[OOP 3](#_Toc194)

[Inheritance 3](#_Toc5854)

[Polymorphism 3](#_Toc2484)

[Encapsulation 3](#_Toc20774)

[Abstraction 3](#_Toc7330)

[MVC and Web API 3](#_Toc4008)

[CORS? 3](#_Toc17752)

[Global asax? 3](#_Toc2672)

[Controller action return types in ASP.NET Core web API 3](#_Toc4441)

[Transient vs scoped vs singleton 4](#_Toc8189)

[Version history 5](#_Toc22569)

[For dotnet framework 5](#_Toc22737)

[For dotnet core 5](#_Toc6917)

[Views? 6](#_Toc2262)

[Partial Views? 6](#_Toc24395)

[What is view engine 6](#_Toc1189)

[IActionResult? 6](#_Toc16664)

[Ways to pass data to views? 6](#_Toc21032)

[Strongly vs weakly types models 6](#_Toc10220)

[Strongly typed - viewmodel 6](#_Toc14132)

[Weakly typed 7](#_Toc5457)

[Viewdata attribute? 8](#_Toc24076)

[View data vs view bag 9](#_Toc1495)

[Razor pages 10](#_Toc18159)

[Model Binding 10](#_Toc24299)

[Can we use http get to delete the record by passing an id? 10](#_Toc14110)

[References 10](#_Toc30790)

[C# 10](#_Toc6391)

[Func vs Predicate vs Action vs Lambda 10](#_Toc1556)

[Const vs Readonly 10](#_Toc7714)

[Async await? 11](#_Toc4344)

[TAP - Task based asynchronous programming? 11](#_Toc15327)

[Assembly info file? 11](#_Toc21408)

[Process vs thread vs task? 11](#_Toc6894)

[Message queues 11](#_Toc20868)

[LINQ 11](#_Toc8115)

[LINQ Keywords with samples 11](#_Toc22783)

[Deferred vs Immediate execution 11](#_Toc5252)

[IEnumerable and IQueryable 12](#_Toc29421)

[Is and as keywords 13](#_Toc3897)

[First, firstordefault, single 13](#_Toc5228)

[Async parallelism 13](#_Toc13438)

[Instance for abstract class cannot be created. What is that we are achieving by doing this? 13](#_Toc19318)

[Select vs SelectMany 13](#_Toc20086)

[Include and theninclude 14](#_Toc9576)

[How to pass data to base class parameterized constructor 14](#_Toc6575)

[Foreground vs background threads 14](#_Toc18104)

[IComparable vs IComparer 14](#_Toc628)

[Print Duplicate numbers without looping 14](#_Toc9575)

[Print 1000 times without looping 15](#_Toc7687)

[Dotnet framework and core 15](#_Toc9201)

[Dotnet core Features? 15](#_Toc17987)

[Dotnet Framework features? 15](#_Toc30921)

[Deferred execution? 15](#_Toc13239)

[Web 15](#_Toc28319)

[CDN? 15](#_Toc4414)

[Rest Architecture 15](#_Toc11275)

[Serverless architecture? 15](#_Toc7574)

[What is SSL/TLS? 15](#_Toc25934)

[Public key and private key 15](#_Toc21135)

[Symmetric and Asymmetric cryptography 16](#_Toc12607)

[Http vs Https? 16](#_Toc1731)

[Encryption algorithms? 16](#_Toc17070)

[Web sockets? 16](#_Toc26102)

[Filters in web api? 16](#_Toc4537)

[Various http codes with explanation 16](#_Toc15756)

[Database 16](#_Toc7400)

[Versions of SQL server 16](#_Toc17862)

[Indexing? 17](#_Toc26636)

[Various indexing strategies? 17](#_Toc5519)

[Clustered vs non clustered index? 17](#_Toc20274)

[Normalization? 17](#_Toc7728)

[Types? 18](#_Toc13877)

[Stored procedures? 18](#_Toc13599)

[How to optimize and points to be considered while implementing stored procedures? 18](#_Toc8610)

[SQL vs No SQL 18](#_Toc9667)

[Cursor? 18](#_Toc27669)

[Constraints? 18](#_Toc20970)

[Triggers? 18](#_Toc9023)

[Advantages of triggers 18](#_Toc24372)

[Transaction? 18](#_Toc12837)

[DBCC Commands 18](#_Toc8408)

[Primary key vs unique key 18](#_Toc27230)

[Features of SSMS 18](#_Toc10331)

[Execution plan 18](#_Toc19557)

[How to backup database in MS SQL 19](#_Toc32054)

[Various backup strategies 19](#_Toc12076)

[What is execution plan 19](#_Toc14975)

[Clustered vs non clustered 20](#_Toc10645)

[Function vs stored procedure 20](#_Toc2835)

[What is table scan 21](#_Toc29831)

[Magic tables? 21](#_Toc32723)

[Query: Nagarro 21](#_Toc29352)

[Query: Employees greater than average salary 21](#_Toc22762)

[Query: Take last five recent records from database 21](#_Toc13074)

[Query processing phases 22](#_Toc25152)

[What is raiseerror function in MS SQL 22](#_Toc11894)

