[OOP 3](#_Toc19905)

[Inheritance 3](#_Toc27836)

[Polymorphism 3](#_Toc13389)

[Encapsulation 3](#_Toc9347)

[Abstraction 3](#_Toc8662)

[MVC 3](#_Toc6125)

[Version history 3](#_Toc9125)

[For dotnet framework 3](#_Toc19953)

[For dotnet core 3](#_Toc24677)

[Views? 4](#_Toc18418)

[Partial Views? 4](#_Toc31682)

[What is view engine 4](#_Toc2905)

[IActionResult? 4](#_Toc23595)

[Ways to pass data to views? 4](#_Toc10466)

[Strongly vs weakly types models 4](#_Toc2920)

[Strongly typed - viewmodel 5](#_Toc14145)

[Weakly typed 5](#_Toc28967)

[Viewdata attribute? 6](#_Toc9234)

[View data vs view bag 7](#_Toc24238)

[Razor pages 8](#_Toc5426)

[References 8](#_Toc23886)

[C# 8](#_Toc9712)

[Async await? 8](#_Toc32084)

[TAP - Task based asynchronous programming? 8](#_Toc26299)

[Assembly info file? 8](#_Toc4481)

[Process vs thread vs task? 8](#_Toc31535)

[Message queues 8](#_Toc25430)

[LINQ 8](#_Toc20920)

[LINQ Keywords with samples 8](#_Toc3772)

[IEnumerable and IQueryable 8](#_Toc2341)

[Is and as keywords 9](#_Toc26561)

[First, firstordefault, single 9](#_Toc31701)

[Async parallelism 9](#_Toc9104)

[Instance for abstract class cannot be created. What is that we are achieving by doing this? 9](#_Toc25969)

[Select vs SelectMany 9](#_Toc1478)

[Include and theninclude 9](#_Toc32589)

[Const vs readonly 9](#_Toc7635)

[How to pass data to base class parameterized constructor 10](#_Toc4320)

[Dotnet framework and core 10](#_Toc15373)

[Dotnet core Features? 10](#_Toc7416)

[Dotnet Framework features? 10](#_Toc17558)

[Deferred execution? 10](#_Toc22930)

[Asp Net 10](#_Toc30284)

[Controller action return types in ASP.NET Core web API 10](#_Toc20356)

[Transient vs scoped vs singleton 11](#_Toc11863)

[Web 11](#_Toc10328)

[CDN? 11](#_Toc13591)

[Rest Architecture 11](#_Toc9218)

[Serverless architecture? 11](#_Toc18296)

[What is SSL/TLS? 11](#_Toc15713)

[Public key and private key 11](#_Toc26618)

[Symmetric and Asymmetric cryptography 11](#_Toc31653)

[Http vs Https? 11](#_Toc7408)

[Encryption algorithms? 12](#_Toc8214)

[Web sockets? 12](#_Toc11717)

[Filters in web api? 12](#_Toc6855)

[Various http codes with explanation 12](#_Toc16193)

[Database 12](#_Toc17986)

[Versions of SQL server 12](#_Toc28455)

[Indexing? 12](#_Toc10541)

[Various indexing strategies? 12](#_Toc20906)

[Normalization? 12](#_Toc13942)

[Types? 13](#_Toc1918)

[Stored procedures? 13](#_Toc1932)

[SQL vs No SQL 13](#_Toc6557)

[Cursor? 13](#_Toc10668)

[Constraints? 13](#_Toc28654)

[Triggers? 13](#_Toc7756)

[Advantages of triggers 13](#_Toc8819)

[Transaction? 13](#_Toc28714)

[DBCC Commands 13](#_Toc8368)

[Primary key vs unique key 13](#_Toc2443)

[Features of SSMS 13](#_Toc18783)

[Execution plan 13](#_Toc7593)

[How to backup database in MS SQL 14](#_Toc7341)

[Various backup strategies 14](#_Toc18021)

[What is execution plan 14](#_Toc17824)

