

- **SOLID:**

- 1- What is the difference between design principles and design patterns?

Design principles and design patterns are both essential concepts in the field of software design, but they serve different purposes:

- 1- Design Principles:

- i. Design principles are high-level guidelines or rules that dictate how software should be designed to achieve certain qualities or characteristics, such as maintainability, scalability, flexibility, and reusability.
 - ii. They provide a foundation for making design decisions and help designers create systems that are easier to understand, maintain, and extend.
 - iii. Examples of design principles include :
 - 1. SOLID principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion).
 - 2. DRY (Don't Repeat Yourself).
 - 3. KISS (Keep It Simple, Stupid).
 - 4. YAGNI (You Aren't Gonna Need It).

- 2- Design Patterns:

- iv. Design patterns are reusable solutions to common problems that occur during software development.
 - v. They capture best practices and proven solutions to recurring design problems, providing a template for solving these problems in various contexts.
 - vi. Design patterns are more specific and detailed than design principles, focusing on implementation details and interactions between classes and objects.
 - vii. Examples of design patterns include :
 - 1. Singleton pattern,
 - 2. Factory pattern,
 - 3. Observer pattern,
 - 4. Strategy pattern, and many others.

- 2- Explain coupling, cohesion, abstract, and concrete concepts.

- Coupling:

- i. Coupling refers to the degree of interdependence between software modules or components.
 - ii. Low coupling means that modules are relatively independent of each other, and changes in one module have minimal impact on other modules.
 - iii. High coupling indicates strong interdependence between modules, where changes in one module often require changes in other modules.
 - iv. Low coupling is desirable because it leads to systems that are easier to understand, maintain, and extend.

- Cohesion:

- i. Cohesion measures how closely related and focused the responsibilities of elements within a module are.
 - ii. High cohesion means that elements within a module are strongly related and focused on a single task or responsibility.
 - iii. Low cohesion occurs when elements within a module have weak relationships or are responsible for multiple unrelated tasks.
 - iv. High cohesion is desirable because it leads to modules that are more understandable, maintainable, and reusable.

- Abstract:
 - i. In software engineering, abstraction is the process of simplifying complex systems by hiding unnecessary details and emphasizing essential characteristics. An abstract concept is one that emphasizes what something is, or what it does, rather than how it does it. Abstract classes and interfaces are examples of abstract concepts in object-oriented programming. An abstract class is a class that cannot be instantiated on its own and often contains one or more abstract methods that must be implemented by its subclasses. An interface is a contract that defines a set of methods that a class must implement. Abstract concepts help in designing flexible, modular, and maintainable software systems by promoting code reuse and encapsulation.
- Concrete:
 - i. In contrast to abstract concepts, concrete concepts are tangible and specific. Concrete classes are classes that can be instantiated and directly used to create objects. They provide concrete implementations of functionality defined in abstract classes or interfaces. Concrete classes often inherit from abstract classes or implement interfaces, providing the necessary implementations for the abstract methods or contractually defined methods. Concrete concepts represent actual implementations or instances of abstract concepts and are essential for building functioning software systems.

3- What is the difference between high-level modules and low-level modules?

- High-level modules:
 - i. High-level modules are typically larger, more abstract components of a software system.
 - ii. They encapsulate complex functionality or entire subsystems and provide a high-level view of the system's architecture.
 - iii. High-level modules are often designed to interact with other high-level modules, coordinating their activities to achieve system-level goals.
 - iv. Examples of high-level modules include components like user interfaces, business logic layers, or entire subsystems responsible for specific functional areas of an application.
 - v. High-level modules are usually less concerned with implementation details and more focused on defining the overall structure and behavior of the system.
- Low-level modules:
 - i. Low-level modules are smaller, more detailed components that implement specific functionalities or perform individual tasks.
 - ii. They typically deal with finer-grained operations and handle specific details of system implementation.
 - iii. Low-level modules often have well-defined interfaces and are designed to be reusable across different parts of the system.
 - iv. Examples of low-level modules include utility functions, database access layers, input/output handling modules, or data structures and algorithms.
 - v. Low-level modules are more concerned with implementation details and efficiency, focusing on optimizing performance and resource usage at a lower level of abstraction.

4- What is the difference between an abstract class and an interface in C#?

- Abstract Class:
 - i. An abstract class is a class that cannot be instantiated on its own; it serves as a blueprint for other classes to inherit from.
 - ii. Abstract classes can contain both abstract members (methods, properties, events) and concrete members (methods, properties, fields).
 - iii. Abstract members are declared without implementation and must be implemented by non-abstract derived classes.
 - iv. Abstract classes can also include constructors, fields, properties, and methods with implementations.
 - v. A class can inherit from only one abstract class, but it can implement multiple interfaces.
 - vi. Abstract classes are useful for defining a common base implementation for related classes while allowing for variations in behavior among derived classes.

Example:

```
csharp Copy code

public abstract class Shape
{
    public abstract double Area(); // Abstract method
    public virtual void Display() // Concrete method with implementation
    {
        Console.WriteLine("Displaying shape.");
    }
}
```

- Interface:
 - i. An interface is a contract that defines a set of members (methods, properties, events) that implementing classes must provide.
 - ii. Interfaces cannot contain any implementation details; they only define the signatures of members.
 - iii. Classes can implement multiple interfaces, allowing for a form of multiple inheritance.
 - iv. Interfaces are useful for defining a common set of behaviors that can be implemented by unrelated classes.
 - v. Interfaces are also used for achieving polymorphism and enabling loosely coupled designs through dependency injection.

- Example:

```
csharp Copy code

public interface IResizable
{
    void Resize(double factor); // Method signature without implementation
}
```

- **C#:**

1- Explain the differences between 'Ref parameters' and 'out parameters' in C#.

- ref parameters:

- i. ref parameters are used to pass arguments to a method by reference. This means that any changes made to the parameter inside the method will affect the original variable passed by the caller.
- ii. When using ref parameters, the variable passed as an argument must be initialized before it is passed to the method.
- iii. In the method signature, the ref keyword is used both in the method parameter declaration and in the method call.
- iv. ref parameters can be both read from and written to inside the method.
- v. ref parameters are typically used when the method needs to modify the value of the parameter and have that change reflected back to the caller.
- vi. Example

```
cssharp Copy code  
  
void Increment(ref int x)  
{  
    x++;  
}  
  
int value = 10;  
Increment(ref value);  
Console.WriteLine(value); // Output will be 11
```

- out parameters:

- i. out parameters are similar to ref parameters in that they allow passing arguments by reference. However, out parameters are specifically designed for output values from a method, where the method guarantees to assign a value to the parameter.
- ii. Unlike ref parameters, out parameters do not require the variable to be initialized before it is passed to the method. The method is responsible for initializing the out parameter before returning.
- iii. In the method signature, the out keyword is used in the method parameter declaration but is not used in the method call.
- iv. out parameters must be assigned a value by the method before it returns. If the method fails to assign a value to the out parameter, a compilation error occurs.
- v. out parameters are commonly used when a method needs to return multiple values or when the method needs to indicate success or failure through its return value and provide additional information through an out parameter.
- vi. Example

```
cssharp Copy code  
  
void Divide(int dividend, int divisor, out int quotient)  
{  
    quotient = dividend / divisor;  
}  
  
int result;  
Divide(10, 2, out result);  
Console.WriteLine(result); // Output will be 5
```

2- What are the differences between IEnumerable and IQueryable?

- IEnumerable and IQueryable are both interfaces in C# that represent collections of data, but they serve different purposes and have different capabilities:
- IEnumerable:
- IEnumerable is the most fundamental interface for querying data in C#. It represents a forward-only, read-only collection of elements.
- It is part of the System.Collections namespace and is implemented by most collection types such as arrays, lists, and dictionaries.
- IEnumerable is suitable for querying in-memory collections where all data is already loaded into memory.
- When you use LINQ queries with IEnumerable, the entire dataset is retrieved from the data source into memory before filtering, sorting, or projecting operations are applied.
- It is typically used for querying data in memory, such as collections or arrays.

3- What is the difference between == operator and Equals() method in C#?

- In C#, both the == operator and the Equals() method are used to compare objects for equality, but they operate differently based on the type of comparison being performed:
 - i. == Operator:
 1. The == operator is used for comparing value equality for built-in types and reference equality for reference types.
 2. For built-in types (e.g., int, double, string), the == operator compares the values of the operands. If the values are equal, it returns true; otherwise, it returns false.
 3. For reference types (e.g., classes), the default behavior of the == operator is to compare references rather than the contents of the objects. It checks if the references point to the same memory location. If the references are the same, it returns true; otherwise, it returns false.
 4. The behavior of the == operator for reference types can be overridden by implementing the == operator or by providing a custom implementation of the Equals() method.³
 - ii. Equals() Method:
 1. The Equals() method is a virtual method defined in the System.Object class, and it is overridden in many other classes to provide custom equality comparison logic.
 2. By default, the Equals() method compares reference equality for reference types, similar to the behavior of the == operator.
 3. However, classes can override the Equals() method to provide custom value equality comparison logic. This allows classes to define their own notion of equality based on their internal state.
 4. When overriding the Equals() method, it's recommended to also override the GetHashCode() method to maintain consistency with the behavior of hashing-based collections like Dictionary and HashSet.
 5. Unlike the == operator, the Equals() method can handle null values gracefully, and it should be used when comparing objects for equality in most cases.

4- What are events, delegates, and their types in C#?

- Delegates:
 - i. A delegate is a type that represents references to methods with a particular signature.
 - ii. Delegates allow methods to be passed as parameters or assigned to variables, enabling callbacks and event handling.
 - iii. Delegates provide a way to decouple the caller from the callee, allowing for more flexible and extensible code.
 - iv. There are two main types of delegates in C#:
 - v. Single Delegate: Represented by the delegate keyword, it can hold references to a single method.
 - vi. Multicast Delegate: Allows a delegate object to hold references to multiple methods, which are invoked sequentially when the delegate is called. Multicast delegates are represented by the delegate keyword with the += and -= operators for adding and removing method references.
- Events:
 - i. An event is a special type of member that enables a class to provide notifications to clients when certain actions occur.
 - ii. Events are based on delegates and provide a higher level of encapsulation and protection than raw delegate instances.
 - iii. Events define two methods: add to subscribe to the event and remove to unsubscribe from the event.
 - iv. Events are typically used to implement the observer design pattern, where an object (the subject) maintains a list of dependent objects (observers) and notifies them of changes in its state.
 - v. In C#, events are declared using the event keyword.
 - 1. Types of Delegates:
 - a. Action: Represents a method that takes zero or more input parameters but does not return a value.
 - b. Func: Represents a method that takes zero or more input parameters and returns a value.
 - c. Predicate: Represents a method that takes one input parameter and returns a Boolean value.
 - d. Custom Delegates: Developers can define their own custom delegates with specific method signatures tailored to their application needs.

5- What is boxing and unboxing in C#?