[When to use begin and end keywords 22](#_Toc17538)

[Query: Option to delete duplicates in a table with single column 22](#_Toc14984)

[Query: If first n login is not successful make disable the user login for 30 minutes and enable it after 30 minutes 22](#_Toc30565)

[Delete vs Truncate vs Drop 24](#_Toc19889)

[What is Data integrity 24](#_Toc10003)

[Normalization and Denormalization 25](#_Toc15696)

[Entities and Relationships 25](#_Toc11741)

[What is ACID property in DB 25](#_Toc13082)

[Subquery 26](#_Toc3242)

[React 26](#_Toc14930)

[Features of react? 26](#_Toc24512)

[What is virtual DOM? 26](#_Toc3060)

[Class vs functional component? 26](#_Toc31438)

[Component lifecycle methods 26](#_Toc14602)

[Phases of components 28](#_Toc28268)

[Mounting 28](#_Toc3257)

[Updating 28](#_Toc2660)

[Unmount 28](#_Toc3034)

[React internal Phases 28](#_Toc19333)

[Render Phase 28](#_Toc26125)

[Pre-Commit Phase 28](#_Toc22736)

[Commit Phase 28](#_Toc3237)

[Constructor 28](#_Toc4922)

[getDerivedStateFromProps 28](#_Toc1520)

[Render 28](#_Toc5213)

[ComponentDidMount 28](#_Toc1945)

[ComponentDidUpdate 28](#_Toc7114)

[ComponentWillUnmount 28](#_Toc31031)

[Export vs export default 28](#_Toc10207)

[Reconciliation? 28](#_Toc18508)

[Diffing? 29](#_Toc7751)

[Fiber tree? 29](#_Toc317)

[Hooks? Various hooks? 29](#_Toc18839)

[Explain Usestate hook 29](#_Toc32011)

[Explain Useeffect hook 29](#_Toc23707)

[Explain Usecontext hook 29](#_Toc12758)

[Write a custom hook 29](#_Toc29014)

[Unmounting vs Re-rendering 29](#_Toc25711)

[Flux architectural desgin pattern 29](#_Toc26824)

[Javascript 30](#_Toc11140)

[Promise 30](#_Toc31205)

[Closure 30](#_Toc31024)

[Call, bind and apply 30](#_Toc29044)

[Debounce and throttling 30](#_Toc17902)

[References 30](#_Toc29093)

# 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

## 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

# MVC and Web API

## CORS?

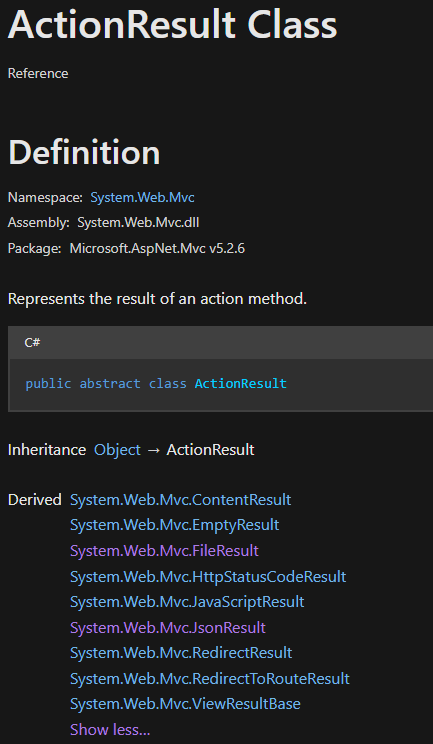
## Global asax?

## 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

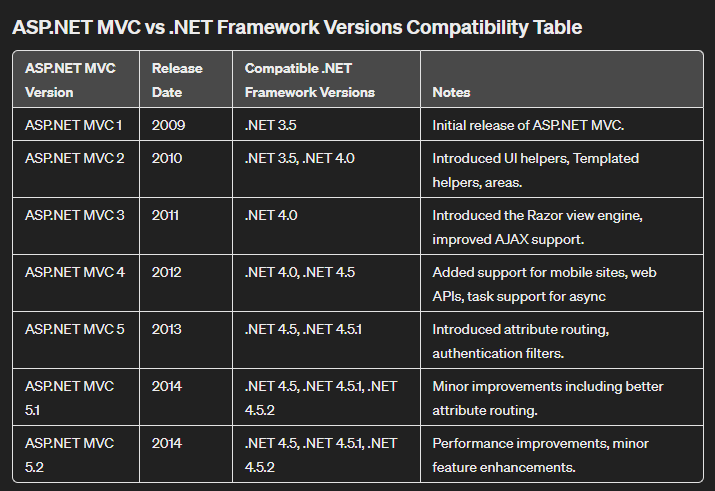
Use cases for each

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.

