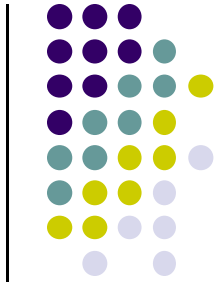
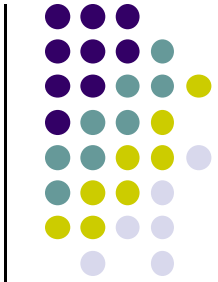


# ACTIVE SERVER PAGES .NET

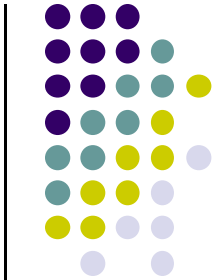
Microsoft .net



# Module 5

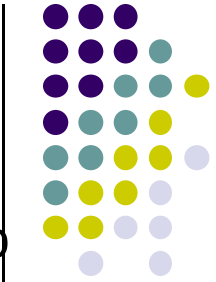
- Programming web applications with ASP.NET
  - Writing HTML pages and forms
  - Maintaining consistency with master pages
  - Designing pages with ASP.NET controls
  - Styling sites with ASP.NET themes
  - Processing ASP.NET forms

# Introduction



- Internet applications
  - Uses internet it any way, request users to register over internet, provide functionality over the network.
  - **Web Applications.** These applications provide content from a server to client machines over the internet. Users view a web application through a web browser.
  - **Web Services.** Components providing processing services from a server to other applications over the internet.
  - **Internet-enabled applications.** Stand-alone applications that incorporate aspects of the internet to provide Online Registration, Help, updates or other services to the user over the Internet.
  - **Peer-to-Peer applications.** Stand-alone applications that use the internet to communicate with other users running their own instances of the application.

# How web applications (WA) work?

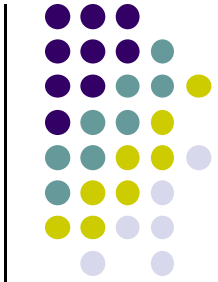


- Client server architecture. Resides on a server and responds to requests from multiple clients over the network.
- On client side, WA is hosted by a browser. The application's UI takes the form of HTML pages that are interpreted & displayed by the client's browser. On the server side, WA runs under IIS.
- IIS manages the application, passes request from clients to the application, and returns the application's responses to the client.
- These Requests & Responses are passes across the internet using HTTP. ( A protocol is a set of rules, that describes how two or more items communicate over a medium, such as the internet.)
- WA composes responses to requests from the resources found on the server.
- The content presented to the user is actually composed dynamically by executable, rather than being served from a static page stored on the server.
- WA = Client Server Application working over the internet.

# ASP.NET?

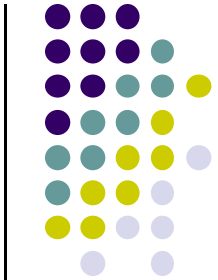
- ASP.NET is the platform that is used to create WA and WS that run under IIS.
- CGIs >> ASPs >> ASP.NET
- Is special because of its tight integration with Microsoft Servers, programming, data access & security tools.
- Provides high level of consistency across WA development.
- ASP.NET is a part of .NET Framework & is made of:
  - **Visual Studio .NET Web development tools** : include visual tools for designing web pages and application templates, project management & deployment tools for WA.
  - **System.Web namespaces**: part of .NET Framework & include the programming classes that deal with web-specific items such as HTTP requests & responses, browsers & e-mail.
  - **Server & HTML controls**: UI components that is used to gather info from and provide responses to users.

# Support keys for ASP.NET programming



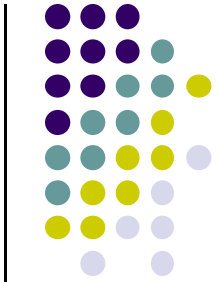
- **Microsoft Internet Information Services (IIS)**: IIS hosts WA on the Windows Server.
- **Microsoft Visual Basic .NET,** Microsoft Visual C# and Microsoft Visual J# Programming languages, have integrated support in Visual Studio .NET for creating WA.
- **The .NET Framework**: complete set of Windows Programming Classes, including the ASP.NET classes as well as classes for other programming tasks such as file access, data type conversion, array and string manipulation, and so on.
- **Microsoft ADO.NET database classes and tools**: these components provide access to MS SQL Server & ODBC databases.
- **Microsoft Application Center Test (ACT):** provides automated way to stress-test WA.

# Advantages of ASP.NET



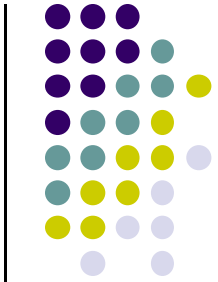
- Close integration with the windows server & programming tools.
- WA created with ASP.NET are easier to create , debug & deploy because those tasks can all be performed within a single development environment – VS .NET.
- On-the-fly updates of deployed web applications, without restarting the server.
- Automatic state management for controls on a web page, so that they can behave much more like windows controls.
- The ability to create new, customized server controls from existing controls.
- Built-in security through the windows server or through other authentication / authorization methods.

# Advantages of ASP.NET - 2



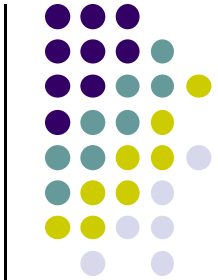
- Integration with ADO.NET to provide database access.
- Full support for XML, CSS  other new & established web standards.
- Built-in features for caching frequently requested web pages on the server, localizing content for specific languages & cultures, and detecting browser capabilities.

# Parts of a WA



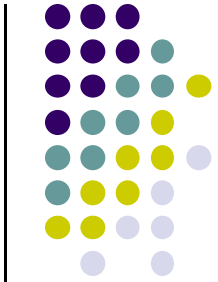
| Part | Types of files | Description |
|---|---|---|
| Content | Web forms, HTML, images, A/V, other data | Determine the appearance of a WA. Can c, contain static text as well as elements which are composed on the fly by the program logic, as in case of a db query. |
| Program Logic | Executable Files, Scripts | Determines how the application responds to the user actions. ASP.NET WA have a DLL file that runs on the server, and they can also include scripts that run on the client machine. |
| Configuration | Web Configuration File, Style Sheets, IIS settings | Determine how the application runs and behaves on the server, who has access, how errors are handled, and other details. |

# Web Forms



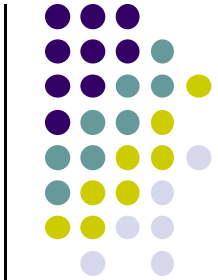
- In a completed WA, the executable portion of the web form is stored in an assembly (.dll) that runs on the server under the control of the ASP.NET worker process (asp_wp.exe), which runs in conjunction with IIS.
- The web form (WF) is the key element of a web application. A WF is a cross between a regular HTML page and a Windows Form. It has the same appearance as and similar behavior to an HTML page, but it also has controls that respond to events and run code, like a windows form.
- The content portion of a WF resides in a content directory of the Web server.

# Request Response Sequence for a ASP.NET WF

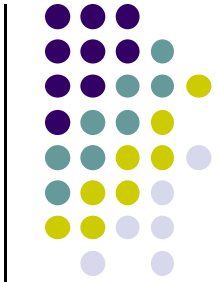


- When a user navigates to one of the web forms from his browser, the following sequence occurs:
  - IIS starts the ASP.NET worker process. The ASP.NET WP loads the assembly associated with the W.
  - The assembly composes a response to the user based on the content of the web form that the user requested and any program logic that provides dynamic content.
  - IIS returns the response to the user in the form of HTML.
  - Once user gets the requested WF, he can enter data, select options, click buttons, and use any other control that appear on the page. Some controls like Buttons, cause the page to be posted back to the server for event processing, and the sequence repeats itself.

