1. **Differences between Stored Procedure and User Defined Function in SQL Server**

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **User Defined Function** | **Stored Procedure** |
| 1 | Function must return a value. | Stored Procedure may or not return values. |
| 2 | Will allow only Select statements, it will not allow us to use DML statements. | Can have select statements as well as DML statements such as insert, update, delete and so on |
| 3 | It will allow only input parameters, doesn't support output parameters. | It can have both input and output parameters. |
| 4 | It will not allow us to use try-catch blocks. | For exception handling we can use try catch blocks. |
| 5 | Transactions are not allowed within functions. | Can use transactions within Stored Procedures. |
| 6 | We can use only table variables, it will not allow using temporary tables. | Can use both table variables as well as temporary table in it. |
| 7 | Stored Procedures can't be called from a function. | Stored Procedures can call functions. |
| 8 | Functions can be called from a select statement. | Procedures can't be called from Select/Where/Having and so on statements. Execute/Exec statement can be used to call/execute Stored Procedure. |
| 9 | A UDF can be used in join clause as a result set. | Procedures can't be used in Join clause |

# 2. [**What are the differences between a clustered and a non-clustered index?**](http://stackoverflow.com/questions/91688/what-are-the-differences-between-a-clustered-and-a-non-clustered-index)

Clustered Index

* Only one clustered index can be there in a table
* Sort the records and store them physically according to the order
* Data retrieval is faster than non-clustered indexes
* Do not need extra space to store logical structure

Non Clustered Index

* There can be any number of non-clustered indexes in a table
* Do not affect the physical order. Create a logical order for data rows and use pointers to physical data files
* Data insertion/update is faster than clustered index
* Use extra space to store logical structure

# [**Difference between Delete and Truncate in sql server**](http://stackoverflow.com/questions/3576229/difference-between-delete-and-truncate-in-sql-server-was-i-wrong)

**DELETE**  
1. DELETE is a DML Command.  
2. DELETE statement is executed using a row lock, each row in the table is locked for deletion.  
3. We can specify filters in where clause  
4. It deletes specified data if where condition exists.  
5. Delete activates a trigger because the operation are logged individually.  
6. Slower than truncate because, it keeps logs.  
7. Rollback is possible.  
  
**TRUNCATE**  
1. TRUNCATE is a DDL command.  
2. TRUNCATE TABLE always locks the table and page but not each row.  
3. Cannot use Where Condition.  
4. It removes all the data.  
5. TRUNCATE TABLE cannot activate a trigger because the operation does not log individual row deletions.  
6. Faster in performance wise, because it doesn't keep any logs.  
7. Rollback is not possible.  
  
DELETE and TRUNCATE both can be rolled back when used with TRANSACTION.  
  
If Transaction is done, means COMMITED, then we cannot rollback TRUNCATE command, but we can still rollback DELETE command from LOG files, as DELETE write records them in Log file in case it is needed to rollback in future from LOG files.

# [**Why do we need boxing and unboxing in C#?**](http://stackoverflow.com/questions/2111857/why-do-we-need-boxing-and-unboxing-in-c)

Boxing is the process of converting a [value type](https://msdn.microsoft.com/en-us/library/s1ax56ch.aspx) to the type **object** or to any interface type implemented by this value type. When the CLR boxes a value type, it wraps the value inside a System. Object and stores it on the managed heap. Unboxing extracts the value type from the object. Boxing is implicit; unboxing is explicit. The concept of boxing and unboxing underlies the C# unified view of the type system in which a value of any type can be treated as an object.

# **What is the difference between Finalize() and Dispose()?**

Finalizers are run by the Garbage Collection before an object that is eligible for collection is reclaimed. Dispose() is meant for cleaning up unmanaged resources, like network connections, files, handles to OS stuff, &c. It works best in conjunction with the using block where the compiler makes sure that Dispose() will be called immediately once you are done with an object – and also ensures that you cannot work with the object anymore once it's disposed.

Finalize: nondeterministic destructor/finalizer called automatically by the Garbage Collector when there are no more references to this instance.

Dispose: deterministically called by the developer on an object implementing IDisposable to free resources.

# **9. How is the using() pattern useful? What is IDisposable? How does it support deterministic finalization?**

As a rule, when you use an IDisposable object, you should declare and instantiate it in a using statement. The using statement calls the Dispose method on the object in the correct way, and (when you use it as shown earlier) it also causes the object itself to go out of scope as soon as Dispose is called. Within the using block, the object is read-only and cannot be modified or reassigned.

The using statement ensures that Dispose is called even if an exception occurs while you are calling methods on the object. You can achieve the same result by putting the object inside a try block and then calling Dispose in a finally block; in fact, this is how the using statement is translated by the compiler. The code example earlier expands to the following code at compile time (note the extra curly braces to create the limited scope for the object):

# **What is Static Constructor in C# ? What is the use ?**

A static constructor is used to initialize any static data, or to perform a particular action that needs performed once only. It is called automatically before the first instance is created or any static members are referenced.

* A static constructor does not take access modifiers or have parameters.
* A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.
* A static constructor cannot be called directly.
* The user has no control on when the static constructor is executed in the program.
* A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.
* Static constructors are also useful when creating wrapper classes for unmanaged code, when the constructor can call the LoadLibrary method.

