutions)
  - Compact Framework (PDA / Mobile solutions)
- Development environments:
  - Visual Studio .NET (VS .NET)
  - Visual Web Developer

❖ COMPILE CODE

- Two models for coding web pages and web services:
  - Inline code:
    - *This model is the closest to traditional ASP.*
    - *All the code and HTML is stored in a single .aspx file.*
  - Code-behind:
    - *This model separates each ASP.NET web page into two files: an .aspx markup file with the HTML and control tags, and a .cs or .vb code file with the source code for the page.*

❖ COMMON LANGUAGE RUNTIME

The common language runtime is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that promote security and robustness. It is considered as the heart of the .NET framework.

The goals of the CLR are as follows:

- Secure and robust execution environment
- Simplified development process
- Multilanguage support
- Simplified management and simplified deployment

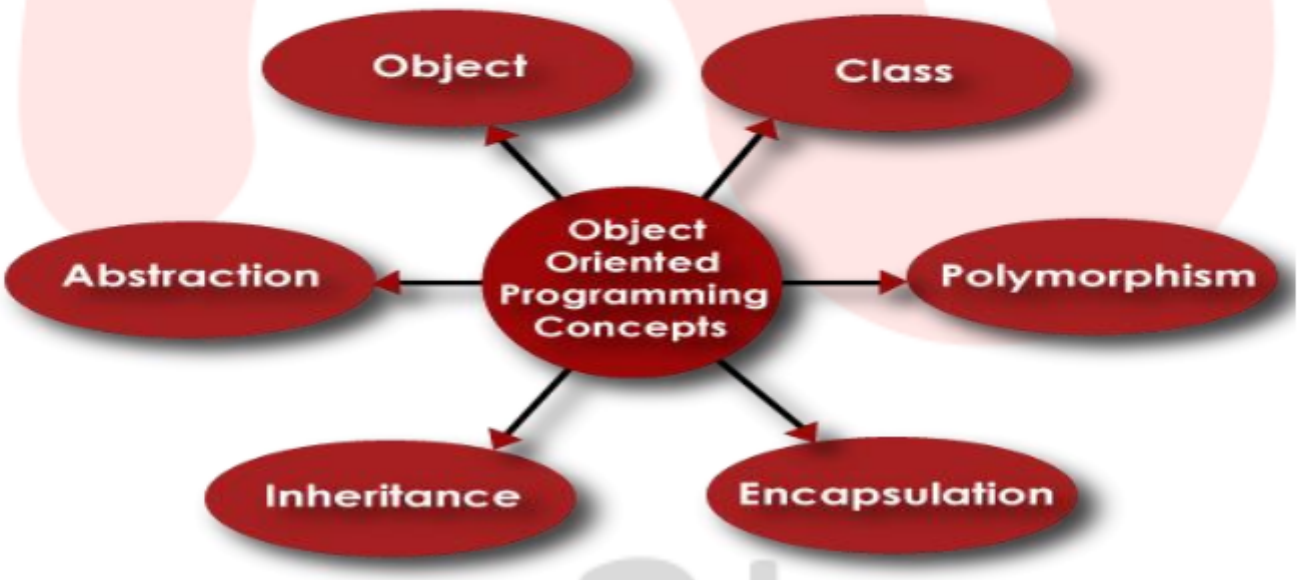
When you create any VB.NET application and if you compiled you code then it generate the CLR's intermediate code named as MSIL (Microsoft Intermediate Language) which is not directly executed by CPU so when we execute the code the MSIL code will be converted in to Native code which are executed by the CPU which is done by JIT compiler.

❖ OBJECT ORIENTED CONCEPTS

Object Oriented Programming (OOP) is one of the most popular programming languages.

In the class-based **object-oriented** programming paradigm, "**object**" refers to a particular instance of a class where the **object** can be a combination of variables, functions, and data structures. A good understanding of OOPs concepts can help in decision making when designing an application. How you should design an application and what language should be used.

Object Oriented programming is a programming style which is associated with the concepts like **class**, **object**, **Inheritance**, **Encapsulation**, **Abstraction**, **Polymorphism**.



OBJECT

Object is representative of the class and is responsible for memory allocation of its data members and member functions. An object is a real world entity having attributes (data type) and behaviors (functions).

It is a real time entity.

CLASS

Class is a data structure that contains data members (constants files, events), member function methods, properties, constructor, destructor, indexers and nested type. It is a collection of objects.

Basically

1) It is a user defined data type.
2) It is a reference type.
3) Infect class is a tag or template for object.

INHERITANCE

Inheritance is a process of deriving the new class from already existing class.

Inheritance is a feature of object-oriented programming that allows code reusability when a class includes property of another class. Considering Human Being a class, which has properties like hands, legs, eyes, mouth, etc, and functions like walk, talk, eat, see, etc.

ENCAPSULATION

Encapsulation means that we want to hide unnecessary details from the user. For example, when we call from our mobile phone, we select the number and press call button. But the entire process of calling or what happens from the moment we press or touch the call button to the moment we start having a phone conversation is hidden from us.

ABSTRACTION

Abstraction is "To represent the essential feature without representing the background details."

Abstraction lets you focus on what the object does instead of how it does it.

Abstraction provides you a generalized view of your classes or objects by providing relevant information.

The concept of abstraction focuses on what an object does, instated of how an object is represented or "how it works." Thus, data abstraction is often used for managing large and complex programs.

REAL-WORLD EXAMPLE OF ABSTRACTION

A Laptop consists of many things such as processor, motherboard, RAM, keyboard, LCD screen, wireless antenna, web camera, usb ports, battery, speakers etc. To use it, you don't need to know how internally LCD screens, keyboard, web camera, battery, wireless antenna, speaker's works. You just need to know how to operate the laptop by switching it on.

POLYMORPHISM

Polymorphism means one thing in many form. Basically polymorphism is capability of one object to behave in multiple ways. Example: A man role changes at home, college, and outside the home. There are following types of polymorphism:

1. ***Static polymorphism(compile time)*** : It is achieved using function overloading and operator overloading.
2. ***Dynamic polymorphism(runtime time)*** : It is achieved using function overriding means using virtual function.

❖ EVENT DRIVEN PROGRAMMING

In the conditions of event-based programming, objects (i.e., users) can initiate some events ('fire events') in the program, and the next thing happening in it is determined by those events. As a result, event-based programming fosters the dynamic interaction between users and computers.

After you design the look of your ASP.NET page using ASP.NET controls, you need to add the application logic. ASP.NET introduces the concept of an event-driven programming. It enables you to write code that executes in response to events raised by particular user actions. Events can be related to the page or the controls on the page. For example, you can write code to make your application do something when the page loads or a button is clicked.

HANDLING PAGE EVENTS

Whenever you request an ASP.NET page, a particular set of events is raised in a particular sequence. This sequence of events is called the page execution lifecycle.

Following is the sequence of events raised whenever you request a page:

1. PreInit
2. Init
3. InitComplete
4. PreLoad
5. Load
6. LoadComplete
7. PreRender
8. PreRenderComplete
9. SaveStateComplete
10. Unload

❖ AUTOMATIC POSTBACKS

- Windows developers have long been accustomed to a rich event model that lets your code react to mouse movements, key presses, and the minutest control interactions.
- But in ASP.NET, client actions happen on the client side, and server processing takes place on the web server.
- once the page is posted back, ASP.NET can fire other events at the same time
- ASP.NET web controls extend this model with an *automatic postback* feature. With this feature, input controls can fire different events, and your server-side code can respond immediately.
- To use automatic postback, you simply need to set the AutoPostBack property of a web control.

❖ DATA BINDING CONTROLS IN ASP.NET

Data-bound web server controls are controls that can be bound to a data source control to make it easy to display and modify data in your web application. All of these controls provide a variety of properties that you can set to control the appearance of the UI that they generate. For scenarios where you need greater control over a control's UI or how it processes input, some of these controls let you specify the generated HTML directly using templates. A template is a block of HTML markup that includes special variables that you use to specify where and how the bound data is to be displayed. When the control is rendered, the variables are replaced with actual data and the HTML is rendered to the browser.

Data-bound web server controls are composite controls that combine other ASP.NET web controls, such as Label and TextBox controls, into a single layout.

Every ASP.NET web form control inherits the DataBind method from its parent Control class, which gives it an inherent capability to bind data to at least one of its properties. This is known as **simple data binding** or **inline data binding**.

Simple data binding involves attaching any collection (item collection) which implements the IEnumerable interface, or the DataSet and DataTable classes to the DataSource property of the control.

On the other hand, some controls can bind records, lists, or columns of data into their structure through a DataSource control. These controls derive from the BaseDataBoundControl class. This is called **declarative data binding**.

The data source controls help the data-bound controls implement functionalities such as, sorting, paging, and editing data collections.

ASP.NET support following data binding controls

- GridView
- DataList
- DetailsView
- FormView
- ListView
- Repeater Control

GRIDVIEW DATA BINDING CONTROL

The GridView control is the successor to the DataGrid and extends it in a number of ways. With this GridView control, you could display an entire collection of data, easily add sorting and paging, and perform inline editing. In addition to just displaying data, the GridView can be used to edit and delete the displayed data as well.

The GridView comes with a pair of complementary view controls: DetailsView and FormView. By combining these controls, you can easily set up master-detail views using very little code and sometimes no code at all.

You can use the GridView control to do the following:

- Automatically bind to and display data from a data source control.
- Select, sort, page through, edit, and delete data from a data source control.

Additionally, you can customize the appearance and behavior of the GridView control by doing the following:

- Specifying custom columns and styles.
- Utilizing templates to create custom user interface (UI) elements.
- Adding your own code to the functionality of the GridView control by handling events.

DATA BINDING WITH THE GRIDVIEW CONTROL

The GridView control provides you with two options for binding to data:

- Data binding using the DataSourceID property, which allows you to bind the GridView control to a data source control. This is the recommended approach because it allows the GridView control to take advantage of the capabilities of the data source control and provide built-in functionality for sorting, paging, and updating.

- Data binding using the DataSource property, which allows you to bind to various objects, including ADO.NET datasets and data readers. This approach requires you to write code for any additional functionality such as sorting, paging, and updating.

When you bind to a data source using the DataSourceID property, the GridView control supports two-way data binding. In addition to the control displaying returned data, you can enable the control to automatically support update and delete operations on the bound data.

FORMATTING DATA DISPLAY IN THE GRIDVIEW CONTROL

You can specify the layout, color, font, and alignment of the GridView control's rows. You can specify the display of text and data contained in the rows. Additionally, you can specify whether the data rows are displayed as items, alternating items, selected items, or edit-mode items. The GridView control also allows you to specify the format of the columns. For information on formatting the GridView control, see the GridView class overview.

EDITING AND DELETING DATA USING THE GRIDVIEW CONTROL

By default, the GridView control displays data in read-only mode. However, the control also supports an edit mode in which it displays a row that contains editable controls such as TextBox or CheckBox controls. You can also configure the GridView control to display a Delete button that users can click to delete the corresponding record from the data source.

The GridView control can automatically perform editing and deleting operations with its associated data source, which allows you to enable editing behavior without writing code. Alternatively, you can control the process of editing and deleting data programmatically, such as in cases where the GridView control is bound to a read-only data source control.
You can customize the input controls that are used when a row is in edit mode using a template. For more information, see the TemplateField class.

DATALIST BINDING CONTROL

The DataList control displays data items in a repeating list, and optionally supports selecting and editing the items. The content and layout of list items in DataList is defined using templates. At a minimum, every DataList must define an ItemTemplate; however, several optional templates can be used to customize the appearance of the list. The following table describes those templates.

TEMPLATE NAME	DESCRIPTION
ItemTemplate	Defines the content and layout of items within the list. Required.
AlternatingItemTemplate	If defined, determines the content and layout of alternating items. If not defined, ItemTemplate is used.
SeparatorTemplate	If defined, is rendered between items (and alternating items). If not defined, a separator is not rendered.
SelectedItemTemplate	If defined, determines the content and layout of the selected item. If not defined, ItemTemplate (AlternatingItemTemplate) is used.
EditItemTemplate	If defined, determines the content and layout of the item being edited. If not defined, ItemTemplate (AlternatingItemTemplate, SelectedItemTemplate) is used.
HeaderTemplate	If defined, determines the content and layout of the list header. If not defined, the header is not rendered.
FooterTemplate	If defined, determines the content and layout of the list footer. If not defined, the footer is not rendered.

DETAILSVIEW BINDING CONTROL

DetailsView is a data-bound user interface control that renders a single record at a time from its associated data source, optionally providing paging buttons to navigate between records. It is similar to the Form View of an Access database, and is typically used for updating and/or inserting new records. It is often used in a master-details scenario where the selected record of the master control (GridView, for example) determines the DetailsView display record.

The DetailsView control gives you the ability to display, edit, insert, or delete a single record at a time from its associated data source. By default, the DetailsView control displays each field of a record on its own line. The DetailsView control is typically used for updating and inserting new records, often in a master/detail scenario where the selected record of the master control determines the record to display in the DetailsView control. The DetailsView control displays only a single data record at a time, even if its data source exposes multiple records.

The DetailsView control relies on the capabilities of the data source control to perform tasks such as updating, inserting, and deleting records. The DetailsView control does not support sorting.

The DetailsView control can automatically page over the data in its associated data source, provided that the data is represented by an object supporting the ICollection interface or that the underlying data source supports paging. The DetailsView control provides the user interface (UI) for navigating between data records. To enable paging behavior, set the AllowPaging property to true.

You select a particular record from the associated data source by paging to that record. The record displayed by the DetailsView control is the current selected record.

THE REPEATER CONTROL

The Repeater control displays data items in a repeating list. Similar to DataList, the content and layout of list items in Repeater is defined using templates. At a minimum, every Repeater must define an ItemTemplate; however, the following optional templates may be used to customize the appearance of the list.

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

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

1) itemTemplate 2) AlternatingitemTemplate 3) HeaderTemplate 4) FooterTemplate

5) SeperatorTemplate

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

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

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

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

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

Unlike DataList, Repeater has no built-in layout or styles. You must explicitly declare all HTML layout, formatting, and style tags within the templates of the control. For example, to create a list within an HTML table, you might declare the <table> tag in the HeaderTemplate, a table row (<tr> tags, <td> tags, and data-bound items) in the ItemTemplate, and the </table> tag in the FooterTemplate.

Example of Repeater Control

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>

<asp:Repeater ID="Repeater1" runat="server" DataSourceID="SqlDataSource1">
<HeaderTemplate>

<table cellspacing="0" rules="all" border="1">
<tr>
<th scope="col" style="width: 80px">
        Roll No
</th>
<th scope="col" style="width: 120px">
        Student
</th>

</tr>

</HeaderTemplate>

<ItemTemplate>
<tr>
<td bgcolor="#CCFFCC">
<asp:Label runat="server" ID="Label1"
        text='<%# Eval("rollno") %>' />
```

```
</td>

<td bgcolor="#CCFFCC">
<asp:Label runat="server" ID="Label2"
                text='<%# Eval("sname") %>' />
</td>
</tr>
</ItemTemplate>

<AlternatingItemTemplate>
<tr>
<td >
<asp:Label runat="server" ID="Label3"
                text='<%# Eval("rollno") %>' />
</td>
<td >
<asp:Label runat="server" ID="Label4"
                text='<%# Eval("sname") %>' />
</td>
</tr>
</AlternatingItemTemplate>

<SeparatorTemplate >
<tr>
<td >
          =========
</td>
</SeparatorTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
        SelectCommand="SELECT * FROM [stud]"></asp:SqlDataSource>
</div>
</form>
</body>
</html>
```

FORMVIEW

FormView is a data-bound user interface control that renders a single record at a time from its associated data source, optionally providing paging buttons to navigate between records. It is similar to the DetailsView control, except that it requires the user to define the rendering of each item using templates, instead of using data control fields.

The main difference between DetailsView and FormView is that DetailsView has a built-in tabular rendering, whereas FormView requires a user-defined template for its rendering. The FormView and DetailsView object model are very similar otherwise. The following example demonstrates a FormView control bound to an ObjectDataSource. The ItemTemplate property of FormView contains data-bound Image, Label and HyperLink controls.

The FormView control is typically used for updating and inserting new records. It is often used in master/detail scenarios where the selected record of the master control determines the record to display in the FormView control. For more information and an example, see Modifying Data Using a FormView Web Server Control.
The FormView control relies on the capabilities of the data source control to perform tasks such as updating, inserting, and deleting records. The FormView control displays only a single data record at a time, even if its data source exposes multiple records.

❖ **WEB SERVER CONTROLS**

❖ **HTML SERVER CONTROLS (BASIC HTML SERVER CONTROL)**

❖ **VALIDATION CONTROLS**

ASP.NET INCLUDES SIX VALIDATION CONTROLS.

- <asp:RequiredFieldValidator>
- <asp:RangeValidator>
- <asp:CompareValidator>
- <asp:RegularExpressionValidator>
- <asp:CustomValidator>
- <asp:ValidationSummary>

- you can use more than one validator for the same control.
- you can't validate RadioButton or CheckBox controls
- you can validate the TextBox(the most common choice) and other controls such as ListBox, DropDownList, RadioButtonList,HtmlInputText, HtmlTextArea, and HtmlSelect.

THE VALIDATION PROCESS

- You can use the validation controls to verify a page automatically when the user submits it or to verify it manually in your code.
- Every button has a CausesValidation property, which can be set to true or false.
  - If false, ASP.NET will ignore the validation controls, the page will be posted back, and your event-handling code will run normally.
  - If true (the default), ASP.NET will automatically validate the page when the user clicks the button.

THE BASEVALIDATOR CLASS

- The validation control classes are found in the System.Web.UI.WebControls namespace and inherit from the BaseValidator class.
- This class defines the basic functionality for a validation control.
- BaseValidator Members
  - ControlToValidate
  - EnableClientScript
  - Display
  - Enabled
  - ErrorMessage
  - ValidationGroup

THE REQUIREDFIELDVALIDATOR CONTROL

- It's work is to ensure that the associated control is not empty
- E.g.
  - <asp:TextBox runat="server" ID="Name" />

<asp:RequiredFieldValidator runat="server"

ControlToValidate="Name" ErrorMessage="Name is required" Display="dynamic">

*

</asp:RequiredFieldValidator>

THE RANGEVALIDATOR CONTROL

- The RangeValidator control verifies that an input value falls within a predetermined range.
- It has three specific properties: Type, MinimumValue, and MaximumValue.
  - *Type* defines the type of the data that will be typed into the input control and validated.
  - The available values are Currency, Date, Double, Integer, and String.
- E.g.

```
<asp:TextBox runat="server" ID="DayOff" />

<asp:RangeValidator runat="server" Display="dynamic"

ControlToValidate="DayOff" Type="Date"

ErrorMessage="Day Off is not within the valid interval"

MinimumValue="08/05/2002" MaximumValue="08/20/2002">

*

</asp:RangeValidator>
```

THE COMPAREVALIDATOR CONTROL

- The CompareValidator control compares a value in one control with a fixed value or, more commonly, a value in another control.
- For example, this allows you to check that two text boxes have the same data or not.
- E.g.-1

```
<asp:TextBox runat="server" ID="Age" />

<asp:CompareValidator runat="server" Display="dynamic"
ControlToValidate="Age" ValueToCompare="18"

ErrorMessage="You must be at least 18 years old"

Type="Integer" Operator="GreaterThanEqual"> *

</asp:CompareValidator>
```

- E.g.-2

```
<asp:TextBox runat="server" TextMode="Password" ID="Password" />

<asp:TextBox runat="server" TextMode="Password"

ID="Password2" />

<asp:CompareValidator runat="server" ControlToValidate="Password2"
    ControlToCompare= "Password" ErrorMessage="The passwords don't match"

Type="String" Display="dynamic">

<img src="imgError.gif" border="0" alt="The passwords don't match">

</asp:CompareValidator>
```

THE REGULAREXPRESSIONVALIDATOR CONTROL

- It allows you to validate text by matching against a pattern defined in a regular expression.
- You simply need to set the regular expression in the ValidationExpression property.
- For example, the following control checks that the text input in the text box is a valid e-mail address:

```
<asp:TextBox runat="server" ID="Email" />

<asp:RegularExpressionValidator runat="server"

ControlToValidate="Email" ValidationExpression= ".*@.{2,}\..{2,}" ErrorMessage=
    "E-mail is not in a valid format" Display="dynamic"> *

</asp:RegularExpressionValidator>
```

- Metacharacters for Matching Single Characters
- Metacharacters for Matching Types of Characters
- Quantifiers
- E-mail address : * \S+@\S+\.\S+

THE CUSTOM VALIDATOR CONTROL

- if you need more advanced or customized validation, then the CustomValidator control is what you need.
- TheCustomValidator allows you to execute your custom client-side and server-side validation routines.
- It takes two parameters: a reference to the validator and a custom argument object.
- This object provides a Value property that contains the current value of the associated input control (the value you have to validate) and an IsValid property through which you specify whether the input value is valid.

THE VALIDATIONSUMMARY CONTROL

- The ValidationSummary control doesn't perform any validation. Instead, it allows you to show a summary of all the errors in the page.
- This summary displays the ErrorMessage value of each failed validator.
- The summary can be shown in a client-side JavaScript message box (if theShowMessageBox property is true) or on the page (if the ShowSummary property is true).
- Youcan set both ShowMessageBox and ShowSummary to true to show both types of summaries.
- E.g.