# Web Form Components

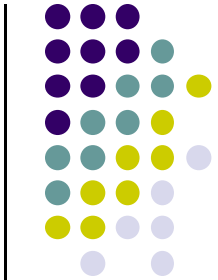


| Component | Examples | Description |
|---|---|---|
| Server controls | TextBox, Label, Button, ListBox, DropDownList, DataGrid | These controls respond to the user events by running event procedures **on the server.** |
| HTML controls | TextArea, Table, Image, Submit & Reset buttons | These represent the standard visual elements provided in HTML. |
| Data controls | SqlConnection, SqlCommand, OleDbConnection, OleDbCommand, DataSet | Provide a way to connect to , perform commands on, and retrieve data from SQL, OLE databases & XML data files. |
| System components | FileSystemWatcher, EventLog, MessageQueue | Provide access to various system level events that occur on the server. |



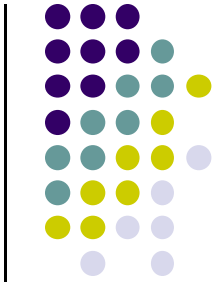
- NOTE: Server & HTML controls are used to create UI for the WA, on a WF. The data controls and system components appear on the web form only at the design time, to provide a visual way to set their properties and handle their events. At run-time, data controls & system components do not have a visual representation.

# WF Vs. Windows Forms



- Significant differences are as follows:
  - Tools: Web forms can't use the standard windows controls. Instead, they use Server controls, HTML controls, user controls or custom controls created specially for WF.
  - UI: The appearance of the WF is determined by the browser that displays it,
  - Lifetime: Windows forms are instantiated, exist for as long as needed, and are destroyed. Web forms are instantiated, sent to the browser, and then usually thrown away. This means that all the variables and objects declared in WF are usually not available after the WF is displayed! To get anything interesting done, information is saved in special state objects provided by ASP.NET.
  - Execution: The executable portion of a WA live on the web server, providing ultimate client-server environment. All communication between the client and server occurs through HTML.

# The files in a WF Project

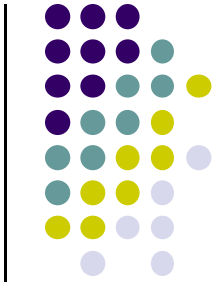


- The WF is only one of the 11 files VS .NET generates when it creates a new WF project.

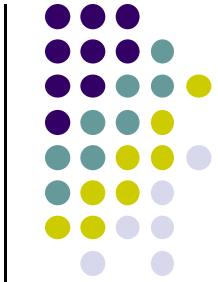
| Sr. No. | File name | Contains |
|---|---|---|
| 1 | AssemblyInfo.cs | All of the attributes that are written into the compiled assembly, including version, company name, GUID, and so on. |
| 2 | Global.asax | The global events that occur in the WA, such as when the application starts or ends. Only one Global.asax file is permissible per project and it exists in the root folder of the project. |
| 3 | Global.asax.cs | The code used in Global.asax. |
| 4 | Styles.css | The style definition to be used for the HTML generated by the project. |

| Sr. No. | File name | Contains |
|---|---|---|
| 5 | Web.config | The settings web server uses when processing this project. These settings specify how errors are reported, what type of user authentication to use, etc. |
| 6 | Projectname.vsdisco | Description of the Web Services that this project provides. This file is used for dynamic discovery of the web services included in a WA. |
| 7 | WebForm1.aspx | The visual description of a web form. |
| 8 | WebForm1.aspx.cs | The code that responds to events on the WF. |
| 9 | WebForm1.aspx.resx | The XML resources used by the WF. |
| 10 | Projectname.csproj | The project listing the files & settings used at design time. |
| 11 | Projectname.csproj.webinfo | This file tracks the root virtual folder for the web application. |

# File types for WF Projects

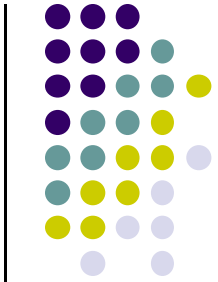


| File extension | Project item | Description |
|---|---|---|
| .aspx | WF | Each WF constitutes an ASP.NET Web Page in the WA. Have code files associated with them with extension .aspx.cs |
| .htm | HTML page | Web pages with NO server code |
| .cs | Class or module | Code that defines objects in the application is stored in classes |
| .ascx | Web user control | Built from other WF and server controls |
| .asmx | Web Services | Web services that expose classes for remote execution over a network, such as Internet. |



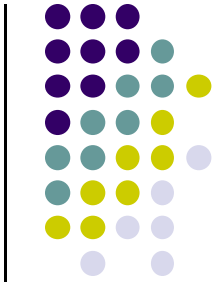
| File extension | Project item | Description |
|---|---|---|
| .xml | XML file | Data files that store information |
| .xsd | XML schema | Schema files that describe the format and constraints to apply to store data |
| .xslt | XML style sheet | Formatting rules to apply when displaying XML data. |

# Events in the life cycle of a WA



- A WA lives as long as it has active sessions, whereas WF live for barely a moment.
- The life of a WA begins when a browser requests the start page of the application. Web server starts the assembly that responds to that request. ML to respond to the request, and post that response to the browser. It then destroys the instance of the web form
- Postback events cause the browser to send the page's data (view state) back to the server for event processing.
- When the server receives the view state, it creates a new instance of the WF, fills in the data from the view state, and processes any events that occurred.
- As soon as the server is finished, it posts the resulting HTML back in the browser & destroys the instance of the WF.
- When the user stops using the WA for a period of time, the user's session times out & ends.
- If there are no other session from other users, the application ends.

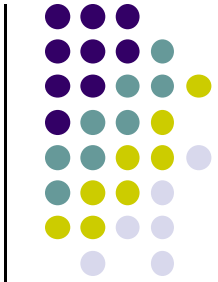
# Preserving data on a web form



- Because WF are short lived, ASP/NET takes special steps to preserve the data entered in the controls on a W.
- Data entered in the controls is sent with each request and restored to controls in Page_Init. The data in these controls is then available in the Page_Load event.
- The data that ASP.NET preserves between requests is called the WF's View state.
- By default, a WF's view state is available only withing that WF. To make it accessible to other WF in an application, the data is needed to be saved in a state variable in the Application or Session objects. These objects provide two levels of scope:
  - Application State variables: available to all users of an application. Multiple-use global data. All session can read or write these variables.
  - Session State variables: available only to a single session. Like global data in a standard windows application. Only the current session has access to its Session state.

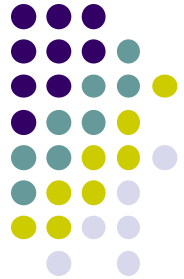
- Application & Session state variables are not declared the way you declare regular variables. They are created on the fly in code.
- You can save any type of data in a state variable, from a simple integer to whole objects.
- Application state variables must be initialized in visual C# before performing any operations on them.

# Application & Session Events



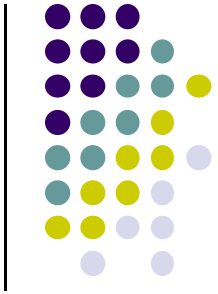
- Code, that responds to Application & Session events, is written in Global.asax
- Use Application events to initialize objects & data that you want to make available to all the current sessions of your WA.
- Use Session events to initialize data that you want to keep throughout individual sessions, but that you don't want to share between sessions.
- In WF, a session is a unique instance of the browser. A single user can have multiple instances of the browser running on the machine.
- If each instance visits your WA, each instance has a unique session.

