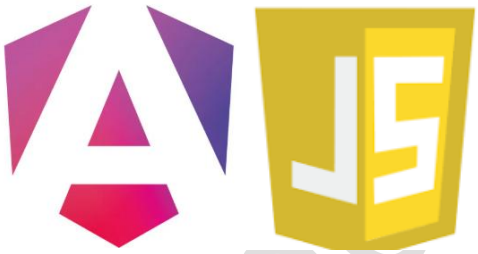

100+ .Net Full Stack

Interview Questions and Answers

Sai Reddy 5/14/25 Interview Tips

100+ Dot Net Full Stack Interview Q&A

General HR Questions

1. Tell me about yourself:

I am a .NET Full Stack Developer with [X] years of experience in developing web applications using C#, ASP.NET Core, Angular, JavaScript, and SQL Server. I've worked on end-to-end project development, API integrations, and performance optimization. I'm passionate about clean code and continuous learning. I also enjoy mentoring juniors and collaborating with cross-functional teams.

2. What are your strengths and weaknesses?

My strengths include strong problem-solving skills, quick learning ability, and writing clean, maintainable code. I'm also very organized and reliable under pressure.
As for weaknesses, I used to over-focus on perfection, but I've learned to balance quality with deadlines.

3. Why are you looking for a change?

I'm looking for a change to grow professionally by working on more challenging projects and enhancing my skills. I also seek a collaborative culture and better opportunities for learning modern technologies. My current role has limited growth potential.

4. Why do you want to join our company?

I've heard great things about Nagarro's innovative culture and focus on technology excellence. The company's diverse projects and global presence excite me. I believe it would be a great platform to learn, contribute, and grow.

5. Where do you see yourself in 5 years?

In five years, I see myself as a senior full-stack developer or team lead, contributing to complex solutions and mentoring others. I want to deepen my technical expertise and expand into architecture or project leadership roles.

6. Are you comfortable working in a team?

Yes, absolutely. I enjoy team collaboration and believe it leads to better problem-solving and knowledge sharing. I've always been an active team player and communicator.

7. Are you open to relocation?

Yes, I'm open to relocation if the opportunity aligns with my career goals. I value flexibility and am comfortable adapting to new environments.

8. What is your current salary and expected salary?

My current CTC is ₹3LPA per annum. Based on my experience and market standards, I am expecting ₹8. However, I am open to negotiation for the right opportunity.

9. How do you handle pressure or deadlines?

I prioritize tasks, plan ahead, and remain calm under pressure. I break down large tasks and stay focused on solutions. I also communicate early if risks arise to ensure timely delivery.

10. What is your notice period?

My notice period is 45 days. However, I am willing to negotiate for an early release if required.

.NET & Backend Development

1. What are the new features in C# 10/11?

C# 10 Features:

- Global using directives (global using System;)
- File-scoped namespaces (namespace Demo;)
- Record structs (value-type records)
- Constant interpolated strings
- Improved lambda expression support

C# 11 Features:

- Raw string literals (""" for multi-line or unescaped strings)
- List patterns ([1, 2, ..])
- Required members in constructors
- Generic math support
- UTF-8 string literals ("text"u8)

2. What is the difference between ref, out, and in parameters?

- ref: Passes a variable **by reference**, and **must be initialized** before passing.
- out: Passes by reference too, but the value **must be assigned inside the method**.
- in: Passes by reference but as **read-only**; cannot be modified inside the method.

```
void Test(ref int x, out int y, in int z) { ... }
```

3. What is the difference between a class and a struct?

Feature	Class	Struct
Type	Reference type	Value type
Storage	Heap	Stack
Inheritance	Supports inheritance	Does not support inheritance
Default constructor	Allowed	Not allowed
Use case	Complex objects	Small, lightweight objects like Point, Color

4. What is a delegate? Explain with example.

A **delegate** is a type that references a method with a specific signature. It allows methods to be passed as parameters.

```
delegate int Calculator(int a, int b);

Calculator add = (x, y) => x + y;
Console.WriteLine(add(2, 3)); // Output: 5
```

Delegates are used for callbacks, events, and asynchronous programming.