```
<asp:ValidationSummary runat="server" ID="ValidationSum"
ShowSummary="true" DisplayMode="BulletList"  HeaderText="<b>Please review
the following errors:</b>" />
```

❖ NAVIGATION CONTROLS

Navigation controls are very important for websites.Navigation controls are basically used to navigate the user through webpage .It is more helpful for making the navigation of pages easier .There are three controls in ASP.NET ,Which are used for Navigation on the webpage.

1. TreeView control
2. Menu Control

TREEVIEW CONTROL

The TreeView control is used to display hierarchical data, such as a table of contents or file directory, in a tree structure.

A Tree View control displays a hierarchical list of items using lines to connect related items in a hierarchy. Each item consists of a label and an optional bitmap. Windows Explorer uses a Tree View control to display directories. You can use the Tree View control in any situation in which you need to display hierarchical data.

The TreeView control is used for logically displaying the data in a hierarchical structure. We can use this navigation control for displaying the files and folders on the webpage.

TREEVIEW NODE TYPES

The TreeView control is made up of one or more nodes. Each entry in the tree is called a node. The following table describes the three different node types.

TreeView control node types

NODE TYPE	DESCRIPTION
Root	A node that has no parent node and one or more child nodes.
Parent	A node that has a parent node and one or more child nodes.
Leaf	A node that has no child nodes.

Although a typical tree has only one root node, the TreeView control allows you to add multiple root nodes to your tree structure. This is useful when you want to display item listings without displaying a single main root node, as in a list of product categories.

Each node has a Text property and a Value property. The value of the Text property is displayed in the TreeView control, while the Value property is used to store any additional data about the node, such as data passed to the postback event that is associated with the node.

MENU Control

The ASP.NET menu control allows you to develop both statically and dynamically displayed menus for your ASP.NET Web pages.

The Menu control has two modes of display: static and dynamic. Static display means that the Menu control is fully expanded all the time. The entire structure is visible, and a user can click on any part. In a dynamically displayed menu, only the portions you specify are static, while their child menu items are displayed when the user holds the mouse pointer over the parent node.

The Menu control is used to create a menu of hierarchical data that can be used to navigate through the pages. The Menu control conceptually contains two types of items. First is StaticMenu that is always displayed on the page, Second is DynamicMenu that appears when opens the parent item.

Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. are implemented through style properites of <table, tr, td/> tag.

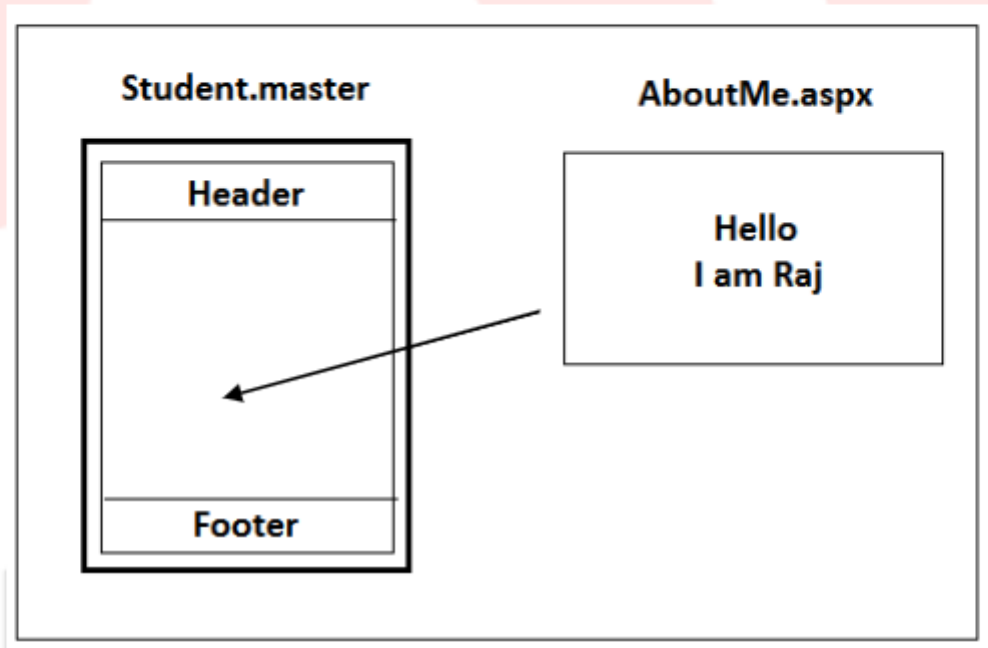
Following are some important properties that are very useful.

PROPERTIES OF MENU CONTROL	
DataSourceID	Indicates the data source to be used (You can use .sitemap file as datasource).
Text	Indicates the text to display in the menu.
Tooltip	Indicates the tooltip of the menu item when you mouse over.
Value	Indicates the nondisplayed value (usually unique id to use in server side events)
NavigateUrl	Indicates the target location to send the user when menu item is clicked. If not set you can handle MenuItemClick event to decide what to do.
Target	If NavigationUrl property is set, it indicates where to open the target location (in new window or same window).
Selectable	true/false. If false, this item can't be selected. Usually in case of this item has some child.
ImageUrl	Indicates the image that appears next to the menu item.
ImageToolTip	Indicates the tooltip text to display for image next to the item.
PopOutImageUrl	Inidcates the image that is displayed right to the menu item when it has some subitems.
Target	If NavigationUrl property is set, it indicates where to open the target location (in new window or same window).
STYLES OF MENU CONTROL	
StaticMenuStyle	Sets the style of the parent box in which all menu items appears.
DynamicMenuStyle	Sets the style of the parent box in which dynamic menu items appears.
StaticMenuItemStyle	Sets the style of the individual static menu items.
DynamicMenuItemStyle	Sets the style of the individual dynamic menu items.
StaticSelectedStyle	Sets the style of the selected static items.
DynamicSelectedStyle	Sets the style of the selecdted dynamic items.
StaticHoverStyle	Sets the mouse hovering style of the static items.
DynamicHoverStyle	Sets the mouse hovering style of the dynamic items (subitems).

- **LOGIN CONTROL**
- **MASTER PAGE**

- MasterPages allows you to create a consistent look and behavior for all the pages (or group of pages) in your web application.

- MasterPage means not HomePage.
- A MasterPage provides a template for other pages, with shared layout and functionality.
- If you want to disply standard header, footer and menu in each page in your website, you can create the standard header, footer and menu in a MasterPage.
- Microsoft has used a concept from PowerPoint application with Master pages. In powerpoint you can create a master slide that acts as a template for all other slides. The background and common text on the master slide is automatically used in other slides.
- Likewise the Master Page is used to create the look and feel of ASP.NET web application.
- Master Pages help us build consistent and maintainable user interface.
- Remember that It is not related with color-combination concept; it is related with **Layout** – like frameset of HTML.
- They allow you to layout a page and use it over and over.
- The MasterPage defines placeholders for the content, which can be overridden by content pages. The output result is a combination of the master page and content page.
- The content page contains the content you want to display.



- When user request the content page, ASP.NET merges the pages to produce output that combines the layout of the Master Page with the content of the content page.

Characteristics: -

- Extension of MasterPage is '.master' it also has a code behind file.
- MasterPage can not be run directly.

- MasterPage can be nested.
- MasterPage allows centralizing the common functionality of the pages so that we can make updates at one place only.
- Website can contain many Master Pages and you can create different set of content pages for each Master Page.
- A MasterPage has 2 parts:
    1. Content that appears on each page that inherits the master page.
    2. Regions that can be customized by the pages inheriting the master page.
- A Master Page need to specify both the parts common to all pages and the parts that are customizable.
- Use ContentPlaceHolder control to specify a region that can be customize.
- Master Page should contain at least one CntentPlaceHolder.

ContentPlaceHolder:-

- When a new Master Page is initially added that time you will see ContentPlaceHolder1 on the webpage.
- ContentPlaceHolders are used to allocate places on the MasterPage that will be changed on the content pages.
- ContentPlaceHolder is a control in MasterPage. The content from the content page appears in the area made by ContentPlaceHolder.
- There are two ContentPlaceHolder on any MasterPage.
    1. In the Head section.
    2. In the between <form> </form>
- In the head you specify certain Meta fields such as the page description and keywords. These vary from page to page.
- In the form section the actual webpage contents will presents.
- You can have multiple ContentPlaceHolders on the same page.

Content Pages:-

- The content pages contain the content you want to display.
- For example – Home.aspx, AboutUs.aspx, ContactUs.aspx, etc... are content pages.
- You define the content for the master page's placeholder controls by creating individual content pages, which are ASP.NET pages that are bound to a specific Master Page.
- When users request the content pages, they merge with the MasterPage to produce output that combines the layout of the MasterPage with the contents from the content page.

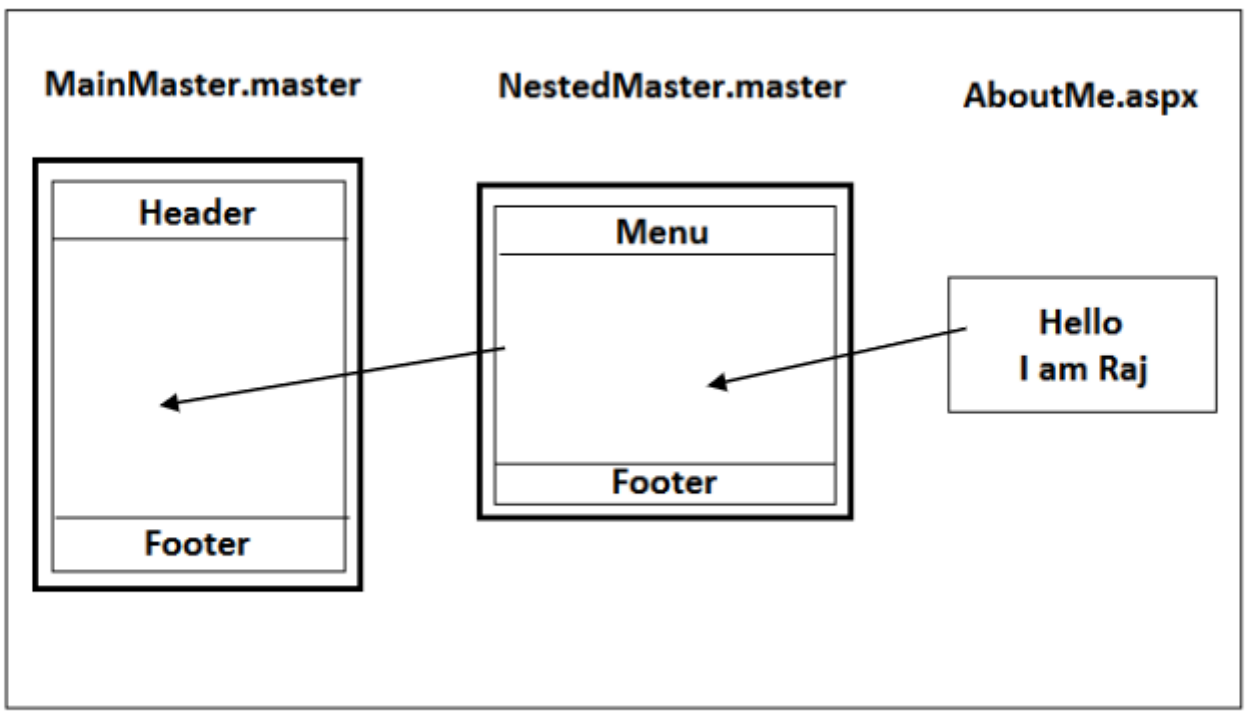
Run-time Behavior of Master Pages:-

- Users request a page by typing the URL of the content page.
- ASP.NET fetches the page.

- When the page is fetched, the @page directive is read. If the directive references a Master Page, the Master Page is read as well. If this is the first time the pages have been requested, both pages are compiled.
- ASP.NET merges the content into the ContentPlaceHolders on the MasterPage.
- The MasterPage with the updated content is merged into the control tree of the content page.
- ASP.NET renders result to the browser.

Nested MasterPage

- When one Master Page references another as its master, it is said to be a Nested Master Page.
- For example- you can create one Master Page for the entire website and then nest Master Pages below the site Master Page for individual sectons.



- A number of nested masters can be componentized into a single master.
- There is no architectural limitations to the number of child master that can be created.
- The depth of nesting also does not impact on performance significantly.
- The advantages of this kind of structuring is that a number of child masters can be created to be subordinated to the overall look and feel of the site defined by the parent master, while the child master give the child pages some uniqueness. While a parent master defines the overall layout of the pages header, body and footer, the child master expands the body for a group of pages.
- Just like the parent master page child masters have the extension '.master' too.
- It contains all the controls that are mapped to ContentPlaceHolders on the parent master page.

- Child masters also have ContentPlaceHolders of their own to display contents of its child pages.

❖ THEMES & CSS

- A Master Page enables user to share content across multiple pages in a website, while a Theme enables you to control the appearance of the content.
- Like the concept of changing the Theme in "Orkut".
- A theme is a collection of property settings that allow you to define the look of pages and controls, and then apply the look consistently across pages in a web application, across an entire web application, or across all web application on a server.
- The concept is when one user is visiting site and seeing it, another user can be view the same site, but gets a completely different experience.
- Themes can be thought of as a container where you store your style sheets, images, skin files, etc.
- For theme here is am App_Theme folder, we can add this folder by right clicking on web project add new folder.
- Themes are applied on the server: so we can change it runtime.
- A theme folder can contain a variety of different types of files:
    - Skins
    - Cascading Style Sheets (CSS)
    - Images and other resources.
- Themes are control-based, not HTML-based: One of the biggest advantages of themes is that unlike CSS, it allows you to define and reuse almost any control property.
- Themes can be applied through configuration files.
- Only one theme can be applied to each page.

SKIN FILE:-

- A skin generally contains visual properties (for look) for one or more kinds of ASP.NET controls (server control)
- A skin file has the file name extension '.skin'
- It contains property settings for individual controls such as Button, Label, TextBox, Calendar, etc. server side controls.
- You can define skins in a separate file for each control or define all the skins for a theme in a single file.

DEFAULT SKIN:-

- A default skin automatically applies to all controls of the same type when a theme is applied to a page.
- The SkinId is not defined.

- Only one default control skin per control type is allowed in the same theme.
- For example, if you create a default skin for a TextBox control, the control skin applies to all TextBox controls on pages that use the theme.

NAMED SKIN:-

- It is a control skin with a **SkinID** property set.
- Named skin do not automatically apply to controls by type.
- In Named skin, user can decide when you want to apply the Skin.
- The SkinID should be uniquely defined because duplicate SkinID's per control type are not allowed in the same theme.

CSS FILE:-

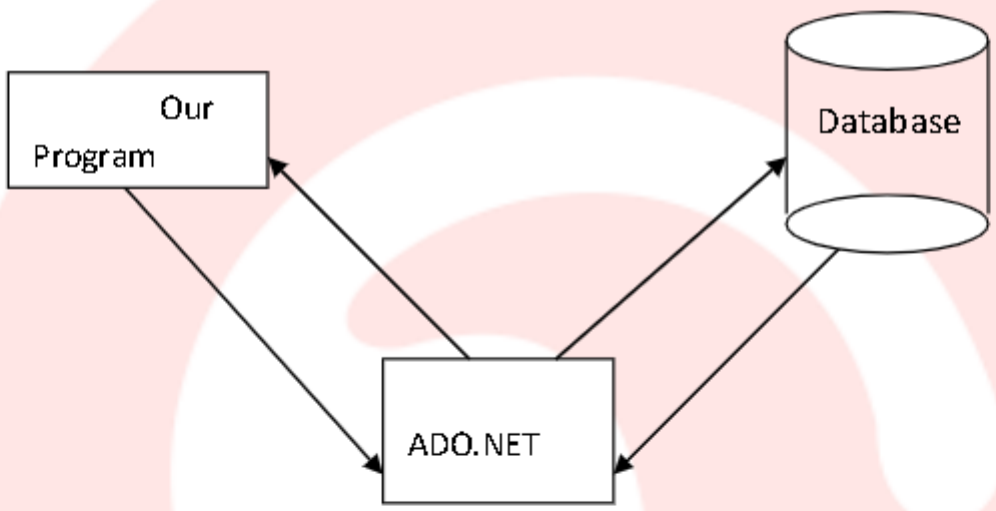
- A Cascading Style Sheet (CSS) may be added to a theme by placing it under the named theme subdirectory.
- Extension of the CSS file is '.css'.
- When you put a '.css' file in the theme directory, the style sheet is applied automatically as part of the theme.
- The CSS will be applied to all pages with that theme applied.

POINTS TO REMEMBER:-

- Each and every control has property **Enable Theme**, user can set either true or false.
- It indicates whether the control can be themed or not.
- If the user set it as a false then the effect never comes. By default it's True.

❖ ADO.NET

- ADO.NET is a part of the Microsoft .Net Framework.
- ADO.NET consists of a set of classes used to handle data access
- ADO.NET is entirely based on XML
- It's stands for ActiveX® Data Objects.
- ADO.NET has the ability to separate data access mechanisms, data manipulation mechanisms and data connectivity mechanisms.
- ADO.NET is a set of classes that allow applications to read and write information in databases.



- ADO.NET can be used by any .NET language. It's a concept. It's not a programming language.
- We need to add **Syastem.Data** namespace for work with ADO.NET.
- ADO.NET introduces the concept of **disconnected architecture.**

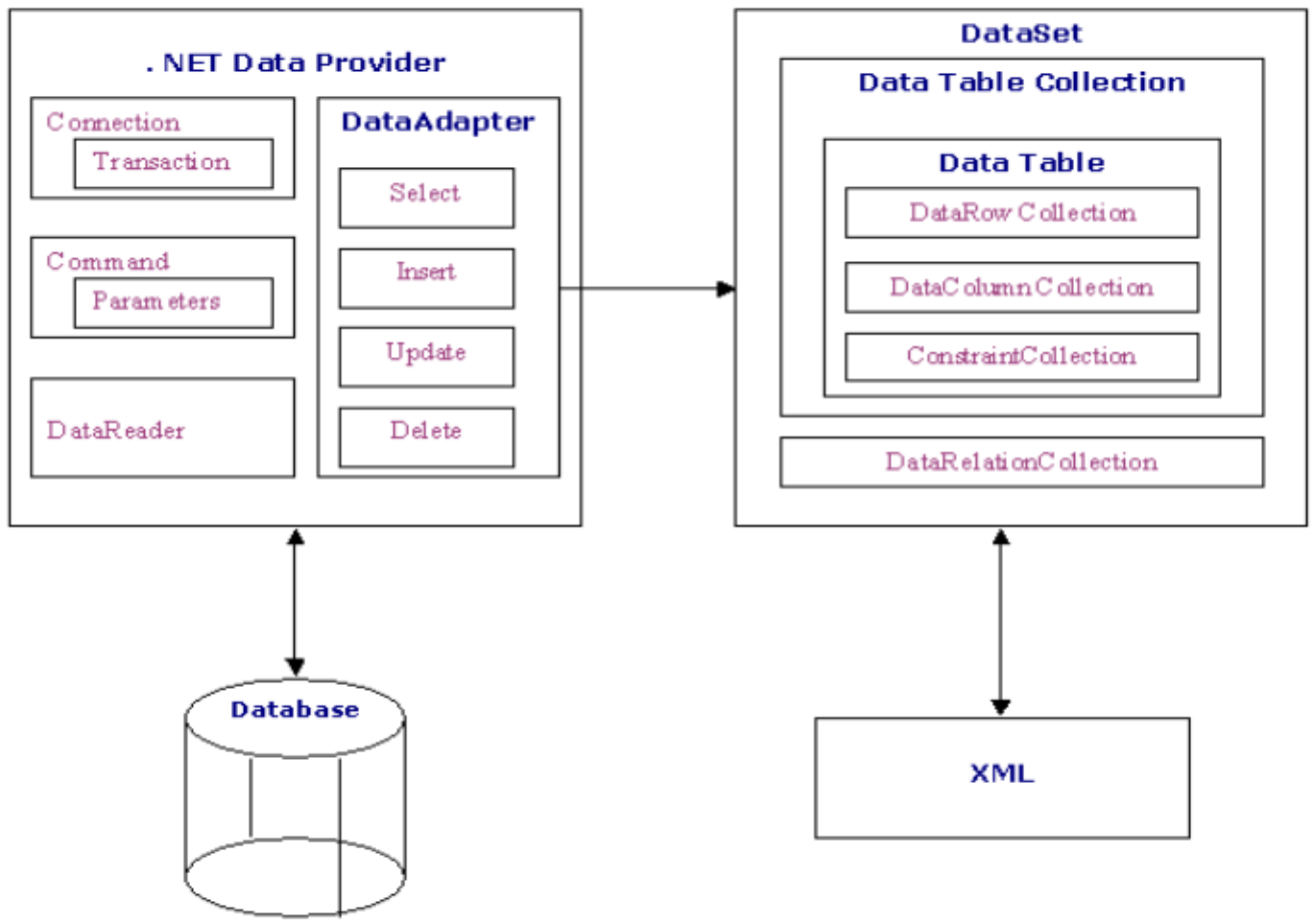
❖ Disconnected environment

- In traditional data access components, we make a connection to the database server and then interact with it. The application stays connected to the database server even when it is not using database service. It is waste the valuable and expensive database resource.
- ADO.NET solves this problem by managing a local buffer of persistent data called **DataSet**. Our application automatically connects to the database server when it needs to pass some query and then disconnects immediately after getting the result back and storing it in dataset.

❖ There are two major components of ADO.NET

- **DataSet:-** It represents either an entire database or a subset of database. It can contain tables and relationships between those tables.
- **Data Provider :-** It is a collection of components like Connection, Command, DataReader, DataAdapter objects and handles communication with a physical data store and the dataset.

❖ **ADO.NET Architecture**



ADO .NET Data Architecture

❖ **Data Provider**

- The data provider is responsible for providing and maintaining the connection to the database.
- It is a set of classes that can be used for communicating with database and manipulating data.
- The data provider connects to the data source on behalf of ADO.NET.
- The data source can be Microsoft SQL server or Oracle database and OLEDB data provider.
- The data provider components are specific to data source.

- **The following list display various data providers:**

For SQL	Imports System.Data.SqlClient namespace. It provides data access for Microsoft SQL Server.

For OLED

Imports System.Data.OleDb namespace.

We can use OLE DB for connect Microsoft Access.

For

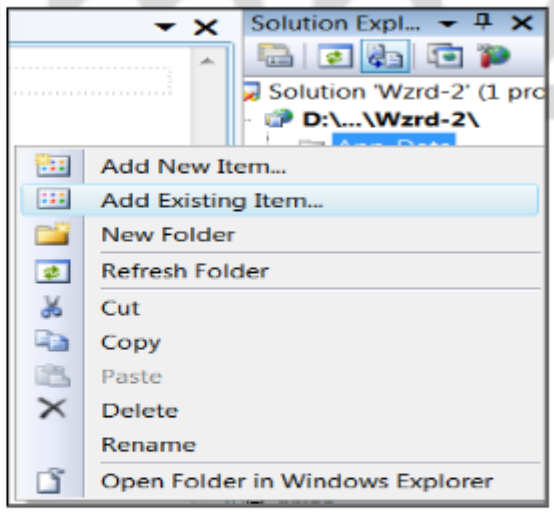
Imports System.Data.OracleClient namespace.

It provides data access for Oracle.

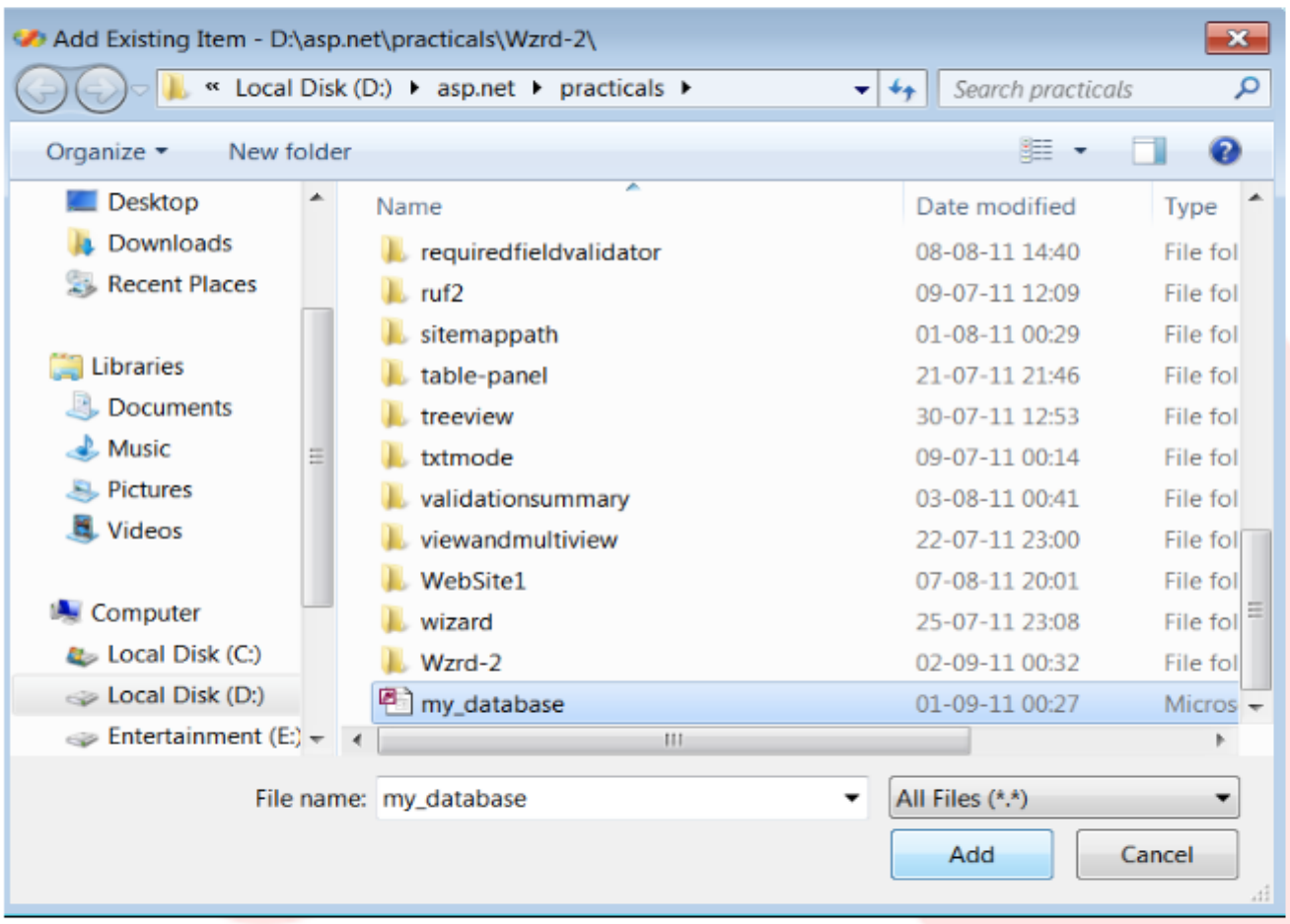
- **Following are the four core objects that make up a .NET Framework data provider.**
  1. **Connection :-** It establishes a connection to the data source.In Oledb the connection can establish using OledbConnection object.
  2. **Command :-** Fires SQL commands or perform some action on the data source such as insert, update and delete.
     In Oledb the command can fires using Oledbcommand object.
  3. **DataAdapter :-** It is a bridge between Data source and Dataset object for transferring data.
     In Oledb the dataadapter can create using OledbDataAdapter object.
  4. **DataReader :-** It reads records in a read-only, forward-only mode. In Oledb the datareader can create using OledbDataReader object.

❖ Wizard through Connectivity

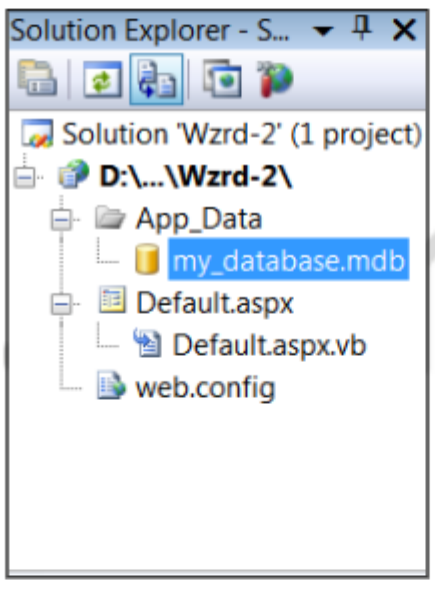
- Create a database in Microsoft access, add table and also insert records in it.
- In solution explorer window, right click on **App_Data** and then click on Add Existing Item...



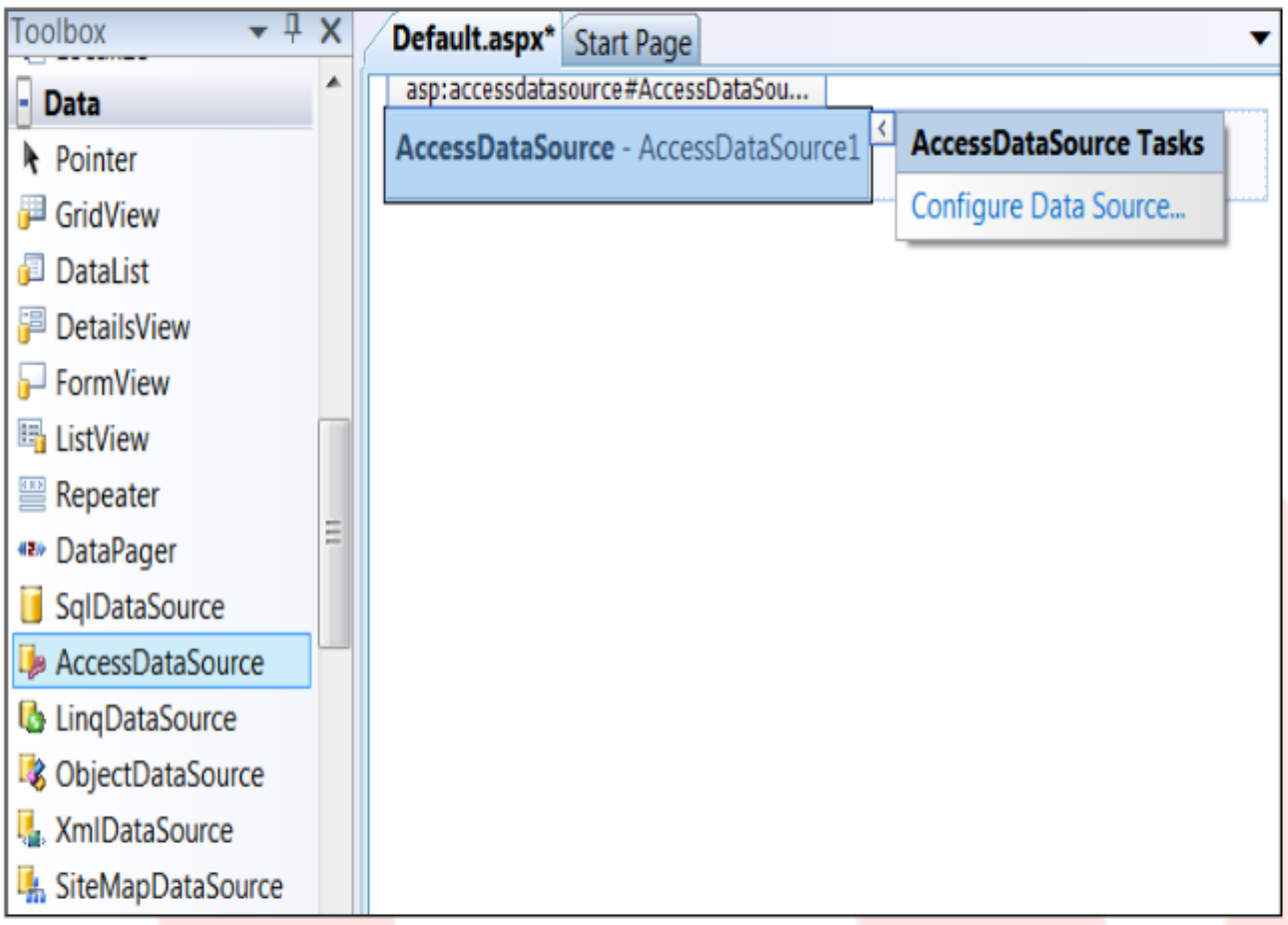
- Now, Add your database from the particular location.



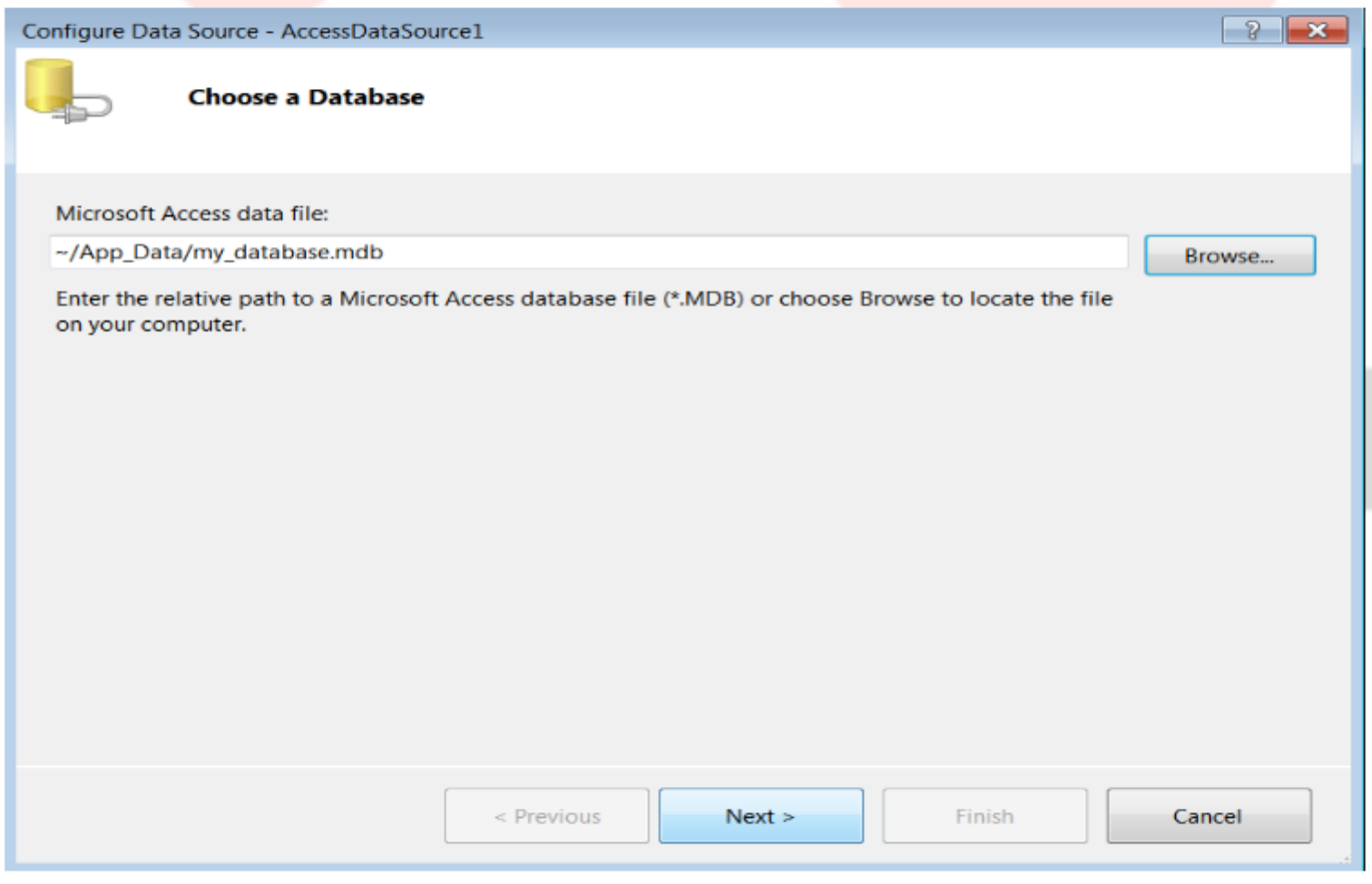
- Your Database automatically stored in APP_DATA folder.



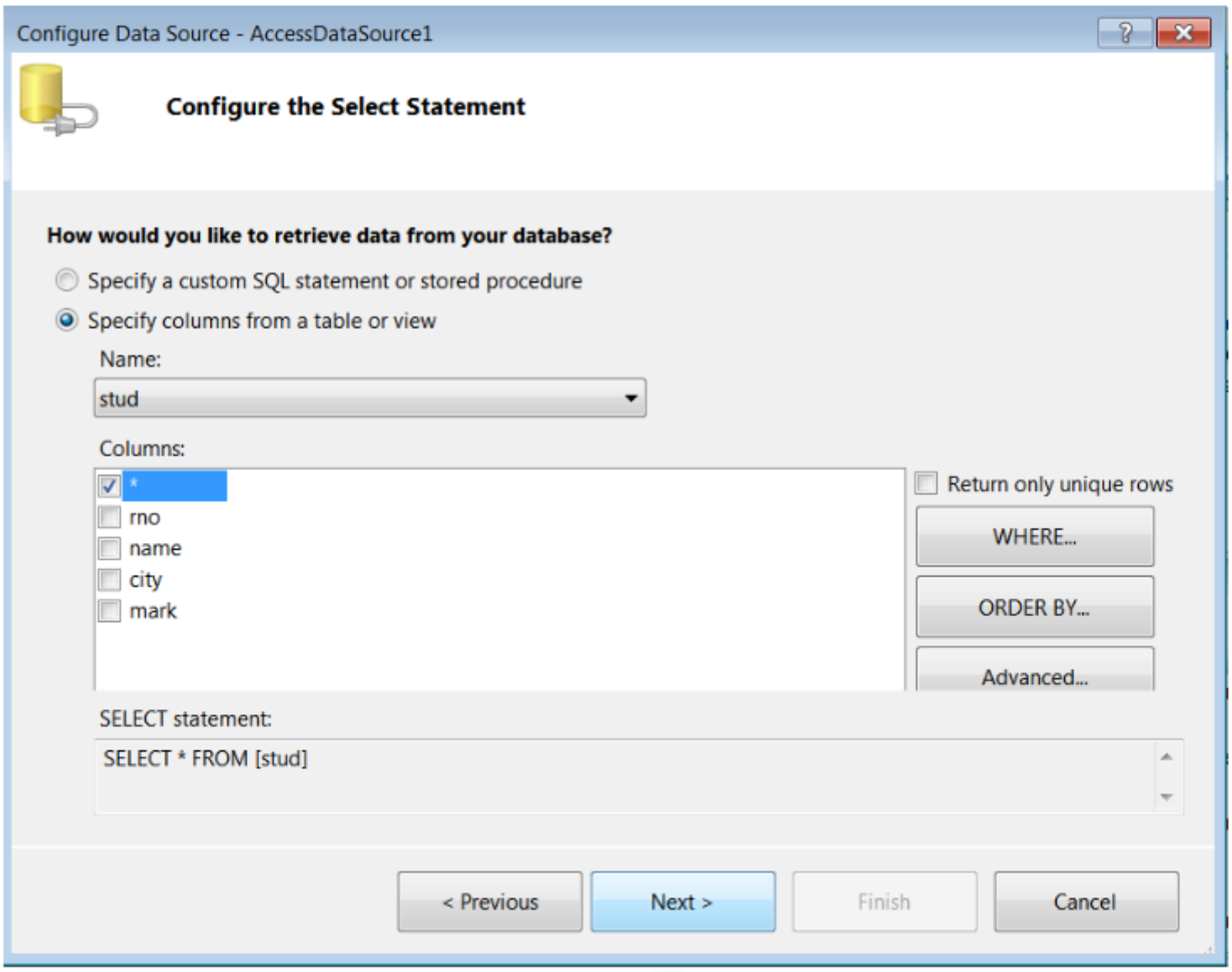
- Now, Add AccessDataSource in our WebPage and then configure your Data Source.



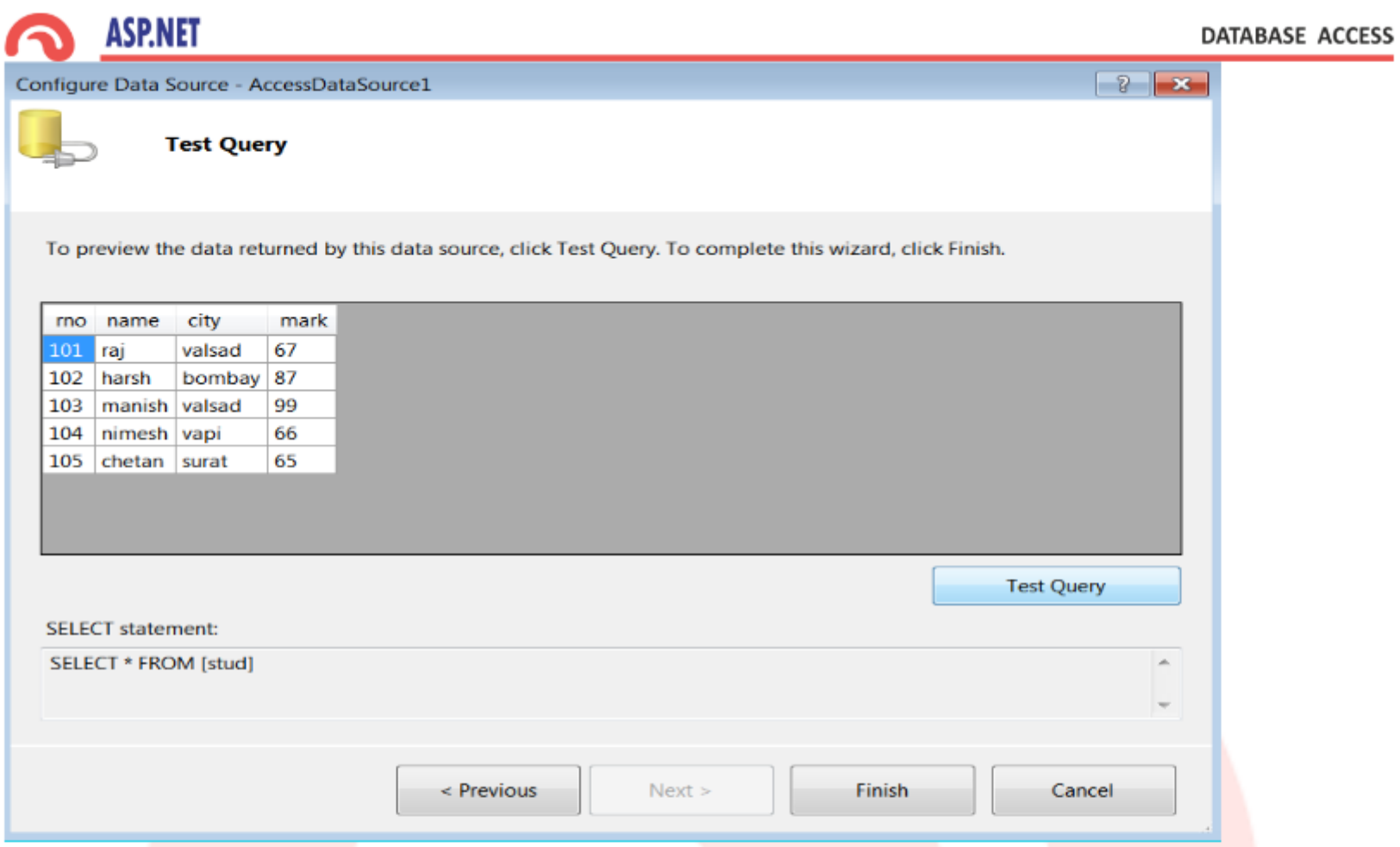
- Click on Browse button and choose your Database and then click on Next.



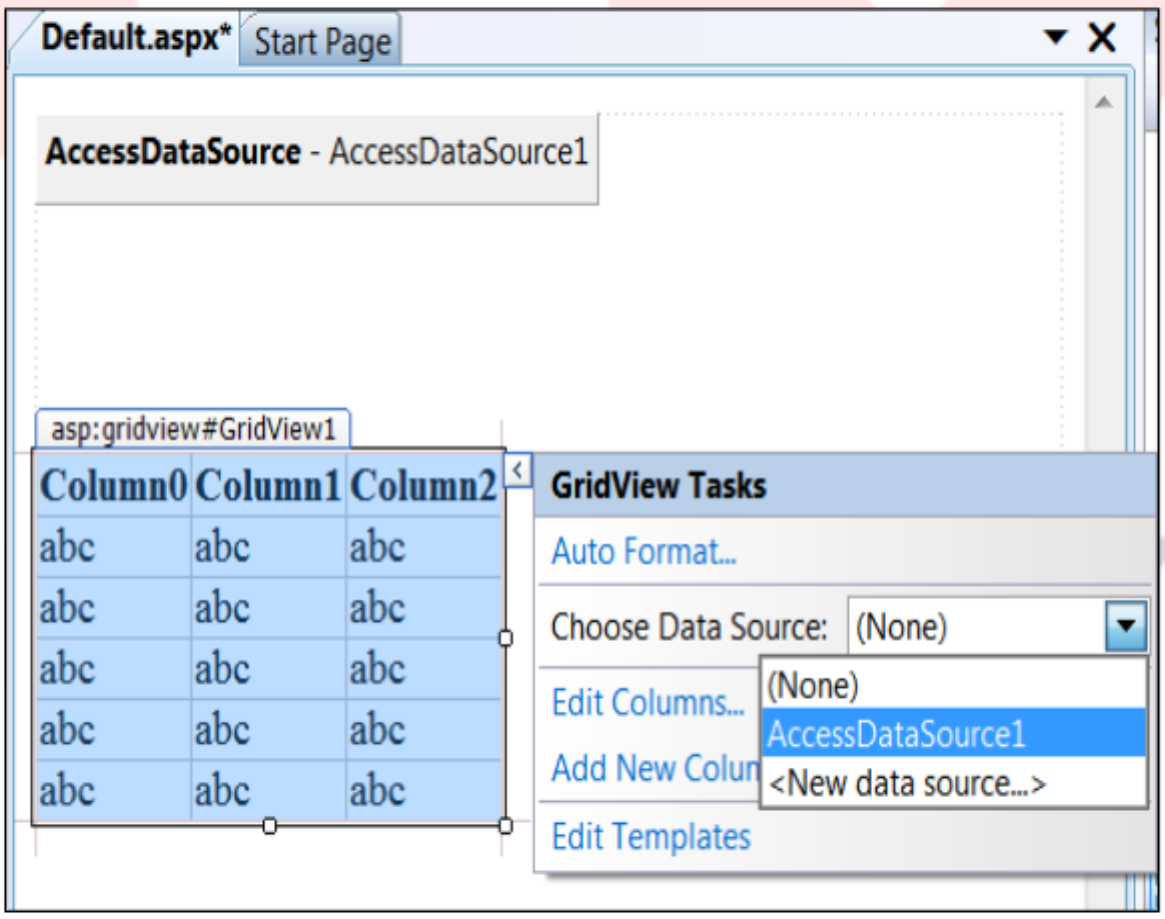
- Now, configure the Select Statement. We have to select our specific table and the columns to be retrieve from the table. Here, we checked for all columns so all columns will be retrieve from the table.
- Click on Next.



- Click on **Test Query** Button so that the data returned by the Data Source will be displayed.
- To complete this wizard click on Finish.



- Now, Add a GridView control on our WebPage and Bind it with Database by selecting **AccessDataSource1** from Choose Data Source.



- Now, you can Run your application.

❖ Connectivity With Coding

- For any of the ADO.NET program with CODING, first we need to Imports namespace called System.Data and then add SQLClient for SQL server and add OLEDB for Access before any class declaration.
- We will use the System.Data.Oledb.

Ex:-