# Application event handlers



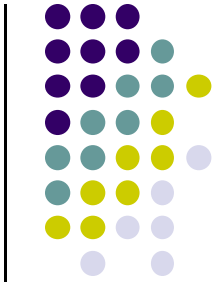
| Event handler name | Occurs when |
|---|---|
| Application_Start | The first user visits a page within your web application. |
| Application_End | There are no more users of the web application. |
| Application_BeginRequest | At the beginning of each request to the server. A request happens every time a browser navigates to any of the pages in the application. |
| Application_EndRequest | At the end of each request to the server. |
| Session_Start | A new user visits a page within your application |
| Session_End | A user stops requesting pages from the WA & their session times out. Session times out after a period specified in the Web.config file. |

# Web form events



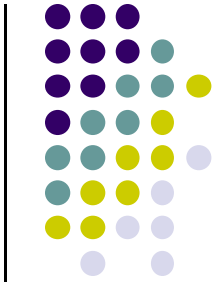
| Event handler name | Occurs when |
|---|---|
| Page_Init | The server controls are loaded and inititalized from the WF's view state. This is the first stage in a WF's life cycle. |
| Page_Load | The server controls are loaded in the Page object. View state info is available at this point, so this is where you put code. |
| Page_PreRender | The application is about to render the Page object. |
| Page_Unload | The page is unloaded from memory. |
| Page_Disposed | The Page object is released from the memory. This is the last event in the life of a Page object. |
| Page_Error | An unhandled exception occurs |
| Page_AbortTransaction | A transaction is aborted |
| Page_CommitTransaaction | A transaction is accepted. |
| Page_DataBinding | A server control on the page binds to a data source. |

# Server control events



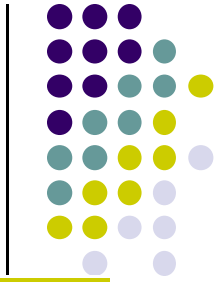
- Server controls have their own set of events that occur in response to user actions. There are three types of server control events:
  - Postback events: these events cause the webpage to be sent back for immediate processing. Postback events affect perceived performance as they trigger a round trip to the server. The Button, Link Button, and Image Button Controls all cause postback events.
  - Cached events: these events are saved in the page's view state to be processed when a postback event occurs. The TextBox, DropDownList, ListBox, Radio Button and CheckBox controls provide cached events.
  - Validation events: these events are handled on the page without posting back or caching. The validation server controls use these types of events.

# How IIS & ASP.NET manage processes

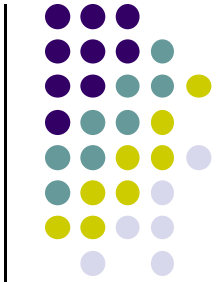


- When IIS receives a request for a resource within a WA, IIS uses aspnet_isapi.dll to call the ASP.NET worker process (aspnet_wp.exe). The ASP.NET worker process loads the WA's assembly, allocating one process space, called the **application domain**, for each application
- Sessions are important for 2 reasons:
  - They allow ASP.NET to keep user-specific data called the *Session* state.
  - They determine when the application ends. When the last session ends, the application ends.

# WA Namespace hierarchy

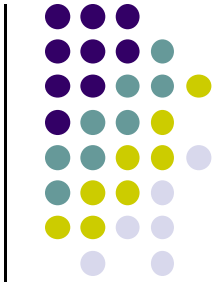


| Namespace | Contains classes for |
|---|---|
| System.web | The *Application, Browser, Cache, Cookies, Exception, Request, Response, Server* and *Trace* objects. The Application object defined in Global.asax is based on the Application class. |
| System.Web.SessionState | The *Session* object. Use this class to create an use Web Services. |
| System.Web.UI | The *Page* & *Control* objects. Use these classes within a WF to create and control an application's UI. WF are based on the Page class. |
| System.Web.UI.WebControls | All *Server* control objects. |
| System.Web.UI.HTMLControls | All HTML control objects. |
| System.Web.Caching | The Cache object, to control server side caching to improve application performance. |
| System.Web.Mail | The MailMessage, MailAttachment and SmtpMail objects. Use to mail messages from your WA. |
| System.Web.Security | Authentication objects and modules. Use these classes to authenticate users & provide security within the WA. |



- When programming a WA, you deal directly with two types of objects derived from classes in the Web namespaces:
  - The **Application object**: derived from the HttpApplication class. Definition resides in Global.asax.
  - **Web Form objects**: derived from the Page class. Definition resides in Web Form modules.

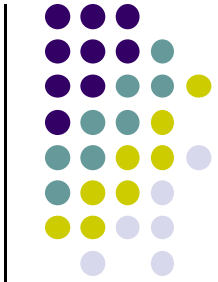
# Using the Application object



- Top level object in a WA's object hierarchy.
- Use Application object to configure your application, to save state info and to respond to application-wide events.

| Property / Method | Use to |
|---|---|
| Application | Save data items in the Application state |
| Context | Get Handler, Trace, Cache, Error and other objects for the current context. |
| Modules | Access HTTP modules |
| Request | Read a request & get Browser, ClientCertificates, Cookies and Files object from the current request. |
| Response | Write text or data to a response and get Cache, Cookies and Output objects from the current response. |
| Server | Process requests and responses |
| Session | Save data items in the Session state. |
| User | Get authentication information about the user making the current request. |

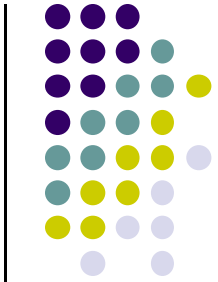
# Using the Page object



- Controls your application's UI.

| Property / Method | Use to |
|---|---|
| Application | Save data items in the Application state |
| Cache | Controls how responses are cached on the server |
| Controls | Get at controls on the page |
| Request | Read a request & get Browser, ClientCertificates, Cookies and Files object from the current request. |
| Response | Write text or data to a response and get Cache, Cookies and Output objects from the current response. |
| Server | Process requests and responses |
| Session | Save data items in the Session state. |
| Trace | Turn tracing on or off and write to the trace log. |

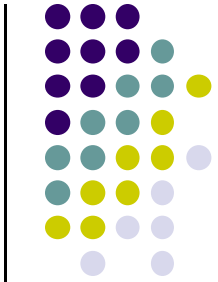
# Using the Request object



- Contains the info sent by the client browser when a page is requested from the application.

| Property / Method | Use to |
|---|---|
| Browser | Determines the capabilities of the browser making the request. |
| ClientCertificates | Authenticate the client. |
| Cookies | Get info from the client in the form of cookies |
| Files | Get files that are uploaded by the client. |
| InputStream | Read & write to the raw data sent in the request. |

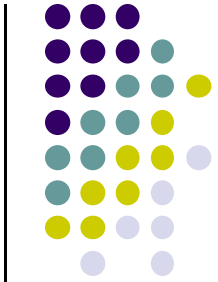
# Using the Response object



- Used to form the response sent from the server to the client browser.

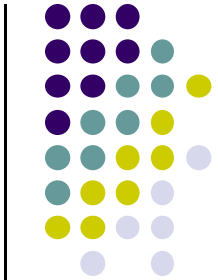
| Property / Method | Use to |
|---|---|
| Cache | Determines how the server caches responses before they are sent to the client. |
| Cookies | Set the content of cookies to send to the client. |
| Output | Get or set the raw data returned to the client as the response |

# Maintaining state information



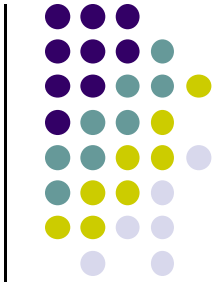
- WF are created & destroyed every time a client browser makes a request. Because of this characteristic, variables declared within a web form do no retain their value after a page is displayed.
- ASP.NET provides the following ways to retain variables between requests:
  - **Context.Handler object:** use this object to retrieve public members of one WF's class from a subsequently displayed WF.
  - **Query Strings:** use these strings to pass information between requests and responses as a part of the web address.
  - **Cookies**: use them to store small amount of info on a client.
  - **View State**: ASP.NET stores items added to a page's ViewState property as hidden fields on the page.
  - **Session State**: use session state variables to store items that you want keep local to the current session (single user)
  - **Application State**: to store items that you want be available to all users of the application.