- Boxing:
 - i. Boxing is the process of converting a value type (such as int, double, or struct) to an object reference.
 - ii. When a value type is boxed, a new object is created on the heap to hold the value, and a reference to that object is returned.
 - iii. The value type's data is copied into the newly created object, and the object reference is used to access its members.
 - iv. Boxing allows value types to be treated as objects, enabling them to be used in contexts where reference types are expected, such as collections like ArrayList or method parameters that accept object.
- Unboxing:
 - i. Unboxing is the reverse process of boxing, where an object reference holding a boxed value type is converted back into the original value type.
 - ii. Unboxing extracts the value stored in the boxed object and assigns it to a variable of the appropriate value type.
 - iii. Unboxing requires an explicit cast, as the runtime type of the object reference may not match the type of the value being unboxed.
 - iv. If the unboxing operation fails due to an incompatible type, an InvalidCastException is thrown at runtime.

6- What are access modifiers in C#?

- 1- Public:
 - a. Members with the public access modifier are accessible from any other code in the same assembly or from external assemblies.
 - b. Public members can be freely accessed and used by any code that has visibility of the containing type.
- 2- Private:
 - a. Members with the private access modifier are accessible only within the containing type (class or struct).
 - b. Private members cannot be accessed from outside the containing type, including derived classes.
- 3- Protected:
 - a. Members with the protected access modifier are accessible within the containing type and by derived classes.
 - b. Protected members cannot be accessed from outside the containing type or its derived classes.
- 4- Internal:
 - a. Members with the internal access modifier are accessible within the same assembly but not from external assemblies.
 - b. Internal members can be accessed by any code within the same assembly, regardless of namespace.
- 5- protected internal:
 - a. Members with the protected internal access modifier are accessible within the same assembly and by derived classes, both within the assembly and from external assemblies.
 - b. This modifier combines the accessibility of both protected and internal, allowing for broader access than protected alone.
- 6- private protected (C# 7.2 and later):
 - a. Members with the private protected access modifier are accessible within the containing assembly by derived types, but only if they are in the same assembly.
 - b. This modifier is a stricter form of protected internal, allowing access only to derived types within the same assembly.

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

1- Semantics:

- a. Struct: Structs are value types. When you create a variable of a struct type, the variable directly contains the data, and assignments copy the entire data. Structs are typically used to represent lightweight objects that have value semantics, such as numeric types (int, float, etc.) and small immutable data structures.
- b. Class: Classes are reference types. When you create a variable of a class type, the variable contains a reference (memory address) to the actual data, and assignments copy the reference, not the data itself. Classes are used to represent more complex objects with behavior and state, and they support inheritance, polymorphism, and other object-oriented features.

2- Memory Allocation:

- a. Struct: Instances of structs are usually allocated on the stack or as part of other objects, such as arrays or other structs. They are generally more lightweight and have lower overhead compared to classes.
- b. Class: Instances of classes are allocated on the heap, and memory is managed by the garbage collector. This allows for dynamic memory allocation and more flexible object lifetimes but may incur additional overhead due to garbage collection.

3- Default Behavior:

- a. Struct: Structs have a default parameterless constructor that initializes all fields to their default values (zero-initialized for numeric types, null for reference types).
- b. Class: Classes have a default constructor if no constructor is explicitly defined. If a constructor is defined, the default constructor may or may not be generated by the compiler.

4- Inheritance and Polymorphism:

- a. Struct: Structs do not support inheritance. They cannot be derived from other types or serve as base types for other types. They also do not support polymorphism.
- b. Class: Classes support inheritance, allowing for the creation of hierarchies of related types. They can be used as base classes for derived classes, and they support polymorphism through method overriding and interface implementation.

5- Passing by Value vs. Passing by Reference:

- a. Struct: When passed as method arguments, structs are passed by value, meaning that a copy of the struct is passed to the method. This can be less efficient for large structs.
- b. Class: When passed as method arguments, classes are passed by reference, meaning that the method receives a reference to the original object. This allows for more efficient handling of large objects.

8- What are the differences between overloading and overriding in C#?

1- Overloading:

- a. Definition: Overloading refers to defining multiple methods in the same class with the same name but different parameter lists.
- b. Signature: Overloaded methods must have different parameter lists, which may differ in the number of parameters, types of parameters, or parameter modifiers (such as ref, out, or params).
- c. Return Type: Overloaded methods can have the same or different return types.
- d. Accessibility: Overloaded methods can have different access modifiers (such as public, private, protected, or internal).
- e. Static vs. Instance: Overloaded methods can include static methods, instance methods, or both.
- f. Compile-Time Resolution: Overloaded methods are resolved at compile time based on the number and types of arguments passed to them.

- **.Net Core:**

- 1- What is middleware in .Net Core?

In .NET Core, middleware refers to components that are used to handle HTTP requests and responses in ASP.NET Core applications. Middleware components are arranged in a pipeline, where each component in the pipeline processes an incoming HTTP request or outgoing HTTP response. Middleware provides a modular way to add cross-cutting concerns, such as logging, authentication, authorization, error handling, compression, and caching, to an application.

Here are some key points about middleware in .NET Core:

- **Pipeline-based Processing:** Middleware components are arranged in a sequence, forming a request processing pipeline. When an HTTP request is received, it passes through each middleware component in the pipeline, allowing each component to inspect, modify, or handle the request before passing it on to the next component. Similarly, the response passes through the pipeline in reverse order.
- **Flexible and Composable:** Middleware components are highly composable and can be added, removed, or reordered in the pipeline to customize the behavior of the application. This allows developers to easily add or remove functionality without modifying the core application logic.
- **Use Cases:** Middleware can be used for a wide range of purposes, including authentication (e.g., JWT authentication, cookie authentication), authorization (e.g., role-based access control), request/response logging, exception handling, CORS (Cross-Origin Resource Sharing) policies, response compression, static file serving, routing, and more.
- **Implementation:** Middleware in ASP.NET Core is implemented as classes that expose a method named `Invoke` or `InvokeAsync`. This method takes an `HttpContext` object representing the current HTTP request and response and performs some action based on the request context. Middleware components can be defined as inline functions, classes, or third-party libraries.
- **Middleware Ordering:** The order of middleware registration in the `Startup` class determines the order in which middleware components are executed in the pipeline. Middleware registered earlier in the pipeline is executed first, while middleware registered later is executed last.

2- What are the differences between action filter and resource filter middleware?

In ASP.NET Core, both action filters and resource filters are types of middleware that allow developers to add cross-cutting concerns to MVC controllers and their associated actions. However, they serve different purposes and operate at different stages of the request processing pipeline:

1- Action Filters:

- i. Purpose: Action filters are used to add pre- and post-processing logic to controller actions.
Execution Stage: Action filters are executed before and after the execution of controller actions.
- ii. Granularity: Action filters are associated with individual controller actions or applied globally to all actions within a controller or across all controllers.
- iii. Capabilities: Action filters can perform tasks such as logging, authentication, authorization, validation, caching, error handling, response modification, and more.
- iv. Interfaces: Action filters implement the `IActionFilter`, `IAsyncActionFilter`, `IResultFilter`, or `IAsyncResultFilter` interfaces, depending on the desired behavior (synchronous or asynchronous, before or after action execution).

2- Resource Filters:

- a. Purpose: Resource filters are used to add pre- and post-processing logic to the entire MVC pipeline, including routing and controller action selection.
- b. Execution Stage: Resource filters are executed before and after routing and controller action selection but before action filters.
- c. Granularity: Resource filters are applied globally to the entire MVC application and are not associated with specific controller actions.
- d. Capabilities: Resource filters are typically used for tasks such as logging, global authentication, global authorization, setting up shared context data, request/response transformation, and other application-wide concerns.
- e. Interfaces: Resource filters implement the `IResourceFilter` or `IAsyncResourceFilter` interfaces.

3- What are the differences between scoped, transient, and singleton services in .Net Core?

In ASP.NET Core, services are components that provide functionality to an application, such as data access, logging, configuration, and more. When registering services with the built-in dependency injection container, you can specify the service lifetime, which determines how long instances of the service should be kept and reused. The three main service lifetimes supported in ASP.NET Core are scoped, transient, and singleton. Here are the differences between them:

1- Transient Services:

- a. Lifetime: Transient services are created each time they are requested. A new instance is provided for each injection or service request.
- b. Usage: Transient services are suitable for lightweight, stateless components that are inexpensive to create and dispose of, such as repositories, data access objects, and utilities.
- c. Scope: Transient services are scoped to the lifetime of the service request or the current operation. Each request or operation gets its own instance of the transient service.

2- Scoped Services:

- a. Lifetime: Scoped services are created once per client request (HTTP request in web applications) and are reused within that request.
- b. Usage: Scoped services are typically used for components that need to maintain state or context for the duration of a request, such as database contexts, unit of work, or caches.
- c. Scope: Scoped services are scoped to the lifetime of the client request. Within the same request, multiple injections or service requests will receive the same instance of the scoped service.

3- Singleton Services:

- a. Lifetime: Singleton services are created once and reused throughout the lifetime of the application.
- b. Usage: Singleton services are suitable for components that are stateless or have shared state across the application, such as configuration settings, logging providers, or caching mechanisms.
- c. Scope: Singleton services are scoped to the lifetime of the application. Once created, the same instance is reused for all injection or service requests throughout the application's lifetime.

4- What is API architecture and its different types?

API architecture refers to the design principles, patterns, and practices used to build and structure application programming interfaces (APIs) in software development. API architecture determines how different components of an API interact with each other, how data is exchanged, and how clients consume the API. The choice of API architecture depends on factors such as scalability, performance, security, and ease of use. Here are some common types of API architectures:

d. REST (Representational State Transfer):

- Overview: REST is an architectural style for designing networked applications. It is based on a set of principles that emphasize stateless communication, resource-based URIs, standard HTTP methods (GET, POST, PUT, DELETE), and uniform interfaces.
- Characteristics: RESTful APIs use HTTP methods to perform CRUD (Create, Read, Update, Delete) operations on resources represented by URIs. They typically return data in JSON or XML format and rely on HTTP status codes for error handling.
- Advantages: RESTful APIs are widely adopted, easy to understand, and well-suited for web-based applications, mobile apps, and distributed systems. They promote scalability, reliability, and loose coupling between clients and servers.

e. GraphQL:

- Overview: GraphQL is a query language and runtime for APIs developed by Facebook. It enables clients to request only the data they need and allows for complex data fetching and manipulation in a single request.
- Characteristics: GraphQL APIs define a schema that describes the types of data available and the operations that can be performed. Clients send queries specifying the data they want, and the server responds with JSON payloads matching the structure of the query.
- Advantages: GraphQL provides flexibility, efficiency, and type safety. It allows clients to retrieve multiple resources in a single request, reducing over-fetching and under-fetching of data. It is particularly well-suited for complex and evolving data models.

f. SOAP (Simple Object Access Protocol):

- Overview: SOAP is a protocol for exchanging structured information in the implementation of web services. It defines a standard XML-based messaging format for communication between client and server.
- Characteristics: SOAP APIs use XML for message formatting and typically rely on the HTTP or HTTPS protocol for transport. They support features such as security (via WS-Security), reliability, and transactionality.
- Advantages: SOAP provides a standardized, platform-independent communication protocol suitable for enterprise-level applications and scenarios requiring strong security and transactional support.

g. gRPC (Google Remote Procedure Call):

- Overview: gRPC is an open-source RPC (Remote Procedure Call) framework developed by Google. It uses Protocol Buffers (protobuf) as the interface definition language (IDL) and HTTP/2 as the transport protocol.
- Characteristics: gRPC APIs define services and message types using protobuf, enabling efficient binary serialization and deserialization of data. It supports features such as bidirectional streaming, authentication, and load balancing.
- Advantages: gRPC provides high-performance, language-agnostic RPC communication suitable for microservices architectures, distributed systems, and real-time applications.

h. Event-Driven Architecture (EDA):

- Overview: Event-Driven Architecture is an approach where the flow of information is based on the occurrence of events. In this architecture, components communicate asynchronously through events, decoupling producers and consumers.
- Characteristics: EDA involves event producers emitting events when certain conditions are met, and event consumers reacting to those events by executing predefined actions. Events are typically delivered via message brokers or event streams.
- Advantages: EDA promotes loose coupling, scalability, and responsiveness. It enables systems to react to changes in real-time, handle asynchronous processing, and support event sourcing and event-driven integration.

