**C# Basics**

* **What is C#? What is the difference between C# and .NET?**

C# (pronounced as "C sharp") is a programming language developed by Microsoft within its .NET framework. It is designed for building a variety of applications that run on the .NET platform. C# is widely used for developing desktop applications, web applications, mobile apps, games, and more.

.NET is a **framework** for building and running software's and applications.

* **What is OOPS? What are the main concepts of OOPS?**

OOPS stands for Object-Oriented Programming (OOP), which is a programming paradigm based on the concept of "objects", which can contain data in the form of fields (attributes or properties) and code in the form of procedures (methods or functions). OOP aims to organize software design around real-world entities and their interactions.

**Class**: A class is a blueprint for creating objects. It defines the properties and behaviors (methods) that objects of the class will have.

**Object**: An object is an instance of a class. It represents a real-world entity and encapsulates data and behaviors.

**Polymorphism**: It refers to the ability of different classes to be treated as instances of the same class through a common interface. There are two main types of polymorphism in C#: compile-time polymorphism (also known as method overloading) and runtime polymorphism (also known as method overriding). Let's explore each:

**Compile Time is also known as Early Binding or Overloading:**

In Compile time polymorphism or Early Binding, we will use multiple methods with the same name but different types of parameters or maybe the number of parameters. Because of this, we can perform different-different tasks with the same method name in the same class, also known as Method overloading. Return type can be diff.

**Run Time is also known as Late Binding or Overriding:**

Run time polymorphism is also known as late binding, we can use the same method names with the same signatures, which means the same type or number of parameters, but not in the same class because the compiler doesn’t allow that at compile time. Therefore, we can use that bind at run time in the derived class when a child or derived class object is instantiated. That’s why we call it Late Binding. We have to create my parent class functions as partial and in the driver or child class as override functions with the override keyword. Return type must be same.

In this example, the Display method in the DerivedClass overrides the Display method in the BaseClass. When you call Display on an instance of BaseClass that references a DerivedClass object, the DerivedClass method is called.

e.g.

BaseClass baseDerivedObj = new DerivedClass();

baseDerivedObj.Display(); // Output: Display method in DerivedClass

**Method Hiding :**

Method hiding occurs when a derived class defines a method with the same name and signature as a method in its base class. The derived class method hides the base class method. To explicitly indicate that a method is hiding a base class method, the new keyword is used.

In this example, the Display method in the DerivedClass hides the Display method in the BaseClass. When you call Display on an instance of BaseClass that references a DerivedClass object, the BaseClass method is called.

e.g.

BaseClass baseDerivedObj = new DerivedClass();

baseDerivedObj.Display(); // Output: Display method in BaseClass

**Encapsulation**: Encapsulation is the bundling of data (attributes) and methods (functions) that operate on the data into a single unit or class. OR

Encapsulation is the mechanism of hiding the internal details of an object and only exposing what is necessary. This is typically achieved using access modifiers (private, protected, public, etc.).

**Inheritance**: Inheritance is the mechanism by which one class can acquire the properties and behaviors of another class. It promotes code reusability and establishes a hierarchical relationship between classes.

Inheritance is creating a **PARENT-CHILD** relationship between two classes, where child class will **automatically** get the properties and methods of the parent.

Inheritance is good for: **REUSABILITY** and **ABSTRACTION** of code

Types of Inheritance:

**Single Inheritance**: In single inheritance, a class can inherit properties and behavior from only one superclass. C# supports single inheritance, meaning a class can inherit from only one base class.

**Multilevel Inheritance**: In multilevel inheritance, a derived class inherits properties and behavior from another derived class, creating a chain of inheritance.

**Hierarchical Inheritance**: In hierarchical inheritance, multiple derived classes inherit from the same base class. Each derived class forms its own branch of the inheritance hierarchy.

**Multiple Inheritance** (Not Supported in C#): Multiple inheritance refers to a class inheriting properties and behavior from more than one base class. While some programming languages support multiple inheritance, such as C++, C# does not directly support it. However, C# provides a similar feature through interfaces.

**Interface Inheritance**: While not strictly inheritance in the traditional sense, interfaces in C# allow a class to inherit method signatures (without implementation) from one or more interfaces. A class can implement multiple interfaces, effectively achieving a form of multiple inheritance.

**Abstraction**: It involves hiding the implementation details of a system or component and only showing the essential features or functionalities to the outside world. In other words, abstraction allows you to focus on what an object does rather than how it does it. OR

Abstraction is the concept of hiding complex implementation details and showing only the necessary features of an object. This can be achieved using abstract classes and interfaces.

* **What is Virtual Keyword**

The virtual keyword in C# is used to modify a method, property, indexer, or event declaration and allow it to be overridden in a derived class. When a method or property is marked as virtual, it provides a default implementation that can be overridden by any class that inherits from the base class.

* **What are the types of classes in C#?**

**Regular Classes**: These are the most common type of classes used in C#. They can contain fields, properties, methods, events, indexers, constructors, and other members.

**Abstract Classes**: Abstract classes cannot be instantiated directly. They are used as base classes from which other classes can be derived. Abstract classes may contain abstract members that must be implemented by derived classes.

**Sealed Classes**: Sealed classes cannot be inherited. They are typically used to prevent further derivation or extension of the class. Sealed classes are often used to create immutable types.

**Static Classes**: Static classes cannot be instantiated and can only contain static members such as methods, properties, and fields. They are used to create utility classes that provide a set of related functionality without needing to create an instance of the class.

**Partial Classes**: Partial classes allow a class's definition to be split into multiple files. This can be useful for organizing large classes or when different developers are working on different parts of the same class.

**Nested Classes**: Nested classes are classes that are defined within another class. They can be static or non-static. Nested classes are often used to encapsulate helper classes or to logically group related classes together.

* **What are Namespaces?**

In C#, a namespace is a logical grouping of related classes, interfaces, structs, enums, and delegates. Namespaces are used to organize code and to prevent name conflicts by grouping related elements under a common name. They provide a way to logically separate different parts of a program, making it easier to manage and maintain.

* **How to prevent a class from being Inherited?**

By using **SEALED** keyword in class && By using **STATIC** keyword in base class and also you can use private constructor.

Difference between sealed & static is, you can create the object of sealed class, but you cannot create the object of static class.

* **Are private class members inherited to the derived class?**

No, the private members cannot be inherited to derived class.

Only public and protected class members can be inherited.

* **What is the difference between Method Overriding and Method Hiding?**

In Method Hiding, you can completely hide the implementation of the methods of a base class from the derived class using the new keyword.

* **What is the difference between an Abstract class and an Interface**

|  |  |  |
| --- | --- | --- |
|  | **Abstract class** | **Interface** |
| Purpose: | An abstract class is a class that cannot be instantiated on its own; it serves as a base class for other classes.  It can contain both abstract (with no implementation) and non-abstract (with implementation) methods.  It allows the definition of common functionality while providing a partial implementation. | An interface is a contract that defines a set of methods and properties without providing any implementation.  It allows a class to inherit from multiple interfaces, enabling a form of multiple inheritance. |
| Inheritance: | A class can inherit from only one abstract class at a time.  It supports both complete and partial code reuse. | A class can implement multiple interfaces.  It supports a more flexible way of achieving code reuse through multiple inheritance of contracts. |
| Constructors: | Abstract classes can have constructors.  Constructors are used to initialize the state of the class and its derived classes. | Interfaces cannot have constructors.  They don't define the state of an object. |
| Access Modifiers: | Abstract classes can have access modifiers for their members.  Abstract classes can have fields (variables) with access modifiers. | All members of an interface are implicitly public, and it cannot have access modifiers (public, private, etc.) for its members.  Fields are not allowed in interfaces. |

* **What are Access Specifiers? – Default modifier for class is internal**

Access specifiers are keywords to specify the accessibility of a class, method, property, field.

The keywords are – Public, Private, Protected, Internal, Protected Internal.

**public:** The public access modifier makes a type or member accessible from any other code in the same assembly or another assembly.

**private:** The private access modifier restricts the accessibility to the containing type. Members marked as private are not accessible from outside the class.

**protected:** The protected access modifier allows access to the member from within its class or from derived classes.

**Internal:** The internal access modifier limits access to the current assembly. Members marked as internal are accessible only within the same assembly.

**protected internal:** The protected internal access modifier combines the behavior of both protected and internal. It allows access from within the same assembly or from derived classes in any assembly.

**private protected:** Introduced in C# 7.2, the private protected access modifier allows access to members within the same assembly and from derived classes in the same assembly.

* **What is Boxing and Unboxing? Where to use Boxing and Unboxing in real applications?**

**Boxing**: Boxing is the process of converting a value type to a reference type by encapsulating the value type in an object.

**Unboxing**: Unboxing is the process of converting a reference type to a value type by extracting the value from the boxed object.

We internally use boxing when item is added to Arraylist. And we use unboxing when item is extracted from Arraylist.

* **What is the difference between “String” and “StringBuilder”?**

**Mutability**:

String: Strings in C# are immutable, meaning their values cannot be changed after they are created. Any operation that appears to modify a string actually creates a new string object.

StringBuilder: StringBuilder, on the other hand, is mutable. It provides methods to modify its content without creating new string objects. This makes it more efficient when dealing with large amounts of string manipulation operations.

**Usage Scenarios**:

String: Strings are suitable for situations where the content is fixed or will not change frequently. They are often used to represent text literals, constants, or data that does not require frequent modification.

StringBuilder: StringBuilder is more suitable for scenarios where the content of the string needs to be frequently modified, such as building strings dynamically, concatenating multiple strings in a loop, or performing complex string manipulations.

In summary, String is immutable and efficient for representing fixed or unchanging text, while StringBuilder is mutable and more efficient for dynamic string manipulation operations.

* **What is the difference between “continue” and “break” statement?**

Using a break statement, you can 'jump out of a loop,' whereas using a continue statement, you can 'jump over one iteration' and resume your loop execution.

