✅SQL Interview Questions (All Levels)

□ Basic Level

1. **What is SQL? What are the different types of SQL commands?**

Answer:
SQL (Structured Query Language) is a standard language used to **communicate with relational databases** for performing operations such as querying, inserting, updating, and deleting data.

SQL command categories:

- **DDL (Data Definition Language):** `CREATE`, `ALTER`, `DROP`, `TRUNCATE`
- **DML (Data Manipulation Language):** `INSERT`, `UPDATE`, `DELETE`
- **DQL (Data Query Language):** `SELECT`
- **DCL (Data Control Language):** `GRANT`, `REVOKE`
- **TCL (Transaction Control Language):** `COMMIT`, `ROLLBACK`, `SAVEPOINT`

2. **What is a primary key and foreign key?**

Answer:

- **Primary Key:**
  - Uniquely identifies each row in a table.
  - Cannot have NULLs.
  - Only one primary key per table (can be composite).
  - Example: `EmployeeID` in an `Employees` table.
- **Foreign Key:**
  - Creates a link between two tables.
  - Refers to the primary key in another table.
  - Ensures referential integrity.
  - Example: `DepartmentID` in `Employees` referencing `Departments`.

3. **What is the difference between WHERE and HAVING clauses?**

Answer:

Clause	Purpose	Used With
`WHERE`	Filters rows **before** aggregation	Any SQL query
`HAVING`	Filters results **after** aggregation (GROUP BY)	Aggregate queries

Example:

```
sql
CopyEdit
SELECT Department, COUNT(*)
FROM Employees
WHERE Active = 1
GROUP BY Department
HAVING COUNT(*) > 5;
```

4. **What are joins? Explain different types of joins (INNER, LEFT, RIGHT, FULL).**

Answer:
Joins are used to **retrieve data from multiple tables** based on related columns.

- **INNER JOIN**: Returns records that match in both tables.
- **LEFT JOIN (LEFT OUTER)**: All records from the left table + matched records from the right.
- **RIGHT JOIN (RIGHT OUTER)**: All records from the right table + matched from the left.
- **FULL JOIN (FULL OUTER)**: All records when there's a match in either table.

Example (INNER JOIN):

```
sql
CopyEdit
SELECT e.Name, d.DepartmentName
FROM Employees e
INNER JOIN Departments d ON e.DeptId = d.Id;
```

5. **What is normalization? What are its types?**

Answer:
Normalization is the process of **organizing data to reduce redundancy** and improve data integrity.

Common normal forms:

- **1NF (First Normal Form):** Remove repeating groups, use atomic values.
- **2NF:** Remove partial dependencies (applies to composite keys).
- **3NF:** Remove transitive dependencies.
- **BCNF:** A stricter version of 3NF.

In my experience, normalization helps maintain **data consistency** and simplifies **database maintenance**, but in some cases (e.g., reporting), denormalization is used for performance.

6. **What is a view? How is it different from a table?**

Answer:
A **view** is a **virtual table** based on a SELECT query. It does **not store data** physically, only the SQL logic.

Differences:

Feature	Table	View
Data storage	Stores data	No data, just a definition
Editable	Yes	Sometimes (if no joins, group by, etc.)
Performance	Faster (indexed)	Slightly slower (computed at runtime)

Use views to **simplify complex queries**, apply **security** (restricted columns), or create **reusable business logic**.

7. **What is the difference between DELETE, TRUNCATE, and DROP?**

Answer:

Command	Deletes Data	Removes Structure	Rollback Possible	Speed
DELETE	Yes	No	Yes (if in transaction)	Slower (row-by-row)
TRUNCATE	Yes (all rows)	No	No	Fast
DROP	Yes	Yes	No	Fastest

Example:

```
sql
CopyEdit
DELETE FROM Employees WHERE DeptId = 5; -- Conditional delete
TRUNCATE TABLE Employees; -- Deletes all data
DROP TABLE Employees; -- Deletes table structure and data
```

8. **What is indexing? Why is it important?**

Answer:
An index is a **performance optimization** structure that allows faster data retrieval.

- Works like a book index — speeds up searches on large tables.
- **Types**: Clustered, Non-clustered, Unique, Composite, Full-text
- SQL Server uses **B-Trees** internally.

Example:

```
sql
CopyEdit
CREATE NONCLUSTERED INDEX idx_name ON Employees(Name);
```

In my experience, indexing **significantly improves SELECT** performance but should be balanced as it can **slow down INSERT/UPDATE** operations.

9. **What is a stored procedure? How do you call it?**

Answer:
A **stored procedure** is a precompiled group of SQL statements saved in the database, often used for **modular and secure database logic**.

Creating:

```
sql
CopyEdit
CREATE PROCEDURE GetEmployeeById
    @EmpId INT
AS
BEGIN
    SELECT * FROM Employees WHERE Id = @EmpId;
END
```

Calling:

```
sql
CopyEdit
EXEC GetEmployeeById @EmpId = 1;
```

I use stored procedures for **reusable logic, better performance, and security** (e.g., role-based execution).

10. **What is a trigger in SQL?**

Answer:
A **trigger** is a **special stored procedure** that automatically runs **in response to INSERT, UPDATE, or DELETE events** on a table.

Example:

```
sql
CopyEdit
CREATE TRIGGER trg_LogDelete
ON Employees
AFTER DELETE
AS
BEGIN
    INSERT INTO EmployeeLog (EmpId, DeletedOn)
    SELECT Id, GETDATE() FROM deleted;
END
```

Triggers are useful for **auditing**, **enforcing business rules**, and **automatic logging**, but I use them cautiously due to their hidden execution and potential performance impact.

□ **Intermediate Level**

1. **Write a SQL query to fetch the second-highest salary from an employee table.**

Answer:

```
sql
CopyEdit
SELECT MAX(Salary) AS SecondHighestSalary
FROM Employees
WHERE Salary < (SELECT MAX(Salary) FROM Employees);
```

✅Alternate using `ROW_NUMBER()`:

```
sql
CopyEdit
SELECT Salary
FROM (
  SELECT Salary, ROW_NUMBER() OVER (ORDER BY Salary DESC) AS rn
  FROM Employees
) AS ranked
WHERE rn = 2;
```

2. **How do you find duplicate rows in a table?**

Answer:

```
sql
CopyEdit
```

```
SELECT Name, COUNT(*)
FROM Employees
GROUP BY Name
HAVING COUNT(*) > 1;
```

If checking across multiple columns:

```
sql
CopyEdit
SELECT Name, DeptId, COUNT(*)
FROM Employees
GROUP BY Name, DeptId
HAVING COUNT(*) > 1;
```

3. **Explain ACID properties in SQL.**

Answer:

Property	Description
Atomicity	All steps in a transaction are completed or none at all.
Consistency	The database remains in a valid state before and after the transaction.
Isolation	Concurrent transactions are isolated from each other.
Durability	Once committed, changes remain even if the system crashes.