```
Imports System.Data.OleDb

Partial Class _Default

Inherits System.Web.UI.Page
```

- The System.Data contains basic enumerations and classes.

❖ Connection object :-

The Connection object creates the connection to the database.

- The Connection object represents the physical connection to a data source.
- The Connection object contains all of the information required to open a connection to the database.
- Basic Steps :-
  1. Create Database.
  2. Create Connection object
  3. Set the Provider.
  4. Specify the data source.

- Connection object have **ConnectionString** property, It has a group of semi-colon-separated attributes.
- Depends on the parameter specified in the **Connection String,** ADO.NET Connection object connect to the specified Database.

Ex:-

```
Imports System.Data.OleDb

Partial Class _Default

Inherits System.Web.UI.Page
```

```
Protected Sub Page_Load(ByVal sender As.....

Dim con As New OleDbConnection("Provider=Microsoft.jet.OLEDB.4.0;

Data Source=D:\asp.net\practicals\Connectivity\App_Data\my_database.mdb")
```

- **Methods of Connection object.**
  - Open :- It opens the connection to our database.
  - Close :- It closes the database connection.
  - Dispose :-It releases the resources on the connection object.
  - State :- Returns state of connection object. Returns 0, if not connected and 1, if connected.

❖ **Command object** :-

It is depends on Connection Object.

- Command objects are used to execute commands to a database across a data connection.
- The Command object in ADO.NET executes SQL statements against the data source specified in the Connection object.
- Command object can support SQL statements that return single value; one or more sets of rows or no value at all.
- The Command object has a property called **CommandText**, which contains a String (Query) value that represents the command that will be executed in the Data Source.

- **Properties of Command object**

1. **Connection Property :-**

This property contains data about the connection string. It must be set on the OledbCommand object before it is executed.

- For the Command to execute properly, the connection must be open at the time of execution.
- We can initialize it as given below:

```
Dim cmd As New OleDbCommand("select * from stud", con)

OR

Dim str As String

str = "select * from stud"
```

```
Dim cmd As New OleDbCommand(str, con)
```

2. CommandText Property :-

This property specifies the SQL string or stored procedure. (It's specify SQL statement) **Ex :-**

```
Imports System.Data.OleDb

Partial Class _Default

    Inherits System.Web.UI.Page

Protected Sub Page_Load(ByVal sender As Object,ByVal e As System.EventArgs) Handles
Me.Load

    Dim con As New OleDbConnection ("Provider=
    Microsoft.jet.OLEDB.4.0; Data Source= D:\asp.net\practicals\
    Connectivity\App_Data\my_database.mdb")
  Dim cmd As New OleDbCommand
  cmd.Connection = con
    cmd.CommandType = Data.CommandType.Text
cmd.CommandText = "select * from stud"
```

3. Parameters Collection Property :-

If we want to update values in the student table, but w don't know the values at design time, we make use of placeholders.

- These are variables prefixed with "@" symbol.
- Our code will look like this....

```
cmd.CommandText = "insert into stud(rno,name,city,mark)
values(@rno,@name,@city,@mark)"
```

- Next, we have to create parameters that will be used to insert values.
- For this, we need to add parameters to the parameters collection of the OledbCommand object. This is done so that the values added through the parameters collection & placeholders get included in the SQL statement.
- To add parameters to the OledbCommand object, we write the following code:

Ex: -

```
Imports System.Data.OleDb
Partial Class _Default
   Inherits System.Web.UI.Page
 Protected Sub btnAddRec_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles btnAddRec.Click

    Dim rno, mark As Integer
    Dim name, city As String

Dim con As New OleDbConnection (" Provider= Microsoft.jet.OLEDB.4.0; Data Source=
D:\asp.net\practicals\Connectivity\ App_Data\my_database.mdb")

      Dim cmd As New OleDbCommand

        cmd.Connection = con
    cmd.CommandType = Data.CommandType.Text
    cmd.CommandText = "insert into stud(rno,name,city,mark)
values(@rno,@name,@city,@mark)"

      rno = 101
      name = "raj"
      city = "valsad"
      mark = 55

  'Add parameters

    cmd.Parameters.Add("@rno", OleDbType.Integer)

    cmd.Parameters.Add("@name", OleDbType.Char)

    cmd.Parameters.Add("@city", OleDbType.Char)

    cmd.Parameters.Add("@mark", OleDbType.Integer)


  'Initialize parameters

    cmd.Parameters.Item("@rno").Value = rno

    cmd.Parameters.Item("@name").Value = name
```

```
cmd.Parameters.Item("@city").Value = city

cmd.Parameters.Item("@mark").Value = mark


con.Open()

cmd.ExecuteNonQuery()

con.Close()

End Sub

End Class
```

- **Methods of Command object**

  - Command object has three important methods for execute queries.
  - Connection should be open for below method :

    1. ExecuteScalar
    2. ExecuteNonQuery
    3. ExecuteReader

1. **ExecuteScalar:-**

   - It's use with aggregate functions such as Max, Min, Avg, Sum, Count...
   - It returns only one value at a time.
   - It fetches the data from the database.

Example:-

First put two textboxes and two buttons in your web page.

First button will count the total number of records available in a **stud** table and second button will find out the maximum **mark**.

Now, write the following code :

```
Imports System.Data.OleDb

Partial Class methds_cmdobj

Inherits System.Web.UI.Page
```

```
    Dim con As New OleDbConnection("Provider=Microsoft.jet.OLEDB.4.0;Data Source=
D:\asp.net\practicals\connectivity\App_Data\my_database.mdb")

Protected Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Button2.Click

Dim cmd As New OleDbCommand("select max(mark) from stud",con)

con.Open()

TextBox2.Text = cmd.ExecuteScalar()

con.Close()

End Sub

Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Button1.Click

 Dim cmd As New OleDbCommand("select count(rno) from stud",con)

 con.Open()

 TextBox1.Text = cmd.ExecuteScalar()

 con.Close()

End Sub

End Class
```

2. ExecuteNonQuery:-

- It's used for fire Insert, Update, Delete queries to the Database.
- It requires open connection.
- It returns number of rows affected. It means if we delete 5 records then it returns 5.

- **Example of ExecuteNonQuery for Delete**

```
Imports System.Data.OleDb

Partial Class methds_cmdobj

Inherits System.Web.UI.Page
```

```
Dim con As New OleDbConnection("Provider=Microsoft.jet.OLEDB.4.0;Data Source=
D:\asp.net\practicals\connectivity\App_Data\my_database.mdb")

    Protected Sub btnDel_Click(ByVal sender As Object, ByVal e As
System.EventArgs)Handles btnDel.Click

  Dim ans As Integer

  Dim cmd As New OleDbCommand("delete from stud where rno=" & TextBox1.Text, con)

   con.Open()

   ans = cmd.ExecuteNonQuery()

   con.Close()

     MsgBox("Total No. Of Records Deleted :" & ans)

  End Sub

End Class
```

- **Example of ExecuteNonQuery for Insert**

```
Imports System.Data.OleDb

Partial Class methds_cmdobj

Inherits System.Web.UI.Page

Dim con As New OleDbConnection("Provider=Microsoft.jet.OLEDB.4.0;Data Source=
D:\asp.net\practicals\connectivity\App_Data\my_database.mdb")

 Protected Sub btnAdd_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
btnAdd.Click

  Dim str As String

  Dim ans As Integer

  str = "insert into stud values(" & TextBox4.Text & ",'" & TextBox5.Text & "','" & TextBox6.Text
& "'," & TextBox7.Text & ")"

  Dim cmd As New OleDbCommand(str, con)

  con.Open()
```

```
    ans = cmd.ExecuteNonQuery()

    con.Close()

    MsgBox("Record Inserted Successfully :-" & ans)

  End Sub

End Class
```

- **Example of ExecuteNonQuery for Update**

```
Imports System.Data.OleDb

Partial Class methds_cmdobj

Inherits System.Web.UI.Page

Dim con As New OleDbConnection("Provider=Microsoft.jet.OLEDB.4.0;Data Source=
D:\asp.net\practicals\connectivity\App_Data\my_database.mdb")

Protected Sub Button5_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Button5.Click

    Dim str As String

    Dim ans As Integer

    str = "update stud set name='" & TextBox5.Text & "', city= '" & TextBox6.Text & "', mark= " &
TextBox7.Text & " where rno= " &TextBox4.Text

    Dim cmd As New OleDbCommand(str, con)

    con.Open()

    ans = cmd.ExecuteNonQuery()

    con.Close()

    MsgBox(ans & ": Record Successfully Updated")

End Sub

End Class
```

3. ExecuteReader:-

- When we execute the command object using ExecuteReader method that time it returns DataReader
- DataReader is used to retrieve a read-only, forward-only stream of data from a Database.
- We can not perform any insert, update and delete operations in the DataReader.
- DataReader is opposite of DataSet.
- We can not move previous in the DataReader.
- DataReader requires open connection.
- DataReader can hold one table at a time.
- DataReader can increase application performance by retrieving data as soon as it is available and (by default) storing only one row at a time in memory, reducing system overhead.
- It has item collection.
- For example :- select rno, name, city from student then rno is Item(0), name is Item(1) and city is Item(2).
- DataReader have read method to read next records.
- DataReader is set by command object's ExecuteReader method.
- Example:- display data using DataReader in two dropdown list. (select RollNo and display name according to it).