# **Why do we need a private Constructor?**

**Factory**

Private constructors can be useful when using a factory pattern (in other words, a static function that's used to obtain an instance of the class rather than explicit instantiation).

public class MyClass

{

private static Dictionary<object, MyClass> cache =

new Dictionary<object, MyClass();

private MyClass() { }

public static MyClass GetInstance(object data)

{

MyClass output;

if(!cache.TryGetValue(data, out output))

cache.Add(data, output = new MyClass());

return output;

}

}

# **Difference between protected and protected internal?**

|  |  |
| --- | --- |
| accepted | The "protected internal" access modifier is a *union* of both the "protected" and "internal" modifiers.  From [MSDN, Access Modifiers (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/ms173121.aspx):  [**protected**](http://msdn.microsoft.com/en-us/library/bcd5672a.aspx):  The type or member can be accessed only by code in the same class or struct, or in a class that is derived from that class.  [**internal**](http://msdn.microsoft.com/en-us/library/7c5ka91b.aspx):  The type or member can be accessed by any code in the same assembly, but not from another assembly.  **protected internal**:  The type or member can be accessed by any code in the assembly in which it is declared, or from within a derived class in another assembly. Access from another assembly must take place within a class declaration that derives from the class in which the protected internal element is declared, and it must take place through an instance of the derived class type. |

## Manual Initialization of AngularJs Application

If you need to have more control over the initialization process, you can use a manual bootstrapping method instead. Examples of when you'd need to do this include using script loaders or the need to perform an operation before Angular compiles a page.

Here is an example of manually initializing Angular:

<!doctype html>

<html>

<body>

<div ng-controller="MyController">

Hello {{greetMe}}!

</div>

<script src="http://code.angularjs.org/snapshot/angular.js"></script>

<script>

angular.module('myApp', [])

.controller('MyController', ['$scope', function ($scope) {

$scope.greetMe = 'World';

}]);

angular.element(function() {

angular.bootstrap(document, ['myApp']);

});

</script>

</body>

</html>

Note that we provided the name of our application module to be loaded into the injector as the second parameter of the [angular.bootstrap](https://docs.angularjs.org/api/ng/function/angular.bootstrap) function. Notice that angular.bootstrap will not create modules on the fly. You must create any custom [modules](https://docs.angularjs.org/guide/module)before you pass them as a parameter.

You should call angular.bootstrap() after you've loaded or defined your modules. You cannot add controllers, services, directives, etc after an application bootstraps.

**Note:** You should not use the ng-app directive when manually bootstrapping your app.

This is the sequence that your code should follow:

1. After the page and all of the code is loaded, find the root element of your AngularJS application, which is typically the root of the document.
2. Call [angular.bootstrap](https://docs.angularjs.org/api/ng/function/angular.bootstrap) to [compile](https://docs.angularjs.org/guide/compiler) the element into an executable, bi-directionally bound application.

## Things to keep in mind

There a few things to keep in mind regardless of automatic or manual bootstrapping:

* While it's possible to bootstrap more than one AngularJS application per page, we don't actively test against this scenario. It's possible that you'll run into problems, especially with complex apps, so caution is advised.
* Do not bootstrap your app on an element with a directive that uses [transclusion](https://docs.angularjs.org/api/ng/service/$compile#transclusion), such as [ngIf](https://docs.angularjs.org/api/ng/directive/ngIf), [ngInclude](https://docs.angularjs.org/api/ng/directive/ngInclude) and [ngView](https://docs.angularjs.org/api/ngRoute/directive/ngView). Doing this misplaces the app [$rootElement](https://docs.angularjs.org/api/ng/service/$rootElement) and the app's [injector](https://docs.angularjs.org/api/auto/service/$injector), causing animations to stop working and making the injector inaccessible from outside the app.

|  |  |
| --- | --- |
| **WCF** | **ASP.NET Web API** |
| Enables building services that support multiple transport protocols (HTTP, TCP, UDP, and custom transports) and allows switching between them. | HTTP only. First-class programming model for HTTP. More suitable for access from various browsers, mobile devices etc enabling wide reach. |
| Enables building services that support multiple encodings (Text, MTOM, and Binary) of the same message type and allows switching between them. | Enables building Web APIs that support wide variety of media types including XML, JSON etc. |
| Supports building services with WS-\* standards like Reliable Messaging, Transactions, Message Security. | Uses basic protocol and formats such as HTTP, WebSockets, SSL, JQuery, JSON, and XML. There is no support for higher level protocols such as Reliable Messaging or Transactions. |
| Supports Request-Reply, One Way, and Duplex message exchange patterns. | HTTP is request/response but additional patterns can be supported through [SignalR](https://github.com/SignalR/SignalR)and WebSockets integration. |
| WCF SOAP services can be described in WSDL allowing automated tools to generate client proxies even for services with complex schemas. | There is a variety of ways to describe a Web API ranging from auto-generated HTML help page describing snippets to structured metadata for OData integrated APIs. |
| Ships with the .NET framework. | Ships with .NET framework but is open-source and is also available out-of-band as independent download. |

# **Q. Difference between == and .Equals()?**

“==” compares if the object references are same while “.Equals()” compares if the contents are same.

# **Q. What are generics?**

Generics are the most powerful feature of C# 2.0. Generics allow you to define type-safe data structures, without committing to actual data types. This results in a significant performance boost and higher quality code, because you get to reuse data processing algorithms without duplicating type-specific code.