These ensure **data integrity** and **reliable transactions**.

4. **What is the difference between clustered and non-clustered index?**

Answer:

Feature	Clustered Index	Non-Clustered Index
Storage	Sorts and stores the actual data rows	Stores pointers to data rows
Count per table	Only one (because data can be sorted one way)	Can have multiple
Speed	Faster for range queries	Slower for large reads but flexible

Example:

```
sql
CopyEdit
CREATE CLUSTERED INDEX idx_emp_id ON Employees(Id);
CREATE NONCLUSTERED INDEX idx_emp_name ON Employees(Name);
```

5. **How do you optimize a slow-running SQL query?**

Answer:
I usually follow these steps:

- Use **indexes** on frequently searched columns.
- Avoid `SELECT *`; only select required columns.
- Analyze **execution plan** to find bottlenecks.

- Optimize **joins** (especially nested or cross joins).
- Replace **subqueries** with **joins** when applicable.
- Use **WHERE**, **LIMIT**, or **TOP** to reduce scanned rows.
- Check for **parameter sniffing** and use **option (recompile)** if needed.

6. **What is the difference between UNION and UNION ALL?**

Answer:

Feature	**`UNION`**	**`UNION ALL`**
Duplicates	Removes duplicates	Includes duplicates
Performance	Slower due to distinct sort	Faster
Use case	When you need distinct records	When duplicates are okay

```
sql
CopyEdit
SELECT Name FROM Table1
UNION
SELECT Name FROM Table2;

-- vs
SELECT Name FROM Table1
UNION ALL
SELECT Name FROM Table2;
```

7. **What are subqueries and correlated subqueries?**

Answer:

- **Subquery**: A query inside another query, executed once.

  ```
  sql
  CopyEdit
  SELECT Name FROM Employees
  WHERE DeptId = (SELECT Id FROM Departments WHERE Name = 'HR');
  ```

- **Correlated Subquery**: Uses data from the outer query and runs **per row**.

  ```
  sql
  CopyEdit
  SELECT e.Name
  FROM Employees e
  WHERE Salary > (SELECT AVG(Salary) FROM Employees WHERE DeptId = e.DeptId);
  ```

8. **How do you implement transactions in SQL Server?**

Answer:

```
sql
CopyEdit
BEGIN TRANSACTION;

BEGIN TRY
    UPDATE Accounts SET Balance = Balance - 100 WHERE Id = 1;
```

```
    UPDATE Accounts SET Balance = Balance + 100 WHERE Id = 2;

    COMMIT;
END TRY
BEGIN CATCH
    ROLLBACK;
    -- Log error
END CATCH;
```

I use transactions for **data consistency** in multi-step operations.

9. **What is a CTE (Common Table Expression)?**

Answer:
A **CTE** is a temporary result set defined within a `WITH` clause, used to simplify complex queries like recursion or joins.

Example:

```
sql
CopyEdit
WITH TopEmployees AS (
  SELECT Name, Salary
  FROM Employees
  WHERE Salary > 50000
)
SELECT * FROM TopEmployees;
```

✅Used in recursive queries like hierarchical (tree/manager) structures.

10. **What is a temporary table? When do you use it?**

Answer:
A **temporary table** is a table created in **tempdb** for **storing intermediate results** during session execution.

Syntax:

```
sql
CopyEdit
CREATE TABLE #TempEmployees (Id INT, Name VARCHAR(50));
INSERT INTO #TempEmployees SELECT Id, Name FROM Employees;
```

✅Use cases:

- When processing large intermediate datasets.
- For breaking complex queries into parts.
- Used in **stored procedures** to store results temporarily.

□ Advanced Level

1. **How do you handle deadlocks in SQL Server?**

Answer:
A **deadlock** occurs when two or more processes block each other by holding locks the other needs.

To handle deadlocks:

- Use **`TRY...CATCH`** to handle the deadlock error (error code 1205).
- Use `SET DEADLOCK_PRIORITY LOW` to allow lower priority processes to be chosen as the victim.
- Avoid long-running transactions and **keep transactions short**.
- Always access resources in the **same order** across procedures.
- Use **`WITH (NOLOCK)`** cautiously for reads when real-time consistency is not critical.

You can also detect them using **SQL Server Profiler**, **Extended Events**, or `sys.dm_tran_locks`.

2. **Explain isolation levels in SQL and their impact on performance.**

Answer:
Isolation levels define how one transaction **is isolated from the effects of others**:

Isolation Level	**Read Uncommitted**	**Read Committed**	**Repeatable Read**	**Serializable**	**Snapshot**
Dirty Read	✓	✗	✗	✗	✗
Non-repeatable Read	✓	✓	✗	✗	✗
Phantom Read	✓	✓	✓	✗	✗
Locks Taken	Low	Moderate	High	Highest	None
Performance	Fastest	Balanced	Slower	Slowest	High (uses TempDB)

I typically use **Read Committed (default)**, and **Snapshot** when high concurrency is required with minimal blocking.

3. **What are indexed views and when should you use them?**

Answer:
An **indexed view** is a materialized view where the **results are stored on disk** with a clustered index.

Use when:

- Query involves complex **joins/aggregations** and is frequently used.
- You need **faster reads** and can tolerate slower writes (as changes to base tables update the view index).
- Under strict conditions (e.g., deterministic functions, schema binding).

