[OOP 3](#_Toc8910)

[Inheritance 3](#_Toc24693)

[Polymorphism 3](#_Toc18905)

[Prevent derived classes from overriding virtual members? 4](#_Toc13720)

[Encapsulation 4](#_Toc3530)

[Abstraction 4](#_Toc14679)

[MVC and Web API 5](#_Toc18164)

[CORS? 5](#_Toc10044)

[Global asax? 6](#_Toc32058)

[Controller action return types in ASP.NET Core web API 6](#_Toc8244)

[Transient vs scoped vs singleton 7](#_Toc27671)

[Version history 7](#_Toc16107)

[For dotnet framework 7](#_Toc25636)

[For dotnet core 7](#_Toc6092)

[Views? 8](#_Toc23947)

[Partial Views? 8](#_Toc18967)

[What is view engine 8](#_Toc27461)

[IActionResult? 8](#_Toc12985)

[Ways to pass data to views? 8](#_Toc32022)

[Strongly vs weakly types models 8](#_Toc5827)

[Strongly typed - viewmodel 8](#_Toc3749)

[Weakly typed 9](#_Toc2736)

[Viewdata attribute? 10](#_Toc5161)

[View data vs view bag 11](#_Toc17637)

[Razor pages 12](#_Toc14703)

[Model Binding 12](#_Toc15620)

[Can we use http get to delete the record by passing an id? 12](#_Toc28249)

[Design patterns followed in Web Api application 12](#_Toc18190)

[What is repository pattern? 12](#_Toc21504)

[Explain Decorator Pattern? 12](#_Toc4203)

[References 13](#_Toc30739)

[C# 13](#_Toc22092)

[Func vs Predicate vs Action vs Lambda 13](#_Toc30782)

[Const vs Readonly 13](#_Toc10245)

[Async await? 14](#_Toc13708)

[TAP - Task based asynchronous programming? 14](#_Toc23888)

[Assembly info file? 14](#_Toc16427)

[Process vs thread vs task? 14](#_Toc26522)

[Message queues 14](#_Toc9720)

[LINQ 14](#_Toc7884)

[LINQ Keywords with samples 14](#_Toc25274)

[Deferred vs Immediate execution 14](#_Toc27940)

[IEnumerable and IQueryable 15](#_Toc26364)

[Is and as keywords 16](#_Toc6943)

[First, firstordefault, single, singleordefault 16](#_Toc4833)

[Async parallelism 17](#_Toc17293)

[Instance for abstract class cannot be created. What is that we are achieving by doing this? 17](#_Toc11365)

[Select vs SelectMany 17](#_Toc31028)

[Include and theninclude 17](#_Toc21743)

[How to pass data to base class parameterized constructor 17](#_Toc24602)

[Foreground vs background threads 18](#_Toc30375)

[IComparable vs IComparer 18](#_Toc15904)

[Print Duplicate numbers without looping 18](#_Toc20679)

[Print 1000 times without looping 18](#_Toc32335)

[Dispose vs finalize 18](#_Toc9002)

[Can we have dictionary key to be null? 18](#_Toc21102)

[Explain all methods of object class 18](#_Toc14484)

[Can we inherit static class? 18](#_Toc508)

[IComparable, IEqualityComparer, Equals 19](#_Toc27432)

[Nullable reference type? 19](#_Toc29867)

[Shallow vs deep copy and Equals vs ReferenceEquals? 19](#_Toc2433)

[Dotnet framework and core 20](#_Toc30094)

[Dotnet core Features? 20](#_Toc13736)

[Dotnet Framework features? 20](#_Toc7373)

[Deferred execution? 20](#_Toc5239)

[How to create, deploy and do version management of nuget package? 20](#_Toc31708)

[Web 20](#_Toc6515)

[CDN? 20](#_Toc18569)

[Rest Architecture 20](#_Toc23080)

[Serverless architecture? 21](#_Toc25948)

[What is SSL/TLS? 21](#_Toc28025)

[Public key and private key 21](#_Toc23075)

[Symmetric and Asymmetric cryptography 21](#_Toc16023)

[Http vs Https? 21](#_Toc9623)

[Encryption algorithms? 21](#_Toc14449)

[Web sockets? 21](#_Toc22916)

[Filters in web api? 21](#_Toc5594)

[Various http codes with explanation 21](#_Toc15578)

[Database 21](#_Toc17344)

[Versions of SQL server 21](#_Toc19341)

[Indexing? 22](#_Toc28190)

[Various indexing strategies? 22](#_Toc27183)

[Clustered vs non clustered index? 22](#_Toc21916)

[Normalization? 22](#_Toc5035)

[Types? 23](#_Toc15085)

[Stored procedures? 23](#_Toc14731)

[How to optimize and points to be considered while implementing stored procedures? 23](#_Toc22129)

[SQL vs No SQL 23](#_Toc17836)

[Cursor? 23](#_Toc16969)

[Constraints? 23](#_Toc25987)

[Triggers? 23](#_Toc32208)

[Advantages of triggers 23](#_Toc1896)

[Transaction? 24](#_Toc10711)

[DBCC Commands 24](#_Toc8406)

[Primary key vs unique key 24](#_Toc3363)

[Features of SSMS 24](#_Toc20434)

[Execution plan 24](#_Toc16048)

[How to backup database in MS SQL 24](#_Toc9291)

[Various backup strategies 24](#_Toc6155)

[What is execution plan 25](#_Toc23943)

[Clustered vs non clustered 25](#_Toc16266)

[Function vs stored procedure 25](#_Toc1035)

[What is table scan 26](#_Toc12573)

[Magic tables? 26](#_Toc19349)

[Query: Nagarro 26](#_Toc8912)

[Query: Employees greater than average salary 27](#_Toc17960)

[Query: Take last five recent records from database 27](#_Toc18997)

[Query processing phases 27](#_Toc30779)

[What is raiseerror function in MS SQL 27](#_Toc29266)

[When to use begin and end keywords 28](#_Toc8066)

[Query: Option to delete duplicates in a table with single column 28](#_Toc3526)

[Query: If first n login is not successful make disable the user login for 30 minutes and enable it after 30 minutes 28](#_Toc18673)

[Delete vs Truncate vs Drop 30](#_Toc3900)

[What is Data integrity 30](#_Toc30696)

[Normalization and Denormalization 31](#_Toc25198)

[Entities and Relationships 31](#_Toc28584)

[What is ACID property in DB 31](#_Toc29252)

[Subquery 32](#_Toc31104)

[Explain CAP theorm 32](#_Toc19662)

[Query: To get all column names of a table 32](#_Toc5459)

[React 32](#_Toc15400)

[Features of react? 32](#_Toc12137)

[What is virtual DOM? 32](#_Toc9112)

[Class vs functional component? 32](#_Toc3494)

[Component lifecycle methods 32](#_Toc10840)

[Phases of components 34](#_Toc30622)

[Mounting 34](#_Toc18921)

[Updating 34](#_Toc30750)

[Unmount 34](#_Toc17391)

[React internal Phases 34](#_Toc19486)

[Render Phase 34](#_Toc25562)

[Pre-Commit Phase 34](#_Toc29249)

[Commit Phase 34](#_Toc21224)

[Constructor 34](#_Toc10058)

[getDerivedStateFromProps 34](#_Toc14674)

[Render 34](#_Toc16167)

[ComponentDidMount 34](#_Toc32143)

[ComponentDidUpdate 34](#_Toc1934)

[ComponentWillUnmount 34](#_Toc15134)

[Export vs export default 34](#_Toc2105)

[Reconciliation? 34](#_Toc5173)

[Diffing? 35](#_Toc22083)

[Fiber tree? 35](#_Toc24131)

[Hooks? Various hooks? 35](#_Toc6969)

[Explain Usestate hook 35](#_Toc26408)

[Explain Useeffect hook 35](#_Toc13780)

[Explain Usecontext hook 35](#_Toc16786)

[Write a custom hook 35](#_Toc18819)

[Unmounting vs Re-rendering 35](#_Toc14563)

[Flux architectural desgin pattern 35](#_Toc31520)

[Javascript 36](#_Toc20862)

[Promise 36](#_Toc19602)

[Closure 36](#_Toc8321)

[Call, bind and apply 36](#_Toc22048)

[Debounce and throttling 36](#_Toc24107)

[References 36](#_Toc21445)

[General 36](#_Toc28792)

[Explain about your current project 36](#_Toc28106)

# OOP

## Inheritance

Inheritance enables you to create new classes that **R**euse, **E**xtend, and **M**odify the behavior defined in other classes. The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class

## Polymorphism

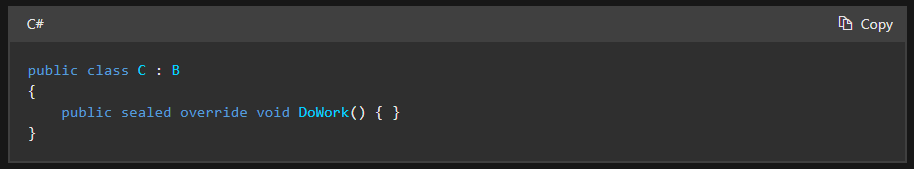
Polymorphism - Greek word - "many-shaped" and it has two distinct aspects:

At run time, objects of a derived class may be treated as objects of a base class in places such as method parameters and collections or arrays. When this polymorphism occurs, the object's declared type is no longer identical to its run-time type.

Base classes may define and implement virtual methods, and derived classes can override them, which means they provide their own definition and implementation. At run-time, when client code calls the method, the CLR looks up the run-time type of the object, and invokes that override of the virtual method. In your source code you can call a method on a base class, and cause a derived class's version of the method to be executed.

