Feature	`private`	`public`
Access	Accessible only within the same class.	Accessible from anywhere in the program.
Visibility	Hidden from outside the class.	Visible and accessible by all classes and methods.
Usage	Used for internal logic or data encapsulation.	Used for exposing an interface to interact with objects.
Inheritance	Not accessible by derived classes.	Accessible by derived classes.

Other Access Modifiers in C#:

In addition to `private` and `public`, C# provides other access modifiers to control accessibility:

1. **`protected`**: Accessible within the same class and derived classes.
2. **`internal`**: Accessible within the same assembly (project), but not from other assemblies.
3. **`protected internal`**: Accessible within the same assembly or by derived classes in other assemblies.
4. **`private protected`**: Accessible within the containing class or derived classes within the same assembly.

Summary:

- **Private**: Restricts access to members, keeping them hidden from other classes and ensuring that only the class itself can modify its internal state.
- **Public**: Makes class members visible and accessible from any part of the application, exposing an interface for interaction with other objects.
- Understanding when to use `private` or `public` helps in maintaining proper encapsulation, security, and control over how objects in a system interact with one another.

Can we assign null to Struct variable?

🕐 **Non-nullable structs** cannot hold `null`.
🕐 You can assign `null` to a **nullable struct** by using `Nullable<T>` or `T?` syntax.

🕐 Always check whether a nullable struct has a value before using it to avoid exceptions.

What is interface?

An **interface** is a defined boundary or point of interaction between different systems, components, or entities, enabling them to communicate and work together. In software, it often refers to a contract that specifies methods and properties that implementing classes must provide, without defining how they should be implemented.

What is Web Services ?

A **web service** is a standardized way for different applications or systems to communicate over the internet using protocols such as HTTP, XML, SOAP, or REST. It allows software applications to interact and share data and functionality, regardless of their underlying platforms or programming languages.

Define Custom control ?

A **Custom Control** in C# is a user-defined component that extends the functionality of existing controls by deriving from a base control class. Custom controls are created to provide specific behaviors, properties, and appearance that are not available in standard controls. They are defined in a separate class file, can encapsulate complex functionality, and can be reused across multiple applications or projects, often allowing for more customization and versatility compared to user controls.

Example : Event Handling etc

Define User Control ?

A **User Control** in C# is a composite control that combines multiple existing controls into a single reusable component. It is defined in a `.ascx` file for ASP.NET applications or in a `.UserControl` file for Windows Forms applications. User controls allow developers to encapsulate a group of related controls and their associated behavior, making it easier to manage and reuse them across different forms or pages within an application. User controls provide design-time support, enabling drag-and-drop placement in visual design environments.

Ex:- creating the Page With using Drag and Drop .

What is Trigger ?

A **trigger** is a database object that automatically executes a specified action or set of actions in response to certain events on a particular table or view. Triggers are commonly used to enforce data integrity, implement business rules, or perform auditing by executing actions such as inserting, updating, or deleting data automatically when specified conditions are met. They can be defined for events like `INSERT`, `UPDATE`, or `DELETE` and can run either before or after these operations.

What is joins and Types of Joins ?

In SQL, **joins** are used to combine rows from two or more tables based on a related column between them. Here are the main types of joins:

Join Type	**Description**
INNER JOIN	Returns rows with matching values in both tables.
LEFT JOIN	Returns all rows from the left table, with `NULL` where there's no match.
RIGHT JOIN	Returns all rows from the right table, with `NULL` where there's no match.
FULL JOIN	Returns all rows when there is a match in either table.
CROSS JOIN	Returns the Cartesian product of the two tables (all combinations of rows).
SELF JOIN	Joins a table with itself to relate rows within the same table.

What are DML and DDL Commands ?

🕐 **Definition**: DML commands are used to modify and manage data within existing database tables. These commands handle the manipulation of the actual data itself.

Key DML Commands:

- `SELECT`: Retrieves data from the database.
- `INSERT`: Adds new data/rows into a table.
- `UPDATE`: Modifies existing data within a table.
- `DELETE`: Removes data/rows from a table.

DDL (Data Definition Language)

- **Definition**: DDL commands are used to define, alter, and manage the structure of database objects like tables, indexes, views, and schemas. These commands affect the schema (structure) of the database rather than the data itself.
- **Key DDL Commands**:
    - `CREATE`: Creates a new table, view, index, or other database objects.
    - `ALTER`: Modifies the structure of an existing table or other database objects (e.g., adding/removing columns).
    - `DROP`: Deletes a table, view, index, or other database objects from the database.
    - `TRUNCATE`: Removes all data from a table but keeps the structure intact.
    - `RENAME`: Renames an existing database object like a table.

What is Delegates and Events?

