Table of Contents

[A. Net And C# 4](#_Toc392593506)

[1. 3 Pillars of OOPS Concepts: 4](#_Toc392593507)

[2. General Page Life-Cycle Stages(SILVER) 4](#_Toc392593508)

[3. The key differences between an abstract class and an interface are: 8](#_Toc392593509)

[4. Inheritance can be classified to 5 types. 9](#_Toc392593510)

[5. Namespace and Using 9](#_Toc392593511)

[6. The using Keyword 9](#_Toc392593512)

[7. Interface Definition 9](#_Toc392593513)

[6. Polymorphism: 10](#_Toc392593514)

[Static Polymorphism (Overloading) 10](#_Toc392593515)

[Function Overloading: 10](#_Toc392593516)

[Dynamic Polymorphism (Overriding) 10](#_Toc392593517)

[7. Delegates: 11](#_Toc392593518)

[8. Difference between Different Console read Methods: 11](#_Toc392593519)

[9. What is the difference between Primary key constraint and Unique key constraint? 12](#_Toc392593520)

[10. Different page navigation techniques in asp.net: 12](#_Toc392593521)

[11. Cross Page Posting : 15](#_Toc392593522)

[12. Net Application Vs Non .Net Application: 15](#_Toc392593523)

[13. DLL Hell Problem 16](#_Toc392593524)

[13. Validation Server Controls 17](#_Toc392593525)

[14. Cookies: 18](#_Toc392593526)

[I. Types of Cookies: 18](#_Toc392593527)

[II. Use of Cookies: 19](#_Toc392593528)

[III. Cookies Attribute: 19](#_Toc392593529)

[IV. Implementation: 20](#_Toc392593530)

[V. Limitation of cookies 21](#_Toc392593531)

[VI. Cookie Munging: 21](#_Toc392593532)

[15. ADO.Net Concepts: 22](#_Toc392593533)

[ADO.NET Objects 22](#_Toc392593534)

[I. The SqlCommand Object 23](#_Toc392593535)

[II. The SqlDataReader Object 24](#_Toc392593536)

[III. The DataSet Object 24](#_Toc392593537)

[IV. The SqlDataAdapter Object 25](#_Toc392593538)

[16. HTTP Methods: GET vs. POST 26](#_Toc392593539)

[What is HTTP? 26](#_Toc392593540)

[Two HTTP Request Methods: GET and POST 26](#_Toc392593541)

[The GET Method 26](#_Toc392593542)

[The POST Method 26](#_Toc392593543)

[Compare GET vs. POST 27](#_Toc392593544)

[Other HTTP Request Methods 28](#_Toc392593545)

[17. Difference between web.config and machine.config in asp.net 28](#_Toc392593546)

[Description: 28](#_Toc392593547)

[SQL: 29](#_Toc392593548)

[18. Difference between Union & Union All: 29](#_Toc392593549)

[UNION Example 30](#_Toc392593550)

[Example 30](#_Toc392593551)

[UNION ALL Example 30](#_Toc392593552)

[Example 30](#_Toc392593553)

[19. Difference between having and where clause: 30](#_Toc392593554)

[13. Filtering Groups(Where and Having): 31](#_Toc392593555)

[14. Joins in sql server: 31](#_Toc392593556)

[15. Advanced Joins: 33](#_Toc392593557)

[16. Self-join in sql server 33](#_Toc392593558)

[17. Group By: 34](#_Toc392593559)

[18. Stored procedures: 34](#_Toc392593560)

[19. Advantages of using stored procedures: 36](#_Toc392593561)

[20. Views in sql server: 37](#_Toc392593562)

[21. Indexed views in sql server: 38](#_Toc392593563)

[22. Limitations of views: 39](#_Toc392593564)

[23. Indexes in sql server: 40](#_Toc392593565)

[24. Non Clustered Index: 41](#_Toc392593566)

[25. Summary of the differences between clustered and non-clustered indexes 41](#_Toc392593567)

[26. Unique and Non-Unique Indexes: 42](#_Toc392593568)

[27. DML Triggers: 43](#_Toc392593569)

[28. Instead of insert trigger: 45](#_Toc392593570)

[29. Transactions in SQL Server 45](#_Toc392593571)

[30. Different Types of SQL Server Functions 46](#_Toc392593572)

[Types of Function 47](#_Toc392593573)

[1. System Defined Function 47](#_Toc392593574)

[a) Scalar Function 47](#_Toc392593575)

[b) Aggregate Function 48](#_Toc392593576)

[User Defined Function 49](#_Toc392593577)

[Scalar Function 49](#_Toc392593578)

[Inline Table-Valued Function 50](#_Toc392593579)

[Multi-Statement Table-Valued Function 51](#_Toc392593580)

[33. Difference between Stored Procedure and Function in SQL Server 53](#_Toc392593581)

[Basic Difference 53](#_Toc392593582)

[Advance Difference 53](#_Toc392593583)

[34. Select nth max in column of SQL table: 53](#_Toc392593584)

[35. Comma Separated Column: 54](#_Toc392593585)

[36. Cascading referential integrity constraint: 54](#_Toc392593586)

# Net And C#

## 3 Pillars of OOPS Concepts:

**Encapsulation:**

Encapsulation is the first pillar or principle of object-oriented programming. In simple words, “Encapsulation is a process of binding data members (variables, properties) and member functions (methods) into a single unit”. And Class is the best example of encapsulation.

Through encapsulation a class can hide the internal details of how an object does something. Encapsulation solves the problem at the implementation level.

* A class or structure can specify how accessible each of its members (variables, properties, and methods) is to code outside of the class or structure. Encapsulation simplifies the interaction between objects. An object can use another object without knowing all its data or how its data is maintained. For example, a Client object might have name, address, company, and department properties. If a Bank object wants to use a Client object, it can request the name and address for the bank without needing to know the company and department details of the Client object.
* With the help of encapsulation, a class can change the internal implementation without hurting the overall functionality of the system.
* Encapsulation protects abstraction.

**Inheritance:** Concept of Interface, Abstract Class.

**Polymorphism:** Function overloading, operator overloading, function overriding.

## [General Page Life-Cycle Stages](javascript:void(0))(SILVER)

In general terms, the page goes through the stages outlined in the following table. In addition to the page life-cycle stages, there are application stages that occur before and after a request but are not specific to a page. For more information, see [Introduction to the ASP.NET Application Life Cycle](http://go.microsoft.com/fwlink/?LinkId=133108) and [ASP.NET Application Life Cycle Overview for IIS 7.0](http://msdn.microsoft.com/en-us/library/bb470252.aspx).

Some parts of the life cycle occur only when a page is processed as a postback. For postbacks, the page life cycle is the same during a partial-page postback (as when you use an [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) control) as it is during a full-page postback.

|  |  |
| --- | --- |
| **Stage** | **Description** |
| Page request | The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page), or whether a cached version of the page can be sent in response without running the page. |
| Start(S) | In the start stage, page properties such as [Request](http://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) and [Response](http://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) are set. At this stage, the page also determines whether the request is a postback or a new request and sets the [IsPostBack](http://msdn.microsoft.com/en-us/library/system.web.ui.page.ispostback.aspx) property. The page also sets the [UICulture](http://msdn.microsoft.com/en-us/library/system.web.ui.page.uiculture.aspx) property. |
| Initialization(I) | During page initialization, controls on the page are available and each control's [UniqueID](http://msdn.microsoft.com/en-us/library/system.web.ui.control.uniqueid.aspx) property is set. A master page and themes are also applied to the page if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state. |
| Load(L) | During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state. |
| Postback event handling(V-Validate, E-Event Handling) | If the request is a postback, control event handlers are called. After that, the [Validate](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.basevalidator.validate.aspx) method of all validator controls is called, which sets the [IsValid](http://msdn.microsoft.com/en-us/library/system.web.ui.ivalidator.isvalid.aspx) property of individual validator controls and of the page. (There is an exception to this sequence: the handler for the event that caused validation is called after validation.) |
| Rendering(R) | Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the [Render](http://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) method for each control, providing a text writer that writes its output to the [OutputStream](http://msdn.microsoft.com/en-us/library/system.web.httpresponse.outputstream.aspx) object of the page's [Response](http://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) property. |
| Unload | The [Unload](http://msdn.microsoft.com/en-us/library/system.web.ui.control.unload.aspx) event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as [Response](http://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) and [Request](http://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) are unloaded and cleanup is performed. |

Within each stage of the life cycle of a page, the page raises events that you can handle to run your own code. For control events, you bind the event handler to the event, either declaratively using attributes such as onclick, or in code.

Pages also support automatic event wire-up, meaning that ASP.NET looks for methods with particular names and automatically runs those methods when certain events are raised. If the AutoEventWireup attribute of the [@ Page](http://msdn.microsoft.com/en-us/library/ydy4x04a.aspx) directive is set to true, page events are automatically bound to methods that use the naming convention of Page\_event, such as Page\_Load and Page\_Init. For more information on automatic event wire-up, see [ASP.NET Web Server Control Event Model](http://msdn.microsoft.com/en-us/library/y3bwdsh3.aspx).