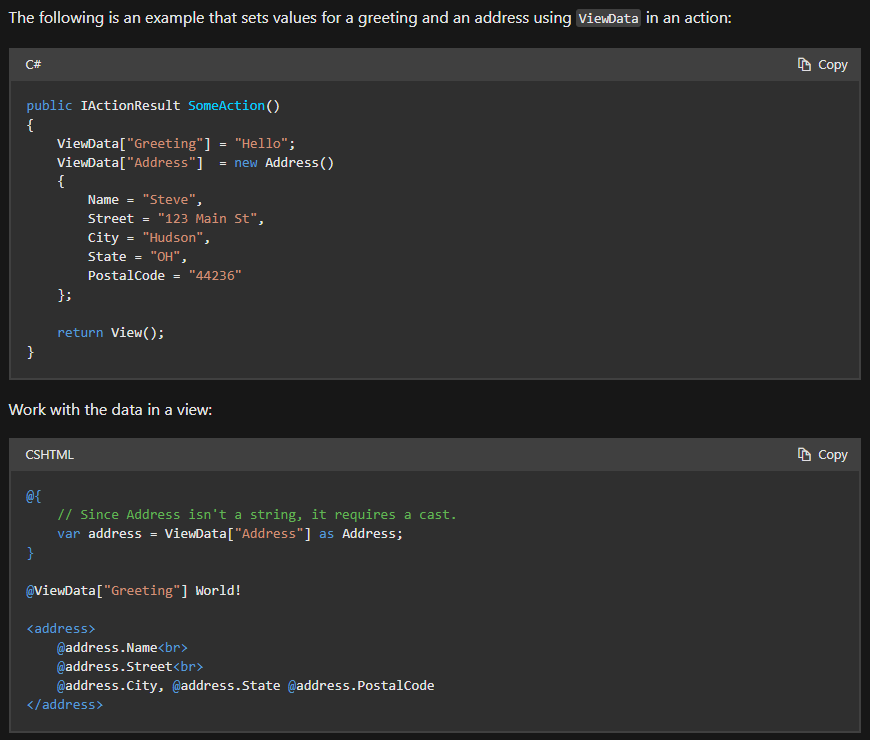
***SS Explanation: Address is a viewmodel. Passed as a parameter for View from contact controller.***



### Weakly typed

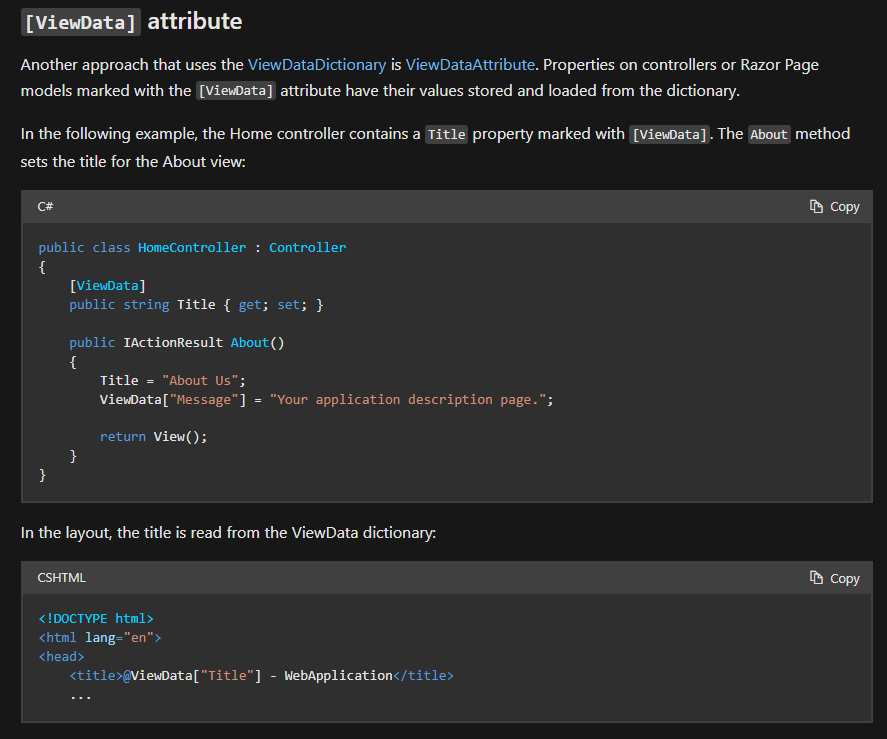
The **ViewData** property is a **dictionary** of weakly typed objects. The **ViewBag** property is a **wrapper around ViewData** that provides dynamic properties for the underlying ViewData collection.

Both are dynamically resolved at runtime.

