1. **What is a Garbage collector? What are the different generations in the Garbage collector?**

Garbage collection is a form of automatic memory management. The Garbage collection manages the allocation and release of memory for an application. It attempts to reclaim memory that was allocated by the program, which is no longer referenced.

The Garbage collector stores new objects in generation 0. Objects created early in the application’s lifetime that survive collections are promoted and stored in generations 1 and 2.

Generation 0 is the youngest and contains short-lived objects.

Generation 1 contains short-lived objects and serves as a buffer between short-lived objects and long-lived objects.

Generation 2 contains long-lived objects.

1. **What are Access Modifiers and how to use them?**

Access Modifiers are keywords that set the level of accessibility for classes, methods, variables and other members of a class in Object-oriented programming languages such as Java, C#, and C++. They control how the components of a class are accessed in different parts of the code, providing a way to enforce encapsulation.

The types and use cases of Access Modifiers:

Public: Accessible from anywhere in the program, both inside and outside the class and package, as well as other classes and packages

Private: Accessible only within the same class. It is hidden from other classes and cannot be accessed directly from outside the class, even by subclasses.

Protected: Accessible within the same class, subclasses and within the same package (in Java) or assembly (in C#); It is useful when you want to allow access to derived classes but hide them from other parts of the program.

Default (No Modifier): has package-private access, which can only be accessed by classes in the same packages.

1. **​​What are extension methods and give me examples of built in extension methods. Have you created an extension method in your project? If yes, how with examples?**

Extension methods are special static methods in a static class which enables us to add new methods to build-in types and custom types without creating new derived type.

**WordCount** method to the string class;

**LINQ** methods provide several powerful extension methods for querying and manipulating collections and data sets: Where, Select, OrderBy.

**Enumerable** methods are typically used with collections like List or arrays: ToList, ToArray.

1. **What are lambda expressions, where have u used lambda expressions？**

**Lambda Expression** are short blocks of code which takes in parameters and returns a value. They are similar to methods, but they do not need a name, and they can be implemented right in the body of a method.

1. **Write LINQ, GroupBy, Join, and Where conditions.**

// Two classes SetUp:

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

public string Category { get; set; }

}

public class Sale

{

public int SaleId { get; set; }

public int ProductId { get; set; }

public int QuantitySold { get; set; }

public DateTime SaleDate { get; set; }

}

// A list of products and a list of sales

var products = new List<Product>

{

new Product { ProductId = 1, Name = "Laptop", Category = "Electronics" },

new Product { ProductId = 2, Name = "Desk Chair", Category = "Furniture" },

new Product { ProductId = 3, Name = "Monitor", Category = "Electronics" }

};

var sales = new List<Sale>

{

new Sale { SaleId = 1, ProductId = 1, QuantitySold = 3, SaleDate = DateTime.Today },

new Sale { SaleId = 2, ProductId = 1, QuantitySold = 2, SaleDate = DateTime.Today },

new Sale { SaleId = 3, ProductId = 2, QuantitySold = 5, SaleDate = DateTime.Today },

new Sale { SaleId = 4, ProductId = 3, QuantitySold = 1, SaleDate = DateTime.Today }

};

// LINQ query with GroupBy condition:

var totalSalesByProduct = sales

.GroupBy(s => s.ProductId)

.Select(g => new

{

ProductId = g.Key,

TotalQuantitySold = g.Sum(s => s.QuantitySold)

})

.ToList();

// LINQ query with Join condition:

var productSales = sales

.Join(products,

sale => sale.ProductId,

product => product.ProductId,

(sale, product) => new

{

product.Name,

sale.QuantitySold,

sale.SaleDate

})

.ToList();

// LINQ query with Where condition:

var electronicsProducts = products.Where(p => p.Category == "Electronics").ToList();

// LINQ query combined GroupBy, Join, and Where:

var electronicsSales = sales

.Join(products,

sale => sale.ProductId,

product => product.ProductId,

(sale, product) => new { sale, product })

.Where(x => x.product.Category == "Electronics")

.GroupBy(x => x.product.Name)

.Select(g => new

{

ProductName = g.Key,

TotalQuantitySold = g.Sum(x => x.sale.QuantitySold)

})

.ToList();

1. **What does sealed keyword does in C# ?**

It is used to limit a class’s or a member’s ability to inherit from anther class. A sealed class or member cannot be inherited by another class or overridden by a derived class.