In concept, generics are similar to C++ templates, but are drastically different in implementation and capabilities.

A great example are the many collection classes in .NET. Each collection class has it's own implementation of how the collection is created and managed. But they use generics to allow their class to work with collections of any type.

# **Q. What are the different types of polymorphism?**

Compile Time Polymorphism

Method overloading is a great example. You can have two methods with the same name but with different signatures. The compiler will choose the correct version to use at compile time. Operator overloading is another good example.

Run-Time Polymorphism

Overriding a virtual method from a parent class in a child class is a good example. Another is a class implementing methods from an Interface.

# **Q. What is the difference between a static and const variable?**

A constant value cannot change. A static variable exists to a function, or class, rather than an instance or object.

These two concepts are not mutually exclusive, and can be used together.

Static variables are common across all instances of a type.

Constant variables are specific to each individual instance of a type but their values are known and fixed at compile time and it cannot be changed at runtime.

Unlike constants, static variable values can be changed at runtime.

# **Q. Union and Union All in SQL Server?**

UNION OPERATOR (Alias: DISTINCT UNION ORDERED LIST): is used to combine multiple result sets into one result set and will remove any duplicates rows that exist. Basically it is performing a DISTINCT operation across all columns in the result set.

UNION ALL OPERATOR: is used to combine multiple result sets into one result set, but it does not remove any duplicate rows. Because this does not remove duplicate rows this process is faster, but if you don’t want duplicate records you will need to use the UNION operator instead.

Performance TIP: Compared UNION ALL operator, UNION operator has the extra overhead of removing duplicate rows and sorting results. So, If we know that all the records returned by our query is unique from union then use UNION ALL operator instead of UNION Operator.

Following are the constraints for using UNION/UNION ALL Operator:

All the query’s which need to combine need to have the same number of columns

Column should be of the same data type/compatible data types

ORDER BY clauses can only be issued for the overall result set and not within each result set

Column names of the final result set will be from the first query.

# **Q. Casting vs using the 'as' keyword in the CLR**

The as operator can only be used on reference types, it cannot be overloaded, and it will return null if the operation fails. It will never throw an exception.

Casting can be used on any compatible types, it can be overloaded, and it will throw an exception if the operation fails.

The choice of which to use depends on the circumstances. Primarily, it's a matter of whether you want to throw an exception on a failed conversion.

The as keyword works the same as an explicit cast between compatible reference types with the major difference that it does not raise an exception if conversion fails. Rather, it yields a null value in the target variable. Since Exceptions are very expensive in terms of performance, it is considered a much better method of casting.

# **Q. What is $emit and $on?**

**$emit** = sending data from child to parent controller

**$obroadcast** = sending data from parent to childcontroller

**$on** = listen for the events

https://toddmotto.com/all-about-angulars-emit-broadcast-on-publish-subscribing/

# **Q. Why can't static classes have non-static methods and variables?**

Static classes can't be instantiated in the first place, so even if you could declare non-static (instance) members, they can never be accessed. Since there really isn't a point allowing it for that reason, the language simply prohibits it.

# **Q. Difference between ActionResult and ViewResult?**

**ViewResult** derives from **ActionResult**. Other derived classes include **JsonResult** and **PartialViewResult**. You declare it this way so you can take advantage of polymorphism and return different types in the same method.

# **Q. How do you generate proxy for WCF services?**

* Adding Service Reference
* Implementing ClientBase
* Using Tool i.e. SvcUtil.exe

# **Q. Keep and peek in TempData?**

TempData is also a dictionary object that stays for the time of an HTTP Request. So, TempData can be used to maintain data between one controller action to the other controller action.

TempData is used to check the null values each time. TempData contain two method keep() and peek() for maintain data state from one controller action to others.

When TempDataDictionary object is read, At the end of request marks as deletion to current read object.

The keep() and peek() method is used to read the data without deletion the current read object.

You can use Peek() when you always want to hold/prevent the value for another request. You can use Keep() when prevent/hold the value depends on additional logic.

Overloading in TempData.Peek() & TempData.Keep() as given below.

TempData.Keep() have 2 overloaded methods.

void keep() : That menace all the data not deleted on current request completion.

void keep(string key) : persist the specific item in TempData with help of name.

TempData.Peek() no overloaded methods.

object peek(string key) : return an object that contain items with specific key without mak.

# **Q. How to Force Garbage collector to run?**

The best practice is to not force a garbage collection in most cases.

GC.Collect();

GC.WaitForPendingFinalizers();

It will call GC explicitly throughout your code, don't forget to call GC.WaitForPendingFinalizers(); after GC.Collect().

# **Q.** A resolve is a property you can attach to a route in both ngRoute and the more robust UI router. A resolve contains one or more promises that must resolve successfully before the route will change. This means you can wait for data to become available before showing a view, and simplify the initialization of the model inside a controller because the initial data is given to the controller instead of the controller needing to go out and fetch the data.

# **Q. Difference between static class and singleton pattern?**

**Static Class:-**

You cannot create the instance of static class.

Loaded automatically by the .NET Framework common language runtime (CLR) when the program or namespace containing the class is loaded.

Static Class cannot have constructor.

We cannot pass the static class to method.

We cannot inherit Static class to another Static class in C#.

A class having all static methods.