# Summary of the .NET Framework BCL.



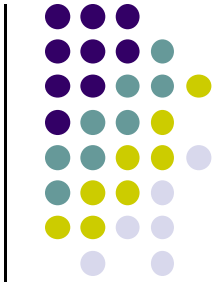
- The .NET Framework includes classes, interfaces, and value types that expedite and optimize the development process and provide access to system functionality. To facilitate interoperability between languages, most .NET Framework types are CLS-compliant and can therefore be used from any programming language whose compiler conforms to the common language specification (CLS).
- The .NET Framework types are the foundation on which .NET applications, components, and controls are built. The .NET Framework includes types that perform the following functions:
  - Represent base data types and exceptions.
  - Encapsulate data structures.
  - Perform I/O.
  - Access information about loaded types.
  - Invoke .NET Framework security checks.
  - Provide data access, rich client-side GUI, and server-controlled, client-side GUI.
- The .NET Framework provides a rich set of interfaces, as well as abstract and concrete (non-abstract) classes. You can use the concrete classes as is or, in many cases, derive your own classes from them. To use the functionality of an interface, you can either create a class that implements the interface or derive a class from one of the .NET Framework classes that implements the interface.

# Module 6



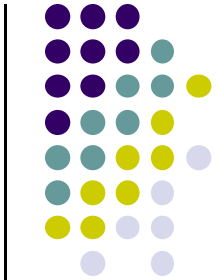
- Storing and Retrieving Data with ADO.NET

# Introduction



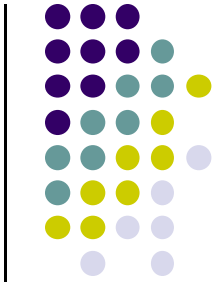
- Visual Studio.NET provides access to databases through the set of tools and namespaces collectively referred to as Microsoft ADO.NET. Data access is ADO.NET is standardized to be mostly independent of the source of data – once you've established a connection to the database, you use a consistent set of objects, properties & methods,, regardless of the type of database you are using.

# Understanding ADO.NET



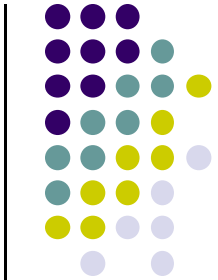
- There are 4 layers to data access in ADO.NET.
  - Physical data store: this can be a SQL, an OLE, or an Oracle database or an XML file.
  - The data provider: this consists of the Connection object and command object that create the in-memory representation of the data.
  - The data set: this is the in-memory representation of the tables and relationships that you work with in your application.
  - The data view: this determines the presentation of a table in the data set. Typically used to filter or sort data to present to the user. Each data set has a default data view matching the row order used to create the data set.
- The data provider layer provides abstraction between the physical data sore and the data set you work within the code. Once the dataset is created, it doesn't matter where the data comes from or is stored. This architecture is known as disconnected because the data set is independent of the data store.

# Types of DB connections in ADO.NET



- Use an OleDbConnection object to connect to a MS Access or third-party database, like MySQL. OLE database connections use the OleDbDataAdapter object to perform commands and return data.
- Use a SqlConnection object to connect to a MS SQL Server db. SQL db connections use the SqlDbDataAdapter object to perform commands and return data.
- Use an OracleConnection object to connect to Oracle databases. Oracle db connections use the OracleDataAdapter object to perform commnds and return data.
- XML files can directly be accessed from datasets using the DataSet object's ReadXML and WriteXML methods. XML files are static representation of DataSets. ADO.NET uses XML for all data transfer across the Internet.

# ADO.NET namespaces

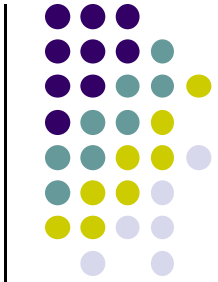


- ADO.NET provides its objects, properties and methods through the three namespaces.

| Namespace | Provides |
|---|---|
| System.Data | Classes, types and services for creating and accessing data sets and their subordinate objects. |
| System.Data.Sqlclient | Classes and types for accessing MS SQL Server databases. |
| System.Data.OracleClient | Classes and types for accessing Oracle databases |
| System.Data.OleDb | Classes and types for accessing other databases. |

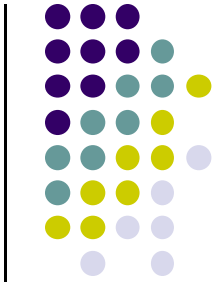
- While working with db, add the above namespaces as per requirement to the code.

# To access a db through ADO.NET



- Create a connection to the database using a connection object.
- Invoke a command to create a DataSet object using an adapter object.
- Use the DataSet object in code to display data or to change items in the database.
- Invoke a command to update the database from the DataSet object using an adapter object.
- Close the database connection if opened explicitly ussing the Open method. Invoking commands without first invoking the Open method implicitly opens and closes the connection with each request.

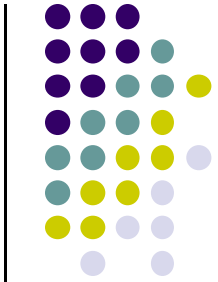
# Changing records in a database



- The DataSet object is the central object in ADO.NET. Any additions, deletions or changes to records in a database are generally done through a DataSet object.

- To change records through a DataSet object, follow these steps:
  - Get a DataSet object.
  - Modify the DataSet object.
  - Update the database from the DataSet object by calling the data adapter's Update Method.

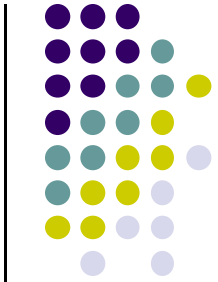
# How ADO.NET Refers to Objects



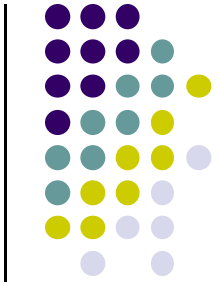
- When you create connection, adapter and data set objects in design mode, you enable data typing for those objects. This means that you can use the specific names from the database schema to identify tables, rows and fields.
- This is a big change from ADO, which provided only untyped references to data objects.

```
//Typed reference to the Contacts table's HomePhone column.
    dataSet1.Contacts.HomePhoneColumn.Caption = "@Home";

    //Untyped reference to the Contacts table's HomePhone Column.
    dataSet1.Tables["Contacts"].Columns["HomePhone"].Caption = "@Home";
```

# Creating a db connection at runtime. (not from designer pane)



- Using data access objects in code follows the same sequence of events as it does in Design mode:
  - Create the data connection object.
  - Create a data adapter object.
  - Create a data set object.
  - Invoke methods on the adapter object to fill or update the data set.
  - Use data binding or another technique to display the data from the data set.



The following code creates data objects and displays data from a Microsoft SQL data provider.

