ASP.NET CORE

1. **Types of subquery**

Subqueries, also known as inner queries or nested queries, are queries nested inside another query. They are commonly used in SQL to retrieve data based on the results of another query. Here are the main types of subqueries:

1. Single-Row Subquery

Purpose: Returns a single row and a single column.

Use Case: Often used with comparison operators like =, <, >, etc.

Example:

SELECT employee\_name

FROM employees

WHERE employee\_id = (SELECT manager\_id FROM departments WHERE department\_name = 'Sales');

2. Multiple-Row Subquery

Purpose: Returns multiple rows but typically a single column.

Use Case: Used with operators like IN, ANY, ALL.

Example:

SELECT employee\_name

FROM employees

WHERE department\_id IN (SELECT department\_id FROM departments WHERE location\_id = 100);

3. Multiple-Column Subquery

Purpose: Returns multiple columns and multiple rows.

Use Case: Typically used in SELECT statements, often combined with IN, EXISTS, or comparison operators.

Copy code

SELECT employee\_id, employee\_name

FROM employees

WHERE (department\_id, job\_id) IN (SELECT department\_id, job\_id FROM job\_history WHERE employee\_id = 101);

4. Correlated Subquery

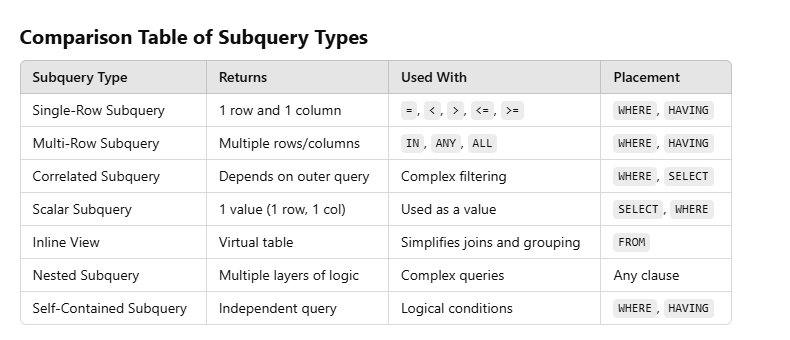
Purpose: The subquery depends on the outer query for its values.

Use Case: Used when the inner query needs a value from the outer query to execute.

Example:

SELECT e1.employee\_id, e1.employee\_name

FROM employees e1

WHERE salary > (SELECT AVG(salary) FROM employees e2 WHERE e1.department\_id = e2.department\_id);  


1. **Trim()**

The Trim() function is commonly used in programming languages to remove whitespace characters (spaces, tabs, etc.) from the beginning and end of a string. Here's how it works in several popular programming languages:

In C#, the Trim() method is a member of the String class.

Usage:

string str = " Hello, World! ";

string trimmedStr = str.Trim();

// trimmedStr is now "Hello, World!"

1. **Cookies**

Cookies are small pieces of data stored on the user's device by the web browser while browsing a website. They serve various purposes, such as maintaining session information, tracking user behavior, and storing user preferences. Here are the main types of cookies:

1. Session Cookies

Purpose: These cookies are temporary and are deleted once the user closes their browser.

Use Case: Used for storing temporary information, such as login status or items in a shopping cart.

Example: Maintaining the user's logged-in state while navigating through different pages of a website.

2. Persistent Cookies

Purpose: These cookies remain on the user's device for a set period or until manually deleted.

Use Case: Used for remembering login information, language preferences, and personalization settings across sessions.

Example: Remembering a user's login credentials for future visits.

3. First-Party Cookies

Purpose: Set by the website the user is currently visiting.

Use Case: Used for remembering user preferences, login details, and other settings relevant to the website.

Example: Storing user preferences such as language or theme on a website.

4. Third-Party Cookies

Purpose: Set by domains other than the one the user is currently visiting.

Use Case: Commonly used for tracking user behavior across multiple websites for advertising and analytics purposes.