. Delegates

A **delegate** is a type that defines a method signature. It acts like a function pointer, allowing you to reference methods that match that signature. Delegates are primarily used for **callback methods** and for passing methods as parameters to other methods.

- **Type-safe**: Delegates ensure that the method being called matches the signature defined by the delegate.
- **Encapsulates method references**: A delegate can refer to a single method or a list of methods (multicasting).
- **Allows dynamic method invocation**: You can dynamically change the method that a delegate points to.

Events

An **event** is a mechanism in C# that is based on delegates and is used to provide **notifications**. Events allow a class to notify other classes or objects when something of interest happens, following the **publisher/subscriber model**.

- **Built on delegates**: Events use delegates under the hood but provide a more structured and restricted way to handle notifications.
- **Encapsulation**: Outside classes can only subscribe or unsubscribe to an event but cannot trigger it directly. Only the class that declares the event can raise it.
- **Supports multiple subscribers**: Multiple methods (subscribers) can respond to the same event.

🕐 **Delegates** are references to methods with a specific signature and can be used to implement callback functions or to pass methods as parameters.

🕐 **Events** are a specialized form of delegates that provide a safe way to notify subscribers when something happens in the class, implementing the observer pattern.

Key Differences Between Delegates and Events:

Feature	Delegates	Events
Usage	Directly used for method references.	Typically used for notifications.
Access	Can be called or assigned from anywhere.	Can only be raised by the class that declares it.
Subscription	Methods can be added or removed.	Methods can only subscribe or unsubscribe.
Encapsulation	No restrictions; anyone can invoke the delegate.	Restricted access; only the declaring class can raise the event.

Can we have Private Virtual Method?

🕐 **Private Virtual Method**: The method `Display` is declared as `private` and `virtual` in the `BaseClass`. It can be overridden by derived classes, but it can't be accessed directly outside the `BaseClass`.
🕐 **Calling the Private Method**: The `Show` method in `BaseClass` can call the `Display` method since it is within the same class.

🕐 **Derived Class**: The derived class `DerivedClass` cannot override `Display` because it is private. Therefore, any attempt to do so will result in a compile-time error.

What is Polymorphism?

Polymorphism is one of the core principles of object-oriented programming (OOP), alongside encapsulation and inheritance. The term polymorphism comes from the Greek words "poly" (meaning many) and "morph" (meaning form or shape). In programming, it refers to the ability of different classes to be treated as instances of the same class through a common interface.

1. Compile-time Polymorphism

This type of polymorphism is resolved during the compile time. The two most common forms are:

- **Method Overloading**: Multiple methods with the same name but different parameter lists (number or type of parameters) within the same class.

2. Run-time Polymorphism

This type of polymorphism is resolved during runtime. The most common form is:

- **Method Overriding**: A derived class provides a specific implementation of a method that is already defined in its base class. This is commonly achieved through virtual methods and overridden methods.

What is Ajax ?

AJAX (Asynchronous JavaScript and XML) is a web development technique used for creating interactive and dynamic web applications. It allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes, without requiring a full page refresh. This results in a smoother and more responsive user experience.

Key Components of AJAX

1. **Asynchronous**: AJAX allows web applications to send and receive data asynchronously. This means that the web page can continue to function while data is being fetched from the server in the background.
2. **JavaScript**: AJAX relies on JavaScript to send requests to the server and to process the response. JavaScript is used to manipulate the DOM (Document Object Model) of the web page.
3. **XML (or JSON)**: While the original term includes XML, AJAX can actually use various formats for data exchange, including:
    - **XML**: Used for structured data and is self-describing.

- **JSON (JavaScript Object Notation)**: A lightweight data interchange format that is easy to read and write for humans and easy to parse and generate for machines. It has become the preferred format in modern applications.
- **HTML**: Directly updating parts of the web page with HTML data.
- **Plain text**: Simple text responses.

How AJAX Works

The typical workflow of an AJAX operation is as follows:

1. **User Action**: A user triggers an event (like clicking a button or submitting a form) that requires data to be fetched or sent to the server.
2. **AJAX Request**: JavaScript creates an AJAX request (usually using the `XMLHttpRequest` object or the newer `fetch` API) and sends it to the server.
3. **Server Processing**: The server processes the request, performs any necessary actions (like querying a database), and sends back a response (usually in JSON or XML format).
4. **Update the Web Page**: The JavaScript receives the response and updates the relevant part of the web page without refreshing the entire page.

Advantages of AJAX

- **Improved User Experience**: AJAX enables smoother and faster interactions by loading data in the background without page reloads.
- **Reduced Server Load**: Only the necessary data is requested, reducing the amount of data sent over the network.
- **Enhanced Performance**: Loading data asynchronously improves the perceived performance of web applications.

