**TECHNICAL SKILLS:**

1. **What is Object-Oriented Programming (OOP)? And why it is important?**

Object-Oriented Programming (OOP) is a programming paradigm that organizes code around objects rather than actions. It uses concepts like **classes** and **objects** to model real-world entities. OOPS integrates real-world concepts into programming such as inheritance, polymorphism, encapsulation and abstraction.

1. **Why do we need to use OOPs? Or Important of OOPS ?**

OOP (Object-Oriented Programming) simplifies programming by organizing code into reusable objects, reducing redundancy through inheritance, and ensuring data security with encapsulation. It also enhances program flexibility using polymorphism, making the code more adaptable and maintainable.

1. **What is Class?**

A class is a logical structure , and object are the physical implementations of that structure.

A class is a template or blueprint from which objects are created,

that defines the properties(attributes) and behaviours(methods) that object of that class will have.

1. **What is Objects?**

In object-oriented programming, an object is an instance of a class. In real-world an entity that has state(attributes) and its behavior (methods) is known as an object

For Example:

A Car is an object. It has states (name, color, model) and its behavior (changing gear, applying brakes)

1. **What is the difference between a class and an object?**

A **class** is like a blueprint for creating objects, while an **object** is an instance of a class. For example, "Dog" is a class that defines characteristics like breed, size, and behaviors like barking. When you create a "GoldenRetriever" as a specific dog, that becomes an object.

1. **What is the difference between Primitive and Non-Primitive datatypes?**

Non-primitive data types are called **reference types** because they refer to objects.

The main difference between **primitive** and **non-primitive** data types are:

* Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
* Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.

Other Differences:

* A primitive type has always a value, while non-primitive types can be null.
* A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
* The size of a primitive type depends on the data type, while non-primitive types have all the same size.

**e.g for Primitive:**

These are predefined by Java and represent basic values:

1. **int** - Represents integer values.  
   Example: int age = 25;  
   Here, age is a primitive data type holding the integer value 25.

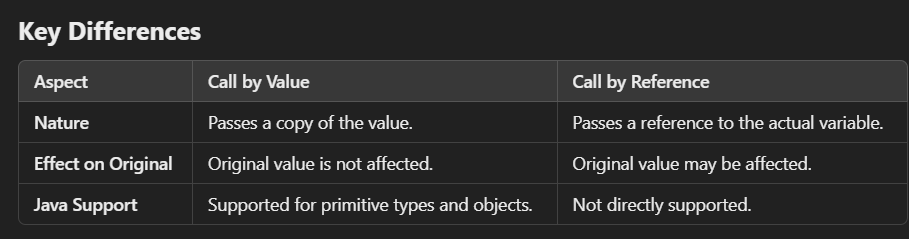
**e.g for Non-Primitive:**

**String** - A sequence of characters.  
Example: String name = "John Doe";  
Here, name is a non-primitive data type because String is a class, and it provides various methods like .length() to perform operations.

1. **Call by value and Call by reference:**

In **call by value**, a copy of the actual value is passed to the method. Changes made to the parameter inside the method do not affect the original value outside the method.

In **call by reference**, the reference (address) of the variable is passed to the method. Changes made to the parameter inside the method affect the original variable.



1. **What is JDK?**

A complete toolkit for developing the Java Application or programs. Whenever we are supposed to write the code, we need to have JDK for sure. It allows you to compile , run, debug java programs.

It includes (JDK) = JRE + JVM

1. **What is JRE?**

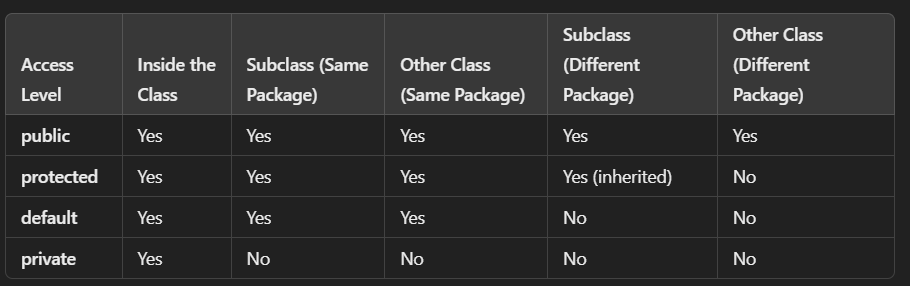
 **Definition**: JRE is the runtime environment required to run Java applications. It is a package that have all the neccesary libraries that we need to run the code.