The break statement is used to exit a loop prematurely, regardless of the loop condition.

When encountered, break immediately terminates the loop, and the program continues with the next statement after the loop.

The continue statement is used to skip the rest of the loop's body for the current iteration and move on to the next iteration.

* **Exception Handling in C#?**

TRY − A try block is a block of code inside which any error can occur.

CATCH − When any error occur in TRY block then it is passed to catch block to handle it.

FINALLY − The finally block is used to execute a given set of statements, whether an exception occur or not.

The base class for all exceptions is the **System.Exception** class.

**Throwing Exceptions (throw):**

You can throw an exception explicitly using the throw keyword. This typically occurs when an error condition is detected within your code.

Throw ex will change the stack trace, where as throw will preserve the whole **stack trace**.

**Catching Exceptions (try-catch):**

To handle exceptions, you use a try block to enclose the code that might throw an exception.

You then use one or more catch blocks to specify how to handle specific types of exceptions.

If an exception occurs within the try block, the runtime searches for a matching catch block, and the corresponding block is executed.

**Finally Block (finally):**

You can optionally include a finally block after the catch blocks. The code in the finally block is always executed, regardless of whether an exception occurs or not.

Use the finally block to perform cleanup operations, such as closing files or releasing resources, that should occur regardless of exceptions.

**Multiple Catch Blocks:**

You can have multiple catch blocks to handle different types of exceptions. The runtime searches for the first matching catch block in the order they appear.

* **Some of built in exceptions**

System.ArgumentException - Thrown when one of the arguments provided to a method is not valid.

System.ArgumentNullException - Thrown when a null reference is passed to a method that does not accept it as a valid argument.

System.ArgumentOutOfRangeException - Thrown when the value of an argument is outside the allowable range of values as defined by the invoked method.

System.ArithmeticException - Base class for errors in an arithmetic, casting, or conversion operation.

System.DivideByZeroException - Thrown when there is an attempt to divide an integral or decimal value by zero.

System.FormatException - Thrown when the format of an argument is invalid, or a format conversion fails.

System.IndexOutOfRangeException - Thrown when an attempt is made to access an element of an array or collection with an index that is outside its bounds.

System.NullReferenceException - Thrown when there is an attempt to dereference a null object reference.

System.OutOfMemoryException - Thrown when there is not enough memory to continue the execution of a program.

* **What is the difference between Finally and Finalize?**

Finally, is used in exception handling.

Finalize is a method which is automatically called by the garbage collector to dispose the no longer needed objects.

* **Explain Generics in C#? When and why to use them in real applications?**

Generics allows us to make classes and methods - type independent or type safe.

With help of Generics, one needs to write a method/class/interface only once and use it for any type whereas, in non-generics, the code needs to be written again and again whenever needed

* **Difference between IEnumerable and IQueryable:**

|  |  |
| --- | --- |
| **IEnumerable** | **IQueryable** |
| IEnumerable operations are performed in-memory. | IQueryable operations are typically executed on the data source (e.g., a database). |
| IEnumerable supports deferred execution, but the data must be in-memory. | IQueryable supports deferred execution, allowing queries to be translated and executed on the server. |
| Use IEnumerable for in-memory collections and when working with LINQ to Objects. | Use IQueryable when querying data from an external data source like a database. |
| IEnumerable bring all result from database and then filter it at code side, which is a network load and performance issue. | IQueryable filter the result at database only and then get only filtered result, therefore less network load and better performance |

* **What is the difference between IEnumerable and IEnumerator in C#?**

**IEnumerable**:

IEnumerable is an interface that represents a collection or sequence of items that can be enumerated or iterated over.

It contains a single method, GetEnumerator(), which returns an IEnumerator interface.

Implementing IEnumerable allows an object to be used in a foreach loop or other constructs that require enumeration.  
IEnumerable represents a read-only collection, meaning it’s suitable when you just want to allow **iteration without modification**. Since GetPeople() only reads data without changing it, IEnumerable is a good fit. Supports deferred execution support.  
**Deferred execution** in LINQ (Language-Integrated Query) refers to the concept of delaying the execution of a query until its results are actually needed

**IEnumerator**:

IEnumerator is an interface that represents an enumerator, which is responsible for iterating over the elements of a collection.

It defines methods and properties for moving through the collection, accessing the current element, and resetting the enumerator.

* **What is a Constructor? When to use constructor in real applications?**

A constructor is a specialized method in the class which gets executed when a class object is created.

Constructor name will same as of Class name.

A constructor is used to set default values for the class.

Constructors are used in real applications to initialize object state, perform dependency injection, define custom initialization logic, create immutable objects, allocate and cleanup resources, and initialize static members.

Types of Constructors

**Default Constructor:**

A default constructor is a constructor that takes no parameters.

If a class does not define any constructors explicitly, a default constructor is provided automatically by the compiler.

It initializes the object's fields to their default values (e.g., null for reference types, 0 for numeric types).

**Parameterized Constructor**:

A parameterized constructor is a constructor that takes one or more parameters.

It allows you to initialize object fields with specific values provided as arguments when creating an instance of the class.

**Copy Constructor:**

A copy constructor is a constructor that creates a new object by copying the state of an existing object of the same type.

It is used to initialize a new object with the same values as another object.

Copy constructors are not directly supported in C#, but you can achieve similar behavior by defining a constructor that takes an instance of the same type as a parameter.

**Static Constructor:**

A static constructor is a special type of constructor that is called automatically before the first instance of a class is created or any static members are accessed.

It is used to initialize static fields or perform one-time initialization tasks for a class.

No parameters or access modifiers (public, private, etc.) are allowed.

Cannot be called explicitly and is called only once per application domain.

**Private Constructor:**

When a constructor is created with a private specifier, it is not possible for other classes to derive from this class, neither is it possible to create an instance of this class.

* **What is Constructor overloading?**

Constructor Overloading is a technique to define multiple constructors within a class with different sets of parameters.

* **What is Destructor?**

Destructors in C# are methods inside the class used to destroy instances of that class when they are no longer needed.

The Destructor is called implicitly by the .NET Framework’s Garbage collector

* **What is the difference between Pass by Value and Pass by Reference Parameters**

**Pass by Value:**

In pass by value, a copy of the argument's value is passed to the method or function. The method operates on this copy, and any changes made to the parameter inside the method do not affect the original variable outside of the method.

In C#, primitive data types (such as integers, floats, and characters) are passed by value by default.

Pass by value is straightforward and ensures that the original variable's value remains unchanged after the method call.

**Pass by Reference:**

In pass by reference, a reference to the original variable's memory location is passed to the method or function. Any changes made to the parameter inside the method directly affect the original variable outside of the method.

In C#, you can pass parameters by reference explicitly using the ref keyword.

Pass by reference allows methods to modify the original variables directly, providing a way to implement "out" parameters and allowing for more complex behavior.

* **What is the difference between “out” and “ref” parameters**

C#, both out and ref keywords are used to pass arguments by reference to a method, but they have different purposes and usage requirements.

**out Parameter**

Purpose: Used to return multiple values from a method.

Initialization Requirement: The variable does not need to be initialized before being passed to the method. However, the method must assign a value to the out parameter before it returns.

Usage: Use out when you need the method to output a value to the caller.

**ref Parameter**

Purpose: Used to pass a variable to a method by reference so that the method can read and modify its value.

Initialization Requirement: The variable must be initialized before it is passed to the method.

Usage: Use ref when the variable should be modified by the method and the original value should be used inside the method.

* **Diff bet “is” and “as” keyword**

The ‘is’ operator is used to check the type of an object.

The ‘as’ operator is used to perform conversion bet compatible objects.

* **What is “params” keyword? When to use params keyword in real applications**

Params keyword is used as a parameter which can take the **VARIABLE** number of parameters**.**

It is useful when programmer don’t have any prior knowledge about the number of parameters to be used.

* **What are Extension Methods in C#? When to use extension methods in real application**

Extension method allows you to add new methods in the existing class without modifying the source code of the original class.

Extension method must be static because this will be directly called from the class name, not by the object creation.

this keyword is used for binding this method with the main class.

* **What are Delegates in C#? When to use delegates in real applications**

A delegate in C# is like a contract or an agreement that allows one method to call another method. It's a type that represents a reference to a method with a specific signature, allowing you to pass methods around as if they were variables.

A Delegate is a variable that holds the reference to a method or Pointer to a method.

A delegate can refer to more than one methods of same return type and parameters.

When to use delegate?

When we need to pass a method as a parameter.

e.g.:-

public delegate int Operation(int a, int b);

public class Program

{

public static int Add(int a, int b)

{

return a + b;

}

public static int Multiply(int a, int b)

{

return a \* b;

}

public static void Main()

{

// Instantiate the delegate

Operation op = Add;

// Call the delegate

int result = op(3, 4);

Console.WriteLine(result); // Output: 7

// Reassign the delegate to another method

op = Multiply;

result = op(3, 4);

Console.WriteLine(result); // Output: 12

}

}

* **What are the Events C#**

Events in C# are a way for objects to signal or announce that something has happened. They're like notifications or announcements that one object can make to let other objects know that a particular action or change has occurred.

* **Difference between events an delegates**

The event is a notification mechanism that depends on delegates

An event is dependent on a delegate and cannot be created without delegates.

Event is like a wrapper over the delegate to improve its security.

* **What is “this” keyword in C#? When to use it in real applications**

The this keyword refers to the current instance of the class in which it appears. It's a reference to the object on which the method or property is being invoked.

*this* keyword avoids the name confusion between class fields and constructor parameters.

* **What is the difference between “Readonly” and “Constant” variables**

**Value Assignment:**

const: Constants must be assigned a value at the time of declaration, and this value cannot be changed throughout the program's execution. They are implicitly static, meaning they belong to the type itself rather than to individual instances.

readonly: Readonly variables can only be assigned a value once, either at the time of declaration or in the constructor of the class. After initialization, the value of a readonly field cannot be changed.