5. What are async and await in C#?

- async marks a method as asynchronous.
- await is used to **pause execution** until a Task completes, without blocking the thread.

```
async Task<string> GetDataAsync() {
   await Task.Delay(1000);
   return "Done";
}
```

6. What is dependency injection?

Dependency Injection (DI) is a design pattern that **provides dependencies (services) from outside** rather than creating them inside the class. It promotes loose coupling and testability.

```
public class Service {
   private readonly ILogger _logger;
   public Service(ILogger logger) {
      _logger = logger;
   }
}
```

7. What is garbage collection and how does it work?

Garbage Collection (GC) is a CLR feature that **automatically frees memory** used by objects no longer referenced.

It works in **generations** (Gen 0, 1, 2) to optimize performance. GC runs in the background and cleans up heap memory to prevent memory leaks.

8. What are SOLID principles? Can you explain them with examples?

- **S – Single Responsibility Principle:** A class should have only one reason to change.
  Example: Separate Invoice logic from InvoicePrinter.
- **O – Open/Closed Principle:** Code should be open for extension, closed for modification.
  Use interfaces or abstract classes to add new behavior.
- **L – Liskov Substitution Principle:** Subclasses should be replaceable with base classes.
  Derived classes shouldn't break functionality.
- **I – Interface Segregation Principle:** Prefer multiple small interfaces over one large one.
  Avoid forcing classes to implement unused methods.
- **D – Dependency Inversion Principle:** Depend on abstractions, not concrete classes.
  Use interfaces for decoupling and inject them.

ASP.NET Core MVC / Web API

1. What is the difference between ASP.NET MVC and Web API?

- ASP.NET MVC is used to build web applications that return HTML views.
- Web API is used to build RESTful services that return data (usually JSON or XML).
- MVC is for UI + backend logic; Web API is for data services consumed by clients like Angular or mobile apps.
- In ASP.NET Core, MVC and Web API are unified under a single framework.

2. What is middleware in ASP.NET Core?

- Middleware is a component in the request/response pipeline that processes HTTP requests.
- Examples: authentication, logging, routing, CORS, exception handling.

- Each middleware can perform operations before and after the next middleware in the pipeline.

```
app.UseMiddleware<CustomLoggingMiddleware>();
```

3. What are filters in MVC?

- Filters are used to run logic before or after controller actions.
- Types: Authorization filters, Action filters, Result filters, Exception filters.
- Example: Logging, authentication, error handling.

```
[Authorize]
public IActionResult Dashboard() { ... }
```

4. How do you secure your Web APIs?

- Use authentication and authorization (JWT tokens, OAuth, or Identity).
- Apply [Authorize] attribute on controllers or actions.
- Use HTTPS and API keys.
- Implement rate limiting and CORS policies.

5. What are RESTful services?

- REST (Representational State Transfer) is an architectural style using standard HTTP verbs: **GET, POST, PUT, DELETE, etc.**
- RESTful services are stateless, resource-based, and support CRUD operations.
- Responses are typically in JSON or XML format.

6. How do you handle exception handling in Web APIs?

- Use try-catch blocks for specific cases.
- For global handling, use Exception Middleware or filters.

```
app.UseExceptionHandler("/error"); // Middleware
```

- Return proper HTTP status codes (e.g., 400, 500) with error messages.

7. What is model binding and model validation?

- Model binding maps HTTP request data (query string, form, route, JSON) to C# objects.
- Model validation checks if data meets defined rules using data annotations.

```
public class User {
  [Required]
  public string Name { get; set; }
}
```

- Use ModelState.IsValid to check validation status.