1. **What is JVM?**

 **Definition**: The JVM is a virtual machine it is a component of JRE which actually executes the code. It’s responsible for executing Java bytecode, making Java platform-independent.

### What are access specifiers and what is their significance?

Access specifiers in Java control the visibility and accessibility of classes, methods, and variables. There are four main types:



 **public** - Accessible from any other class.

 **protected** - Accessible within the same package and by subclasses in different packages.

 **default** (no modifier) - Accessible only within the same package.

 **private** - Accessible only within the same class.

1. **Define Encapsulation.**

Encapsulation is the bundling of data and methods that operate on the data into a single unit (class).

*->*By providing only a setter or getter method, you can make the class **read-only or write-only**.

->It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

class Student {

// Private field

private String name;

// Public getter method to access the private field

public String getName() {

return name;

}

// Public setter method to set the value of the private field

public void setName(String name) {

this.name = name;

}

}

public class Main {

public static void main(String[] args) {

// Create an object of Student class

Student student = new Student();

// Use setter method to set the value of name

student.setName("John");

// Use getter method to get the value of name

System.out.println("Student Name: " + student.getName());

}.

}

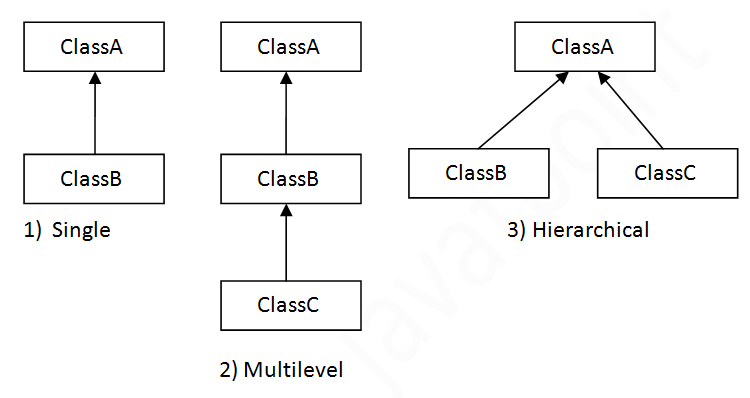
1. **What is Inheritance?**

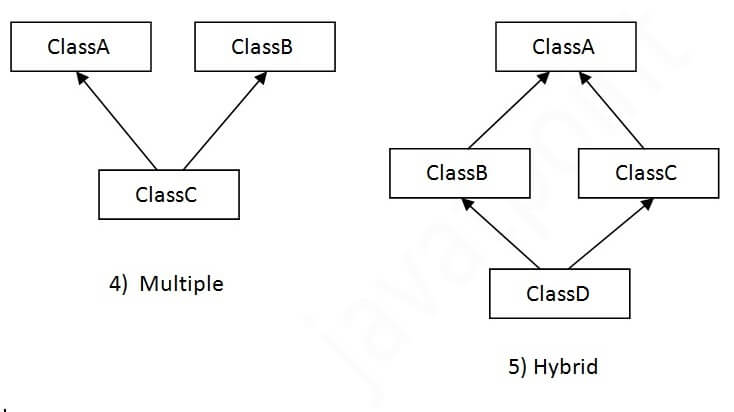
- Inheritance is one of the OOPS concept wherein one class can inherit the data members of another class by using Parent Child relation.

- 'extends' keyword is used to inherit another class.

- 'implements' keyword is used to inherit another interface.

**Reusability:**Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class.





## Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

1. **Explain Polymorphism.**

**->**Polymorphism means the ability of objects to take on many forms.

->”poly” means many “morphism” means way to represent.

->there are two types of polymorphism

1. Compile-time polymorphism -Achieved via method overloading.
2. Run-time polymorphism- Achieved via method overriding

->It occurs during inheritance.

**Method Overloading:**

->Method overloading means having two or more methods in a class with the same name but different arguments or parameters.

->argument , return type can be varied.

->eg: int myMethod(int x)

float myMethod(float x)

double myMethod(double x, double y)

**Method Overriding:**

-> If subclass has the same method as declared in the parent class, it is known as method overriding.

->Method overriding is used for runtime polymorphism

->if you want to check the method is overriden or not just put @override above it.

#### Rules for Java Method Overriding

* The method must have the same name as in the parent class
* The method must have the same parameter as in the parent class.