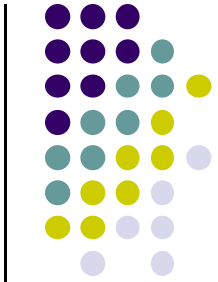
```
SqlDataAdapter m_adptContactMgmt;
Public void Page_Load(object sender, EventArgs e)
{
    //(1) Create the data connection.
    SqlConnection ContactMgmt = new
    SqlConnection("server=(local);database=Contacts;Trusted_Connection=yes");

    //(2) Create a data adapter.
    m_adptContactMgmt = new SqlDataAdapter("select * from Contacts",ContactMgmt);

    //(3) Create a data set.
    DataSet dsContacts = new DataSet();

    //(4) Fill the data set.
    m_adptContactMgmt.Fill(dsContacts, "Contacts");

    //(5) Display the table in a data grid using data binding.
    DataGrid1.DataSource = dsContacts.Tables["Contacts"].DefaultView;
    Datagrid1.DataBind();
}
```

# Using Data Sets on Web Forms

- Most ASP.NET server controls support data binding , which is a way to link data in your application such as datasets, to properties of a control.
- In particular, the DataGrid and DataList server controls are specifically designed to make displaying data sets on a web form easy and efficient.

# Displaying a Data Set in a DataGrid Control

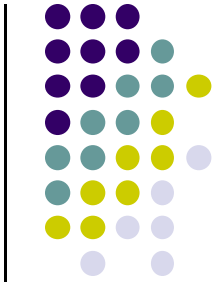
- Create a db connection, adapter and dataset objects.
- Add a DataGrid control to the web form.
- Set the DataSource property of the DataGrid control to the name of the data set.
- Add code to the web form's Page_Load event procedure to fill the data set from the adapter and to bind the data from the dataset to the DataGrid control.



- Following event procedure creates a data set from the Contacts database's Contacts table and then displays the DataGrid control.

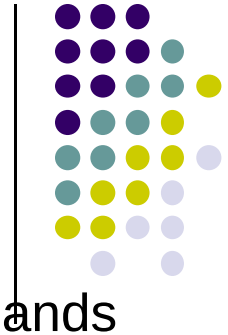
```
private void Page_Load (object sender. EventArgs e)
{
//Fill the data set.
adptContacts.Fill(dsContacts1);
//Bind the data to the DataGrid control.
grdContacts.DataBind();
}
```

# Displaying a Data Set in a DataList Control



- Displays info from a Data Set as a list of rows, rather than rows and columns as in a DataGrid control.
- To display a data set in a DataList control, follow these steps:
  - Create the database connecton, adapter and data set objects.
  - Add a DataList control in a Web Form.
  - Set the DataSource property of the DataList control to the name of the data set.
  - Add code to the web form's Page_Load event procedure to fill the data set from the adapter and to bind the data from the data set to the DataList control.
  - Edit the DataList control's Header, item and separator templates to create the appearance of the DataList.

# Executing Commands on a Database

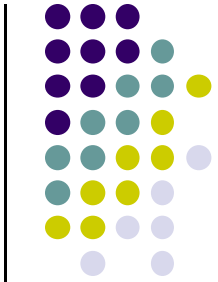


- In addition to working on the datasets, you can perform commands directly on a database connection. The database connection object provides these three command methods:

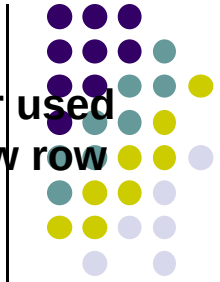
  - ExecuteScalar: Performs query commands that returns a single value, such as counting the number of records in a table.

  - ExecuteNonQuery: Performs commands that change the database but do not return a specific value, including adding and deleting items from a database. The ExecuteNonQuery method returns the number of rows affected by the command.

  - ExecuteReader: Reads records sequentially from the database.

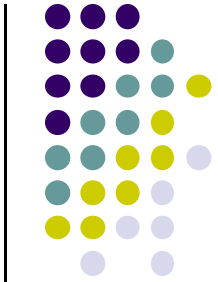


- To use these command methods, follow these steps:
  - Create a connection to the database.
  - Open the connection.
  - Create a command object containing the SQL command or stored procedure to execute.
  - Execute the method on the command object.
  - Close the database connection.
- Always use exception handling to ensure that the database connection is closed whether or not the command succeeds.
- Call the connection's Close method from a *finally* exception-handling clause.



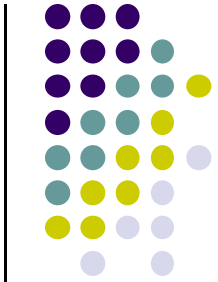
**Following code executes a SQL MAX function to retrieve the highest number used for ContactID and then uses that value as the seed value for adding a new row to the Contacts table.**

```
SqlConnection connContacts = new SqlConnection("integrated security = SSPI; data
    source=(local);initial catalog = Contacts");

Public void btnAdd_Click(object sender,EventArgs e)
{
try
{
//create a SQL command to get a unique ContactID.
SqlCommand cmdNewID = new SqlCommand("SELECT MAX(ContactID) FROM
    Contacts", connContacts);

//open the database connection.
connContacts.Open();
//Execute the SQL Command.
int intNextID = (int)cmdNewID.ExecuteScalar() +1;
```

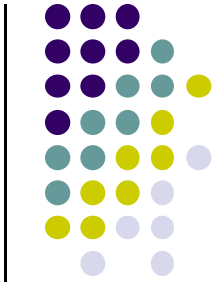


```
//create a command to add a new row.
SqlCommand cmdAddRow = new SqlCommand(String.Format("INSERT INTO
    Contacts(ContactID, FirstName, LastName, WorkPhone) VALUES
    ({0},'{1}','{2}','{3}')", intNextID, txtFirstName.Text, txtLastName.Text,
    txtPhone.Text), connContacts);

//Execute the command.
if(cmdAddRow.ExecuteNonQuery() > 0)
{
    lblMsg.Text = "Record Added.";
    //Database changed, so refresh list.
    RefreshList();
    //clear text boxes.
    ClearText();
}
Else
    {
    lblMsg.Text="Couldn't add record.");
    }
}
```

```
Catch(Exception ex)
    {
    lblMsg.Text = "Couldn't access database due to this error:"+ex.Message);
    }
Finally
    {
    //Close connection.
    connContacts.Close();
    }
}
```

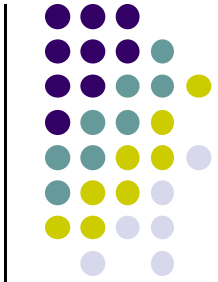


# Stored Procedures - Introduction



- Stored procedures (sprocs) are generally an ordered series of Transact-SQL statements bundled into a single logical unit. They allow for variables and parameters, as well as selection and looping constructs. A key point is that sprocs are stored in the database rather than in a separate file.