```
Imports System.Data.OleDb

Partial Class methds_cmdobj

Inherits System.Web.UI.Page

Dim con As New OleDbConnection("Provider=Microsoft.jet.OLEDB.4.0;Data Source=D:\asp.net\practicals\connectivity\App_Data\my_database.mdb")

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load

 If Not Page.IsPostBack Then

 Dim str As String

str = "select rno,name from stud"

Dim cmd As New OleDbCommand(str, con)

con.Open()

Dim dr As OleDbDataReader
```

```
dr = cmd.ExecuteReader

While dr.Read

DropDownList1.Items.Add(dr.Item(0))

DropDownList2.Items.Add(dr.Item(1))

End While

con.Close()

End If

End Sub

Protected Sub DropDownList1_SelectedIndexChanged(ByVal sender As Object, ByVal e As
System.EventArgs)Handles DropDownList1.SelectedIndexChanged

  DropDownList2.SelectedIndex = DropDownList1.SelectedIndex

End Sub

Protected Sub DropDownList2_SelectedIndexChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles DropDownList2.SelectedIndexChanged

    DropDownList1.SelectedIndex = DropDownList2.SelectedIndex

End Sub

End Class
```

❖ **Differences Between DataReader and DataSet:**

DataReader	DataSet
It is directly connected to database system.	It is generally used to disconnected architecture of Ado.Net
It can hold one table at a time.	It can hold multiple tables at a time. It represents a complete set of data including related tables, constraints and relationships among the tables.
Use multiple user updated data every time.	If multiple users are using the database and the database needs to be updated every time, you must not use the DataSet.
It provides Read only data. We cannot update records.	We can make changes in the DataSet.
No local storage is required.	It reads data from database and stores in local system.
Improve application performance.	DataSet requires lot of memory space compare to DataReader.
It holds one row at a time is stored in memory.	It holds multiple records at a time.
No relationships can be maintained.	Relationships can be maintained.

❖ **DataAdapter object** :-

It is associated Command and Connection objects.

- The DataAdapter is a class at the core of ADO.NET's disconnected data access.
- It is a bridge between the Database and DataSet.
- The DataAdapter provides a set of methods and properties to retrieve and save data between a DataSet and Database.
- The DataAdapter is used fill method to fill a DataSet.
- The DataAdapter can commit the changes to the Database by it's **Update** method.
- The DataAdapter provides four properties that represent Database Commands:

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

- When the DataAdapter fills a DataSet, It will create the necessary tables and columns for the returned data.
- There are two ways to initialize it:

```
Dim da As New OleDbDataAdapter(cmd)
```

OR

```
Dim da As New OleDbDataAdapter

da.SelectCommand = cmd
```

❖ **DataSet :**

ADO.NET caches data locally on the client and store that data into DataSet.

- The DataSet is a disconnected, in-memory representation of data. It's not exact copy the Database.
- It can be considered as a local copy of some portions of the Database.
- The DataSet contains a collection of one or more DataTable objects made up of rows and columns of data.
- Tables can be identified in DataSet using DataSet's **Tables property.**
- It also contains primary key, foreign key, constraint and relation information about the data in the DataTable objects.
- Whatever operations are made by the user it is stored temporary in the DataSet, when the use of this DataSet is finished, changes can be made back to the central database for updating.
- DataSet doesn't know where the data it contains came from, and in fact it can contain data from multiple sources.
- Remember that The DataSet is not connected to a data source after it is populated.
- Inside a DataSet, like in a Database, there are tables, columns, relationships, constraints, view, and so on...
- The DataSet is populated DataAdapter's **Fill** method.

Ex:-

```
Dim ds As New Data.DataSet

da.Fill(ds)
```

- **Example:- Display Data in the GridView using DataSet.**

First of all we have to put one **GridView** control on a web page and then write the following code.

```
Imports System.Data.OleDb
```

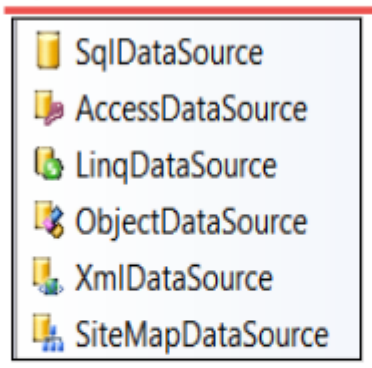
```
Partial Class _Default

Inherits System.Web.UI.Page

Protected Sub Page_Load(ByVal sender As Object, ByVal e  As System.EventArgs) Handles
Me.Load

Dim con As New OleDbConnection("Provider= Microsoft.jet.OLEDB.4.0; Data Source=
D:\asp.net\practicals\connectivity\App_Data\my_database.mdb")

    Dim cmd As New OleDbCommand("select * from stud",con)

    Dim da As New OleDbDataAdapter(cmd)

    Dim ds As New Data.DataSet

    da.Fill(ds)

    GridView1.DataSource = ds

    GridView1.DataBind()