// Superclass

class Animal {

// Method to be overridden

public void sound() {

System.out.println("Animals make sound");

}

}

// Subclass

class Dog extends Animal {

// Overriding the sound method

@Override

public void sound() {

System.out.println("Dog barks");

}

}

// Subclass

class Cat extends Animal {

// Overriding the sound method

@Override

public void sound() {

System.out.println("Cat meows");

}

}

// Main class to test method overriding

public class Main {

public static void main(String[] args) {

Animal myAnimal; // Reference of parent class

myAnimal = new Dog(); // Object of Dog

myAnimal.sound(); // Calls the overridden method in Dog

myAnimal = new Cat(); // Object of Cat

myAnimal.sound(); // Calls the overridden method in Cat

}

}

1. **What is Abstraction?**

Abstraction is the process of hiding the implementation details and showing only the essential features of the object. It helps to reduce programming complexity and effort.

// Abstract class

abstract class Animal {

// Abstract method (no implementation)

abstract void sound();

// Concrete method (implementation provided)

void eat() {

System.out.println("This animal eats food.");

}

}

// Concrete subclass

class Dog extends Animal {

@Override

void sound() {

System.out.println("The dog barks.");

}

}

public class Main {

public static void main(String[] args) {

Animal myDog = new Dog(); // Abstract class reference, subclass object

myDog.sound(); // Output: The dog barks.

myDog.eat(); // Output: This animal eats food.

}

}

1. **What is a constructor?**

A constructor in java is a special method used to initialize the object. It is called automatically when an object of a class its created. The name of the constructor must be same as the class name, and constructor does not have a return type.

”This” keyword directly access the reference variable

There are 2 types of constructor : Default Construtor, Paramterized Constructor

**class Student {**

**// Instance variables**

**String name;**

**int age;**

**// Default constructor**

**Student() {**

**System.out.println("Default constructor called");**

**name = "Unknown";**

**age = 0;**

**}**

**// Parameterized constructor**

**Student(String n, int a) {**

**System.out.println("Parameterized constructor called");**

**name = n;**

**age = a;**

**}**

**// Method to display information**

**void display() {**

**System.out.println("Name: " + name + ", Age: " + age);**

**}**

**public static void main(String[] args) {**

**// Creating objects**

**Student s1 = new Student(); // Default constructor**

**s1.display(); // Output: Name: Unknown, Age: 0**

**Student s2 = new Student("Alice", 20); // Parameterized constructor**

**s2.display(); // Output: Name: Alice, Age: 20**

**}**

**}**

1. **What is the 'super' keyword used for?**

* Super keyword in java is a reference variable which is used to refer the immediate parent of the class object.
* Super keyword is used to invoke(call) the parent class method or constructor.

1. Access Parent class variable:

class Parent {

String name = "Parent Name";

}

class Child extends Parent {

String name = "Child Name";

void display() {

System.out.println("Child name: " + name); // Access child class variable

System.out.println("Parent name: " + super.name); // Access parent class variable

}

}

public class Main {

public static void main(String[] args) {

Child obj = new Child();

obj.display();

}

}

1. Access parent class methods:

**super** can invoke the parent class method if it is overridden in the child class.

class Parent {

void showMessage() {

System.out.println("Message from Parent Class");

}

}

class Child extends Parent {

void showMessage() {

System.out.println("Message from Child Class");

}

void display() {

super.showMessage(); // Call parent class method

showMessage(); // Call child class method

}

}

public class Main {

public static void main(String[] args) {

Child obj = new Child();

obj.display();

}

}

1. Access parent class constructor:

When creating a child class object, the parent class constructor is called first. Use super() to explicitly call a specific parent class constructor.

class Parent {

Parent() {

System.out.println("Parent class default constructor");

}

Parent(String message) {

System.out.println("Parent constructor says: " + message);

}

}

class Child extends Parent {

Child() {

super("Hello from Child to Parent!"); // Call parameterized parent constructor

System.out.println("Child class constructor");

}

}

public class Main {

public static void main(String[] args) {

Child obj = new Child(); // Creates an object of Child

}

}

1. **What is a static method?**

A static method belongs to the class rather than any specific instance. It can be called on the class itself without creating an object of the class.

class Calculator {

// Static method to perform addition

static int add(int a, int b) {

return a + b;

}

}

public class Main {

public static void main(String[] args) {

// Call static method without creating an object

int result = Calculator.add(10, 20);

System.out.println("Sum: " + result);

}

}

* Static method cannot access instance variables

class Example {

int instanceVar = 10; // Instance variable

// Static method

static void display() {

// System.out.println(instanceVar); // Error: Cannot access instanceVar in static context

System.out.println("Static methods cannot directly access instance variables.");

}

}

public class Main {

public static void main(String[] args) {

Example.display();

}

}

Why Use Static Methods?