### Prevent derived classes from overriding virtual members?

A derived class can stop virtual inheritance by declaring an override as sealed.



## Encapsulation

A class or struct can specify how accessible each of its members is to code outside of the class or struct. Methods and variables that aren't intended to be used from outside of the class or assembly can be hidden to limit the potential for coding errors or malicious exploits.

## Abstraction

An abstraction is a type that describes a contract but does not provide a full implementation of the contract. Abstractions are usually implemented as abstract classes or interfaces, and they come with a well-defined set of reference documentation describing the required semantics of the types implementing the contract. Some of the most important abstractions in the .NET Framework include Stream, IEnumerable<T>, and Object.

You can extend frameworks by implementing a concrete type that supports the contract of an abstraction and using this concrete type with framework APIs consuming (operating on) the abstraction.

A meaningful and useful abstraction that is able to withstand the test of time is very difficult to design. The main difficulty is getting the right set of members, no more and no fewer. If an abstraction has too many members, it becomes difficult or even impossible to implement. If it has too few members for the promised functionality, it becomes useless in many interesting scenarios.

Too many abstractions in a framework also negatively affect usability of the framework. It is often quite difficult to understand an abstraction without understanding how it fits into the larger picture of the concrete implementations and the APIs operating on the abstraction. Also, names of abstractions and their members are necessarily abstract, which often makes them cryptic and unapproachable without first understanding the broader context of their usage.

However, abstractions provide extremely powerful extensibility that the other extensibility mechanisms cannot often match. They are at the core of many architectural patterns, such as plug-ins, inversion of control (IoC), pipelines, and so on. They are also extremely important for testability of frameworks. Good abstractions make it possible to stub out heavy dependencies for the purpose of unit testing. In summary, abstractions are responsible for the sought-after richness of the modern object-oriented frameworks.

❌ DO NOT provide abstractions unless they are tested by developing several concrete implementations and APIs consuming the abstractions.

✔️ DO choose carefully between an abstract class and an interface when designing an abstraction.

✔️ CONSIDER providing reference tests for concrete implementations of abstractions. Such tests should allow users to test whether their implementations correctly implement the contract.

The abstract modifier indicates that the thing being modified has a missing or incomplete implementation. The abstract modifier can be used with classes, methods, properties, indexers, and events. Use the abstract modifier in a class declaration to indicate that a class is intended only to be a base class of other classes, not instantiated on its own. Members marked as abstract must be implemented by non-abstract classes that derive from the abstract class.

# MVC and Web API

## CORS?

Cross-Origin Resource Sharing (CORS) is a security feature that restricts web applications from making requests to a domain different from the domain from which the first resource was served. This is an important security measure to prevent malicious websites from accessing sensitive data through a user's browser. However, in modern web applications, especially those that use APIs, you often need to allow requests from different origins.

In the context of ASP.NET Core Web API, handling CORS involves configuring the middleware to support cross-origin requestsCross Origin Resourse Sharing

services.AddCors(options =>

{

options.AddPolicy("AllowSpecificOrigin",

builder => builder.WithOrigins("http://example.com")

.AllowAnyHeader()

.AllowAnyMethod());

});

app.UseCors("AllowSpecificOrigin");

[EnableCors("AllowSpecificOrigin")]

public class TestController : ControllerBase

{

// Controller methods

}

services.AddCors(options =>

{

options.AddPolicy("AllowMultipleOrigins",

builder => builder.WithOrigins("http://example.com", "http://example.org")

.AllowCredentials()

.AllowAnyHeader()

.AllowAnyMethod());

});

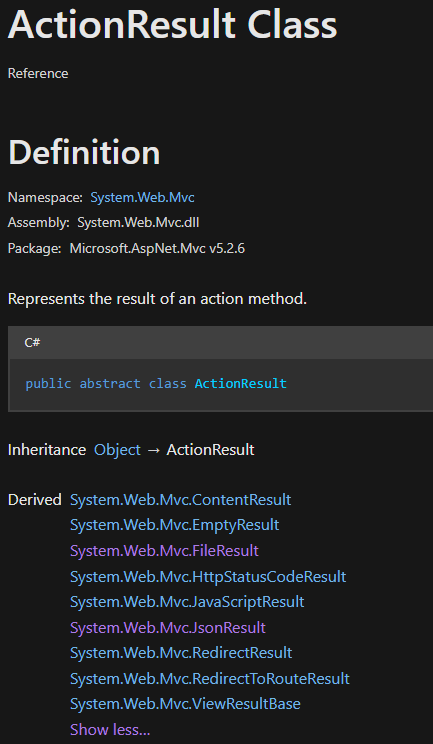
## Global asax?

## Controller action return types in ASP.NET Core web API

Reference: <https://learn.microsoft.com/en-us/aspnet/core/web-api/action-return-types?view=aspnetcore-8.0>

1. Specific type
2. IActionResult
3. ActionsResult<T>
4. HttpResults

Derived types of ActionResult



ActionResult<T> vs just returning T

ActionResult<T>

Description: A generic class that combines the flexibility of ActionResult with the strong typing of a response model T.

Advantages:

Can return both T (a strongly-typed model) or any other IActionResult type like NotFound, BadRequest, etc.

Offers greater flexibility for returning different HTTP status codes.

T (Model Only)

Description: Returning just the model type directly.

Advantages:

Simplifies the code when only a successful response is expected.

Useful for APIs where only a specific model response is required.

Comparison Table

| Feature | ActionResult<T> | T (Model) |
| --- | --- | --- |
| Strong Typing | Yes | Yes |
| Status Code Flexibility | Yes | No |
| Return Model Only | Yes | Yes |
| Return Other Status Codes | Yes (e.g., NotFound, BadRequest) | No (Implicitly returns 200 OK if valid) |
| Overloading Capability | Yes (Supports overloads) | No (Model must be returned directly) |

## Transient vs scoped vs singleton

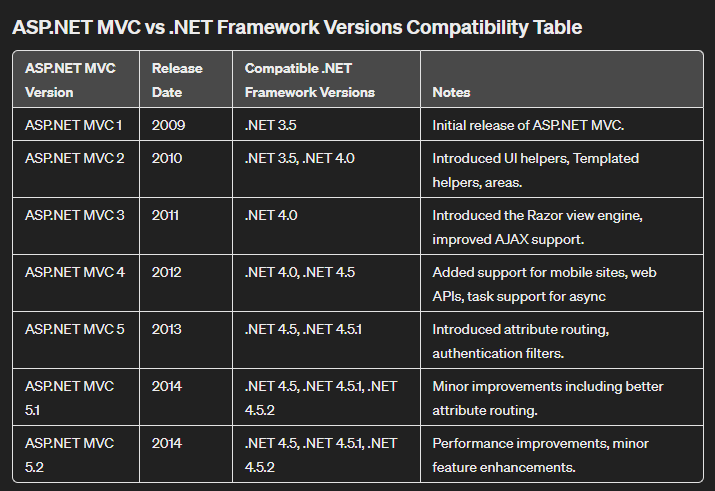
Use cases for each

Singleton: Caching

Scoped: web api Tokens

## Version history

### For dotnet framework



### For dotnet core

| **Version** | **Release Date** | **Key Features** |
| --- | --- | --- |
| ****ASP.NET Core 1.0**** | June 2016 | Introduced cross-platform support for Windows, macOS, and Linux. Modular HTTP request pipeline. Integrated dependency injection. |
| ****ASP.NET Core 1.1**** | November 2016 | Improvements to Azure integration, URL rewriting, and more middleware components. |
| ****ASP.NET Core 2.0**** | August 2017 | Introduction of Razor Pages, compatibility layer with .NET Framework libraries, and performance enhancements. |
| ****ASP.NET Core 2.1**** | May 2018 | Implementation of SignalR for real-time web functionality, GDPR-related features, and improved build performance. |
| ****ASP.NET Core 2.2**** | December 2018 | Enhancements in HTTP/2 support, health checks for apps, and overall performance improvements. |
| ****ASP.NET Core 3.0**** | September 2019 | Switch to .NET Core 3.0, support for Windows Desktop Apps, significant JSON API improvements, and enhanced support for gRPC. |
| ****ASP.NET Core 3.1**** | December 2019 | Long-term support version, minor updates to features introduced in 3.0, focused on stability and enhancements. |
| ****ASP.NET Core 5.0**** | November 2020 | Skipped version 4 to align with .NET 5, unified .NET SDK experience, new Web API improvements, Blazor updates, and more performance improvements. |
| ****ASP.NET Core 6.0**** | November 2021 | Continued improvements to Blazor, minimal APIs for hosting and routing in web applications, and enhanced performance. |
| ****ASP.NET Core 7.0**** | November 2022 | Updates to minimal APIs, improved support for gRPC and SignalR, further performance enhancements, and more features in Blazor for building interactive client-side UIs. |

## Views?

## Partial Views?

## What is view engine

## IActionResult?

## Ways to pass data to views?

Strongly typed data

**viewmodel**

Weakly typed data

**ViewData (ViewDataAttribute)**

**ViewBag**

## Strongly vs weakly types models

Strong typing (or strongly typed) means that every variable and constant has an explicitly defined type (for example, string, int, or DateTime). The validity of types used in a view is checked at compile time.

Weak types (or loose types) means that you don't explicitly declare the type of data you're using. You can use the collection of weakly typed data for passing small amounts of data in and out of controllers and views.

### Strongly typed - viewmodel

In view, Specify a model using the **@model directive**. **Use** the model **with** **@Model**.

Only one **@model** can be used in view.

Same and different models can be considered for view and viewmodel. But its recommended to use different models for flexibility.