[Clustered vs non clustered 15](#_Toc16567)

[Function vs stored procedure 15](#_Toc23372)

[What is table scan 15](#_Toc17779)

[Magic tables? 15](#_Toc251)

[Query: Nagarro 15](#_Toc10586)

[React 16](#_Toc1160)

[Features of react? 16](#_Toc1681)

[What is virtual DOM? 16](#_Toc21609)

[Class vs functional component? 16](#_Toc758)

[Component lifecycle methods 16](#_Toc13125)

[Constructor 17](#_Toc8677)

[getDerivedStateFromProps 17](#_Toc32013)

[Render 17](#_Toc17776)

[ComponentDidMount 17](#_Toc12772)

[Export vs export default 17](#_Toc25117)

[Reconciliation? 17](#_Toc23912)

[Diffing? 18](#_Toc16788)

[Fiber tree? 18](#_Toc25877)

[Hooks? Various hooks? 18](#_Toc21495)

[Explain Usestate hook 18](#_Toc24272)

[Explain Useeffect hook 18](#_Toc19636)

[Explain Usecontext hook 18](#_Toc22764)

[Write a custom hook 18](#_Toc28423)

[Unmounting vs Re-rendering 18](#_Toc1822)

[Javascript 18](#_Toc20370)

[Closure 18](#_Toc9272)

[Call, bind and apply 18](#_Toc15285)

[Debounce and throttling 18](#_Toc5800)

[References 18](#_Toc15489)

# OOP

## Inheritance

## Polymorphism

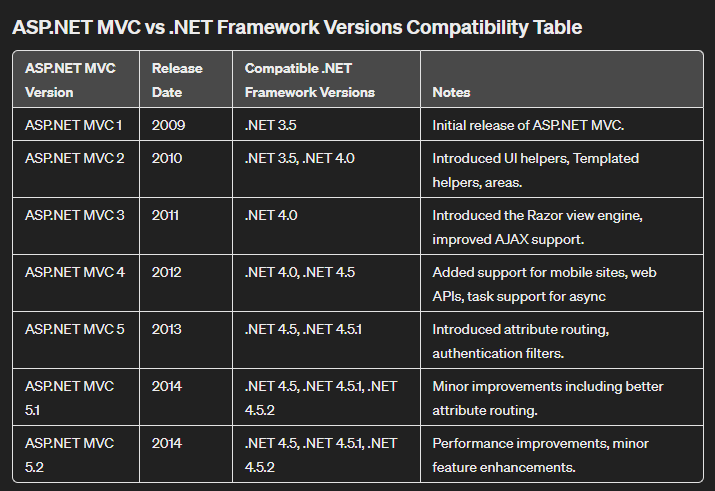
## Encapsulation

## Abstraction

# MVC

### 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.

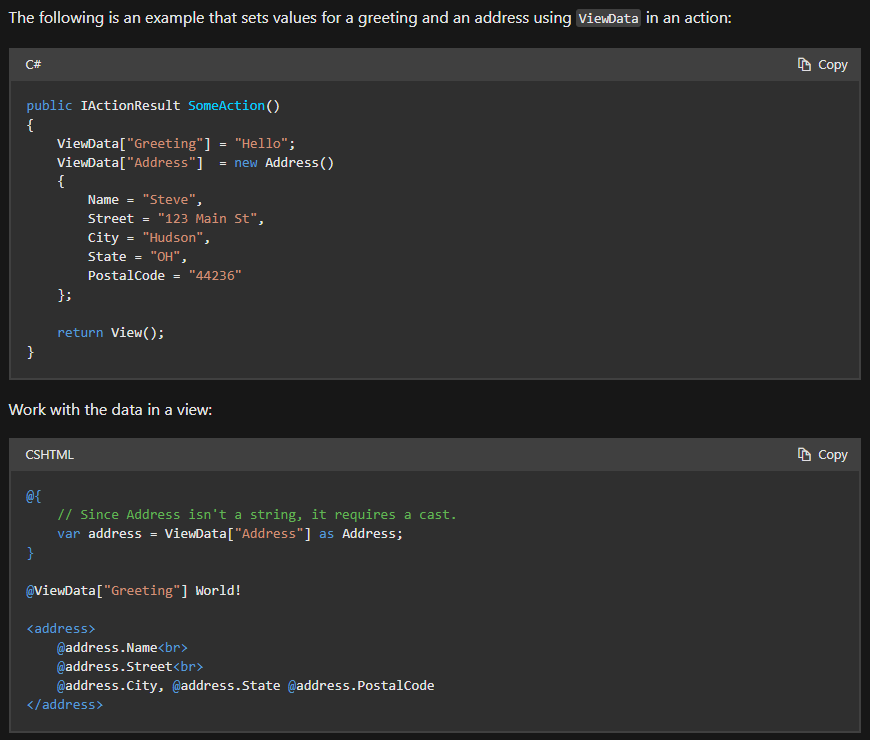
***SS Explanation: Address is a viewmodel. Passed as a parameter for View from contact controller.***