Utility Methods: For operations like Math.sqrt(), Collections. Sort().

1. **What is the final method?**

A final method is a method that cannot be overridden by subclasses. Once defined in a class, its implementation cannot be changed in any subclass.

Final Keyword:

* Final keyword can be used with variable, method or class.
* It prevent its content from being modified.
* When declared with class, it prevent the class from being extended .

1. **What is an abstract class?**

An abstract class is a class that cannot be instantiated on its own and is meant to be subclassed. It may contain abstract methods, which are declared but not implemented in the abstract class.

1. **What is an interface?**

-> Another way to achieve abstraction in java is with Interface.

->An interface is an completely “abstract class” that is used to group a related method With empty bodies.

-> Interface cannot be instantiated.

While you cannot instantiate an interface directly, you can use implementing classes, anonymous inner classes, or lambda expressions to achieve the desired behavior.

1. **What is the 'this' keyword used for?**

The 'this' keyword is used to refer to the current object within an instance method or constructor. It is used to differentiate between instance variables and local variables with the same name.

class Student {

String name;

int age;

// Constructor

Student(String name, int age) {

this.name = name; // Refers to the instance variable

this.age = age;

}

void display() {

System.out.println("Name: " + this.name + ", Age: " + this.age);

}

}

public class Main {

public static void main(String[] args) {

Student student = new Student("Alice", 20);

student.display(); // Output: Name: Alice, Age: 20

}

}

1. **What is garbage collection?**

Garbage collection is the process of automatically reclaiming memory occupied by objects that are no longer in use by the program. It helps in preventing memory leaks and managing memory efficiently.

1. **What is a deep copy and a shallow copy?**

**Deep:**

A deep copy creates a new object with entirely separate memory. Changes in the copied object do not affect the original object

**Shallow:**

A shallow copy creates a new reference to the same object in memory. Any changes in one object are reflected in the other.

class abcd {

    private *int* a;

*int* b;

    abcd(*int* a, *int* b) {

*// Parameterized Constructor*

*this*.a = a;

*this*.b = b;

*// this refers to the current object in the class not the paramter in the*

*// constructor*

    }

    abcd(*String* a, *int* b) {

    }

    abcd() {

*// Default Constructor*

    }

    abcd(abcd obj) {

*// Copy constructor*

*// Deep copy (The created object have new reference, so changing value will not*

*// affect the original object)*

*this*.a = obj.a;

*this*.b = obj.b;

    }

    public static *void* main(*String*[] args) {

        abcd obj = *new* abcd(5, 10);

        abcd obj2 = *new* abcd();

        obj2.a = 30;

        obj2.b = 40;

*// deep copy (copy constructor)*

        abcd obj3 = *new* abcd(obj);

        obj3.a = 15;

        obj3.b = 20;

        System.out.println(obj.a);

        System.out.println(obj3.a);

        System.out.println(obj.b);

*// shallow copy*

        abcd obj4 = obj;

        System.out.println(obj4.a);

        obj4.a = 50;

        System.out.println(obj.a);

        System.out.println(obj.b);

        obj.a = 10;

        obj2.a = 20;

        System.out.println(obj4.a);

        System.out.println(obj2.a);

    }

}

class jegan extends abcd {

    public *void* meth() {

        abcd obj = *new* abcd(40, 50);

*// System.out.println(obj.a);*

*// since a is private , cant access in a sub class*

    }

}

1. **What is a singleton class?**

A singleton class is a class that allows only one instance of itself to be created and provides a global point of access to that instance.

1. **What is the purpose of the 'extends' keyword in Java?**

The 'extends' keyword is used to create a subclass that inherits properties and behavior from a superclass. It establishes an inheritance relationship between classes.

1. **What is the purpose of the 'final' keyword in Java?**

The 'final' keyword is used to restrict the modification of classes, methods, and variables. It can be applied to classes to prevent subclassing, methods to prevent overriding, and variables to create constants.