End Sub

End Class
```

❖ **Data Controls**

- Data control enable you to connect a web page to various sources of data, which includes databases and XML files.
- Data controls also let you display data on the page in tables or in other formats, and enable users to edit data.

❖ **Data Source Controls :-**

- we always use the **AccessDataSource** control in conjunction with a data-bound control to retrieve data from a relational database and to display, edit and sort data on a web page with little or no code.
- The **AccessDataSource** can support any relational database that can be connected to using an ADO.NET provider, such as the SqlClient, Oledb, OracleClient.



- These Data Source fall within one of two categories:
  Represent Tabular Data
    - SQLDataSource
    - AccessDataSource
    - ObjectDataSource

  Represent Tabular and hierarchical Data
    - XMLDataSource
    - SiteMapDataSource

- Databound controls are associated with one of these datasources with its DataSourceId property.

❖ **DataBound Controls :-**

- There are three main DataBound controls.

List Control:

- BulletedList
- CheckboxList
- DropDownList
- Listbox
- RadioButtonList

Hierarchical databound Controls:

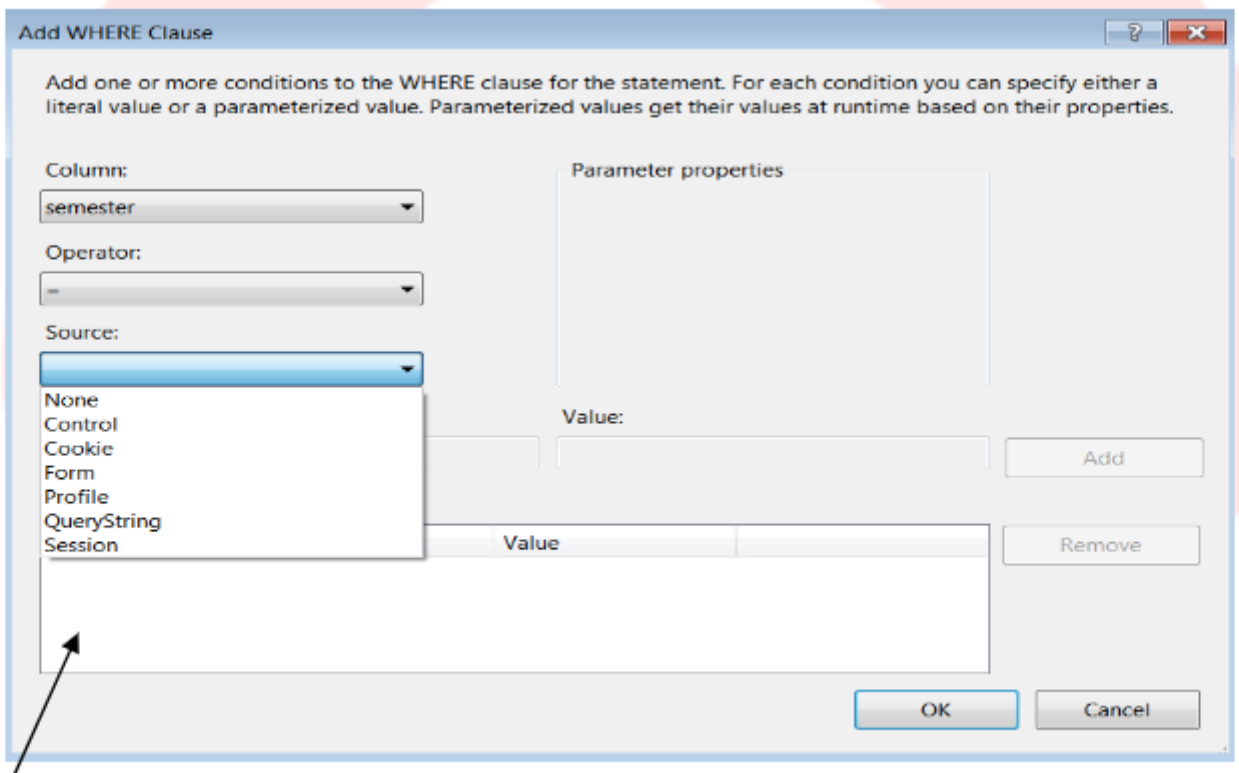
- Menu
- TreeView

Tabular databound Controls:

- Display a set of data
  - GridView
  - DataList
  - Repeater

- Display a single data item at a time
  - DetailsView

❖ **Parameters Options in DataSource Controls :-**



Parameters

Source	Meaning
None	Static Value.
ControlParameter	Value of a control or page property.
CookieParameter	Value of a browser cookie.
FormParameter	Value of an HTML form field.
ProfileParameter	Value of a Profile property.

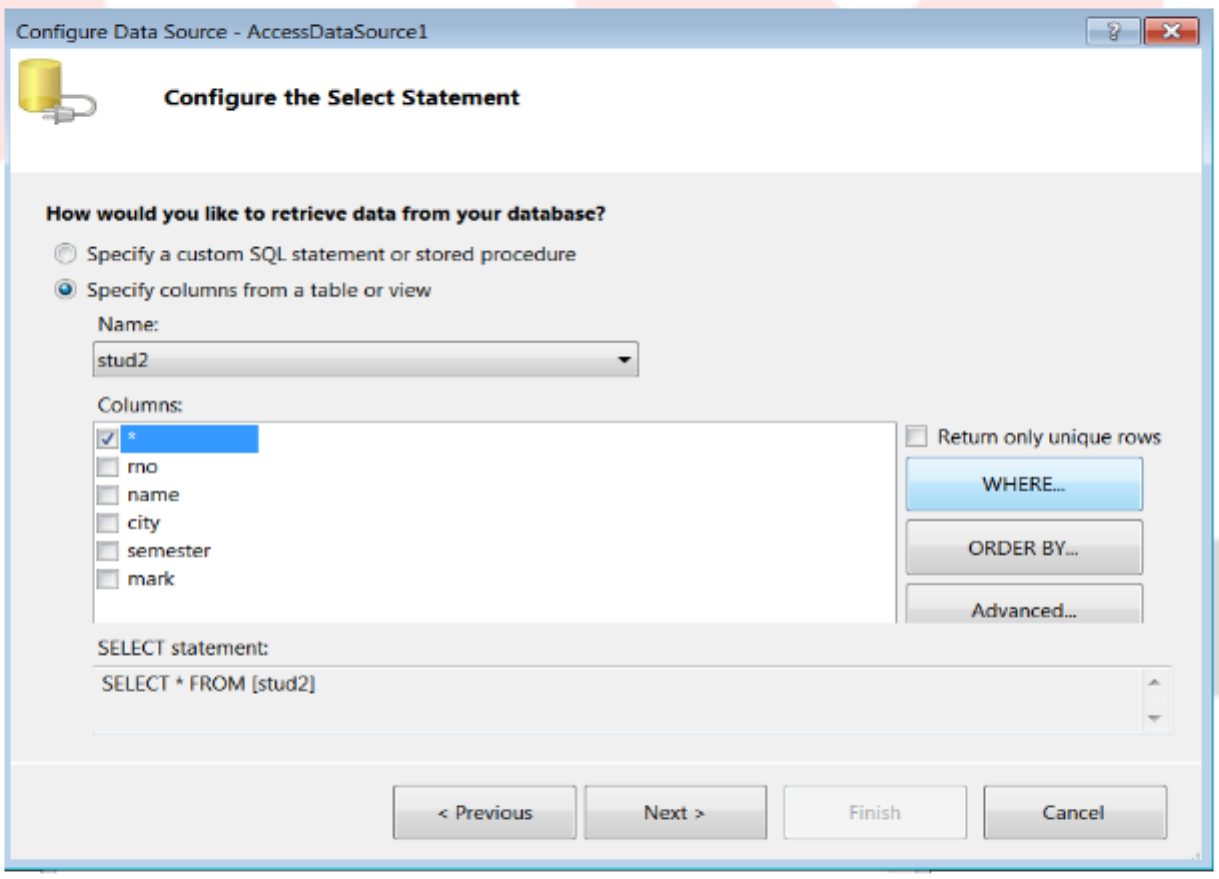
QueryStringParameter	Value of a query string field.
SessionParameter	Value of an item stored in a session.

Adding a WHERE Clause

- In the previous example we saw how to use the **AccessDataSource** control to retrieve data directly from a database. using the configure Data Source wizard, we could choose the database and then select the columns to return from a table; enter a custom SQL statement.
- Whether selecting columns from a table or entering a custom SQL statement, the **AccessDataSource** control's **SelectCommand** property is assigned the resulting ad-hoc SQL SELECT statement and it is this SELECT statement that is executed when the **Select( )** method of **AccessDataSource** is invoked.
- Now, create a Parameterized Query:

Ex:- (1) Example of Parameter passing in AccessDataSource. Enter semester and display details of the students.

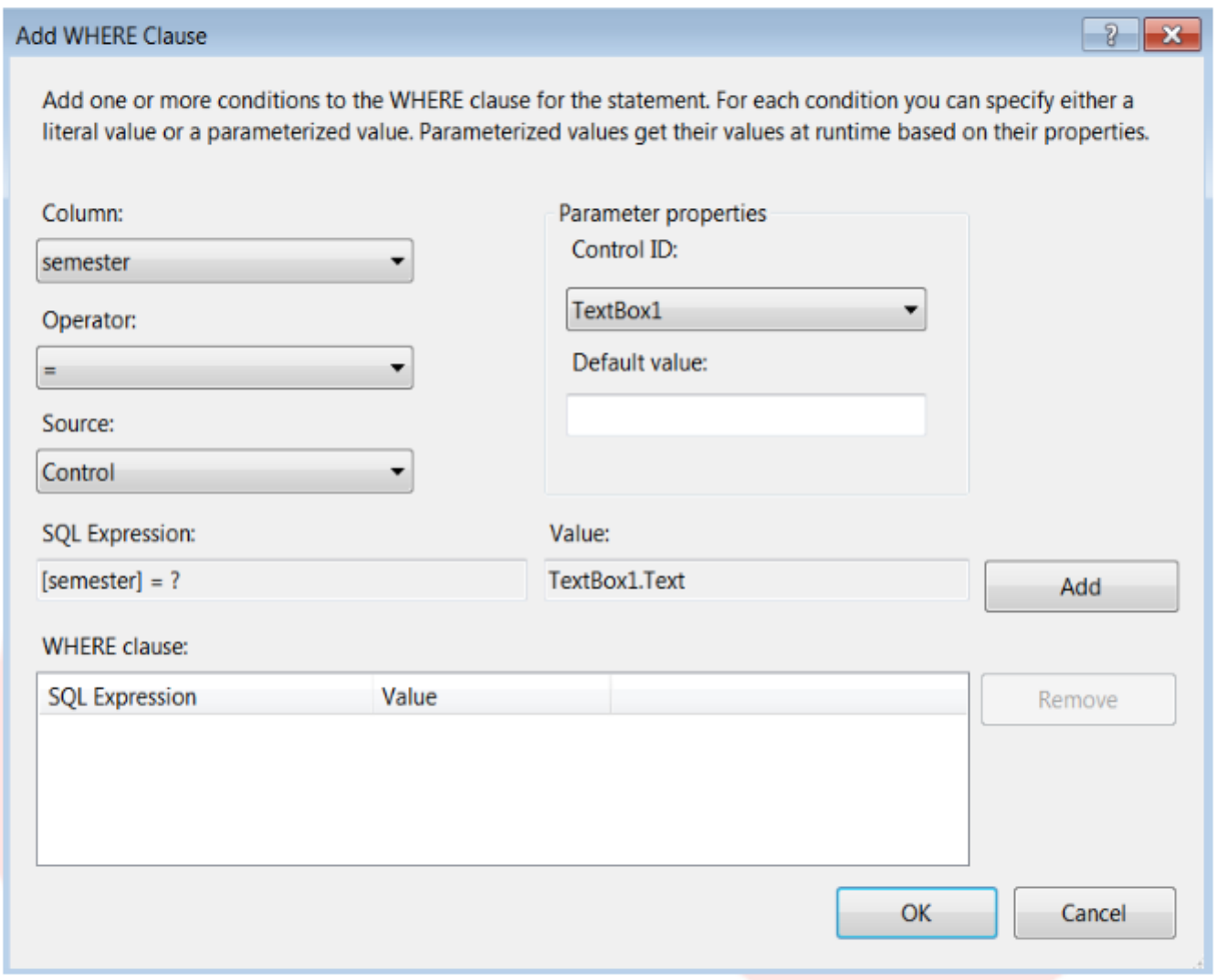
- Click the WHERE button, which brings up the Add WHERE Clause dialog box.

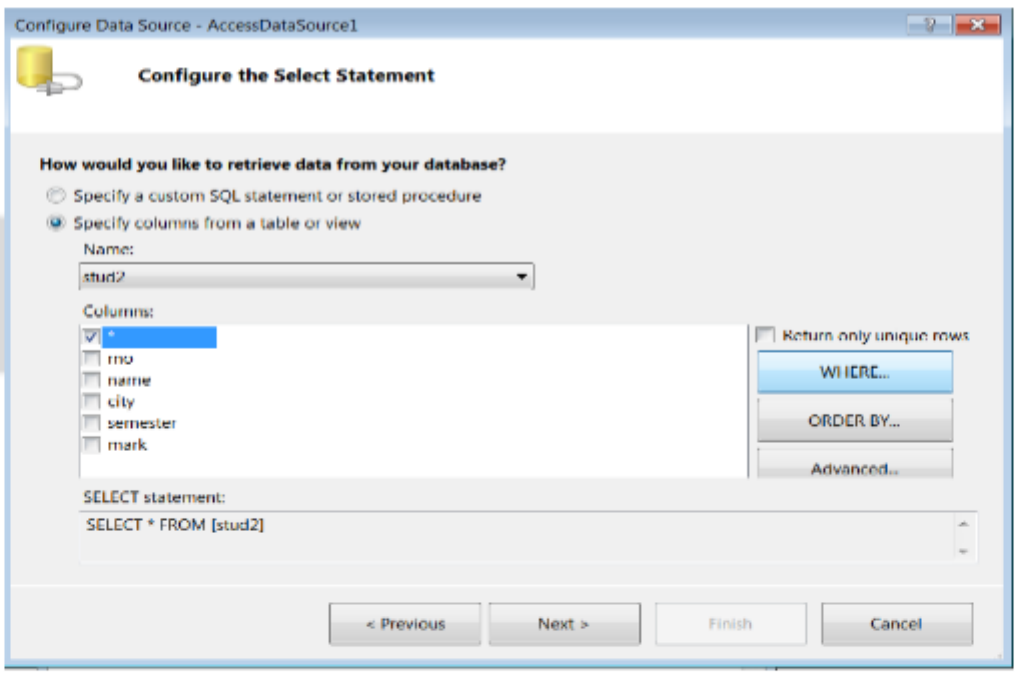


- Now, Add Parameters:

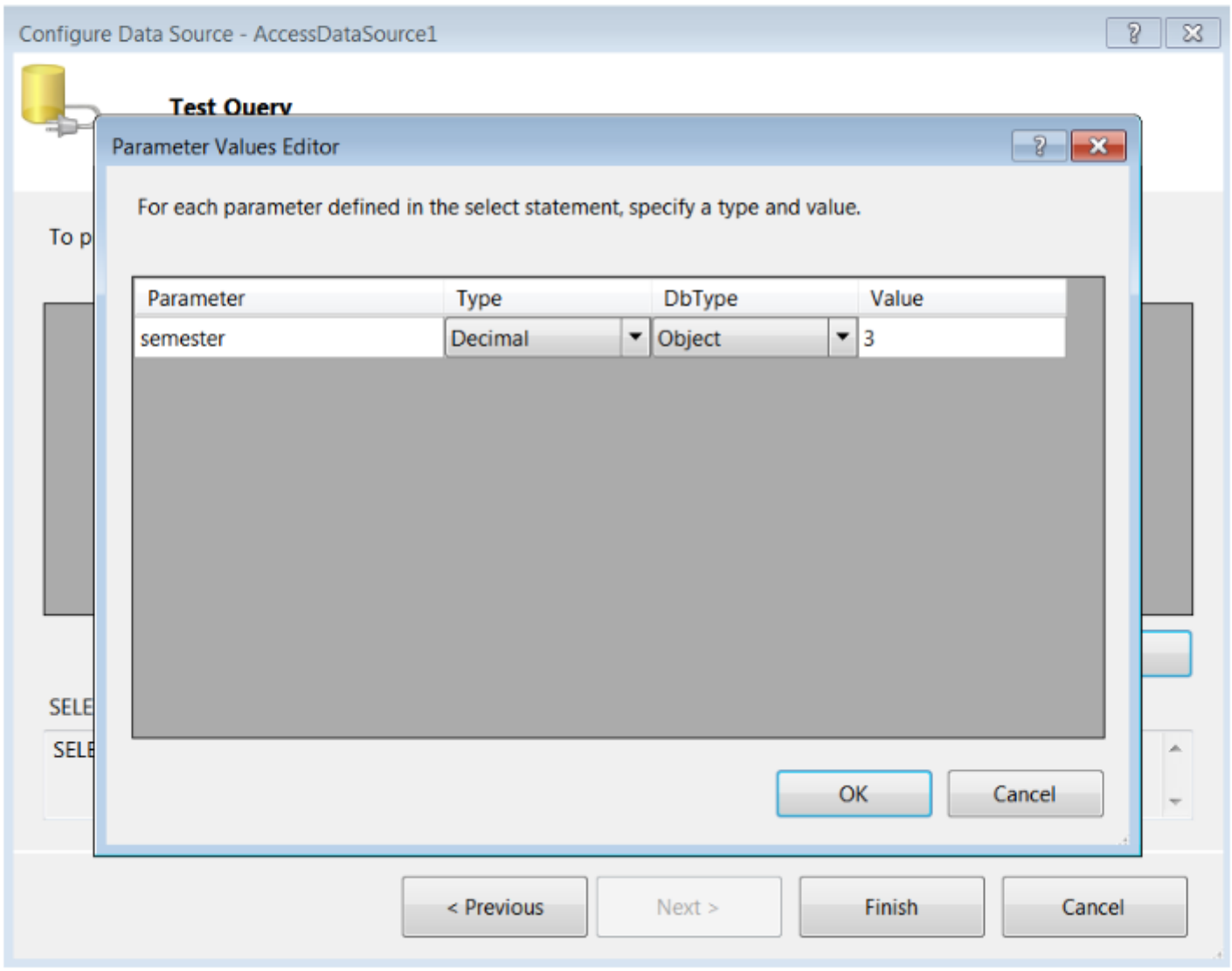
Select "semester" for Column, "=" for Operator, "Control" for Source and "TextBox1" for Control Id.

- Now, Click on Add button and then Click on OK button, which brings back to the Configure DataSource dialog box.

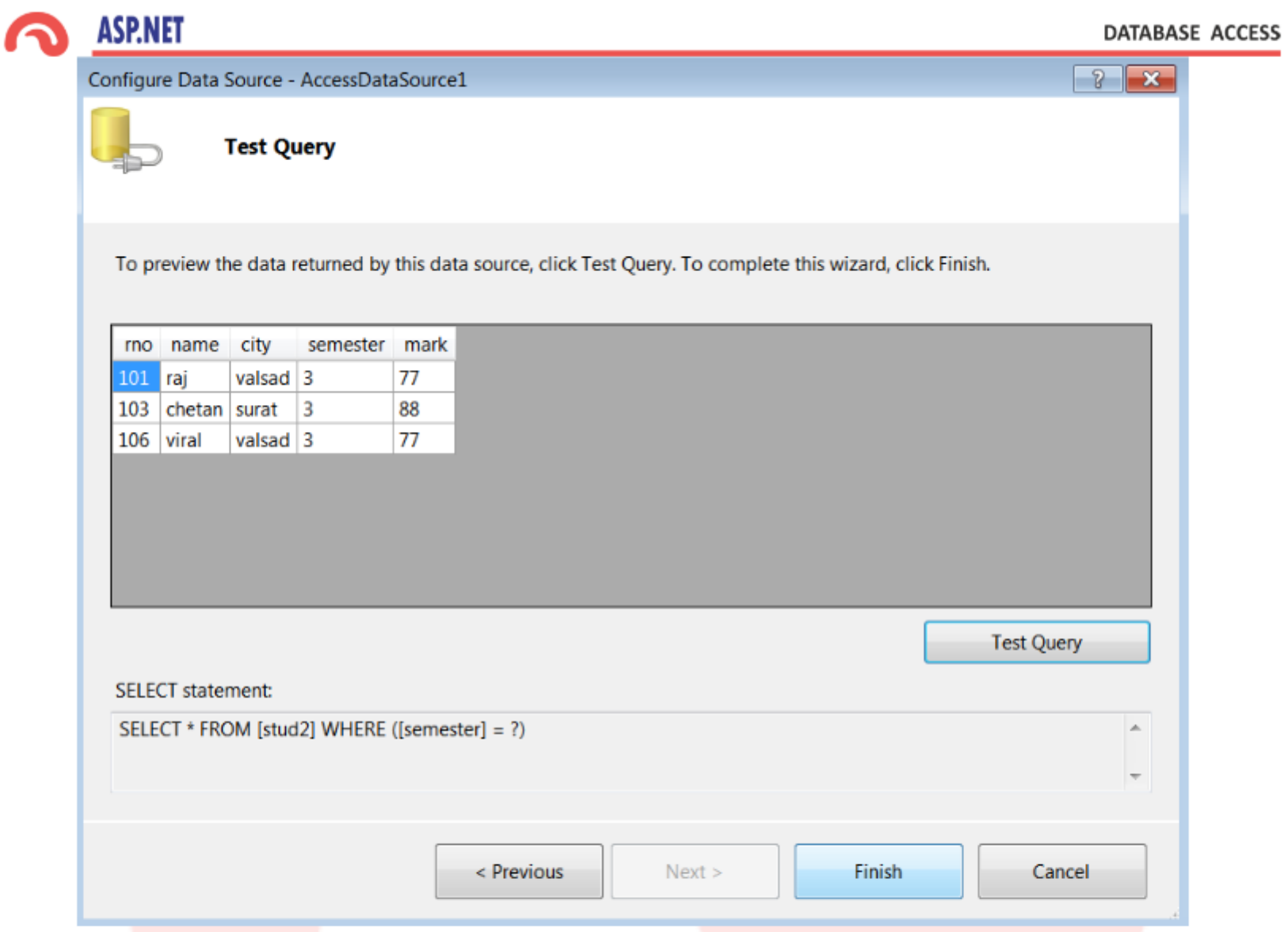




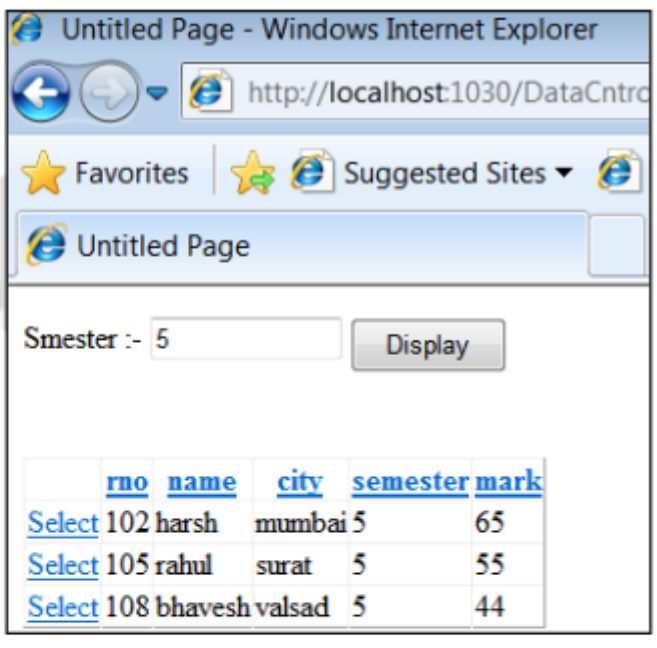
- Now, Click on Next button and then Click on **Test Query** button. It will display the **Parameter Values Editor.**
- In the **Parameter Values Editor** we will specify type and value. Here we choose Decimal as **Type** and enter value 3 for **value** field.
- Click on 'OK' button.



- You can see the data returned by the DataSource.

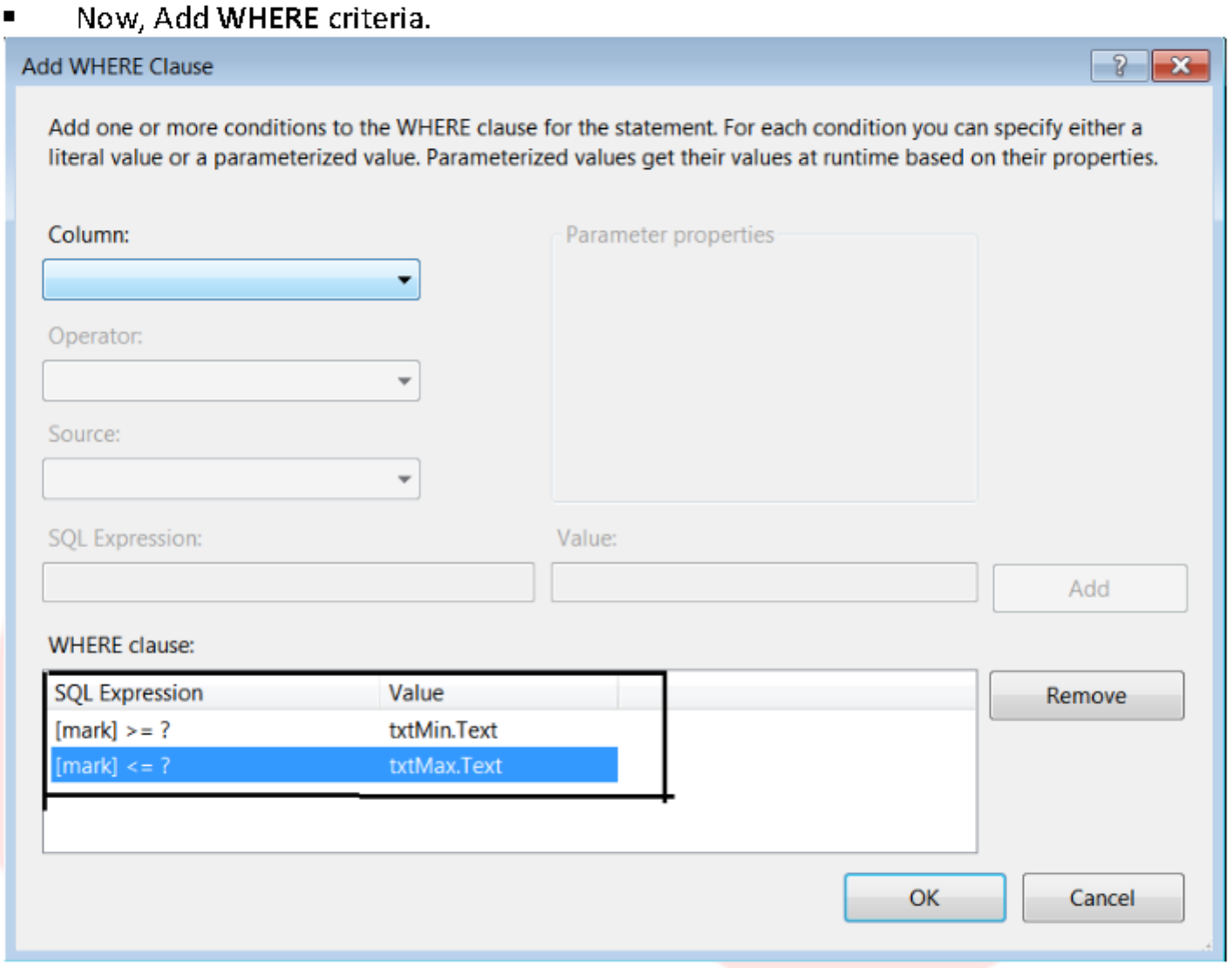


- To complete this wizard, Click 'Finish'.
- Now. you can Run your application.
- If you enter the value-**5** in the TextBox1, then display the details of all the students who are in semester-5.

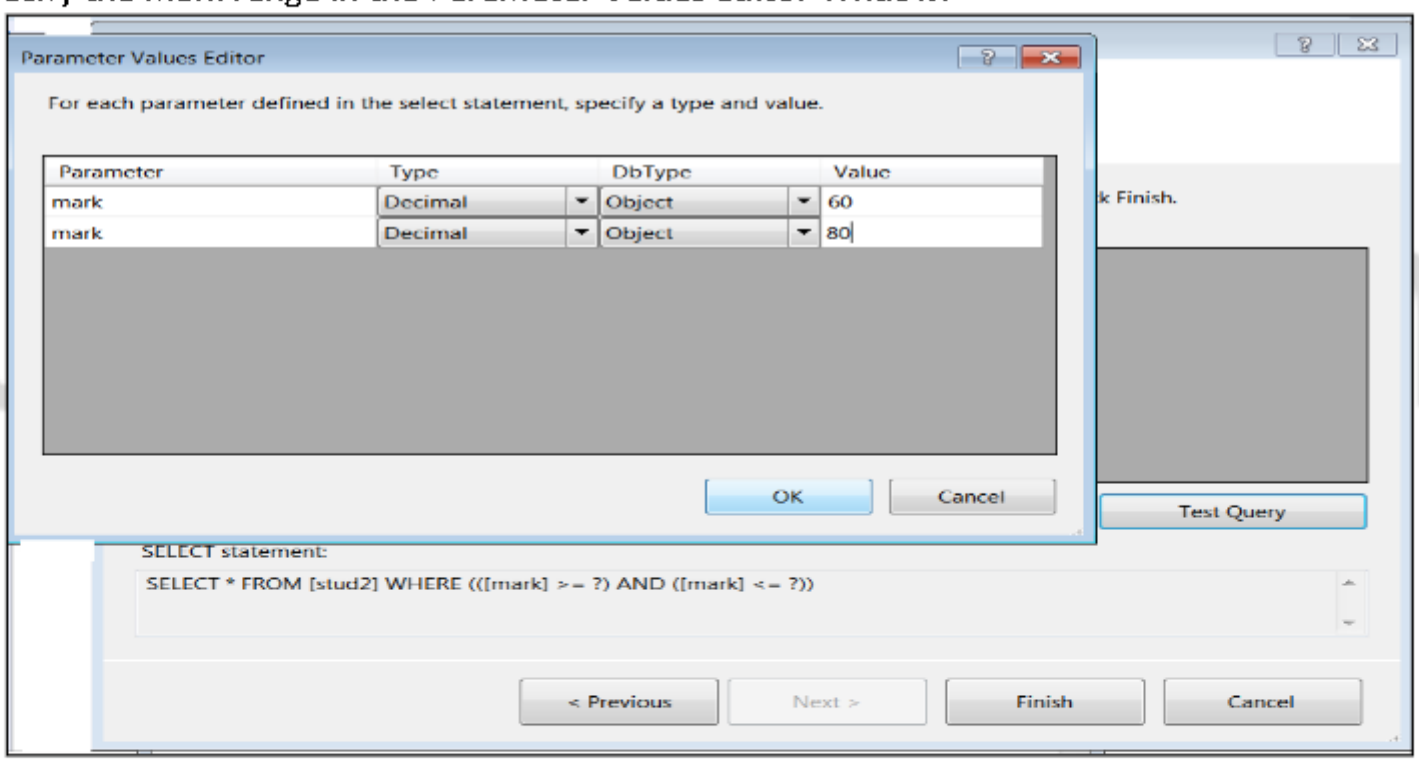


Ex:- (2) Display students details whose mark is in selected Range.

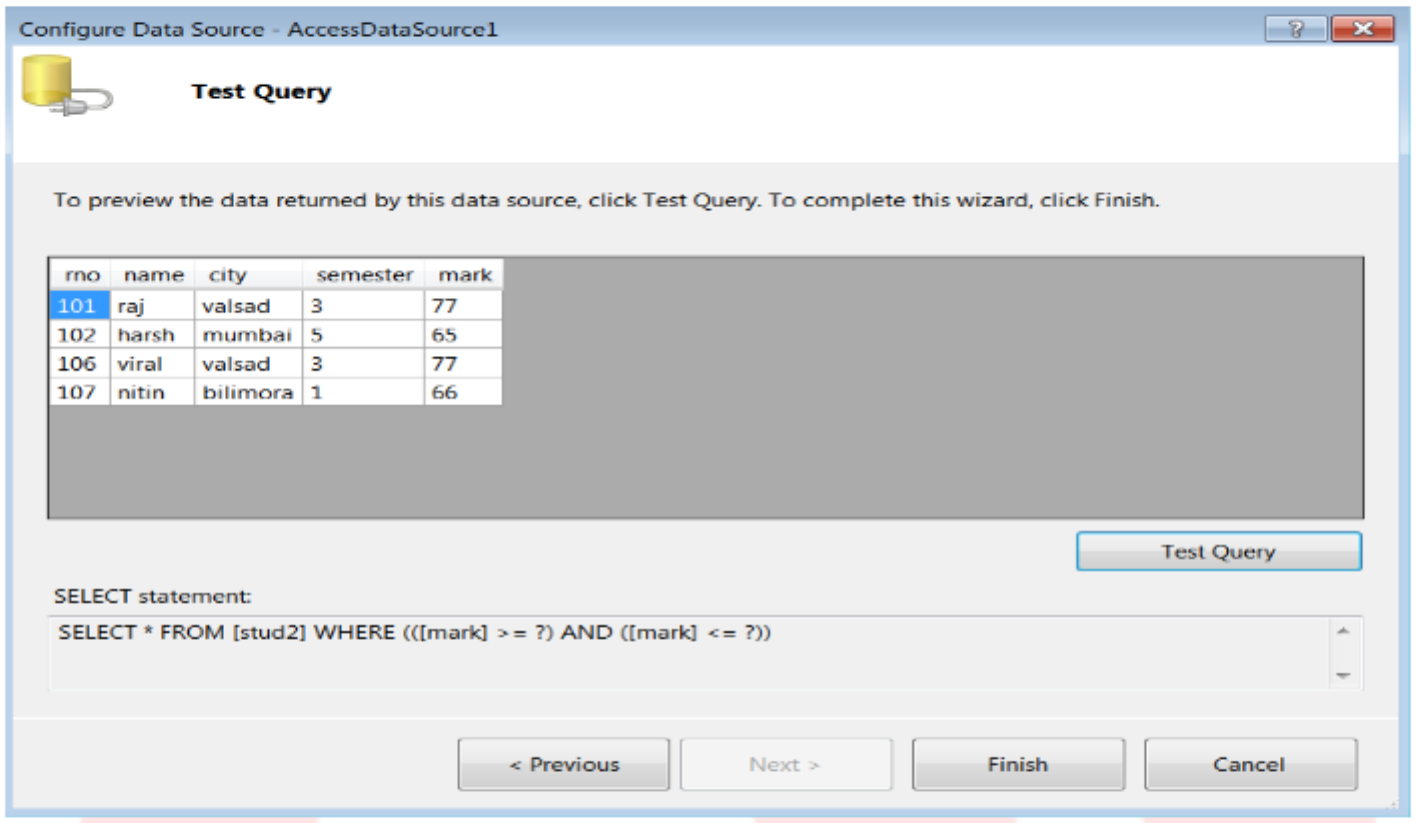
- First of all put Two Label, Two TextBoxes, One Button, AccessDataSource control and GridView control on the web page.
- Configure Data Source. Select table and columns to be retrieve.
- Now, Add **WHERE** criteria.



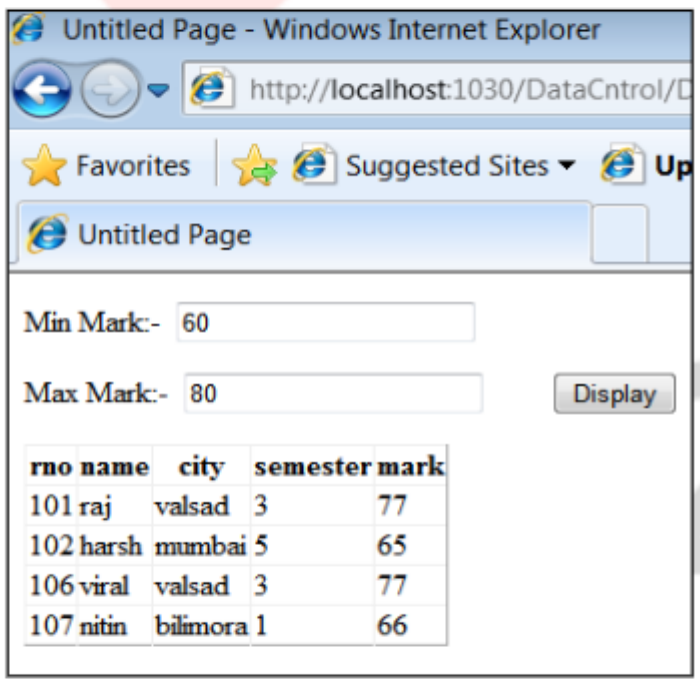
- Specify the mark range in the Parameter Values Editor Wndow.



- You can see the data returned by the DataSource.



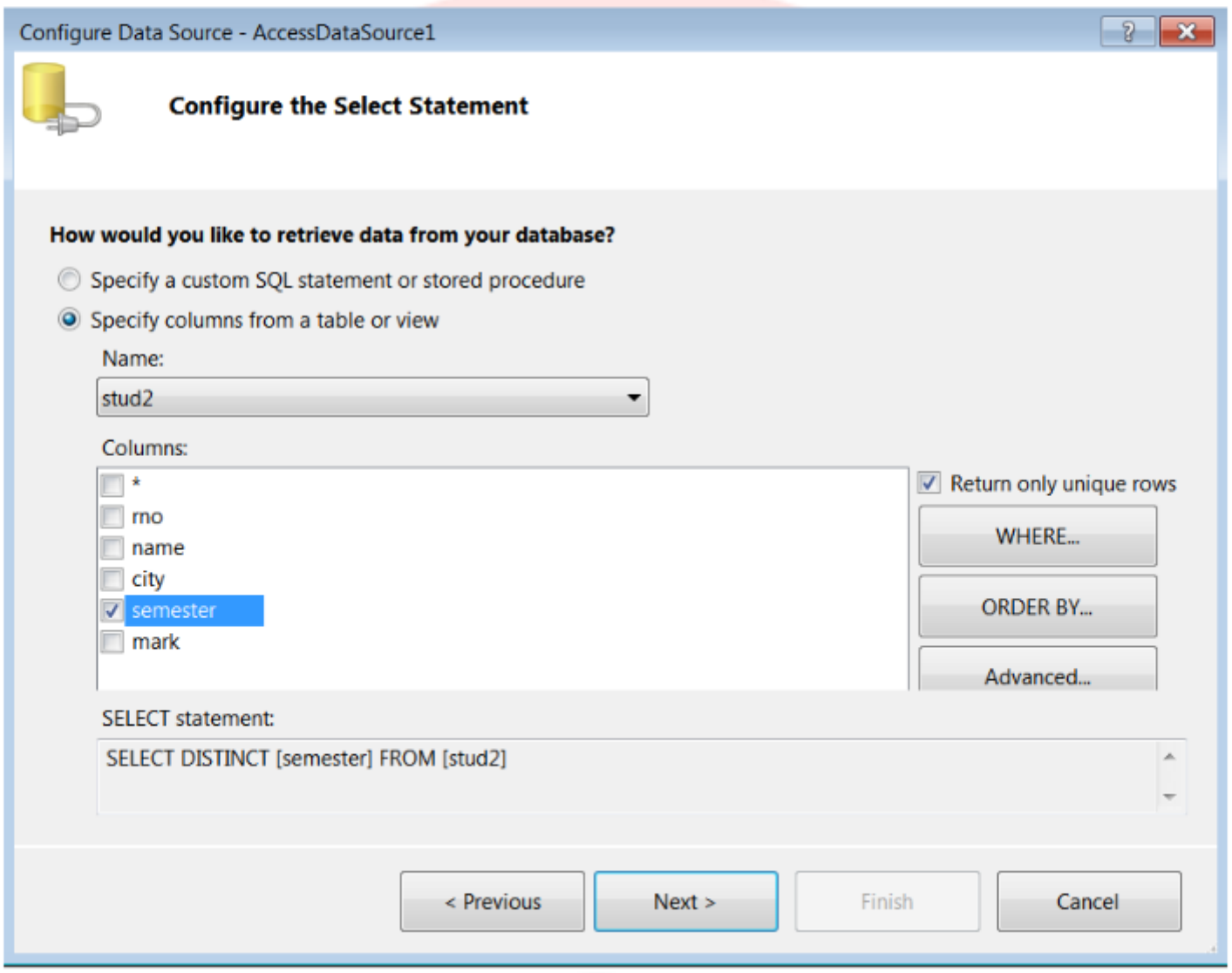
- To complete this wizard, Click 'Finish'.
- Now. you can Run your application.



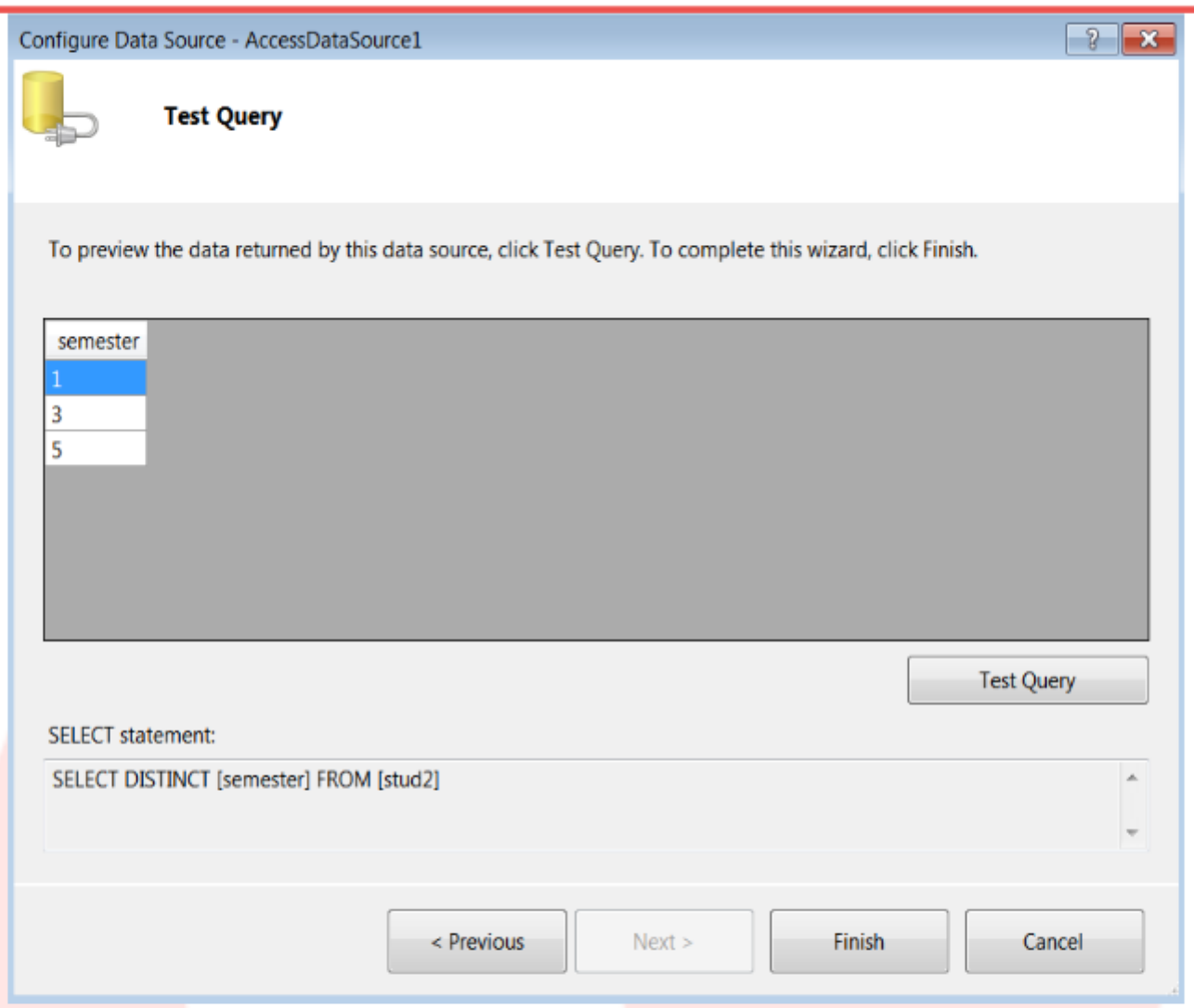
Note:- Remember that to display data in the **GridView** we have to Bind it with the **AccessDataSource** by selecting **AccessDataSource1** from **Choose Data Source.**

Ex:- (2) Using session pass the Parameter.

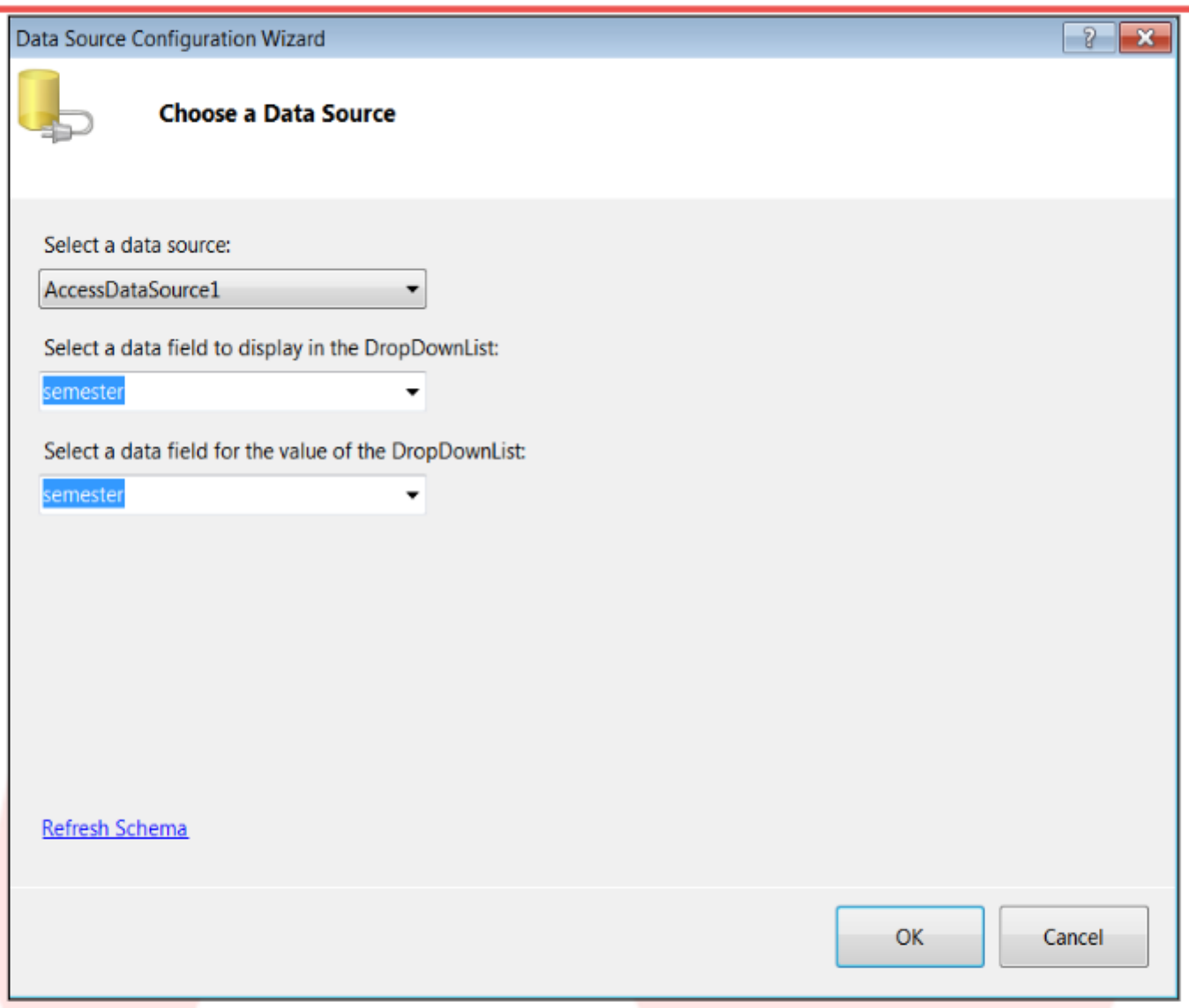
- Create two web pages named page1.aspx and page2.aspx.
- Put the **AccessDataSource, DropdownList and a Display Button** on the page1.
- Now, configure the Data Source ----> choose the Database. Click on Next.
- Select **semester** column and also tick **Return only unique rows.**
- Click on Next button.



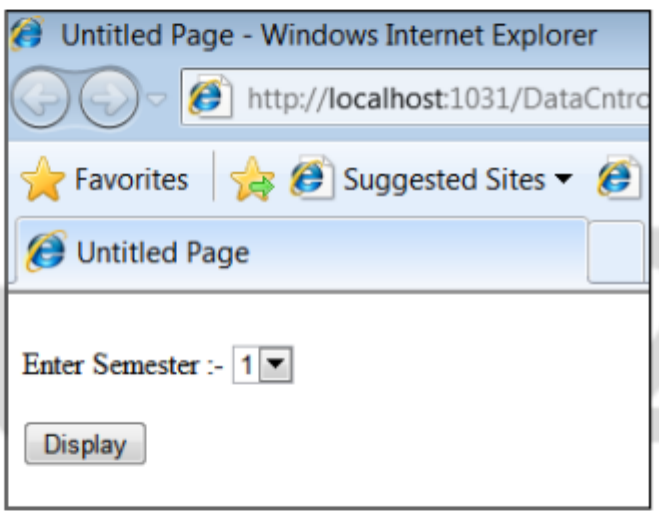
- Click on **Test Query** button so that following dialog box will display, which contain the data returned by the Datasource.



- To complete this wizard, Click Finish.
- Attach **DropdownList1** with **AccessDataSource** control.
- Select **Choose Data Source** from the DropdownList smart tag.
- Select a Data Source and Data Field and then Click on OK button.



- **Output View of page1.aspx.**

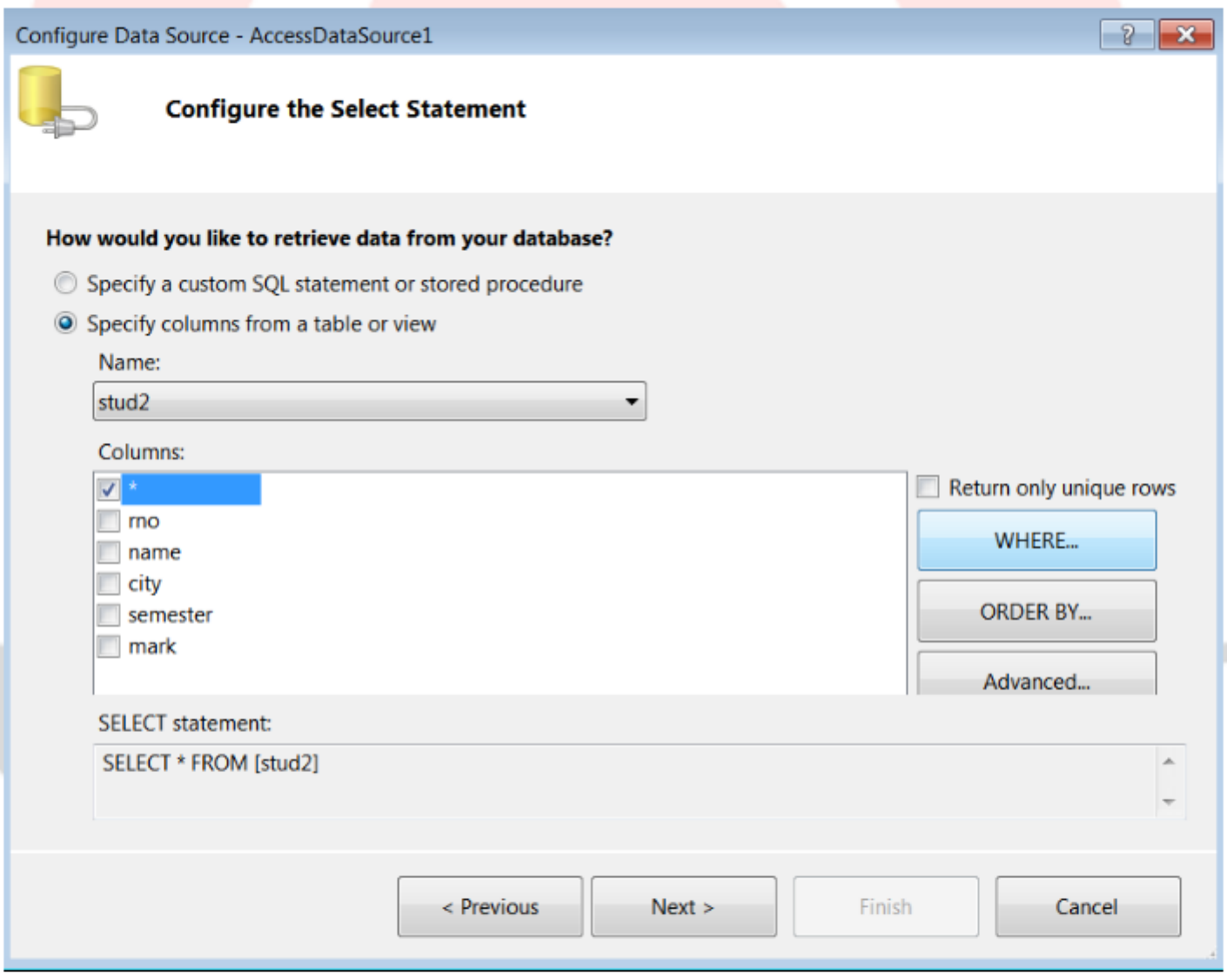


- **Coding View of page1.**