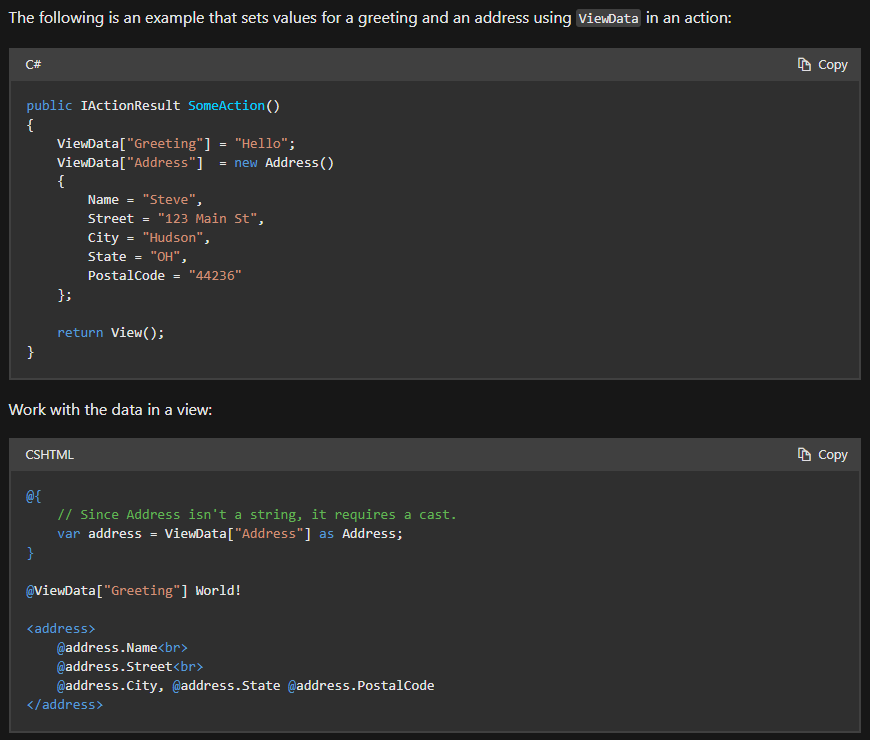
***SS Explanation: Address is a viewmodel. Passed as a parameter for View from contact controller.***



### Weakly typed

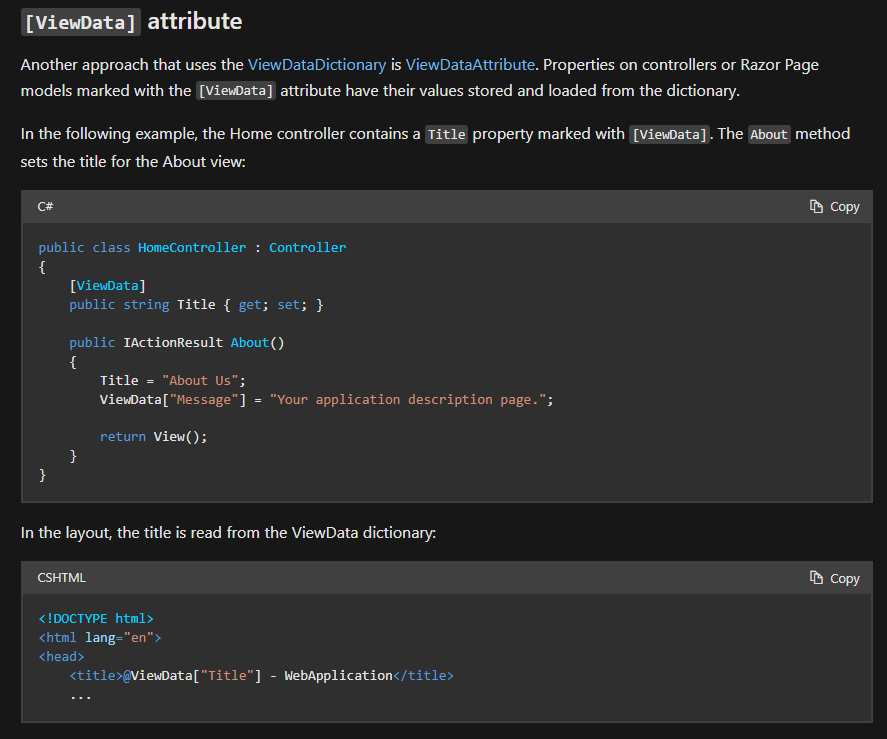
The **ViewData** property is a **dictionary** of weakly typed objects. The **ViewBag** property is a **wrapper around ViewData** that provides dynamic properties for the underlying ViewData collection.

Both are dynamically resolved at runtime.

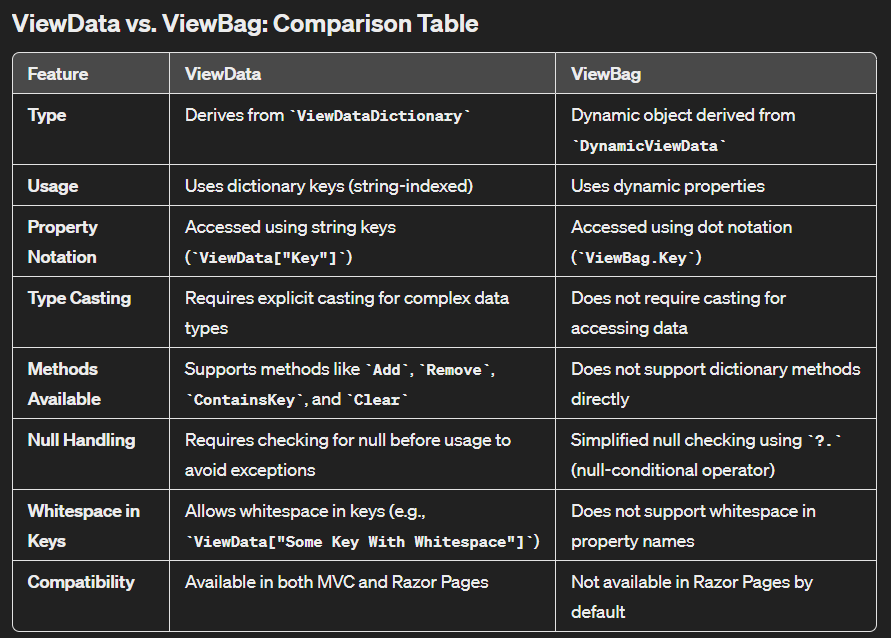


### Viewdata attribute?

Add this attribute in controller and use it in view.



### View data vs view bag



### Razor pages

## Model Binding

TODO:: Go through the below link and update

<https://learn.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-8.0>

## Can we use http get to delete the record by passing an id?

Yes its doable but not recommended as per REST standard. Use HttpDelete

## Design patterns followed in Web Api application

Repository Pattern, Dependency Injection (DI) Pattern, Decorator Patter

## What is repository pattern?

Abstraction for data access layer. Participants: IRespository and Repository. The Repository should implement IRespository. We will have multiple repository like SQL, Mock or other data source.

## Explain Decorator Pattern?

Attach additional responsibilities to an object dynamically.

Alternative to sub classing for extending funcitonality. If class cannot be extended I.e. if its sealed then use decorator.

Its an example of structural design pattern

Constriant: Hyundai and suzuki are sealed class.

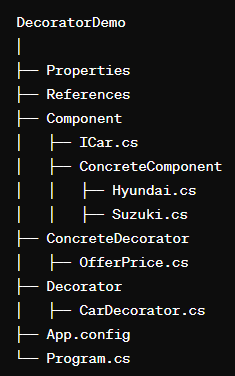
To apply offer price to this cars, we have CarDecorator abstract class that implements ICar and takes ICar instance as parameter. Also adds a new abstract method GetDiscountedPrice().

Write new concrete decorator class OfferPrice that implements CarDecorator.

ICar HynCar=new Hyundai();

CarDecorator Decorator=new OfferPrice(HynCar);

Decorator.GetDiscountedPrice(). This will provide the discount price for specific car.



Refer this video for above classes explanation

https://www.youtube.com/watch?v=YObNH6IkIos

## References

https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0

# C#

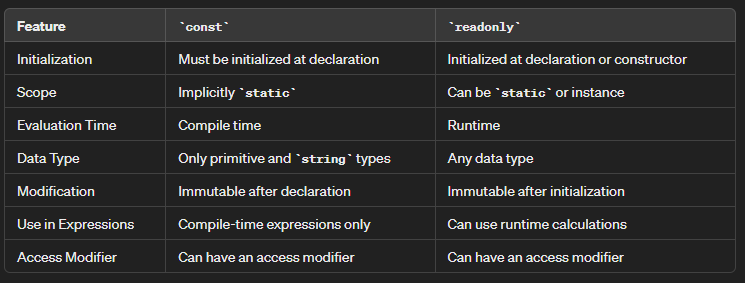
### Func vs Predicate vs Action vs Lambda

### Const vs Readonly

**C**onst - **C**omplie

**R**eadonly - **R**untime

Can add static to both const and readonly



| **Feature** | **const** | **readonly** |
| --- | --- | --- |
| Initialization | Must be **initialized at declaration** | Can be initialized at **declaration or within constructor** |
| Value Change | Cannot be changed after initialization | Cannot be changed after initialization |
| Scope | Local or static scope | Instance or static scope |
| Usage | Suitable for compile-time constants | For runtime constants or values determined at runtime |
| Evaluation | Evaluated and replaced with literal values by compiler during compilation | Evaluated at runtime, values determined dynamically during program execution |

### Async await?

### TAP - Task based asynchronous programming?

### Assembly info file?

### Process vs thread vs task?

### Message queues

### LINQ

Language Integrated Query. Strongly typed syntax. One language for many sources.