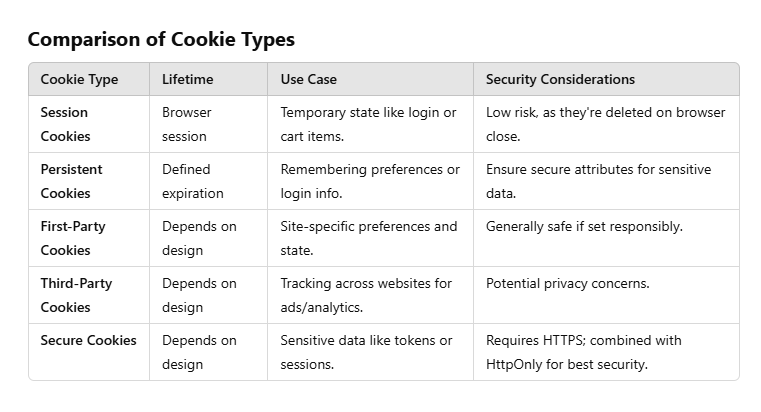
Example: Tracking user activity across different websites to serve personalized ads.

5. Secure Cookies

Purpose: Only transmitted over secure HTTPS connections.

Use Case: Often used for storing sensitive information like authentication tokens.

Example: Ensuring the security of session cookies by transmitting them over HTTPS.



1. **Insert record in 2 tables at the same time**

Inserting records into two tables at the same time can be achieved using transactions to ensure that either both inserts succeed or both fail, maintaining database integrity. Here's how to do it in several popular database systems:

BEGIN TRANSACTION;

-- Insert into the first table

INSERT INTO Table1 (Column1, Column2)

VALUES (Value1, Value2);

-- Insert into the second table

INSERT INTO Table2 (ColumnA, ColumnB)

VALUES (ValueA, ValueB);

-- Commit the transaction

COMMIT TRANSACTION;

-- If there's an error, rollback the transaction

BEGIN CATCH

ROLLBACK TRANSACTION;

THROW;

END CATCH;

1. **Composite Key**

A composite key is a type of primary key that consists of two or more columns in a database table. The combination of these columns uniquely identifies each row in the table. Composite keys are useful when a single column is not sufficient to uniquely identify a row.

Characteristics of Composite Keys

Uniqueness: The combination of the columns in the composite key must be unique for each row in the table.

Multiple Columns: It involves two or more columns.

Natural Key: Often used when the columns that make up the key have a natural relationship.

1. **VAR vs VARCHAR**

Feature CHAR VARCHAR

Length Fixed Variable

Storage Uses defined length, padded with spaces Uses actual string length + overhead

Performance Potentially faster for fixed length Potentially slower due to variable length

Use Case Fixed-length data (e.g., codes) Variable-length data (e.g., names)

1. **Identity column**

An identity column in SQL is a special type of column that automatically generates unique values for new rows. It's commonly used for primary keys when you want each row to have a unique identifier without manually assigning it.

1. **Suppose 10 rows present and all are deleted then what will be id of new row**

In SQL Server, the identity column continues to increment from its last value, even if all rows are deleted. Thus, if you have a table with an identity column and you delete all the rows, the next row inserted will have the identity value that follows the last value before the deletion.

1. **Can function in c# return multiple values**

In C#, a function (or method) can indeed return multiple values, but not directly. Instead, you can use several approaches to achieve this.

1. Using out Parameters

You can use out parameters to return multiple values from a method. The out keyword specifies that a parameter is passed by reference and is intended to be assigned a value inside the method.

2. Using Tuples

Tuples are a convenient way to return multiple values from a method. Tuples can hold a fixed number of elements of various types.

3. Using Dictionary or KeyValuePair

For scenarios where you need to return multiple values associated with keys, you can use Dictionary or KeyValuePair.

1. **Tuples**

In C#, a tuple is a data structure that can hold a fixed number of items of different types. Tuples are useful for grouping together multiple values into a single object without needing to define a separate class or struct.

Key Features of Tuples in C#

Fixed Size: Tuples have a fixed size determined at the time of their creation.

Different Types: Each element in a tuple can be of a different type.

Immutable: Once created, tuples are immutable, meaning their values cannot be changed.

Anonymous Types: Tuples are often used when you need to return multiple values from a method or function without creating a custom type.