Asynchronous and DOM ?

- 🕐 **Asynchronous programming** allows web applications to perform tasks without blocking the main thread, improving user experience by keeping interfaces responsive.
- 🕐 The **DOM** represents the structure of a web page as a tree of objects, enabling dynamic manipulation and interaction through JavaScript.

which binding Supports the sessions WCF?

Binding	Session Support
BasicHttpBinding	No
WSHttpBinding	Yes
WSDualHttpBinding	Yes
NetTcpBinding	Yes
NetNamedPipeBinding	Yes
NetWebHttpBinding	No

what are properies in web services ?

The properties of web services play a crucial role in their design, implementation, and interaction with clients. Understanding these properties helps developers create robust, flexible, and secure web services that can effectively meet the needs of various applications and users.

1. Interoperability

- **Definition**: Web services are designed to allow different applications from various sources to communicate with each other.
- **Importance**: This is achieved through the use of standard protocols and formats such as SOAP, REST, XML, and JSON, ensuring that diverse systems can work together seamlessly.

2. Loose Coupling

- **Definition**: Web services are loosely coupled, meaning that the consumer and provider of the service are independent of one another.
- **Importance**: Changes made to the service do not require changes to the client application, which promotes flexibility and easier maintenance.

3. Statelessness

- **Definition**: Most web services are stateless, meaning that each request from a client contains all the information necessary to understand and process that request.
- **Importance**: This simplifies the server design because it does not need to remember the state of client interactions, making scaling and load balancing easier.

4. Discoverability

- **Definition**: Web services can be discovered by clients through mechanisms such as WSDL (Web Services Description Language) for SOAP services or using API documentation for RESTful services.
- **Importance**: This property allows developers to understand how to interact with the service without needing to dive deep into the implementation details.

5. Standardized Protocols

- **Definition**: Web services utilize standardized protocols such as HTTP/HTTPS for communication, SOAP for message formatting, and REST for architectural style.
- **Importance**: This standardization ensures broad compatibility and simplifies the process of developing web services.

6. Scalability

- **Definition**: Web services can be designed to scale up or down based on demand.
- **Importance**: This property is essential for handling varying loads efficiently, whether through horizontal scaling (adding more machines) or vertical scaling (upgrading existing machines).

7. Platform Independence

- **Definition**: Web services can run on any platform and can be written in any programming language, as long as they adhere to the standard protocols.
- **Importance**: This allows organizations to use different technologies and still communicate with one another.

8. Security

- **Definition**: Web services can implement security protocols such as WS-Security for SOAP services or OAuth and HTTPS for RESTful services to ensure secure data transmission and authentication.
- **Importance**: This property is crucial for protecting sensitive data and maintaining trust between service providers and consumers.

9. Error Handling

- **Definition**: Web services should define clear mechanisms for error reporting and handling.
- **Importance**: Proper error handling ensures that clients can understand what went wrong during a service call and take appropriate action.

10. Versioning

- **Definition**: Web services should support versioning to allow for updates and changes without breaking existing client applications.
- **Importance**: This property enables developers to enhance or modify services while maintaining backward compatibility with older clients.

What is Desgin Pattern and Types of Desgin Pattern ?

Design patterns are essential tools in software engineering that provide proven solutions to common design problems. Understanding the types of design patterns and their appropriate use cases can lead to better software architecture, improved code quality, and enhanced collaboration among developers. By adopting design patterns, teams can create systems that are easier to maintain and extend, ultimately leading to more successful projects.

Design Pattern Types

Category	Purpose	Examples
Creational	Object creation and management	Singleton, Factory Method, Builder
Structural	Object composition and organization	Adapter, Decorator, Facade
Behavioral	Object interaction and communication	Observer, Strategy, Command

What is Pivot ?

The `PIVOT` keyword in SQL Server is a powerful tool for transforming and summarizing data by turning rows into columns. It simplifies complex queries, especially when creating reports or when you need to see categories of data as columns. However, it requires explicit listing of columns and aggregation functions, making it essential to understand the data structure when using it.

```
SELECT Product, [Jan], [Feb], [Mar]

FROM

(

   SELECT Product, Month, Sales

   FROM Sales

) AS SourceTable

PIVOT

(

    SUM(Sales)

    FOR Month IN ([Jan], [Feb], [Mar])

) AS PivotTable;
```

sql Angents , cluster index and non cluster indexs?

SQL Server Agent

SQL Server Agent is a component of Microsoft SQL Server that allows you to automate administrative tasks such as running SQL queries, backups, or other database-related tasks on a scheduled basis. It is often used for automating repetitive tasks that need to be executed periodically or in response to specific events.