Ways to implement: Query syntax and fluent syntax.

### LINQ Keywords with samples

### Deferred vs Immediate execution

| ****Aspect**** | ****Immediate Execution**** | ****Deferred Execution**** |
| --- | --- | --- |
| ****Definition**** | Execution of the query is performed as soon as it is declared. | Execution of the query is delayed until the result is specifically requested. |
| ****Execution Trigger**** | Triggered immediately by the declaration of the query. | Triggered by enumerating or converting the query result to a collection (**ToList**, **ToArray**, etc.). |
| ****Result Type**** | Produces a collection (e.g., **List**, **Array**) directly. | Produces a query object (e.g., **IEnumerable**, **IQueryable**) representing the query. |
| ****Common Methods**** | - **ToList()** | - **Where** |
|  | - **ToArray()** | - **Select** |
|  | - **Count()** | - **OrderBy** |
|  | - **First()**, **Single()**, **Max()**, etc. | - **GroupBy** |
| ****Execution Context**** | Executes in the current process or scope. | Executes on-demand in the appropriate context (current process, remote database, etc.). |
| ****Performance Implications**** | May involve processing the entire dataset upfront. | More efficient in terms of data loading, but may require extra optimizations. |
| ****Examples in LINQ**** | ```csharp | ```csharp |
|  | var numbers = new List<int> {1, 2, 3, 4}; | var query = numbers.Where(n => n > 2); |
|  | var count = numbers.Count(); | // Query not executed yet |
|  | var selected = numbers.ToList(); | var result = query.ToList(); |
|  | ``` | ``` |
| ****Use Cases**** | - Aggregation (e.g., **Count**, **Sum**, **Max**) | - Filtering (**Where**) |
|  | - Conversion to collections (**ToList**, **ToArray**) | - Projections (**Select**) |
|  | - Single element retrieval (**First**, **Single**) | - Ordering (**OrderBy**) |
|  | - Query execution during debugging | - Building queries step-by-step |
| ****Pros**** | - Provides immediate results for analysis. | - Delays execution to optimize data fetching. |
|  | - Useful for quick aggregation. | - Allows query modification before execution. |
|  | - Simplifies error detection in the query. | - May lead to performance gains by avoiding redundant data loading. |
| ****Cons**** | - May lead to unnecessary data processing. | - Delayed errors if queries are not correctly defined. |
|  | - Less efficient if data is not required immediately. | - Difficult to track query execution order. |
| ****When to Use**** | - For quick results and analysis. | - For building efficient, step-by-step queries. |
|  | - When aggregating or converting to a collection. | - When working with large datasets requiring optimized filtering. |

### IEnumerable and IQueryable

| **Feature** | **IEnumerable** | **IQueryable** |
| --- | --- | --- |
| Definition | An interface that defines one method, **GetEnumerator**, which returns an enumerator that can iterate over a collection. | An interface for querying data that inherits from **IEnumerable** and allows LINQ queries to be executed against a data source. |
| Namespace | **System.Collections** or **System.Collections.Generic** | **System.Linq** |
| Execution | Executes queries in the memory of the application (client-side). | Translates queries into SQL or another query language and executes them on the data source (server-side). |
| Use Cases | Best for working with in-memory data collections like arrays, lists, etc. | Best for querying data from remote data sources, like databases or web services. |
| Deferred Execution | Supports deferred execution through methods that return **IEnumerable**. | Supports deferred execution, and allows for query modifications before execution. |
| Data Retrieval | Retrieves all data from the source and then applies filters or transformations in-memory. | Applies filters or transformations before retrieving data, fetching only the necessary data from the source. |
| Performance | Potentially less efficient for large datasets as it operates on all data in-memory. | More efficient for large datasets or remote data sources due to its ability to query data source directly. |
| Suitability for Data Size | More suitable for small to medium datasets. | More suitable for large datasets or when working with data operations that need to be optimized by the database engine. |
| Examples | **List<T>**, **Array**, **Collection<T>** | **IQueryable<T>** often used with Entity Framework to query databases directly. |

### Is and as keywords

### First, firstordefault, single, singleordefault

| **Method** | **Description** | **When to Use** | **Example Code** |
| --- | --- | --- | --- |
| **Single** | Retrieves a single, specific element from a sequence. Throws an exception if not exactly one element matches. | Use when you're sure there's exactly one matching element. | **csharp List<int> numbers = new List<int> { 1, 2, 3, 4 }; int result = numbers.Single(n => n == 3);** |
| **SingleOrDefault** | Retrieves a single element from a sequence, or a default value if the sequence is empty. Throws an exception if more than one element matches. | Use when there should be at most one matching element. | **csharp List<int> numbers = new List<int> { 1, 2, 3, 4 }; int result = numbers.SingleOrDefault(n => n == 5); // returns 0** |
| **First** | Retrieves the first element of a sequence. Throws an exception if the sequence is empty. | Use when you need the first element and the sequence isn’t empty. | **csharp List<int> numbers = new List<int> { 1, 2, 3, 4 }; int result = numbers.First(); // returns 1** |
| **FirstOrDefault** | Retrieves the first element of a sequence, or a default value if the sequence is empty. | Use when the first element is needed but the sequence might be empty. | **csharp List<int> numbers = new List<int> { 1, 2, 3, 4 }; int result = numbers.FirstOrDefault(n => n > 4); // returns 0** |

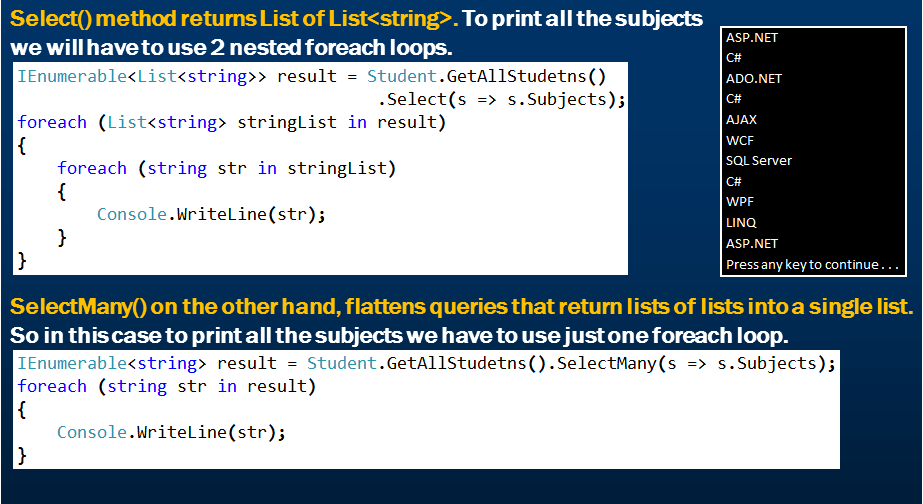
### Async parallelism

### Instance for abstract class cannot be created. What is that we are achieving by doing this?

### Select vs SelectMany

**Select:** Projects each element of a sequence into a new form.

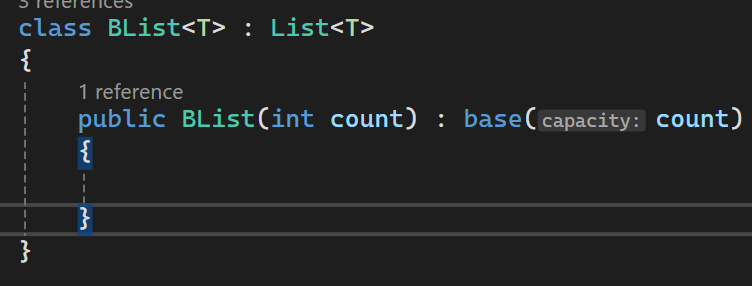
**SelectMany:** Projects each element of a sequence to an IEnumerable<T> and **flattens** the resulting sequences into one sequence.



### Include and theninclude

### How to pass data to base class parameterized constructor

Here List<T> has one constructor with (int capacity) as parameter

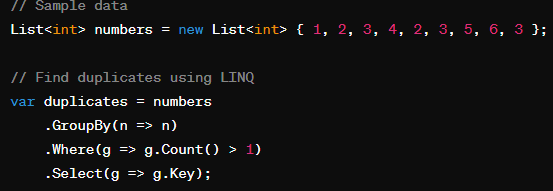


### Foreground vs background threads

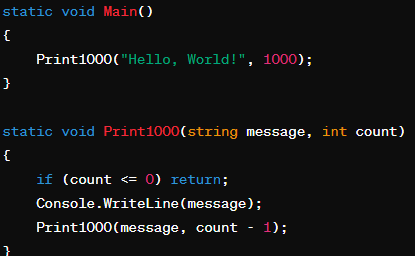
### IComparable vs IComparer

### Print Duplicate numbers without looping

TODO:: Need to run and check



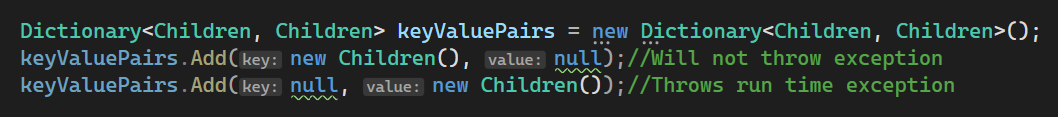
### Print 1000 times without looping



### Dispose vs finalize

### Can we have dictionary key to be null?

Will throw runtime exception



### Explain all methods of object class

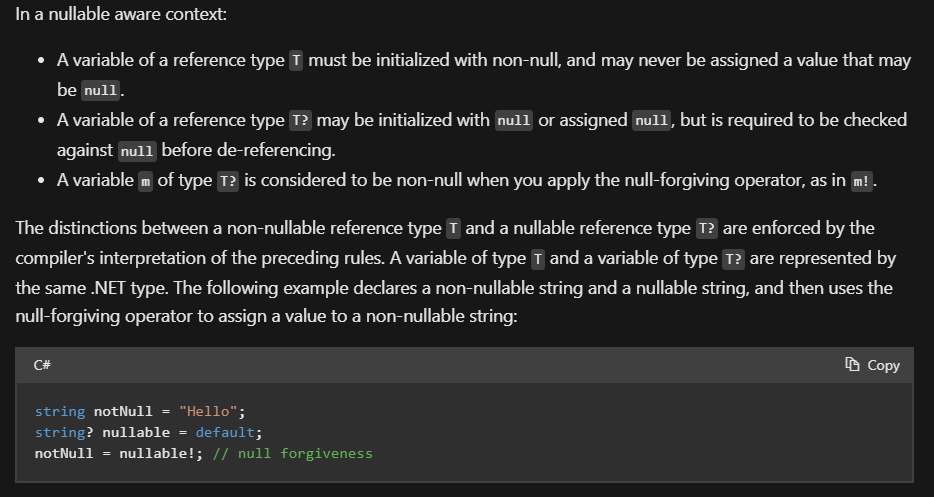
### Can we inherit static class?