8. How is routing handled in ASP.NET Core?

- ASP.NET Core uses endpoint routing, defined in Program.cs or Startup.cs.

```
app.MapControllerRoute(
 name: "default",
 pattern: "{controller=Home}/{action=Index}/{id?}");
```

- Attribute routing is also supported:

```
[Route("api/[controller]")]
public class ProductsController : ControllerBase { }
```

Data Access

1. What are the components of ADO.NET?

ADO.NET has two main types of components:

- **Connected components** (work with live DB connection):
  - SqlConnection
  - SqlCommand
  - SqlDataReader
  - SqlTransaction
- **Disconnected components** (work without continuous DB connection):
  - DataSet
  - DataTable
  - DataAdapter
  - DataRelation

2. What is the difference between ExecuteReader, ExecuteScalar, and ExecuteNonQuery?

Method	Purpose	Return Type
ExecuteReader()	Executes SQL query and returns **rows of data**	SqlDataReader
ExecuteScalar()	Returns **single value** (1st column of 1st row)	object
ExecuteNonQuery()	Executes **INSERT, UPDATE, DELETE**	int (number of rows affected)

```
command.ExecuteReader();     // SELECT
command.ExecuteScalar();     // SELECT COUNT(*)
command.ExecuteNonQuery();   // UPDATE, DELETE
```

3. How do you handle transactions in ADO.NET?

Transactions ensure that a set of database operations are executed **as a unit**, either all succeed or all fail.

```
using (SqlConnection conn = new SqlConnection(connectionString))
{
   conn.Open();
   SqlTransaction transaction = conn.BeginTransaction();
   SqlCommand command = conn.CreateCommand();
   command.Transaction = transaction;

   try
   {
      command.CommandText = "INSERT INTO Orders ...";
      command.ExecuteNonQuery();

      command.CommandText = "INSERT INTO OrderDetails ...";
      command.ExecuteNonQuery();

      transaction.Commit();
   }
   catch
```

```
        {
            transaction.Rollback();
        }
    }
```

Entity Framework (EF)

1. What is Entity Framework and how does it work?

Entity Framework (EF) is an **Object-Relational Mapper (ORM)** that allows .NET developers to work with a database using C# objects.

It handles:

- Mapping between classes and database tables.
- Generating SQL queries.
- Tracking changes and saving data with SaveChanges().

EF simplifies data access and reduces the need for manual ADO.NET code.

2. What is the difference between EF Core and EF 6?

Feature	EF 6	EF Core
Framework	Full .NET Framework	Cross-platform (.NET Core & .NET 5/6/7+)
Performance	Slower	Faster
LINQ Support	Good	Better and improving
NoSQL Support	Not supported	Some support (e.g., Cosmos DB)
Migrations	Available	More powerful and flexible

3. What are the different ways to load related entities?

- **Eager Loading**: Load related data using .Include() at query time.
- context.Orders.Include(o => o.OrderDetails).ToList();
- **Lazy Loading**: Load related data only when accessed (requires proxies and virtual navigation properties).
- **Explicit Loading**: Manually load data using .Entry().Reference().Load().
- context.Entry(order).Reference(o => o.Customer).Load();

4. What is Code First vs Database First?

- **Code First**: You define your C# classes (models), and EF generates the database schema.
    - Used with migrations.
    - Great for development-first approach.
- **Database First**: You start with an existing database and EF generates the C# classes (using .edmx or scaffolding).
    - Good for legacy databases or DBAs who manage schemas.

5. How do migrations work in EF Core?

EF Core Migrations track changes to your model and update the database schema accordingly.

Basic steps:

1. Add migration:
2. dotnet ef migrations add InitialCreate
3. Apply migration:
4. dotnet ef database update

Dapper:

1. What is Dapper and how does it differ from EF?

- **Dapper** is a **micro ORM** developed by Stack Overflow that works directly with ADO.NET.
- It maps query results to C# objects using extension methods on IDbConnection.
- **EF** is a full-fledged ORM that handles LINQ queries, change tracking, and schema migrations.

Feature	Dapper	Entity Framework (EF)
Speed	Very fast (lightweight)	Slower than Dapper
Learning Curve	Simple	More features, more learning
Change Tracking	No	Yes
Migrations	No	Yes
LINQ Support	Limited	Full LINQ support

2. When would you prefer Dapper over EF?

You should prefer **Dapper** when:

- Performance is critical (e.g., high-volume API, batch reads).
- You need fine control over SQL queries and tuning.
- You're working with **stored procedures** or legacy databases.
- You don't need tracking or complex relationships.
- Application requires lightweight and fast data access with fewer layers.