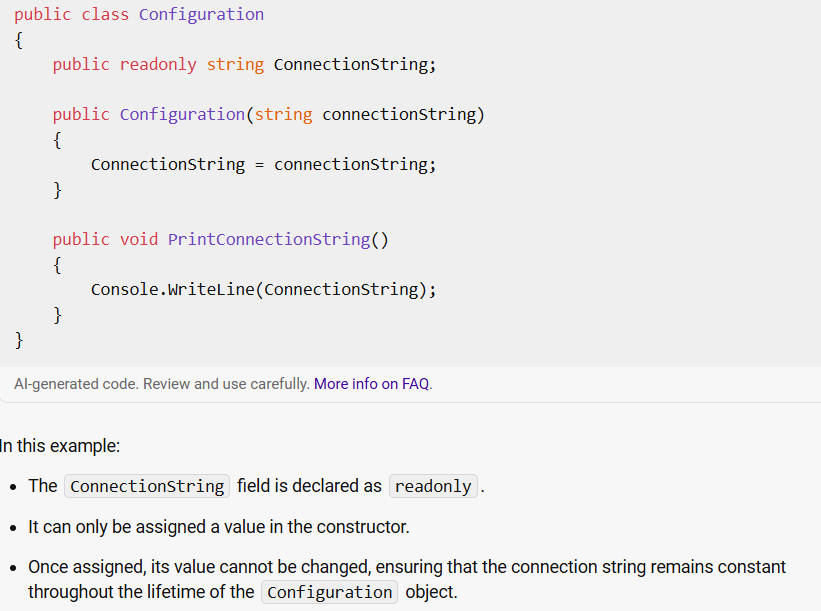
**Usage:**

const: Constants are typically used for values that are known at compile time and do not change during program execution, such as mathematical constants or configuration settings.

readonly: Readonly variables are used when a value needs to be initialized at runtime or when the value is dependent on constructor parameters. They are commonly used for properties or fields whose values can only be determined at runtime.

**Scope**:

const: Constants have a global scope and can be accessed anywhere within the same assembly. They are replaced with their literal values during compilation, so they do not have a memory location at runtime.

readonly: Readonly variables have the same scope as other variables and fields. They have a memory location at runtime and can have different values for each instance of a class.  


**Static vs. Instance**:

const: Constants are implicitly static and belong to the type itself. They can be accessed using the type name without needing an instance of the class.

readonly: Readonly variables can be either static (shared across all instances of a class) or instance-specific (unique to each instance). Instance readonly variables must be initialized in the constructor of the class.

* **What is “Static" class? When to use static class in real application**

In C#, a static class is a class that cannot be instantiated and can only contain static members (fields, methods, properties, and events). The static keyword is used to define such classes.

**Cannot be instantiated:** Static classes cannot be instantiated using the new keyword. They are typically used to provide utility functionality or services that do not require instantiation.

**Contain only static members**: Static classes can only contain static members, such as static methods, properties, fields, and events. They cannot contain instance members.

**Implicitly sealed**: Static classes are implicitly sealed, meaning they cannot be inherited from. You cannot derive a class from a static class, nor can you mark a static class as sealed

* **What is the difference between “var” and “dynamic” in C#**

VAR - The type of the variable is decided by the compiler at **compile time**.

DYNAMIC - The type of the variable is decided at **run time**.

* **What is Enum keyword used for?**

In C#, an enum (enumeration) is a user-defined data type that consists of named constants representing distinct values or named integer constants. Enums make code more readable and self-documenting by providing meaningful names to numeric values, making it easier to understand the purpose of variables and parameters in the code.

Enums are declared using the enum keyword, followed by the enum name and a set of named constants enclosed in curly braces.

* **What is LINQ? When to use LINQ in real applications**

LINQ (Language Integrated Query) is a set of features introduced in C# and .NET Framework that provides a uniform way to query and manipulate data from different data sources using a SQL-like syntax. LINQ allows developers to write queries directly within their C# code, making it easier to work with collections, databases, XML, and other data sources.

In simple terms, LINQ enables you to write queries to filter, sort, group, and transform data using a syntax that is similar to SQL queries.

* **What is Lambda Expressions? What is the use in real applications**

In simple terms, lambda expressions in C# are a way to write short, inline functions without needing to define a separate method. They're like mini-functions that you can create on the fly and use wherever you need them.

A lambda expression is used to simplify the syntax of anonymous methods

e.g. Func<int, int, int> add = (x, y) => x + y;

Console.WriteLine(add(3, 4)); // Output: 7

* **What is the difference between First and FirstOrDefault methods in LINQ**

**First() Method:**

The First() method returns the first element of a sequence.

If the sequence is empty, it throws an InvalidOperationException.

**FirstOrDefault() Method:**

The FirstOrDefault() method returns the first element of a sequence or a default value if the sequence is empty.

It does not throw an exception if the sequence is empty; instead, it returns the default value for the type (e.g., null for reference types, 0 for numeric types).

**Single**

Purpose: Returns the only element of a sequence that satisfies a specified condition and throws an exception if more than one such element exists.

Exceptions: Throws InvalidOperationException if the sequence contains more than one element or if the sequence is empty.

**SingleOrDefault**

Purpose: Returns the only element of a sequence that satisfies a specified condition or a default value (usually null for reference types and default(T) for value types) if no such element exists. Throws an exception if more than one such element exists.

Exceptions: Throws InvalidOperationException if the sequence contains more than one element that matches the condition.

**.NET Framework – Basics**

* **What are the important components of .NET framework**

**CLR (Common Language Runtime)**

CLR manages the execution of programs for example operations like memory management, security checks etc.

CLR provides execution of managed code only (Code written in any .NET-supported language such as C#, Visual Basic .NET, or F#).

**CTS(Common Type System)**

CTS is a set of rules that define how types are declared, used, and managed in the .NET Framework. Types like int, string, double.

**CLS(Common Language Specification)**

CLS is a subset of CTS. It defines a set of rules that every language must follow which runs under .NET framework.

The benefit is, if you create same program logic in different languages of .NET, then the compiler will generate same dll.

**FCL(Framework Class Library)**

FCL is the collection of classes, namespaces, interfaces and value types that are used for .NET applications.

For example, String, Arraylist, List classes are provided by FCL only.

* **What is an Assembly? What are the different types of assembly in .NET**

Assembly is unit of deployment like EXE or a DLL.

When you create a code and build the solution, then the .NET Framework convert your code into Intermediate Language and that is placed inside the assembly(dll), which you can find inside bin folder.

There are 3 types of assemblies:

**Private assembly** - A private assembly can be used by a single application only. It is not accessible outside. So, all the projects you create will by default create private assembly only.

**Public/ shared assembly** - Shared assemblies are usually libraries of code, which multiple applications can use. It is registered in the global assembly cache(GAC).

**Satellite assembly** - A satellite Assembly is defined as an assembly with resources only, no executable code

* **What are Serialization and Deserialization? What are the types of serialization**

**Serialization:**

Serialization is the process of converting an object into a stream of bytes or a string representation that can be easily stored or transmitted. This allows the object to be saved to a file, sent over a network, or stored in a database.

Serialization is the process of converting an object into a format that can be stored, transmitted, or reconstructed later.

**Deserialization:**

Deserialization is the process of reconstructing an object from its serialized form. It involves reading the serialized data and converting it back into an object with the same state and structure as the original object.

Deserialization is the process of converting serialized data, such as binary/ XML/ json data, back into an object.

**Types of serialization:**

**Binary Serialization:** Binary serialization converts objects into a binary format, representing the object's state as a sequence of bytes. This format is efficient and compact but not human-readable. It's suitable for storing data on disk or transmitting data between systems.

**XML Serialization:** XML (Extensible Markup Language) serialization represents objects as XML documents. It converts object data into XML format, making it human-readable and platform-independent. XML serialization is often used for configuration files, web services, or interoperability between different systems.

**JSON Serialization:** JSON (JavaScript Object Notation) serialization converts objects into JSON format, which is lightweight, human-readable, and easy to parse. JSON serialization is commonly used in web development, APIs (Application Programming Interfaces), and data exchange between web clients and servers.

* **What is Garbage Collection(GC)?**

The garbage collector (GC) **manages** the allocation and release of memory in .NET framework.

Garbage collection is one of the responsibilities of CLR only.

Garbage collection release the objects which are no longer needed.

.NET framework has a mechanism to decide which objects are no longer needed and that mechanism we call generations.

Generation is a mechanism to collect the short-lived objects more frequently than the longer-lived object.

* **Using keyword in GC**

The using keyword in C# is used to ensure that the Dispose method of an IDisposable object is called when the object is no longer needed. This is particularly useful for managing unmanaged resources like file handles, database connections, or network streams, which the garbage collector does not automatically clean up.

* **What is the difference between “Dispose” and “Finalize”?**

Both Dispose and Finalize methods are used to release the unmanaged objects which are no longer required.

Finalize method is called automatically by the garbage collector.

But the Dispose method is called explicitly by the code to release any unmanaged object

OR

The Dispose method is called explicitly by client code to release resources when they are no longer needed. It can be called implicitly using the statement, which ensures that the Dispose method is called when the object goes out of scope.

On the other hand, the Finalize method is used to perform cleanup operations on an object just before it is garbage collected.

The main difference between the two methods is that the Dispose method is deterministic and can be explicitly called by client code. In contrast, the Finalize method is non-deterministic and is called by the garbage collector at an undetermined time.

It is important to note that objects that implement the Dispose method should also implement the Finalize method as a backup mechanism in case the client code does not call the Dispose method.

* **What is the difference between Process and Thread**

A process is an instance of a program with its own memory space and system resources.