Syntax:

var tuple1 = new Tuple<int, string, bool>(1, "Hello", true);

var tuple2 = (Id: 1, Message: "Hello", IsActive: true);

Console.WriteLine($"Id: {tuple2.Id}, Message: {tuple2.Message}, IsActive: {tuple2.IsActive}");

1. **In sql how will you retrieve odd no id of emp**

To retrieve records with odd-numbered IDs from a table in SQL, you can use the modulo operator % to filter IDs that are odd. The modulo operator returns the remainder of a division operation. For example, an ID is odd if dividing it by 2 gives a remainder of 1.

SELECT \* FROM Employees WHERE EmpID % 2 = 1

1. **Output caching**

Output Caching is a technique used to improve the performance and scalability of web applications by storing the output of requests so that subsequent requests can be served faster. Instead of processing the same request repeatedly, the cached output can be served directly, reducing the need for expensive operations like database queries or complex computations.

In ASP.NET Core, output caching can be implemented using middleware or response caching.

Example using **ResponseCaching Middleware**:

1. **LINQ Queries**

**To find the 5 highet num from given list**

var result = employeeList.Distinct().Take(5).OrderByDescending();

**Cross Join using LINQ**

var crossJoin = from customer in customers from order in orders select new { Customer = customer, Order = order };

**To perfrom Left Join in LINQ using GroupJoin and SelectMany**

var query = from customer in customers join order in orders on customer.CustomerId equals order.CustomerId into customerOrders from order in customerOrders.DefaultIfEmpty() select new { CustomerName = customer.Name, OrderId = order?.OrderId ?? 0, Product = order?.Product ?? "No Product" };

**To retrieve the 3rd record with the name "Tushar"**

var thirdTushar = employees

.Where(e => e.Name == "Tushar")

.Skip(2) // Skip the first 2 records

.FirstOrDefault(); // Get the 3rd record, or null if not found

**Intersect**

To find the common dates from two lists

var commonDates = list1.Intersect(list2);

**Union**

Purpose: Combines two sequences and returns distinct elements from both sequences.

var unionResult = list1.Union(list2);

**Except**

Purpose: Produces the set difference, which means it returns the elements from the first sequence that are not present in the second sequence.

var exceptResult = list1.Except(list2);

**Concat**

Purpose: Concatenates two sequences and includes all elements from both sequences (not distinct).

var concatResult = list1.Concat(list2);

**Distinct**

Purpose: Returns distinct elements from a sequence by removing duplicate values.

var distinctResult = list1.Distinct();

**Where**

Purpose: Filters a sequence based on a predicate.

var filteredResult = list1.Where(date => date > new DateTime(2023, 7, 21));

**Select**

Purpose: Projects each element of a sequence into a new form.

var selectResult = list1.Select(date => date.ToString("yyyy-MM-dd"));

**Join**

Purpose: Correlates the elements of two sequences based on matching keys.

var joinResult = list1.Join(

list2,

date1 => date1,

date2 => date2,

(date1, date2) => new { Date1 = date1, Date2 = date2 }

);

**GroupBy**

Purpose: Groups the elements of a sequence according to a specified key selector function.

var groupByResult = list1.GroupBy(date => date.Month);

**OrderBy / OrderByDescending**

Purpose: Sorts the elements of a sequence in ascending or descending order.

var orderByResult = list1.OrderBy(date => date);

var orderByDescResult = list1.OrderByDescending(date => date);

**Aggregate**

Purpose: Applies an accumulator function over a sequence.

var aggregateResult = list1.Aggregate((current, next) => current.AddDays(1));

**First / FirstOrDefault**

Purpose: Returns the first element of a sequence, or a default value if no element is found.

var firstResult = list1.First();

var firstOrDefaultResult = list1.FirstOrDefault();

**Last / LastOrDefault**

Purpose: Returns the last element of a sequence, or a default value if no element is found.

var lastResult = list1.Last();

var lastOrDefaultResult = list1.LastOrDefault();

**Single / SingleOrDefault**

Purpose: Returns the only element of a sequence, or a default value if the sequence is empty. Throws an exception if there is more than one element.

var singleResult = list1.Single();

var singleOrDefaultResult = list1.SingleOrDefault();

**Take / TakeWhile**

Purpose: Returns a specified number of contiguous elements from the start of a sequence.

var takeResult = list1.Take(2);

var takeWhileResult = list1.TakeWhile(date => date < new DateTime(2023, 7, 23));