3. How do you perform joins and stored procedure calls using Dapper?

Join Example:

```
var sql = @"SELECT o.*, c.*
        FROM Orders o
        INNER JOIN Customers c ON o.CustomerId = c.Id";

var orderDict = new Dictionary<int, Order>();

var orders = connection.Query<Order, Customer, Order>(
   sql,
   (order, customer) => {
      order.Customer = customer;
      return order;
   },
   splitOn: "Id" // split based on second object's key
).ToList();
```

Stored Procedure Example:

```
var parameters = new DynamicParameters();
parameters.Add("@UserId", 1);

var result = connection.Query<User>(
   "GetUserById",
```

```
    parameters,
    commandType: CommandType.StoredProcedure
);
```

Frontend – Angular / HTML / CSS / JavaScript

Angular

1. What is Angular and what are its key features?

Angular is a **TypeScript-based front-end framework** developed by Google for building single-page applications (SPAs).

Key features:

- Component-based architecture
- Two-way data binding
- Dependency injection
- Built-in routing and HTTP client
- RxJS for reactive programming
- Ahead-of-Time (AOT) compilation

2. What is the difference between components and directives?

- **Components** are the main building blocks of UI in Angular. Each has a template (HTML), logic (TS), and styling (CSS).
- **Directives** are used to **add behavior** to DOM elements (e.g., *ngIf, *ngFor, custom directives).
- Components **have a view**, while directives **do not**.

3. What is dependency injection in Angular?

Dependency Injection (DI) in Angular is a design pattern that **injects dependencies (services or objects)** into components or other services.

constructor(private userService: UserService) {}

Angular's DI system automatically handles service creation and lifecycle, promoting modularity and testability.

4. What are lifecycle hooks?

Lifecycle hooks are special methods Angular calls at different stages of a component's life.

Common hooks:

- ngOnInit() – runs after component is initialized
- ngOnChanges() – when input properties change
- ngOnDestroy() – before component is destroyed

  ngOnchanges, ngOnInit, ngDoCheck(), ngaftercontentinit(), ngaftercontentchecked(), ngafterviewInit(), NgAfterviewChecked(), ngonDestroy().

5. What is a service and how do you use it?

A **service** is a reusable class that contains business logic or shared code (e.g., API calls, data manipulation).

Usage:

1. Create service:
2. @Injectable({ providedIn: 'root' })
3. export class UserService { ... }
4. Inject in component:
5. constructor(private userService: UserService) {}

6. How do you handle routing in Angular?

Angular uses the RouterModule to define routes in an app.

Steps:

1. Define routes:
2. const routes: Routes = [
3. { path: 'home', component: HomeComponent },
4. { path: 'about', component: AboutComponent }
5.];
6. Import RouterModule.forRoot(routes) in AppModule.
7. Use <router-outlet> in HTML and [routerLink] for navigation.

7. How do you communicate between components (Parent-Child)?

- **Parent to Child**: Use @Input() decorator.
- @Input() userData: string;
- **Child to Parent**: Use @Output() with EventEmitter.
- @Output() notify = new EventEmitter<string>();
- **Sibling or distant communication**: Use a shared service with Subjects or Observables.

8. How do you make HTTP calls in Angular?

Use Angular's HttpClient service from @angular/common/http.

1. Import HttpClientModule in AppModule.
2. Inject HttpClient in a service or component:
3. constructor(private http: HttpClient) {}
4. Make requests:

```
this.http.get<User[]>('/api/users').subscribe(data => { ... });
```

9.What is Pure & Impure Pipes in Angular. What is difference between them?

pipes are used to transform data in templates. **Pure pipes** and **impure pipes** differ primarily in how they handle changes to input data and when they are recalculated.

Pure Pipes:

- **Definition**: A **pure pipe** only re-evaluates when the input **reference** changes. Angular checks the input value and runs the pipe only if the input data changes (reference change).
- **Performance**: Pure pipes are **efficient** because they are called only when Angular detects a change in the input reference, reducing unnecessary recalculations.
- **Default Behavior**: By default, all pipes in Angular are pure.

Example of a Pure Pipe:

Let's create a simple pipe that converts a string to uppercase. This pipe will only re-run if the input string changes (not if it's just updated within the component).