Better performance (static methods are bonded on compile time)

**Singleton:-**

You can create one instance of the object and reuse it.

Singleton instance is created for the first time when the user requested.

Singleton class can have constructor.

You can create the object of singleton class and pass it to method.

Singleton class does not say any restriction of Inheritance.

We can dispose the objects of a singleton class but not of static class.

Methods can be overridden.

Can be lazy loaded when need (static classes are always loaded).

We can implement interface (static class cannot implement interface).

# **Q. Difference between Virtual and Abstract Methods**

# **Abstract Method**

Abstract Method resides in abstract class and it has no body.

Abstract Method must be overridden in non-abstract child class.

# **Virtual Method**

Virtual Method can reside in abstract and non-abstract class.

It is not necessary to override virtual method in derived but it can be.

Virtual method must have body ....can be overridden by "override keyword".

# **Q. Difference between DataReader and DataSet?**

DataReader is a forward-only iterator over a set of results. It's usually the most efficient way to deal with records when you don't need random access (in other words you can't go backwards). It is "scalable" to any number of records, at least in terms of memory pressure, since it only loads one record at a time. One typical way to get a DataReader is by using the ExecuteReader method of a DbCommand.

DataSet represents a set of DataTable objects. More often than not, it will just contain one table, but if you do a query with multiple SELECT statements, the DataSet will contain a table for each. Because this is an in-memory representation, you have to be careful about how much data you pull into a DataSet. You can "Fill" a DataSet using the Fill method of a DataAdapter.

DataAdapter is a kind of "pipe" that funnels data from a DB engine into a DataSet. That's why you'll have one DataAdapter implementation for each DB provider type. One DataSet, many providers.

DataView is like a virtual subset of a DataTable.

# **Q. What is the difference between Dictionary and List?**

List<> and Dictionary<,> - pretty different data structures which used for different purposes, List is simply a set of items and Dictionary is a set of key-value pairs.

Dictionary is pretty useful when you have a set of complex objects and want to have fast access by let's say ObjectName/ObjectId, in this case you create IDictionary<string, TObject> where key would be ObjectId and Value would be an object itself.

**Some differences:**

List persist order of the items, Dictionary does not.

List allow fast access by index.

List support built in QuickSort algorithm for fast data sorting.

Dictionary allows ~O(1) time complexity to access an item (value) by a key.

# **Differences between Hashtable and Dictionary?**

**Dictionary**:

Dictionary returns error if we try to find a key which does not exist.

Dictionary faster than a Hashtable because there is no boxing and unboxing.

Dictionary is a generic type which means we can use it with any data type.

**Hashtable**:

Hashtable returns null if we try to find a key which does not exist.

Hashtable slower than dictionary because it requires boxing and unboxing.

Hashtable is not a generic type,

# **Q. Why Dictionary is much more faster than List??**

When using Dictionary you are using a key to retrieve your information, which enables it to find it more efficiently, with List you are using Single Linq expression, which since it is a list, has no other option other than to look in entire list for wanted the item.

# **Q. List vs ArrayList ?**

Yes, pretty much. List<T> is a generic class. It allows storing values of a specific type without casting to or from object (which would have incurred boxing/unboxing overhead when T is a value type in the ArrayList case). ArrayList simply stores object references. As a generic collection, List<T> implements the generic IEnumerable<T> interface and can be used easily in LINQ (without requiring any Cast or OfType call).

ArrayList belongs to the days that C# didn't have generics. It's deprecated in favor of List<T>. You shouldn't use ArrayList in new code that targets .NET >= 2.0 unless you have to interface with an old API that uses it.

ArrayList provides some thread-safety through the Synchronized property, which returns a thread-safe wrapper around the collection. The wrapper works by locking the entire collection on every add or remove operation. Therefore, each thread that is attempting to access the collection must wait for its turn to take the one lock. This is not scalable and can cause significant performance degradation for large collections.

List<T> does not provide any thread synchronization; user code must provide all synchronization when items are added or removed on multiple threads concurrently.

# **Q. Exact difference between overriding and hiding**

1. Overriding

In case of the overriden property, base class' virtual method's slot is replaced by a different implementation. Compiler sees the method as virtual, and must resolve its implementation during run-time using the object's virtual table.

{

Base b = new Base();

Console.WriteLine(b.Name); // prints "Base"

b = new Overriden();

// Base.Name is virtual, so the vtable determines its implementation

Console.WriteLine(b.Name); // prints "Overriden"

Overriden o = new Overriden();

// Overriden.Name is virtual, so the vtable determines its implementation

Console.WriteLine(o.Name); // prints "Overriden"

}

2. Hiding

When a method or a property is hidden using the new keyword, the compiler creates a new non-virtual method for the derived class only; base class' method remains untouched.

If the type of the variable is Base (i.e. only contains the virtual method), its implementation will be resolved through the vtable. If the type of the variable is New, then the non-virtual method or property will be invoked.

{

Base b = new Base();

Console.WriteLine(b.Name); // prints "Base"

b = new New();

// type of `b` variable is `Base`, and `Base.Name` is virtual,

// so compiler resolves its implementation through the virtual table

Console.WriteLine(b.Name); // prints "Base"

New n = new New();

// type of `n` variable is `New`, and `New.Name` is not virtual,

// so compiler sees `n.Name` as a completely different property

Console.WriteLine(n.Name); // prints "New"

}

# **Q. Whats is CTS and CLS?**

Another building block of the .NET platform is the Common Type System, or CTS. The CTS specification fully describes all possible data types and programming constructs supported by the runtime, specifies how these entities can interact with each other, and details how they are represented in the .NET metadata format (more information on metadata later in this chapter; see Chapter 16 for complete details).