A thread is the smallest unit of process, that shares memory and resources with other threads within the same process.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Process** | **Thread** |
| **Definition** | Independent program instance | Smallest unit of execution within a process |
| **Isolation** | Isolated memory and resources | Shared memory and resources within a process |
| **Resource Allocation** | Each process has its own memory and resources | Threads share memory and resources |
| **Overhead** | High (due to separate memory space and IPC) | Low (due to shared memory and resources) |
| **Communication** | Requires IPC mechanisms | Direct communication through shared memory |
| **Concurrency** | Can run multiple processes concurrently | Can run multiple threads concurrently |
| **Stability** | Crash in one process doesn't affect others | Crash in one thread can affect the entire process |

* **Explain Multithreading**

Multithreading refers to the ability to execute multiple threads of code concurrently within a single process.

Multithreading allows you to perform multiple tasks simultaneously, such as downloading data while displaying a progress bar.

To create multithreaded application in C#, we need to use SYSTEM.THREADING namespace.

* **What is the difference between synchronous and asynchronous programming?**

**Synchronous Programming**:

In synchronous programming, tasks are executed one after the other, in sequential order.

When a task is started, the program waits for it to finish before moving on to the next task.

Tasks are executed in a blocking manner, meaning that the program waits for each task to complete before proceeding to the next one.

**Asynchronous Programming:**

In asynchronous programming, tasks are executed independently and concurrently, allowing multiple tasks to run simultaneously.

When a task is started, the program does not wait for it to finish before moving on to the next task.

Tasks are executed in a non-blocking manner, meaning that the program can continue to execute other tasks while waiting for certain operations to complete asynchronously.

* **What is the role of Async and Await**

The async keyword is used to mark a method as an asynchronous method.

The await keyword is used to pause the execution of an asynchronous method until a specified operation completes.

The await keyword is used inside an async method to pause its execution until the awaited task completes.

**Async**:-

Definition: The async keyword is used to mark a method as asynchronous. It allows the method to contain await expressions, enabling asynchronous operations within it.

Method Signature: An async method typically has a return type of Task, Task<T>, or void. For event handlers, void is used, but for other methods, it's recommended to use Task or Task<T> to allow callers to await the method.

**Await**:-

Definition: The await keyword is used to pause the execution of an async method until the awaited task completes. It asynchronously waits for the task to finish and then resumes execution of the method.

Non-Blocking: While the method is paused, the calling thread is free to do other work, making the application more responsive.

* **Design Patterns**

In C#, design patterns are reusable, general solutions to common problems that arise during software development. Design patterns provide templates and guidelines for structuring code in a way that promotes flexibility, maintainability, and scalability.

**Creational Patterns:** They help in managing how objects are created, ensuring that the process is flexible and reusable. Think of them as blueprints for constructing different types of objects.

**Singleton pattern:** The Singleton pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to that instance. It is useful when exactly one object is needed to coordinate actions across the system.

Implementation Guidelines:

Private Parameterless Constructor: Create a private constructor to prevent external instantiation.

Sealed Class: Declare the class as sealed to prevent inheritance.

Static Variable: Use a private static variable to hold the single instance.

sealed class Logger

{

private static Logger \_instance;

// Private constructor

private Logger() { /\* Initialization logic \*/ }

public static Logger Instance

{

get

{

if (\_instance == null)

\_instance = new Logger();

return \_instance;

}

}

// Other methods and properties...

}

**Structural Patterns**: These patterns help in organizing classes and objects. They help ensure that if one part of a system changes, the entire system doesn't need to do so. Think of them as the framework that holds everything together, ensuring that changes in one part don't break the whole system.

**Behavioral Patterns**: These patterns deal with how objects interact with each other. They define how different parts of a system communicate and collaborate. Think of them as the rules of engagement between objects, guiding their interactions in a flexible and efficient way.

* **What are partial classes in C#? Why do we need partial classes?**

A partial class is only used to split the definition of a class into two or more classes in the same source code file or more than one source file. You can create a class definition in multiple files, which will be compiled as one class at run time. Also, when you create an instance of this class, you can access all the methods from all source files with the same object.

* **Difference between Generic and non generic:**

With help of Generics, one needs to write a method/class/interface only once and use it for any type whereas, in non-generics, the code needs to be written again and again whenever needed

* **Difference between ToString() and Convert.ToString():**

The ToString() method and the Convert.ToString() method in C# both convert an object to its string representation, but they have key differences in terms of usage and behavior.

**ToString()**

Definition: A method that is called on an instance of an object to convert it to a string representation.

Usage: Can be used directly on any object.

Null Handling: Throws a NullReferenceException if the object is null.

Custom Implementation: Can be overridden in custom classes to provide a specific string representation.

int number = 123;

string result = number.ToString(); // Converts the integer to a string "123"

**Convert.ToString()**

Definition: A static method that converts an object to a string.

Usage: Used to convert an object to a string, handling null values gracefully.

Null Handling: Returns null if the object is null.

Flexibility: Can handle various types and null values without throwing an exception.

int number = 123;

string result = Convert.ToString(number); // Converts the integer to a string "123"

* **What is the difference between a struct and a class in C#?**

|  |  |
| --- | --- |
| **Struct** | **Class** |
| A struct is a value type.  Instances of structs are stored on the stack, and their memory is directly allocated where they are declared.  Suitable for small, lightweight objects. | A class is a reference type.  Instances of classes are stored on the heap, and variables store references (memory addresses) to the actual objects.  Suitable for larger, more complex objects. |
| Structs are usually immutable (their state cannot be changed after creation). | Classes are mutable by default, meaning their state can be modified after creation. |
| Structs do not support inheritance. They cannot be derived from other structs or classes. | Classes support inheritance. They can be used as base classes for other classes, allowing for the creation of class hierarchies. |
| Generally, structs can offer better performance for small, frequently used objects because of their value type nature. They avoid heap allocations and are stored inline. | Classes may involve more overhead due to heap allocations and references. They are suitable for larger objects and situations where mutability is required. |

**SQL – Basics**

* **What is the difference between DBMS and RDBMS?**

**DBMS (Database Management System):**

A DBMS is a software system that allows users to define, create, manage, and manipulate databases.

Examples of DBMS include Microsoft Access, SQLite, MongoDB, and Redis.

DBMS stores data as file. No relationship between data. Normalization is not present. It deals with small quantity of data

**RDBMS (Relational Database Management System):**

An RDBMS is a specific type of DBMS that is based on the relational model of data, introduced by E.F. Codd.

In an RDBMS, data is organized into tables with rows and columns. It enforces relationships between tables using constraints such as primary keys, foreign keys, and unique keys.

Examples of RDBMS include Oracle Database, MySQL, Microsoft SQL Server, PostgreSQL, and SQLite (which can function as both a DBMS and RDBMS depending on how it's used).

RDBMS stores data in **TABULAR** form. Data is stored in the form of tables which are **RELATED** to each other. Eg: Foreign key relationship. **NORMALIZATION** is present. It deals with **LARGE** amount of data.

* **What is a Constraint in SQL? What are the types of constraints**

**PRIMARY KEY constraint:** In SQL, a PRIMARY KEY constraint is a column or a set of columns in a table that uniquely identifies each record in that table. The primary key must contain unique values, and it cannot contain NULL values. The primary key plays a crucial role in ensuring data integrity and facilitating relationships between tables.

**UNIQUE constraint:** In SQL, the UNIQUE constraint is used to ensure that all values in a specified column or a set of columns are unique within a table. Unlike the primary key constraint, a unique constraint allows NULL values, and it does not automatically create an index on the column(s). However, only one NULL value is allowed in the column(s) when a unique constraint is applied.

**Foreign key constraint:** A foreign key is a column or set of columns in one table that refers to the primary key in another table. It establishes a link between the two tables, enforcing referential integrity.

**Composite key constraint:** A composite key consists of multiple columns that, together, uniquely identify each record in a table. The combination of values in the composite key must be unique.

**CHECK constraint:**

In SQL, a CHECK constraint is a type of constraint that is used to specify a condition that must be true for each row in a table. The CHECK constraint ensures that the values in a particular column or a combination of columns meet certain conditions defined by an expression.

**NOT NULL** constraint tells that we cannot store a null value in a column

**DEFAULT** constraint specifies a default value for the column when no value is specified by the user.

* **What are Triggers and types of triggers**

Triggers are stored programs, which are **AUTOMATICALLY** executed or fired when some events (insert, delete and update) occur. Triggers are commonly used to enforce business rules, maintain data integrity, and automate certain database-related tasks. There are two main types of triggers: BEFORE triggers and AFTER triggers.

**BEFORE Triggers:** A BEFORE trigger is executed before the event that triggers it (e.g., before an INSERT, UPDATE, or DELETE operation). It can be used to validate or modify the data before it is actually modified in the table.

**AFTER Triggers:** An AFTER trigger is executed after the event that triggers it (e.g., after an INSERT, UPDATE, or DELETE operation). It can be used to perform additional actions, such as logging, updating other tables, or sending notifications, after the data has been modified.

* **Types of Views :**

In SQL, a view is a virtual table that is based on the result of a SELECT query. Unlike a physical table, a view does not store the data itself but provides a way to represent the result of a query as if it were a table. Views are used to simplify complex queries, provide a layer of abstraction over the underlying tables, and control access to specific columns or rows of a table.

**Simple Views**: Simple views are based on a single table or a straightforward SELECT query.

They don't involve complex logic such as joins, subqueries, or aggregations.

**Complex Views**: Complex views involve multiple tables, join operations, subqueries, or aggregations.

They are used to encapsulate more complex logic and provide a consolidated view of data from multiple sources.

**Indexed Views (Materialized Views):** Indexed views, also known as materialized views, store the result set physically, allowing for faster data retrieval.

They are particularly useful for scenarios where the underlying data changes infrequently compared to the frequency of data retrieval.