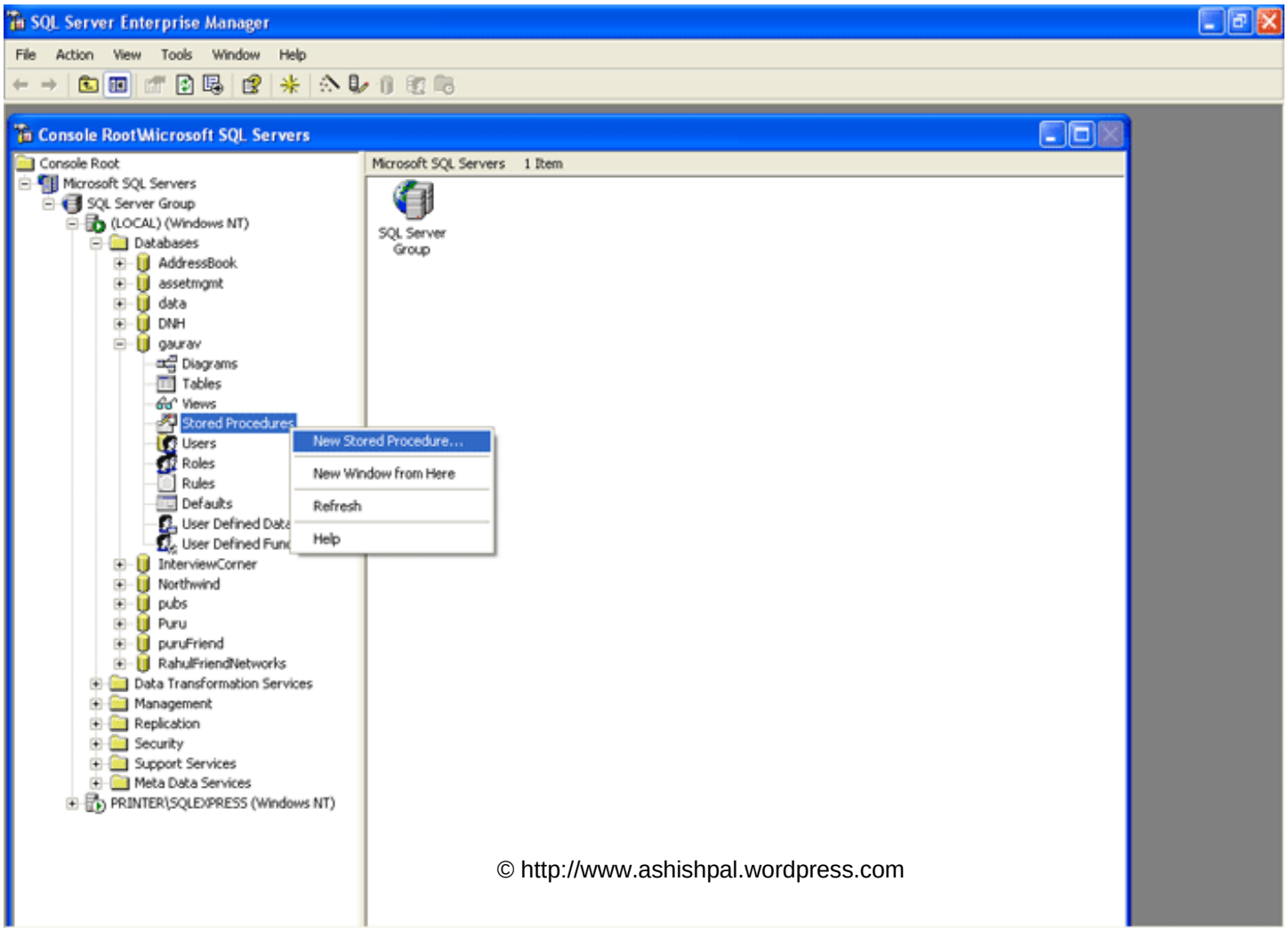
- **Advantages** over simply sending individual statements to the server include:
  - Referred to using short names rather than a long string of text; therefore, less network traffic is required to run the code within the sproc.
  - Pre-optimized and precompiled, so they save an incremental amount of time with each sproc call/execution.
  - Encapsulate a process for added security or to simply hide the complexity of the database.
  - Can be called from other sprocs, making them reusable and reducing code size.



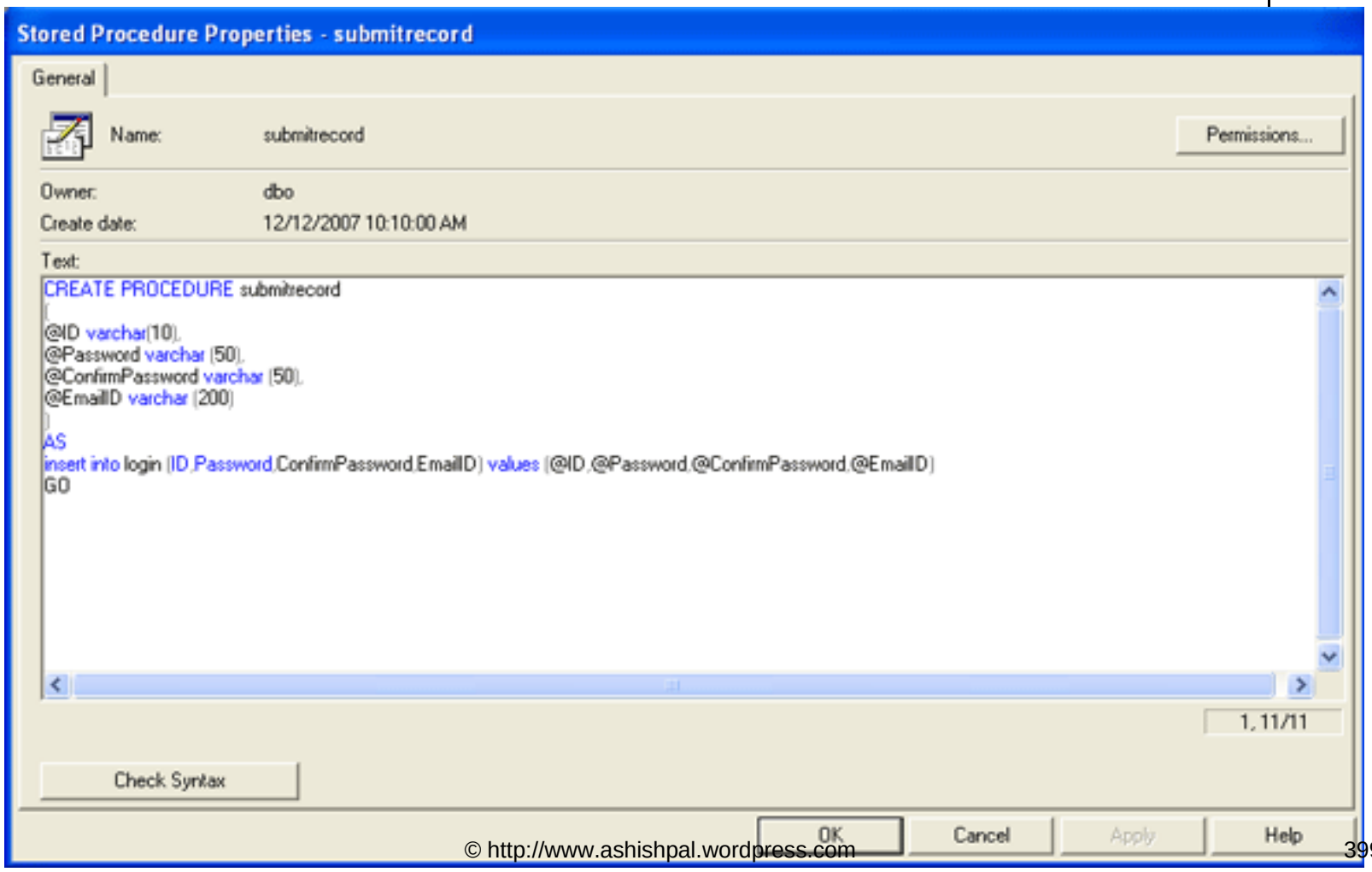
- **Parameterization**
  - A stored procedure gives us some procedural capability, and also gives us a performance boost by using mainly two types of parameters:
    - Input parameters
    - Output parameters
  - From outside the sproc, parameters can be passed in either by position or reference.
- **Declaring Parameters**
  - The name
  - The datatype
  - The default value
  - The direction
- **The syntax is :**
  - @parameter_name [AS] datatype [= default|NULL] [VARYING] [OUTPUT|OUT]

**Let's now create a stored procedure named "Submitrecord".**

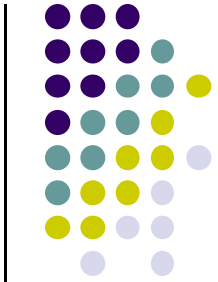
**First open Microsoft SQL Server -> Enterprise Manager, then navigate to the database in which you want to create the stored procedure and select New Stored Procedure.**



**See the below Stored Procedure Properties for what to enter, then click OK.**



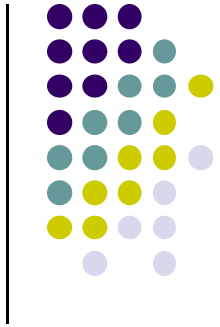
Now create an application named Store Procedure in .net to use the above sprocs.