Understand that a given .NET-aware language might not support each and every feature defined by the CTS. The Common Language Specification (CLS) is a related specification that defines a subset of common types and programming constructs that all .NET programming languages can agree on. Thus, if you build .NET types that only expose CLS-compliant features, you can rest assured that all .NET-aware languages can consume them. Conversely, if you make use of a data type or programming construct that is outside of the bounds of the CLS, you cannot guarantee that every .NET programming language can interact with your .NET code library.

# **Q. What's the difference between a temp table and table variable in SQL Server?**

Temp table: A Temp table is easy to create and back up data.

Table variable: But the table variable involves the effort when we usually create the normal tables.

Temp table: Temp table result can be used by multiple users.

Table variable: But the table variable can be used by the current user only.

Temp table: Temp table will be stored in the tempdb. It will make network traffic. When we have large data in the temp table then it has to work across the database. A Performance issue will exist.

Table variable: But a table variable will store in the physical memory for some of the data, then later when the size increases it will be moved to the tempdb.

Temp table: Temp table can do all the DDL operations. It allows creating the indexes, dropping, altering, etc..,

Table variable: Whereas table variable won't allow doing the DDL operations. But the table variable allows us to create the clustered index only.

Temp table: Temp table can be used for the current session or global. So that a multiple user session can utilize the results in the table.

Table variable: But the table variable can be used up to that program. (Stored procedure)

Temp table: Temp variable cannot use the transactions. When we do the DML operations with the temp table then it can be rollback or commit the transactions.

Table variable: But we cannot do it for table variable.

Temp table: Functions cannot use the temp variable. Moreover we cannot do the DML operation in the functions.

Table variable: But the function allows us to use the table variable. But using the table variable we can do that.

Temp table: The stored procedure will do the recompilation (can't use same execution plan) when we use the temp variable for every sub sequent calls.

Table variable: Whereas the table variable won't do like that.

# **Q. What is partial class?**

It is possible to split the definition of a class or a structure, an interface or a method over two or more source files. Each source file contains a section of the type or method definition, and all parts are combined when the application is compiled.

**Q. Difference between Abstraction and Encapsulation?**  
  
Abstraction is a process. It is the act of identifying the relevant qualities and behaviors an object should possess. Encapsulation is the mechanism by which the abstraction is implemented.

|  |  |
| --- | --- |
| Abstraction | Encapsulation |
| Abstraction solves the problem in the design level. | Encapsulation solves the problem in the implementation level. |
| Abstraction is used for hiding the unwanted data and giving only relevant data. | Encapsulation is hiding the code and data into a single unit to protect the data from outer world. |
| Abstraction is set focus on the object instead of how it does it. | Encapsulation means hiding the internal details or mechanics of how an object does something. |
| Abstraction is outer layout in terms of design.  For Example: - Outer Look of a iPhone, like it has a display screen. | Encapsulation is inner layout in terms of implementation. For Example: - Inner Implementation detail of a iPhone, how Display Screen are connect with each other using circuits |

# **Q. Response. Redirect vs Server. Transfer?**

Response.Redirect() will send you to a new page, update the address bar and add it to the Browser History. On your browser you can click back.

Server.Transfer() does not change the address bar. You cannot hit back.

I use Server.Transfer() when I don't want the user to see where I am going. Sometimes on a "loading" type page.

Otherwise I'll always use Response.Redirect().

# **Q. Http Handler vs Http Module?**

An ASP.NET HTTP handler is the process (frequently referred to as the "endpoint") that runs in response to a request made to an ASP.NET Web application. The most common handler is an ASP.NET page handler that processes .aspx files. When users request an .aspx file, the request is processed by the page through the page handler. You can create your own HTTP handlers that render custom output to the browser.

An HTTP module is an assembly that is called on every request that is made to your application. HTTP modules are called as part of the ASP.NET request pipeline and have access to life-cycle events throughout the request. HTTP modules let you examine incoming and outgoing requests and take action based on the request.

Statistics and logging Because HTTP modules are called on every request, you can gather request statistics and log information in a centralized module.

# **Q. Difference between delegate and events?**

An Event declaration adds a layer of abstraction and protection on the delegate instance. This protection prevents clients of the delegate from resetting the delegate and its invocation list and only allows adding or removing targets from the invocation list.

# **Q. Application and Session State have a very important difference:**

Application state is a data repository available to all classes in an ASP.NET application. Application state is stored in memory on the server and is faster than storing and retrieving information in a database. Unlike session state, which is specific to a single user session, application state applies to all users and sessions. Therefore, application state is a useful place to store small amounts of often-used data that does not change from one user to another.

Session state variables are available across all pages, but only for a given single session. Session variables are like single-user global data. Only the current session has access to its Session state.

Application State variables are available across all pages and across all sessions. Application State variables are like multi-user global data. All sessions can read and write Application State variables.

# **Q. If we have two versions of the same assembly in GAC how to we make a choice?**

You need to specify “bindingRedirect” in your config file.