Cannot be inherited. Its considered to be sealed class

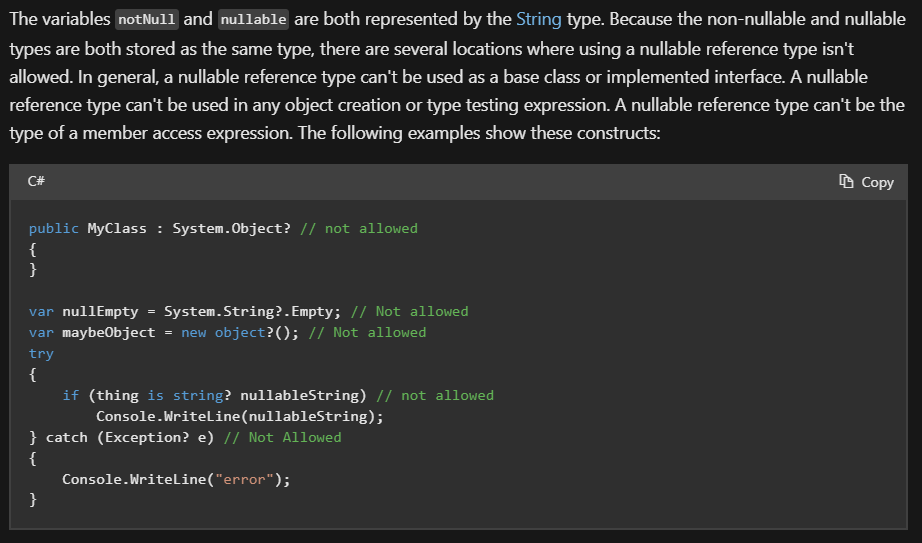
### IComparable, IEqualityComparer, Equals

### Nullable reference type?

The compiler uses those annotations(? or !(null forgiving operator)-> to assign null for a notnull variable) to help you find potential null reference errors in your code. There's no runtime difference between a non-nullable reference type and a nullable reference type. The compiler doesn't add any runtime checking for non-nullable reference types. The benefits are in the compile-time analysis. The compiler generates warnings that help you find and fix potential null errors in your code. You declare your intent, and the compiler warns you when your code violates that intent.



Not allowed in few places



### Shallow vs deep copy and Equals vs ReferenceEquals?

| ****Aspect**** | ****Shallow Copy**** | ****Deep Copy**** |
| --- | --- | --- |
| ****Description**** | Copies the outer object, but not the nested or referenced objects. References point to the same instances as in the original. | Copies the outer object and recursively copies all objects it references. Each copy is an entirely separate instance. |
| ****C# Example**** | Suppose we have an object **Person** with a property **Address**. A shallow copy of a **Person** object would create a new **Person** instance, but both would reference the same **Address** object. | For the same **Person** object, a deep copy would create a new **Person** instance and also a new, separate **Address** instance, so modifications to one will not affect the other. |
| ****Equals**** | TODO | TODO |
| ****Reference Equals**** | TODO | TODO |

# Dotnet framework and core

## Dotnet core Features?

## Dotnet Framework features?

## Deferred execution?

## How to create, deploy and do version management of nuget package?

# Web

## CDN?

## Rest Architecture

TODO ::: Update the six constraints

## Serverless architecture?

## What is SSL/TLS?

## Public key and private key

<https://www.youtube.com/watch?v=0ctat6RBrFo&t=343s>

## Symmetric and Asymmetric cryptography

## Http vs Https?

| **Feature** | **HTTP** | **HTTPS** |
| --- | --- | --- |
| Protocol | Hypertext Transfer Protocol | Secure Hypertext Transfer Protocol |
| Encryption and security | No encryption | Encrypted using SSL/TLS encryption |
| Port | 80 | 443 |
| Certificate | Not required | Required, to verify server's identity and establish secure connection |
| Performance | Generally faster due to lack of encryption | Slightly slower due to encryption overhead, but performance impact is minimal with modern hardware and protocols |
| Data Integrity | Data can be intercepted and modified in transit | Data integrity is maintained, as data is encrypted and tamper-proof |
| Authentication | No authentication by default | Server authentication and sometimes client authentication through certificates |

## Encryption algorithms?

## Web sockets?

## Filters in web api?

## Various http codes with explanation

# Database

## Versions of SQL server

2022 is the latest version



## Indexing?

### Various indexing strategies?

### Clustered vs non clustered index?

| **Feature** | **Clustered Index** | **Non-Clustered Index** |
| --- | --- | --- |
| ****Physical Organization**** | Directly organizes the table's data. | Organized separately from table data. |
| ****Leaf Nodes**** | Contains actual data rows. | Contains pointers to actual data rows. |
| ****Number Per Table**** | One per table. | Multiple per table. |
| ****Search Performance**** | Faster for range queries. | Slower for range queries compared to clustered. |
| ****Space Requirements**** | Requires no additional storage space. | Requires additional storage space. |

## Normalization?

Normalization in SQL is a process designed to minimize redundancy and dependency by organizing fields and table of a database. The main goal of normalization is to reduce redundancy and ensure data integrity. It's used widely in relational database design.

### Types?

## Stored procedures?

### How to optimize and points to be considered while implementing stored procedures?

1. Use parameterized strored procedures

## SQL vs No SQL

| **Feature** | **SQL (Relational Databases)** | **NoSQL (Non-relational Databases)** |
| --- | --- | --- |
| ****Data Structure**** | Structured data in tables with rows and columns | Flexible data models (key-value, document, graph, columnar) |
| ****Schema**** | Fixed schema; modifications require alterations to tables | Dynamic schema for unstructured data |
| ****Scalability**** | Vertical scaling (increase server capacity) | Horizontal scaling (add more servers) |
| ****Transactions**** | ACID (Atomicity, Consistency, Isolation, Durability) | BASE (Basically Available, Soft state, Eventual consistency) |
| ****Querying**** | Complex querying using SQL | Query capabilities vary by type (e.g., simple lookups, aggregation) |
| ****Consistency**** | Strong consistency | Consistency can vary (strong, eventual, causal) |
| ****Use Case**** | Well-suited for complex queries and reporting | Ideal for large volumes of data with simple queries |
| ****Examples**** | MySQL, PostgreSQL, Oracle, SQL Server | MongoDB, Cassandra, DynamoDB, Couchbase |
| ****Best for**** | Banking systems, CRM systems, complex applications | Real-time web apps, big data analytics, distributed databases |

## Cursor?

## Constraints?

## Triggers?

## Advantages of triggers

## Transaction?

## DBCC Commands

DBCC (Database Console Commands) in SQL Server are a set of commands that provide maintenance, validation, and other utility operations on a SQL Server database. These commands can be used for a variety of tasks such as checking database integrity, managing database storage, collecting and analyzing performance information, and more.

TODO:: Add some DBCC commands

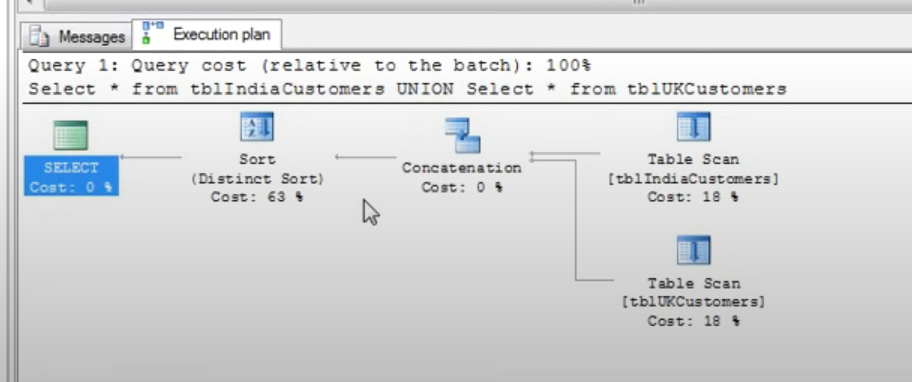
## Primary key vs unique key

## Features of SSMS

### Execution plan

Tool for understanding how SQL Server processes your queries. This can help optimize query performance by showing you how SQL Server accesses the data and combines it to produce the desired result

Ctrl+L is the shortcut to get this panel



### How to backup database in MS SQL

In SSMS, right click on the database and select Task-> Back up

## Various backup strategies

TODO:: Explain in detail

Full - Complete copy of entire database

Differential - A differential backup only includes the data that has changed since the last full backup. This means it's generally smaller and quicker to create than a full backup.