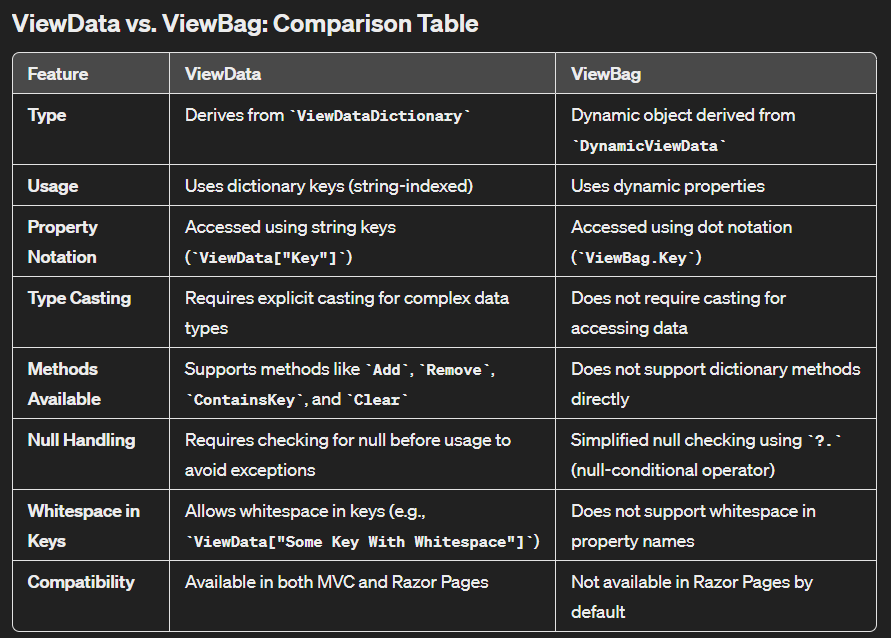


### Viewdata attribute?

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

## References

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

# C#

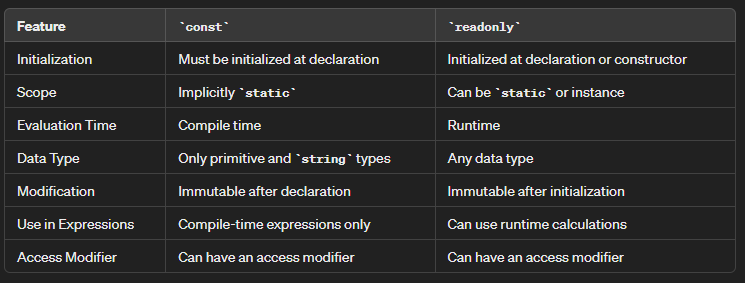
### Func vs Predicate vs Action vs Lambda

### 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

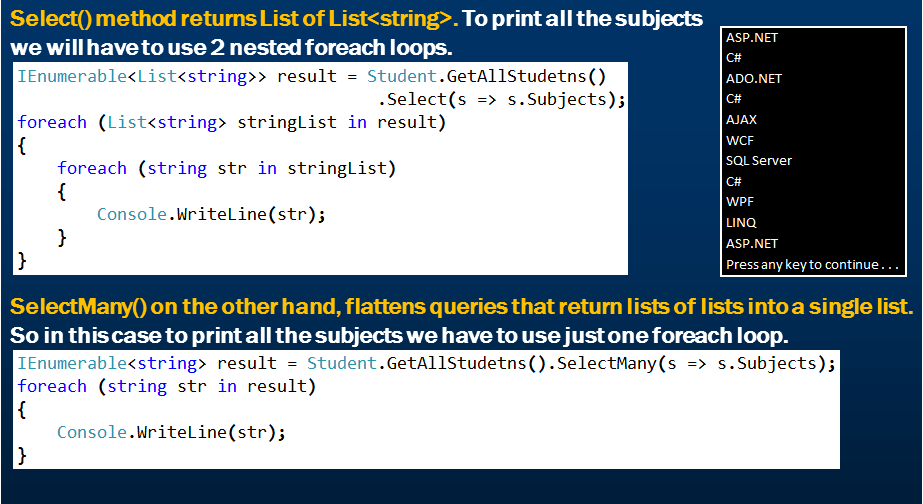
### Async parallelism

### 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.

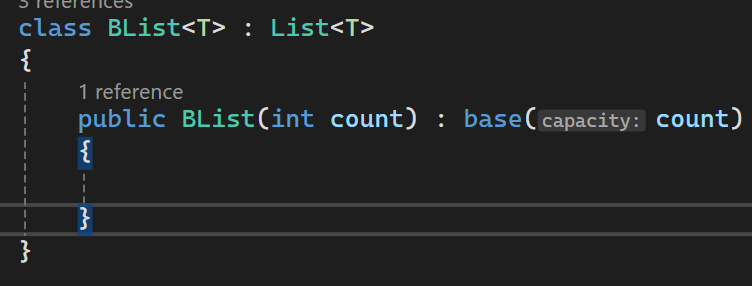
**SelectMany:** Projects each element of a sequence to an IEnumerable<T> and **flattens** the resulting sequences into one sequence.