#### Weakly typed

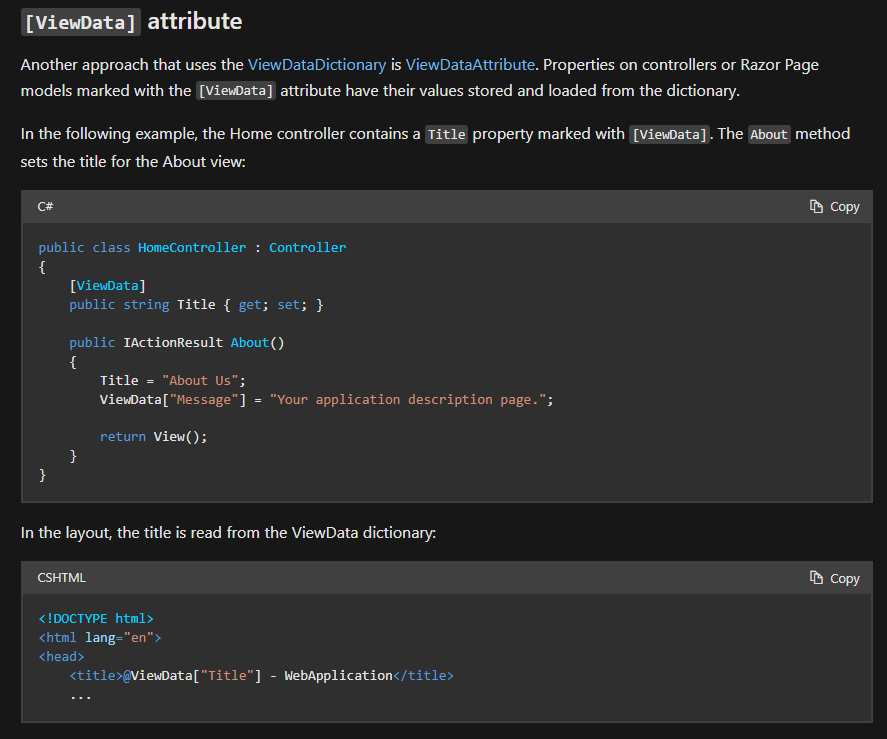
The **ViewData** property is a **dictionary** of weakly typed objects. The **ViewBag** property is a **wrapper around ViewData** that provides dynamic properties for the underlying ViewData collection.

Both are dynamically resolved at runtime.