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'uppercasePipe',
  pure: true  // Default is true, so it's pure
})
export class UppercasePipe implements PipeTransform {
  transform(value: string): string {
```

```
  return value.toUpperCase(); } }
```

In this example, Angular will re-run the transform() method only when the value input changes, i.e., when the reference to the string changes.

Impure Pipes:

- **Definition**: An **impure pipe** recalculates the output **whenever Angular checks the view**, regardless of whether the input reference has changed. This includes scenarios where the pipe's input is a complex object, array, or something that might change internally (e.g., array contents or object properties).
- **Performance**: Impure pipes are less efficient because they are recalculated on every change detection cycle, even if the input reference hasn't changed.
- **Usage**: Impure pipes are typically used when you need to track changes in objects that are mutable, like arrays, or when your pipe relies on other sources of change (e.g., global state).

Example of an Impure Pipe:

Let's create an impure pipe that filters an array and returns the elements that match a given condition. Since the array might change (e.g., elements could be added or removed), Angular needs to re-run this pipe on every change detection cycle.

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'filterArray',
  pure: false  // This makes the pipe impure
})
export class FilterArrayPipe implements PipeTransform {
  transform(value: any[], condition: string): any[] {
    return value.filter(item => item.includes(condition));
  }
}
```

In this case, Angular will re-run the pipe every time change detection runs, even if the array reference itself doesn't change, as long as any internal change occurs (like adding/removing items from the array).

Key Differences:

Feature	Pure Pipe	Impure Pipe
Recalculation Trigger	Only when input reference changes	Every change detection cycle
Efficiency	More efficient (recalculates less)	Less efficient (recalculates more)
Default Behavior	True by default	Must be explicitly set to false
Use Cases	Simple transformations (e.g., string manipulations)	Complex objects/arrays that might change internally

JavaScript

1. What is the difference between var, let, and const?

Keyword	Scope	Reassignable	Hoisted	Notes
var	Function	Yes	Yes (initialized as undefined)	Legacy, avoid in modern JS

Keyword	Scope	Reassignable	Hoisted	Notes
let	Block	Yes	No	Use for variables that change
const	Block	No	No	Use for constants (must be initialized)

2. What is a closure in JavaScript?

A **closure** is a function that **remembers its outer lexical environment**, even after the outer function has finished executing.

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    console.log(count);
  };
}
const counter = outer();
counter(); // 1
counter(); // 2
```

Closures are useful for data privacy and maintaining state.

3. What is the difference between == and ===?

- == is **loose equality** – it compares values **after type coercion**.
- '5' == 5 // true
- === is **strict equality** – it compares **value and type**.
- '5' === 5 // false

Always prefer === for type-safe comparisons.

4. What is event bubbling and capturing?

- **Bubbling**: The event starts at the target element and **bubbles up** to the root (document).
- **Capturing**: The event starts at the root and **captures down** to the target.

You can control this using the third parameter in addEventListener.

```
element.addEventListener('click', handler, true); // capturing
element.addEventListener('click', handler, false); // bubbling
```

5. Explain promises and async/await.

- A **Promise** represents a value that may be available **now, later, or never**.

```
const promise = new Promise((resolve, reject) => {
  resolve("Done");
});
```

- **async/await** is syntactic sugar over Promises to write asynchronous code in a **cleaner, synchronous-looking** way.

```
async function fetchData() {
  try {
    const res = await fetch('/api/data');
```

```
    const data = await res.json();
    console.log(data);
  } catch (err) {
    console.error(err);
  }
}
```

HTML/CSS

1. What are semantic HTML elements?

Semantic HTML elements provide meaning to the web content, helping both browsers and developers understand the structure of the webpage better. These elements also improve accessibility and SEO.

Examples of semantic elements:

```
<header>, <footer>, <article>, <section>, <nav>, <main>
```

2. What is the box model?

The **box model** defines the layout structure of an element, consisting of:

- **Content**: The actual content (e.g., text or images).
- **Padding**: Space between content and border.
- **Border**: The border surrounding the padding (optional).
- **Margin**: Space between the element's border and adjacent elements.

In total, the box model looks like this:

```
Margin -> Border -> Padding -> Content
```

3. How does Flexbox work?

Flexbox (Flexible Box Layout) is a CSS layout model that enables easy alignment and distribution of items within a container, even when their sizes are unknown or dynamic.

- **Main axis**: Horizontal or vertical direction where items are laid out.
- **Cross axis**: Perpendicular to the main axis.

Flexbox makes it simple to:

- Center items vertically or horizontally.
- Control the order of elements.
- Automatically adjust items' sizes.