### Include and theninclude

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

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

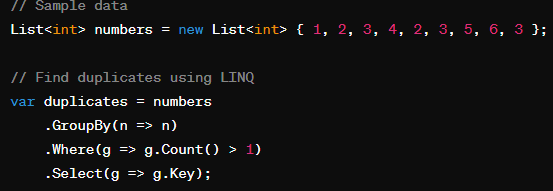


### Foreground vs background threads

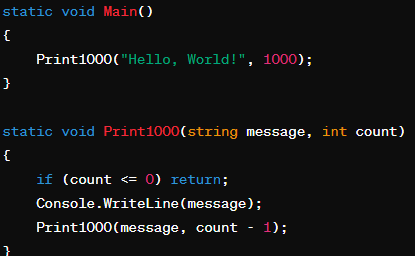
### IComparable vs IComparer

### Print Duplicate numbers without looping

TODO:: Need to run and check



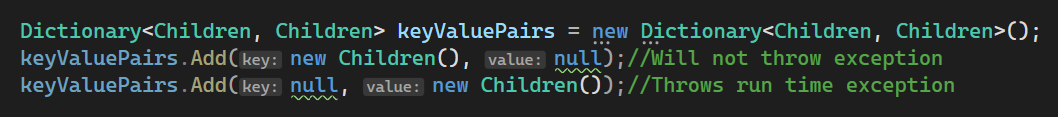
### Print 1000 times without looping



### 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

### Equals vs ReferenceEquals

### IComparable, IEqualityComparer, Equals

# Dotnet framework and core

## Dotnet core Features?

## Dotnet Framework features?

## Deferred execution?

# 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?

## SQL vs No SQL

## 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

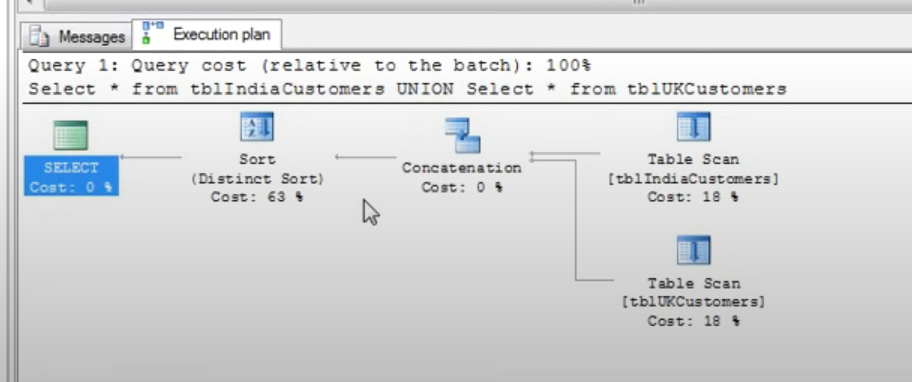
## Primary key vs unique key

## 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.

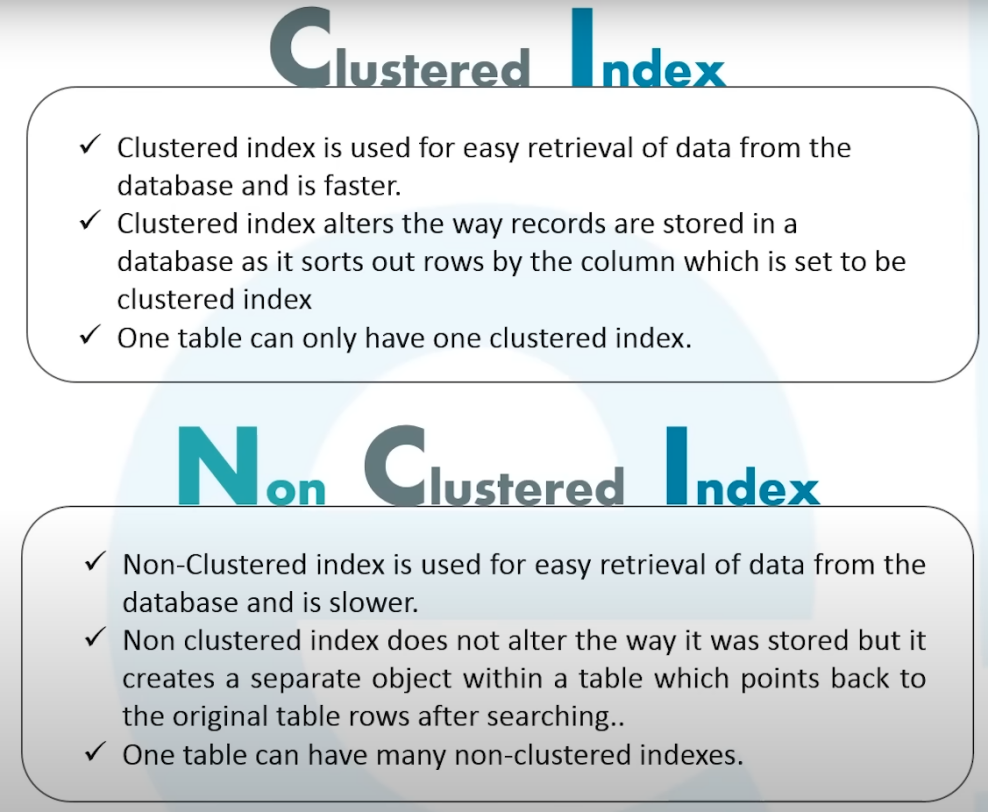
**Indexes Utilization:** The plan shows whether and how indexes are used. Using indexes can drastically reduce the amount of data that needs to be scanned, thereby improving performance.