**Skip / SkipWhile**

Purpose: Bypasses a specified number of elements in a sequence and then returns the remaining elements.

var skipResult = list1.Skip(1);

var skipWhileResult = list1.SkipWhile(date => date < new DateTime(2023, 7, 23));

1. **How will you differentiate between api methods, one is called from mobile and one from desktop**

Using route attributes to differentiate between API methods based on the client type (mobile or desktop) is a clean and organized approach. This method involves defining separate routes for different client types directly in the API controller.

using Microsoft.AspNetCore.Mvc;

[ApiController]

public class DataController : ControllerBase

{

// Route for Mobile clients

[HttpGet("api/mobile/data")]

public IActionResult GetMobileData()

{

// Mobile-specific logic

return Ok("Data for Mobile");

}

// Route for Desktop clients

[HttpGet("api/desktop/data")]

public IActionResult GetDesktopData()

{

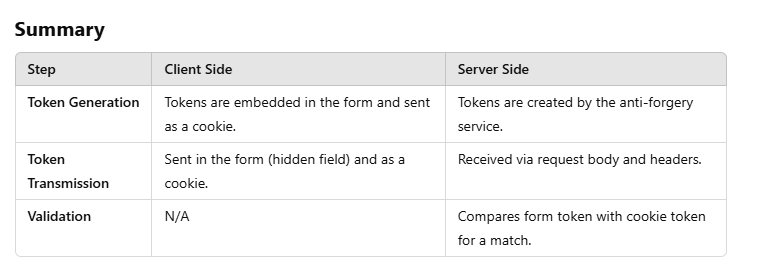
// Desktop-specific logic

return Ok("Data for Desktop");

}

}

1. **how antiforgery token will be validated in client and server**

Anti-forgery tokens are used to prevent Cross-Site Request Forgery (CSRF) attacks in web applications. In ASP.NET MVC and ASP.NET Core, the anti-forgery mechanism helps ensure that the requests made to the server are legitimate and originate from the authenticated user. Here's how anti-forgery tokens are generated and validated both on the client and server sides:  


1. **What is CORS**

Cross-Origin Resource Sharing (CORS) is a security feature implemented by web browsers that allows web applications to request resources from a different origin (domain) than the one that served the web page. This is often necessary in modern web applications where services and data might be distributed across different domains.

1. **Task, Reflection, Threads, Process**

1. **Task**

A Task represents an asynchronous operation in .NET. It is a unit of work that can be executed asynchronously and can return a value.

Used to perform non-blocking operations such as I/O-bound operations (e.g., database queries, web service calls) without blocking the main thread.

Improves the scalability and responsiveness of web applications by freeing up the main thread to handle other requests.

2. **Reflection**

Reflection is the ability of a program to inspect and manipulate its own structure and behavior at runtime. It allows you to obtain information about assemblies, modules, and types, and to create and invoke types dynamically.

3. **Thread**

A Thread is the smallest unit of execution within a process. Multiple threads can run concurrently within a single process, each performing a different task.

Often used in scenarios such as dependency injection, testing frameworks, and custom attribute processing.

4. **Process**

A Process is an instance of a running application. It contains the program code and its current activity, and it is the unit of resource allocation and scheduling in the operating system.

1. **Best practices in .NET development ensure that your code is maintainable, scalable, and efficient. Here are some key practices to follow:**

1. Code Quality and Readability

Use Meaningful Names: Name variables, methods, and classes clearly and descriptively.

Consistent Naming Conventions: Follow .NET naming conventions (PascalCase for classes and methods, camelCase for variables).

Comment and Document Code: Use XML comments for public members and inline comments for complex logic.

Adhere to SOLID Principles: Ensure your code is modular and maintainable.

2. Error Handling

Use Exceptions Appropriately: Throw exceptions only for exceptional conditions. Do not use them for control flow.

Catch Specific Exceptions: Avoid catching general exceptions (e.g., catch (Exception ex)); catch specific exceptions instead.

Log Errors: Use logging frameworks like Serilog or NLog to log errors and other significant events.

3. Code Structure and Organization

Follow Project Structure Conventions: Use standard folder structures (e.g., Controllers, Models, Views for MVC).