The following table lists the page life-cycle events that you will use most frequently. There are more events than those listed; however, they are not used for most page-processing scenarios. Instead, they are primarily used by server controls on the ASP.NET Web page to initialize and render themselves. If you want to write custom ASP.NET server controls, you need to understand more about these events. For information about creating custom controls, see [Developing Custom ASP.NET Server Controls](http://msdn.microsoft.com/en-us/library/zt27tfhy.aspx).

|  |  |
| --- | --- |
| **Page Event** | **Typical Use** |
| [PreInit](http://msdn.microsoft.com/en-us/library/system.web.ui.page.preinit.aspx) | Raised after the start stage is complete and before the initialization stage begins.  Use this event for the following:   * Check the [IsPostBack](http://msdn.microsoft.com/en-us/library/system.web.ui.page.ispostback.aspx) property to determine whether this is the first time the page is being processed. The [IsCallback](http://msdn.microsoft.com/en-us/library/system.web.ui.page.iscallback.aspx) and [IsCrossPagePostBack](http://msdn.microsoft.com/en-us/library/system.web.ui.page.iscrosspagepostback.aspx) properties have also been set at this time. * Create or re-create dynamic controls. * Set a master page dynamically. * Set the [Theme](http://msdn.microsoft.com/en-us/library/system.web.ui.page.theme.aspx) property dynamically. * Read or set profile property values.   Note**Note:**  If the request is a postback, the values of the controls have not yet been restored from view state. If you set a control property at this stage, its value might be overwritten in the next event. |
| [Init](http://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) | Raised after all controls have been initialized and any skin settings have been applied.  The [Init](http://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) event of individual controls occurs before the [Init](http://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) event of the page.  Use this event to read or initialize control properties. |
| [InitComplete](http://msdn.microsoft.com/en-us/library/system.web.ui.page.initcomplete.aspx) | Raised at the end of the page's initialization stage. Only one operation takes place between the [Init](http://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) and [InitComplete](http://msdn.microsoft.com/en-us/library/system.web.ui.page.initcomplete.aspx) events: tracking of view state changes is turned on. View state tracking enables controls to persist any values that are programmatically added to the [ViewState](http://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) collection. Until view state tracking is turned on, any values added to view state are lost across postbacks. Controls typically turn on view state tracking immediately after they raise their [Init](http://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) event.  Use this event to make changes to view state that you want to make sure are persisted after the next postback. |
| [PreLoad](http://msdn.microsoft.com/en-us/library/system.web.ui.page.preload.aspx) | Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the [Request](http://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) instance. |
| [Load](http://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) | The [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object calls the [OnLoad](http://msdn.microsoft.com/en-us/library/system.web.ui.control.onload.aspx) method on the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object, and then recursively does the same for each child control until the page and all controls are loaded.  The [Load](http://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) event of individual controls occurs after the [Load](http://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) event of the page.  Use the [OnLoad](http://msdn.microsoft.com/en-us/library/system.web.ui.control.onload.aspx) event method to set properties in controls and to establish database connections. |
| Control events | Use these events to handle specific control events, such as a [Button](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.aspx) control's [Click](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.click.aspx) event or a [TextBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx) control's [TextChanged](http://msdn.microsoft.com/en-us/library/system.web.ui.mobilecontrols.textbox.textchanged.aspx) event.  Note**Note**  In a postback request, if the page contains validator controls, check the [IsValid](http://msdn.microsoft.com/en-us/library/system.web.ui.page.isvalid.aspx) property of the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) and of individual validation controls before performing any processing. |
| [LoadComplete](http://msdn.microsoft.com/en-us/library/system.web.ui.page.loadcomplete.aspx) | Raised at the end of the event-handling stage.  Use this event for tasks that require that all other controls on the page be loaded. |
| [PreRender](http://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) | Raised after the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object has created all controls that are required in order to render the page, including child controls of composite controls. (To do this, the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object calls [EnsureChildControls](http://msdn.microsoft.com/en-us/library/system.web.ui.control.ensurechildcontrols.aspx) for each control and for the page.)  The [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object raises the [PreRender](http://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) event on the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object, and then recursively does the same for each child control. The [PreRender](http://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) event of individual controls occurs after the [PreRender](http://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) event of the page.  Use the event to make final changes to the contents of the page or its controls before the rendering stage begins. |
| [PreRenderComplete](http://msdn.microsoft.com/en-us/library/system.web.ui.page.prerendercomplete.aspx) | Raised after each data bound control whose [DataSourceID](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.databoundcontrol.datasourceid.aspx) property is set calls its [DataBind](http://msdn.microsoft.com/en-us/library/system.web.ui.control.databind.aspx) method. |
| [SaveStateComplete](http://msdn.microsoft.com/en-us/library/system.web.ui.page.savestatecomplete.aspx) | Raised after view state and control state have been saved for the page and for all controls. Any changes to the page or controls at this point affect rendering, but the changes will not be retrieved on the next postback. |
| [Render](http://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) | This is not an event; instead, at this stage of processing, the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object calls this method on each control. All ASP.NET Web server controls have a [Render](http://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) method that writes out the control's markup to send to the browser.  If you create a custom control, you typically override this method to output the control's markup. However, if your custom control incorporates only standard ASP.NET Web server controls and no custom markup, you do not need to override the [Render](http://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) method. For more information, see [Developing Custom ASP.NET Server Controls](http://msdn.microsoft.com/en-us/library/zt27tfhy.aspx).  A user control (an .ascx file) automatically incorporates rendering, so you do not need to explicitly render the control in code. |
| [Unload](http://msdn.microsoft.com/en-us/library/system.web.ui.control.unload.aspx) | Raised for each control and then for the page.  In controls, use this event to do final cleanup for specific controls, such as closing control-specific database connections.  For the page itself, use this event to do final cleanup work, such as closing open files and database connections, or finishing up logging or other request-specific tasks.  Note**Note**  During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream. If you attempt to call a method such as the Response.Write method, the page will throw an exception. |

## The key differences between an [abstract class](http://us2.php.net/abstract) and an [interface](http://us2.php.net/interface) are:

* **Abstract** classes can have consts, members**, method stubs and defined methods**, whereas **interfaces** can **only** have **consts and methods stubs**.
* Methods and members of an **abstract class can be defined with any visibility**, whereas all methods of an **interface must be defined as public**.
* When inheriting an abstract class, the child class must define the abstract methods, whereas an interface can extend another interface and methods don't have to be defined.
* A child class can only extend a single abstract (or any other) class, **whereas an interface can extend or a class can implement multiple other interfaces.**
* A child class can define abstract methods with the same or less restrictive visibility, whereas a class implementing an interface must define the methods with the exact same visibility.
* Protected might be the way that the framework enforces the mechanism. But the framework is preventing you from doing something that doesn't make sense.
* An abstract class is a class that contains some methods that are not yet implemented. They must be implemented by **a derived 'concrete' class**. As such, we can say that an abstract class is 'incomplete'. If you were to instantiate one, it will have 'holes' where those methods are. The framework is preventing you from building a class with holes in it. **Hence you can only instantiate a concrete class.**

## Inheritance can be classified to 5 types.

* Single Inheritance (one father one child)
* Hierarchical Inheritance (one father multiple child, child have their multiple children)
* Multi-Level Inheritance (one father one child, child has one child)
* Hybrid Inheritance (one father one child, child has two children)
* Multiple Inheritance (two father one child)—**Not Possible in C#**

## Namespace and Using

A namespace is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace will not conflict with the same class names declared in another.

namespace namespace\_name

{

// code declarations(Contains Classes and functions)

}

To call the namespace-enabled version of either function or variable, prepend the namespace name as follows:

namespace\_name.item\_name;

## The using Keyword

The **using** keyword states that the program is using the names in the given namespace. For example, we are using the **System** namespace in our programs. The class Console is defined there.

## Interface Definition

An interface is defined as a **syntactical contract** that all the classes inheriting the interface should follow. The interface defines the 'what' part of the syntactical contract and the deriving classes define the 'how' part of the syntactical contract.

Interfaces define properties, methods and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members. It often helps in providing a standard structure that the deriving classes would follow.

Abstract classes to some extent serve the same purpose, however, they are mostly used when only few methods are to be declared by the base class and the deriving class implements the functionalities.

public interface ITransactions

## Polymorphism:

The word polymorphism means having many forms. In object-oriented programming paradigm, polymorphism is often expressed as 'one interface, multiple functions'.

Polymorphism can be static or dynamic. In static polymorphism the response to a function is determined at the compile time. In dynamic polymorphism, it is decided at run-time.

## Static Polymorphism (Overloading)

The mechanism of linking a function with an object during **compile time** is called **early binding**. It is also called static binding. C# provides two techniques to implement static polymorphism. These are:

* Function overloading
* Operator overloading

## Function Overloading:

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the **types and/or the number of arguments in the argument list**. You cannot overload function declarations that differ only by return type.

## Dynamic Polymorphism (Overriding)

C# allows you to create abstract classes that are used to provide partial class implementation of an interface. Implementation is completed when a derived class inherits from it. Abstract classes contain abstract methods **(Marked by Virtual Keyword if not abstract),** which are implemented by the derived class. The derived classes have more specialized functionality **(Using Override keyword).**

Please note the following rules about abstract classes:

* **You cannot create an instance of an abstract class.**
* You cannot declare an abstract method outside an abstract class.
* When a **class is declared sealed**, it **cannot be inherited**, **abstract classes cannot be declared sealed.**

## Delegates:

A delegate is a reference type variable that holds the reference to a method. The **reference can be changed at runtime.**

Delegates are especially **used for implementing events and the call-back methods**. All delegates are implicitly derived from the **System.Delegate** class.

Declaring Delegates:

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which have the same signature as that of the delegate.

For example, consider a delegate: **(Signature is like a Method, an extra delegate keyword is used.)**

public delegate int MyDelegate (string s);

Instantiating Delegates:

public delegate void printString(string s);

...