**Updatable Views:** Updatable views allow modifications (INSERT, UPDATE, DELETE) through the view, and the changes are reflected in the underlying tables.

Updatability depends on certain conditions, such as the absence of certain constructs in the view definition.

**Partitioned Views:** Partitioned views are used for managing large datasets by dividing them into smaller, more manageable pieces or partitions.

Each partition is stored in a separate table or file.

**Schema-Binding Views**: Schema-binding views are views that are bound to the schema of the underlying tables. This means that changes to the underlying tables require corresponding changes to the view.

* **What is the difference between Having clause and Where clause?**

WHERE Clause is used before GROUP BY Clause.

HAVING Clause is used after GROUP BY Clause.

WHERE Clause cannot contain AGGREGATE function.

HAVING Clause can contain aggregate function

* **JOINS in SQL**

A join clause is used to **combine** rows from two or more tables, based on a related column between them.

**INNER JOIN** : The INNER JOIN in SQL is used to combine rows from two or more tables based on a related column between them. It returns only the rows where there is a match in both tables.

**LEFT OUTER JOIN (or LEFT JOIN):** Returns all rows from the left table and the matched rows from the right table. If there is no match, NULL values are returned for columns from the right table.

**RIGHT OUTER JOIN (or RIGHT JOIN**): Returns all rows from the right table and the matched rows from the left table. If there is no match, NULL values are returned for columns from the left table.

**FULL OUTER JOIN (or FULL JOIN):** Returns all rows when there is a match in either the left or the right table. If there is no match, NULL values are returned for columns from the table without a match.

**SELF-JOIN:** A self-join in SQL is a regular join, but the table is joined with itself. It is used when you want to compare rows within the same table. To perform a self-join, you need to use aliases to differentiate between the two instances of the same table.

**Cross Join :** A cross join in SQL, also known as a Cartesian join, returns the Cartesian product of the two tables involved. This means it combines all rows from the first table with all rows from the second table, resulting in a result set that contains every possible combination of rows.

**UNION**

Syntax: SELECT columns FROM table1 UNION SELECT columns FROM table2 ...

UNION merges the results of two or more SELECT statements into a single result set.

It removes duplicate rows from the combined result set. Only unique rows are retained.

**UNION ALL**

Syntax: SELECT columns FROM table1 UNION ALL SELECT columns FROM table2 ...

UNION ALL also merges the results of two or more SELECT statements into a single result set.

Unlike UNION, it retains all rows from the combined result set, including duplicate rows.

* **What are Indexes in SQL Server?**

SQL Indexes are used in relational databases to retrieve data **VERY FAST.**

Indexes in SQL are database objects that provide a fast and efficient way to look up rows in a table based on the values in one or more columns. Indexes play a crucial role in improving query performance by speeding up data retrieval operations.

Indexes provide a way to locate data without having to scan the entire table, resulting in significant performance improvements for SELECT, UPDATE, DELETE, and JOIN operations.

**Clustered Indexes :**

A clustered index determines the physical order of data in a table.

Each table can have only one clustered index.

In SQL Server, if you set a primary key on a column, then it will **automatically create** a clustered index on that particular column

**Non-Clustered Indexes**

A non-clustered index is stored at one place and table data is stored in another place.

Each table can have multiple non-clustered indexes.

In a non-clustered index, a separate structure (usually a B-tree) is created to store the index key values along with pointers to the corresponding rows in the table.

* **What is transaction in sql**

A transaction in SQL is like a bundle of actions that you want to treat as a single unit. It's a way to ensure that a group of related operations either all happen together successfully, or none of them happen at all.

The properties of a transaction are typically summarized by the acronym ACID:

**Atomicity**: A transaction is atomic, meaning that it is treated as a single indivisible unit of work. Either all of the operations in the transaction are completed successfully, or none of them are. There is no partial completion.

**Consistency**: Transactions preserve the consistency of the database. This means that the database remains in a valid state both before and after the transaction, regardless of any failures that may occur during the transaction.

**Isolation**: Transactions are isolated from each other, meaning that the changes made by one transaction are not visible to other transactions until the first transaction is committed. This ensures that transactions do not interfere with each other.

**Durability**: Once a transaction is committed, its changes are permanently saved to the database even in the event of a system failure. This ensures that the changes made by committed transactions are not lost.

* **What is stored procedure in sql**

A stored procedure in SQL is a collection of one or more SQL statements or commands that are stored in the database and can be executed at a later time. Stored procedures offer several advantages, including code reuse, modularity, and improved performance. They are widely used for encapsulating business logic, performing data operations, and automating tasks within a database. Stored procedures can accept input parameters, perform operations such as querying or modifying data, and return result sets or output parameters.

* **What is function in sql**

In SQL, a function is a database object that encapsulates a series of SQL statements into a single, reusable unit.

Functions can accept input parameters, perform calculations or operations, and then return a result.

Functions can be categorized into two main types: scalar functions and table-valued functions.

**Scalar Functions:**

Scalar functions return a single value, and they can be used in SQL expressions, SELECT statements, WHERE clauses, and other parts of queries. Examples of scalar functions include mathematical functions, string functions, and date functions.

**Table-Valued Functions:**

Table-valued functions return a table as their result. They can be used in the FROM clause of a SELECT statement. There are two main types of table-valued functions: inline table-valued functions and multi-statement table-valued functions.

* **What is the difference between Stored Procedure and Functions**

**Stored Procedure**

SP may or may not return a value

Can have input/output parameters

We can call function inside SP

We cannot use SP in SQL statements like SELECT, INSERT, UPDATE, DELETE, MERGE, etc.

We can use try-catch exception handling in SP

We can use transactions inside SP.

**Functions**

Function must return a value

Only has input parameters

Cannot call SP inside a function

We can use them with function. SELECT \*, dbo.fnCountry(city.lon g) FROM city;

We can not use try-catch in functions

We can not use transactions inside functions.

* **What is the difference between scope\_identity and @@identity?**

@@IDENTITY function returns the last identity value generated within the **current session**, regardless of the scope.

The scope\_identity() function returns the last identity created in the same session and the same **scope**.

* **What is CTE in SQL Server**

In SQL Server, CTE stands for Common Table Expression. It's a temporary result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs are defined using the WITH clause and can simplify complex queries by breaking them down into smaller, more manageable parts.

* **What is Cursor**

In SQL, a cursor is a database object used to retrieve, manipulate, and navigate through a result set row by row. Unlike standard SQL queries that operate on the entire result set at once, cursors allow for row-by-row processing. This can be useful when you need to perform operations that are dependent on the specific values of each row.

* **What is the difference between Delete, Truncate and Drop commands**

**Delete**

It is a DML**.** It is used to delete one or all rows from the table based on where condition, but it will not delete schema.It can be rollback.

**Truncate**

Itis a DDL. It is used to delete all rows from the table, but it will not delete schema. It cannot be rollback.

**Drop**

It is a DDL. It is used to delete all rows from the table with structure/schema. It cannot be rollback

* **What are Magic Tables in SQL Server**

**Magic tables** are the temporary logical tables that are created by the SQL server, whenever there are **insertion or deletion or update**( D.M.L) operations.

**INSERTED –** When any insert query executed, then the recently inserted row gets added to the INSERTED magic table

**DELETED –** When any delete query executed, then the recently deleted row gets added to the DELETED magic table.

* **Select second highest salary**

SELECT MAX(Salary) AS SecondHighestSalary

FROM Employees

WHERE Salary < (SELECT MAX(Salary) FROM Employees);

* To create a new table in SQL that copies the structure (columns, data types, constraints, etc.) of an existing table but doesn't contain any data, you can use the CREATE TABLE statement with the LIKE clause. The LIKE clause allows you to copy the structure of an existing table to the new table.

-- Step 1: Create a new table (NewTable) with the same structure as an existing table (SourceTable)

CREATE TABLE NewTable AS

SELECT \*

FROM SourceTable

WHERE 1 = 0; -- Ensures that no data is copied initially

-- Step 2: Copy data from SourceTable to NewTable

INSERT INTO NewTable

SELECT \*

FROM SourceTable;

* **To update the revised\_date column in the salary table by adding 2 months to the transaction\_date column, you can use the DATEADD function**

UPDATE salary SET revised\_date = DATEADD(month, 2, transaction\_date);

* **Find the 3rd highest salary**

WITH RankedSalaries AS (

SELECT Salary, DENSE\_RANK() OVER (ORDER BY Salary DESC) AS Rank

FROM Employees

)

SELECT Salary AS ThirdHighestSalary

FROM RankedSalaries

WHERE Rank = 3;

* **To get the latest salary for each employee based on the credited\_date.**

SELECT \* FROM emp e JOIN salary s ON e.emp\_id = s.emp\_id WHERE s.credited\_date

= (SELECT MAX(s2.credited\_date) FROM salary s2 WHERE s2.emp\_id = e.emp\_id);

* **Normalization in sql**

Normalization is the process of organizing data in a database. It includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency

**First Normal Form (1NF):** This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.

**Second Normal Form (2NF**): 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.

**Third Normal Form (3NF):** 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column should be directly related to the primary key, and not to any other columns in the same table.

**Boyce-Codd Normal Form (BCNF**): BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.

**Fourth Normal Form (4NF):** 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.

**Fifth Normal Form (5NF):** 5NF is the highest level of normalization and involves decomposing a table into smaller tables to remove data redundancy and improve data integrity.

* **WITH (NOLOCK) :**

In SQL Server, the WITH (NOLOCK) hint is used to specify that a SELECT statement should not issue shared locks to read data. It allows the query to read data without waiting for other transactions to release locks, potentially improving the query's performance. However, it comes with potential risks and should be used carefully.