5- What is WebSocket API architecture?

WebSocket API architecture refers to the design and implementation of APIs using the WebSocket protocol for real-time, bidirectional communication between clients and servers. WebSocket API architecture typically involves defining message formats, establishing WebSocket connections, handling events, and managing data exchange between clients and servers. Here's an overview of WebSocket API architecture:

i. WebSocket Protocol:

- WebSocket is a communication protocol that provides full-duplex, bi-directional communication channels over a single TCP connection.
- It enables real-time data exchange between clients and servers, allowing both parties to send and receive messages asynchronously without the overhead of traditional HTTP requests and responses.

j. API Design:

- Define the message format: WebSocket APIs often define message formats for exchanging data between clients and servers. This may include specifying message types, headers, and payload structures.
- Define API endpoints: WebSocket APIs can define endpoints for connecting to the WebSocket server, subscribing to channels or topics, sending messages, and handling events.

k. Server-Side Implementation:

- WebSocket server: Implement a WebSocket server to handle incoming WebSocket connections from clients. The server listens for WebSocket upgrade requests and establishes WebSocket connections with clients.
- Event handling: Implement event handlers on the server to process WebSocket connection events, such as open, close, error, and message events. These handlers manage the lifecycle of WebSocket connections and handle incoming messages from clients.
- Business logic: Implement business logic on the server to process incoming messages, perform operations, and generate responses. This may include authentication, authorization, data processing, and interaction with other services or databases.

l. Client-Side Implementation:

- WebSocket client: Implement a WebSocket client on the client-side to establish WebSocket connections with the server. The client connects to the WebSocket server, sends messages, and receives messages asynchronously.
- Event handling: Implement event handlers on the client to handle WebSocket connection events, such as open, close, error, and message events. These handlers manage the WebSocket connection lifecycle and process incoming messages from the server.

- Message handling: Implement logic to parse incoming messages from the server, extract data, and update the client's UI or perform other actions based on the received data.
- m. Security and Scalability:
 - Security: Implement security measures, such as authentication, encryption, and access control, to protect WebSocket connections and prevent unauthorized access to sensitive data.
 - Scalability: Design WebSocket APIs and server infrastructure to handle a large number of concurrent WebSocket connections and scale horizontally to accommodate growing user traffic.

6- How to use SignalR library to open WebSockets?

- Install-Package Microsoft.AspNetCore.SignalR
- Creating hub class
- Configure SignalR in Startup: `app.UseEndpoints(endpoints =>`

```
{
    endpoints.MapHub<ChatHub>("/chatHub");

    // Additional endpoint mappings can be added for other hubs
});
```
- Example to push data `"await Clients.All.SendAsync("ReceiveMessage", user, message);"`

7- What are API methods and their differences?

API methods, also known as API endpoints or operations, are functions or procedures exposed by an API that clients can invoke to perform specific tasks or operations. API methods define the interface through which clients interact with the API, specifying the actions that can be taken and the parameters required for those actions. Here are the common types of API methods and their differences:

1- GET Method:

- a. Purpose: The GET method is used to retrieve data from the server.
- b. Use Case: GET requests are typically used to read or retrieve resources from the server without modifying them.
- c. Idempotent: GET requests are considered idempotent, meaning that multiple identical requests should have the same effect as a single request. They should not have any side effects on the server.
- d. Example: Retrieving user information, fetching a list of products, or querying database records.

2- POST Method:

- a. Purpose: The POST method is used to submit data to the server to create new resources or perform actions that change the server's state.
- b. Use Case: POST requests are commonly used for creating new records, submitting form data, or executing operations that modify data on the server.
- c. Non-Idempotent: POST requests are non-idempotent, meaning that multiple identical requests may result in different outcomes if the server state is modified with each request.
- d. Example: Creating a new user account, submitting an order, or uploading a file.

3- PUT Method:

- a. Purpose: The PUT method is used to update or replace an existing resource on the server with the provided data.
- b. Use Case: PUT requests are used to update entire resources or replace them with a new representation.

- c. Idempotent: PUT requests are idempotent, meaning that multiple identical requests should have the same effect as a single request. They are intended for operations that can be safely repeated without unintended side effects.
 - d. Example: Updating user information, modifying a product's details, or replacing a document.
- 4- PATCH Method:
 - a. Purpose: The PATCH method is used to partially update an existing resource on the server with the provided data.
 - b. Use Case: PATCH requests are used when only a subset of the resource's attributes need to be updated, allowing clients to make incremental changes.
 - c. Non-Idempotent: PATCH requests are typically non-idempotent, as multiple identical requests may result in different outcomes if the server state is modified differently with each request.
 - d. Example: Changing a user's password, updating specific fields of a product, or modifying parts of a document.
- 5- DELETE Method:
 - a. Purpose: The DELETE method is used to remove an existing resource from the server.
 - b. Use Case: DELETE requests are used to delete resources or data permanently from the server.
 - c. Idempotent: DELETE requests are idempotent, meaning that multiple identical requests should have the same effect as a single request. They are intended for operations that can be safely repeated without unintended side effects.
 - d. Example: Deleting a user account, removing a product from inventory, or deleting a file.

8- What are media types used to send data in requests?

- [FromQuery]:
 - i. Purpose: Binds parameters from the query string of the URL.
 - ii. Usage: Typically used for simple data retrieval where parameters are passed in the URL.
- [FromBody]:
 - i. Purpose: Binds parameters from the request body of the HTTP request.
 - ii. Usage: Used for complex data types or when the data is sent in the request body, such as JSON or XML payloads.
- [FromForm]:
 - i. Purpose: Binds parameters from form data sent in a application/x-www-form-urlencoded or multipart/form-data request.
 - ii. Usage: Commonly used for form submissions or file uploads.
- Other [From] Attributes: *
 - i. [FromRoute]: Binds parameters from route data.
 - ii. [FromHeader]: Binds parameters from HTTP headers.
 - iii. [FromServices]: Binds parameters from the dependency injection container.
 - iv. [FromQuery(Name="customName")]: Allows specifying a custom query parameter name.
 - v. [FromBody(Name="customName")]: Allows specifying a custom JSON property name.

- 9- What is a background job and how can I use it? A background job, also known as a background task or asynchronous job, refers to a task or process that is executed independently of the main application thread or user interaction. Background jobs are typically used to perform time-consuming or resource-intensive operations, such as data processing, file manipulation, sending emails, or interacting with external services, without blocking the main application flow or user experience.

Here's how you can use background jobs in your application:

- **Background Job Frameworks/Libraries:**
 - i. Utilize existing background job frameworks or libraries available in your programming language or framework of choice. Examples include Hangfire for .NET applications, Celery for Python, Sidekiq for Ruby on Rails, and Bull for Node.js.
 - ii. These frameworks provide features such as job scheduling, execution management, retries, monitoring, and distributed processing, making it easier to implement and manage background tasks in your application.
- **Task Queues and Workers:**
 - i. Set up a task queue or message broker to manage the execution of background jobs. Clients enqueue jobs into the queue, and worker processes dequeue and execute them asynchronously.
 - ii. Popular task queue/message broker solutions include RabbitMQ, Redis with RQ (Redis Queue), Apache Kafka, Amazon SQS (Simple Queue Service), and Azure Service Bus.
- **Async/Await and Background Services:**
 - i. Use async/await programming patterns and background service components provided by your programming language or framework to execute background tasks asynchronously within the same application process.
 - ii. In ASP.NET Core, for example, you can create a background service by implementing the **BackgroundService** class and overriding the **ExecuteAsync** method to perform background processing.
- **Distributed Task Processing:**
 - i. For high-performance or scalable applications, consider distributing background job processing across multiple nodes or servers to handle increased workload and ensure fault tolerance.
 - ii. This can be achieved using distributed task processing frameworks like Hangfire Pro, Redis Cluster, or by deploying multiple instances of your application with load balancing.
- **Monitoring and Error Handling:**
 - i. Implement monitoring and error handling mechanisms to track the status of background jobs, handle failures, and retry failed jobs automatically.
 - ii. Use logging and monitoring tools to track job execution, detect errors, and troubleshoot issues in production environments.
- **Job Scheduling and Recurring Tasks:**
 - i. Schedule background jobs to run at specific times or intervals using built-in scheduling features provided by background job frameworks or task queues.
 - ii. Implement recurring tasks for periodic operations such as data backups, report generation, or system maintenance.

10- What is JWT?

JWT stands for JSON Web Token. It is an open standard (RFC 7519) for securely transmitting information between parties as a JSON object. JWTs are commonly used for authentication and authorization in web applications and APIs.

Here's a breakdown of JWT's components and how it works:

- 1- Structure: JWTs consist of three parts separated by dots (.): header, payload, and signature. The format looks like this: xxxxx.yyyyy.zzzzz.
- 2- Header: The header typically consists of two parts: the type of token (which is JWT) and the signing algorithm being used, such as HMAC SHA256 or RSA. This part of the token is JSON-encoded.
- 3- Payload: The payload contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: reserved, public, and private. Reserved claims are predefined and include information such as issuer, subject, expiration time, etc. Public claims are defined by the JWT specification and can be used by anyone. Private claims are custom claims created by the parties involved.
- 4- Signature: The signature is created by combining the encoded header, the encoded payload, and a secret key (if using HMAC) or a private key (if using RSA). This signature is used to verify that the message wasn't changed along the way and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.
- 5- How It Works: When a user logs in to a web application, the server creates a JWT containing some user information and signs it with a secret key. This JWT is then sent to the client and stored, usually in local storage or a cookie. On subsequent requests, the client includes the JWT in the request headers. The server verifies the JWT's signature to ensure its authenticity and extracts user information from the payload to authenticate and authorize the user.