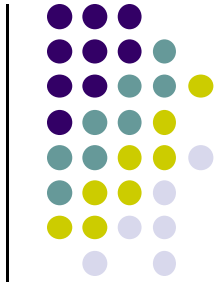
## Stored Procedure.aspx page code

- <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server“> <title>Store Procedure</title> </head>
<body> <form id="form1" runat="server“> 
<asp:Label ID="Label1" runat="server" Text="ID"></asp:Label>
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br /><br />
<asp:Label ID="Label2" runat="server" Text="Password"></asp:Label>
<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox><br /><br />
<asp:Label ID="Label3" runat="server" Text="Confirm Password"></asp:Label>
<asp:TextBox ID="TextBox3" runat="server"></asp:TextBox><br /><br />
<asp:Label ID="Label4" runat="server" Text="Email ID"></asp:Label>
<asp:TextBox ID="TextBox4" runat="server"></asp:TextBox><br /><br /><br />
<asp:Button ID="Button1" runat="server" Text="Submit Record" OnClick="Button1_Click" />
 </form> </body> </html>

## Stored Procedure.aspx.cs page code

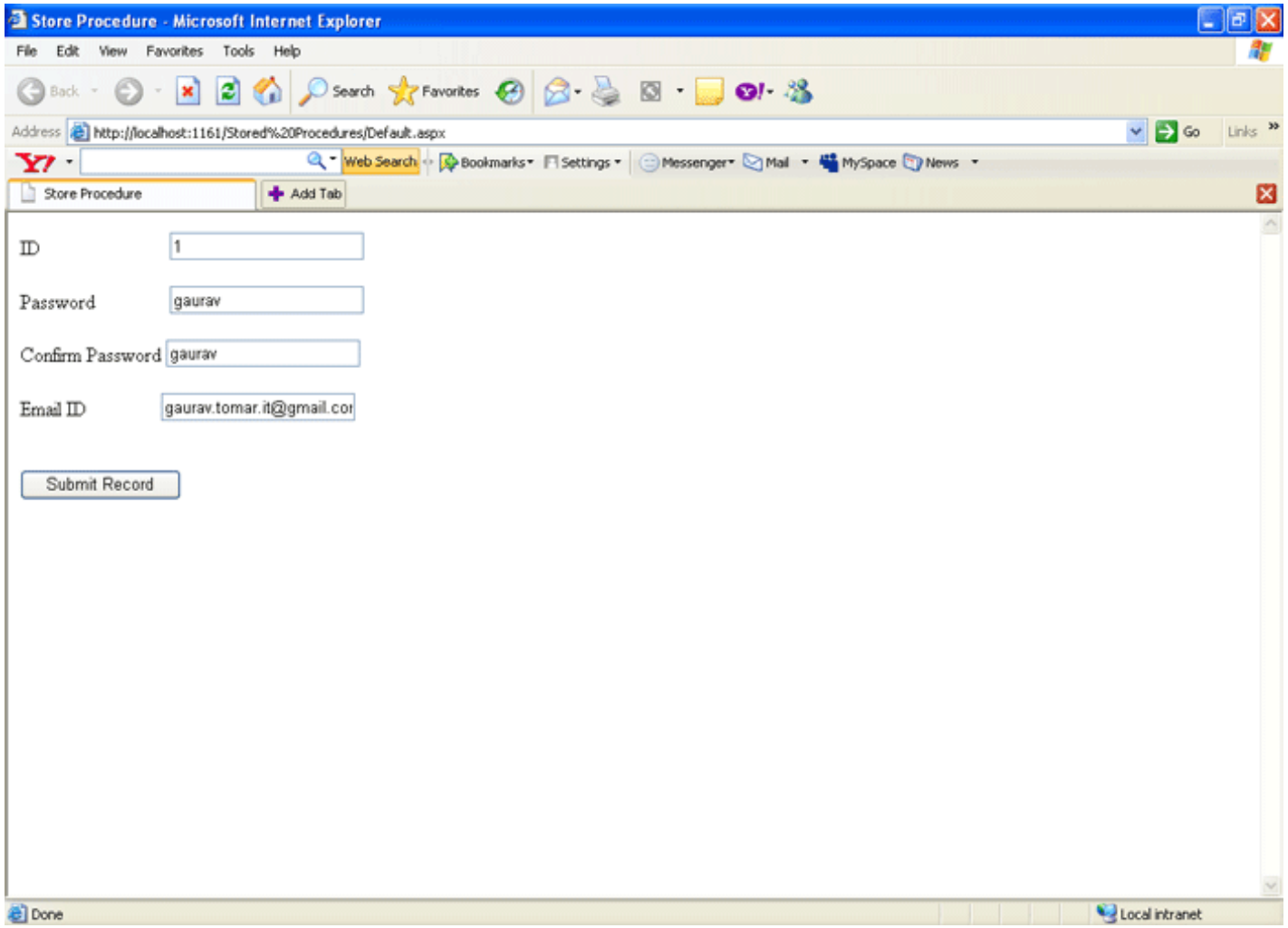


```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.SqlClient;
public partial class _Default : System.Web.UI.Page
{
    DataSet ds = new DataSet();
    SqlConnection con;
    //Here we declare the parameter which we have to use in our application
    SqlCommand cmd = new SqlCommand();
    SqlParameter sp1 = new SqlParameter();
    SqlParameter sp2 = new SqlParameter();
    SqlParameter sp3 = new SqlParameter();
    SqlParameter sp4 = new SqlParameter();
```