```
Partial Class page1

Inherits System.Web.UI.Page
```

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


        Session("semester") = DropDownList1.Text

        Response.Redirect("page2.aspx")

 End Sub

End Class
```

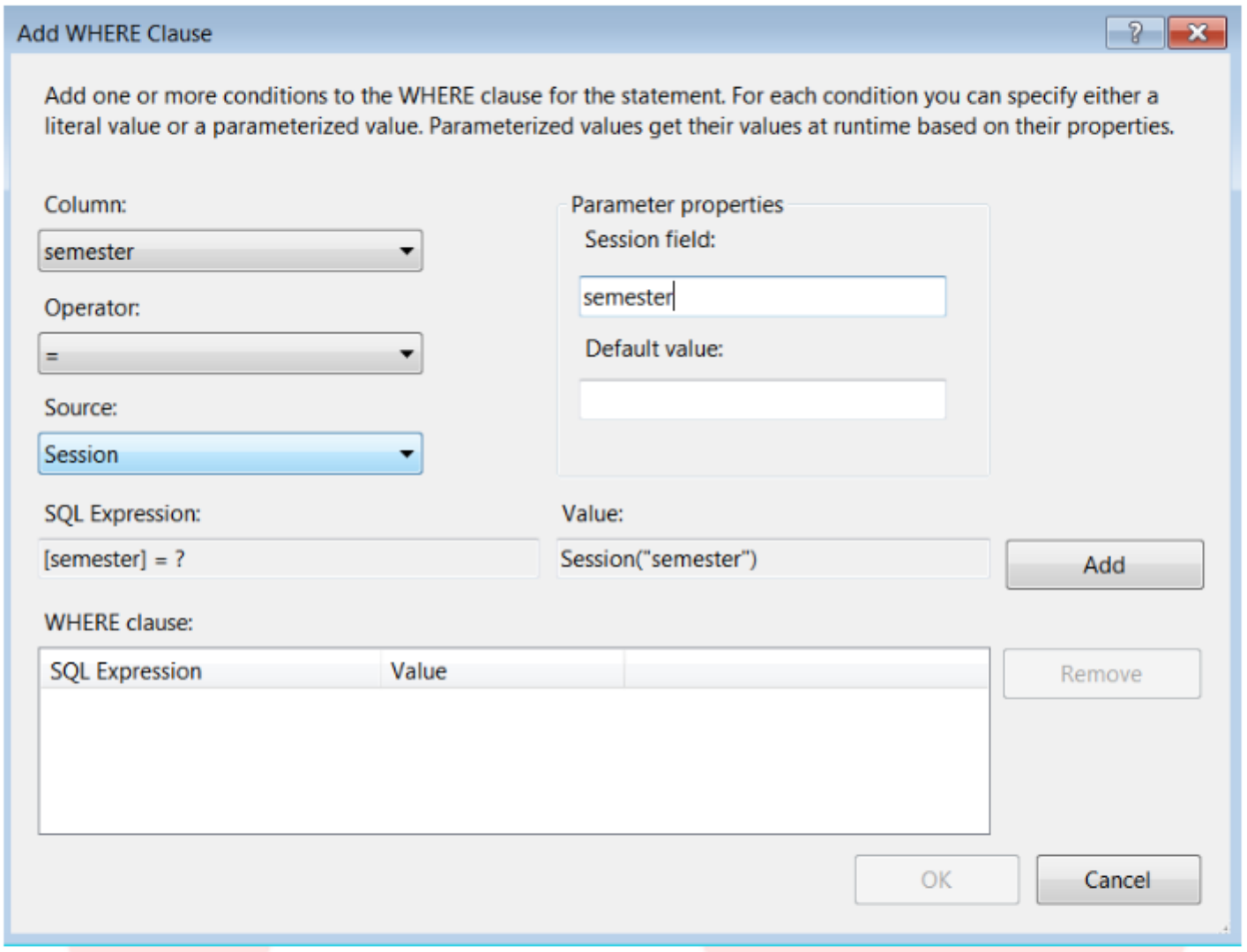
- Now, Put the **AccessDataSource** and **GridView** control on the page2.
- Now, configure AccessDataSource, choose the database and Click on Next button.
- Choose table and columns and then Click on WHERE button to Add where criteria. (here we select * for all columns)



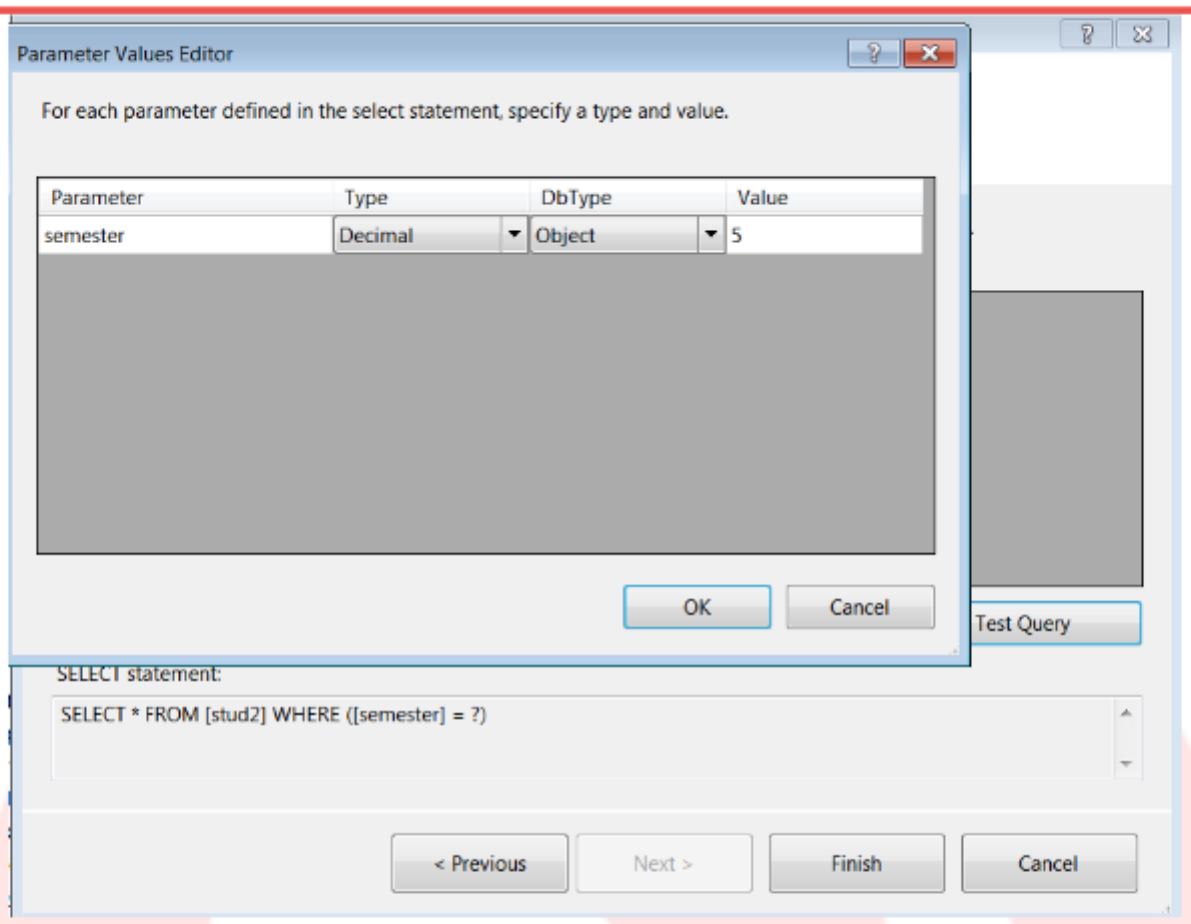
- Now, select column:, Operator and Source. Give the name for the session field.

Here, we select "semester" for column, "=" for Operator and "Session" for Source. We also specify the session field as "semester"

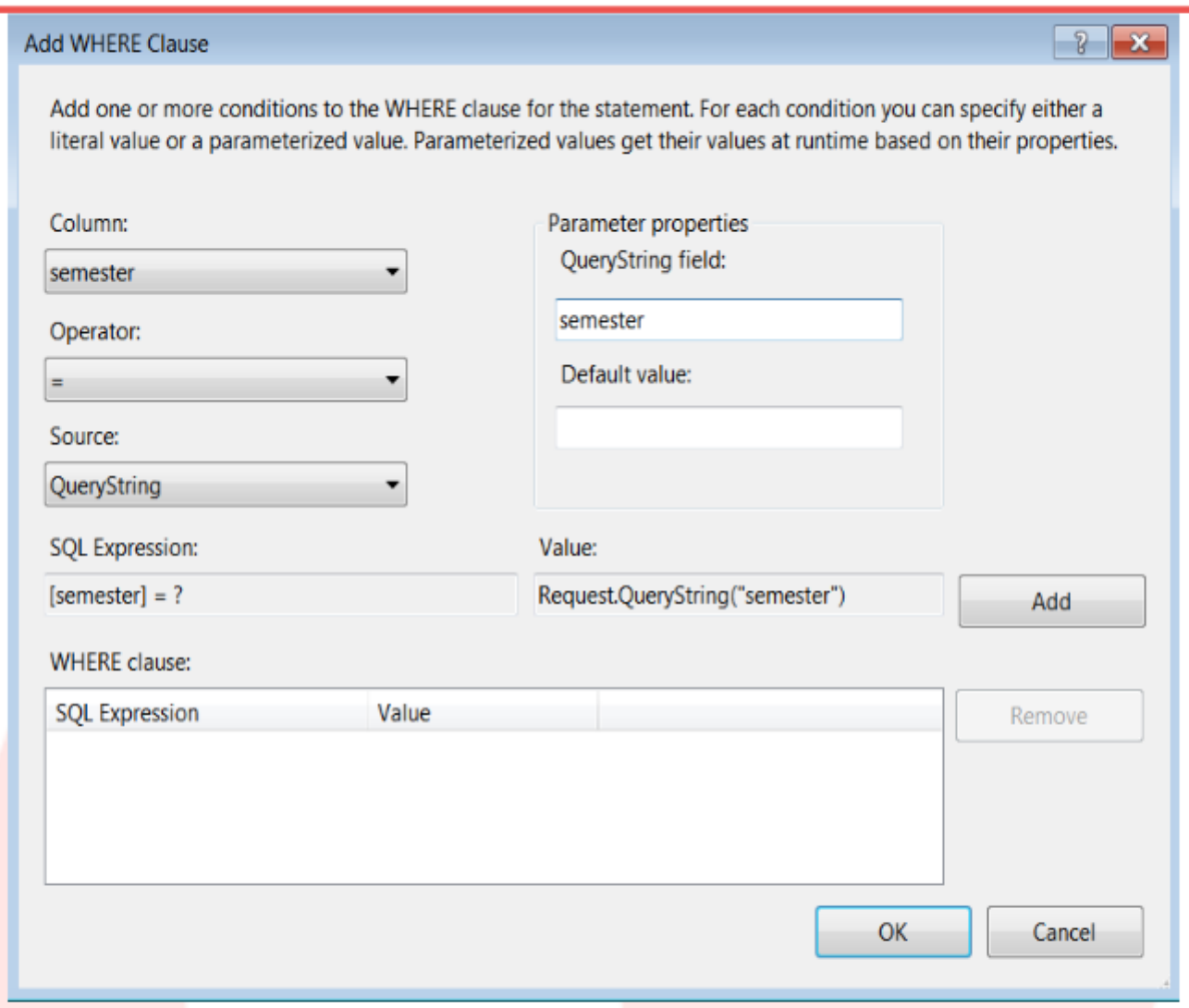
- Click on Add button and then Click on Ok button.



- Click on Next button and then click on **Test Query** button.
- Specify the Type and Value in the Parameter Values Editor and then click on OK.



- Now, Click on Finish to complete this wizard.
- Don't forget to choose Data Source for GridView.
- Same as above we can use Query String.



❖ **GridView Control :-**

- It is the only DataBound control which has inbuilt edit, update, delete, sort, paging data functionality. We can enable this functionality by setting the properties of GridView without even writing a single line of code.
- It provides more flexibility in displaying and working with data from your database in comparison with any other controls.
- The GridView control enables you to connect to a Datasource and display data in a tabular format.
- Using this control is very easy when compared to other databound controls to display data. You have to just assign the DataSource property to a Data Table and calling **DataBind( )** method.
- The main **disadvantage of GridView control is, it does not have the flexibility to display data in any other format except table.**

Some Properties:

- DataSource:- Gets or sets the data source object that contains the data to populate the control.
- AllowPaging:- true/false. Indicate whether the control should support paging.

- Allowsorting:- true/false. Indicate whether the control should support Sorting.
- AutoGenerateEditButton:- true/false. Indicates whether a separate column should be added to edit the record.
- AutoGenerateDeleteButton:- true/false. Indicates whether a separate column should be added to Delete the record.
- AutoGenerateSelectButton:- true/false. Indicates whether a separate column should be added to Select the particular record.
- AlternatingRowStyle:- Define the style properties for every alternate row in the GridView.
- Caption :- Gets or sets the caption of the GridView.
- PageIndex :- Gets or sets the number of records to display in one page of GridView.

Event:

- PageIndexChanging and PageIndexChanged :- Both events occur when the page link is clicked.

❖ RESPONSE OBJECT

The Response object represents a valid HTTP response that was received from the server. The response header properties are read-only.

PROPERTIES OF THE OBJECT ARE AS FOLLOWS:

NAME	DESCRIPTION
Body:	Gets the body of the HTTP response. Only the portion of the body stored in the response buffer is returned
BytesRecv:	Gets the number of bytes the client received in the response
BytesSent:	Gets the number of bytes send in the HTTP request
CodePage:	Gets or sets the code page used for setting the body of the HTTP response
ContentLength:	Gets the size, in bytes, of the response body
Headers:	Gets a collection of headers in the response
HeaderSize:	Gets the combined size, in bytes, of all the response headers
HTTPVersion:	Gets the HTTP version used by the server for this response
Path:	Gets the path that was requested
Port:	Gets the server port used for the request
ResultCode:	Gets the server's response status code
Server:	Gets the name of the server that sent the response
TTFB:	Gets the number of milliseconds that have passed before the first byte of the response was received
TTLB:	Gets the number of milliseconds that passed before the last byte of the response was received
UseSSL:	Checks whether the server and client used an SSL connection for the request and response

SAMPLE CODE

The following sample code assumes that you have a Button control and four RadioButtoin controls with a common GroupName property on a Web Form. The user will be sent to a new Web Page if he makes a selection from one of the radio button's and hits the Submit button.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e
As_System.EventArgs) Handles MyBase.Load

Dim ccc As String = "Response Object"

Response.Write(ccc)
Response.Write("Using Response object")

'using the response object's write method to write some text
```

```
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As_System.EventArgs) Handles Button1.Click
If RadioButton1.Checked = True Then
Response.Redirect("http://www.google.com")
'using the response object's redirect method to redirect the user to another web
page
ElseIf RadioButton2.Checked = True Then

Response.Redirect("http://www.amazon.com")
ElseIf RadioButton3.Checked = True Then

Response.Redirect("http://www.yahoo.com")
ElseIf RadioButton4.Checked = True Then

Response.Redirect("http://www.startvbdotnet.com")
End If

End Sub
```

The ASP.NET provides a class called ***HttpResponse*** which is defined in the namespace ***System.Web***. This class provides various methods and properties which help you use various information related to a web response.

An instance of this class is created by default in all the pages, so that you can use this object without creating again each time in all the pages. The name of this object is ***Response***.

FREQUENTLY USED METHODS AND PROPERTIES OF RESPONSE OBJECT:

RESPONSE.WRITE

This method is used to write dynamic text to the web page.