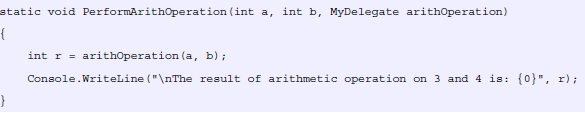
printString ps2 = new prinString(WriteToFile);

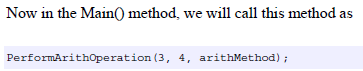
printString ps1 = new printString(WriteToScreen);

**(Add is the method name)**



Passing Delegates to a method:





## Difference between Different Console read Methods:

**Console.Read();  
Console.Readline();  
Console.ReadKey();**  
Now Lets see what is the differences in all these methods:  
 **1. Console.Read():--** method accept the String and **return the string** as well.  
 **2. Console.ReadLine():--**method accept the **String but return Integer.**  
 **3. Console.ReadKey():--**method **accept the Character and also return Character**.   
  
That's why we mostly use the Console.ReadKey() method, for coming back to source code from output window .   
  
Because when we only press the character we directly come on source code. If you will use the Console.Read() and Console.ReadLine method then   
you need to press Enter, to come back to the source code rather than any character.

## What is the difference between Primary key constraint and Unique key constraint?

**This question is asked very frequently in interviews.  
1.** A table can have only one primary key, but more than one unique key  
**2.** Primary key does not allow nulls, where as unique key allows one null

## Different page navigation techniques in asp.net:

* + 1. **Hyperlink control.**
    2. **Response.Redirect**
    3. **Server.Transfer**
    4. **Server.Execute**
    5. **Cross-Page postback**
    6. **Window.Open**

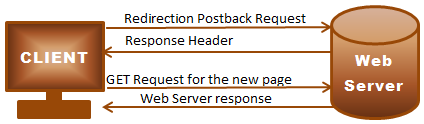
**1. Hyperlink control- Uses** NavigateURL property.

The Hyperlink control does not expose any server side events

**2. Response.Redirect()**

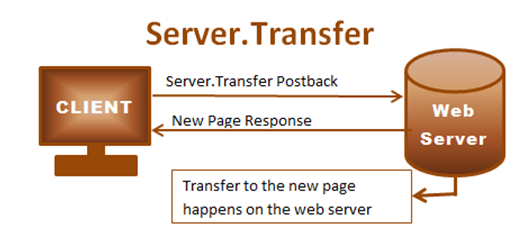
**if you want to intercept a click event** use the Button, LinkButton or the ImageButton server control. In the button click event, call Response.Redirect() method.

When the user clicks the button, the web server receives, a request for redirection. The server then sends a response header to the client. The client then automatically issues a new GET request to the web server. The web server will then serve the new page. So, in short, Response.Redirect **causes 2 request/response cycles.**

note that when Response.Redirect is used the **URL in the address bar changes** and the browser history is maintained.  
Response.Redirect() can be used to navigate pages/websites on the same web server or on a different web server  


**3. Server.Transfer()**

**The following are the differences between Server.Transfer and Response.Redirect**  
**1.** Just like hyperlink and Response.Redirect, Server.Transfer is used to navigate to other pages/sites running on the same web server.  
**2.** Server.Transfer cannot be used to navigate to sites/pages on a different web server.  
**3.** Server.Transfer **does not change the** **URL in the address bar**  
**4.** Server.Transfer is faster than Response.Redirect as the redirection happens on the server in one Request/Response cycle. Response.Redirect() involves 2 Request/Response cycles.  
**5.** With Server.Transfer the Form Variables from the original request are preserved.



* 1. **Server.transfer URL does not change, it can’t be used for redirection to the other servers.**

1. **Server.execute**

**Server.Transfer and Server.Execute are similar in many ways.**  
**1.** The URL in the browser remains the first page URL.  
**2.** Server.Transfer and Server.Execute can only be used to navigate to sites/pages on the same web server. Trying to navigate to sites/pages on a different web server, causes runtime exception.  
**3.** Server.Transfer and Server.Execute preserves the Form Variables from the original request.  
  
**The major difference between Server.Transfer and Server.Execute is that**, Server.Transfer terminates the execution of the current page and starts the execution of the new page, whereas Server.Execute process the second Web form without leaving the first Web form. After completing the execution of the first webform, the control returns to the second webform

1. **Cross page posting**

Cross page posting allows to post one page to another page. By default, when you click a button, the webform posts to itself. If you want to post to another webform on a button click, set the PostBackUrl of the button, to the page that you want to post to.

## Cross Page Posting :

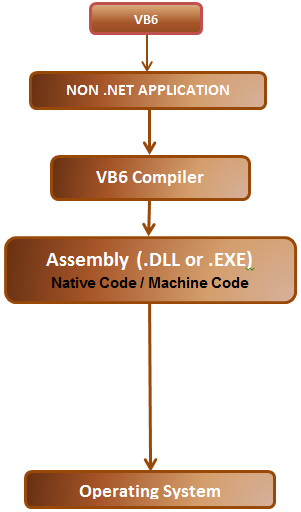
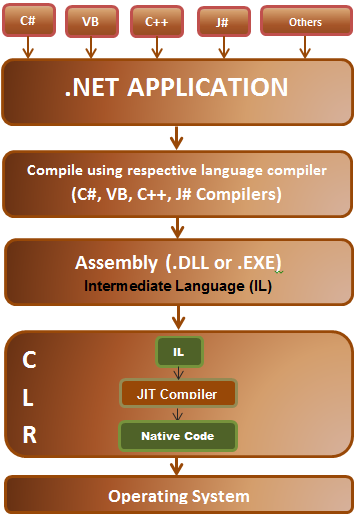
Cross page posting allows to post one page to another page. By default, when you click a button, the webform posts to itself. If you want to post to another webform on a button click**, set the PostBackUrl of the button**, to the page that you want to post to.

|  |
| --- |
| <asp:Button ID="btnCrossPagePostback" runat="server"  Text="Cross Page Postback - WebForm2"  Width="250px" **PostBackUrl="~/WebForm2.aspx"/>** |

**.aspx.cs code**: **Page.IsCrossPagePostBack** Property is used to indicate whether the page is involved in a cross-page postback.

protected void Page\_Load(object sender, EventArgs e)  
{  
**Page previousPage = Page.PreviousPage;  
if (previousPage != null && previousPage.IsCrossPagePostBack)**  
{  
lblName.Text = ((TextBox)previousPage.FindControl("txtName")).Text;  
lblEmail.Text = ((TextBox)previousPage.FindControl("txtEmail")).Text;  
}  
else  
{  
lblStatus.Text = "Landed on this page using a technique other than cross page post back";  
}  
}  
  
The problem with FindControl() method is that, if you mis-spell the **ControlID**, we could get a runtime NullRefernceException. In the next video we will discuss about obtaining a strongly typed reference to the previous page, which can avoid NullRefernceExceptions.

## Net Application Vs Non .Net Application:

## DLL Hell Problem

|  |
| --- |
| http://1.bp.blogspot.com/-GdfFIGaLmQs/TeU4FcNPW6I/AAAAAAAAAGk/lPEo1eZNHm0/s1600/DllHell.png |

**1.** I have 2 applications, A1 and A2 installed on my computer.   
  
**2.** Both of these applications use shared assembly shared.dll  
  
**3.** Now, I have a latest version of Application - A2 available on the internet.  
  
**4.** I download the latest version of A2 and install it on my machine.  
  
**5.** This new installation has over written Shared.dll, which is also used by Application - A1.  
  
**6.** Application - A2 works fine, but A1 fails to work, because the newly installed Shared.dll is not backward compatible.  
  
So, DLL HELL is a problem where one application will install a new version of the shared component that is not backward compatible with the version already on the machine, causing all the other existing applications that rely on the shared component to break. With .NET versioning we do not have DLL HELL problem anymore.

**To overcome this Problem** we can make the Shared Assembly as Strong Named Assembly

With the version name, now if any New Version is installed in GAC that should also be A Strong name Assembly but now with the different version Number. So in that case the older version and newer version both will be available in GAC and hence in result we will not face any Dll hell problem. Application A1 will be continuing to use the previous version while Application A2 will using newer Version of the shared Assembly.

# Validation Server Controls

A Validation server control is used to validate the data of an input control. If the data does not pass validation, it will display an error message to the user.

The syntax for creating a Validation server control is:

< asp:control\_name id="some\_id" runat="server" />

|  |  |
| --- | --- |
| **Validation Server Control** | **Description** |
| [CompareValidator](http://www.w3schools.com/aspnet/control_comparevalidator.asp) | Compares the value of one input control to the value of another input control or to a fixed value |
| [CustomValidator](http://www.w3schools.com/aspnet/control_customvalidator.asp) | Allows you to write a method to handle the validation of the value entered |
| [RangeValidator](http://www.w3schools.com/aspnet/control_rangevalidator.asp) | Checks that the user enters a value that falls between two values |
| [RegularExpressionValidator](http://www.w3schools.com/aspnet/control_regularexpvalidator.asp) | Ensures that the value of an input control matches a specified pattern |
| [RequiredFieldValidator](http://www.w3schools.com/aspnet/control_reqfieldvalidator.asp) | Makes an input control a required field |
| [ValidationSummary](http://www.w3schools.com/aspnet/control_validationsummary.asp) | Displays a report of all validation errors occurred in a Web page |

# Cookies:

Cookies are the small text files that the Web server writes on the client machine when the client's browser accesses their web site. Cookies can be stored in plain text or can be stored in encrypted form.