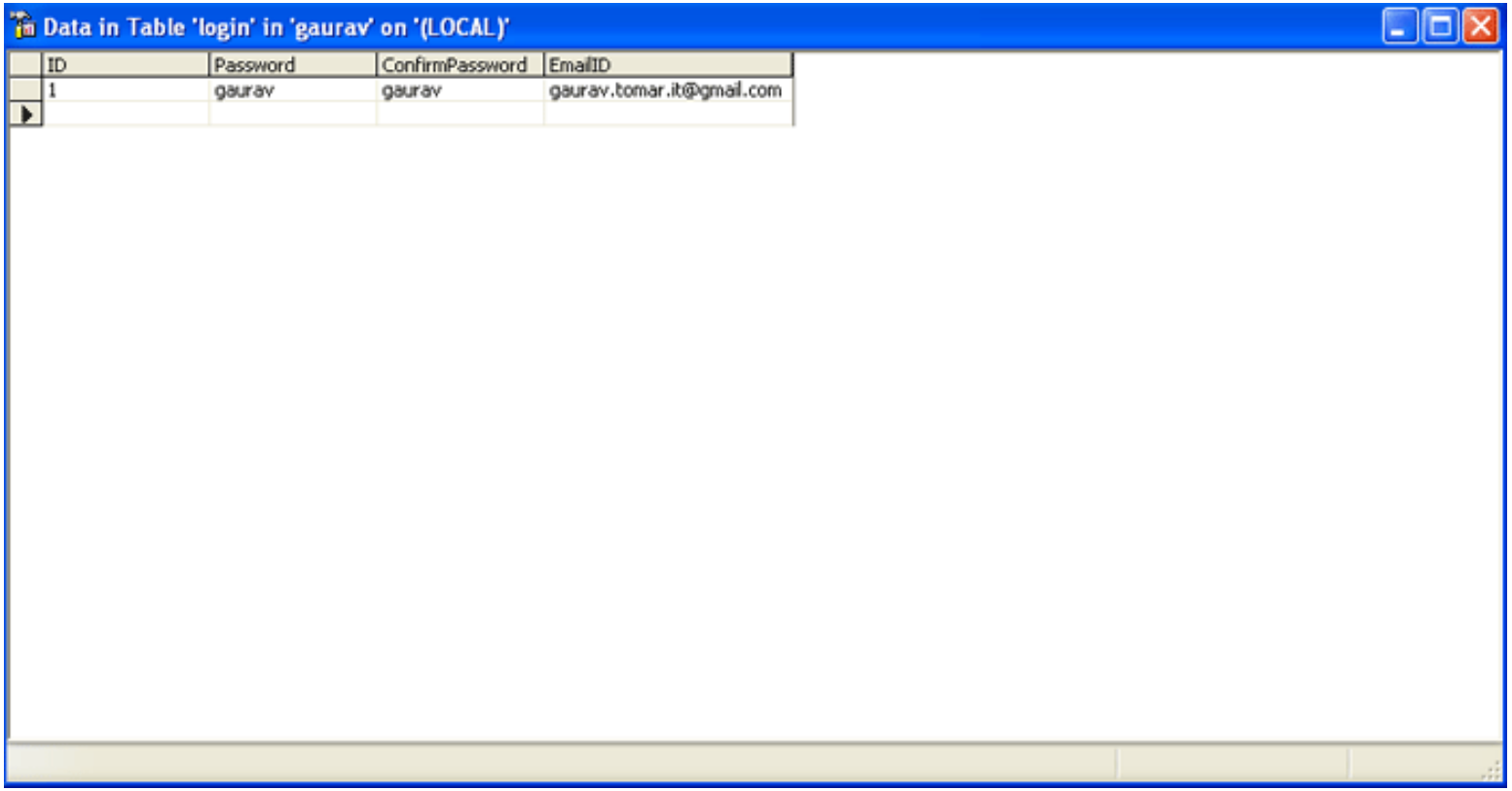


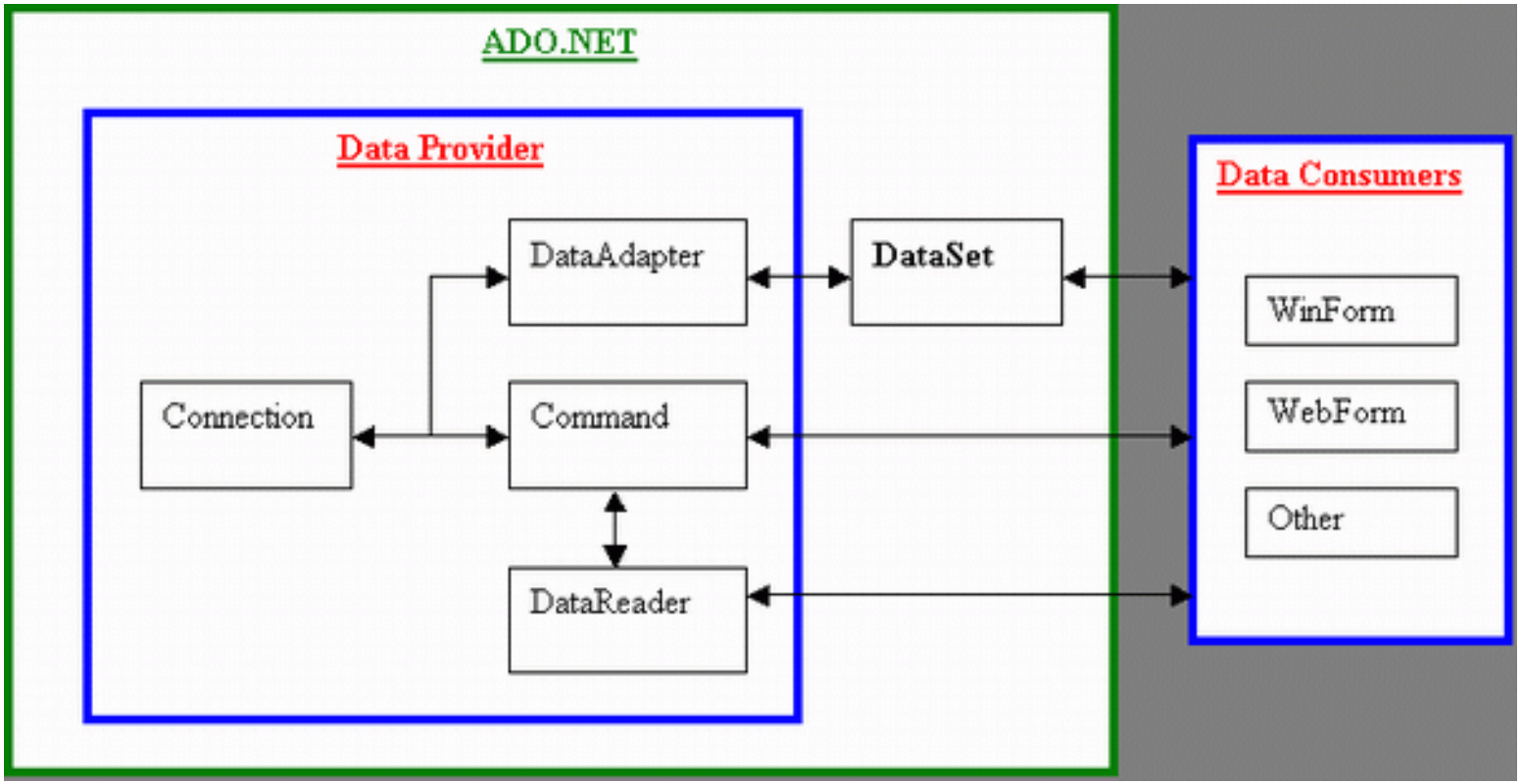
```
protected void Page_Load(object sender, EventArgs e)
    {
    }
protected void Button1_Click(object sender, EventArgs e)
    {
        con = new SqlConnection("server=(local); database=Contacts;
    uid=sa;pwd=");
        cmd.Parameters.Add("@ID", SqlDbType.VarChar).Value =
    TextBox1.Text;
        cmd.Parameters.Add("@Password", SqlDbType.VarChar).Value =
    TextBox2.Text;
        cmd.Parameters.Add("@ConfirmPassword", SqlDbType.VarChar).Value =
    TextBox3.Text;
        cmd.Parameters.Add("@EmailID", SqlDbType.VarChar).Value =
    TextBox4.Text;
        cmd = new SqlCommand("submitrecord", con);
        cmd.CommandType = CommandType.StoredProcedure;
        con.Open();
        cmd.ExecuteNonQuery();
        con.Close();
    }
    }
```

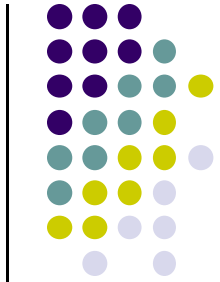
When we run the application, the window will look like this:



After clicking the submit button the data is appended to the database as seen below in the SQL Server table record:




ADO.NET
Data Provider
DataAdapter
DataSet
Connection
Command
DataReader
Data Consumers
WinForm
WebForm
Other



# All the very best for End Term Examination, Folks!!