```
.container {
  display: flex;
  justify-content: center;  /* Horizontally center items */
  align-items: center;      /* Vertically center items */
}
```

4. How do you make a responsive website using Bootstrap?

To create a responsive website using **Bootstrap**:

1. **Include the Bootstrap CSS** file in your project.
2. Use Bootstrap's **grid system** with rows and columns (.container, .row, .col-*).
3. Apply **responsive classes** like .col-md-4 to control layout for different screen sizes (small, medium, large).
4. Use Bootstrap's built-in classes like .img-fluid for responsive images and .d-none for hiding elements on specific breakpoints.

Example of a responsive grid:

```
<div class="container">
```

```
 <div class="row">
  <div class="col-md-4">Column 1</div>
  <div class="col-md-4">Column 2</div>
  <div class="col-md-4">Column 3</div>
 </div>
</div>
```

5. What is the difference between inline, inline-block, and block?

- **Inline**:
  - Does not start on a new line.
  - Only takes up as much width as necessary.
  - Cannot set width/height.
  - Example: , <a>
- **Inline-block**:
  - Behaves like inline but allows setting width and height.
  - Elements stay on the same line unless the width exceeds the container.
  - Example: <img>, <button>
- **Block**:
  - Takes up the full width of the parent, starts on a new line.
  - Can have width/height.
  - Example: , <p>

SQL

1. What are joins in SQL? Explain different types.

Joins are used to combine rows from two or more tables based on a related column between them. Common types of joins:

- **INNER JOIN**: Returns only rows that have matching values in both tables.

```
SELECT * FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

- **LEFT JOIN (OUTER JOIN)**: Returns all rows from the left table and matched rows from the right table. If no match, NULL values are returned for right table columns.

```
SELECT * FROM Orders
LEFT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

- **RIGHT JOIN (OUTER JOIN)**: Returns all rows from the right table and matched rows from the left table. If no match, NULL values are returned for left table columns.

```
SELECT * FROM Orders
RIGHT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

- **FULL OUTER JOIN**: Returns rows when there is a match in **either** the left or right table. If no match, NULL values are returned for non-matching rows.

```
SELECT * FROM Orders
FULL OUTER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

- **CROSS JOIN**: Returns the Cartesian product of both tables (every row of the first table is combined with every row of the second table).

```
SELECT * FROM Orders
CROSS JOIN Products;
```

2. What is the difference between WHERE and HAVING?

- **WHERE**: Filters rows **before** aggregation (used in SELECT, UPDATE, DELETE queries). It cannot be used with aggregate functions.

```
SELECT * FROM Orders WHERE Amount > 100;
```

- **HAVING**: Filters rows **after** aggregation (used with GROUP BY and aggregate functions like COUNT, SUM).

```
SELECT CustomerID, COUNT(*) FROM Orders
GROUP BY CustomerID
HAVING COUNT(*) > 5;
```

3. What is indexing? What types of indexes are there?

Indexing is a technique used to speed up the retrieval of data from a table. It creates a **data structure** that allows quick lookup of rows based on column values.

Types of indexes:

- **Unique Index**: Ensures that the values in the indexed column are unique.
- **Composite Index**: Index on multiple columns.
- **Clustered Index**: Sorts and stores the data rows in the table based on the index (each table can have only one clustered index).
- **Non-clustered Index**: Stores a separate structure from the table data, pointing to the rows.
- **Full-text Index**: Used for searching large text fields.

4. What are stored procedures, triggers, and views?

- **Stored Procedure**: A precompiled collection of SQL statements that can be executed with a single call. It can accept parameters and return results.