**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. |

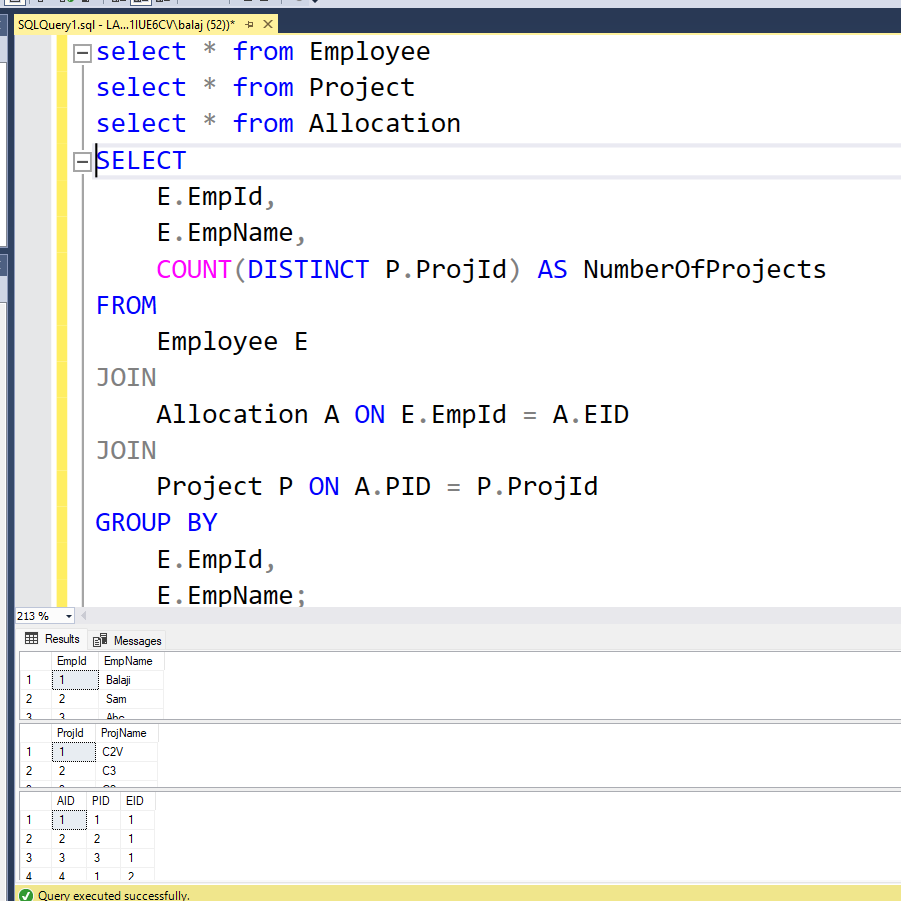
## What is table scan

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.

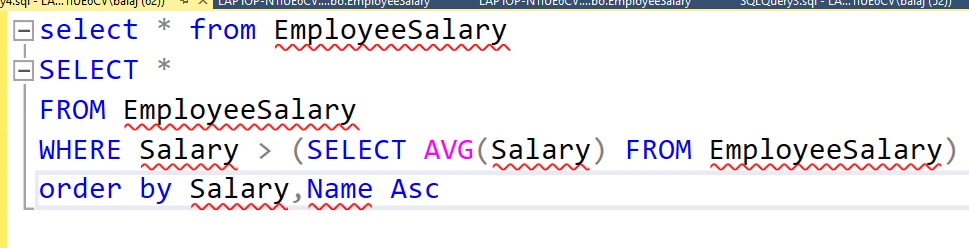
## Magic tables?

## Query: Nagarro

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



## 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

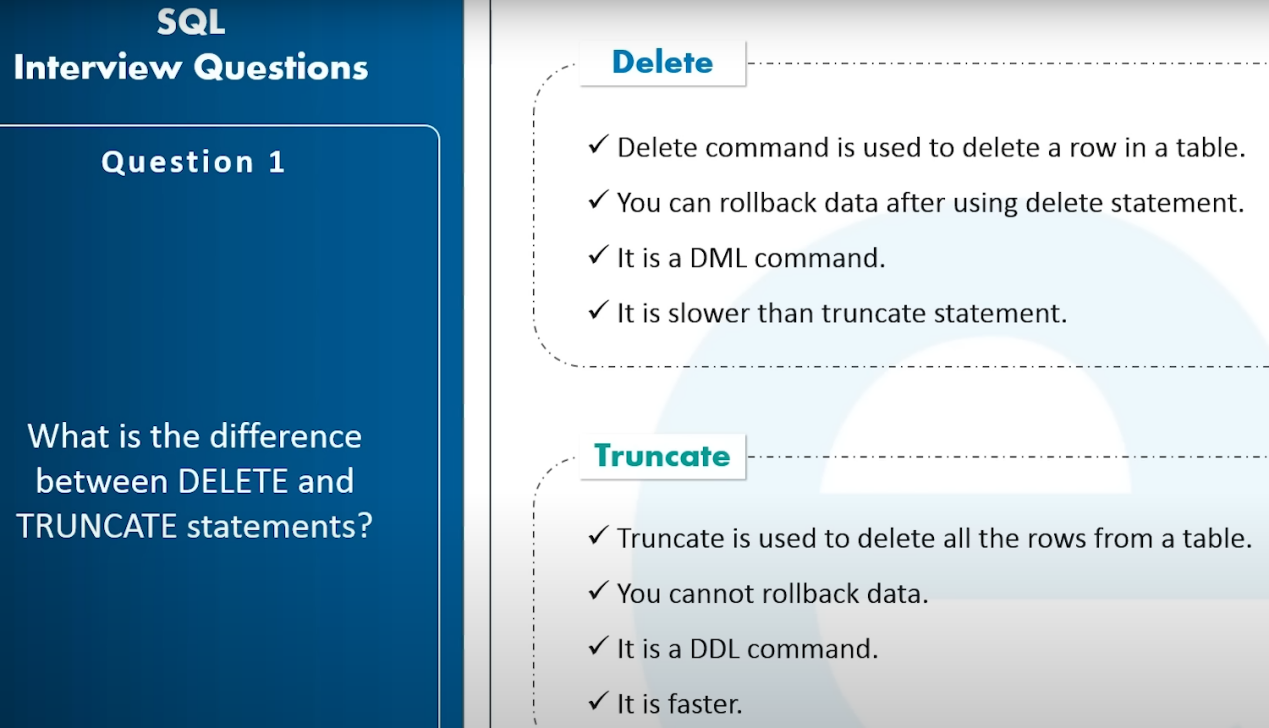
SELECT 'User not found' AS Message;