Transaction Log - Include the activities in the transaction log since the last log backup. This allows the recovery of a database to any point in time, including right up to the point of a failure.

## What is execution plan

An execution plan in SQL, also known as a query plan, is a sequence of steps used by a database management system to fetch data. It's essentially a roadmap that the database engine uses to gather the information necessary to fulfill a query.

Here are the key components and aspects of an SQL execution plan:

**Operations:** These are the individual actions the database takes to execute a query. Common operations include scans (like table scans and index scans), joins (like nested loops, hash joins, or merge joins), and sorts.

**Order of Operations:** The execution plan details the order in which these operations occur. This is crucial for understanding how a query will behave, especially with complex queries involving multiple joins and subqueries.

**Cost Estimation:** Each step in the execution plan comes with a cost estimate, which is typically based on factors like CPU usage, I/O operations, and network costs. The database uses these costs to determine the most efficient way to execute the query.

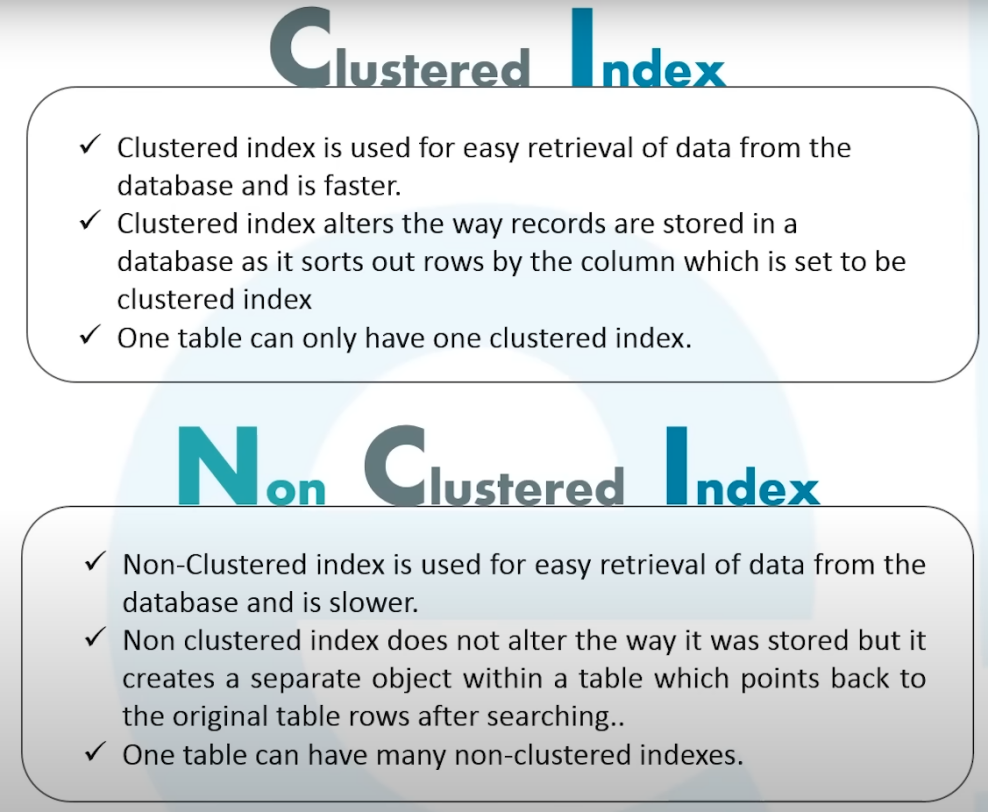
**Indexes Utilization:** The plan shows whether and how indexes are used. Using indexes can drastically reduce the amount of data that needs to be scanned, thereby improving performance.

**Parallel Execution:** For databases that support parallel processing, the execution plan will indicate if the query will be executed in parallel, which can speed up processing time significantly.

**Predicted Row Counts:** The plan estimates the number of rows that each operation will process, which helps in understanding the flow of data through the query.

To view an execution plan, most SQL databases offer built-in tools or commands, like EXPLAIN in PostgreSQL or MySQL, and EXPLAIN PLAN in Oracle SQL. These tools provide a visual or textual representation of the plan, which can be critical for optimizing and troubleshooting queries.

## Clustered vs non clustered



## Function vs stored procedure

| ****Aspect**** | ****Function**** | ****Stored Procedure**** |
| --- | --- | --- |
| ****Purpose**** | Primarily used to compute and return a value. | Used to perform a sequence of operations on the database. |
| ****Return Type**** | Must return a value (scalar or table). | May return zero, one, or multiple values; can also return record sets. |
| ****Invocation**** | Can be called from SQL queries (e.g., SELECT, WHERE clause). | Called using a specific CALL statement or EXEC command. |
| ****Operations**** | Generally **read-only**; used for calculations and data retrieval. | Can perform **read and write operations**; can manipulate database data. |
| ****Transaction Control**** | Usually does not handle transactions internally. | Can manage transactions (**BEGIN, COMMIT, ROLLBACK**). |
| ****Modularity**** | Good for encapsulating reusable logic or calculations. | Suitable for encapsulating business logic, which may involve multiple operations. |
| ****Parameter Passing**** | Only input parameters. | Can accept both input and output parameters. (nultiple) |
| ****Usage Context**** | Ideal for data transformation that does not affect database state. | Ideal for complex processes that might involve multiple steps and conditional logic. |
| ****Security**** | Can enhance security by providing limited data manipulation. | Can enforce security by controlling modifications to database data. |

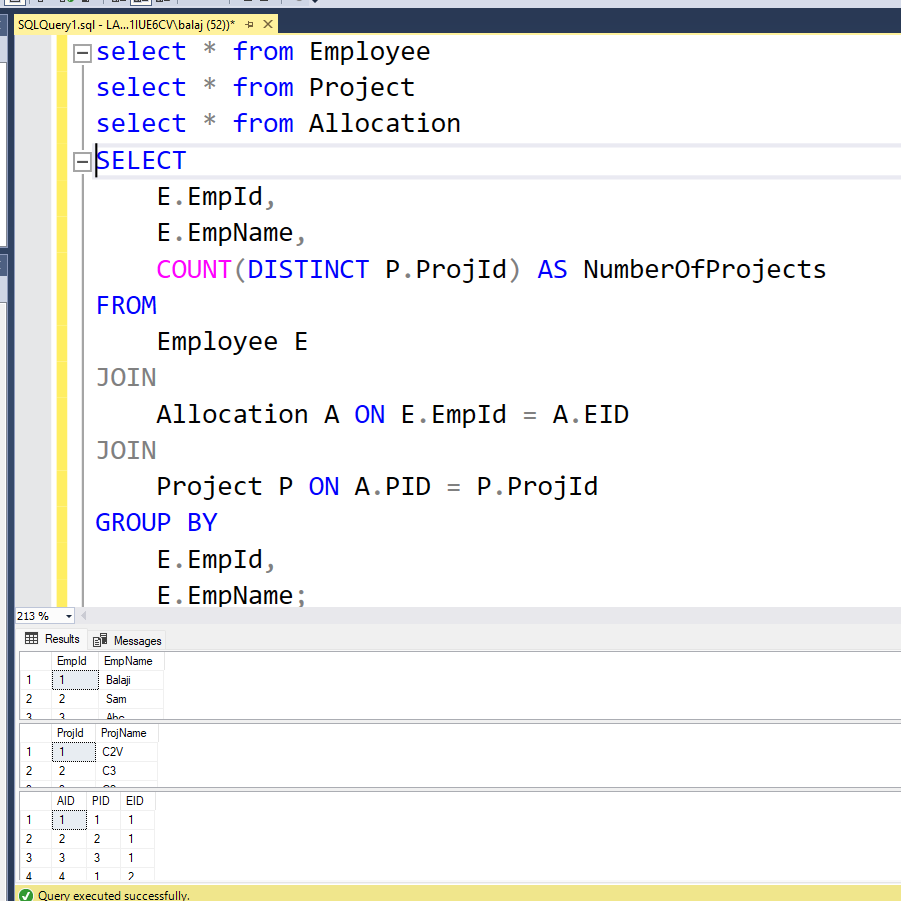
## What is table scan

If indexing is not implemented for tables then we need to iterate the entire table to find the matching element. This is said to be table scan.

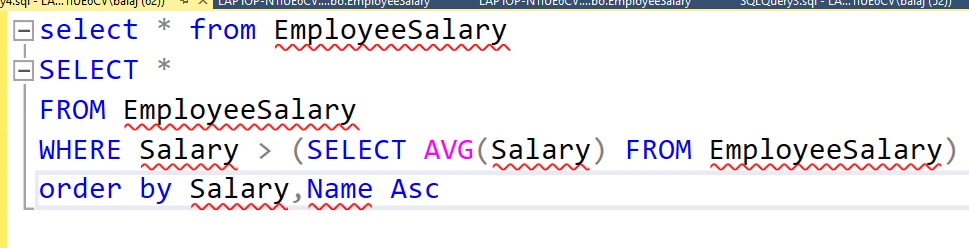
## Magic tables?