Layered Architecture: Separate concerns into layers (e.g., presentation, business logic, data access).

Use Dependency Injection: Leverage the built-in DI container or alternatives like Autofac.

4. Performance

Optimize for Performance: Profile and optimize performance-critical parts of your application.

Avoid Premature Optimization: Focus on writing clear and correct code first, then optimize as needed.

Use Asynchronous Programming: Use async and await to improve responsiveness.

5. Security

Validate Input: Always validate and sanitize user input to prevent attacks like SQL injection and XSS.

Use Secure Storage: Store sensitive data securely using encryption.

Authentication and Authorization: Implement proper authentication and authorization mechanisms using libraries like ASP.NET Identity.

6. Testing

Write Unit Tests: Ensure your code is testable and write unit tests for critical components.

Use Test Frameworks: Leverage frameworks like xUnit, NUnit, or MSTest.

Automate Testing: Integrate tests into your build pipeline using CI/CD tools.

7. Version Control and CI/CD

Use Version Control: Use Git or another version control system to track changes and collaborate with others.

Automate Builds and Deployments: Use CI/CD tools like Azure DevOps or GitHub Actions to automate builds and deployments.

8. Code Reviews and Collaboration

Conduct Code Reviews: Regularly review code with peers to catch issues early and share knowledge.

Use Pull Requests: Utilize pull requests for collaboration and code review.

9. Configuration Management

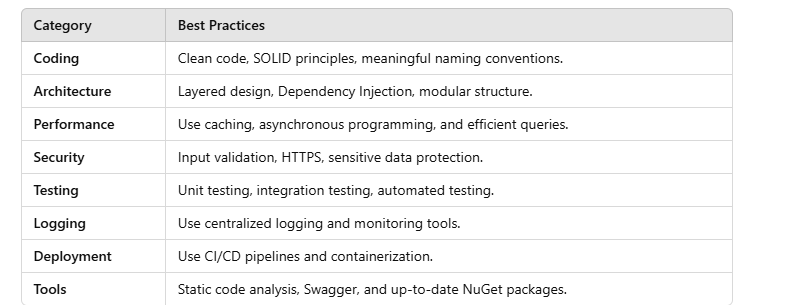
Use Configuration Files: Store configurations in appsettings.json or environment variables, not hard-coded in your application.

Secure Configuration Data: Protect sensitive configuration data using secrets management tools.

10. Documentation and Communication

Maintain Documentation: Keep documentation up-to-date, including README files and API documentation.

Communicate Effectively: Ensure clear communication within the team, especially regarding changes and issues.



1. **What is Uniform Interface?**

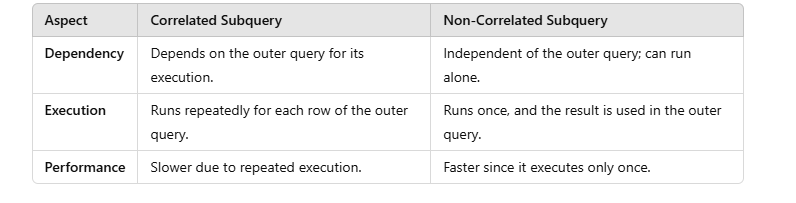
The **Uniform Interface** is a critical component of RESTful APIs, enabling standardization and simplicity in client-server interactions. By following this principle, developers can create scalable, interoperable, and easily maintainable web services.

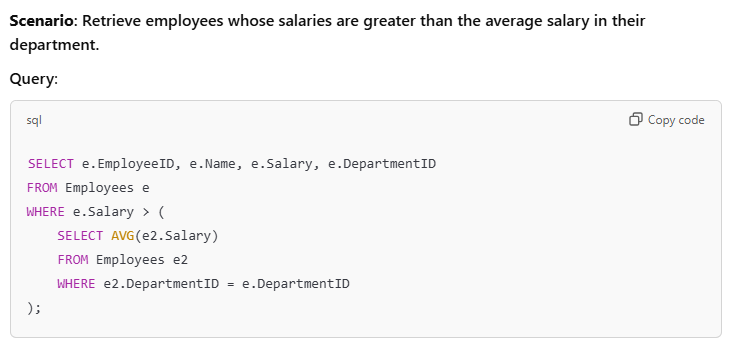
1. **What is Monolithic Architecture/ Centralized system**