Since web applications are stateless we need some way to manage the state of the current client request. The state management can be done at server side or client side. Cookies are actually used to identify the users and facilitate the state management.

Note: There are various other state management techniques. Cookies are client side state management techniques and there are other client side state management techniques available too.

# Types of Cookies:

Cookies can be classified into various types based on their lifetime behavior and the domain they are stored for. Major type of cookies is:

1. Session Cookies
2. Persistent Cookies
3. Secure Cookies
4. Third Party Cookies

**Session Cookies**: This cookie lives in memory of the client computer and its lifetime depends on the current browser session. If the user closes the browser these cookies are deleted from the client machine. If the user visits the website again after closing the browser these cookies will not be available.

**Persistent Cookies**: Persistent cookies are the cookies that are stored on the secondary storage of the client machine. These cookies do not depend on the browser session. If the user closes the browser and then access the website again then these cookies will still be available. The lifetime of these cookies are specified in cookies itself (as expiration time). The maximum age of such cookies could be 1 year.

**Secure Cookies**: These cookies have an attribute for security. There cookies can only be accessed by the HTTPS connections and not HTTP connections. The reason for having this type of cookie is that it lessen the chances of cookie stealing/eavesdropping(more on this later in the article)

**HttpOnly Cookies**: This mode will allow the cookie to be accessed using HTTP or HTTPS requests. Such cookies will not be accessible by any other methods(JavaScript APIs for instance)

**Third Party Cookies**: First party cookies are the cookies which set the domain of the cookie same as the domain or sub-domain of the website that is being browsed. Third Party Cookies on the other hand are the cookies with domain set to different domain then the website being browsed. These cookies are mainly used for tracking user browsing patterns and/or finding the Advertisement recommendations for the user.

# Use of Cookies:

**State Management (Session Management)**

Web Page Personalization: Web page personalization can also be achieved using cookies. User can set there personalization preferences and these preferences can be saved on server. Using cookies we can identify the same user and then load the personalized version for him.

**Tracking User:**

Cookies are also user to track the user browsing patterns. This is mainly done to identify whether the user is visiting the site for the first time or is he a returning user. Also This is being done to find the Ad recommendations for the user.

# Cookies Attribute:

**Secure:** When this attribute is specified the cookie can only be accessed over HTTP. This reduces the

chances of cookies getting stolen or eavesdropping of cookies.

**Domain and Path:** These two attributes are to identify the web site and the particular URL of that website for which this cookie is being set.

**HTTPOnly:** using this attribute the cookies are forced to be used over HTTP or HTTPS only. This reduced the chances of cross site scripting because the JavaScript APIs will not be able to access cookies.

**Expires:** This attribute specifies whether the cookie is persistent or non persistent. If we don't specify this attribute the cookie will be non persistent i.e. closing the browser will remove the cookie from the browser memory. If this attribute is specified then the cookie will be written on the client machine and will be valid till the time specified in this attribute is reached.

# Implementation:

|  |
| --- |
| **HttpCookie cookie = new HttpCookie("UName");**  ***//Set the cookie value***  **cookie.Value = TextBox1.Text;**  ***//make it a persistant cookie by setting the expiration time***  **cookie.Expires = DateTime.Now.AddDays(1);**  ***//Push the cookie to the client computer.***  **Response.Cookies.Add(cookie);**  **Label1.Text = TextBox1.Text;**  **if (Request.Cookies["UName"] != null)**  **{**  **Label1.Text = Request.Cookies["UName"].Value;**  **}**  **else**  **{**  **Label1.Text = "Guest";**  **}** |

If I want to see the actual cookie file then I can find that in the "C:\Document and Settings\USERNAME\Cookies" folder. To view these files I have to set the folder options to view the hidden files and operating system files.

# Limitation of cookies

* Since cookies data travel over the network between client and server, there are chances that the cookie can be intercepted in between and someone can use it to mimic our session on the server.
* In case of persistent cookies we should never save the sensitive data in cookies as they can be read by anyone.
* Another limitation of using cookies is their size. The browsers often limit the size of the cookie file (4MB in most cases) which is why we should avoid storing large data in cookies.

# Cookie Munging:

Now we have seen that the most crucial use of cookies in ASP.NET framework is in tracking sessions

and implementing Forms authentication. Now what if the user has disabled cookies in his browser.

If the user has disabled the cookies then the ASP.NET framework uses the URL to keep track of session and authentication data. the unique session ID is then put in the urls and used to track the user session. If the web page contain links i.e. hrefs then the same session ID will also be associated with all the href links. This process in ASP.NET terminology known as cookie munging.

## ADO.Net Concepts:

### ADO.NET Objects

The connection helps identify the database server, the database name, user name, password

#### Creating a SqlConnection Object

A SqlConnection is an object, just like any other C# object. Most of the time, you just declare and instantiate the SqlConnection all at the same time, as shown below:

SqlConnection conn = new SqlConnection(

"Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");

|  |  |
| --- | --- |
| **Connection String Parameter Name** | **Description** |
| Data Source | Identifies the server. Could be local machine, machine domain name, or IP Address. |
| Initial Catalog | Database name. |
| Integrated Security | Set to SSPI to make connection with user's Windows login |
| User ID | Name of user configured in SQL Server. |
| Password | Password matching SQL Server User ID. |

#### Connection String in Web.config file:

<connectionStrings>

<add name="NorthwindConnectionString"(String name) connectionString="Data Source=serverName(Name of Server);Initial Catalog=Northwind;Persist Security Info=True;User ID=userName;Password=password"

providerName="System.Data.SqlClient"/>

</connectionStrings>

conn.Open();  
  
// 3. Pass the connection to a command object  
SqlCommand cmd = new SqlCommand("select \* from Customers", conn)

rdr = cmd.ExecuteReader();

while (rdr.Read())  
 {  
 Console.WriteLine(rdr[0]);  
 }  
}  
finally // Confirm that at least this action will occur after try block execution  
{  
// close the reader  
if (rdr != null)  
{  
rdr.Close();  
}  
  
// 5. Close the connection  
if (conn != null)  
{  
conn.Close();  
}  
}  
}  
}

The result set is returned as a *SqlDataReader* and the *while* loop reads the first column from each row of the result set.

# The SqlCommand Object

specify the actions you want to occur . A command object uses a connection object to figure out which database to communicate.

You can use a command object alone, to execute a command directly, or assign a reference to a command object to an SqlDataAdapter, which holds a set of commands that work on a group of data as described below.

string insertString = @"  
insert into Categories  
(CategoryName, Description)  
values ('Miscellaneous', 'Whatever doesn''t fit elsewhere')";  
  
// 1. Instantiate a new command with a query and connection  
SqlCommand cmd = new SqlCommand(insertString, conn);  
  
// 2. Call ExecuteNonQuery to send command  
cmd.ExecuteNonQuery();

cmd.CommandText = deleteString;  
  
// 3. Set the Connection property  
cmd.Connection = conn;

**Getting Single values**

count, sum, average, int count = (int)cmd.ExecuteScalar();

# The SqlDataReader Object

The data reader object allows you to obtain the results of a SELECT statement from a command object. For performance reasons, the data returned from a data reader is a fast forward-only stream of data. This means that you can only pull the data from the stream in a sequential manner This is good for speed, but if you need to manipulate data, then a DataSet is a better object to work with.

A SqlDataReader is a type that is good for reading data in the most efficient manner possible. You can \*not\* use it for writing data. SqlDataReaders are often described as fast-forward firehose-like streams of data.

You can read from SqlDataReader objects in a forward-only sequential manner. Once you've read some data, you must save it because you will not be able to go back and read it again.

, the SqlDataReader returns data via a sequential stream. To read this data, you must pull data from a table row-by-row Once a row has been read, the previous row is no longer available.

SqlDataReader rdr = cmd.ExecuteReader();

The *ExecuteReader* method of the SqlCommand object, *cmd* , returns a SqlDataReader instance.

#### Working with Disconnected Data - The DataSet and SqlDataAdapter

# The DataSet Object

DataSet objects are in-memory representations of data. They contain multiple Datatable objects, which contain columns and rows, just like normal database tables. support disconnected operations on data.

It is the SqlDataAdapter that manages connections with the data source and gives us disconnected behavior. The SqlDataAdapter opens a connection only when required and closes it as soon as it has performed its task. For example, the SqlDataAdapter performs the following tasks when filling a DataSet with data:

1. Open connection
2. Retrieve data into DataSet
3. Close connection

In between the Fill and Update operations, data source connections are closed

Also, if the amount of data is so large that holding it in memory is impractical, you will need to use SqlDataReader for read-only data.

DataSet dsCustomers = new DataSet();

# The SqlDataAdapter Object

Some situations also call for caching data in memory to minimize the number of database calls for data that does not change. The data adapter makes it easy for you to accomplish these things by helping to manage data in a disconnected mode. The data adapter fills a DataSet object when reading the data and writes in a single batch when persisting changes back to the database.

You will have a data adapter defined for each table in a DataSet.

SqlDataAdapter daCustomers = new SqlDataAdapter(  
"select CustomerID, CompanyName from Customers", conn); .

daCustomers.Fill(dsCustomers, "Customers");

dgCustomers.DataSource = dsCustomers; (GridView dgCustomers)  
dgCustomers.DataMember = "Customers";

DataSet can hold multiple tables and this would allow you to expand each available table. To specify exactly which table to use, set the DataGrid's *DataMember* property to the name of the table.