Example:

```
sql
CopyEdit
CREATE VIEW SalesSummary WITH SCHEMABINDING AS
SELECT ProductId, SUM(Amount) AS Total
FROM dbo.Sales
GROUP BY ProductId;

CREATE UNIQUE CLUSTERED INDEX idx_SalesSummary ON SalesSummary(ProductId);
```

4. **How do you implement pagination in SQL Server?**

Answer:
Using `OFFSET` and `FETCH NEXT` in SQL Server 2012+:

```
sql
CopyEdit
SELECT Name, Salary
FROM Employees
ORDER BY Id
OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

For earlier versions:

```
sql
CopyEdit
WITH CTE AS (
  SELECT ROW_NUMBER() OVER (ORDER BY Id) AS RowNum, Name
  FROM Employees
)
SELECT * FROM CTE WHERE RowNum BETWEEN 11 AND 20;
```

5. **What is a performance execution plan? How do you read it?**

Answer:
An **execution plan** shows how SQL Server **executes a query** — the steps and their cost.

View it using:

- `SET SHOWPLAN_XML ON` / SSMS "Display Estimated Execution Plan" (Ctrl+L)
- "Actual Execution Plan" after running a query (Ctrl+M)

Key elements to watch:

- **Table Scans** (slow) vs. **Index Seeks** (fast)
- **Join types** (Nested Loops, Merge Join, Hash Join)
- **Operator costs (%)**
- **Missing index suggestions**

I regularly tune queries using the execution plan and add **covering indexes** where needed.

6. **How can you monitor long-running queries in SQL Server?**

Answer:
You can monitor them using:

- **Activity Monitor** in SSMS
- `sys.dm_exec_requests` to view currently running queries
- `sys.dm_exec_query_stats` to find historically slow queries
- **Query Store** (SQL Server 2016+)
- **Extended Events** for custom tracking
- **Profiler** or **third-party tools** like Redgate or SolarWinds

Example:

```
sql
CopyEdit
SELECT * FROM sys.dm_exec_requests
WHERE status = 'running' AND total_elapsed_time > 10000;
```

7. **What are SQL Server functions (scalar, table-valued)?**

Answer:

- **Scalar Function**: Returns a single value.

  ```
  sql
  CopyEdit
  CREATE FUNCTION fn_GetAge(@DOB DATE) RETURNS INT AS
  BEGIN
      RETURN DATEDIFF(YEAR, @DOB, GETDATE());
  END
  ```

- **Inline Table-Valued Function (iTVF)**: Returns a table using a single `SELECT`.

  ```
  sql
  CopyEdit
  CREATE
  ```

8. **How do you implement full-text search in SQL Server?**

Answer:
Steps:

1. Enable **Full-Text Search** feature in SQL Server.
2. Create a **Full-Text Catalog** and **Full-Text Index**.
3. Use `CONTAINS` or `FREETEXT` for querying.

Example:

```
sql
CopyEdit
SELECT * FROM Articles
WHERE CONTAINS(Content, '("AI" AND "Machine Learning")');
```

It's powerful for **natural language searches** on large text columns like blog content or product descriptions.

9. **What is the difference between IN, EXISTS, and ANY?**

Operator	**Checks**	**Best Used When**
`IN`	List of static or subquery values	Subquery is small or fixed
`EXISTS`	Subquery returns rows	Subquery is large and indexed
`ANY`	Compares with any returned value	With =, >, < comparisons

Examples:

```
sql
```

```
CopyEdit
-- IN
SELECT * FROM Employees WHERE DeptId IN (SELECT Id FROM Departments WHERE Name = 'HR');

-- EXISTS
SELECT * FROM Employees e WHERE EXISTS (SELECT 1 FROM Departments d WHERE d.Id = e.DeptId AND
d.Name = 'HR');

-- ANY
SELECT * FROM Employees WHERE Salary > ANY (SELECT Salary FROM Employees WHERE DeptId = 3);
```

10. **Have you used Window Functions (ROW_NUMBER(), RANK(), LEAD(), LAG())? Explain with examples.**

Answer:
Yes, I regularly use window functions for **analytics and pagination**.

- `ROW_NUMBER()`: Unique row number per partition/order
- `RANK()`: Gives same rank for ties (gaps in sequence)
- `DENSE_RANK()`: Same rank for ties (no gaps)
- `LEAD()`/`LAG()`: Access next/previous row's value

Example:

```
sql
CopyEdit
-- Get top 3 salaries per department
SELECT Name, DepartmentId, Salary,
       RANK() OVER (PARTITION BY DepartmentId ORDER BY Salary DESC) AS Rank
FROM Employees;

-- Compare current and previous salary
SELECT Name, Salary,
       LAG(Salary) OVER (ORDER BY HireDate) AS PreviousSalary
FROM Employees;
```

Window functions are great for **running totals, comparisons, pagination, and trends**.

✅.NET Core Interview Questions (All Levels)**

□ Basic Level

1. **What is .NET Core? How is it different from .NET Framework?**

Answer:
.NET Core is a **cross-platform**, **open-source**, and **lightweight** framework developed by Microsoft for building modern applications, including web, cloud, and console apps.

Key differences from .NET Framework:

- .NET Core supports **Windows, Linux, and macOS**, whereas .NET Framework is **Windows-only**.
- It provides **high performance and scalability**, especially for web APIs and microservices.
- **Side-by-side versioning** is supported in .NET Core, which is not available in the .NET Framework.
- It's also more modular, using NuGet packages for just what you need.

2. **What are the advantages of using .NET Core?**

Answer:

- **Cross-platform**: Runs on Windows, Linux, and macOS.
- **High performance**: Optimized for modern workloads and microservices.
- **Modular and lightweight**: Uses NuGet-based packages so apps only include what's needed.
- **Built-in Dependency Injection** support.
- **Fast development and deployment** via CLI, Docker, and Azure DevOps integration.
- **Side-by-side versioning**: Multiple versions of .NET Core can run on the same machine.
- **Open-source and active community support**.

3. **What is the use of Program.cs and Startup.cs files in a .NET Core project?**

Answer:

- Program.cs: Entry point of the application. It creates the **host**, configures logging, dependency injection, and web server (Kestrel).
  Example:

  ```
  csharp
  CopyEdit
  public static void Main(string[] args)
  {
     CreateHostBuilder(args).Build().Run();
  }
  ```

- Startup.cs: Contains application **startup logic**, like:
  - ConfigureServices(): Registers services (DI container).
  - Configure(): Sets up middleware (request pipeline).

Together, they define how the application initializes and handles HTTP requests.

4. **What is Kestrel in .NET Core?**

   Answer:
   Kestrel is a **cross-platform, high-performance web server** built into ASP.NET Core. It handles HTTP requests and serves as the default web server.

   In production, it's often used **behind a reverse proxy** like **IIS or NGINX** for features like SSL termination, load balancing, and header forwarding.

5. **What is a NuGet package and how do you use it?**

Answer:
A NuGet package is a **compiled library (DLL)** bundled with metadata, used to share reusable code across .NET projects. Examples: Newtonsoft.Json, EntityFrameworkCore.

To use:

- Via CLI: dotnet add package <PackageName>
- Or through **Visual Studio** → Manage NuGet Packages → Browse → Install

I often use NuGet for libraries like AutoMapper, Serilog, Swashbuckle (Swagger), etc.

6. **What are Middlewares in .NET Core?**

Answer:
Middlewares are components in the **HTTP request pipeline** that handle requests/responses. They can:

- Log requests
- Authenticate/authorize
- Serve static files
- Handle errors, etc.

Each middleware calls the **next one in the pipeline** or short-circuits it.

Example:

```
csharp
CopyEdit
app.UseAuthentication();
app.UseRouting();
app.UseAuthorization();
```

I've also written **custom middleware** for logging and request validation.

7. **What is Dependency Injection (DI)? How is it implemented in .NET Core?**

   Answer:
   Dependency Injection is a **design pattern** used to manage object dependencies, improving **testability, maintainability, and modularity**.

In .NET Core, it's built-in. You register services in ConfigureServices():

```
csharp
CopyEdit
services.AddScoped<IMyService, MyService>();
```

Then inject it via constructor:

```
csharp
CopyEdit
public MyController(IMyService service)
{
    _service = service;
}
```

I use DI to inject repositories, services, and configuration values.

8. **What are the common return types for Web APIs?**

Answer:

- IActionResult: Generic return type supporting various responses like Ok(), BadRequest(), etc.
- ActionResult<T>: Combines IActionResult and a specific data type for better type safety.
- Task<T> or Task<IActionResult>: Used for asynchronous methods.

Example:

```
csharp
CopyEdit
public async Task<ActionResult<User>> GetUser(int id)
{
    var user = await _userService.GetById(id);
    return Ok(user);
}
```

9. **How do you handle exceptions in .NET Core?**

Answer:

- At a **global level**, I use UseExceptionHandler() or UseDeveloperExceptionPage() in Startup.cs.
- For Web APIs, I implement **global exception handling middleware** that logs errors and returns proper HTTP status codes.
- I also use **try-catch blocks** within services for anticipated exceptions.

Example:

```
csharp
CopyEdit
app.UseExceptionHandler(errorApp =>
{
    errorApp.Run(async context =>
    {
        context.Response.StatusCode = 500;
        await context.Response.WriteAsync("An error occurred.");
    });
});
```

For production apps, I integrate with **Serilog** or **Application Insights** for centralized logging.

 Intermediate Level

1. **Explain the ASP.NET Core pipeline.**

Answer:
The ASP.NET Core pipeline is a sequence of **middleware components** that process HTTP requests and responses.

- Each middleware can perform operations on the request, pass it to the next middleware, and process the response.
- It's configured in Startup.cs → Configure() method.

Example pipeline:

```
csharp
CopyEdit
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});
```

In my projects, I use this pipeline to handle **logging**, **authentication**, **CORS**, **exception handling**, etc.

2. **How does routing work in .NET Core Web API?**

Routing in ASP.NET Core maps **incoming URLs to controller actions** using route templates.

- Defined via **attribute routing** (preferred in Web APIs):

  ```
  csharp
  CopyEdit
  [Route("api/[controller]")]
  public class ProductsController : ControllerBase
  {
      [HttpGet("{id}")]
      public IActionResult Get(int id) { ... }
  }
  ```

- Or using **conventional routing** in Startup.cs with MapControllerRoute.

.NET Core uses UseRouting() and UseEndpoints() to enable routing.

3. **What is Model Binding in .NET Core?**

Answer:
Model binding is the process where ASP.NET Core **automatically maps HTTP request data** (from query string, form data, body, etc.) to method parameters or models.

Example:

```
csharp
CopyEdit
[HttpPost]
public IActionResult Create([FromBody] Product product)
```

Here, the framework binds JSON data from the request body to the Product object.

Model binding supports:

- [FromQuery], [FromRoute], [FromBody], [FromForm], etc.

4. **What is the difference between IActionResult and ActionResult<T>?**

   Answer:

Feature	IActionResult	ActionResult<T>
Return Type	Flexible response types	Strongly typed + flexible response
Best Use	When returning various result types	When returning a specific model + status
Benefits	Good for custom control flow	Type safety + Swagger documentation

Example:

```
csharp
CopyEdit
public IActionResult Get() => Ok();
public ActionResult<Product> Get(int id) => Ok(product);
```

In my projects, I prefer ActionResult<T> for clarity and better API docs (e.g., Swagger).

5. **How do you implement authentication and authorization in .NET Core?**

Answer:
I've implemented both **JWT-based** and **cookie-based** authentication.

Steps:

1. **Authentication**: Configure JWT/cookie authentication in Startup.cs.
2. **Authorization**: Use [Authorize] attribute or policy-based access.

Example (JWT):

```
csharp
CopyEdit
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(...);

app.UseAuthentication();
app.UseAuthorization();
```

- I use claims-based roles and policies for fine-grained control:

```
csharp
CopyEdit
[Authorize(Roles = "Admin")]
```

6. **What are configuration providers in .NET Core?**

Answer:
Configuration providers are sources from which .NET Core reads configuration settings.

Examples:

- appsettings.json
- Environment variables
- Command-line arguments
- Secret Manager (for dev)
- Azure Key Vault (for production)

They're loaded in CreateHostBuilder():

```
csharp
CopyEdit
.ConfigureAppConfiguration((ctx, config) =>
{
   config.AddJsonFile("appsettings.json")
         .AddEnvironmentVariables();
});
```

I've used different providers depending on **deployment environments** and **security needs**.

7. **What is the role of appsettings.json? How do you use it for environment-specific settings?**

Answer:
appsettings.json is the primary **configuration file** for storing settings like connection strings, logging, and custom keys.

For environments:

- Use files like appsettings.Development.json, appsettings.Production.json
- Environment is set using ASPNETCORE_ENVIRONMENT variable.

Example:

```
json
CopyEdit
"ConnectionStrings": {
  "Default": "Server=.;Database=MyDb;Trusted_Connection=True;"
}
```

In Startup.cs:

```
csharp
```

```
CopyEdit
var conn = Configuration.GetConnectionString("Default");
```

I've used this for **multi-environment deployments** via Azure DevOps.

8. **How do you create a custom middleware in .NET Core?**

Answer:
A custom middleware handles HTTP requests in a custom way before passing to the next component.

Steps:

1. Create a class with InvokeAsync(HttpContext context, RequestDelegate next)
2. Register it in Configure()

Example:

```
csharp
CopyEdit
public class LoggingMiddleware
{
    private readonly RequestDelegate _next;
    public LoggingMiddleware(RequestDelegate next) => _next = next;

    public async Task InvokeAsync(HttpContext context)
    {
        // Log something
        await _next(context);
    }
}
```

Register:

```
csharp
CopyEdit
app.UseMiddleware<LoggingMiddleware>();
```

I've created middleware for **request logging**, **header validation**, and **global exception handling**.

9. **Explain the difference between transient, scoped, and singleton services.**

Answer:

Lifetime	**Description**	**Use Case Example**
Transient	New instance every time it's requested	Lightweight, stateless services
Scoped	Same instance per HTTP request	EF Core DbContext, unit of work
Singleton	One instance for the application's lifetime	Caching, configuration services

Registration:

```
csharp
CopyEdit
services.AddTransient<IMyService, MyService>();
services.AddScoped<IMyRepo, MyRepo>();
services.AddSingleton<ILogger, MyLogger>();
```

I carefully choose lifetimes to avoid issues like **DbContext scope conflicts**.

10. **What is Entity Framework Core? How do you use it in .NET Core?**

Answer:
EF Core is a lightweight ORM that allows interacting with databases using **C# classes instead of SQL**.

Common steps:

1. Create models and DbContext.
2. Configure DbContext in Startup.cs.
3. Use DbContext in services/repositories.
4. Perform CRUD with LINQ.

Example:

```
csharp
CopyEdit
services.AddDbContext<AppDbContext>(options =>
   options.UseSqlServer(Configuration.GetConnectionString("Default")));
```

In code:

```
csharp
CopyEdit
var products = _context.Products.Where(p => p.Price > 100).ToList();
```

I use **code-first migration**, **LINQ**, and **asynchronous queries** with EF Core in my projects.

 Advanced Level

1. **How do you implement logging in .NET Core using ILogger?**

Answer:
.NET Core provides built-in logging via `ILogger<T>`. It supports various providers like Console, Debug, EventLog, and third-party tools like **Serilog**, **NLog**, or **Application Insights**.

Setup:

```
csharp
CopyEdit
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;
    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

    public IActionResult Index()
    {
        _logger.LogInformation("Index page accessed.");
        return View();
    }
}
```

I usually configure logs in `appsettings.json` and in production, I use **Serilog** with **rolling file** or **Azure Log Analytics**.

2. **What is the Unit of Work and Repository Pattern? How have you implemented them?**

Answer:
The **Repository Pattern** abstracts data access logic, and **Unit of Work** coordinates multiple repositories using a shared DbContext to handle transactions.

Implementation:

- `IRepository<T>` with methods like `Add`, `Update`, `GetById`, etc.
- `IUnitOfWork` to group multiple repositories and manage commit/rollback.

```
csharp
CopyEdit
public interface IUnitOfWork : IDisposable
{
    IProductRepository Products { get; }
    IOrderRepository Orders { get; }
    int Complete();
}
```

In my project, this helped maintain **separation of concerns**, especially when working with complex transactions across entities.

3. **How do you handle database migrations in EF Core?**

Answer:
I use **code-first migrations** to evolve the database schema alongside the models.

Commands:

```
bash
CopyEdit
dotnet ef migrations add AddProductTable
dotnet ef database update
```

Steps:

1. Define changes in models.
2. Run `Add-Migration`.
3. Run `Update-Database`.

For multiple environments, I maintain **separate connection strings** and use **migration history table** for version control.

4. **Explain asynchronous programming in .NET Core using async and await.**

Answer:
Async programming in .NET Core improves scalability by **freeing up threads during I/O-bound operations** like DB or API calls.

Example:

```
csharp
CopyEdit
public async Task<IActionResult> GetProducts()
{
    var products = await _productService.GetAllAsync();
    return Ok(products);
}
```

- `async` makes the method asynchronous.
- `await` pauses execution until the task completes.

This is critical in web APIs to avoid thread starvation under load.

5. **What is CQRS and have you used it in any projects?**

Answer:
CQRS (Command Query Responsibility Segregation) separates read (queries) and write (commands) logic into different models.

- **Commands**: Change state (create, update, delete).
- **Queries**: Return data only, no side effects.

In my project, I used CQRS with **MediatR**:

- Queries and commands were handled by separate handlers.
- Improved performance, especially for read-heavy modules.

Used along with **Event Sourcing** in one module to decouple services.

6. **How do you implement caching in ASP.NET Core?**

Answer:
I've implemented both **in-memory caching** and **distributed caching**.

In-Memory Caching:

```
csharp
CopyEdit
services.AddMemoryCache();

public class ProductService
{
    private readonly IMemoryCache _cache;
    public ProductService(IMemoryCache cache)
    {
        _cache = cache;
    }

    public Product GetProduct(int id)
    {
        return _cache.GetOrCreate($"product_{id}", entry => {
            entry.AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5);
            return _repository.GetById(id);
        });
    }
}
```

Also used **Distributed Redis Cache** for load-balanced apps hosted on Azure.

7. **What are filters in ASP.NET Core (Action Filter, Exception Filter)?**

Answer:
Filters allow code to run **before or after** action methods.

- **Action Filters**: Run before/after controller actions (e.g., logging, validation).
- **Exception Filters**: Handle unhandled exceptions globally.
- **Authorization Filters**: Enforce security policies.

Example:

```
csharp
CopyEdit
public class LogActionFilter : IActionFilter
{
    public void OnActionExecuting(ActionExecutingContext context) { /* log */ }
    public void OnActionExecuted(ActionExecutedContext context) { /* log */ }
}
```

I've used **global exception filters** to centralize error handling and **custom action filters** for logging and auditing.

8. **How do you secure Web APIs in .NET Core?**

Answer:
I usually implement **JWT token-based authentication** with role-based access control.

Steps:

- Use `Microsoft.AspNetCore.Authentication.JwtBearer`
- Configure authentication in `Startup.cs`
- Apply `[Authorize]` attribute and custom policies

Example:

```
csharp
CopyEdit
[Authorize(Roles = "Admin")]
public IActionResult GetAllUsers() { ... }
```

Additionally:

- Use **HTTPS redirection**
- Protect sensitive data with **Azure Key Vault**
- Enable **CORS policies**
- Validate inputs to prevent **injection attacks**

9. **Have you used SignalR? What is it and what are its use cases?**

Answer:
Yes. **SignalR** is a library for **real-time communication** between client and server using WebSockets (or fallback protocols).

Use cases:

- Live chat apps
- Real-time dashboards/notifications
- Collaborative apps (e.g., whiteboards, docs)

In my project, I used SignalR to push real-time order status updates to the frontend without polling.

Setup:

```
csharp
CopyEdit
services.AddSignalR();
app.UseEndpoints(endpoints => {
    endpoints.MapHub<ChatHub>("/chathub");
});
```

10. **Explain how CI/CD works with .NET Core applications.**

Answer:
CI/CD automates **build, test, and deployment** processes. I've used **Azure DevOps pipelines** for .NET Core apps.

CI Process:

- Trigger on push/PR
- Restore packages → Build solution → Run unit tests → Generate artifacts

CD Process:

- Pick artifact from CI
- Deploy to Azure Web App or IIS
- Use stages: Dev → QA → Production
- Approvals & rollback configured

YAML Example:

```
yaml
CopyEdit
trigger:
  branches:
    include: [main]

pool:
  vmImage: 'windows-latest'

steps:
- task: UseDotNet@2
  inputs:
    packageType: 'sdk'
    version: '7.x'

- task: DotNetCoreCLI@2
  inputs:
    command: 'build'
    projects: '**/*.csproj'
```

This setup gave us **fast, repeatable, and traceable deployments**.

-

3. Azure DevOps & Azure Cloud

 Basic Level

- What is Azure DevOps and its main components?

Answer:
Azure DevOps is a **DevOps lifecycle toolset** by Microsoft for **planning, developing, testing, and delivering** applications.

Main components:

- **Azure Boards**: Work item tracking (Agile, Scrum, Kanban).
- **Azure Repos**: Git repositories with branch policies.
- **Azure Pipelines**: CI/CD automation.
- **Azure Test Plans**: Manual and exploratory testing.
- **Azure Artifacts**: Package management (NuGet, npm, etc.).

I've used Boards and Pipelines daily in my projects for agile tracking and CI/CD automation.

- What are pipelines in Azure DevOps?

Answer:
Pipelines are workflows that **automate the build, test, and deploy** process.

- **CI (Continuous Integration):** Compiles code, runs tests.
- **CD (Continuous Deployment):** Pushes artifacts to environments (Dev, QA, Prod).
- Pipelines can be built using:
  - **Classic editor (GUI-based)** or
  - **YAML (code-based)**.

I've used both formats for .NET Core build and deploy processes.

- What is a service connection in Azure DevOps?

Answer:
A service connection securely stores credentials to connect Azure DevOps to **external services**, like:

- Azure subscriptions (for deployments)
- GitHub, DockerHub, etc.

Types:

- Azure Resource Manager
- GitHub personal access token
- Docker registry credentials

I typically use **Azure Resource Manager** connections to deploy apps to **App Service** or **Key Vault**.

- Explain the concept of a YAML-based pipeline.

Answer:
YAML pipelines define CI/CD logic using .yml files stored in the repository, offering **version control**, **reusability**, and **flexibility**.

Example:

```
yaml
CopyEdit
trigger:
  branches: [main]

jobs:
- job: Build
  pool:
    vmImage: 'windows-latest'
  steps:
  - task: DotNetCoreCLI@2
    inputs:
      command: 'build'
      projects: '**/*.csproj'
```

I've used YAML pipelines to **build, test, and deploy** .NET Core apps with **Dev, QA, Prod environments**.

□ **Intermediate Level**

- How do you integrate your .NET Core app with an Azure DevOps pipeline?

Answer:
Steps:

1. Push code to Azure Repos/GitHub.
2. Create a YAML pipeline or use the Classic editor.
3. Use built-in tasks:
   - DotNetCoreCLI@2 for build, restore, publish, test.
   - PublishBuildArtifacts@1 to generate artifacts.
4. Add a release pipeline to deploy to App Service.

I’ve used this setup for **CI/CD of APIs and front-end apps**, deploying to **Azure App Services** or containers.

- What is the difference between Classic and YAML pipelines?

Feature	Classic Pipeline	YAML Pipeline
UI/UX	GUI-based (drag-and-drop)	Code-based (YAML file)
Version Control	Not stored in repo	Stored and versioned with source code
Flexibility	Less flexible for branching	Easy multi-branch or condition setup

I’ve used Classic for quick proof-of-concepts and YAML for long-term projects requiring **DevOps as Code**.

- How do you use Azure Artifacts and NuGet packages?

Answer:
Azure Artifacts lets you host and consume NuGet, npm, Maven, etc.

Use cases:

- Host internal libraries/packages.
- Share versioned modules across teams.

Steps:

- Create a feed in Azure Artifacts.
- Push NuGet packages using dotnet nuget push.
- Add the feed URL in nuget.config.

I've used Artifacts to **share custom libraries** across microservices.

 Advanced Level

- How do you secure secrets and credentials in Azure DevOps?

Answer:

1. Use **Azure Key Vault** and reference secrets in YAML.
2. Use **Pipeline Secrets** (Library → Variable Groups → Mark as secret).
3. Avoid hardcoding secrets in YAML; instead, use environment variables.

Example:

```
yaml
CopyEdit
env:
  ConnectionString: $(MySecret)
```

I integrate Azure Key Vault for **connection strings**, **API keys**, and use **RBAC** to limit access.

- How do you set up CD to Azure App Services?

Answer:

1. Create a **service connection** to Azure.
2. Use **release pipeline** or YAML job with AzureWebApp@1 task:

```
yaml
CopyEdit
- task: AzureWebApp@1
  inputs:
    azureSubscription: 'MyConnection'
    appType: 'webApp'
    appName: 'my-dotnet-app'
    package: '$(System.DefaultWorkingDirectory)/**/*.zip'
```

I configure **slot-based deployments** for production with **approval gates** in Dev → QA → Prod flow.

- What is infrastructure as code? Have you used ARM templates or Bicep?

Answer:
Infrastructure as Code (IaC) automates provisioning of cloud infrastructure using code.

Options in Azure:

- **ARM Templates** (JSON-based)
- **Bicep** (simplified syntax for ARM)
- **Terraform** (cross-platform)

I've used **ARM templates and Bicep** to deploy:

- App Service plans
- Key Vaults
- SQL databases
- Storage accounts

Benefits include **repeatable infrastructure**, **version control**, and **consistency** across environments.

- How would you rollback a failed deployment?

  Answer:

  Enable **deployment slots** and use **swap on success** strategy.

  Use **Release Pipeline gates** for approval and **manual intervention**.

  Implement **automatic rollback** via health check failure or **custom scripts**.

  Maintain **previous deployment artifacts** for rollback.

  I configure pipelines to deploy to **staging slots first**, validate, then swap to production.

✅4. Programming Languages & OOPs (C#, Python)

C# & OOPs

- What is the difference between abstract class and interface?

Feature	Abstract Class	Interface
Inheritance	Single inheritance	Multiple inheritance
Members with logic	Can have implemented methods	Cannot have implementations (until C# 8 default methods)
Fields/Constructors	Can have fields and constructors	Cannot contain fields or constructors
Use Case	When base class has **shared code**	When **contracts** are needed without implementation

Example:

csharp
CopyEdit
```
abstract class Animal {
   public abstract void Speak();
   public void Eat() => Console.WriteLine("Eating...");
}

interface IWalkable {
   void Walk();
}
```

Usage:
I use **abstract classes** for base business logic and **interfaces** for defining behavior (like IDisposable, IRepository<T>).

- What are value types and reference types?
- **Value Types** (e.g., int, float, bool, struct):
  - Stored on the **stack**
  - Copied by value
  - Fast but limited in size
- **Reference Types** (e.g., class, string, array, object):
  - Stored on the **heap**
  - Copied by reference
  - Allow shared state

Example:

csharp
CopyEdit
```
int a = 5;
int b = a; // b is a copy

Person p1 = new Person();
Person p2 = p1; // both refer to the same object
```

- What is polymorphism and where have you used it?

Polymorphism allows different classes to be treated through a common interface or base class.

- **Compile-time (Overloading):** Same method name, different signatures
- **Runtime (Overriding):** Base class method overridden in derived class using virtual and override

Example:

csharp
CopyEdit

```
public class Animal {
   public virtual void Speak() => Console.WriteLine("Animal speaks");
}
public class Dog : Animal {
   public override void Speak() => Console.WriteLine("Dog barks");
}
```

Usage:
I used polymorphism for implementing **business rules** via interfaces (e.g., IValidator) and strategy patterns.

- Explain the use of async and await.

Answer:
async and await enable **asynchronous programming** using **Task-based** APIs.

- Improves responsiveness
- Avoids thread blocking
- Common in web/API calls and I/O operations

Example:

```
csharp
CopyEdit
public async Task<string> GetDataAsync() {
   var response = await httpClient.GetStringAsync("https://api.example.com/data");
   return response;
}
```

Usage:
In .NET Core APIs, I use async for:

- **Database access** via EF Core
- **API integrations**
- **File I/O operations**

- What is boxing and unboxing?

Answer:

- **Boxing:** Converting a **value type to object** type (stored in heap)
- **Unboxing:** Converting back to **value type** from object

Example:

```
csharp
CopyEdit
int num = 123;
object obj = num; // Boxing
int result = (int)obj; // Unboxing
```

Impact:
Boxing can affect performance; hence I avoid it in **loops and collections** (use generics like List<int>).

✅**6. Software Design Principles (SOLID, CQRS)**

 SOLID Principles

- What is the Single Responsibility Principle? Provide a real example.

Answer:
Single Responsibility Principle (SRP) states that a class should **have only one reason to change**, i.e., it should only do one thing.

Bad Example (violates SRP):

```
csharp
CopyEdit
public class Invoice {
   public void CalculateTotal() { ... }
   public void SaveToDatabase() { ... }
   public void GeneratePDF() { ... }
}
```

Good Example (SRP):

```
csharp
CopyEdit
public class InvoiceCalculator { public void CalculateTotal(Invoice invoice) { ... } }
public class InvoiceRepository { public void Save(Invoice invoice) { ... } }
public class InvoicePDFGenerator { public void Generate(Invoice invoice) { ... } }
```

Where I used it:
In my e-commerce project, I split invoice logic into **calculation**, **persistence**, and **report generation** to maintain **modularity and testability**.

- Explain Open/Closed Principle.

Answer:
Open/Closed Principle: Classes should be **open for extension but closed for modification**.

Example:

```
csharp
CopyEdit
public interface IDiscount {
   decimal ApplyDiscount(decimal amount);
}

public class StudentDiscount : IDiscount { ... }
public class SeniorDiscount : IDiscount { ... }

public class BillingService {
   public decimal GetFinalAmount(IDiscount discount, decimal amount) {
      return discount.ApplyDiscount(amount);
   }
}
```

You can add new discount strategies without modifying BillingService.

- What is the Interface Segregation Principle? When do you use it?

Answer:
A class should **not be forced to implement interfaces it doesn't use**. Break down large interfaces into **smaller, specific ones**.

Bad Example:

csharp
CopyEdit
```
public interface IWorker {
   void Work();
   void Eat();
}
```

Good Example:

csharp
CopyEdit
```
public interface IWorkable { void Work(); }
public interface IFeedable { void Eat(); }
```

Where I used it:
In a background job system, I separated IProcessJob, IScheduleJob, and ILogJob to keep responsibilities clean.

- How does the Dependency Inversion Principle work in a layered architecture?

Answer:
DIP states:

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Use **interfaces** and **dependency injection** to decouple layers.

Example:

csharp
CopyEdit
```
// Application Layer
public class OrderService {
   private readonly IOrderRepository _repo;
   public OrderService(IOrderRepository repo) { _repo = repo; }
}