## Query: Nagarro

List all employee name with number of projects allocated for each employee



## Query: Employees greater than average salary



## Query: Take last five recent records from database

***SELECT TOP 5 \****

***FROM Products***

***ORDER BY Id DESC;***

## Query processing phases

From, On, Join, Where, Group by, withcube or with rollup, having, select, distinct, orderby, top, offset-fetch

## What is raiseerror function in MS SQL

TODO::: Update

## When to use begin and end keywords

**For all control flow statements,**

IF @x > @y

**BEGIN**

PRINT 'X is greater than Y';

UPDATE MyTable SET X = Y WHERE ID = 1;

**END**

ELSE

BEGIN

PRINT 'X is not greater than Y';

UPDATE MyTable SET Y = X WHERE ID = 1;

END

**For Stored procedure, user-defined functions and triggers**

## Query: Option to delete duplicates in a table with single column

PARTITION BY divides the query result set into partitions.

WITH EmployeesCTE AS

(

SELECT \*, ROW\_NUMBER()OVER(PARTITION BY ID ORDER BY ID) AS RowNumber

FROM Employees

)

DELETE FROM EmployeesCTE WHERE RowNumber > 1

## Query: If first n login is not successful make disable the user login for 30 minutes and enable it after 30 minutes

Check for highlighted comments in below SP first to get high level idea.

**CREATE TABLE** Users (

UserID INT PRIMARY KEY IDENTITY,

Username NVARCHAR(100) UNIQUE,

PasswordHash NVARCHAR(255),

IsLockedOut BIT DEFAULT 0,

LockoutEndTime DATETIME NULL

);

**CREATE TABLE** FailedLogins (

UserID INT,

AttemptTime DATETIME,

FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

**CREATE PROCEDURE** ValidateUserLogin

@Username NVARCHAR(100),

@Password NVARCHAR(100),

@Now DATETIME,

@MaxFailedAttempts INT = 3,

@LockoutDurationMinutes INT = 30

AS

BEGIN

DECLARE @UserID INT;

DECLARE @PasswordHash NVARCHAR(255);

DECLARE @IsLockedOut BIT;

DECLARE @LockoutEndTime DATETIME;

SELECT

@UserID = UserID,

@PasswordHash = PasswordHash,

@IsLockedOut = IsLockedOut,

@LockoutEndTime = LockoutEndTime

FROM Users

WHERE Username = @Username;

**-- If the user is found**

IF @UserID IS NOT NULL

BEGIN

**-- Check if the user is currently locked out**

IF @IsLockedOut = 1 AND @LockoutEndTime > @Now

BEGIN

SELECT 'Account is locked. Try again later.' AS Message;

RETURN;

END

**-- Validate password (hash verification)**

IF @Password = @PasswordHash

BEGIN

**-- Successful login: Reset failed login attempts and unlock the user**

DELETE FROM FailedLogins WHERE UserID = @UserID;

UPDATE Users

SET IsLockedOut = 0, LockoutEndTime = NULL

WHERE UserID = @UserID;

SELECT 'Login successful' AS Message;

END

ELSE

BEGIN

**-- Log failed attempt**

INSERT INTO FailedLogins (UserID, AttemptTime) VALUES (@UserID, @Now);

**-- Check how many attempts have been made in the last 30 minutes**

IF (

SELECT COUNT(\*)

FROM FailedLogins

WHERE UserID = @UserID AND AttemptTime > DATEADD(MINUTE, -@LockoutDurationMinutes, @Now)

) >= @MaxFailedAttempts

BEGIN

**-- Lock the account**

UPDATE Users

SET IsLockedOut = 1, LockoutEndTime = DATEADD(MINUTE, @LockoutDurationMinutes, @Now)

WHERE UserID = @UserID;

SELECT 'Account locked due to too many failed attempts. Try again later.' AS Message;

END

ELSE

BEGIN

SELECT 'Invalid username or password' AS Message;

END

END

END

ELSE

BEGIN

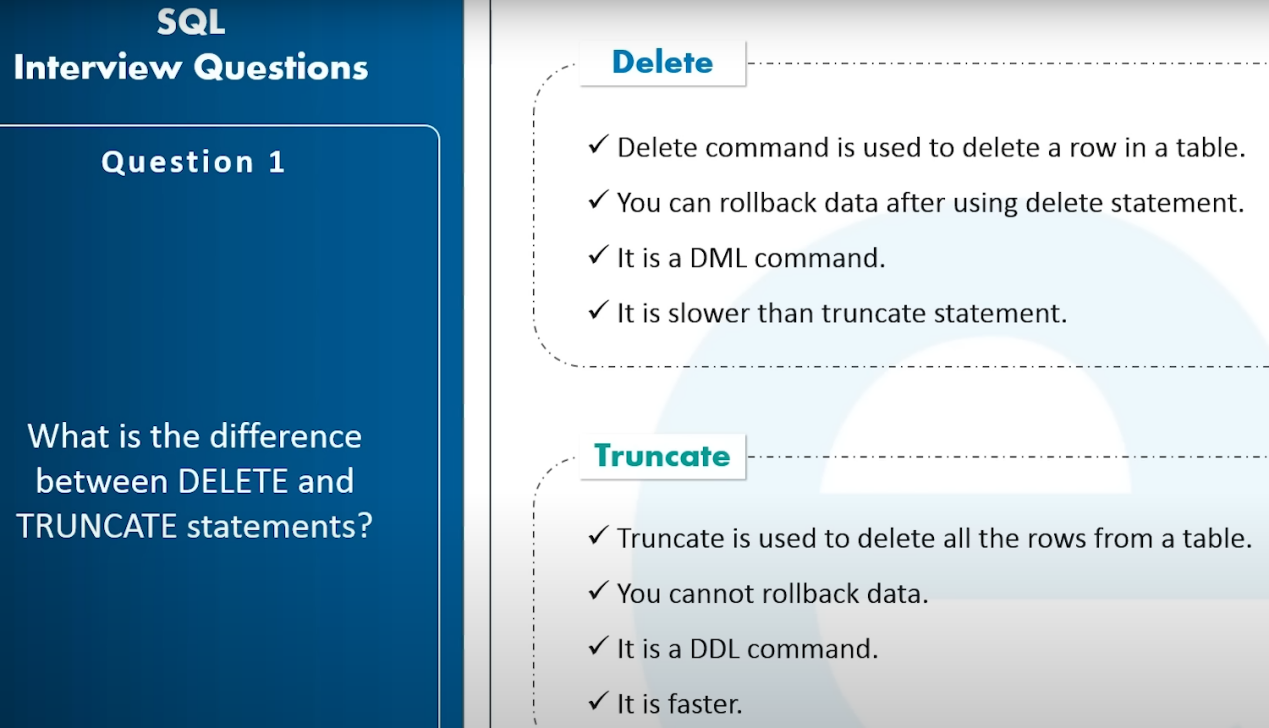
SELECT 'User not found' AS Message;

END

END

## Delete vs Truncate vs Drop

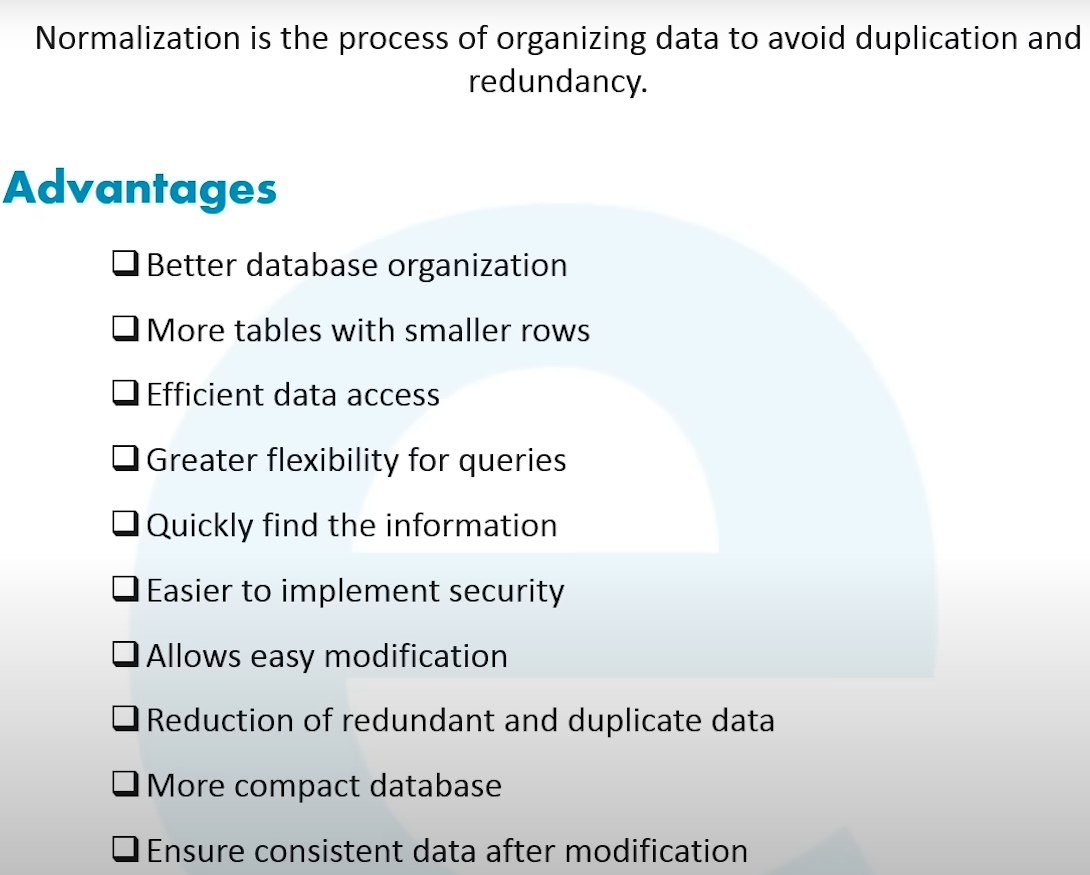
Drop: Removes a table and it cannot be rolled back from the database.