**Updating Changes**

daCustomers.Update(dsCustomers, "Customers");

#### Adding Parameters to Commands

SqlCommand cmd = new SqlCommand(

"select \* from Customers where city = @City", conn);

SqlParameter param = new SqlParameter();

param.ParameterName = "@City";

param.Value = inputCity;

cmd.Parameters.Add(param);

#### Executing a Stored Procedure:

// 1. create a command object identifying

// the stored procedure

SqlCommand cmd = new SqlCommand(

"CustOrderHist", conn);

// 2. set the command object so it knows

// to execute a stored procedure

cmd.CommandType = CommandType.StoredProcedure;

// 3. add parameter to command, which

// will be passed to the stored procedure

cmd.Parameters.Add(

new SqlParameter("@CustomerID", custId));

## HTTP Methods: GET vs. POST

## What is HTTP?

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

## Two HTTP Request Methods: GET and POST

Two commonly used methods for a request-response between a client and server are: GET and POST.

* **GET** - Requests data from a specified resource
* **POST** - Submits data to be processed to a specified resource

## The GET Method

**Note that query strings (name/value pairs) is sent in the URL of a GET request:**

/test/demo\_form.asp**?name1=value1&name2=value2**

**Some other notes on GET requests:**

* GET requests can be cached
* GET requests remain in the browser history
* GET requests can be bookmarked
* GET requests should never be used when dealing with sensitive data
* GET requests have length restrictions
* GET requests should be used only to retrieve data

## The POST Method

**Note that query strings (name/value pairs) is sent in the HTTP message body of a POST request:**

POST /test/demo\_form.asp HTTP/1.1  
Host: w3schools.com  
**name1=value1&name2=value2**

**Some other notes on POST requests:**

* POST requests are never cached
* POST requests do not remain in the browser history
* POST requests cannot be bookmarked
* POST requests have no restrictions on data length

## Compare GET vs. POST

The following table compares the two HTTP methods: GET and POST.

|  |  |  |
| --- | --- | --- |
|  | **GET** | **POST** |
| **BACK button/Reload** | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| **Bookmarked** | Can be bookmarked | Cannot be bookmarked |
| **Cached** | Can be cached | Not cached |
| **Encoding type** | application/x-www-form-urlencoded | application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data |
| **History** | Parameters remain in browser history | Parameters are not saved in browser history |
| **Restrictions on data length** | Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |
| **Restrictions on data type** | Only ASCII characters allowed | No restrictions. Binary data is also allowed |
| **Security** | GET is less secure compared to POST because data sent is part of the URL  Never use GET when sending passwords or other sensitive information! | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| **Visibility** | Data is visible to everyone in the URL | Data is not displayed in the URL |

## Other HTTP Request Methods

The following table lists some other HTTP request methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| HEAD | Same as GET but returns only HTTP headers and no document body |
| PUT | Uploads a representation of the specified URI |
| DELETE | Deletes the specified resource |
| OPTIONS | Returns the HTTP methods that the server supports |
| CONNECT | Converts the request connection to a transparent TCP/IP tunnel |

## Difference between web.config and machine.config in asp.net

### Description:

.Net Framework provides **three types of files** known as **Configuration Files** which contains information for **controlling and managing .Net Applications i.e.Windows(Windows Forms and WPF) and Web Applications(Asp.Net,MVC).** These are **Machine.Config, App.Config and Web.Config.**

These files are **XML based configuration files**.All the information is in the format of **XML** which can be **easily modified and readable by .Net Applications**.

#### Machine.Config:

* This file is the **top in the hierarchy of configuration files**.
* It is a **global settings file** for all the applications that run on the **.Net Framework** on the machine.
* All the .Net application will **use this configuration information in their applications**.
* This file is automatically installed whenever .Net Framework is installed in the System.Only one file per the system. This file is located at **C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG.**
* All the information in this file is **readable by the applications but can not be modified**. This file can be maintained by the **system administrator**.
* Only **one machine configuration file** is exists on server.
* This file stores **configuration information at system level** so it can be called as **machine level configuration file**.

#### Web.Config:

* This file is automatically created when you create **Asp.Net Web Application Project**.
* This file contains configuration information only for **one web application** only.
* All the machine.config file settings can be **inherited** here and also **can be overidable**.
* This file contains **application level configuration** so it can be called as **application level configuration file.**
* One web application **can contain many web.config file**s based on the **number of folders** in the application.
* You can find what **Asp.Net Server Configuration can be stored** here at [Asp.Net Settings Schema at MSDN](http://msdn.microsoft.com/en-us/library/b5ysx397%28VS.71%29.aspx)
* You can store information like **connections strings, domains, application properties,session information, Cookie information,Viewstate information** etc in **web.config** file.

#### Web.Config:

Web.Config files are purely for web applications.

This file is automatically created when we create new web projects.

we can have many web.config files in web project where each folder can contain single web.config file only.

In this files we can able to store global information which can be accessed based on location of the

config file. Whenever we change these file automatically website will restart with modified details.

#### App.Config:

App.Config files are for Windows Application.

This file is not add by default when we create new project. We need to externally add these to file to our project by Right Clicking on project --> Select Add New Item--> Select "Application Configuration file".

Modifying App.Config file will not affect the running application until we restart the application.Only one app.config file is allowed in one project.

# SQL:

# Difference between Union & Union All:

**Note:** UNION cannot be used to list ALL cities from the two tables. If several customers and suppliers share the same city, each city will only be listed once. UNION selects only distinct values. Use UNION ALL to also select duplicate values!

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

# UNION Example

The following SQL statement selects all the **different** cities (only distinct values) from the "Customers" and the "Suppliers" tables:

## Example

SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;

# UNION ALL Example

The following SQL statement uses UNION ALL to select **all** (duplicate values also) cities from the "Customers" and "Suppliers" tables:

## Example

SELECT City FROM Customers  
UNION ALL  
SELECT City FROM Suppliers  
ORDER BY City;

# Difference between having and where clause:

* **Difference between WHERE and HAVING clause:**

1. WHERE clause can be used with - Select, Insert, and Update statements, where as HAVING clause can only be used with the Select statement.

* 2. WHERE filters rows before aggregation (GROUPING), whereas, HAVING filters groups, after the aggregations are performed.
* 3. Aggregate functions cannot be used in the WHERE clause, unless it is in a sub query contained in a HAVING clause, whereas, aggregate functions can be used in Having clause.

BAD SQL:

select employee, sum(bonus) from emp\_bonus

**group by employee where sum(bonus) > 1000;**

**GOOD SQL:**

**select employee, sum(bonus) from emp\_bonus**

**group by employee having sum(bonus) > 1000;**

# Filtering Groups(Where and Having):

WHERE clause is used to filter rows before aggregation, where as HAVING clause is used to filter groups after aggregations. The following 2 queries produce the same result.  
  
Filtering rows using WHERE clause, before aggrgations take place:  
Select City, SUM(Salary) as TotalSalary  
from tblEmployee  
Where City = 'London'  
group by City  
  
Filtering groups using HAVING clause, after all aggrgations take place:  
Select City, SUM(Salary) as TotalSalary  
from tblEmployee  
group by City  
Having City = 'London'

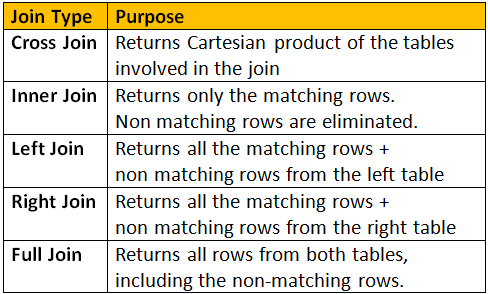
**It is also possible to combine WHERE and HAVING**  
**Select City, SUM(Salary) as TotalSalary**  
**from tblEmployee**  
**Where Gender = 'Male'**  
**group by City**  
**Having City = 'London'**

# Joins in sql server:

**In SQL server, there are different types of JOINS.**  
1. CROSS JOIN  
2. INNER JOIN   
3. OUTER JOIN

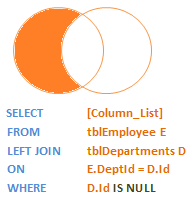
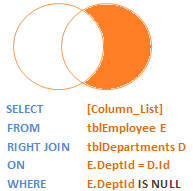
**Outer Joins are again divided into 3 types**  
1. Left Join or Left Outer Join  
2. Right Join or Right Outer Join  
3. Full Join or Full Outer Join

|  |  |  |  |
| --- | --- | --- | --- |
| **CROSS JOIN** | **JOIN or INNER JOIN** | **LEFT JOIN or LEFT OUTER JOIN** | **FULL JOIN** |
| CROSS JOIN, produces the cartesian product of the 2 tables involved in the join. For example, in the Employees table we have 10 rows and in the Departments table we have 4 rows. So, a cross join between these 2 tables produces 40 rows. Cross Join shouldn't have ON clause. | So, in summary, INNER JOIN, returns only the matching rows between both the tables. Non matching rows are eliminated. | **LEFT JOIN**, returns all the matching rows + non matching rows from the left table. In reality, INNER JOIN and LEFT JOIN are extensively used.  **RIGHT JOIN**, returns all the matching rows + non matching rows from the right table | returns all rows from both the left and right tables, including the non matching rows. |
| SELECT Name, Gender, Salary, DepartmentName FROM tblEmployee CROSS JOIN tblDepartment | SELECT Name, Gender, Salary, DepartmentName FROM tblEmployee INNER JOIN tblDepartment ON tblEmployee.DepartmentId = tblDepartment.Id | SELECT Name, Gender, Salary, DepartmentName FROM tblEmployee LEFT OUTER JOIN tblDepartment ON tblEmployee.DepartmentId = tblDepartment.Id | SELECT Name, Gender, Salary, DepartmentName FROM tblEmployee FULL OUTER JOIN tblDepartment ON tblEmployee.DepartmentId = tblDepartment.Id |