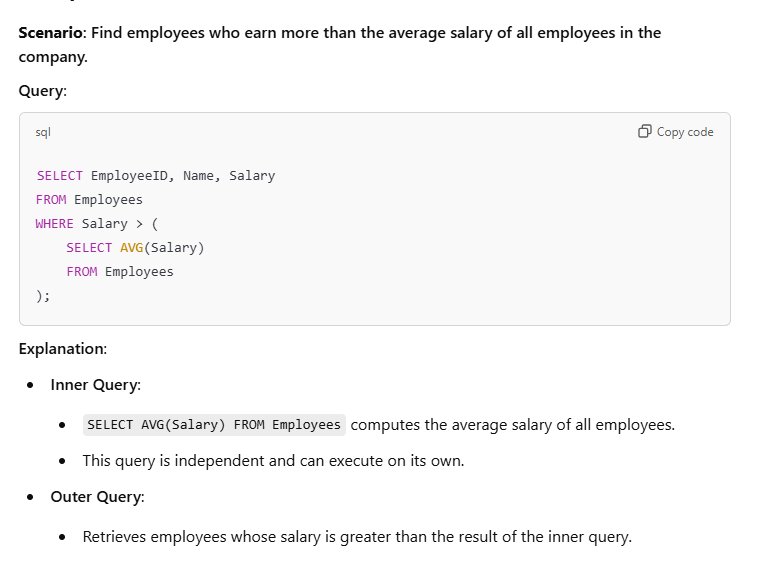
If all the components and functionalities of a project are entangled and combined in a single codebase, then that is a monolithic application.

Monolithic Architecture has less complexity => Easier to understand => Higher productivity

1. **What is correlated subquery?**



A **correlated subque**ry is a subquery that contains a reference to a table that also appears in the outer query. For:   
**Non-Correlated subquery**

****

1. **Server side and client side validation**

Real-World Example

Client-Side Validation:

* Ensures the email field is not empty and follows a valid format before submission.
* Feedback: "Please enter a valid email."

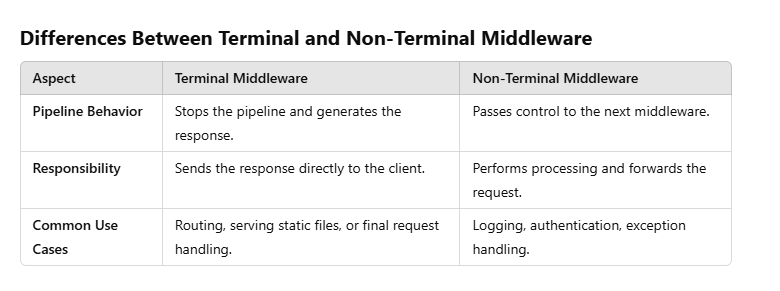
Server-Side Validation:

* Checks if the submitted email already exists in the database.
* Feedback: "This email is already registered."

Summary

While client-side validation improves user experience by providing immediate feedback, server-side validation ensures security and reliability by safeguarding against malicious or bypassed requests. Both are essential for building secure and user-friendly applications.

1. **Types of Middleware**

There are two types of middleware in . NET Core: terminal and non-terminal middleware. Terminal middleware is the final middleware component in the pipeline. It is responsible for sending the response back to the client.  


1. **What are the WCF contracts**

WCF (Windows Communication Foundation) contracts are a key concept in WCF that define the service's operations, data types, and the structure of the messages exchanged between client and service. There are three main types of contracts in WCF:

**Service Contracts**:

Define the operations that the service provides.

Use the [ServiceContract] attribute to define a service contract interface.

Use the [OperationContract] attribute to define methods within the service contract interface.

**Data Contracts:**

Define the data types that are used in service operations.

Use the [DataContract] attribute to define a data contract class.

Use the [DataMember] attribute to define the members of the data contract class that will be serialized.

**Message Contracts:**

Define the structure of the SOAP messages.

Use the [MessageContract] attribute to define a message contract class.

Use the [MessageHeader] and [MessageBodyMember] attributes to define the headers and body parts of the message.