1. **What are ref vs out keywords?**

In C#, **ref** and **out** are keywords used to pass parameters by reference to a method.

The **ref** keyword is used to pass an argument by reference, meaning the method can be read and modify the argument’s value, and changes made in the method will be reflected outside the method. The **ref** parameter must be initialized before it is passed to the method.

The **out** keyword is also used to pass an argument by reference. But the **out** parameter doesn’t need to be initialized before it is passed to the method. The method is expected to assign a value to it before the method completes.

1. **Can you explain about Normalization, especially 3NF and how it is helpful? Tell me scenarios where Denormalization is useful?**

**Normalization** is a database design technique used to reduce data redundancy and improve data integrity by organizing tables and establishing relationships among them. **Normalization** is broken into several “normal forms” (NFs), each with a set of rules designed to make the database more efficient and consistent.

The **Third Normal Form** (3NF) is a stage in the normalization process that ensures each non-primary key attribute in a table is only dependent on the primary key and not on other non-key attributes. This reduces redundancy and prevents certain types of anomalies during data updates.

**3NF benefits:**

**Reduces Redundancy**: By organizing data into separate tables, normalization eliminates duplicate data entries, reducing storage requirements and improving efficiency.

**Improves Data Integrity**: Normalization minimizes the risk of anomalies (insertion, update, and deletion anomalies) by ensuring that each piece of data is stored in one place, making data more consistent.

**Facilitates Better Maintenance**: With reduced redundancy and dependencies, it is easier to maintain data integrity and make changes in one place without impacting other tables.

**Denormalization** is the process of merging tables or introducing redundancy in a database to improve read performance.

**Scenarios where Denormalization is Useful**:

Data Warehousing and Reporting

High-Performance Applications

Caching Frequently Accessed Data

Data Redundancy for Avialability

1. **When do you use stored procedures and when do you use function? Differences between them?**

Cases to Use Stored Procedures:

Data Modification: to perform complex data manipulation, such as inserting, updating, or deleting records.

Batch Processing: when multiple SQL statements need to be executed together, such as batch operations.

Transaction Management: when need to control transactions.

Security: when need to encapsulate complex logic and restrict direct access to the underlying data.

Complex logic: when the operation involves multiple statements, loops, conditions.

Cases to Use Functions:

Calculations: perform calculation and return a result.

Reusability in Queries: need reusable logic that can be embedded in queries.

Return Table Data: Use table-valued functions when need to return a result set that can be treated like a table in queries.

Deterministic results: When the logic is deterministic and always returns the same result for the same input.

Stored Procedures are for more complex operations, including data modifications, and can’t be used directly in queries. Functions are ideal for calculations or returning values in queries, and they cannot modify the data or handle transactions.

1. **Explain Primary Key, Foreign Key and Unique Key with differences?**

Primary Key defines a unique identifier for each row in a table. Unique key enforces uniqueness but does not necessarily define a unique identifier. Foreign Key establishes a relationship between two tables.

1. **What is the difference Between Union and Union all?**

Union only keeps unique records. While Union All keeps all records including duplicates.

1. **Explain all the joins?**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

INNER JOIN returns records that have matching values in both tables.

LEFT (OUTER) JOIN returns all records from the left table, and the matched records from the right table.

RIGHT (OUTER) JOIN returns all records from the right table, and the matched records from the left bale.

FULL (OUTER) JOIN returns all records when there is a match in either left or right table.