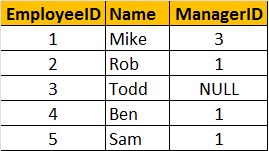
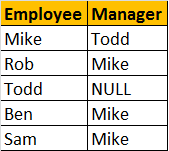
# Advanced Joins:

Retrieve only the non matching rows from the left table  
Retrieve only the non matching rows from the right table  
Retrieve only the non matching rows from both the left and right table

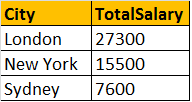
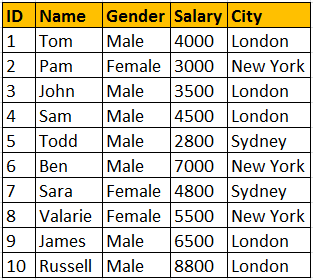
# Self-join in sql server

Select E.Name as Employee, M.Name as Manager  
from tblEmployee E  
Left Join tblEmployee M  
On E.ManagerId = M.EmployeeId

# Group By:

**Group by** clause is used to group a selected set of rows into a set of summary rows by the values of one or more columns or expressions. It is always used in conjunction with one or more aggregate functions.

**Select City, SUM(Salary) as TotalSalary**  
**from tblEmployee Group by City**  


**Note:** If you omit, the group by clause and try to execute the query, you get an error -Column 'tblEmployee.City' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause. 

# Stored procedures:

(Just like a Method used in OOPS)

A stored procedure is group of T-SQL (Transact SQL) statements. If you have a situation, where you write the same query over and over again, you can save that specific query as a stored procedure and call it just by it's name.

Create Procedure spGetEmployees  
as  
Begin  
  Select Name, Gender from tblEmployee  
End

**Note:** When naming user defined stored procedures, Microsoft recommends not to use**"sp\_"** as a prefix. All system stored procedures, are prefixed with **"sp\_"**. This avoids any ambiguity between user defined and system stored procedures and any conflicts, with some future system procedure

**To execute the stored procedure:**

1.spGetEmployees  
2. EXEC spGetEmployees  
3. Execute spGetEmployees

**Creating a stored procedure with input parameters:**

Create Procedure spGetEmployeesByGenderAndDepartment   
@Gender nvarchar(50),  
@DepartmentId int  
as  
Begin  
  Select Name, Gender from tblEmployee Where Gender = @Gender and DepartmentId = @DepartmentId  
End

**Execution:**  
EXECUTE spGetEmployeesByGenderAndDepartment 'Male', 1

Or

EXECUTE spGetEmployeesByGenderAndDepartment @DepartmentId=1, @Gender ='Male'

**To encrypt the text of the SP**, use WITH ENCRYPTION option. Once, encrypted, you cannot view the text of the procedure, using **sp\_helptext** system stored procedure.

Alter Procedure spGetEmployeesByGenderAndDepartment   
@Gender nvarchar(50),  
@DepartmentId int  
WITH ENCRYPTION  
as  
Begin  
  Select Name, Gender from tblEmployee Where Gender = @Gender and DepartmentId = @DepartmentId  
End

To delete the SP, use DROP PROC 'SPName' or DROP PROCEDURE 'SPName'

**Stored procedures with output parameters**

Create Procedure spGetEmployeeCountByGender  
@Gender nvarchar(20),  
@EmployeeCount int Output  
as  
Begin  
 Select @EmployeeCount = COUNT(Id)   
 from tblEmployee   
 where Gender = @Gender  
End

**To execute this stored procedure with OUTPUT parameter**  
  
**1.** First initialise a variable of the **same datatype** as that of the **output parameter**. We have declared @EmployeeTotal integer variable.   
**2.** Then pass the @EmployeeTotal variable to the SP. You have to specify the **OUTPUT**keyword. If you don't specify the OUTPUT keyword, the variable will be **NULL**.   
**3.** Execute

Declare @EmployeeTotal int  
Execute spGetEmployeeCountByGender 'Female', @EmployeeTotal output  
Print @EmployeeTotal

or

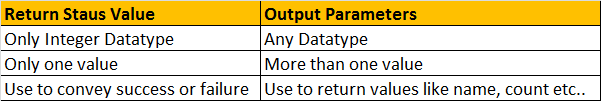
Declare @EmployeeTotal int  
Execute spGetEmployeeCountByGender @EmployeeCount = @EmployeeTotal OUT, @Gender = 'Male'  
Print @EmployeeTotal

# Advantages of using stored procedures:

**The following advantages of using Stored Procedures over adhoc queries (inline SQL)**  
**1. Execution plan retention and reusability** - Stored Procedures are compiled and their execution plan is cached and used again, when the same SP is executed again. Although adhoc queries also create and reuse plan, the plan is reused only when the query is textual match and the datatypes are matching with the previous call. Any change in the datatype or you have an extra space in the query then, a new plan is created.

**2. Reduces network traffic** - You only need to send, EXECUTE SP\_Name statement, over the network, instead of the entire batch of adhoc SQL code.  
  
**3. Code reusability and better maintainability** - A stored procedure can be reused with multiple applications. If the logic has to change, we only have one place to change, where as if it is inline sql, and if you have to use it in multiple applications, we end up with multiple copies of this inline sql. If the logic has to change, we have to change at all the places, which makes it harder maintaining inline sql.  
  
**4. Better Security** - A database user can be granted access to an SP and prevent them from executing direct "select" statements against a table.  This is fine grain access control which will help control what data a user has access to.  
  
**5. Avoids SQL Injection attack** - SP's prevent sql injection attack. [Please watch this video on SQL Injection Attack, for more information.](http://csharp-video-tutorials.blogspot.com/2012/06/sql-injection-attack.html)

**Difference between return values and output parameters**



# Views in sql server:

A view is nothing more than a**saved SQL query**. A view can also be considered as a **virtual table**.

Create View vWEmployeesByDepartment  
as  
Select Id, Name, Salary, Gender, DeptName  
from tblEmployee  
join tblDepartments  
on tblEmployee.DepartmentId = tblDepartment.DeptId  
  
SELECT \* from vWEmployeesByDepartment

The View itself, does not store any data by default.

**Advantages of using views:**

* 1. Views can be used to reduce the **complexity of the database schema**, for non IT users. The sample view, **vWEmployeesByDepartment**, hides the complexity of joins. Non-IT users, finds it easy to query the view, rather than writing complex joins.  
       
     2. Views can be used as a mechanism to implement **row and column level security**.  
     **Row Level Security:**  
     For example, I want an end user, to have access only to IT Department employees. If I grant him access to the underlying tblEmployees and tblDepartments tables, he will be able to see, every department employees. To achieve this, I can create a view, which returns only IT Department employees, and grant the user access to the view and not to the underlying table.

**?) Is it possible to Insert, Update and delete rows**, from the underlying tblEmployees table, using view vWEmployeesDataExceptSalary?

**A. Yes**, SQL server views are updateable.

**The following query updates, Name column from Mike to Mikey**.

Update vWEmployeesDataExceptSalary   
Set Name = 'Mikey' Where Id = 2

**Along the same lines**, it is also possible to insert and delete rows from the base table using views.  
Delete from vWEmployeesDataExceptSalary where Id = 2  
Insert into vWEmployeesDataExceptSalary values (2, 'Mikey', 'Male', 2)

**Note:** conclusion - If a view is based on multiple tables, and if you update the view, it may not update the underlying base tables correctly. To correctly update a view, that is based on multiple table, INSTEAD OF triggers are used.

# Indexed views in sql server:

**What is an Indexed View or What happens when you create an Index on a view?**  
A **standard** or **Non-indexed** view is just a stored SQL query. When, we try to retrieve data from the view, the data is actually retrieved from the underlying base tables. So, a view is just a virtual table it does not store any data, by default.  
  
**However, when you create an index**, on a view, the view gets materialized. This means, the view is now, capable of storing data. In SQL server, we call them Indexed views and in Oracle, Materialized views.

**Script to create view vWTotalSalesByProduct**

Create view vWTotalSalesByProduct  
with SchemaBinding  
as  
Select Name,   
SUM(ISNULL((QuantitySold \* UnitPrice), 0)) as TotalSales,   
COUNT\_BIG(\*) as TotalTransactions  
from dbo.tblProductSales  
join dbo.tblProduct  
on dbo.tblProduct.ProductId = dbo.tblProductSales.ProductId  
group by Name  
  
**If you want to create an Index**, on a view, the following rules should be followed by the view. For the complete list of all rules, please check [MSDN](http://msdn.microsoft.com/en-us/library/ms191432(v=sql.105).aspx).

1. The view should be created with SchemaBinding option  
  
2. If an Aggregate function in the SELECT LIST, references an expression, and if there is a possibility for that expression to become NULL, then, a replacement value should be specified. In this example, we are using, ISNULL () function, to replace NULL values with ZERO.  
  
3. If GROUP BY is specified, the view select list must contain a COUNT\_BIG(\*) expression  
  
4. The base tables in the view, should be referenced with 2 part name. In this example, tblProduct and tblProductSales are referenced using dbo.tblProduct and dbo.tblProductSales respectively.  
  