```
CREATE PROCEDURE GetOrdersByCustomer(@CustomerID INT)
AS
SELECT * FROM Orders WHERE CustomerID = @CustomerID;
```

Trigger: A special type of stored procedure that automatically executes when a certain event occurs (e.g., INSERT, UPDATE, DELETE).

```
CREATE TRIGGER AfterInsertOrder
ON Orders
FOR INSERT
AS
BEGIN
  PRINT 'New order inserted';
END;
```

View: A virtual table that provides a predefined query result. It simplifies complex queries and allows for abstraction.

```
CREATE VIEW ActiveOrders AS
SELECT * FROM Orders WHERE Status = 'Active';
```

5. How do you optimize SQL queries?

To optimize SQL queries, you can:

1. **Use indexes** to speed up searches and joins.
2. **Avoid SELECT *** and select only the columns you need.
3. **Avoid subqueries** if joins can be used.
4. **Use WHERE conditions** to reduce the dataset early.

5. **Use proper joins**: Avoid unnecessary joins and prefer INNER JOIN over OUTER JOIN unless necessary.
6. **Analyze and optimize execution plans** (e.g., using EXPLAIN in MySQL).
7. **Limit the number of rows returned** using LIMIT or TOP when applicable.
8. **Consider using batch processing** for large datasets.

6. What is normalization and denormalization?

- **Normalization**: The process of organizing database tables to reduce redundancy and dependency by splitting large tables into smaller ones. Common normal forms:
  - 1NF (First Normal Form): Ensures that each column contains atomic values.
  - 2NF (Second Normal Form): Removes partial dependencies.
  - 3NF (Third Normal Form): Removes transitive dependencies.
- **Denormalization**: The process of intentionally introducing redundancy into the database by merging tables or copying data to improve query performance, especially in read-heavy applications.

7. How do you handle transactions in SQL?

Transactions ensure that a series of database operations are executed as a single unit, either **all succeed** or **all fail**.

- **Begin Transaction**: Starts the transaction.
- **Commit**: Saves changes made during the transaction.
- **Rollback**: Reverts all changes made during the transaction in case of an error.

```
BEGIN TRANSACTION;

UPDATE Orders SET Status = 'Shipped' WHERE OrderID = 1;
UPDATE Inventory SET Quantity = Quantity - 1 WHERE ProductID = 10;