# **Q. Solid principle?**

**Open-Closed Principle**, which states that software entities – such as classes, modules, functions and so on – should be open for extension but closed for modification. The idea is that it’s often better to make changes to things like classes by adding to or building on top of them (using mechanisms like sub classing or polymorphism) rather than modifying their code.

**Liskov Substitution Principle**, which states that subclasses should be substitutable for the classes from which they were derived. For example, if MySubclass is a subclass of MyClass, you should be able to replace MyClass with MySubclass without bunging up the program.

# **Q. How performance is affected due to boxing and unboxing?**

When boxing and unboxing happens the data needs to jump from stack memory to heap and vice-versa which is a bit of memory intensive process. As a good practice avoid boxing and unboxing where ever possible.

# **Q. How can we add relation between tables in a Dataset?**

dataSet.Relations.Add(dataSet.Tables["Customers"].Columns["customerId"],

dataSet.Tables["Orders"].Columns["customerId"])

# **Q. Difference between SingleOrDefault and FirstOrDefault()?**

SingleOrDefault() operator would throw an exception if more than one elements are satisfied the condition where as FirstOrDefault() will not through any exception for the same.

# **Q. Difference between ReadOnly and Constant?**

**ReadOnly**: The value will be initialized only once from the constructor of the class. The difference is that the value of a static **readonly** field is set at run time, so it can have a different value for different executions of the program.

However, the value of a **const** field is set to a compile time **constant.**

# **Q.** [**Javascript call() & apply() vs bind()?**](http://stackoverflow.com/questions/15455009/javascript-call-apply-vs-bind)

|  |  |
| --- | --- |
|  | Use .bind() when you want that function to later be called with a certain context, useful in events. Use .call() or .apply() when you want to invoke the function immediately, and modify the context.  Call/apply call the function immediately, whereas bind returns a function that when later executed will have the correct context set for calling the original function. This way you can maintain context in async callbacks, and events. |

# **Q. Base and New keyword in C#.**

The **base** keyword is used to refer to the base class when chaining constructors or when you want to access a member (method, property, anything) in the base class that has been overridden or hidden in the current class. For example.

**Base** is used when you override a method in a derived class but just want to add additional functionality on top of the original functionality

For example:

// Calling the Area base method:

public override void Foo()

{

base.Foo(); //Executes the code in the base class

RunAdditionalProcess(); //Executes additional code

}

# **Q. Difference between Ref and Out parameter in C#.**

The ref modifier means that:

a) The value is already set and the method can read and modify it.

The out modifier means that:

a) The Value isn't set and can't be read by the method until it is set.

b) The method must set it before returning.

Or, ref tells the compiler that the object is initialized before entering the function, while out tells the compiler that the object will be initialized inside the function.

# **Q. What is the difference between “Typeof” and “GetType”?**

GetType is a virtual method on Object - this means given an instance of a class, you can retrieve the corresponding Type object.

typeof is a C# operator - this is used to perform a compile time lookup i.e. Given a Symbol representing a Class name, retrieve the Type object for it.

if (typeof(String) == "test".GetType())

Q. What is Garbage Collection in .Net?

In the common language runtime (CLR), the garbage collector serves as an automatic memory manager. It provides the following benefits:

* Enables you to develop your application without having to free memory.
* Allocates objects on the managed heap efficiently.
* Reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations. Managed objects automatically get clean content to start with, so their constructors do not have to initialize every data field.
* Provides memory safety by making sure that an object cannot use the content of another object.

## Generations

The heap is organized into generations so it can handle long-lived and short-lived objects. Garbage collection primarily occurs with the reclamation of short-lived objects that typically occupy only a small part of the heap. There are three generations of objects on the heap:

* **Generation 0**. This is the youngest generation and contains short-lived objects. An example of a short-lived object is a temporary variable. Garbage collection occurs most frequently in this generation.

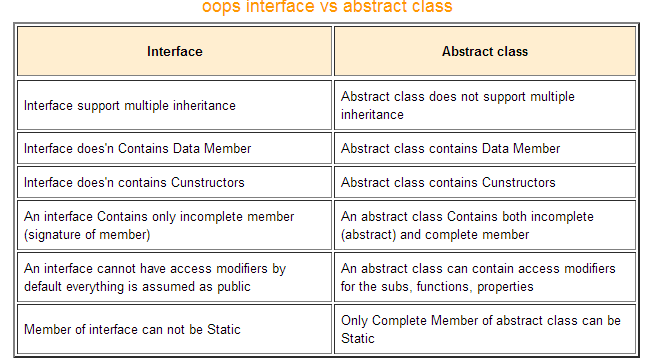
Newly allocated objects form a new generation of objects and are implicitly generation 0 collections, unless they are large objects, in which case they go on the large object heap in a generation 2 collection.

Most objects are reclaimed for garbage collection in generation 0 and do not survive to the next generation.