**Now, let's create an Index on the view:**  
The first index that you create on a view, must be a unique clustered index. After the unique clustered index has been created, you can create additional nonclustered indexes.  
Create Unique Clustered Index UIX\_vWTotalSalesByProduct\_Name  
on vWTotalSalesByProduct(Name) 

the data is retrurned from the view itself, rather than retrieving data from the underlying base tables.

Indexed views, can significantly improve the performance of queries that involves JOINS and Aggeregations. The cost of maintaining an indexed view is much higher than the cost of maintaining a table index.   
  
Indexed views are ideal for scenarios, where the underlying data is not frequently changed. Indexed views are more often used in OLAP systems, because the data is mainly used for reporting and analysis purposes. Indexed views, may not be suitable for OLTP systems, as the data is frequently addedd and changed

# Limitations of views:

**You cannot pass parameters to a view**

**Rules and Defaults cannot be associated with views.**

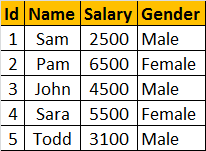
**Views cannot be based on temporary tables.**

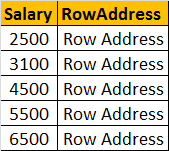
**The ORDER BY clause is invalid in views** unless TOP or FOR XML is also specified.

# Indexes in sql server:

Indexes are used by queries to find data from tables quickly. Indexes are created on tables and views. Index on a table or a view, is very similar to an index that we find in a book.

In fact, the existence of the right indexes, can drastically improve the performance of the query. If there is no index to help the query, then the query engine, checks every row in the table from the beginning to the end. This is called as Table Scan. Table scan is bad for performance. 

CREATE Index IX\_tblEmployee\_Salary    
ON tblEmployee (SALARY ASC)

**The index stores salary of each employee, in the ascending order** as shown below. The actual index may look slightly different.  


**To delete or drop the index:**When dropping an index, specify the table name as well  
Drop Index tblEmployee.IX\_tblEmployee\_Salary

**Clustered Index:**  
A clustered index determines the physical order of data in a table. For this reason, a table can have only one clustered index

Note that **Id**column is marked as **primary key**. Primary key, constraint create **clustered indexes automatically** if no clustered index already exists on the table and a nonclustered index is not specified when you create the PRIMARY KEY constraint.   
**Because of the fact that, a clustered index dictates the physical storage order** of the data in a table, a table can contain only one clustered index.

# Non Clustered Index:

A non-clustered index is analogous to an index in a textbook. The data is stored in one place, the index in another place. The index will have pointers to the storage location of the data. Since, the non-clustered index is stored separately from the actual data, a table can have more than one non clustered index, just like how a book can have an index by Chapters at the beginning and another index by common terms at the end.

Create NonClustered Index IX\_tblEmployee\_Name  
ON tblEmployee(Name)

**Difference between Clustered and NonClustered Index:**  
1. **Only one clustered index per table**, where as you can have more than one non clustered index  
2. **Clustered index is faster than a non clustered index**, because, the non-clustered index has to refer back to the table, if the selected column is not present in the index.  
3. **Clustered index determines the storage order of rows in the table**, and hence doesn't require additional disk space, but whereas a Non Clustered index is stored separately from the table, additional storage space is required.

# Summary of the differences between clustered and non-clustered indexes

**Here’s a summary of the differences:**

* Non clustered indexes store both a value and a pointer to the actual row that holds that value. Clustered indexes don’t need to store a pointer to the actual row because of the fact that the rows in the table are stored on disk in the same exact order as the clustered index – and the clustered index actually stores the row-level data in it’s leaf nodes.
* A clustered index determines the order in which the rows of the table will be stored on disk – and it actually stores row level data in the leaf nodes of the index itself. A non-clustered index has no effect on which the order of the rows will be stored.
* Using a clustered index is an advantage when groups of data that can be clustered are frequently accessed by some queries. This speeds up retrieval because the data lives close to each other on disk. Also, if data is accessed in the same order as the clustered index, the retrieval will be much faster because the physical data stored on disk is sorted in the same order as the index.
* A clustered index can be a disadvantage because any time a change is made to a value of an indexed column, the subsequent possibility of re-sorting rows to maintain order is a definite performance hit.
* A table can have multiple non-clustered indexes. But, a table can have only one clustered index.

# Unique and Non-Unique Indexes:

**Unique index** is used to enforce uniqueness of key values in the index. Let's understand this with an example.  
**Create the Employee table using the script below**

CREATE TABLE [tblEmployee]  
(  
 [Id] int Primary Key,  
 [FirstName] nvarchar(50),  
 [LastName] nvarchar(50),  
 [Salary] int,  
 [Gender] nvarchar(10),  
 [City] nvarchar(50)  
)  
  
**Since, we have marked Id column**, as the Primary key for this table, a UNIQUE CLUSTERED INDEX gets created on the Id column, with Id as the index key.   
  
**We can verify** this by executing the sp\_helpindex system stored procedure as shown below.  
Execute sp\_helpindex tblEmployee

the UNIQUE index is used to enforce the uniqueness of values and primary key constraint

**Creating a UNIQUE NON CLUSTERED index** on the FirstName and LastName columns.

Create Unique NonClustered Index UIX\_tblEmployee\_FirstName\_LastName  
On tblEmployee(FirstName, LastName)

There are no major differences between a unique constraint and a unique index.

**By default, a PRIMARY KEY constraint**, creates a unique clustered index

**A UNIQUE constraint or a UNIQUE index** cannot be created on an existing table

**By default, duplicate values are not allowed on key columns**, when you have a unique index or constraint.

Advantages:

1. **Not only, the SELECT statement, even the following DELETE and UPDATE** statements can also benefit from the index. To update or delete a row, SQL server needs to first find that row, and the index can help in searching and finding that specific row quickly.
2. **Indexes can also help queries e.g. in Order By command**
3. **GROUP BY queries can also benefit from indexes**

**Disadvantages of Indexes:**

1. Additional Disk Space (Non-Clustered).
2. Insert Update and Delete statements can become slow (Indexes also needs to be modified).
3. certain extent, a composite index, can cover a query

**What is a covering query?**  
**If all the columns** that you have requested in the SELECT clause of query are present in the index, then there is no need to look up in the table again. The requested columns data can simply be returned from the index.

# DML Triggers:

In SQL server there are 3 types of triggers  
1. DML triggers (Insert, update, delete)  
2. DDL triggers (Create, Alter, DROP)  
3. Logon trigger

* **In general, a trigger is a special kind of stored procedure** that automatically executes when an event occurs in the database server.
* **DML stands for Data Manipulation Language.** INSERT, UPDATE, and DELETE statements are DML statements. DML triggers are fired, whenever data is modified using INSERT, UPDATE, and DELETE events.
* **DML triggers can be again classified into 2 types.**  
  1. After triggers (Sometimes called as FOR triggers)  
  2. Instead of triggers
* **After triggers, as the name says, fires after the triggering action**. The INSERT, UPDATE, and DELETE statements, causes an after trigger to fire after the respective statements complete execution.
* **On the hand, as the name says, INSTEAD of triggers, fires instead of the triggering action**. The INSERT, UPDATE, and DELETE statements, can cause an INSTEAD OF trigger to fire INSTEAD OF the respective statement execution.

**Example for AFTER TRIGGER for INSERT event on tblEmployee table:**  
CREATE TRIGGER tr\_tblEMployee\_ForInsert  
ON tblEmployee  
FOR INSERT  
AS  
BEGIN  
 Declare @Id int  
 Select @Id = Id from inserted

insert into tblEmployeeAudit   
 values('New employee with Id  = ' + Cast(@Id as nvarchar(5)) + ' is added at ' +cast(Getdate() as nvarchar(20)))  
END  
  
**In the trigger, we are getting the id from inserted table.** So, what is this inserted table? INSERTED table, is a special table used by DML triggers. When you add a new row into tblEmployee table, a copy of the row will also be made into inserted table, which only a trigger can access. You cannot access this table outside the context of the trigger. The structure of the inserted table will be identical to the structure of tblEmployee table.  
  
**So, now if we execute the following INSERT statement on tblEmployee.**Immediately, after inserting the row into tblEmployee table, the trigger gets fired (executed automatically), and a row into tblEmployeeAudit, is also inserted.

**Insert into tblEmployee values (7,'Tan', 2300, 'Female', 3)**

**Example for AFTER TRIGGER for DELETE event on tblEmployee table:**  
CREATE TRIGGER tr\_tblEMployee\_ForDelete  
ON tblEmployee  
FOR DELETE  
AS  
BEGIN  
 Declare @Id int  
 Select @Id = Id from deleted  
   
 insert into tblEmployeeAudit   
 values('An existing employee with Id  = ' + Cast(@Id as nvarchar(5)) + ' is deleted at ' +Cast(Getdate() as nvarchar(20)))  
END

**After Update Trigger:**

Create trigger tr\_tblEmployee\_ForUpdate  
on tblEmployee  
for Update  
as  
Begin

# Instead of insert trigger:

INSTEAD OF triggers are fired instead of the triggering event (INSERT, UPDATE or DELETE events). In general, INSTEAD OF triggers are usually used to correctly update views that are based on multiple tables.

# Transactions in SQL Server

**What is a Transaction?**  
**A transaction is a group of commands** that change the data stored in a database. A transaction is treated as a single unit. A transaction ensures that, either all of the commands succeed, or none of them. If one of the commands in the transaction fails, all of the commands fail, and any data that was modified in the database is rolled back. In this way, transactions maintain the integrity of data in a database.