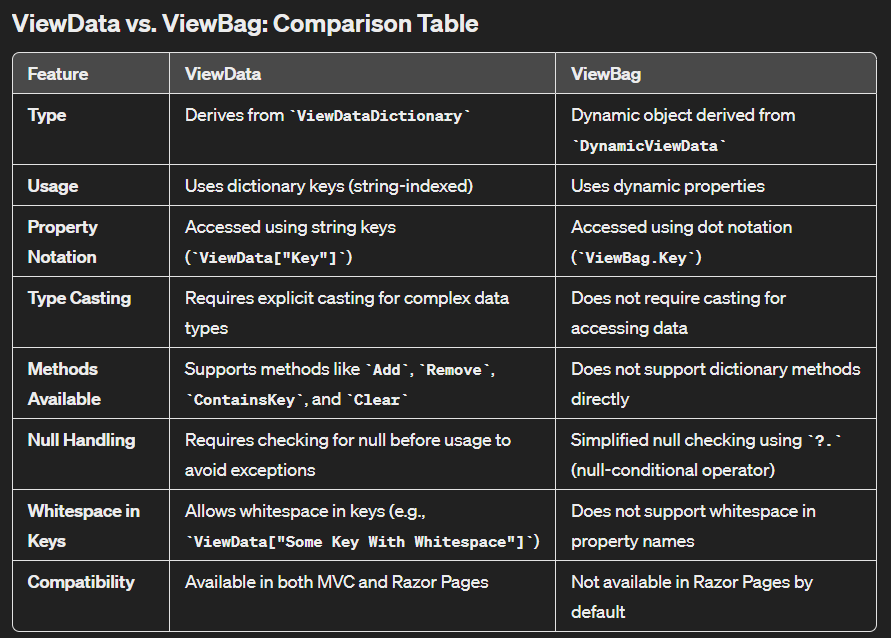


#### Viewdata attribute?

Add this attribute in controller and use it in view.