* **Generation 1**. This generation contains short-lived objects and serves as a buffer between short-lived objects and long-lived objects.
* **Generation 2**. This generation contains long-lived objects. An example of a long-lived object is an object in a server application that contains static data that is live for the duration of the process.

|  |  |  |
| --- | --- | --- |
| **Features** | **Web Service** | **WCF** |
| Hosting | It can be hosted in IIS | It can be hosted in IIS, windows activation service, Self-hosting, Windows service |
| Programming | [WebService] attribute has to be added to the class | [ServiceContract] attribute has to be added to the class |
| Model | [WebMethod] attribute represents the method exposed to client | [OperationContract] attribute represents the method exposed to client |
| Operation | One-way, Request- Response are the different operations supported in web service | One-Way, Request-Response, Duplex are different type of operations supported in WCF |
| XML | System.Xml.serialization name space is used for serialization | System.Runtime.Serialization namespace is used for serialization |
| Encoding | XML 1.0, MTOM(Message Transmission Optimization Mechanism), DIME, Custom | XML 1.0, MTOM, Binary, Custom |
| Transports | Can be accessed through HTTP, TCP, Custom | Can be accessed through HTTP, TCP, Named pipes, MSMQ,P2P, Custom |
| Protocols | Security | Security, Reliable messaging, Transactions |

|  |  |
| --- | --- |
| **Shadowing** | **Overriding** |
| Shadowing is a VB.Net concept. It also known as method hiding in C#. Using this concept we can provide a new implementation for the base class method without overriding it. | Overriding allows us to re-write a base class function with a different definition. |
| Using the “new” keyword we can do the shadowing or method hiding. | C# uses the virtual/abstract and override keyword for method overriding. |
| Shadowing redefines an entire method or function. | Overriding redefines only the implementation of a method or function. |
| Showing is used to protect against subsequent base class modification. | Overriding does polymorphism by defining a different implementation. |
| We can change the access modifier. | We cannot change the access modifier. The access modifier must be the same as in the base class method or function. |
| There is no control of a base class on shadowing. In other words, a base class element cannot enforce or stop shadowing. | The base class has some control over the overriding. Using the keyword abstract, the base class forces the child (derived) class to implement the function or method. |
| Shadowing an element (function method or property) can be inherited further in a child (derived) class. The shadowed element is still hidden. | The same as shadowing, overriding an element is inherited further in a derived class and the overridden element is still overridden. |
| In shadowing, the signature of an element could be different. | In overriding, the signature of the element must be the same. |
| In shadowing, the base class cannot access the newly created child (derived) class method. This is because the base class has the same name of the element. | In concept, the base class can be accessed using the child object's overridden method. |



**Question 5: What is the differences Between DataReader and DataSet?**

|  |  |  |
| --- | --- | --- |
| No | Data Reader | DataSet |
| 1 | Used in a connected architecture | Used in a disconnected architecture. |
| 2 | Provides better performance | Provides lower performance. |
| 3 | DataReader object has read-only access | A DataSet object has read/write access |
| 4 | DataReader object supports a single table based on a single SQL query of one database | A DataSet object supports multiple tables from various databases. |
| 5 | A DataReader object is bound to a single control. | A DataSet object is bound to multiple controls. |
| 6 | A DataReader object has faster access to data. | A DataSet object has slower access to data. |
| 7 | A DataReader object must be manually coded. | A DataSet object is supported by Visual Studio tools. |
| 8 | We can't create a relation in a data reader. | We can create relations in a dataset. |
| 9 | Whereas a DataReader doesn't support data reader communicates with the command object. | A Dataset supports integration with XML Dataset communicates with the Data Adapter only. |
| 10 | DataReader cannot modify data. | A DataSet can modify data. |

**What is SqlCommand Object?**  
The SqlCommand carries the SQL statement that needs to be executed on the database. SqlCommand carries the command in the CommandText property and this property will be used when the SqlCommand calls any of its execute methods.

The Command Object uses the connection object to execute SQL queries.

The queries can be in the form of Inline text, Stored Procedures or direct Table access.

An important feature of Command object is that it can be used to execute queries and Stored Procedures with Parameters.

If a select query is issued, the result set it returns is usually stored in either a DataSet or a DataReader object.

The three important methods exposed by the SqlCommand object is shown below:

ExecuteScalar is useful for returning a single value from the database. For example, using this method we can retrieve a sum of sales made by a specific product, total number of records in the employee table, unique id by supplying filtering conditions and so on. Since this method performs faster we do not need to go for the Reader method just to retrieve a single scalar value.  
  
ExecuteNonQuery is useful for performing data manipulation on the database. Simply, the ExecuteNonQuery is for executing the DML statements. The return value of the ExecuteNonQuery is an integral value that represents the number of rows affected by the Operation.  
  
ExecuteReader is used when we need to retrieve rows and columns of data using the SQL select statements. As the data retrieved is a table of data, ExecuteReader returns SqlDataReader. We should iterate through this object to get the required values.

**What is the DataAdapter Object in ADO.NET?**  
A Data Adapter represents a set of data commands and a database connection to fill the dataset and update a SQL Server database.  
  
A Data Adapter contains a set of data commands and a database connection to fill the dataset and update a SQL Server database. Data Adapters form the bridge between a data source and a dataset.  
  
Data Adapters are designed depending on the specific data source. The following table shows the Data Adapter classes with their data source.

|  |  |
| --- | --- |
| **Provider-Specific Data Adapter classes** | **Data Source** |
| SqlDataAdapter | SQL Server |
| OledbDataAdapter | OLE DB provider |
| OdbcDataAdapter | ODBC driver |
| OracleDataAdapter | Oracle |

A Data Adapter supports mainly the following two methods:

* **Fill ():**The Fill method populates a dataset or a data table object with data from the database. It retrieves rows from the data source using the SELECT statement specified by an associated select command property.  
    