END

END

## Delete vs Truncate vs Drop

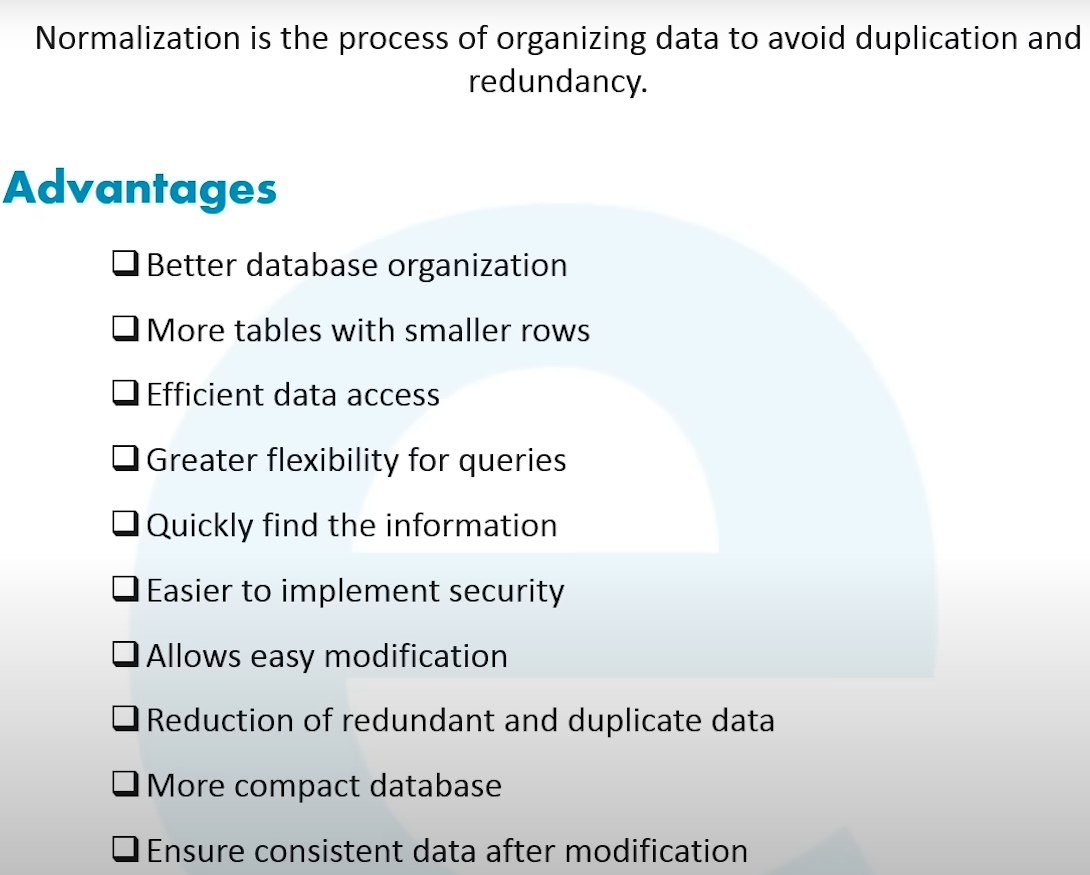
Drop: Removes a table and it cannot be rolled back from the database.