#### View data vs view bag



### Razor pages

## References

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

# C#

### 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

### 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

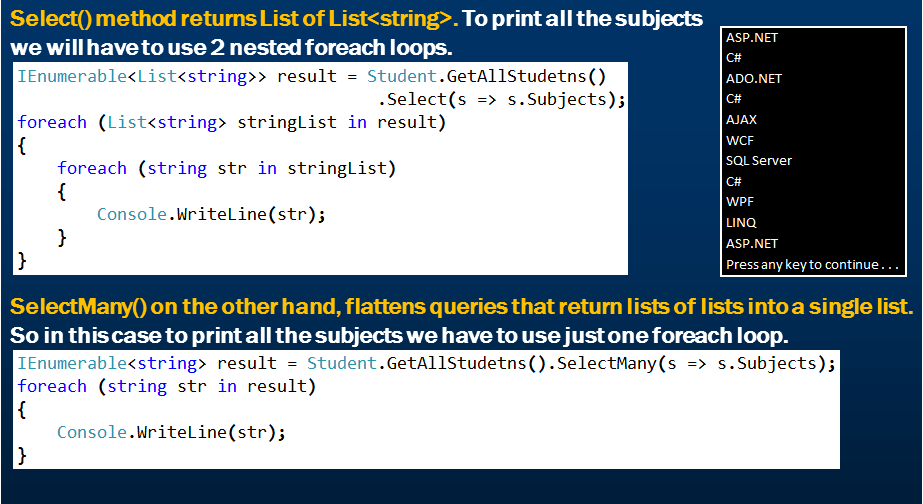
### Async parallelism

### 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

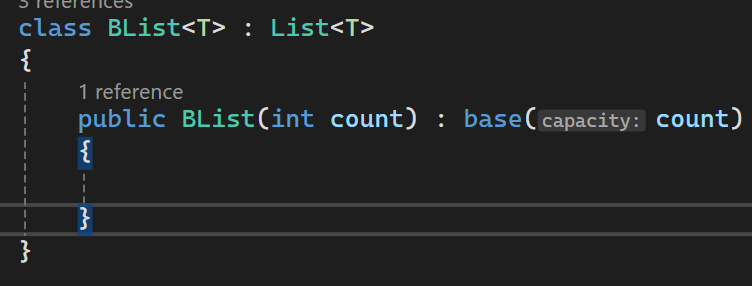
### Const vs readonly

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 |

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

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



# Dotnet framework and core

## Dotnet core Features?

## Dotnet Framework features?

## Deferred execution?

# Asp Net

## 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. IAction Result
3. ActionsResult<T>
4. HttpResults

## Transient vs scoped vs singleton

# 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?

## 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?

## 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

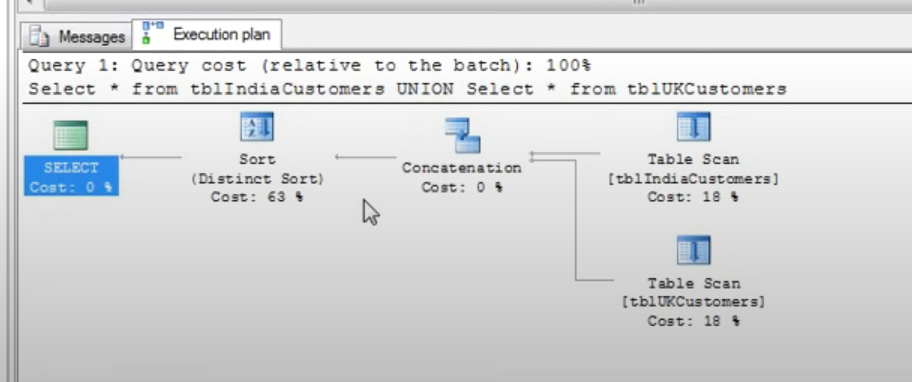
## Primary key vs unique key

## 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

TODO::

## Function vs stored procedure

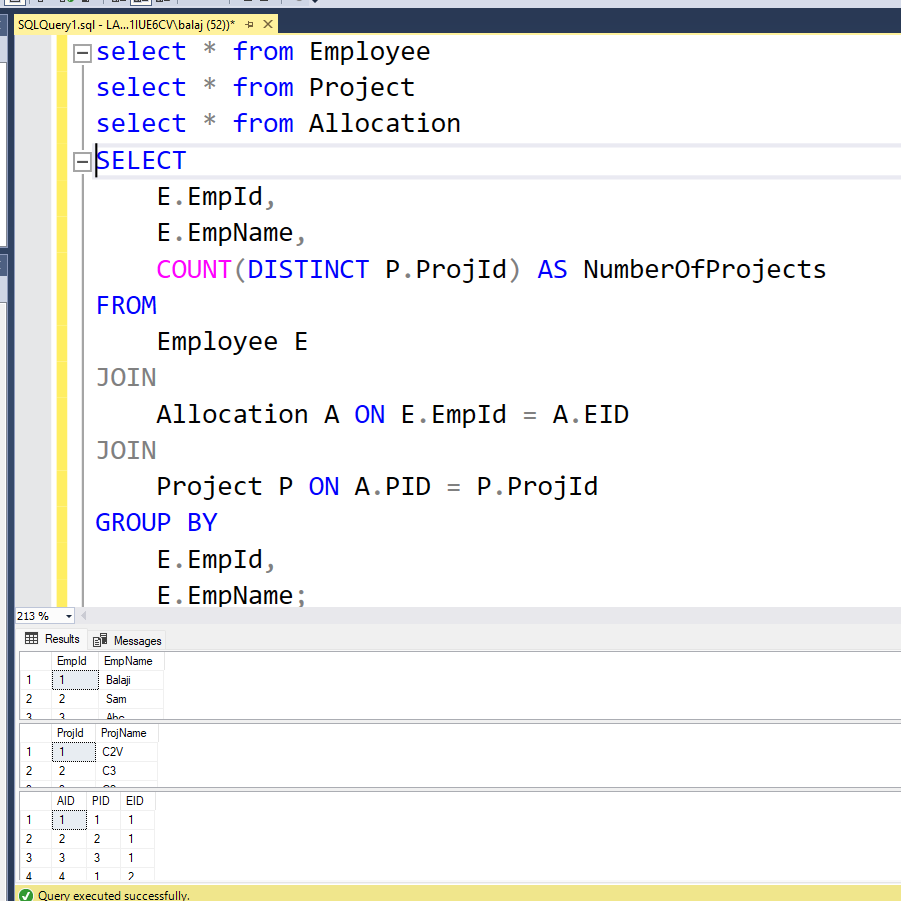
| ****Aspect**** | ****Function**** | ****Stored Procedure**** |
| --- | --- | --- |
| ****Purpose**** | Primarily used to compute and return a value. | Used to perform a sequence of operations on the database. |
| ****Return Type**** | Must return a value (scalar or table). | May return zero, one, or 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**** | Can accept parameters but usually limited to input parameters. | Can accept both input and output parameters. |
| ****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