JWTs are popular because they are compact, self-contained, and can be easily transmitted via URL parameters, headers, or cookies. They are also stateless, meaning the server doesn't need to keep track of session data. However, it's crucial to store JWTs securely, especially if they contain sensitive information, and to use strong encryption algorithms and secure key management practices to prevent unauthorized access or tampering.

11- What is the difference between different HTTP status codes?

HTTP status codes are standardized response codes that indicate the outcome of an HTTP request made by a client to a server. Each status code is a three-digit integer where the first digit defines the class of response, and the subsequent two digits define the specific status within that class.

Here's a breakdown of the different classes of HTTP status codes and their meanings:

- 1xx Informational:
 - i. These status codes indicate that the server has received the request and is processing it.
 - ii. Examples: 100 Continue, 101 Switching Protocols.
- 2xx Success:
 - i. These status codes indicate that the request was successfully received, understood, and accepted.
 - ii. Examples: 200 OK, 201 Created, 204 No Content.
- 3xx Redirection:
 - i. These status codes indicate that further action needs to be taken to complete the request.
 - ii. Examples: 301 Moved Permanently, 302 Found, 304 Not Modified.
- 4xx Client Error:
 - i. These status codes indicate that there was an error on the client's side, such as a malformed request or unauthorized access.
 - ii. Examples: 400 Bad Request, 401 Unauthorized, 404 Not Found.
- 5xx Server Error:
 - i. These status codes indicate that there was an error on the server's side, preventing it from fulfilling the request.
 - ii. Examples: 500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable.

Here are the main differences between different HTTP status codes:

- 2xx vs. 4xx/5xx:
 - i. 2xx status codes indicate successful requests where the server was able to process the request as expected.
 - ii. 4xx and 5xx status codes indicate errors, either on the client's side (4xx) or on the server's side (5xx), preventing the request from being successfully processed.
- 3xx Redirection vs. Other Classes:
 - i. 3xx status codes indicate that the client must take additional action to complete the request, such as following a redirect.
 - ii. Other status code classes (1xx, 2xx, 4xx, 5xx) indicate the outcome of the request or the presence of an error.
- Specificity of Status Codes:
 - i. Each status code within a class indicates a specific scenario or outcome of the request. For example, 200 OK indicates a successful request, 404 Not Found indicates that the requested resource was not found, and 500 Internal Server Error indicates an unexpected server error.

12- How to secure a .Net Web API?

Securing a .NET Web API involves implementing various security measures to protect the API from unauthorized access, data breaches, and other security threats. Here are some common practices for securing a .NET Web API:

- Authentication:
 - i. Implement authentication mechanisms to verify the identity of clients accessing the API. .NET provides built-in authentication options such as JWT (JSON Web Tokens), OAuth, and OpenID Connect.
 - ii. Use ASP.NET Core Identity for user authentication and management, including features like user registration, login, and password management.
- Authorization:
 - i. Use authorization policies and roles to control access to different parts of the API based on user roles and permissions.
 - ii. Implement role-based access control (RBAC) or claims-based authorization to restrict access to specific endpoints or resources.
 - iii. Consider using attribute-based authorization to decorate controller actions or methods with [Authorize] attributes to enforce authentication and authorization rules.
- HTTPS and TLS Encryption:
 - i. Always use HTTPS (HTTP Secure) to encrypt data transmitted between clients and the API server.
 - ii. Enable TLS (Transport Layer Security) encryption to secure communications and protect against eavesdropping, man-in-the-middle attacks, and data tampering.
- Input Validation and Sanitization:
 - i. Validate and sanitize input data received from clients to prevent injection attacks, such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
 - ii. Use input validation techniques such as model validation, data annotations, and input sanitization libraries to ensure that input data is safe and conforms to expected formats.
- Rate Limiting and Throttling:
 - i. Implement rate limiting and request throttling to mitigate the risk of abuse, denial-of-service (DoS) attacks, and brute force attacks.
 - ii. Use middleware or third-party libraries to enforce rate limits based on IP addresses, user accounts, or API keys to prevent excessive requests from overwhelming the API server.
- Security Headers:
 - i. Set security headers in HTTP responses to prevent common security vulnerabilities, such as cross-site scripting (XSS), clickjacking, and content sniffing.
 - ii. Use HTTP headers like Content-Security-Policy (CSP), X-Content-Type-Options, X-Frame-Options, and X-XSS-Protection to enhance security and protect against various attacks.
- Secure Coding Practices:
 - i. Follow secure coding practices to minimize security vulnerabilities in the API codebase, such as input validation, output encoding, parameterized queries, and secure session management.
 - ii. Regularly review and update dependencies, libraries, and frameworks used in the application to address security vulnerabilities and patch known vulnerabilities.
- Logging and Monitoring:
 - i. Implement logging and monitoring mechanisms to track and audit API activities, detect suspicious behavior, and respond to security incidents in real-time.
 - ii. Integrate logging frameworks like Serilog or NLog to log security-related events and anomalies, and use monitoring tools like Application Insights or ELK Stack for performance monitoring and security analytics.

13- What are the differences between XML and JSON?

XML (eXtensible Markup Language) and JSON (JavaScript Object Notation) are both popular data interchange formats used for structuring and transmitting data between systems. While they share some similarities, they have distinct differences in terms of syntax, usage, and features. Here's a comparison of XML and JSON:

- 1- Syntax:
 - a. XML: Uses tags enclosed in angle brackets (<tag></tag>) to define elements and hierarchical structure. Attributes can be included within tags (<tag attribute="value"></tag>).
 - b. JSON: Uses key-value pairs separated by colons ("key": "value") to represent data. Data is structured as objects (surrounded by curly braces {}) or arrays (surrounded by square brackets []).
- 2- Readability:
 - a. XML: More verbose and less human-readable due to the use of tags and attributes. Requires closing tags for elements.
 - b. JSON: More concise and easier to read and write for humans due to its lightweight syntax. Does not require closing elements like XML.
- 3- Data Types:
 - a. XML: Supports various data types, including text, numbers, dates, and custom-defined data types using XML Schema.
 - b. JSON: Supports primitive data types such as strings, numbers, booleans, arrays, and objects. Does not support custom data types out of the box.
- 4- Extensibility:
 - a. XML: Highly extensible and allows for the creation of custom tags and attributes. Can be validated and defined using XML Schema.
 - b. JSON: Less extensible than XML. Does not support namespaces or schema validation out of the box, but can be extended with additional specifications like JSON Schema.
 - c. Parsing and Processing:
 - d. XML: Requires an XML parser to parse and process XML documents. XML parsers are available in many programming languages and platforms.
 - e. JSON: Easily parsed and processed using built-in JSON parsers available in most programming languages. JSON's lightweight syntax makes it faster to parse and manipulate compared to XML.
- 5- Usage:
 - a. XML: Traditionally used in web services (SOAP), configuration files (e.g., XML configuration files in .NET applications), data interchange, and document markup.
 - b. JSON: Widely used in modern web development, particularly in RESTful APIs, AJAX requests, configuration files (e.g., package.json in Node.js), and storing and exchanging data between web applications.
- 6- Encoding:
 - a. XML: Uses UTF-8 or other character encodings to represent text data. Allows specifying the encoding in the XML declaration (<?xml version="1.0" encoding="UTF-8"?>).
 - b. JSON: Uses Unicode (UTF-8) encoding by default for representing text data. Supports ASCII and UTF-16 as well.

14- What are the differences between .Net and Mono?

.NET and Mono are both open-source software frameworks used for developing and running cross-platform applications, but they have some key differences in terms of origin, compatibility, and ecosystem. Here's a comparison:

Origin:

.NET: Developed by Microsoft, .NET is a software framework primarily targeted at Windows operating systems. It provides a comprehensive platform for building and running applications on Windows, including desktop, web, mobile, and cloud-based applications.

Mono: Mono is an open-source implementation of the .NET framework, originally developed by Xamarin (acquired by Microsoft in 2016). It was created to enable .NET applications to run on non-Windows platforms, such as Linux, macOS, and other Unix-like operating systems.

Compatibility:

.NET: The official .NET framework from Microsoft is primarily designed for Windows environments. It includes the Common Language Runtime (CLR), class libraries, and development tools for building and running applications on Windows.

Mono: Mono aims to provide cross-platform compatibility for .NET applications. It implements the CLR, class libraries, and development tools compatible with the .NET framework, allowing .NET applications to run on various operating systems, including Linux, macOS, and Windows.

Ecosystem:

.NET: .NET has a rich ecosystem of development tools, libraries, frameworks, and services provided by Microsoft and the .NET community. It includes tools like Visual Studio, Visual Studio Code, and Azure services for building, testing, and deploying .NET applications.

Mono: Mono extends the .NET ecosystem to non-Windows platforms, providing developers with the ability to build and run .NET applications on Linux, macOS, and other Unix-like systems. It also supports popular development tools like Visual Studio Code and JetBrains Rider for cross-platform development.

Performance:

.NET: .NET applications running on Windows typically benefit from optimized performance and integration with the Windows operating system and hardware. Microsoft continuously improves the performance of the .NET framework through updates and optimizations.

Mono: Mono strives to achieve performance parity with the official .NET framework on supported platforms. While performance may vary depending on the specific platform and workload, Mono aims to provide acceptable performance for .NET applications running on non-Windows environments.

Community and Support:

.NET: .NET benefits from the extensive support and resources provided by Microsoft, including documentation, tutorials, forums, and official support channels. It also has a large and active community of developers contributing to the ecosystem.

Mono: Mono is supported by an active community of contributors, developers, and users. It has its own documentation, forums, and community resources, as well as contributions from the broader .NET community.

15- What are the differences between SDK and runtime in .NET Core?

In .NET Core, the terms "SDK" (Software Development Kit) and "runtime" refer to different components of the .NET Core platform. Here's a breakdown of the differences between the two:

1- SDK (Software Development Kit):

- a. The .NET Core SDK includes everything developers need to build, debug, and publish .NET Core applications.
- b. It includes the .NET Core runtime, libraries, command-line tools (such as dotnet CLI), and other utilities necessary for developing applications.
- c. The SDK provides tools for creating new projects, managing dependencies, compiling code, running tests, and packaging applications for deployment.
- d. Developers typically install the .NET Core SDK on their development machines to create and maintain .NET Core applications.

2- Runtime:

- a. The .NET Core runtime, also known as the "runtime environment," is a subset of the .NET Core SDK that is required to run .NET Core applications.
- b. It includes the Common Language Runtime (CLR), Just-In-Time (JIT) compiler, Base Class Library (BCL), and other components necessary for executing .NET Core applications.
- c. The runtime is responsible for loading and executing managed code, providing memory management, exception handling, and other runtime services.
- d. Users who want to run .NET Core applications on their systems (e.g., in production environments or on end-user machines) need to install the .NET Core runtime, which provides the necessary infrastructure to execute the applications.

- **Entity Framework and LINQ:**

- 1- What is Entity Framework?

Entity Framework (EF) is an open-source object-relational mapping (ORM) framework for .NET applications, developed by Microsoft. It enables developers to work with relational databases using strongly-typed .NET objects, eliminating the need for manual SQL queries and database interactions.