```
Response.Write(DateTime.Now.ToString())
```

The above code will generate the current time as text and display to the user. But note that the text will be displayed on the top of the page. There is no way you can specify the location and format the text.

If you want to move the location of the text, you may have to do something like below:

```
Response.Write("<table width=500>")

Response.Write("<tr>")
Response.Write("<td align=right>")

Response.Write("<BR>")
Response.Write("<BR>")
```

```
Response.Write("<BR>")
Response.Write(DateTime.Now.ToString())
Response.Write("</td>")
Response.Write("</tr>")
Response.Write("</table>")
```

ASP.NET provides several web controls including the Label control which allow you to specify the exact location where you want the control to be displayed. Due to this, the Response.Write() method is not widely used now.

RESPONSE.COOKIES

Cookies are used to store small pieces of information in the client computer. In ASP.NET, the Response object is used to send cookies to the client browser. If you select the 'Remember Me' option at the time of login, we use the following code to store your user id in a cookie in your computer. When you come back to this site later, we retrieve the user id from the Request and load your information automatically, without asking you to login again.

```
Response.Cookies("UserId").Value = "your user id"

Response.Cookies("UserId").Expires = DateTime.Now.AddDays(7)
```

We use the above code to store your user id in a cookie. The name of the cookie used is "UserId". The cookie will expire after 7 days. This means, if you come back to our site after 7 days, we will not remember you.

RESPONSE.REDIRECT()

This may be the most frequently used method of the **Response** object. **Response.Redirect()** is used to redirect user from one page to another page or website.

```
Response.Redirect("Login.aspx")
```

When the above line of code is executed, the page will be redirected to the login page.

❖ STATE MANAGEMENT

- Generally, web applications are based on stateless HTTP protocol which does not retain any information about user requests.
- In typical client and server communication using HTTP protocol, page is created each time the page is requested.
- Web pages are stateless and are HTTP-based, which means that it does not automatically indicate whether a sequence of request is all from the same client or a

new client. Every time a client request a page, a new instance of the page is returned and after each roundtrip the page is destroyed.

- State management is defined as the management of the state of one or more user interface controls such as textboxes, buttons, etc.. in a graphical user interface.
- In UI programming technique, the state of UI control depends on the state of other UI controls.
- Developer is forced to implement various state management technique when developing applications which provide customized content and which "remembers" the user.
- A new instance of the web page class is created each time the page is posted to the server.
- In traditional web programming, this would typically mean that all the information associated with the page and the controls on the page would be lost with each round trip.
- **Round trip:-** Whenever a user interaction require processing on the server, the web page is posted back to the server processed and is returned back to the browser. This sequence is called round trip.

- **THE PROCESSING CYCLE FOR AN ASP.NET WEB PAGE:**

1. The user request the page
2. The page dynamically renders markup to the browser, which the user sees as a web page similar to any other page.
3. The user types information or selects from available choices and then clicks a button.
4. The page is posted to the web server. Specifically, the page is posted back to itself.
5. on the web server, the page runs again. The information that the user typed or selected is available to the page.
6. The page performs the processing that you have programmed it to do.
7. The page renders itself back to the browser.

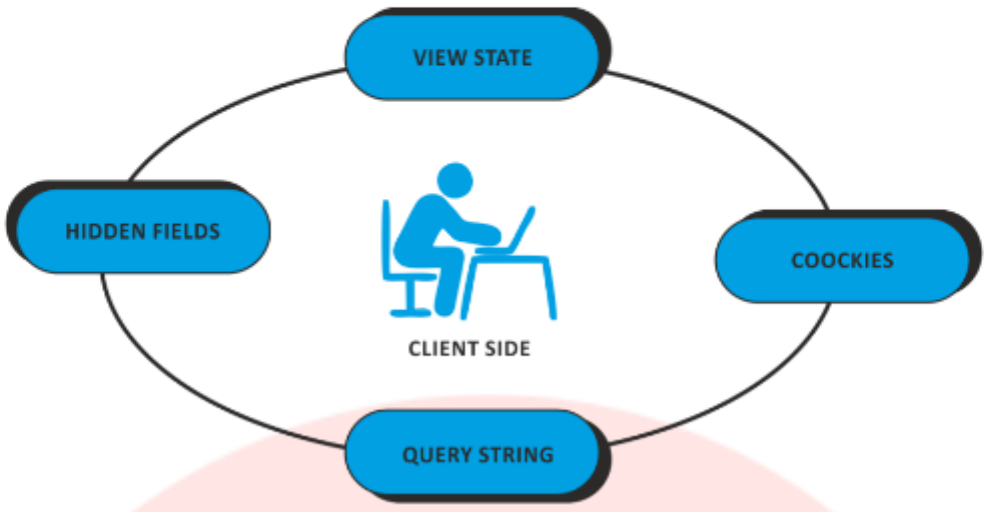
This cycle continues as long as the user is working in the page. Each time user clicks a button, the information in the page is posted to the web server and the page runs again.

Each cycle is referred to as a round trip.

STATE MANAGEMENT

- State management can be divided into two categories:
  1. Client-side management.
  2. Server-side management.

1. CLIENT SIDE MANAGEMENT:



- The following sections describe options for state management that involve storing information either in the page or on the client computer.
- For these options, no information is maintained on the server between round trips.

VIEWSTATE:-

- This is the default method that the page uses to preserve page and control property values between round trips.
- View State can be used to store state information for a single user.
- We can store page specific information.
- View state is a built in feature in web controls to persist data between page post backs.
- Each web page and controls on the page have the EnableViewState property. Using this property you can set View state on/off for each controls.
- By default, EnableViewState property will be set to true.
- ASP.NET framework uses the ViewState property to automatically
- save the values of the web page and each control on the web page prior to rendering the page.
- You can also disable View state for the entire page by adding EnableViewState=false to @page directive.
- The View State is implemented with a hidden form field called _VIEWSTATE, which is automatically created in every web page.
- When page is created on web server this hidden control is populated with state of the controls and when page is posted back to server this information is retrieved and assigned to controls.
- When the page is posted back to the server, the page parses the view-state string at page initialization and restores property information in the page.
- View state mechanism poses performance overhead.
- You can store values in view state as well.

HIDDEN FIELD:-

- Hidden fields are used to store data at the page level.
- As its name says, these fields are not rendered by the browser.
- A hidden field stores a single variable in its value property and must be explicitly added to the page.
- It's just like a standard control for which you can set its properties.
- Whenever a page is submitted to server, hidden fields values are also posted to server along with other controls on the page.
- All the asp.net web controls have built in view state property for state of the control, hidden fields functionality seems to be redundant.
- We can still use it to store insignificant data.
- Do not store any information in a hidden field that is sensitive.
- HiddenField have less number of properties compare to other web server control.

COOKIES:-

THERE ARE TWO TYPES OF COOKIES PERSISTENT AND NON PERSISTENT COOKIES

- persistent cookies are permanent, example of persistent is remember me option used in most login pages. the user name and password are stored for long time
- non persistent cookies are temporarily used for storage, for short span of time
- persistent cookies and non-persistent cookies otherwise called as memorizable cookies and non-memorizable cookies.
- Persistent cookie means the cookie will be expired as soon as the application is closed
- Non-persistent cookie means even the application is closed the data will be remained as per the cookie timeout value.

WHAT ARE COOKIES?

Cookies are small pieces of information stored in the client computer. They are limited to 4K in size. Session Cookies and Persistent Cookies are two types of cookies.

Session Cookies are stored in memory during the client browser session. When the browser shutdown the session cookies are lost.

Example #1:

```
Dim objCookie As New HttpCookie ("Username", "Jim")
Response.Cookies.Add (objCookie)
```

READ COOKIE

```
Request. Cookies ("Username"). Value
```

Persistent Cookies work the same way as session cookies. The difference between the two is that persistent cookies have an expiration date.

Example #2:

```
Dim objCookie As New HttpCookie ("Username"," Jim")
ObjCookie.Expires "#12/31/2005#"
Response.Cookies.Add (objcookie)
```

Read Cookie

```
Request. Cookies ("Username"). Value
```

How to Create a Cookie?

The "Response.Cookies" command is used to create cookies.

Example #3:

```
Response.Cookies("myName")("sarav") = "Saravanan"
Response.Cookies("friends")("selva") = "Selvakumar"
Response.Cookies("friends")("venkat") = "Venkatesh"
Response.Cookies("friends")("hari") = "Hariraam"
Response.Write("Cookies Created!")
```


HOW TO RETRIEVE A COOKIE VALUE?

The "Request.Cookies" command is used to retrieve a cookie value.

Example #4:

```
Response.Write("My Name is:" & Request.Cookies("myName")("sarav"))
```

Output:

My Name is: Saravanan

HOW TO READ ALL COOKIES?

If a cookie contains a collection of multiple values, we say that the cookie has Keys. In the example below, we will create a cookie collection named "friends". The "friends" cookie has Keys that contains information about a friends:

Example #5:

```
Dim fnd As String
For Each fnd in Response.Cookies("friends").Values
Response.Write(Request.Cookies("friends")(fnd) & "")
Next
```

Output:

Selvakumar
Venkatesh
Hariraam

ADVANTAGES:

i) It can be used to store information about a particular client.
ii) It is simple to use.
iii) It is quite secure.
iv) No server resources.

DISADVANTAGES:

i) It has limited size.
ii) No security.

LIMITATIONS OF COOKIES:

(1) it can't store complex data (like dataset ..)

(2) cookies stored in client machine only so there is no SECURITY then what about the data of very securable (like passwords,creditcard details etc..)

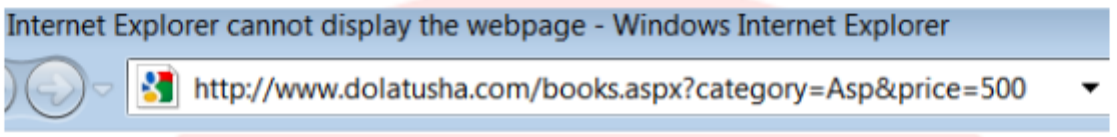
To overcome HTTPcookies limitations, SESSIONS are solution

Persistent cookies are stored on your computer hard disk. They stay on your hard disk and can be accessed by web servers until they are deleted or have expired. Persistent cookies are not affected by your browser setting that deletes temporary files when you close your browser

.Non-persistent cookies are saved only while your web browser is running. They can be used by a web server only until you close your browser. They are not saved on your disk. Microsoft Internet Explorer 5.5 can be configured to accept non-persistent cookies but reject persistent cookies

QUERYSTRING:-

- A query string is information that is appended to the end of a page URL.
- Query strings are usually used to send information from one page to another page. They are passed along with URL in clear text.
- This information is passed in URL of the request.
- A typical URL with a query string looks like:



- In the URL path above, the query string starts with a question mark (?) and includes two attribute/value pairs, one called "category" and the other called "price".
- We can only pass smaller amounts of data using query strings.
- Most browsers impose a limit of 255 characters on URL length.
- Since Query strings are sent in clear text, we can also encrypt query values.
- Syntax:-

SENDING ONLY ONE QUERYSTRING VALUE.

Sender page

```
Response.Redirect("PageName?KeyName=Value")
```

Reciever page

```
Str=Request.QueryString("KeyName")
```

SENDING ONLY ONE QUERYSTRING VALUE.

Sender page

```
Response.Redirect("PageName?KeyName1=Value1&
KeyName2=Value2")
```

Reciever page

```
Str1=Request.QueryString("KeyName1")

Str2=Request.QueryString("KeyName2")
```

SUMMARY OF CLIENT SIDE STATE MANAGEMENT

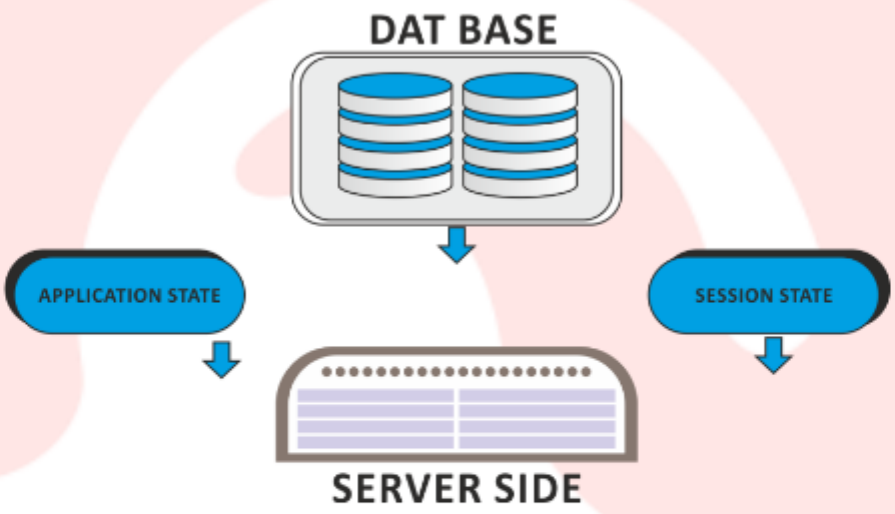
View State:- Use when you need to store small amounts of information for a page that will post back to itself.

Hidden Fields:- As above and security is not an issue that time use.

Cookies:- Use when you need to store small amounts of information on the client PC and security is not an issue.

Query String :- Use when you are transferring small amounts of information from one page to another and security is not an issue.

2. SERVER SIDE MANAGEMENT:



- With server-based state management, you can decrease the amount of information sent to the client in order to preserve state, however it can use costly resources on the server.

APPLICATION:-

- Application object provided a great place to store frequently used pieces of data that changed infrequently, such as the contents of menus or other reference data.
- ASP.NET implements application state using the System.Web.HttpApplicationState class.
- It provides methods for storing information which can be accessed globally.
- Information stored on application state will be available for all the users using the website.
- Application object is used to store data which is visible across entire application and shared across multiple user sessions.
- The application object provides a mechanism for storing that is accessible to all code running within the web application.

- The information that is global to the application may be stored in application object.
- For efficiency, this state is typically stored once and then read many times.
- A collection of user-defined variables which are shared by an ASP.NET application are termed as application state.
- When the Application_OnStart event fires at the time of loading of the first instance of the applications these variables are set and initialized and are available till the last instance of the application exits.
- Application state is stored in the memory of the windows process which is processing user requests on the web server.
- Application state is useful in storing small amount of often-used data.
- Unlike session state, which is specific to a single user session, application state applies to all users and all sessions.
- Syntax:- Application("KeyName") = Value
- Ex:- Application("Name") = "Yatin"

SESSION:-

- ASP.NET allows you to save values by using session state, which is an instance of the HttpSessionState class, for each active web-application session.
- A cookie is very simple and is not suitable for sophisticated storage requirements.
- Session state gives a method to keep more complex objects securely.
- ASP.NET allows programmers to keep any type of objects in session.
- Like Querystring we can use session to store value and pass it to another page, unlike Querystring, session will not display on the end of the URL.
- Session state is similar to application state, except that it is scoped to the current browser session.
- A session is the time for which a particular user interacts with a web application.
- Every client that uses the application will have separate sessions.
- During session the unique identity of the user is maintained internally.
- Session state is ideal for storing user specific information.
- If different users are using your application, each user session will have a different session state.
- If a user leaves your application and then returns later, the second user session will have a different session state from the first.
- Do not store large quantities of information in session state.
- A collection of user-defined session variables are termed as Session State, which are persevered during a user session.
- These variables are accessed using a session collection. These variables have a unique instance to different instances of a user session for the application.
- Session variables can be set to automatically destroyed after a definite period of inactive time, irrespective of the session state whether session ends or not.
- Syntax:- Session("KeyName") = Value
- Ex:- Session("Name") = "Chetan"

IMPORTANT METHODS/PROPERTIES OF SESSION.

- **Session.Abandon :-** cancels the current session.
- **Session.Remove :-** Delete an item from the session-state collection.
  Ex- Session.Remove(Uname)
- **Session.RemoveAll :-** Deletes all session state items.
- **Session.SessionID :-** Get the session Id read only property of a session for the current session.
- **Session.Timeout :-** if a user does not request a page of the ASP.NET application within certain minutes then the session expires.
  Ex- Session.Timeout = 30

DATABASE:-

- Database enables you to store large amounts of information pertaining to state in your web application.
- Some times users continually query the database by using the unique ID, you can save it in the database for use across multiple request for the pages in your site.
- You can store as much information as you like in a database.
- Access to databases requires authentication and authorization.

SUMMARY OF SERVER SIDE STATE MANAGEMENT

Application :- Use when you are storing infrequently changed, global information that is used by many users, and security is not an issue.

Session :- Use when you are storing short-lived information that is specific to an individual session and security is an issue.

Database :- Use when you are storing large amount of information, managing transaction, or the information must survive application and session restarts.

WHAT IS WEB.CONFIG FILE?

- Web.config file, as it sounds like is a configuration file for the Asp .net web application.
- An Asp .net application has one web.config file which keeps the configurations required for the corresponding application.
- Web.config file is written in XML with specific tags having specific meanings.

WHAT IS MACHINE.CONFIG FILE?

- As web.config file is used to configure one asp .net web application, same way Machine.config file is used to configure the application according to a particular machine.
- That is, configuration done in machine.config file is affected on any application that runs on a particular machine.
- Usually, this file is not altered and only web.config is used which configuring applications.

WHAT CAN BE STORED IN WEB.CONFIG FILE?

- There are number of important settings that can be stored in the configuration file. Here are some of the most frequently used configurations, stored conveniently inside Web.config file.

    1) DATABASE CONNECTIONS
    2) SESSION STATES
    3) ERROR HANDLING
    4) SECURITY

1) DATABASE CONNECTIONS:

- The most important configuration data that can be stored inside the web.config file is the database connection string.
- Storing the connection string in the web.config file makes sense, since any modifications to the database configurations can be maintained at a single location. As otherwise we'll have to keep it either as a class level variable in all the associated source files or probably keep it in another class as a public static variable.
- But it this is stored in the Web.config file, it can be read and used anywhere in the program.
- This will certainly save us a lot of alteration in different files where we used the old connection.
- Lets see a small example of the connection string which is stored in the web.config file .

```
<configuration>
 <appSettings>
   <add key="ConnectionString"
       value="server=localhost;uid=sa;
              pwd=;database=DBPerson" />
 </appSettings>
</configuration>
```

- As you can see it is really simple to store the connection string in the web.config file. The connection string is referenced by a key which in this case is "ConnectionString"
- The value attribute of the configuration file denotes the information about the database
- Here we can see that if has database name, userid and password. You can define more options if you want.
- Lets see how we access the connection string from our Asp .net web application.

```
Dim cnn as New OleDbConnection = ConfigurationSettings.AppSettings("ConnectionString");
```

- The small code snippet above is all that is needed to access the value stored inside the Web.config file.

2) Session States

- Session in Asp .net web application is very important. As we know that HTTP is a stateless protocol and we needs session to keep the state alive
- Asp .net stores the sessions in different ways. By default the session is stored in the asp .net process. You can always configure the application so that the session will be stored in one of the following ways.

2.1 Session State Service

There are two main advantages of using the State Service.

- First the state service is not running in the same process as the asp .net application. So even if the asp .net application crashes the sessions will not be destroyed.
- Any advantage is sharing the state information across a Web garden (Multiple processors for the same computer).
- Lets see a small example of the Session State Service.
- <sessionState mode="StateServer" stateConnectionString="tcpip=127.0.0.1:55455"
- sqlConnectionString="data source=127.0.0.1;user id=sa;password=" cookieless="false"
- timeout="20"/>
- **mode:** This can be StateServer or SqlServer. Since we are using StateServer we set the mode to StateServer.
- **stateConnectionString:** connectionString that is used to locate the State Service.
- **sqlConnectionString:** The connection String of the sql server database.
- **cookieless:** Cookieless equal to false means that we will be using cookies to store the session on the client side.

2.2 SQL SERVER

- The final choice to save the session information is using the Sql Server 2000 database. To use Sql Server for storing session state you need to do the following:
- Run the InstallSqlState.sql script on the Microsoft SQL Server where you intend to store the session.
- You web.config settings will look something like this:
- <sessionState mode = "SqlServer" stateConnectionString="tcpip=127.0.0.1:45565" sqlConnectionString="data source="SERVERNAME;user id=sa;password=" cookiesless="false" timeout="20"/>
- SQL Server lets you share session state among the processors in a Web garden or the servers in a Web farm. Apart from that you also get additional space to store the session. And after that you can take various actions on the session stored.
- The downside is SQL Server is slow as compared to storing session in the state in process. And also SQL Server cost too much for a small company.

2.3 INPROC:

- This is another Session State. This one is mostly used for development purposes. The biggest advantage of using this approach is the applications will run faster when compared to other Session state types.
- But the disadvantage is Sessions are not stored when there is any problem that occurs with the application, when there is a small change in the files etc., Also there could be frequent loss of session data experienced.

Error Handling:

- Error handling is one of the most important part of any web application. Each error has to be caught and suitable action has to be taken to resolve that problem. ASP. net web.config file lets us configure, what to do when an error occurs in our application.
- Check the following xml tag in the web.config file that deals with errors:

```
<customErrors mode = "On">
    <error statusCode = "404" redirect = "errorPage.aspx" />
</customErrors>
```

- This tells the Asp.net to display custom errors from a remote client or a local client and to display a page named errorPage.aspx. Error "404" is "Page not found" error.
- If custom error mode is turned "off" than you will see Asp.net default error message. This error messages are good for debugging purposes but should never be exposed to the users. The users should always be presented with friendly errors if any.

3) SECURITY:

The most critical aspect of any application is the security. Asp.net offers many different types of security method which can be used depending upon the condition and type of security you need.

1) NO AUTHENTICATION:

No Authentication means "No Authentication", meaning that Asp.net will not implement any type of security.

2) WINDOWS AUTHENTICATION:

The Windows authentication allows us to use the windows user accounts. This provider uses IIS to perform the actual authentication, and then passes the authenticated identity to your code. If you like to see that what windows user is using the Asp.net application you can use:

User.Identity.Name;

This returns the *DOMAIN\UserName* of the current user of the local machine.

3) PASSPORT AUTHENTICATION:

Passport Authentication provider uses Microsoft's Passport service to authenticate users. You need to purchase this service in order to use it.

4) FORMS AUTHENTICATION:

- Forms Authentication uses HTML forms to collect the user information and than it takes required actions on those HTML collected values.
- In order to use Forms Authentication you must set the Anonymous Access checkbox checked. Now we need that whenever user tries to run the application he/she will be redirected to the login page.