// Infrastructure Layer
public class OrderRepository : IOrderRepository { ... }
```

In Startup.cs:

csharp
CopyEdit
```
services.AddScoped<IOrderRepository, OrderRepository>();
```

This enables **unit testing** and makes swapping implementations easy (e.g., switch from SQL to MongoDB).

□ **CQRS & Patterns**

- What is the main benefit of CQRS over CRUD?

Answer:
CQRS (Command Query Responsibility Segregation) separates **read and write logic** into different models.

Benefits:

- Better scalability (reads and writes can scale independently)
- Clearer domain logic separation
- Optimized data models per operation
- Enables **event sourcing** and **audit logging**

I've used CQRS in modules with **high read traffic**, like dashboard reporting.

- Where would you apply CQRS in a .NET Core app?

Answer:

- In **API layers**: separate Command and Query handlers for each use case.
- Commands update DB, Queries fetch DTOs.
- Use MediatR for dispatching.

Example:

plaintext
CopyEdit
➢POST /CreateOrder → CommandHandler → DbContext.Save()
➢GET /GetOrderById → QueryHandler → DTO response

Used in my inventory project for **order creation and lookup** separation.

- Explain Mediator Pattern and how it relates to CQRS.

Answer:
The **Mediator pattern** defines a central object to **encapsulate communication** between components.

In .NET Core:

- **MediatR** is commonly used to implement CQRS.
- It allows sending commands and queries via IMediator.Send().

Example:

```
csharp
CopyEdit
await _mediator.Send(new CreateUserCommand { Name = "Saurabh" });
```

Benefits:

- Decouples controllers from business logic
- Supports clean architecture and testability

- Difference between Command Handler and Query Handler.

Aspect	Command Handler	Query Handler
Purpose	Performs write operations	Fetches and returns data
Side Effects	Yes (e.g., DB updates, events)	No side effects (read-only)
Returns	Void or identifier/result	DTO or read model
Examples	CreateOrderHandler, UpdateUser	GetUserByIdHandler, GetOrdersList

Where I used it:
In user management, I separated **update profile (command)** from **get user details (query)**.

- What is the Unit of Work Pattern?

Answer:
Unit of Work (UoW) ensures that multiple **related DB operations** are executed as a **single transaction**.

In EF Core:
DbContext acts as Unit of Work.

Benefits:

- Maintains data consistency
- Reduces the number of database calls
- Coordinates multiple repositories

Example:

```
csharp
CopyEdit
using (var context = new AppDbContext()) {
   var user = new User { Name = "Akanksha" };
   var order = new Order { Amount = 500 };

   context.Users.Add(user);
   context.Orders.Add(order);
   context.SaveChanges(); // one atomic transaction
}
```

✅**7. Web Deployment (IIS, GitHub)**

 Deployment via IIS

- What are the steps to deploy a .NET Core web app to IIS?

Answer:
To deploy a .NET Core web application to IIS:

1. **Publish the app**
   Use Visual Studio or CLI:

   ```
   bash
   CopyEdit
   dotnet publish -c Release -o ./publish
   ```

2. **Install .NET Core Hosting Bundle** on the IIS server
   - It configures the reverse proxy and adds the ASP.NET Core Module.
3. **Create a site in IIS:**
   - Go to IIS Manager → Sites → Add Website
   - Point **Physical Path** to publish folder
4. **Set application pool:**
   - Use **No Managed Code**
   - Ensure app pool is using **Integrated pipeline**
5. **Set permissions:**
   - Grant read + execute permission to **IIS_IUSRS** or **Network Service**
6. **Configure bindings:**
   Add port or domain name as required (e.g., localhost:5001, or myapp.local).

Usage:
I've deployed REST APIs and admin portals using IIS with **custom domains**, **HTTPS**, and **logs enabled**.

- How do you configure a hosting environment in IIS?

Answer:

You configure environment-specific settings using:

- **ASPNETCORE_ENVIRONMENT** variable
  Set in:
  - System Environment Variables
  - Or in web.config:

    ```
    xml
    CopyEdit
    <environmentVariables>
      <add name="ASPNETCORE_ENVIRONMENT" value="Production" />
    </environmentVariables>
    ```

- The environment name must match values like Development, Staging, Production in your code (e.g., Startup.cs).

Where I used this:
I configure **different appsettings for staging and production** using this approach.

- How do you troubleshoot a 500 Internal Server Error after deployment?

Answer:

1. **Enable stdout logging** in web.config:

   ```
   xml
   CopyEdit
   <aspNetCore stdoutLogEnabled="true" stdoutLogFile=".\logs\stdout" />
   ```

2. Check logs at logs\stdout\*
3. Ensure:
   - Hosting bundle is installed
   - Application pool has permissions
   - appsettings.Production.json is not missing/invalid
   - Required files in /publish folder exist (like web.config, .dll)
4. Use **Event Viewer** for detailed exceptions.

Pro Tip:
Set launchSettings.json to match IIS port to replicate issues locally.

 Git & GitHub

- What is the difference between git pull and git fetch?

Command	**Description**
git fetch	Gets latest changes from remote, doesn't merge
git pull	Fetch + Merge into current branch automatically

Example:

```
bash
CopyEdit
git fetch origin
git merge origin/main
# vs
git pull origin main
```

Best Practice:
I use git fetch before rebasing to **avoid unintentional merges** in shared branches.

- How do you resolve merge conflicts?

Answer:

1. When you run:

   ```
   bash
   CopyEdit
   git pull origin main
   ```

   and conflict occurs, Git marks it with <<<<<<<, =======, >>>>>>>.

2. Open file → resolve manually or use a GUI tool (e.g., VSCode Source Control)
3. Add and commit:

```
bash
CopyEdit
git add .
git commit -m "Resolved conflict"
```

4. Continue rebase or push.

Where I used this:
During team sprints, resolving conflicts in shared services like Startup.cs or model files is common.

- What is the purpose of .gitignore?

Answer:

.gitignore prevents certain files/folders from being tracked by Git.

Common exclusions:

```
pgsql
CopyEdit
bin/
obj/
*.user
*.log
*.suo
.vscode/
appsettings.Development.json
```

Why it matters:
Prevents clutter, protects sensitive info, and keeps repo clean.

- How do you revert a commit?

Answer:

- To **undo a local commit** (but keep changes):

```
bash
CopyEdit
git reset --soft HEAD~1
```

- To **undo and discard changes:**

```
bash
CopyEdit
git reset --hard HEAD~1
```

- To **revert a pushed commit** (safe for shared branches):

bash
CopyEdit
```
git revert <commit-hash>
```

Where I used this:
If a build breaks after push, I revert the last commit using revert for safe rollback.