  The Fill method leaves the connection in the same state as it encountered before populating the data.
* **Update ():** The Update method commits the changes back to the database. It also analyzes the RowState of each record in the DataSet and calls the appropriate INSERT, UPDATE, and DELETE statements.

**Example:**

1. SqlDataAdapter da=**new** SqlDataAdapter("Select \* from
2. Employee", con);
3. da.Fill(ds,"Emp");
4. bldr =**new** SqlCommandBuilder(da);
5. dataGridView1.DataSource = ds.Tables["Emp"];

**What is DataTable in ADO.NET?**

* DataTable represents a single table in a database.
* In this show row and column.
* DataSet is a collection of data tables.

In this store data record.

DataTable representation in .aspx.cs code,

1. **protected** **void** BinddataTable()
2. {
3. SqlConnection con = **new** SqlConnection("your database connection string");
4. con.Open();
5. SqlCommand cmd = **new** SqlCommand("Write your query or procedure", con);
6. SqlDataAdapter da = **new** SqlDataAdapter(cmd);
7. DataTable dt = **new** DataTable();
8. da.Fill(dt);
9. grid.DataSource = dt;
10. grid.DataBind();
11. }

**What is the DataReader in ADO.Net?**  
DataReader holds only one table at a time.

It only provides read only access mode and cannot write data.

It is not required local storage to data store.

Holds one row at a time.

Uses less memory.

DataReader do not maintain relation.

DataReader representation in .aspx.cs code,

1. **protected** **void** Bind()
2. {
3. SqlConnection con = **new** SqlConnection("your database connection string ");
4. con.Open();
5. SqlCommand cmd = **new** SqlCommand("Write your query or procedure ", con);
6. SqlDataReader dr = cmd.ExecuteReader();
7. grid.DataSource = dr;
8. grid.DataBind();
9. }

**What is the SqlCommandBuilder?**  
CommandBuilder helps you to generate update, delete, and insert commands on a single database table for a data adapter. Similar to other objects, each data provider has a command builder class. The OleDbCommandBuilder, SqlCommonBuilder, and OdbcCommandBuilder classes represent the CommonBuilder object in the OleDb, Sql, and ODBC data providers.  
  
Creating a Command Builder Object:  
  
Creating a CommonedBuider object is pretty simply. You pass a DataAdapter as an argument of the CommandBuilder constructor. For example,

1. // Create a command builder object
2. SqlCommandBuilder builder = **new** SqlCommandBuilder(adapter);

**What do you understand by SqlTransaction class in ADO.NET?**  
The SqlTransaction class is an important class of .NET Framework. It ensures that a body of code will affect a Database or kept the same as previous (Rollback).  
  
At first we should know about it's two most important method which will be used here. They are given below.

Commit(): It commits the transaction. It save changes made in Database during transaction. In simple term we can also say that it shows the end of transaction at that time.

Rollback(): It is used to rollback the transaction. It set the database in previous stage which was, before the begin of transaction.

**What is Connection Pooling in ADO.NET?**  
  
Connection pooling is the ability of reusing your connection to the database. This means if you enable Connection pooling in the connection object, actually you enable the re-use of the connection to more than one user.  
  
ADO.NET uses a technique called connection pooling, which minimizes the cost of repeatedly opening and closing connections. Connection pooling reuses existing active connections with the same connection string instead of creating new connections when a request is made to the database. It involves the use of a connection manager that is responsible for maintaining a list, or pool, of available connections for a given connection string. Several pools exist if different connection strings ask for connection pooling.

**Question 1: What is ADO.NET?**  
ADO stands for Active Data Object and ADO.NET is a set of .NET libraries for ADO.  
  
ADO.NET is a collection of managed libraries used by .NET applications for data source communication using a driver or provider:

Enterprise applications handle a large amount of data. This data is primarily stored in relational databases, such as Oracle, SQL Server, and Access and so on. These databases use Structured Query Language (SQL) for retrieval of data.

To access enterprise data from a .NET application, an interface was needed. This interface acts as a bridge between an RDBMS system and a .NET application. ADO.NET is such an interface that is created to connect .NET applications to RDBMS systems.

In the .NET framework, Microsoft introduced a new version of Active X Data Objects (ADO) called ADO.NET. Any .NET application, either Windows based or web based, can interact with the database using a rich set of classes of the ADO.NET library. Data can be accessed from any database using connected or disconnected architecture.

ADO.NET provides mainly the following two types of architectures:

* Connected Architecture.
* Disconnected Architecture.

**Explain the ExecuteReader method:**

The DataReader object is a forward-only and read-only cursor.

It requires a live connection to the data source.

The DataReader object cannot be directly instantiated. Instead, we must call the ExecuteReader() method of the command object to obtain a valid DataReader object.

**Example:**

1. SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

**Explain the ExecuteXmlReader?**  
  
The execute reader method is flexible when we need the result set in the form of an XML doucment. The ExecuteXmlReader methods returns an instance of XmlReader class.  
  
**Example:** XmlReader xmlreader = cmd.ExecuteXmlReader();XmlDocument xdoc = new XmlDocument();  
Using the XmlDocument class we load the XmlReader object and save it to the File System using the Save method.