```
<authentication mode="Forms">
        <forms loginUrl = "frmLogin.aspx"  name="3345C“
         timeout="1" />
</authentication>
<authorization>
        <deny users="?" />
</authorization>
```

- As you can see we set the Authentication mode to "Forms". The forms loginUrl is the first page being displayed when the application is run by any user.
- The authorization tags has the deny users element which contains "?", this means that
    - full access will be given to the authenticated users and none access will be given to the unauthenticated users.
    - You can replace "?" with "*" meaning that all access is given to all the users no matter what.

❖ SITEMAPPATH SERVER CONTROL

- SiteMapPath is a Navigation Control.
- The SiteMapPath control displays the navigation path to the current page respect to home page. The path acts as clickable links to previous pages.
- It is used to access web pages of the website from one webpage to another. It is displays the map of the site related to its web pages.
- Maintaining the menu of a large web site is difficult and time consuming. The SiteMapPath control obtains navigation data from a site map. This data includes information about the pages in your Web site, such as the URL, title, description, and location in the navigationhierarchy. Storing navigation data in one place makes it easier to add and remove items in the navigational menus of a Web site. If you use it on a page that is not represented in your site map, it will not be displayed.
- It provides a space-saving way to easily navigate a site and serves as a point of reference for where the currently displayed page is within a site. TreeView or Menu might require too much space on a page.
- Make site well-structured or well-linked (internal links)

SYNTAX

```
<asp:SiteMapPath runat="server" />
```

NODES

The SiteMapPath is made up of nodes. Each element in the path is called a node and is represented by a SiteMapNodeItem object.

NODE TYPE	DESCRIPTION
root	A node that anchors or represents base of a hierarchical set of nodes.
parent	A node that has one or more child nodes.
current	A node that represents the currently displayed page.

Property: common property

OTHER IMPORTANT PROPERTY

NodeStyle	Gets the style used for the display text for all nodes in the site navigation path.
ParentLevelsDisplayed	Gets or sets the number of levels of parent nodes the control displays.
PathDirection	Gets or sets the order that the navigation path nodes are rendered in. Values are: RootToCurrent or CurrentToRoot
PathSeparator	It set path separator between nodes in the navigation path.
PathSeparatorStyle	Set the style used for the PathSeparator.
PathSeparatorTemplate	sets a template to use for the path delimiter of a site navigation path.
RenderCurrentNodeAsLink	Indicates whether the site navigation node is rendered as a hyperlink.
RootNodeStyle	Set the style for the root node display text.
RootNodeTemplate	sets a template to use for the root node of a site navigation path.
Site	Gets information about the container that hosts the current control when rendered on a design surface.

METHODS

NAME	DESCRIPTION
DataBind	Infrastructure. Binds a data source to the SiteMapPath control and its child controls.
Dispose	Enables a server control to perform final clean up before it is released from memory.
Focus	Sets input focus to a control.
InitializeItem	Populates a SiteMapNodeItem

EVENT

NAME	DESCRIPTION
DataBinding	Occurs when the server control binds to a data source
Disposed	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested.
Init	Occurs when the server control is initialized, which is the first step in its lifecycle.
ItemDataBound	Occurs after a SiteMapNodeItem has been bound to its underlying SiteMapNode

Load	Occurs when the server control is loaded into the Page object.
PreRender	Occurs after the Control object is loaded but prior to rendering.
Unload	Occurs when the server control is unloaded from memory.

❖ USER CONTROL

ASP.NET USER CONTROLS

- In addition to using Web server controls in your ASP.NET Web pages, you can create your own custom, reusable controls using the same techniques you use for creating ASP.NET Web pages. These controls are called **user controls.**
- A user control is a kind of composite control that works much like an ASP.NET Web page—you can add existing Web server controls and markup to a user control, and define properties and methods for the control. You can then embed them in ASP.NET Web pages, where they act as a unit.
- At times, you might need functionality in a control that is not provided by the built-in ASP.NET Web server controls. In those cases, you can create your own controls. You have two options. **You can create:**
- User controls. User controls are containers into which you can put markup and Web server controls. You can then treat the user control as a unit and define properties and methods for it.
- Custom controls. A custom control is a class that you write that derives from Control or WebControl.
- User controls are substantially easier to create than custom controls, because you can reuse existing controls. They make it particularly easy to create controls with complex user interface elements.

How to: Create ASP.NET User Controls

You create ASP.NET user controls in much the same way that you design ASP.NET Web pages. You can use the same HTML elements and controls on a user control that you do on a standard ASP.NET page. However, the user control does not have html, body, and form elements; and the file name extension must be .ascx.

TO CREATE AN ASP.NET USER CONTROL

1. Open the Web site project to which you want to add user controls. If you do not already have a Web site project, you can create one.
2. On the **Website** menu, click **Add New Item**. The **Add New Item** dialog box appears.
3. In the **Add New Item** dialog box, under **Visual Studio installed templates**, click **Web User Control**.
4. In the **Name** box, type a name for the control By default, the .ascx file name extension is appended to the control name that you type.
5. From the **Language** list, select the programming language that you want to use.

6. Optionally, if you want to keep any code for the user control in a separate file, select the **Place code in separate file** check box.
7. Click **Add**.

The new ASP.NET user control is created and then opened in the designer. The markup for this new control is similar to the markup for an ASP.NET Web page, except that it contains an @ Control directive instead of an @ Page directive, and the user control does not have html, body, and form elements.

Add any markup and controls to the new user control, and add code for any tasks that the user control will perform, such as handling control events or reading data from a data source.

HOW TO: INCLUDE ASP.NET USER CONTROLS IN WEB PAGES VISUAL STUDIO 2010

Adding an ASP.NET user control to a Web page is similar to adding other server controls to the page. However, you must be sure to follow the procedure below so that all of the necessary elements are added to the page.

TO ADD AN ASP.NET USER CONTROL TO A WEB PAGE

1. In Visual Web Developer, open the Web page to which you want to add the ASP.NET user control.
2. Switch to Design view.
3. Select your custom user control file in Solution Explorer, and drag it onto the page.

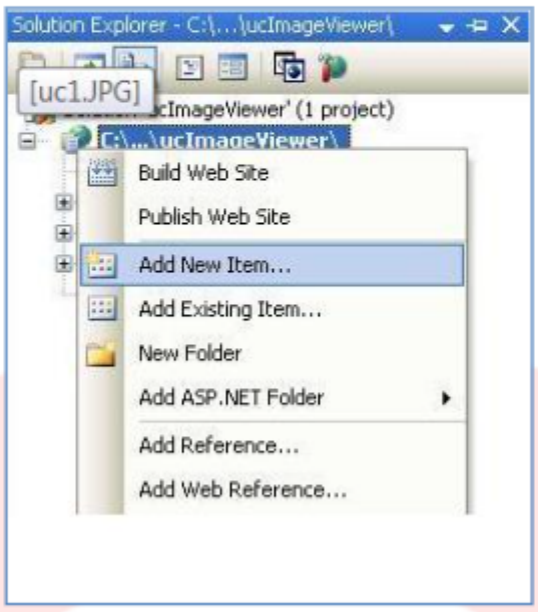
The ASP.NET user control is added to the page. In addition, the designer creates the @ Register directive, which is required for the page to recognize the user control.
You can now work with the control's public properties and methods.

CREATING WEB USER CONTROL IN ASP.NET

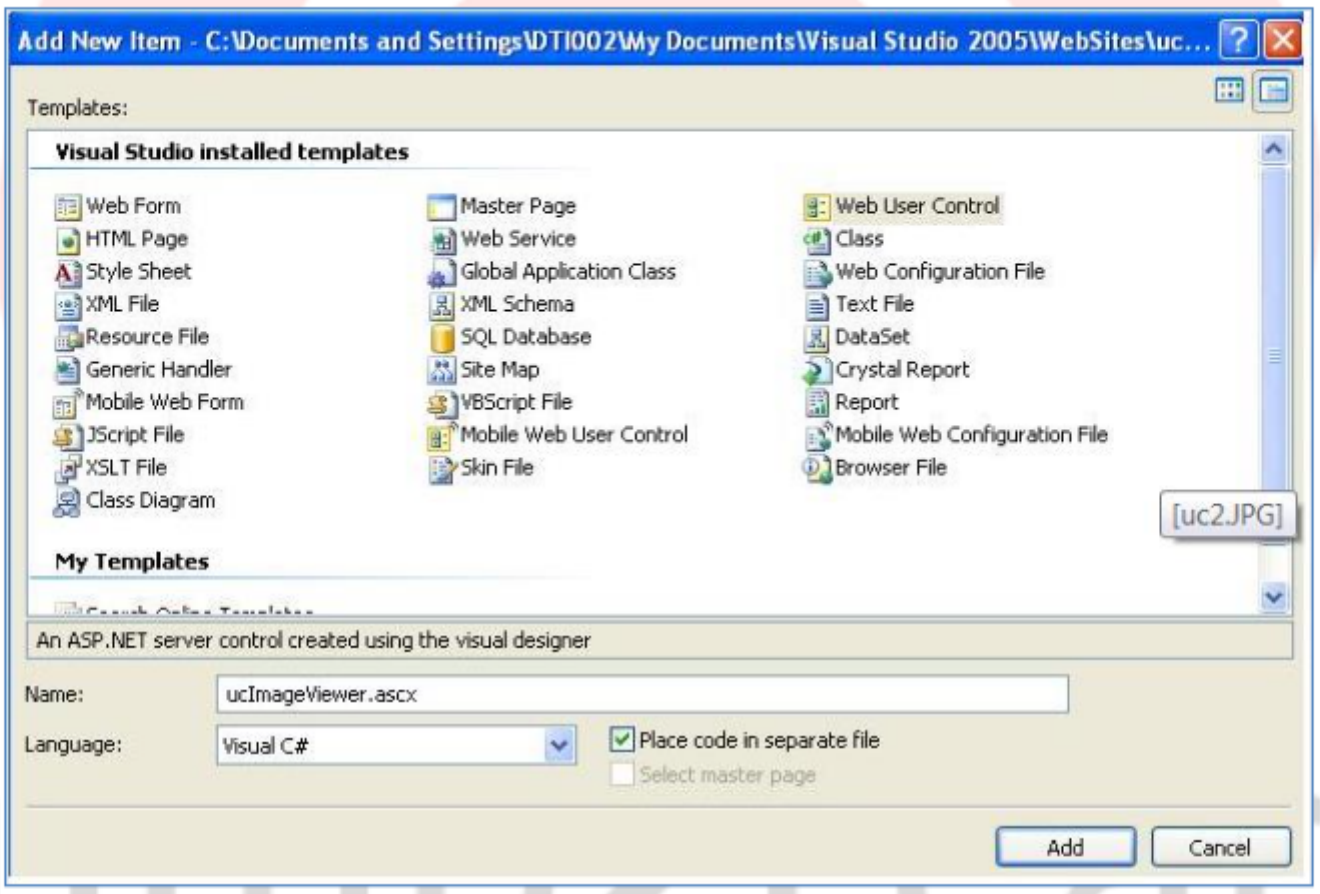
We are going to create a simple ImageViewer Control, which display thumbnail view of image and on clicking "Original View" under it will redirect you to original view of Image. Practically it has is scrap, but I have used here so that example remain simple throughout the discussion.

STEP1: Create Asp.net Website Project

STEP2: Add User Control as shown in figure.



Select "Web User Control" from New Item Window and name it "ucImageViewer"



STEP3: Now Add following code to file .ascx

Here it Adds a Image Server Control and Place a Button control below it. To organize properly it has displayed in table.

```
%@ Control Language="C#" AutoEventWireup="true"CodeFile="ucImageViewer.ascx.cs"
Inherits="ucImageViewer"%>

<table border="0" cellpadding="0" cellspacing="0">
```

```
<tr>
<td align="center">

<asp:Image ID="Image1" runat="server" />

</td>
</tr>
<tr>
<td align="center">

<asp:Button ID="Button1" runat="server" Text="Original View"OnClick="Button1_Click" />

</td>
</tr>
</table>
```

STEP4: Now as we want to view "Original View" of Image, we need to write a code on button click control, so for that switch to code view i.e. open **.ascx.cs** file and add following code.

```
protected void Button1_Click(object sender, EventArgs e)

{
if (ImagePath == string.Empty)

return;
else
Response.Redirect(ImagePath);
}
```

STEP5: Adding public property to Web User Control

Note, here you can access the Image Control property within User Control, but it won't be available when you place user control on web form, so to make desired ImageControl property accessible you need to define public property.

```
public string ImagePath

{
get
{
return m_ImagePath;

}
```

```
set
{
m_ImagePath = value;

}
}
public string ImageName

{
get
{
return m_ImageName;

}
set
{
m_ImageName = value;

}
}

public int ImageHeight

{
get
{
return m_ImageHeight;

}
set
{
m_ImageHeight = value;

}
}

public int ImageWidth

{
get
{
return m_ImageWidth;

}
set
```

```
{
m_ImageWidth = value;

}
}
```

So we are done forming Web User Control, now its turn to render Web User Control on Web Form.

DISPLAYING WEB USER CONTROL ON WEB FORM IN ASP.NET

To display web user control you can simply drag the user control on to web form as you are dragging web server control.

Lets understand manually how we can add the Web User Control on Web Form

STEP1: Add Register directive

Like <%@ Page %> directive, you can create Register directive

```
<%@ Register Src="ucImageViewer.ascx" TagName="ImageViewer" TagPrefix="uc" %>
```

Understanding Attributes of Register directive

Src - It specifies source of web user control.

TagPrefix - It Specifies Prefix for user control, i.e. Namespace to which it belong.

TagName - It Specifies Alias to the user control. i.e. Id to Control.

Now, you need to add the control

```
<uc:ImageViewer ID="ImageViewer1" runat="server" />
```

Note: As we have form public property with Web User Control, we can access

ImagePath - To Specify path of Image.

ImageName - To display Image Name as alternate text.

ImageHeight

ImageWidth

Example:

```
protected void Page_Load(object sender, EventArgs e)

{
ImageViewer1.ImagePath = "Image/shriganesh1.jpg";

ImageViewer1.ImageName = "Shri Ganesh 1st Pic";

ImageViewer1.ImageHeight = 200;

ImageViewer1.ImageWidth = 200;

}
```

DISPLAYING OUTPUT OF WEB USER CONTROL RENDERING

After clicking button, it display Image Original View as shown in figure.

Now, lets understand how to load user control dynamically.

DYNAMICALLY LOADING OF USER CONTROL IN ASP.NET

```
protected void Page_Load(object sender, EventArgs e)

{
Control c = Page.LoadControl("ucImageViewer.ascx");


((ucImageViewer)c).ImagePath = "Image/shriganesh1.jpg";

((ucImageViewer)c).ImageName = "Shri Ganesh Pic";

((ucImageViewer)c).ImageHeight = 150;

((ucImageViewer)c).ImageWidth = 150;


Panel1.Controls.Add(c);
}
```

CREATING A WEB USER CONTROL FROM EXISTING WEB FORM IN ASP.NET

STEP1: Remove all <HTML>, <Body> and <Form> Tags.

STEP2: Change the @Page directive to a @Control directive and ensures that no unsupported attributes remain.
STEP3: Add a ClassName attribute to the @Control directive.

STEP4: Rename the file to a name that reflects its purpose and then change the file extension from .aspx to .ascx.

OVERVIEW OF WEB CUSTOM CONTROL

Web Cutsom Controls provides more flexibility as compare to web user control. With Web Custom Control you can provide

- Design-Time Support
- Data Binding
- Event Handling and more advance feature.

DIFFERENT WAYS TO CREATE WEB CUSTOM CONTROL

There are 3 different ways of creating a web custom control

1. Composite Control - By combining two or more controls.
2. Derived Control - By Inheriting from a server control.
3. From Scratch - By Inheriting from the generic System.Web.UI.Control class.

WEB USER CONTROL VS WEB CUSTOM CONTROL

Difference between Web User Control and Web Custom Control or Points to be consider while Choosing Web User Control or Web Custom Control

- Web User Control can only be used within the same project, while Web Custom Control can be used across many projects.
- Web User Control cannot be added to Visual Studio.Net Toolbox, while Web Custom Control can be added.
- Web Custom Control are better choice when you want dynamic layout tasks in which constituent controls must be created at runtime.

FACTS ABOUT WEB USER CONTROL IN ASP.NET

- When you include a web user control in a web form, the user control participates in the event life cycle for the web form.
- User control Load and Pre-Render events are fired only after the web form's load and PreRender events are fired respectively.
- Web User controls are not precompiled into assemblies like the components and custome controls, they can only be used in web applications that have a physical copy of the user control. Thus, every application that wants to use a user control should have its own copy of that user control.
- Web User Control cannot be added into Visual Studio.Net Toolbox
- Web User Control do not expose their properties through the properties window
- Web User Controls in asp.net are Language neutral, that is user control written in C# can be used on a web form that is written in VB.Net.

❖ WEB SERVICES

- A web service is an entity that you can program to provide a particular functionality to application over the internet.
- A web service allows a website to communicate with other websites irrespective of the programming languages in which they are created.
- A web service can be accessed by any application, regardless of the software and hardware platforms on which the application is running, because the web service complies with common industry standards, such as **Simple Object Access Protocol (SOAP)** and **Web Services Description Language (WSDL).**
- A service does not have any user interface; it only contains the logic for providing specific services to its clients.
- A web service provides an abstraction between the client and the provider of the web service.

ADVANTAGES OF WEB SERVICES:-

- Web services are simple to use; and consequently, they can be implemented on varied platforms.
- Web services are loosely coupled; as a result, their interfaces and methods can be extended.
- Web services do not carry any state information with them so that multiple requests can be processed simultaneously.
- **Interoperability** - This is the most important benefit of Web Services. Web Services typically work outside of private networks, offering developers a non-proprietary route to their

solutions. To the use of standards-based communications methods, Web Services are virtually platform-independent.

- **Usability** - Web Services allow the business logic of many different systems to be exposed over the Web. This gives your applications the freedom to chose the Web Services that they need. Instead of re-inventing the wheel for each client, you need only include additional application-specific business logic on the client-side. This allows you to develop services and/or client-side code using the languages and tools that you want.
- **Reusability** - Web Services provide not a component-based model of application development, but the closest thing possible to zero-coding deployment of such services. This makes it easy to reuse Web Service components as appropriate in other services.
- **Deployability** - Web Services are deployed over standard Internet technologies. This makes it possible to deploy Web Services even over the fire wall to servers running on the Internet on the other side of the globe.

DISADVANTAGES OF WEB SERVICES:-

- Although the simplicity of Web services is an advantage in some respects, it can also be a obstacle. Web services use plain text protocols that use a fairly long-winded method to identify data. This means that Web service requests are larger than requests encoded with a binary protocol. The extra size is really only an issue over low-speed connections, or over extremely busy connections.
- Although HTTP is simple, they weren't really meant for long-term sessions. Typically, a browser makes an HTTP connection, requests a Web page and maybe some images, and then disconnects. The server may periodically send data back to the client. This kind of interaction is difficult with Web services, and you need to do a little extra work to make up for what HTTP doesn't do for you.

- The problem with HTTP, when it comes to Web services is that these protocols are "**stateless**"—the interaction between the server and client is typically brief and when there is no data being exchanged, the server and client have no knowledge of each other. More specifically, if a client makes a request to the server, receives some information, and then immediately crashes, the server never knows that the client is no longer active. The server needs a way to keep track of what a client is doing and also to determine when a client is no longer active.

- Typically, a server sends some kind of session identification to the client when the client first accesses the server. The client then uses this identification when it makes further requests to the server. This enables the server to recall any information it has about the client. A server must usually rely on a timeout mechanism to determine that a client is no longer active. If a server doesn't receive a request from a client after a predetermined amount of time, it assumes that the client is inactive and removes
- any client information it was keeping. This extra overhead means more work for Web service developers.

SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

- It is one of the key element in web service, is a protocol used for messaging. It is completely XML-based.
- SOAP provides a complete set of rules for messages, called SOAP envelopes, as well as the rules for issues, such as data encoding, message handling and message binding, to other protocol, such as HTTP.
- One of the great advantage of using a SOAP message is thatit is not restricted to any particular protocol.
- The two features that make SOAP a widely used protocol are support for remote procedure calls and a platform independency.
- The platform independent feature of SOAP allows a web service to communication with various applications, irrespective of hardware or software platform on which they are running.

WEB SERVICE DESCRIPTION LANGUAGE (WSDL)

- WSDL is an XML-based language that defines Web services.
- Every web service consists of a corresponding WSDL document that specifies the location of the web service and lists all the services that the web service can perform.
- A WSDL document is a simple XML document containing different XML elements to define a web service.

WEB SERVICE PROTOCOLS USED FOR DATA TRANSMISSION

- **HTTP-GET** :- Creates a Query String of the name /value pair and appends it to the URL of the requesting script on the server.
- **HTTP-POST** :- Passes the name/value pair in the body of the HTTP request message.
- **SOAP** :- Exchanges information in a decentralized, distributed environment.
- **MIME** :- It stands for Multipurpose Internet Mail Extension. It defines the standard representation for complex message bodies, such as messages with graphics and audio clips.

❖ ERROR HANDLING

The ***Errors*** caused by a computer program, regardless of the language in which the program is written.

It can be categorized into three major groups:

- Design-time
- Runtime
- Logic.

DESIGN-TIME

A ***Design-time error*** is also known as a **Syntax error**. These occur when the environment you're programming in doesn't understand your code. These are easy to track down in VB.NET, because you get a blue wiggly line pointing them out. If you try to run the programme, you'll get a dialogue box popping up telling you that there were Build errors.

RUNTIME

Runtime errors are a lot harder to track down. As their name suggests, these errors occur when the programme is running.

They happen when your programme tries to do something it shouldn't be doing. An example is trying to access a file that doesn't exist. Runtime errors usually cause your programme to crash. If and when that happens, you get the blame. After all, you're the programmer, and you should write code to trap runtime errors. If you're trying to open a database in a specific location, and the database has been moved, a Runtime error will occur. It's your job to predict a thing like this, and code accordingly.

LOGIC

LOGIC ERRORS also occur when the programme is running. They happen when your code doesn't quite behave the way you thought it would.

A classic example is creating an infinite loop of the type "Do While x is greater than 10". If x is always going to be greater than 10, then the loop has no way to exit, and just keeps going round and round. Logic errors tend not to crash your programme. But they will ensure that it doesn't work properly.

EXCEPTION:

An exception is a problem that arises during the execution of a program. An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

The Exception Handling is categories into two parts:

- Structured Exception Handling
- Unstructured Exception Handling

STRUCTURED EXCEPTION HANDLING

In ***structured exception handing***, we write code surrounded by blocks. If an exception occurs, the block throws the execution control to a predefined handled code, the try, catch, finally statements define these blocks. In other words, if an application handles exceptions that occur during the execution of a block of application code, the code must be placed with in a try statement.

Syntax:

```
Try
  [ tryStatements ]
  [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
  [ catchStatements ]
  [ Exit Try ] ]
[ Catch ... ]
[ Finally
  [ finallyStatements ] ]
End Try
```

Assuming a block will raise and exception, a method catches an exception using a combination of the Try and Catch keywords. A ***Try/Catch*** block is placed around the code that might generate an exception. Code within a ***Try/Catch*** block is referred to as protected code.

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

The codes reside in ***Finally*** block is executed every time if the Exception is generated or not.

Example:

```
Module Module1

  Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
    Dim result As Integer
    Try
      result = num1 \ num2
    Catch e As DivideByZeroException
      Console.WriteLine("Exception caught: {0}", e)
    Finally
      Console.WriteLine("Result: {0}", result)
    End Try
  End Sub

  Sub Main()
    division(25, 0)
    Console.Read()
  End Sub

End Module
```

Result:

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
  at ConsoleApplication1.Module1.division(Int32 num1, Int32 num2) in
E:\CP\Book\ConsoleApplication1\ConsoleApplication1\Module1.vb:line 5
Result: 0
```

If we change the statement division(25, 5) then output is Result : 5

THROWING OBJECTS

You can ***throw*** an object if it is either directly or indirectly derived from the **System.Exception** class.
You can use a throw statement in the catch block to throw the present object as:

Syntax:

```
Throw [ expression ]
```

Example:

```
Module Module1

  Sub Main()
    Try
      Console.WriteLine("Inside the Try block")
      Throw New ApplicationException("A custom exception is being thrown here...")
    Catch e As Exception
      Console.WriteLine(e.Message)
    Finally
      Console.WriteLine("Now inside the Finally Block")
    End Try
    Console.Read()
  End Sub

End Module
```

Result:

```
Inside the Try block
A custom exception is being thrown here...
Now inside the Finally Block
```

UNSTRUCTURED EXCEPTION HANDLING

The ***Unstructured Exception*** handling revolves around the ***On Error GoTo*** statement.

ON ERROR GOTO STATEMENT.

You use this statement to tell VB .NET where to transfer control to in case there's been an exception,

Synatx:

On Error { GoTo [*line* | 0 | -1] | Resume Next }

Where
GoTo *line* - Calls the error-handling code that starts at the line specified at *line*. Here, *line* is a line label or a line number. If a runtime error occurs, program execution goes to the given location. The specified line must be in the same procedure as the On Error statement.
GoTo 0 - Disables the enabled error handler in the current procedure.
GoTo -1 - Same as GoTo 0.
Resume Next - Specifies that when an exception occurs, execution skips over the statement that caused the problem and goes to the statement immediately following. Execution continues from that point.

Example:

```
Module Module1

  Sub Main()
    Dim a, b, c  As Integer
    a=1
    b=0
    On Error GoTo Handler
    c=a / b
    System.Console.WriteLine("The answer is {0}", c)
    Console.Read()
     Exit Sub
Handler:
    System.Console.WriteLine("Number Can not be divide by zero")
    Resume Next
  End Sub
End Module
```

Result:

```
Number Can not be divide by zero
The answer is 0
```

In above example when the code performs a division by zero, which causes an exception. When the exception occurs, control will jump to the exception handler, where I am display a message and then use the ***Resume Next*** statement to transfer control back to the statement immediately after the statement that caused the exception:

RESUME STATEMENT

After an exception occurs, you can use the *Resume* ***statement*** to resume program execution in unstructured exception handling. Here are the possibilities:

- Resume resumes execution with the statement that caused the error.
- Resume Next resumes execution with the statement after the one that caused the error.
- Resume *line* resumes execution at *line,* a line number or label that specifies where to resume execution.

Example:

```
Module Module1
  Sub Main()
     On Error GoTo Handler
     Dim i1 As Integer = 10
     Dim i2 As Integer = 0
     Dim i3 As Integer
      I3 = i1 / i2
LineAfter:
   Console.WriteLine("Press Enter to continue...")
   Console.ReadLine()
   Exit Sub
Handler:
   Console.WriteLine("An overflow error occurred.")
   Resume LineAfter
  End Sub
End Module
```

Result:

```
An overflow error occurred.
Press Enter to continue...
```

Err Object

When error occur, the ***Err*** object contains information about the error, which helps you to determine whether you can attempt to fix the error or ignore the error.

It have several methods that allow you to raise errors or clear the state of the ***Err*** object.

Example:

```
Module Module1

 Sub Main()
  On Error GoTo Handler
```

```
    Dim i1 As Integer = 10
    Dim i2 As Integer = 0
    Dim i3 As Integer
    i3 = i1 / i2
    Console.WriteLine("Press Enter to continue...")
    Console.ReadLine()
    Exit Sub
Handler:
    Console.WriteLine(Err.Description)
    Console.WriteLine("Press Enter to continue...")
    Console.ReadLine()
  End Sub

End Module
```

Result:

```
Arithmetic operation resulted in an overflow.
Press Enter to continue...
```