Create Procedure spUpdateAddress  
as  
Begin  
 Begin Try  
 Begin Transaction  
 Update tblMailingAddress set City = 'LONDON'   
 where AddressId = 1 and EmployeeNumber = 101  
   
 Update tblPhysicalAddress set City = 'LONDON'   
 where AddressId = 1 and EmployeeNumber = 101  
 Commit Transaction  
 End Try  
 Begin Catch  
 Rollback Transaction  
 End Catch  
End

**Transaction Acid Tests:**

A - Atomic  All statements in the transaction either completed successfully or they were all rolled back

C - Consistent

All data touched by the transaction is left in a **logically consistent state**. For example, if stock available numbers are decremented from **tblProductTable**, then, there has to be a related entry in **tblProductSales** table. The inventory can't just disappear.

I – Isolated: The transaction must affect data without interfering with other concurrent transactions, or being interfered with by them. This prevents transactions from making changes to data based on uncommitted information, for example changes to a record that are subsequently rolled back. **Most databases use locking to maintain transaction isolation**.

D - Durable : Once a change is made, it is permanent. If a system error or power failure occurs before a set of commands is complete, those commands are undone and the data is restored to its original state once the system begins running again.

# Different Types of SQL Server Functions

Function is a database object in Sql Server. Basically it is a set of sql statements that accepts only input parameters, perform actions and return the result. Function can return only single value or a table. We can’t use function to Insert, Update, Delete records in the database table(s). For more about stored procedure and function refer the article [Difference between Stored Procedure and Function](http://www.dotnet-tricks.com/Tutorial/sqlserver/7EDL150912-Difference-between-Stored-Procedure-and-Function-in-SQL-Server.html)

## Types of Function

1. System Defined Function

These functions are defined by Sql Server for different purpose. We have two types of system defined function in Sql Server

1. Scalar Function

Scalar functions operates on a single value and returns a single value. Below is the list of some useful Sql Server Scalar functions.

System Scalar Function

Scalar Function

Description

abs(-10.67)

This returns absolute number of the given number means 10.67.

rand(10)

This will generate random number of 10 characters.

round(17.56719,3)

This will round off the given number to 3 places of decimal means 17.567

upper('dotnet')

This will returns upper case of given string means 'DOTNET'

lower('DOTNET')

This will returns lower case of given string means 'dotnet'

ltrim(' dotnet')

This will remove the spaces from left hand side of 'dotnet' string.

convert(int, 15.56)

This will convert the given float value to integer means 15.

1. Aggregate Function

Aggregate functions operates on a collection of values and returns a single value. Below is the list of some useful Sql Server Aggregate functions.

System Aggregate Function

Aggregate Function

Description

max()

This returns maximum value from a collection of values.

min()

This returns minimum value from a collection of values.

avg()

This returns average of all values in a collection.

count()

This returns no of counts from a collection of values.

# User Defined Function

These functions are created by user in system database or in user defined database. We three types of user defined functions.

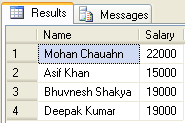
## Scalar Function

User defined scalar function also returns single value as a result of actions perform by function. We return any datatype value from function.

* + 1. **--Create a table**
    2. **CREATE TABLE Employee**
    3. **(**
    4. **EmpID int PRIMARY KEY,**
    5. **FirstName varchar(50) NULL,**
    6. **LastName varchar(50) NULL,**
    7. **Salary int NULL,**
    8. **Address varchar(100) NULL,**
    9. **)**
    10. **--Insert Data**
    11. **Insert into Employee(EmpID,FirstName,LastName,Salary,Address) Values(1,'Mohan','Chauahn',22000,'Delhi');**
    12. **Insert into Employee(EmpID,FirstName,LastName,Salary,Address) Values(2,'Asif','Khan',15000,'Delhi');**
    13. **Insert into Employee(EmpID,FirstName,LastName,Salary,Address) Values(3,'Bhuvnesh','Shakya',19000,'Noida');**
    14. **Insert into Employee(EmpID,FirstName,LastName,Salary,Address) Values(4,'Deepak','Kumar',19000,'Noida');**
    15. **--See created table**
    16. **Select \* from Employee**



* + 1. **--Create function to get emp full name**
    2. **Create function fnGetEmpFullName**
    3. **(**
    4. **@FirstName varchar(50),**
    5. **@LastName varchar(50)**
    6. **)**
    7. **returns varchar(101)**
    8. **As**
    9. **Begin**
    10. **return (Select @FirstName + ' '+ @LastName);**
    11. **end**
    12. **--Calling the above created function**
    13. **Select dbo.fnGetEmpFullName(FirstName,LastName) as Name, Salary from Employee**



## Inline Table-Valued Function

User defined inline table-valued function returns a table variable as a result of actions perform by function. The value of table variable should be derived from a single SELECT statement.

* + 1. **--Create function to get employees**
    2. **Create function fnGetEmployee()**
    3. **returns Table**
    4. **As**
    5. **return (Select \* from Employee)**
    6. **--Now call the above created function**
    7. **Select \* from fnGetEmployee()**



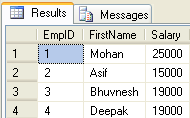
## Multi-Statement Table-Valued Function

User defined multi-statement table-valued function returns a table variable as a result of actions perform by function. In this a table variable must be explicitly declared and defined whose value can be derived from a multiple sql statements.

* + 1. **--Create function for EmpID,FirstName and Salary of Employee**
    2. **Create function fnGetMulEmployee()**
    3. **returns @Emp Table**
    4. **(**
    5. **EmpID int,**
    6. **FirstName varchar(50),**
    7. **Salary int**
    8. **)**
    9. **As**
    10. **begin**
    11. **Insert into @Emp Select e.EmpID,e.FirstName,e.Salary from Employee e;**
    12. **--Now update salary of first employee**
    13. **update @Emp set Salary=25000 where EmpID=1;**
    14. **--It will update only in @Emp table not in Original Employee table**
    15. **return**
    16. **end**

http://www.dotnet-tricks.com/../Content/images/sqlserver/success.png

* + 1. **--Now call the above created function**
    2. **Select \* from fnGetMulEmployee()**



* + 1. **--Now see the original table. This is not affected by above function update command**
    2. **Select \* from Employee**



#### Note

Unlike Stored Procedure, Function returns only single value.

Unlike Stored Procedure, Function accepts only input parameters.

Unlike Stored Procedure, Function is not used to Insert, Update, Delete data in database table(s).

Like Stored Procedure, Function can be nested up to 32 level.

User Defined Function can have upto 1023 input parameters while a Stored Procedure can have upto 2100 input parameters.

User Defined Function can't returns XML Data Type.

User Defined Function doesn't support Exception handling.

User Defined Function can call only Extended Stored Procedure.

User Defined Function doesn't support set options like set ROWCOUNT etc.

# 33. Difference between Stored Procedure and Function in SQL Server

Stored Procedures are pre-compile objects which are compiled for first time and its compiled format is saved which executes (compiled code) whenever it is called. But Function is compiled and executed every time when it is called. For more about stored procedure and function refer the articles [Different types of Stored Procedure](http://www.dotnet-tricks.com/Tutorial/sqlserver/IbUO310312-Different-Types-of-SQL-Server-Stored-Procedures.html) and [Different types of Function](http://www.dotnet-tricks.com/Tutorial/sqlserver/KY3T010412-Different-Types-of-SQL-Server-Functions.html).

## Basic Difference

1. Function must return a value but in Stored Procedure it is optional( Procedure can return zero or n values).
2. Functions can have only input parameters for it whereas Procedures can have input/output parameters .
3. Function takes one input parameter it is mandatory but Stored Procedure may take 0 to n input parameters..
4. Functions can be called from Procedure whereas Procedures cannot be called from Function.

## Advance Difference

1. **Procedure allows SELECT as well as DML(INSERT/UPDATE/DELETE) statement in it whereas Function allows only SELECT statement in it.**
2. Procedures cannot be utilized in a SELECT statement whereas Function can be embedded in a SELECT statement.
3. Stored Procedures cannot be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section whereas Function can be.
4. Functions that return tables can be treated as another row set. This can be used in JOINs with other tables.
5. Inline Function can be thought of as views that take parameters and can be used in JOINs and other Row set operations.
6. Exception can be handled by try-catch block in a Procedure whereas try-catch block cannot be used in a Function.
7. We can go for Transaction Management in Procedure whereas we can't go in Function.

# 34. Select nth max in column of SQL table:

**Get Second maximum value:**

select max(user\_id) as 'max' from user\_details where user\_id in(select distinct top 2 user\_id from user\_details order by user\_id asc)

**Get third maximum value:**

select min(user\_id) from user\_details where user\_id in (select top 3 user\_id from user\_details order by user\_id desc)

# 35. Comma Separated Column:

select login\_id+', '+first\_name as 'loginid and firstname' from user\_details

# 36. Cascading referential integrity constraint:

alter table tbltest2

add constraint fk\_tbltest2\_p\_id1 foreign key(p\_id) references tbltest1(p\_id) on **delete cascade**

alter table tbltest2

add constraint fk\_tbltest2\_p\_id1 foreign key(p\_id) references tbltest1(p\_id) on **update cascade**

Alter table tbltest2

drop constraint fk\_tbltest2\_p\_id1

**However, you have the following options when setting up Cascading referential integrity constraint**

**1. No Action:** This is the default behaviour. No Action specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE or UPDATE is rolled back.  
  
 **2. Cascade:** Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted or updated.  
  
  
**3. Set NULL:** Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to NULL.   
  
  
**4. Set Default:** Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to default values.