## What is Data integrity

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

## Normalization and Denormalization

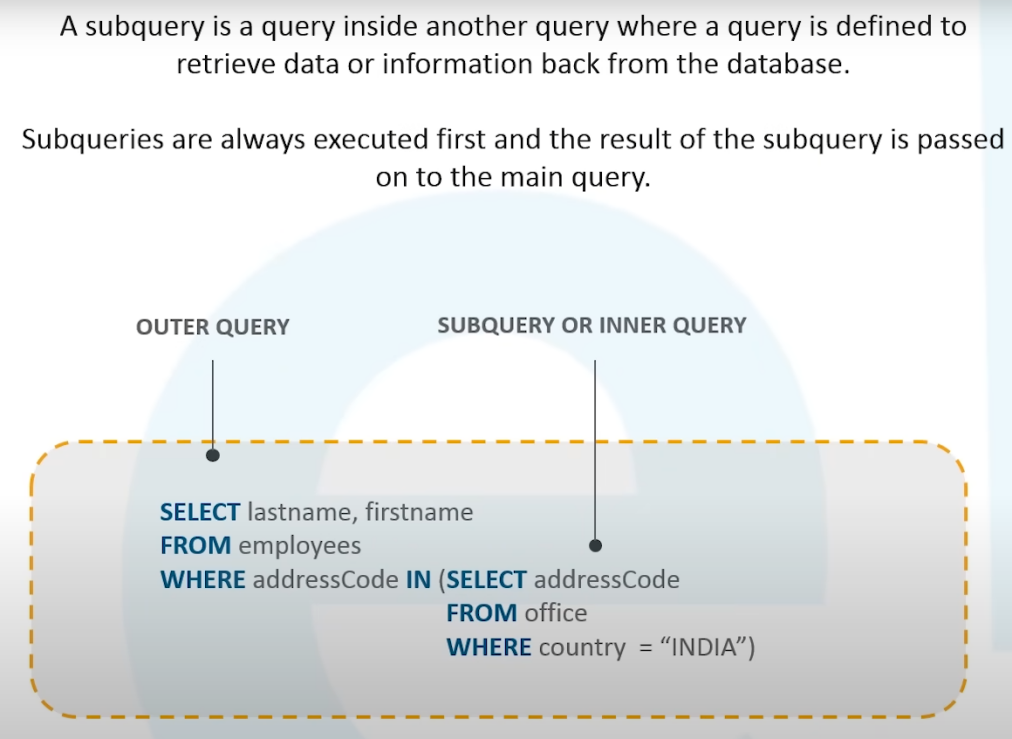


## Entities and Relationships

## What is ACID property in DB

Atomicity, Consistency, Isolation and Durability

## Subquery



# React

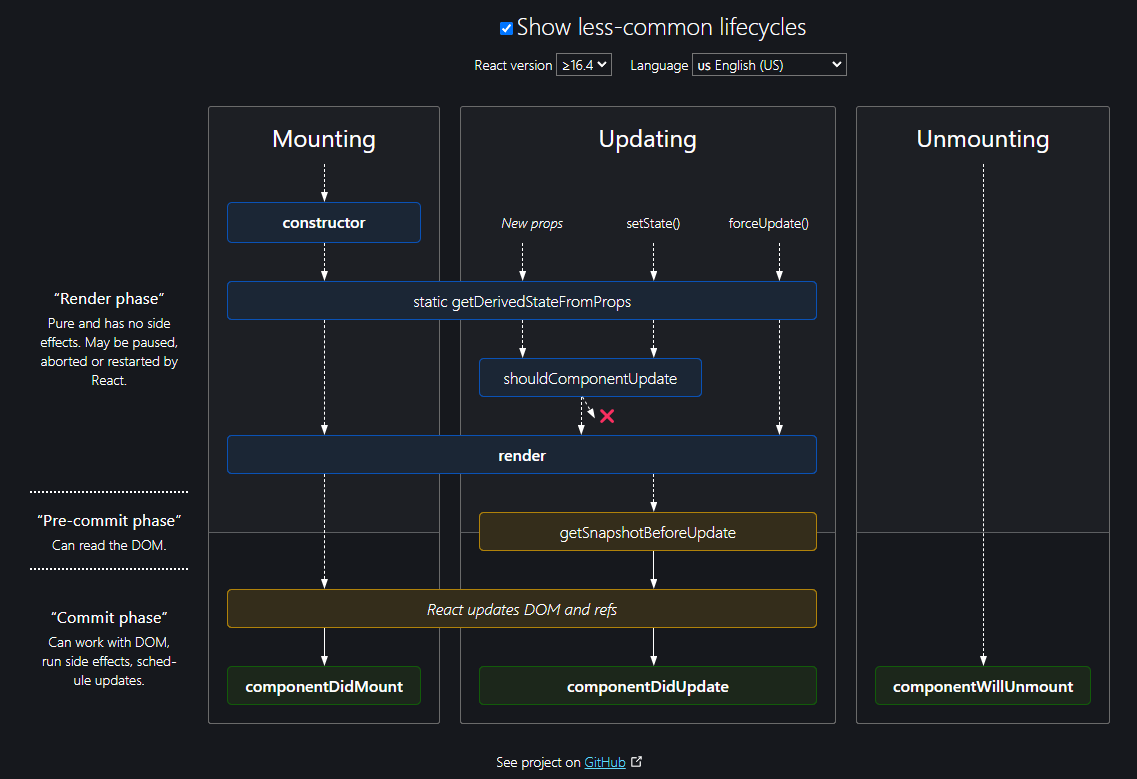
## Features of react?

## 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