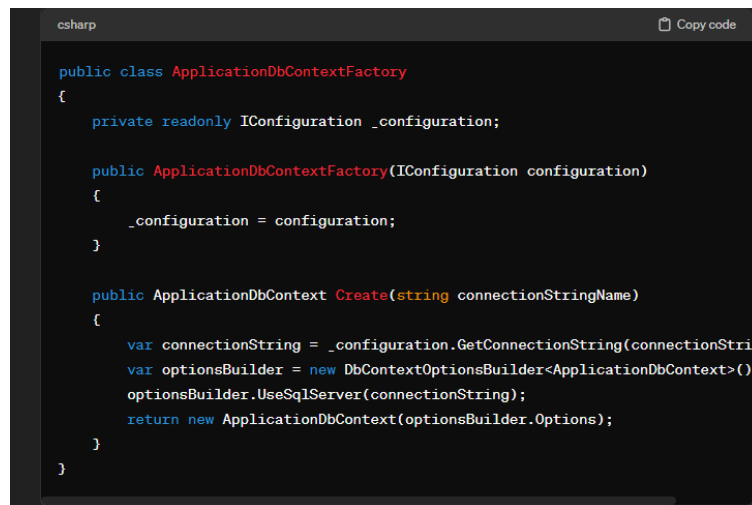
Here are the key components and features of Entity Framework:

- i. Object-Relational Mapping (ORM): Entity Framework provides a mapping between database tables and .NET objects, allowing developers to interact with databases using familiar object-oriented programming concepts such as classes, properties, and relationships.
- ii. Entity Data Model (EDM): EDM is a conceptual model that defines the structure of entities (classes) and their relationships in the application domain. It serves as a bridge between the database schema and the application's object model.
- iii. LINQ to Entities: Entity Framework supports Language Integrated Query (LINQ), allowing developers to write queries against the EDM using C# or VB.NET syntax. LINQ queries are translated into SQL queries and executed against the underlying database.
- iv. Code-First, Database-First, and Model-First Approaches: Entity Framework supports multiple development workflows, including Code-First (where the database schema is generated from the application's domain model), Database-First (where the EDM is generated from an existing database schema), and Model-First (where the EDM is designed using visual designers).
- v. Automatic Change Tracking: Entity Framework automatically tracks changes made to entities within the application, allowing developers to perform CRUD (Create, Read, Update, Delete) operations without writing explicit database update statements.
- vi. Lazy Loading and Eager Loading: Entity Framework supports lazy loading and eager loading mechanisms for loading related entities from the database. Lazy loading defers the loading of related entities until they are accessed, while eager loading loads related entities along with the main entity.
- vii. Transactions and Concurrency Control: Entity Framework provides support for transactions, allowing developers to group multiple database operations into a single transaction with rollback capabilities. It also offers concurrency control mechanisms for handling data consistency in multi-user environments.
- viii. Database Providers: Entity Framework supports multiple database providers, including Microsoft SQL Server, MySQL, PostgreSQL, SQLite, Oracle, and others, allowing developers to work with different database systems seamlessly.

2- How can I set the connection string in Entity Framework and change the database connection in each request?

In Entity Framework, you can set the connection string in various ways depending on your application's requirements and architecture. Additionally, changing the database connection for each request is not a common scenario due to the overhead and potential performance implications. However, if you still need to achieve this, you can use the following approaches:

- 1- Static Connection String:
 - a. Set the connection string statically in the configuration file (e.g., appsettings.json for ASP.NET Core or web.config for ASP.NET MVC) or in code during application startup.
 - b. Then, retrieve the connection string in your DbContext class constructor or DbContextOptions configuration method:
- 2- Dynamic Connection String:
 - a. If you need to change the database connection dynamically, you can dynamically generate or modify the connection string before instantiating the DbContext.
 - b. You can achieve this by creating a custom implementation of IDbContextFactory<TContext> or using a factory method to create DbContext instances with the desired connection string.
 - c. Here's an example of dynamically changing the connection string using a factory method:

A screenshot of a code editor showing C# code for a custom DbContext factory. The code is for a class named ApplicationDbContextFactory. It has a private readonly IConfiguration field named _configuration. The constructor takes an IConfiguration parameter and assigns it to _configuration. There is a public Create method that takes a connection string name as a parameter. Inside the Create method, it gets the connection string from the configuration, creates a new DbContextOptionsBuilder for ApplicationDbContext, uses the connection string with the UseSqlServer method, and returns a new ApplicationDbContext instance with the options builder's options.

```
cssharp Copy code

public class ApplicationDbContextFactory
{
    private readonly IConfiguration _configuration;

    public ApplicationDbContextFactory(IConfiguration configuration)
    {
        _configuration = configuration;
    }

    public ApplicationDbContext Create(string connectionStringName)
    {
        var connectionString = _configuration.GetConnectionString(connectionStringName);
        var optionsBuilder = new DbContextOptionsBuilder<ApplicationDbContext>()
        optionsBuilder.UseSqlServer(connectionString);
        return new ApplicationDbContext(optionsBuilder.Options);
    }
}
```

- d. Then, you can use this factory to create DbContext instances with different connection strings based on your requirements.
- 3- Request-Specific Connection:
 - a. Inject database name inside token payload
 - b. In "OnConfiguring" method in DbContext you can get the database name from the token.
 - c. In DbContextOptionsBuilder object you can set the connection string with UseSqlServer method you only pass the connection string as a parameter.

3- What is the Unit of Work design pattern?

The Unit of Work design pattern is a behavioral design pattern used in software development to manage transactions and the interactions between one or more repositories (data access objects) and the underlying data source (typically a database). It helps maintain data integrity, consistency, and atomicity by ensuring that multiple database operations are treated as a single unit of work that can be committed or rolled back as a whole.

- Key components of the Unit of Work pattern include:
 - i. **UnitOfWork Interface:** Defines the contract for managing transactions and coordinating multiple repository operations within a single unit of work. It typically includes methods such as `BeginTransaction()`, `Commit()`, `Rollback()`, and `SaveChanges()`.
 - ii. **UnitOfWork Implementation:** Provides the concrete implementation of the `UnitOfWork` interface, including the logic for starting, committing, and rolling back transactions, as well as coordinating repository operations.
 - iii. **Repository:** Represents a data access object responsible for interacting with a specific entity or aggregate root in the data source. Repositories encapsulate data access logic and provide CRUD (Create, Read, Update, Delete) operations for entities.

The main benefits of using the Unit of Work pattern include:

- **Transaction Management:** Ensures that multiple database operations are performed within a single transaction, allowing changes to be either committed or rolled back atomically to maintain data integrity and consistency.
- **Simplified Data Access:** Abstracts away the complexity of managing transactions and coordinating repository operations, providing a simpler and more consistent interface for interacting with the data source.
- **Improved Testability and Maintainability:** Facilitates unit testing and isolation of data access logic by decoupling it from the application's business logic. Each repository operation can be tested independently, and changes to data access logic can be made more easily without impacting other parts of the application.
- **Reduced Code Duplication:** Promotes code reuse by centralizing transaction management and data access logic within the `UnitOfWork` implementation, reducing code duplication across multiple repositories.

4- What is the difference between `AsNoTracking` and `Tracking` in Entity Framework?

In Entity Framework, `AsNoTracking()` and `tracking (Tracking)` are methods used to control how entities are tracked and managed by the `DbContext`. They have different behaviors and implications on performance, memory usage, and behavior of the `DbContext`. Here's a comparison:

1- `AsNoTracking()`:

- a. The `AsNoTracking()` method is used to instruct Entity Framework not to track changes to the queried entities. When you call `AsNoTracking()` on a query, Entity Framework retrieves entities from the database but does not add them to the `DbContext`'s change tracker.
- b. This means that changes made to entities retrieved using `AsNoTracking()` are not tracked by the `DbContext`, and they will not be persisted to the database when `SaveChanges()` is called. Additionally, entities retrieved using `AsNoTracking()` are not automatically updated if the underlying data changes in the database.
- c. `AsNoTracking()` is useful for read-only scenarios or when you don't need to track changes to entities, such as when you're querying data for display purposes or performing read-only operations.

2- `Tracking`:

- a. By default, Entity Framework tracks changes to entities retrieved from the database. This means that when you query entities using methods like `FirstOrDefault()`, `ToList()`, or `Find()`, Entity Framework adds the retrieved entities to the change tracker of the `DbContext`.
- b. `Tracking` allows Entity Framework to keep track of changes made to entities and automatically detect and apply those changes when `SaveChanges()` is called. It also enables features like change tracking, lazy loading, and automatic relationship fix-up.
- c. However, `tracking` can lead to increased memory usage and performance overhead, especially when querying large datasets or when you don't need change tracking capabilities.

5- What are lazy loading, eager loading, and explicit loading in Entity Framework?

Lazy loading, eager loading, and explicit loading are techniques used in Entity Framework to load related entities from the database. Each approach has its own advantages and trade-offs, and the choice depends on the specific requirements of the application. Here's an overview of each:

1- Lazy Loading:

- a. Lazy loading is a feature of Entity Framework that allows related entities to be automatically loaded from the database when they are accessed for the first time.
- b. With lazy loading enabled, Entity Framework delays the loading of related entities until they are accessed through navigation properties. This can help reduce the amount of data retrieved from the database initially and improve performance.
- c. Lazy loading is enabled by default in Entity Framework for navigation properties that are marked as virtual. However, it can lead to additional database queries being executed at runtime, which may cause performance overhead, especially in scenarios where multiple related entities are accessed.

2- Eager Loading:

- a. Eager loading is a technique used to load related entities from the database along with the main entity in a single query.
- b. With eager loading, you explicitly include related entities using the `Include()` method or `ThenInclude()` method if there are multiple levels of navigation properties.
- c. Eager loading can help reduce the number of database queries by fetching all required data in a single query, which can improve performance, especially in scenarios where you know in advance that related entities will be accessed.
- d. However, eager loading can result in retrieving more data than necessary, leading to increased memory usage and potential performance overhead, especially if there are deep object graphs or unnecessary related entities included.

3- Explicit Loading:

- a. Explicit loading is a technique used to load related entities from the database on demand, after the main entity has been retrieved.
- b. With explicit loading, you use methods such as `Load()` or `LoadAsync()` to explicitly load related entities for specific navigation properties.
- c. Explicit loading allows you to control when related entities are loaded from the database, providing more flexibility and control over the data retrieval process.
- d. Explicit loading is useful in scenarios where you want to defer the loading of related entities until they are actually needed, which can help improve performance and reduce memory usage by avoiding unnecessary data retrieval.

6- What are the differences between DataAnnotations and FluentAPI?

Data Annotations and Fluent API are two ways to configure the behavior of entities and their properties in Entity Framework. They provide different approaches for defining constraints, validation rules, and mappings between entities and the database schema. Here are the key differences between Data Annotations and Fluent API:

1- Data Annotations:

- a. Data Annotations are attributes applied directly to entity classes and properties using .NET attribute syntax.
- b. They are defined in the `System.ComponentModel.DataAnnotations` namespace and provide a declarative way to specify constraints and validation rules.
- c. Data Annotations are easy to use and understand, as they are applied directly to entity classes and properties within the code.