SELF JOIN is a regular join, but the table is joined with itself.

1. **What are AddSingleton, AddScoped, AddTransient? What’s the difference?**

**AddSingleton**, **AddScoped**, and **AddTransient** are methods for configuring the lifetime of services when using Dependency Injection (DI). They define how often instances of the services are created and injected.

**AddSingleton**: creates a single instance of the service for the entire application lifespan. Once created, this instance is reused across all requests and throughout the app’s lifecycle.

**AddScoped**: creates a new instance of the service for each individual request. If a service is injected multiple times during a single request, the same instance is reused within that request.

**AddTransient**: creates a new instance of the service every time it’s requested. No caching occurs, so each injection point receives a fresh instance of the service.

**AddSingleton** is for global services that maintain state or perform initialization once. **AddScoped** is best for request-specific services that should have a unique state per request. **AddTransient** is suitable for lightweight, stateless services that do not require shared state and can be quickly instantiated without overhead.

1. **Explain Dependency Injection**

**Dependency Injection** is a programming technique that makes a class independent of its dependencies. It achieves that by decoupling the usage of an object from its creation.

1. **Explain MVC's application life cycle.**

In ASP.NET MVC, the MVC life cycle can be defined as a step by step occurring events while processing a request and then generating its response.

The application life cycle, in which the application process starts the running server until the time it stops, and it tagged the two events in the startup file of your application. i.e. the application start and end events.

1. **What will EF return IQueryable or IEnumerable?**

It depends on how to interact with the data.

When use Entity Framework (EF) LINQ queriers directly on the DbSet (e.g., context.Users.Where(…)), EF returns an IQuerable.

When force immediate execution by calling methods like .ToList(), .ToArray()…, EF executes the query immediately and returns an IDnumerable.

1. **How would you see the SQL queries generated by Entity Framework? What tools or coding would you implement?**

There are different methods to view the SQL queries generated by Entity Framework in different versions of EF.

ToQuertString() in EF Core 5+ to get SQL as a string without execution.

Logging with LogTo() in EF Core 3+ to configures logging for all SQL output

Database.log in EF 6 to log SQL to console or file.

1. **Explain the steps of code first approach with migrations**

Setup: Install EF, create DbContext and model classes.

Enable Migrations: Run the initial migration command to scaffold migrations.

Initial Migration: Use database update to create the initial database,

Modify Models: Make changes, then add and apply migrations to update the schema.

Rollback: Use database update to a previous migration if necessary.

Seed Data: Optional, use HasData to set up initial data.

1. **Why do you think we need an abstract class, why do you need an interface? Give few examples, where you have used abstract classes and interfaces in your project**

**Abstract classes** and **Interfaces** serve as two essential tools for building flexible, extensible, and maintainable code in object-oriented programming, but they serve different purposes.

An **abstract class** allows us to define a base class with some shared implementation while leaving some methods unimplemented for subclasses to provide specific details.

An **Interface** defines a contract that a class must follow, but without any implementation details. It’s a way of establishing a clear set of capabilities that a class promises to provide, making it ideal for defining functionality that can be applied to classes in different, unrelated parts of the hierarchy.

1. **What does static modifier do in C# ?**

**Static Modifier** is used to declare a static member, which belongs to the type itself rather than to a specific object.

1. **Write a query to create a view named that displays the total number for each product and their names and categories.**

CREATE VIEW ProductSummary AS

SELECT

p.ProductName,

p.Category,

SUM(pi.Quantity) AS TotalQuantity

FROM

Products p

JOIN

ProductInventory pi ON p.ProductID = pi.ProductID

GROUP BY

p.ProductName,

p.Category;

1. **Add a check constraint to the quantity\_sold column in the Sales table to ensure that the quantity sold is always greater than zero.**

ALTER TABLE Sales

ADD CONSTRAINT chk\_quantity\_sold\_positive

CHECK (quantity\_sold > 0);