IF @@ERROR != 0
   ROLLBACK;
ELSE
   COMMIT;
```

Transactions help maintain data integrity.

Azure DevOps

1. What is Azure DevOps?

Azure DevOps is a **set of development tools** from Microsoft for planning, developing, testing, and deploying software. It provides a full DevOps toolchain to support continuous integration and continuous delivery (CI/CD).

Key features:

- **Azure Repos**: Source control management (Git or TFVC).
- **Azure Pipelines**: CI/CD pipelines for building, testing, and deploying code.
- **Azure Boards**: Agile project management and tracking work items.
- **Azure Test Plans**: Testing tools and managing test cases.
- **Azure Artifacts**: A package management system to host and share packages.

2. How do you set up a CI/CD pipeline?

A CI/CD pipeline in Azure DevOps automates the process of integrating code and deploying it.

1. **Set up a project** in Azure DevOps.
2. **Create a repository** (Azure Repos or GitHub).
3. **Create a Pipeline**:
    - Go to **Pipelines** and click **New Pipeline**.
    - Choose the repository (e.g., Azure Repos Git, GitHub, etc.).
    - Select a template or create a YAML pipeline (defining stages like build, test, deploy).
4. **Configure Build Steps**:
    - Add build tasks (e.g., restore, build, test).
    - Choose a build agent (e.g., Hosted Ubuntu, Windows).
5. **Configure Release Steps**:
    - After build completion, create a release pipeline to deploy to different environments.
    - Configure approval processes, and environment variables, and set deployment triggers.
6. **Run and monitor the pipeline** for success or failure.

3. What is a build and release pipeline?

- **Build Pipeline**: This is responsible for **building** the source code, running tests, and packaging the application. It triggers when changes are made to the repository and creates build artifacts (e.g., .exe, .jar, or Docker image).

Example tasks in a build pipeline:

- Restore dependencies.
- Compile code.
- Run unit tests.
- Publish artifacts.

- **Release Pipeline**: The release pipeline **deploys** the build artifacts to various environments (e.g., Dev, QA, Production) and handles the continuous delivery process. It can include tasks like configuration, approval gates, and deployment.

Example tasks in a release pipeline:

- Deploy to development.
- Run smoke tests.
- Deploy to production.

4. How do you manage source control in Azure Repos?

Azure Repos is a Git-based version control system that helps manage source code.
Key steps to manage source control:

1. **Create a repository** in Azure DevOps under **Repos**.
2. **Clone** the repository locally using Git commands:
3. git clone https://dev.azure.com/your_organization/your_project/_git/your_repo
4. **Add, commit, and push changes**:
5. git add .
6. git commit -m "Your commit message"
7. git push
8. **Branching and Merging**: You can create branches for new features or fixes and merge them via Pull Requests (PRs) to maintain code quality and control.
9. **Branch Policies**: Set branch policies in Azure DevOps (e.g., required reviewers, build validation) to enforce rules.

5. What are artifacts?

Artifacts are the **files or packages** produced by a build pipeline, which can be shared and deployed through a release pipeline.

Examples of artifacts:

- Compiled binaries (e.g., .exe, .dll, .jar)
- Docker images
- Deployment packages (e.g., .zip or .tar files)
- Configuration files
- NuGet, npm, or other package manager files

interpersonal and problem-solving

1. Describe a challenging bug you fixed and how.

I encountered a bug where an API was returning incorrect data intermittently. The issue was traced to a race condition where multiple threads were accessing shared resources without proper synchronization.

Steps taken:

- I reviewed the code to identify shared resources.
- Added locking mechanisms (lock statements in C#) to ensure that only one thread could access the resource at a time.
- After implementing the fix, I added unit tests to cover this scenario and confirmed that the issue was resolved.

The bug was fixed by ensuring thread safety and thorough testing, which prevented the issue from reoccurring.

2. Have you worked in Agile methodology?

Yes, I've worked in **Agile** for several years. My teams followed **Scrum** practices, working in **2-week sprints** with regular stand-ups, sprint planning, and sprint retrospectives.

During each sprint:

- We prioritized tasks based on business value.
- We collaborated closely with stakeholders for feedback.
- At the end of each sprint, we delivered a potentially shippable product increment.

This iterative approach helped improve collaboration, allowed for faster delivery, and enabled us to quickly adapt to changes.

3. How do you handle code reviews?

I view **code reviews** as an opportunity for learning and improving code quality. When reviewing code, I focus on:

- **Readability**: Ensuring the code is clear and easy to understand.
- **Performance**: Checking for any inefficiencies or potential performance issues.
- **Best practices**: Verifying adherence to coding standards and design principles (e.g., SOLID).
- **Test coverage**: Ensuring that the code is well-tested and handles edge cases.

During the review process, I provide constructive feedback, being mindful of the tone and focusing on solutions. I also appreciate feedback on my own code and use it to enhance my skills.

4. Tell me about a time you disagreed with your team lead. How did you handle it?

In one project, I disagreed with my team lead on the approach to handle error logging. I believed using a centralized logging service would improve maintainability, while my lead preferred a more lightweight, local solution.

How I handled it:

- I presented my arguments with clear reasoning, demonstrating the long-term benefits of a centralized logging approach (e.g., easier monitoring, better scalability).
- I also listened to my team lead's concerns, acknowledging their points about simplicity.
- Ultimately, we found a compromise: We implemented centralized logging but made it optional for specific modules, allowing flexibility.

5. How do you prioritize tasks when working on multiple modules?

When managing multiple modules, I follow these steps:

1. **Assess the importance and urgency** of each task, considering business impact and deadlines.
2. **Break down complex tasks** into smaller, manageable chunks, ensuring that critical tasks are completed first.
3. **Use tools like task boards (e.g., JIRA)** to track progress and ensure transparency.
4. **Collaborate with the team** to balance workloads and address any blockers.
5. **Communicate regularly with stakeholders** to align expectations and adjust priorities if needed.