2- Fluent API:

- a. Fluent API is a programmatic way to configure entity properties, relationships, and other aspects of the model using method chaining syntax.
- b. It is defined by the `ModelBuilder` class in Entity Framework and allows for more complex and fine-grained configuration compared to Data Annotations.
- c. Fluent API provides greater flexibility and control over the model configuration, allowing you to define custom mappings, composite keys, indexes, and more.

3- Usage and Flexibility:

- a. Data Annotations are typically used for simple scenarios and straightforward mappings, such as specifying required fields, string lengths, or primary keys.
- b. Fluent API is more suitable for complex mappings and scenarios where fine-grained control over the model configuration is required, such as defining composite keys, configuring table mappings, or specifying custom data types.

4- Separation of Concerns:

- a. Data Annotations mix configuration concerns with entity classes, which can lead to cluttered code and tight coupling between data access and domain logic.
- b. Fluent API promotes a separation of concerns by allowing model configuration to be defined separately from entity classes, which can lead to cleaner and more maintainable code.

7- What is the difference between Single and First methods?

1- Single():

- a. The Single() method is used to retrieve a single entity from a sequence that matches the specified criteria. If the sequence contains more than one matching element, or if the sequence is empty, an exception is thrown.
- b. Use Single() when you expect the sequence to contain exactly one element that matches the specified criteria, and you want to ensure that only one element is returned. It's useful for scenarios where uniqueness is required, such as retrieving a record by its unique identifier.

2- First():

- a. The First() method is used to retrieve the first entity from a sequence that matches the specified criteria. If the sequence is empty, an exception is thrown.
- b. Use First() when you want to retrieve the first element that matches the specified criteria, without necessarily requiring uniqueness. It's useful for scenarios where you're interested in the first occurrence of a record that meets certain conditions.

3- Behavior on Empty Sequences:

- a. Single() throws an exception (InvalidOperationException) if the sequence is empty or if more than one element matches the specified criteria.
- b. First() throws an exception (InvalidOperationException) if the sequence is empty but does not check for uniqueness. It simply returns the first element that matches the criteria if it exists.

4- Performance Implications:

- a. Single() typically performs a more exhaustive search to ensure uniqueness, as it checks for both the presence of a single matching element and the absence of additional matching elements. This can result in slightly higher overhead compared to First().
- b. First() performs a simple search to retrieve the first matching element without checking for uniqueness beyond the first occurrence.

8- What are the differences between FirstOrDefault and First methods?

1- FirstOrDefault():

- a. The FirstOrDefault() method returns the first element from the sequence that matches the specified criteria, or a default value if the sequence is empty.
- b. If the sequence contains elements, FirstOrDefault() behaves the same as First() and returns the first matching element.
- c. If the sequence is empty, FirstOrDefault() returns the default value for the element type (null for reference types, 0 for numeric types, false for boolean types, and default(T) for other value types).

2- First():

- a. The First() method returns the first element from the sequence that matches the specified criteria.
- b. If the sequence is empty, First() throws an exception (InvalidOperationException).
- c. First() is useful when you expect the sequence to contain at least one matching element, and you want to retrieve the first occurrence of that element.

3- Behavior on Empty Sequences:

- a. FirstOrDefault() returns the default value for the element type if the sequence is empty. It does not throw an exception.
- b. First() throws an exception (InvalidOperationException) if the sequence is empty.

4- Performance Implications:

- a. FirstOrDefault() typically has slightly better performance than First() when the sequence is empty, as it does not perform any additional checks or throw exceptions in that case.
- b. First() may have slightly higher overhead when the sequence is empty, as it needs to throw an exception.

9- Explain one-to-one, one-to-many, and many-to-many relations.

In database design and Entity Framework, one-to-one, one-to-many, and many-to-many are types of relationships between entities (tables) in a relational database. These relationships define how data in one table is related to data in another table. Here's an explanation of each type of relationship:

1- One-to-One (1:1) Relationship:

- a. In a one-to-one relationship, each record in one table is associated with exactly one record in another table, and vice versa.
- b. For example, consider a scenario where each employee has exactly one corresponding employee profile. This relationship can be modeled as a one-to-one relationship, where each employee record in the Employees table is associated with exactly one record in the EmployeeProfiles table.
- c. In Entity Framework, a one-to-one relationship can be defined by specifying a navigation property on each entity that points to the other entity.

2- One-to-Many (1:N) Relationship:

- a. In a one-to-many relationship, each record in one table can be associated with one or more records in another table, but each record in the second table is associated with exactly one record in the first table.
- b. For example, consider a scenario where each department can have multiple employees, but each employee belongs to exactly one department. This relationship can be modeled as a one-to-many relationship, where each department record in the Departments table is associated with multiple employee records in the Employees table.
- c. In Entity Framework, a one-to-many relationship is typically represented by a navigation property on the "one" side (e.g., Department) and a collection navigation property on the "many" side (e.g., ICollection<Employee>).

3- Many-to-Many (N:M) Relationship:

- a. In a many-to-many relationship, each record in one table can be associated with one or more records in another table, and vice versa.
- b. For example, consider a scenario where each student can enroll in multiple courses, and each course can have multiple students enrolled. This relationship can be modeled as a many-to-many relationship, where multiple student records in the Students table are associated with multiple course records in the Courses table.
- c. In Entity Framework, a many-to-many relationship is typically represented by creating an intermediate join table (often called a "junction" or "link" table) that contains foreign key columns referencing the primary keys of the two entities involved in the relationship. Each record in the join table represents a pair of related entities.

10- What are indexes?

In the context of databases, an index is a data structure that improves the speed of data retrieval operations on a table by providing quick access to rows based on the values of one or more columns. Indexes are created on specific columns of a table to facilitate efficient querying, sorting, and joining of data. Here are key points about indexes:

- 1- Purpose:
 - a. Indexes are used to optimize the performance of SELECT queries, particularly those that involve filtering, sorting, or joining data based on specific columns.
 - b. By creating indexes on frequently queried columns, database systems can quickly locate the relevant rows without having to scan the entire table, resulting in faster query execution times.
- 2- Structure:
 - a. Internally, indexes are implemented as balanced trees, hash tables, or other data structures that allow for efficient lookup and retrieval of values.
 - b. Each index entry typically consists of the indexed column value(s) and a pointer to the corresponding row in the table.
- 3- Types of Indexes:
 - a. Single-Column Index: An index created on a single column of a table.
 - b. Composite Index (Multi-Column Index): An index created on multiple columns of a table. Composite indexes can improve query performance for queries that involve multiple columns.
 - c. Unique Index: An index that enforces uniqueness constraints on the indexed columns, preventing duplicate values.
 - d. Clustered Index: In some database systems like SQL Server, a clustered index determines the physical order of rows in the table. Each table can have only one clustered index.
 - e. Non-Clustered Index: An index where the order of rows in the index does not match the physical order of rows in the table. Tables can have multiple non-clustered indexes.
- 4- Creation and Maintenance:
 - a. Indexes are created using SQL statements like CREATE INDEX, specifying the columns to be indexed.
 - b. Adding indexes can speed up query performance but may impact insert, update, and delete operations, as the database system needs to maintain index structures whenever data is modified.
- 5- Considerations:
 - a. Indexes are most effective on columns that are frequently used in WHERE clauses, JOIN conditions, and ORDER BY clauses.
 - b. Over-indexing (creating indexes on every column) can degrade performance and increase storage overhead.
 - c. Indexes consume additional storage space and may impact the performance of data modification operations (inserts, updates, deletes).
 - d. Regular monitoring and tuning of indexes are necessary to ensure optimal database performance, as query patterns and data volumes may change over time.

11- Explain composite indexes.

Composite indexes, also known as multi-column indexes, are indexes that are created on multiple columns of a database table. Unlike single-column indexes, which are created on a single column, composite indexes include multiple columns in their index key. This allows for efficient querying and retrieval of data based on combinations of values from multiple columns.

Here are some key points about composite indexes:

1- Purpose:

- a. Composite indexes are used to optimize queries that involve filtering, sorting, or joining data based on multiple columns.
- b. By creating a composite index on a set of columns commonly used together in queries, database systems can quickly locate the relevant rows without having to perform full table scans or multiple index lookups.

2- Structure:

- a. Each entry in a composite index consists of values from the indexed columns concatenated together in a predefined order.
- b. The order of columns in the composite index key is significant and affects the index's ability to support different query patterns.
- c. Queries that match the leftmost prefix of the composite index key can benefit from index usage.

3- Query Optimization:

- a. Composite indexes can improve query performance for queries that involve conditions on multiple columns, such as WHERE clauses with conjunctions or disjunctions, and JOIN operations on multiple columns.
- b. For example, consider a composite index on columns (A, B). This index can efficiently support queries that filter or sort data based on both columns A and B, as well as queries that filter on column A alone.

4- Considerations:

- a. The cardinality and selectivity of columns included in a composite index should be considered when creating indexes. Columns with high selectivity (i.e., a large number of distinct values) are generally more suitable for indexing.
- b. Composite indexes consume additional storage space compared to single-column indexes, as they store index entries for combinations of values from multiple columns.
- c. Care should be taken not to over-index by creating too many composite indexes, as this can increase storage overhead and impact the performance of data modification operations.
- d. Regular monitoring and analysis of query execution plans are necessary to ensure that composite indexes are effectively utilized and to identify opportunities for index optimization.

12- What is a data seeder and how can I create one in my project?

A data seeder, also known as a database seeder or seed data initializer, is a component in a software project that populates the database with initial data. Seed data typically includes predefined records that are necessary for the application to function correctly or for testing purposes. Data seeders are commonly used to populate reference data, default settings, or sample data required for the application's functionality.

Here's how you can create a data seeder in your project:

- 1- Define Seed Data:
 - a. Identify the data that needs to be seeded into the database. This may include default user accounts, reference tables, configuration settings, or any other initial data required by your application.
- 2- Create Seeder Class:
 - a. Create a seeder class that encapsulates the logic for populating the database with seed data. This class should typically be placed in a designated folder or namespace within your project.
- 3- Implement Seeder Logic:
 - a. Implement the logic for seeding the database within the seeder class. This may involve creating new records using Entity Framework, executing SQL scripts, or any other method appropriate for your database technology.
 - b. Ensure that the seeder logic is idempotent, meaning that it can be safely executed multiple times without causing duplicate records or data corruption.
- 4- Invoke Seeder on Application Startup:
 - a. Configure your application to execute the data seeder during application startup. This ensures that the seed data is automatically populated whenever the application is deployed or initialized.
 - b. In ASP.NET Core, you can use the Startup class to configure services and middleware, including the invocation of data seeders. You can call the seeder logic from the Configure method or from a custom initialization method.
- 5- Testing and Maintenance:
 - a. Test the data seeder to ensure that it populates the database correctly with the desired seed data.
 - b. Regularly review and update the seed data as needed to reflect changes in the application requirements or database schema.