SELECT column1, column2

FROM YourTable WITH (NOLOCK)

WHERE your\_condition;

* Deleting duplicate records from a table can be achieved using various methods. I'll provide a few alternative approaches below:

**Method 1**: **Using GROUP BY and HAVING**

DELETE FROM YourTable

WHERE DuplicateColumn IN (

SELECT DuplicateColumn

FROM YourTable

GROUP BY DuplicateColumn

HAVING COUNT(\*) > 1

);

This method uses the GROUP BY clause along with HAVING to identify and delete duplicate records based on a specific column.

**Method 2:** **Using Self-Join**

DELETE t1

FROM YourTable t1

JOIN YourTable t2 ON t1.DuplicateColumn = t2.DuplicateColumn

WHERE t1.PrimaryKeyColumn > t2.PrimaryKeyColumn;

This method uses a self-join to compare records based on a specific column (DuplicateColumn) and deletes the records with higher primary key values, keeping only the earliest occurrence.

**Method 3:** **Using a Temporary Table**

SELECT DISTINCT \*

INTO #TempTable

FROM YourTable;

TRUNCATE TABLE YourTable;

INSERT INTO YourTable

SELECT \* FROM #TempTable;

DROP TABLE #TempTable;

This method involves creating a temporary table with distinct records, truncating the original table, and then inserting the distinct records back.

* **To fetch 50% of the records from a table**

SELECT \* FROM YourTable

FETCH FIRST 50 PERCENT ROWS ONLY;

SELECT \* FROM CUST

WHERE ROWNUM <= (SELECT COUNT(\*) / 2 FROM CUST);

* **Rank() and DensRank()**

In SQL, RANK() and DENSE\_RANK() are both window functions that assign ranks to rows within a result set based on the values of a specified column or expression. They are commonly used in scenarios where you need to assign unique ranks to rows, such as sorting results or identifying top N entries. The primary difference between the two functions lies in how they handle ties (rows with the same values).

**RANK()**

The RANK() function assigns a unique rank to each row within a partition of the result set. If two or more rows have the same value, they receive the same rank, and the subsequent rank(s) are skipped.

SELECT SalesPerson, Amount, RANK() OVER (ORDER BY Amount DESC) AS Rank

FROM Sales;

**DENSE\_RANK()**

The DENSE\_RANK() function also assigns ranks to rows, but it does not skip any ranks when there are ties. This means rows with the same value receive the same rank, and the next rank follows sequentially.

SELECT SalesPerson, Amount, DENSE\_RANK() OVER (ORDER BY Amount DESC) AS DenseRank

FROM Sales;

**ASP.NET – MVC - Part 1**

* **Explain the ASP.NET Core MVC architecture.**

ASP.NET Core MVC architecture is based on the Model-View-Controller (MVC) pattern, cleanly separating concerns in your web application.

Models represent data, Views handle presentation, and Controllers manage the interaction between them.

Models are the building blocks, Views are the blueprints, and Controllers are the construction crew, working together to create a beautiful and functional web experience.

This separation improves code maintainability, testability, and scalability, making your applications well-organized and flexible.

**Models**: Represent the data and logic of your application. They define the entities and their relationships, encapsulate business rules, and handle data access. Model classes typically interact with databases or other data sources.

**Views**: Responsible for presenting the user interface. They use technologies like HTML, CSS, and Razor to render the data received from the models in a visually appealing way. Views do not know the application logic and only focus on presentation.

**Controllers**: Act as the entry point for user interaction and orchestrate the flow of the application. They receive user requests, process them using the models, and ultimately choose which view to render. Controllers interact with both models and views, but never directly with the user interface.

* **What is the Middleware processing**

In ASP.NET Core, middleware is a key concept that allows you to customize the request-handling pipeline. The middleware components in ASP.NET Core are arranged in a pipeline and are responsible for processing requests and responses as they flow through the application. Middleware provides a modular and flexible way to handle various concerns in the web application. Here are some common use cases for middleware in ASP.NET Core:

**Request Processing:** Middleware components can intercept and process incoming HTTP requests before they reach the application's endpoints. This allows for operations such as request logging, authentication checks, or modifying the request headers.

**Response Processing:** Similarly, middleware can process the outgoing HTTP response before it is sent back to the client. This can include operations like adding headers, compressing content, or logging response details.

**Routing:** The built-in routing middleware is responsible for matching incoming requests to the appropriate controller and action based on the URL. It allows you to define URL patterns and route requests accordingly.

**Authentication and Authorization:** Middleware components handle authentication and authorization tasks. ASP.NET Core provides authentication middleware that can authenticate users based on various schemes (e.g., cookies, tokens) and authorization middleware for controlling access to resources.

Authentication is the process of verifying who a user is, while authorization is the process of verifying what they have access to.

**Exception Handling:** Middleware can catch exceptions that occur during request processing.

**Static File Serving:** The static file middleware serves static files (e.g., images, stylesheets, scripts) directly without involving the application logic. It is useful for efficiently serving static assets.

**CORS (Cross-Origin Resource Sharing):** Middleware can handle CORS policies by adding appropriate headers to responses, allowing or restricting cross-origin requests based on the defined policies.

**Logging:** Middleware can log information about requests and responses. This is helpful for debugging, monitoring, and performance analysis. ASP.NET Core includes built-in logging middleware.

**Dependency Injection:** Middleware components can use dependency injection to access services provided by the application's dependency injection container. This allows middleware to leverage other services, such as database access or custom business logic.

**Content Compression:** Middleware can compress response content to reduce bandwidth usage and improve application performance. The UseResponseCompression middleware, for example, enables response compression based on client preferences.

**Health Checks:** Middleware can be used to implement health checks to monitor the status of the application and its dependencies. This is useful for deployment and monitoring scenarios.

* **What is Dependency Injection**

Dependency Injection (DI) is a programming pattern that helps make your code more flexible, maintainable, and testable. **/**

Dependency Injection (DI) is a design pattern and technique in software development where the dependencies of a class (or an object) are provided from the outside rather than being created within the class itself. The main goal of Dependency Injection is to promote loose coupling between components, improve maintainability, and enhance testability of the code.

**Here's a simple analogy:**

**Without Dependency Injection (DI):**

Imagine you want to make a sandwich.

You go to the kitchen, get all the ingredients, and make the sandwich yourself.

This is like a class creating its own dependencies directly.

**With Dependency Injection (DI):**

Now, instead of going to the kitchen yourself, you ask someone else (let's call them the "Sandwich Provider") to make the sandwich for you.

The Sandwich Provider knows how to make sandwiches and gives you the finished one.

This is like a class asking someone else (DI container or framework) to provide its dependencies.

**In programing language:**

The class that needs something (like the sandwich) is often called the "dependent" or "client" class.

The thing it needs (like the sandwich) is called a "dependency" or "service."

So, Dependency Injection is like asking someone else to provide the things your class needs rather than creating them directly. This makes your code more modular, easier to understand, and allows for changes without modifying a lot of code. It also helps a lot when you want to test your code because you can easily replace real dependencies with fake ones for testing purposes.

Here are the key concepts of Dependency Injection:

**Dependent Class:** This is the class or object that requires the services or functionalities of another class (dependency) to perform its tasks.

**Dependency:** This is the class or service that the dependent class relies on to accomplish its functionality. The dependency is "injected" into the dependent class.

**Injection:** Injection refers to the process of providing dependencies to a class. This can be done through the class constructor (constructor injection), through methods (method injection), or by setting properties (property injection).

**Inversion of Control (IoC):** Dependency Injection is often associated with the broader concept of Inversion of Control. In IoC, the control over the flow of a program is inverted compared to traditional programming models. Instead of a class controlling the creation of its dependencies, the control is handed over to an external entity, often a container or framework.

**DI Container:** A Dependency Injection container or IoC container is a tool that manages the instantiation and lifetime of objects (dependencies) and their injection into dependent classes. Examples include the built-in DI container in .NET Core and popular third-party containers like Autofac, Ninject, or Unity.

* **Lifetime of DI:**

In .NET Core, Dependency Injection (DI) is a technique for achieving Inversion of Control (IoC) between classes and their dependencies. It allows for better modularity and testability of your code. When configuring services in DI, you can specify their lifetimes using three primary lifetimes: Singleton, Transient, and Scoped.

**Singleton**

Definition: A Singleton service is created once and shared throughout the application's lifetime.

Usage: Use Singleton when you want to share the same instance for the entire application.

**Transient**

Definition: A Transient service is created each time it is requested.

Usage: Use Transient services when you need a new instance every time the service is requested.

**Scoped**

Definition: A Scoped service is created once per request.

Usage: Use Scoped services when you want to create a new instance for each HTTP request but reuse the same instance within the same request.

When to Use Each

Singleton: Use for services that maintain state across the entire application and do not need to be re-initialized (e.g., configuration settings, logging services).

Transient: Use for lightweight, stateless services that can be created and discarded without any performance concerns (e.g., utility classes, data processing services).

Scoped: Use for services that should be unique per request but can be reused within the same request (e.g., database contexts in web applications).

* **What is a Request delegate and how is it used?**

In ASP.NET Core, a Request delegate is a function that processes and handles an incoming HTTP request. It's the core building block of the request processing pipeline, which is essentially a series of middleware components that handle the request one after the other.

* **Can we change the name of startup.cs class?**

In an ASP.NET Core application, while it's convention to have the Startup.cs class as the default entry point for configuring the application, you can technically change the name of this class. However, doing so requires additional configuration.

**Change the Class Name:**

Rename the Startup.cs file to your desired name, for example, MyCustomStartup.cs.

Update the Program.cs File:

In the Program.cs file, you need to update the WebHost.CreateDefaultBuilder(args) line to specify the custom startup class name.

* **What is the difference between run and use?**

Run() will end the request, and app. Use() will pass the request to next middleware.

* **What is Routing in MVC**

Routing in MVC is the process of mapping a URL request to a specific controller action in a web application.

The routing system is responsible for directing incoming requests to the appropriate controller, based on the URL.