## What is Data integrity

Accuracy, consistency and integrity constraints to enforce business rules on data

## Normalization and Denormalization

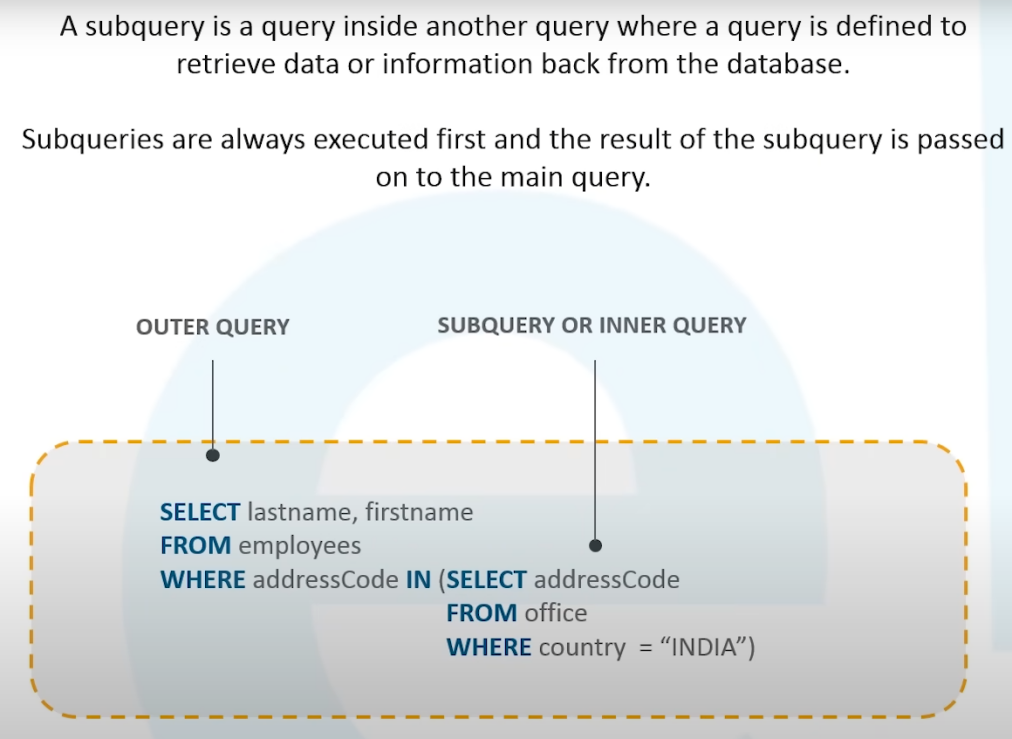


## Entities and Relationships

## What is ACID property in DB

Atomicity, Consistency, Isolation and Durability

## Subquery



## Explain CAP theorm

## Query: To get all column names of a table

**SELECT** COLUMN\_NAME

**FROM** INFORMATION\_SCHEMA.COLUMNS

**WHERE** TABLE\_SCHEMA = 'dbo' -- Replace 'dbo' with your schema name if different

**AND** TABLE\_NAME = 'Project'; -- Replace 'YourTableName' with your actual table name

## Kudvenkat

https://www.youtube.com/playlist?list=PL6n9fhu94yhXcztdLO7i6mdyaegC8CJwR

# React

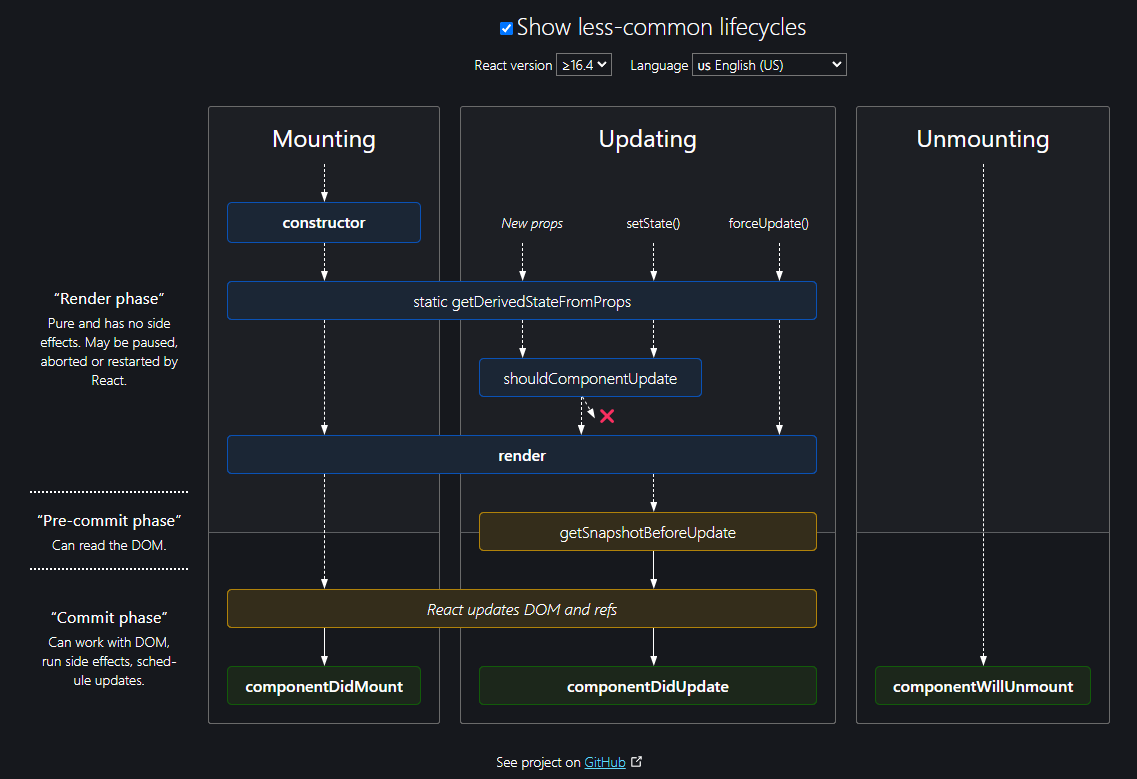
## Features of react?

## What is virtual DOM?

## Class vs functional component?

## Component lifecycle methods

Reference: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/



### Phases of components

#### Mounting

The component is ready to mount in the browser DOM. This phase covers initialization from constructor(), getDerivedStateFromProps(), render(), and componentDidMount() lifecycle methods.

#### Updating

#### Unmount

### React internal Phases

#### Render Phase

The component will render without any side effects. Applies to pure component.

#### Pre-Commit Phase

Before the component actually applies the changes to the DOM, there is a moment that allows React to read from the DOM through the ***getSnapshotBeforeUpdate().***

#### Commit Phase

React works with the DOM and executes the ***final lifecycles*** respectively ***componentDidMount()*** for mounting, ***componentDidUpdate()*** for updating, and ***componentWillUnmount()*** for unmounting.

### Constructor

### getDerivedStateFromProps

### Render

### ComponentDidMount

### ComponentDidUpdate

### ComponentWillUnmount

## Export vs export default

## Reconciliation?

Describes the process by which React updates the DOM based on changes in the component's state or props. The goal of reconciliation is to determine if and how the user interface should be updated in response to data changes.

Key Concepts in Reconciliation

1. **Elements of Different Types:**

If the elements have different types (e.g., from a <button> to an <a>), React will tear down the old tree and build the new tree from scratch, starting at that element.

1. **Elements of the Same Type:**

When comparing two elements of the same type, React keeps the underlying DOM node and only updates the changed attributes or properties. For instance, if the className of an element changes, React will only update the className attribute on the existing DOM node.

1. **Lists and Keys:**

When rendering lists, React uses keys to identify elements. Keys should be stable, predictable, and unique. By providing a unique key for each element, React can re-order, re-create, or remove elements efficiently during the reconciliation process.

## Diffing?

## Fiber tree?

## Hooks? Various hooks?

## Explain Usestate hook

## Explain Useeffect hook

## Explain Usecontext hook

## Write a custom hook

## Unmounting vs Re-rendering

## Flux architectural desgin pattern

"Flux" specifically refers to an architectural pattern used primarily for managing state within web applications. It was developed by Facebook alongside its React library. The Flux architecture promotes a unidirectional data flow, which makes the logic of complex web applications more predictable and easier to understand. Here’s how it works:

**Actions** - Payloads of information, sends data from application to flux store. Its source of information for the store, triggered by user interactions within the app.

**Dispatchers** - Central hub manages all data flow in a Flux application. It receives actions and dispatches them to the appropriate stores.

**Stores** - Stores contain the application's state and logic. They are responsible for managing the state that depends on the application, and they serve as the source of data for the UI components. Stores register with the dispatcher and listen for actions that they are interested in.

**Views** (React Components) - When a store changes, the views that depend on that store are notified. This causes the views to re-render with the new data, ensuring your application’s interface is consistent with the state of your application.

The Flux pattern helps in large applications by simplifying the data flow: changes are always initiated by actions, data flows in one direction, and changes are distributed through central dispatchers to various stores, which update the views. This unidirectional flow contrasts with more traditional two-way data binding techniques and has influenced the development of state management libraries like Redux, which is often used in modern web applications built with React.

# Javascript

## Promise

## Closure

## Call, bind and apply

## Debounce and throttling

## References

<https://github.com/sudheerj>

# General

## Explain about your current project

About wireless charging - react - web api - asp net core(started with version 3 now updgraded to 7) - ado.net - for DB - MS SQL DB

Design patterns

Web api - repository, decorator

Ado.net - factory pattern for SQLConnection and SQLCommand