13- What is migration in Entity Framework?

In Entity Framework, migration refers to the process of managing changes to the database schema over time in a structured and organized manner. Migrations allow developers to update the database schema to reflect changes in the application's data model, such as adding new tables, modifying existing columns, or altering relationships between entities. Here's a breakdown of the key aspects of migrations in Entity Framework:

- 1- Schema Changes:
 - a. Migrations are used to apply schema changes to the database, including creating, modifying, or dropping database objects such as tables, columns, indexes, constraints, and relationships.
- 2- Code-First Approach:
 - a. Entity Framework Code First migrations allow developers to define the data model using classes and then generate the corresponding database schema based on those classes.
 - b. Developers can define entities, relationships, and data annotations or Fluent API configurations to specify the mapping between classes and database tables.
- 3- Automatic vs. Explicit Migrations:
 - a. Entity Framework supports both automatic and explicit migrations.
 - b. Automatic migrations automatically generate migration scripts based on changes detected in the data model. Developers can enable automatic migrations using the Enable-Migrations command in the Package Manager Console.
 - c. Explicit migrations require developers to manually create migration classes and write migration code to specify the changes to be applied to the database schema. Developers can add new migrations using the Add-Migration command.
- 4- Migration History:
 - a. Entity Framework maintains a migration history table in the database to track which migrations have been applied to the database and in what order.
 - b. Each migration class contains instructions for upgrading or downgrading the database schema from one version to another.
 - c. The migration history table ensures that migrations are applied in the correct sequence and prevents applying the same migration multiple times.
- 5- Applying Migrations:
 - a. Developers can apply migrations to the database using the Update-Database command in the Package Manager Console.
 - b. When applying migrations, Entity Framework compares the current state of the database to the migration history and applies any pending migrations in the correct order.
- 6- Rolling Back Migrations:
 - a. Developers can roll back migrations using the Update-Database command with the -TargetMigration option to specify a specific migration to roll back to.
 - b. Rolling back migrations reverses the changes made by the specified migration and updates the migration history accordingly.

14- How to add a new migration?

To add a new migration in Entity Framework Code First approach, you typically follow these steps:

- 1- Ensure Setup:
 - a. Make sure your project is configured with Entity Framework and migrations are enabled. If you haven't already enabled migrations, you can do so by running the following command in the Package Manager Console: "Enable-Migrations" command
- 2- Make Changes to Data Model:
 - a. Modify your data model by adding, removing, or changing entities, properties, or relationships in your code. These changes represent the desired changes to your database schema.
- 3- Create a Migration:
 - a. Once you've made the necessary changes to your data model, you can create a new migration to capture these changes. Run the following command in the Package Manager Console, replacing YourMigrationName with a descriptive name for your migration: "Add-Migration YourMigrationName" command.
 - b. This command will generate a new migration file with the specified name in your project's Migrations folder. The migration file contains code that specifies how to upgrade the database schema to reflect the changes you made to your data model.
- 4- Review and Customize Migration:
 - a. Open the generated migration file and review the code to ensure it accurately reflects the changes you made to your data model.
 - b. You can customize the migration code if needed, such as modifying column data types, adding indexes, or specifying default values.
- 5- Apply Migration:
 - a. Once you're satisfied with the migration code, you can apply the migration to your database by running the following command in the Package Manager Console: "Update-Database" command.
 - b. This command will execute the migration and apply the changes to your database schema. Entity Framework will automatically update the migration history table to record the applied migration.
- 6- Verify Changes:
 - a. After applying the migration, verify that the database schema reflects the changes you made to your data model. You can use tools like SQL Server Management Studio or database migration tools to inspect the database schema.

15- How to rollback a migration?

To rollback a migration in Entity Framework Code First approach, you can use the Update-Database command in the Package Manager Console with the -TargetMigration option to specify the migration to which you want to roll back. Here are the steps to rollback a migration:

- 1- Identify Target Migration:
 - a. Determine which migration you want to roll back to. This can be the previous migration, a specific migration, or the initial migration that created the database schema.
- 2- Open Package Manager Console:
 - a. Open Visual Studio and go to the Tools menu.
 - b. Select NuGet Package Manager, then Package Manager Console.
- 3- Run Update-Database Command:
 - a. In the Package Manager Console, run the Update-Database command with the -TargetMigration option followed by the name of the target migration.
 - b. If you want to roll back to the previous migration, you can use the special LastGoodMigration value as the target migration.
 - c. Here's the general syntax: "Update-Database -TargetMigration YourTargetMigration"
 - d. Replace YourTargetMigration with the name of the migration to which you want to roll back.
 - e. Apply the Command: Press Enter to execute the Update-Database command. Entity Framework will roll back the database schema changes to the specified migration.
 - f. Verify Changes:
 - g. After rolling back the migration, verify that the database schema reflects the changes made by the target migration. You can use tools like SQL Server Management Studio or database migration tools to inspect the database schema.

16- How can I work with database first in Entity Framework?

You can use scaffolding Command to do this :

- 1- Install Entity Framework Core Tools:
 - a. Ensure you have Entity Framework Core Tools installed. If not, you can install it using the following command: "Install-Package Microsoft.EntityFrameworkCore.Tools"
- 2- Run Scaffold-DbContext Command:
- 3- Use the Scaffold-DbContext command to scaffold entity types and a DbContext from an existing database. Here's the basic syntax: "Scaffold-DbContext "YourConnectionString" Microsoft.EntityFrameworkCore.SqlServer -OutputDir YourOutputDirectory"
 - a. Replace "YourConnectionString" with the connection string to your database.
 - b. Replace Microsoft.EntityFrameworkCore.SqlServer with the provider to use (e.g., for SQL Server, MySQL, SQLite).
 - c. Replace YourOutputDirectory with the directory where you want the generated files to be placed.

17- How to use Entity Framework to execute queries?

If you need to execute raw SQL queries using Entity Framework, you can do so using the `DbContext.Database` property, which provides methods for executing raw SQL commands. Here's how you can execute raw SQL queries with Entity Framework:

- 1- Create `DbContext`:
 - a. Ensure you have a `DbContext` class defined in your application. This class represents the database context and provides access to the database.
- 2- Instantiate `DbContext`:
 - a. Create an instance of your `DbContext` class. This instance will be used to interact with the database.
- 3- Execute Raw SQL Query:
 - a. Use the `DbContext.Database` property to access methods for executing raw SQL queries. The `ExecuteSqlCommand` method is typically used to execute non-query SQL commands (e.g., `INSERT`, `UPDATE`, `DELETE`), while the `SqlQuery` method is used to execute queries that return data.
 - b. You can pass parameter values to your SQL query using parameters to prevent SQL injection attacks.
 - c. Example

18- What are SQL injections and how to protect a system from them?

SQL injection is a type of attack where malicious SQL code is inserted into input fields of a web application, with the intent of manipulating the backend database. This can lead to unauthorized access to sensitive data, modification of data, or even deletion of data. SQL injection attacks are one of the most common types of security vulnerabilities in web applications.

Here are some techniques to protect a system from SQL injection attacks:

- 1- Use Parameterized Queries (Prepared Statements): Instead of constructing SQL queries by concatenating strings, use parameterized queries or prepared statements provided by your database framework. Parameterized queries separate SQL code from user input, preventing attackers from injecting malicious SQL code.
- 2- Input Validation: Validate and sanitize all user inputs before using them in SQL queries. Ensure that input data conforms to expected formats and ranges. Reject any input that contains suspicious characters or patterns, such as SQL keywords or special characters like quotes.
- 3- Least Privilege Principle: Limit the privileges of the database user account used by the application. Only grant the necessary permissions required for the application to function properly. Avoid using highly privileged accounts for routine application tasks.
- 4- Avoid Dynamic SQL: Minimize the use of dynamic SQL queries where SQL statements are constructed dynamically at runtime using string concatenation. If dynamic SQL is necessary, ensure that proper input validation and sanitization are applied.
- 5- Escape User Input: If parameterized queries are not feasible, ensure that user input is properly escaped before embedding it in SQL queries. Use appropriate escape functions provided by your database framework to prevent SQL injection.
- 6- Use Stored Procedures: Encapsulate SQL logic in stored procedures whenever possible. Stored procedures provide a layer of abstraction and can help mitigate SQL injection attacks by reducing the surface area of attack.
- 7- Implement Web Application Firewalls (WAF): Use WAFs to filter and monitor incoming HTTP traffic for suspicious patterns and known attack signatures. WAFs can help detect and block SQL injection attempts before they reach the application.
- 8- Regular Security Audits: Conduct regular security audits and code reviews to identify and remediate potential SQL injection vulnerabilities. Test the application using penetration testing techniques to simulate real-world attack scenarios.

19- What are query transactions and how to use them in Entity Framework?

In Entity Framework (EF), query transactions typically refer to the concept of wrapping multiple database operations within a single transaction to ensure atomicity, consistency, isolation, and durability (ACID properties). Transactions are essential for maintaining data integrity, especially when dealing with multiple database operations that need to be treated as a single unit of work.

Here's how you can use transactions in Entity Framework:

- 1- Implicit Transactions: By default, Entity Framework operates within the scope of a transaction for each individual operation (e.g., `SaveChanges`). When you call `SaveChanges`, Entity Framework automatically starts a transaction, executes the SQL commands for inserts, updates, and deletes, and then commits the transaction if successful. If an error occurs, Entity Framework rolls back the
- 2- Explicit Transactions: Sometimes, you may need to perform multiple database operations within a single transaction. In such cases, you can use explicit transactions to manage the transactional behavior manually.

- **SQL**

1- What is DBMS?

- A Database Management System (DBMS) is a program that controls creation, maintenance and use of a database. DBMS can be termed as File Manager that manages data in a database rather than saving it in file systems.

2- What is RDBMS?

- RDBMS stands for Relational Database Management System. RDBMS store the data into the collection of tables, which is related by common fields between the columns of the table. It also provides relational operators to manipulate the data stored into the tables.

3- What is SQL?

- SQL stands for Structured Query Language , and it is used to communicate with the Database. This is a standard language used to perform tasks such as retrieval, updation, insertion and deletion of data from a database. Standard SQL Commands are Select

4- What is a Database?

- Database is nothing but an organized form of data for easy access, storing, retrieval and managing of data. This is also known as structured form of data which can be accessed in many ways. Example: School Management Database, Bank Management Database.

5- What are tables and Fields?

- A table is a set of data that are organized in a model with Columns and Rows. Columns can be categorized as vertical, and Rows are horizontal. A table has specified number of column called fields but can have any number of rows which is called record.
Example:. Table: Employee. Field: Emp ID, Emp Name, Date of Birth. Data: 201456, David, 11/15/1960.

6- What is a primary key?

- A primary key is a combination of fields which uniquely specify a row. This is a special kind of unique key, and it has implicit NOT NULL constraint. It means, Primary key values cannot be NULL

7- What is a unique key?