The routing system is typically configured using a **routing table**, which maps URLs to controller actions.

**Types of routing :-**

**Convention-based Routing:** This is the default routing mechanism used in ASP.NET MVC. Convention-based routing maps URLs to controller actions based on a set of predefined conventions.

**Attribute Routing:** Attribute routing allows you to define routes directly on the controller actions or controller classes using attributes. This gives you more control and flexibility over the routing behavior. With attribute routing, you can define custom URL patterns for individual actions or entire controllers. Attribute routing is particularly useful when you need to define complex or non-standard URL patterns.

**Dynamic (Custom) Routing:** Dynamic routing involves creating custom route definitions in the RouteConfig.cs file or using route constraints. This approach allows you to define custom routing rules based on specific requirements.

* **What is the Routing Table in ASP.NET MVC?**

In ASP.NET MVC, the routing table is a crucial component that maps incoming HTTP requests to the appropriate controller actions. Routing determines how URLs are matched to the corresponding actions in your controllers.

How Routing Works

Request Handling: When an HTTP request arrives, ASP.NET MVC uses the routing table to determine how to handle it.

The routing table is essentially a collection of routes that the application uses to match URLs to controller actions.

Route Definition: Routes are defined in the RouteConfig class, typically found in the App\_Start folder of an ASP.NET MVC project.

Each route consists of a URL pattern and various constraints and defaults.

Basic Route Configuration

Routes are registered in the RouteConfig class inside the RegisterRoutes method. This method is called at application startup.

* **What is the difference between ViewData, ViewBag & TempData?**

ViewData, ViewBag, and TempData are all ways to **pass data** between a controller and a view.

ViewData and ViewBag are used to transfer data from controller to view.

TempData is used to pass data from controller to controller(pass data between two action methods).

ViewBag doesn’t require typecasting, whereas ViewData require the typecasting

Keep(): Keeps all items in TempData for an additional request.

Keep(string key): Keeps a specific item in TempData for an additional request.

Peek(): Reads an item from TempData without marking it for deletion.

* **What is Partial View?**

A partial view is a reusable view component that can be rendered within other views or shared across multiple pages in a web application.

It helps us to reduce code duplication.

* **How Validation works in MVC? What is Data Annotation**

[Required]: Specifies that a property is required and cannot be null or empty.

[StringLength]: Specifies the minimum and maximum length of a string property.

[RegularExpression]: Specifies a regular expression pattern that a string property must match.

[EmailAddress]: Specifies that a string property must be a valid email address format.

[Range]: Specifies the minimum and maximum allowed values for a numeric property.

[Compare]: Compares the value of a property with the value of another property in the same model.

[DataType]: Specifies the data type of a property, which can be used for formatting and validation purposes.

[Display]: Provides metadata for displaying the property, such as the name and order of the property in a view.

[Editable]: Specifies whether a property can be edited or displayed in a view.

[ReadOnly]: Specifies that a property can be displayed in a view but cannot be edited.

[HiddenInput]: Specifies that a property should be hidden in a view.

[DisplayName]: Provides a friendly name for a property that can be used in a view.

[DisplayFormat]: Specifies a format string for displaying the value of a property.

[RequiredIf]: Specifies that a property is required if another property in the model has a specific value.

[CreditCard]: Specifies that a string property must be a valid credit card number format.

* **Explain the concept of MVC Scaffolding?**

In ASP.NET MVC, scaffolding is a technique that **generates code** for common web application.

The scaffolding code can include controllers, views, and data access code automatically generated that can help developers to create web applications more quickly and with less manual coding.

* **What is bundling?**

Bundling in ASP.NET MVC is a technique used to improve the performance of web applications by reducing the number of HTTP requests and optimizing the delivery of CSS and JavaScript files. Bundling allows you to group multiple files into a single bundle, which can then be minified and served as a single request. This reduces load times and improves the performance of your web application

* **What is the difference between Server.Transfer() and Response.Redirect()**

Both Response.Redirect() and Server.Transfer() methods are used to **redirect** the user's browser from one page to another page.

**Response.Redirect** uses **round trip** back to the client for redirecting the page.

It is a **slow** technique, but it maintains the URL history in the client browser for all pages.

In **Server.Transfer** page processing transfers from one page to the other page without making a round-trip back to the client's browser.

It is a **faster** technique, but it does notmaintain the URL history in the client browser for all pages

* **What are the different types of Caching?**

**Page Output Caching**: Page output caching allows you to cache the output of an **entire page**. This is useful when a page contains all static content.

**Fragment Caching:** Fragment caching allows you to cache a **portion or fragment** of a page. This is useful when a page contains dynamic content that can be separated from the rest of the page.

**Data Caching**: Data caching allows you to cache **data** retrieved from a database or other data source. This can improve performance by reducing the number of database calls.

* **What are the types of state management**

State management refer to **storing** data that is required by a web application to track information between user requests and server responses.

**Cookies**: Cookies is a mechanism for storing small amounts of data on the client's machine. For example, it is used to save usernames.

**Hidden Fields**: Hidden fields are HTML form elements that can be used to store data on the client side. They are often used to pass data between pages.

**ViewState:** ViewState is a mechanism that stores the data on the client side as a hidden field. ViewState is useful for persisting data between postbacks in a WebForms application.

**QueryString:** QueryString is a mechanism for passing data from one page to another through the URL. But it is not secure as it is visible to all.

Server-side state management techniques:

**Session State**: Session state is a server-side mechanism that allows you to store user-specific data on the server. Session state uses a session ID to associate data with a specific user and can be used to persist data between page requests.

**Application State**: Application state is a server-side mechanism that allows you to store data that is shared by all users of an application.

**Caching**: Cache is a server-side mechanism that allows you to store frequently accessed data in memory.

* **What are the different ways to store session state in asp.net?**

Session state is a mechanism that enables you to store data on server side for multiple requests. It allows you to persist data between pages.

**In-Process Session State**: This is the default session state management mode in ASP.NET, where session data is stored in memory on the same **web server**.

**State Server Session State:** This mode stores session data in a separate process called the ASP.NET State Service, which runs **outside** the web server process.

**SQL Server Session State**: This mode stores session data in a **SQL Server database**, which provides the highest level of reliability and scalability.

* **Expalin SOLID Principles**

**Solid Principle:**

The SOLID principles are a set of five design principles for writing maintainable and scalable software. Are widely used in object-oriented programming. The SOLID acronym stands for:

**Single Responsibility Principle (SRP):**

A class should have only one reason to change, meaning that it should have only one responsibility.

This principle encourages a class to have a single job or responsibility, making it easier to understand, maintain, and change.

**Open/Closed Principle (OCP):**

Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification.

This principle encourages the use of abstraction and polymorphism, allowing new functionality to be added without altering existing code.

**Liskov Substitution Principle (LSP):**

Subtypes must be substitutable for their base types without altering the correctness of the program.

This principle ensures that objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.

All the base class method must be applicable for the derived class.

**Interface Segregation Principle (ISP):**

A class should not be forced to implement interfaces it does not use.

This principle suggests that it is better to have multiple small, specific interfaces rather than a large, general-purpose one, to avoid forcing classes to implement unnecessary methods.

**Dependency Inversion Principle (DIP):**

High-level modules should not depend on low-level modules. Both should depend on abstractions.

Abstractions should not depend on details; details should depend on abstractions.

This principle encourages the use of dependency injection and inversion of control to achieve a flexible and decoupled architecture.

* **Use of startup.cs class**

The Startup.cs class in .NET Core is where you configure services, middleware, and other aspects of your application. It provides a central place to set up the application and define its behavior during startup

In .NET Core, the Startup.cs class is a fundamental part of the application's configuration and initialization process. It is used to set up and configure various aspects of the application, including services, middleware, and the request processing pipeline. Here are some key aspects of the Startup.cs class in a .NET Core application:

**Configure Services:**

In the ConfigureServices method, you define and configure the services that your application will use. This includes services such as dependency injection, database connections, authentication, and more. Services are registered with the built-in dependency injection container.

**Configure Middleware:**

The Configure method is where you set up the middleware components that make up the request processing pipeline. Middleware components are executed in the order they are added, and they handle various aspects of the HTTP request-response cycle.

**Configuration:**

The Startup class often includes a constructor where you can access configuration settings from various sources such as appsettings.json, environment variables, or command-line arguments.

Configuration settings can be used throughout the application for various purposes.

**Environment-specific Configuration:**

You can use the IHostingEnvironment parameter in the Configure and ConfigureServices methods to conditionally apply configurations based on the environment (Development, Staging, Production).

* **Type casting**

Type casting is the process of converting one type of data into another type. In C#, there are two types of casting: implicit casting and explicit casting.

**Implicit Casting**

Implicit casting, also known as widening or automatic casting, occurs when the destination data type can hold a larger range of values than the source data type. This conversion is done automatically by the compiler without any additional syntax.

**Explicit Casting**

Explicit casting, also known as narrowing or manual casting, occurs when the destination data type cannot hold as large a range of values as the source data type. In this case, a cast operator (datatype) must be used to perform the conversion.

* **Model Binding**

Model binding is a process in ASP.NET MVC where data from various sources, such as HTTP requests, HTML form fields, query strings, and route values, is automatically mapped to properties of a model class. This allows you to work with strongly-typed objects in your controller actions rather than dealing directly with raw request data.