## Magic tables?

## Query: Nagarro

List all employee with number of projects allocated for each employee



# React

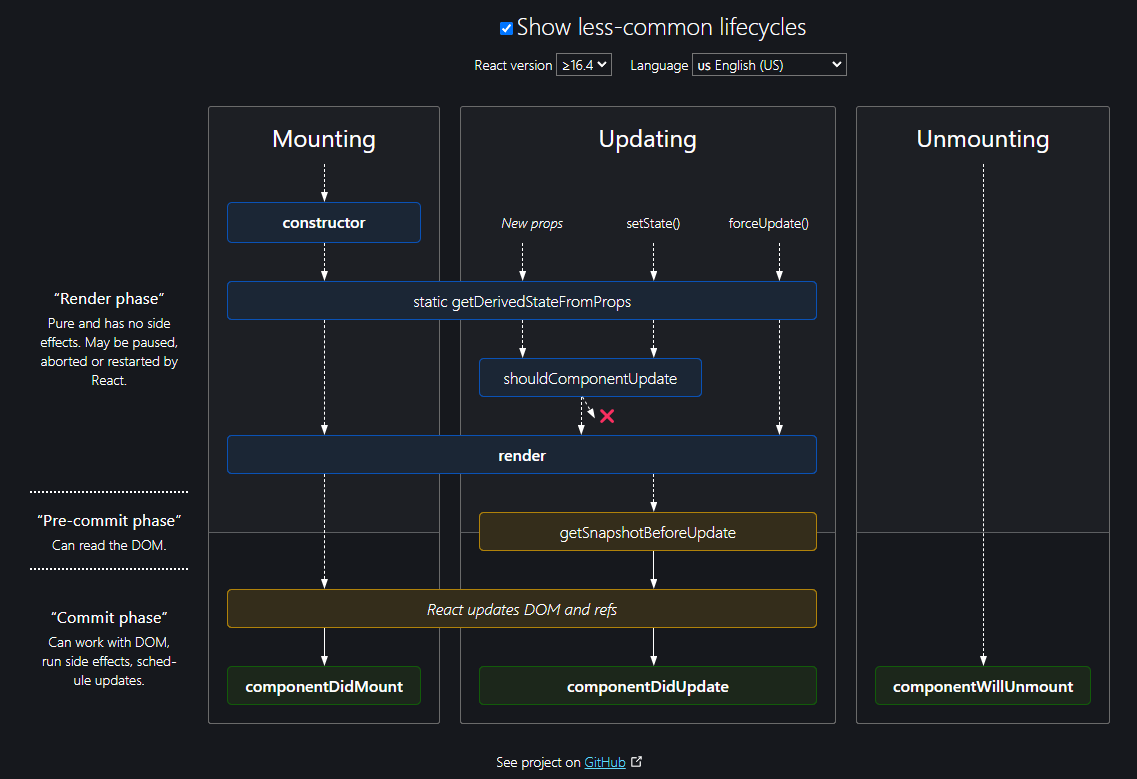
## Features of react?

## What is virtual DOM?

## Class vs functional component?

## Component lifecycle methods

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



### Constructor

### getDerivedStateFromProps

### Render

### ComponentDidMount

## 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

# Javascript

## Closure

## Call, bind and apply

## Debounce and throttling

## References

https://github.com/sudheerj