- A Unique key constraint uniquely identified each record in the database. This provides uniqueness for the column or set of columns. A Primary key constraint has automatic unique constraint defined on it. But not, in the case of Unique Key. There can be many unique constraint defined per table, but only one Primary key constraint defined per table.

8- What is a foreign key?

- A foreign key is one table which can be related to the primary key of another table. Relationship needs to be created between two tables by referencing foreign key with the primary key of another table.

9- What is a join?

- This is a keyword used to query data from more tables based on the relationship between the fields of the tables. Keys play a major role when JOINS are used.

10- What are the types of join and explain each?

- There are various types of join which can be used to retrieve data and it depends on the relationship between tables.
- Inner join. Inner join return rows when there is at least one match of rows between the tables.
- Right Join. Right join return rows which are common between the tables and all rows of Right hand side table. Simply, it returns all the rows from the right hand side table even though there are no matches in the left hand side table.
- Left Join. Left join return rows which are common between the tables and all rows of Left hand side table. Simply, it returns all the rows from Left hand side table even though there are no matches in the Right hand side table.
- Full Join. Full join return rows when there are matching rows in any one of the tables. This means, it returns all the rows from the left hand side table and all the rows from the right hand side table.

11- What is normalization?

- Normalization is the process of minimizing redundancy and dependency by organizing fields and table of a database. The main aim of Normalization is to add, delete or modify field that can be made in a single table.

12- . What is Denormalization.

- DeNormalization is a technique used to access the data from higher to lower normal forms of database. It is also process of introducing redundancy into a table by incorporating data from the related tables.

13- What are all the different normalizations?

- The normal forms can be divided into 5 forms, and they are explained below -.
 - i. First Normal Form (1NF):**. This should remove all the duplicate columns from the table. Creation of tables for the related data and identification of unique columns.
 - ii. Second Normal Form (2NF):**. Meeting all requirements of the first normal form. Placing the subsets of data in separate tables and Creation of relationships between the tables using primary keys.
 - iii. Third Normal Form (3NF):**. This should meet all requirements of 2NF. Removing the columns which are not dependent on primary key constraints.
 - iv. Fourth Normal Form (3NF):**. Meeting all the requirements of third normal form and it should not have multi- valued dependencies.

14- What is a View?

- A view is a virtual table which consists of a subset of data contained in a table. Views are not virtually present, and it takes less space to store. View can have data of one or more tables combined, and it is depending on the relationship.

15- 15. What is an Index?

- An index is performance tuning method of allowing faster retrieval of records from the table. An index creates an entry for each value and it will be faster to retrieve data.

16- . What are all the different types of indexes?

- There are three types of indexes -.
 - i. Unique Index. This indexing does not allow the field to have duplicate values if the column is unique indexed. Unique index can be applied automatically when primary key is defined.
 - ii. Clustered Index. This type of index reorders the physical order of the table and search based on the key values. Each table can have only one clustered index. NonClustered Index.
 - iii. NonClustered Index does not alter the physical order of the table and maintains logical order of data. Each table can have 999 nonclustered indexes.

17- What is a Cursor?

- A database Cursor is a control which enables traversal over the rows or records in the table. This can be viewed as a pointer to one row in a set of rows. Cursor is very much useful for traversing such as retrieval, addition and removal of database records.

18- What is a relationship and what are they?

- Database Relationship is defined as the connection between the tables in a database. There are various data basing relationships, and they are as follows:.
 - One to One Relationship.
 - i. One to Many Relationship.
 - ii. Many to One Relationship.
 - iii. Self-Referencing Relationship

19- What is a query?

- A DB query is a code written in order to get the information back from the database. Query can be designed in such a way that it matched with our expectation of the result set. Simply, a question to the Database.

20- What is subquery?

- A subquery is a query within another query. The outer query is called as main query, and inner query is called subquery. SubQuery is always executed first, and the result of subquery is passed on to the main query.

21- What are the types of subquery?

- There are two types of subquery – Correlated and Non-Correlated. A correlated subquery cannot be considered as independent query, but it can refer the column in a table listed in the FROM the list of the main query. A Non-Correlated sub query can be considered as independent query and the output of subquery are substituted in the main query.

22- What is a stored procedure?

- Stored Procedure is a function consists of many SQL statement to access the database system. Several SQL statements are consolidated into a stored procedure and execute them whenever and wherever required.

23- . What is a trigger?

- A DB trigger is a code or programs that automatically execute with response to some event on a table or view in a database. Mainly, trigger helps to maintain the integrity of the database. Example: When a new student is added to the student database, new records should be created in the related tables like Exam, Score and Attendance tables.

24- What is the difference between DELETE and TRUNCATE commands?

- DELETE command is used to remove rows from the table, and WHERE clause can be used for conditional set of parameters. Commit and Rollback can be performed after delete statement. TRUNCATE removes all rows from the table. Truncate operation cannot be rolled back

25- What are local and global variables and their differences?

- Local variables are the variables which can be used or exist inside the function. They are not known to the other functions and those variables cannot be referred or used. Variables can be created whenever that function is called. Global variables are the variables which can be used or exist throughout the program. Same variable declared in global cannot be used in functions. Global variables cannot be created whenever that function is called.

26- . What is a constraint?

- Constraint can be used to specify the limit on the data type of table. Constraint can be specified while creating or altering the table statement. Sample of constraint are.
 - i. NOT NULL.
 - ii. CHECK.
 - iii. DEFAULT.
 - iv. UNIQUE.
 - v. PRIMARY KEY.
 - vi. FOREIGN KEY.

27- What is data Integrity?

- Data Integrity defines the accuracy and consistency of data stored in a database. It can also define integrity constraints to enforce business rules on the data when it is entered into the application or database.

28- . What is Auto Increment?

- Auto increment keyword allows the user to create a unique number to be generated when a new record is inserted into the table. AUTO INCREMENT keyword can be used in Oracle and IDENTITY keyword can be used in SQL SERVER. Mostly this keyword can be used whenever PRIMARY KEY is used.

29- What is the difference between Cluster and Non-Cluster Index?

- Clustered index is used for easy retrieval of data from the database by altering the way that the records are stored. Database sorts out rows by the column which is set to be clustered index. A nonclustered index does not alter the way it was stored but creates a complete separate object within the table. It point back to the original table rows after searching.

30- What is Datawarehouse?

- Datawarehouse is a central repository of data from multiple sources of information. Those data are consolidated, transformed and made available for the mining and online processing. Warehouse data have a subset of data called Data Marts.

31- What is Self-Join?

- Self-join is set to be query used to compare to itself. This is used to compare values in a column with other values in the same column in the same table. ALIAS ES can be used for the same table comparison.

32- What is Cross-Join?

- Cross join defines as Cartesian product where number of rows in the first table multiplied by number of rows in the second table. If suppose, WHERE clause is used in cross join then the query will work like an INNER JOIN.

33- What is user defined functions?

- User defined functions are the functions written to use that logic whenever required. It is not necessary to write the same logic several times. Instead, function can be called or executed whenever needed.

34- What are all types of user defined functions?

- Three types of user defined functions are.
 - i. Scalar Functions.
 - ii. Inline Table valued functions.
 - iii. Multi statement valued functions. Scalar returns unit, variant defined the return clause. Other two types return table as a return.

35- . What is collation?

- Collation is defined as set of rules that determine how character data can be sorted and compared. This can be used to compare A and, other language characters and also depends on the width of the characters. ASCII value can be used to compare these character data.

36- What are all different types of collation sensitivity?

- Following are different types of collation sensitivity -.
 - i. Case Sensitivity – A and a and B and b.
 - ii. Accent Sensitivity.
 - iii. Kana Sensitivity – Japanese Kana characters.
 - iv. Width Sensitivity – Single byte character and double byte character.

37- . Advantages and Disadvantages of Stored Procedure?

- Stored procedure can be used as a modular programming – means create once, store and call for several times whenever required. This supports faster execution instead of executing multiple queries. This reduces network traffic and provides better security to the data. Disadvantage is that it can be executed only in the Database and utilizes more memory in the database server.

38- What is Online Transaction Processing (OLTP)?

- Online Transaction Processing (OLTP) manages transaction based applications which can be used for data entry, data retrieval and data processing. OLTP makes data management simple and efficient. Unlike OLAP systems goal of OLTP systems is serving real-time transactions. Example – Bank Transactions on a daily basis.

39- What is CLAUSE?

- SQL clause is defined to limit the result set by providing condition to the query. This usually filters some rows from the whole set of records. Example – Query that has WHERE condition Query that has HAVING condition.

40- What is recursive stored procedure?

- A stored procedure which calls by itself until it reaches some boundary condition. This recursive function or procedure helps programmers to use the same set of code any number of times.

41- What is Union, minus and Intersect commands?

- UNION operator is used to combine the results of two tables, and it eliminates duplicate rows from the tables. MINUS operator is used to return rows from the first query but not from the second query. Matching records of first and second query and other rows from the first query will be displayed as a result set.
- INTERSECT operator is used to return rows returned by both the queries.

42- . What is an ALIAS command?

- ALIAS name can be given to a table or column. This alias name can be referred in WHERE clause to identify the table or column. Example-. Select st.StudentID, Ex.Result from student st, Exam as Ex where st.studentID = Ex. StudentID Here, st refers to alias name for student table and Ex refers to alias name for exam table.

43- What is the difference between TRUNCATE and DROP statements?

- TRUNCATE removes all the rows from the table, and it cannot be rolled back. DROP command removes a table from the database and operation cannot be rolled back.

44- What are aggregate and scalar functions?

- Aggregate functions are used to evaluate mathematical calculation and return single values. This can be calculated from the columns in a table. Scalar functions return a single value based on the input value. Example -. Aggregate – max(), count - Calculated with respect to numeric. Scalar – UCASE(), NOW() – Calculated with respect to strings.

45- How can you create an empty table from an existing table?

- Example will be -. Select * into studentcopy from student where 1=2 Here, we are copying student table to another table with the same structure with no rows copied.

46- How to fetch common records from two tables?

- Common records result set can be achieved by -. Select studentID from student.
INTERSECT Select StudentID from Exam

47- . How to fetch alternate records from a table?

- Records can be fetched for both Odd and Even row numbers -. To display even numbers-. Select studentId from (Select rowno, studentId from student) where mod(rowno,2)=0 To display odd numbers-. Select studentId from (Select rowno, studentId from student) where mod(rowno,2)=1 from (Select rowno, studentId from student) where mod(rowno,2)=1.[/sql]

48- How to select unique records from a table?

- Select unique records from a table by using DISTINCT keyword. Select DISTINCT StudentID, StudentName from Student.

49- What is the command used to fetch first 5 characters of the string?

- There are many ways to fetch first 5 characters of the string -. Select SUBSTRING(StudentName,1,5) as studentname from student Select RIGHT(Studentname,5) as studentname from student

50- Which operator is used in query for pattern matching?

- LIKE operator is used for pattern matching, and it can be used as -. 1. % - Matches zero or more characters. 2. _(Underscore) – Matching exactly one character.
- Example -. Select * from Student where studentname like 'a%' Select * from Student where studentname like 'ami_'