* **Filters in MVC**

In ASP.NET MVC, filters are attributes that you can apply to controllers or actions to perform additional processing before or after the action method is executed. Filters allow you to modularize cross-cutting concerns such as logging, authorization, caching, and exception handling. There are several types of filters available in ASP.NET MVC:

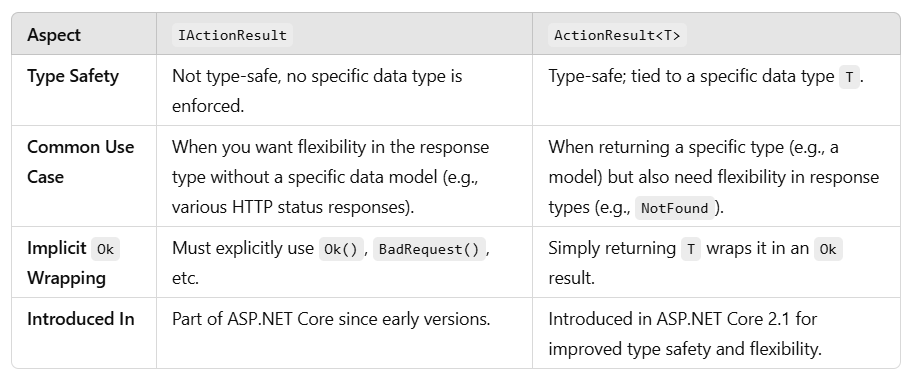
**Types of Filters**

**Authorization Filters**: These filters determine whether a user is authorized to access a particular resource. Examples include [Authorize], [AllowAnonymous], and custom authorization filters.

**Action Filters:** Action filters are executed before and after an action method is called. They can be used for tasks like logging, caching, and modifying the action result. Examples include [HttpPost], [HttpGet], [ValidateAntiForgeryToken], and custom action filters.

**Result Filters:** These filters are executed before and after the execution of the action result. They are used to modify the action result or perform additional processing. Examples include [OutputCache] and custom result filters.

**Exception Filters:** Exception filters handle exceptions that occur during the execution of an action method. They are used to log exceptions, perform error handling, and return custom error responses. Examples include [HandleError] and custom exception filters.

**Resource Filters:** Resource filters are executed at the beginning and end of the request pipeline. They can be used for tasks like logging, authentication, and request/response modification. Examples include global filters and custom resource filters.  
  


 **Use IActionResult** when you’re returning a variety of response types without a specific data model, such as in endpoints that only return status messages.

 **Use ActionResult<T>** when you’re returning a specific data type (like Person) and want the flexibility of also returning different HTTP status codes. This approach is ideal for RESTful APIs where responses often include data objects along with appropriate HTTP status codes.

* **The HTTP methods PUT, POST, GET, and DELETE**

GET: Used for retrieving data.

POST: Used for creating new resources or performing operations.

PUT: Used for updating existing resources or creating them if they don’t exist.

DELETE: Used for deleting resources.

The PATCH HTTP verb is used to apply partial modifications to a resource. Unlike the PUT method, which replaces the entire resource, PATCH updates only the specified fields of a resource.

**ADO.NET & EF**

ADO.NET (ActiveX Data Objects for .NET) is a set of classes in the .NET Framework that allows developers to access and manipulate data stored in databases. It provides a consistent set of APIs for working with various data sources such as SQL Server, Oracle, MySQL, and others.

**DataSet class** - A DataSet is basically a container which gets the data from one or more tables from the database. It follows disconnected architecture.

**DataAdapter class** - A DataAdapter bridges the gap between the disconnected DataSet/ DataTable objects and the physical database.

**DataReader Class** - The DataReader allows you to read the data returned by a SELECT command.

It is read only.

Unlike dataset we cannot update the database via this.

It follows connected architecture.

* **What is Connected architecture and Disconnected architecture**

**Disconnected Architecture -** Disconnected architecture means, you don’t need to connect always to get data from the database.

Get data into DataAdapter.

Manipulate the DataAdapter

Resubmit the data.

It is fast and robust(data will not lose in case of any power failure).

**Connected Architecture -** Connected architecture means you are directly interacting with database, but it is less secure and not robust.

* **What are the Authentication techniques used to connect to SQL Server**

The main authentication techniques are:

**Windows Authentication:** This technique uses the current Windows user account to authenticate to SQL Server. It is the most secure and recommended technique for applications running on a Windows domain.

**SQL Server Authentication:** This technique uses a SQL Server user account and password to authenticate to SQL Server.

**Azure Active Directory Authentication:** This technique allows for authentication using Azure Active Directory (AAD) identities.

* **What is ORM? What are the different types of ORM**

ORM (Object-Relational Mapper) is for mapping objects in your application with database tables.

It is like a wrapper to make database calls simple and easy.

* **What is Entity Framework?**

An Entity Framework (EF) is an open-source **ORM** (Object-Relational Mapper) from Microsoft.

It’s like a **wrapper** on ADO.NET.

Entity Framework minimizes the coding effort.

Entity Framework allows developers to work with database data as if they are working with regular .NET objects.

* **Diff between Entity Framework and ADO.NET**

**Entity Framework**

In EF, code is simpler and short.

EF can generate SQL statements automatically based on LINQ queries.

EF is designed to be database independent, which means same code can work with different databases.

EF is slightly slower than ADO.NET because internally EF use ADO.NET only

**ADO.NET**

In ADO.NET, same code is bigger.

ADO.NET requires developers to write SQL statements themselves.

ADO.NET, is tightly coupled to SQL Server and requires different code for different database systems.

ADO.NET is slightly faster than EF.

* LINEQ quries

To find the records that are common in both list1 and list2 using LINQ, you can use the Intersect method, which returns the common elements from both lists. Here's an example:

var commonRecords = list1.Intersect(list2).ToList();

To find names from a list based on their first character using LINQ, you can use the Where method to filter the names that start with a specified character.

char searchChar = 'A';

var filteredNames = list1.Where(r => !string.IsNullOrEmpty(r.Name) && r.Name.StartsWith(searchChar))

.Select(r => r.Name).ToList();

* **What is meant by DBContext and DBSet?**

DbContext is a class in the Entity Framework that helps in creating the **communication** between the database and the domain/entity class.

The DbSet class represents an **entity set** that can be used for create, read, update, and delete operations.

* **What are the different types of application development approaches used with EF?**

**Database First** – In Database First approach first the database is created and then the entity model is generated from it.

**Code First -** The Code First approach involves creating the data model using C# or VB.NET classes. Developers can define entities, relationships, and other schema elements using code, and then generate the database schema from the code.

**Model First -** The Model First approach involves creating the data model using visual tools such as Entity Data Model Designer in Visual Studio. Developers can create entities, relationships, and other schema elements visually, and then generate the database schema from the model.

* **Diff http code**

1xx: Informational

Informational responses indicate that the server has received the request and is continuing the process.

100 Continue: The server has received the request headers, and the client should proceed to send the request body.

101 Switching Protocols: The server is switching protocols as requested by the client (e.g., switching from HTTP/1.1 to HTTP/2).

2xx: Success

Successful responses indicate that the request was successfully received, understood, and accepted.

200 OK: The request was successful, and the server returned the requested resource.

201 Created: The request was successful, and a new resource was created as a result.

202 Accepted: The request has been accepted for processing, but the processing is not complete.

204 No Content: The server successfully processed the request, but there is no content to return.

3xx: Redirection

Redirection messages indicate that further action is needed to complete the request.

301 Moved Permanently: The requested resource has been permanently moved to a new URL.

302 Found: The requested resource has been temporarily moved to a different URL.

303 See Other: The response to the request can be found at another URL, using a GET method.

304 Not Modified: The resource has not been modified since the last request, so the client can use the cached version.

4xx: Client Errors

Client error responses indicate that there was an error with the request made by the client.

400 Bad Request: The server could not understand the request due to invalid syntax.

401 Unauthorized: The client must authenticate itself to get the requested response.

403 Forbidden: The client does not have access rights to the content.

404 Not Found: The server could not find the requested resource.

405 Method Not Allowed: The request method is known by the server but is not supported by the target resource.

409 Conflict: The request could not be processed because of a conflict in the request.

410 Gone: The resource requested is no longer available and will not be available again.

5xx: Server Errors

Server error responses indicate that the server failed to fulfill a valid request.

500 Internal Server Error: The server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented: The server does not support the functionality required to fulfill the request.

502 Bad Gateway: The server, while acting as a gateway or proxy, received an invalid response from the upstream server.

503 Service Unavailable: The server is not ready to handle the request, often due to maintenance or overload.

504 Gateway Timeout: The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server.

**Web API – Basics**

* **What is Web API? What is the purpose of Web API**

Web API(Application Programming Interface) are HTTP based services that can be accessed from browser or mobile-apps.

Web API enables different software applications to communicate with each other through the internet.

Web API’s mostly use JSON and XML formats to transfer data between different applications.

* **What is the difference Rest API and Web API?**

**REST API (Representational State Transfer API):**

REST is a style for creating APIs (ways for software to talk to each other) on the web.

REST APIs follow specific rules, like using standard web protocols (like HTTP) and having URLs for each piece of data.

They mainly use standard actions like GET (to fetch data), POST (to create data), PUT (to update data), and DELETE (to remove data).

Data is often sent and received in a format like JSON or XML.

REST APIs focus on organizing data as resources and accessing them through URLs.

**Web API:**

Web API is a general term for any API that can be accessed over the web.

It could be RESTful, meaning it follows the rules of REST.

But it could also be based on other styles, like SOAP (a different protocol for communication) or have its own way of working.

Web APIs can use various data formats for communication, including JSON, XML, and others.

They allow different software systems to communicate and share data over the internet.

--253 Page no. **Web API – Authentication & JWT**

**90. What are the types of authentication techniques in web api?**

Authentication is the process of verifying the identity of a user by validating their credentials such as username and password.

**91. The main difference between IIS and Kestrel**

The main difference between IIS and Kestrel is that Kestrel is a cross-platform server. It runs on Linux, Windows, and Mac, whereas IIS is Windows-specific. Another essential difference between the two is that Kestrel is fully open-source, whereas IIS is closed-source and developed and maintained only by Microsoft.

ASP.NET Core

Advantages

Fast and open source

Cross platform

Build in dependency injection

Easy updates

Cloud friendly

Performance