1. **What is Copy Constructor VS Assignment Constructor ?**

* The copy constructor and the assignment operator (=) both are used to initialize one object using another object.
* Copy constructor allocates separate memory to both objects i.e. existing object and newly created object.
* Assignment operator does not allocate new memory for the newly created object

1. **What is Exception Handling ?**

->Exception Handling is used to manage runtime error.

->It allows the programmer to catch error and handle them on their own.

->In try block you can write code that might cause an exception.

->Catch block is used to handle those exceptions.

->Finally block always executes whether an exception occurs or not.

public class BasicExceptionHandling {

public static void main(String[] args) {

try {

// Code that may throw an exception

int result = 10 / 0; // This will throw an ArithmeticException

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

// Catch block to handle ArithmeticException

System.out.println("Error: Division by zero is not allowed.");

} finally {

// Finally block always executes

System.out.println("This will always be executed.");

}

System.out.println("Program continues after exception handling.");

}

}

1. Thread:

->A thread in java is a small, lightweight part of a program that can run independently.

->It allows program to perform multiple task at the same time this is called multithreading.

**Thread Life Cycle**

Threads go through several states during their life:

1. New: The thread is created but not started yet.

2. Runnable: The thread is ready to run but waiting for CPU

time.

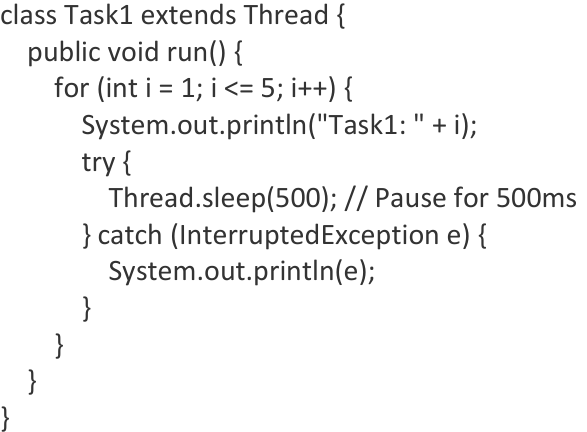
3. Running: The thread is actively executing.

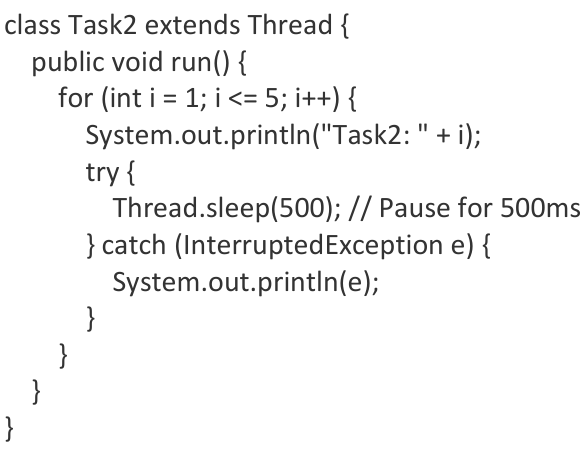
4. Blocked/Waiting: The thread is waiting for a resource or

signal.

5. Terminated: The thread has completed execution.

Eg:





public class Main {

public static void main(String[] args) {

Task1 t1 = newTask1();

Task2 t2 = newTask2();

t1.start(); // Start Task1

t2.start(); // Start Task2

}

}



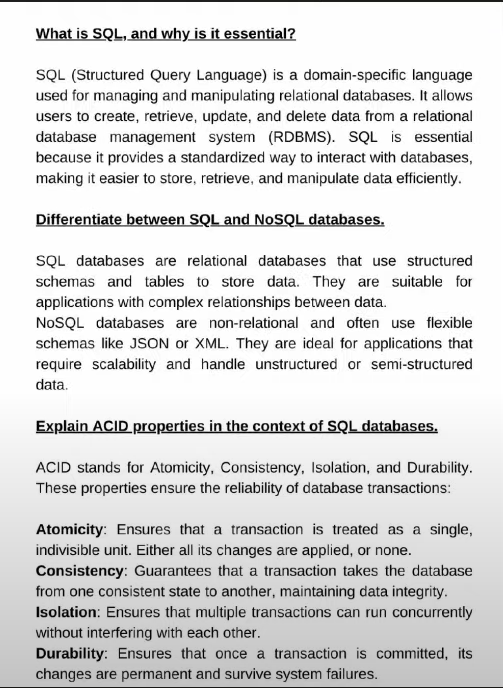
**DBMS:**

**What is DBMS (Database Management System)?**

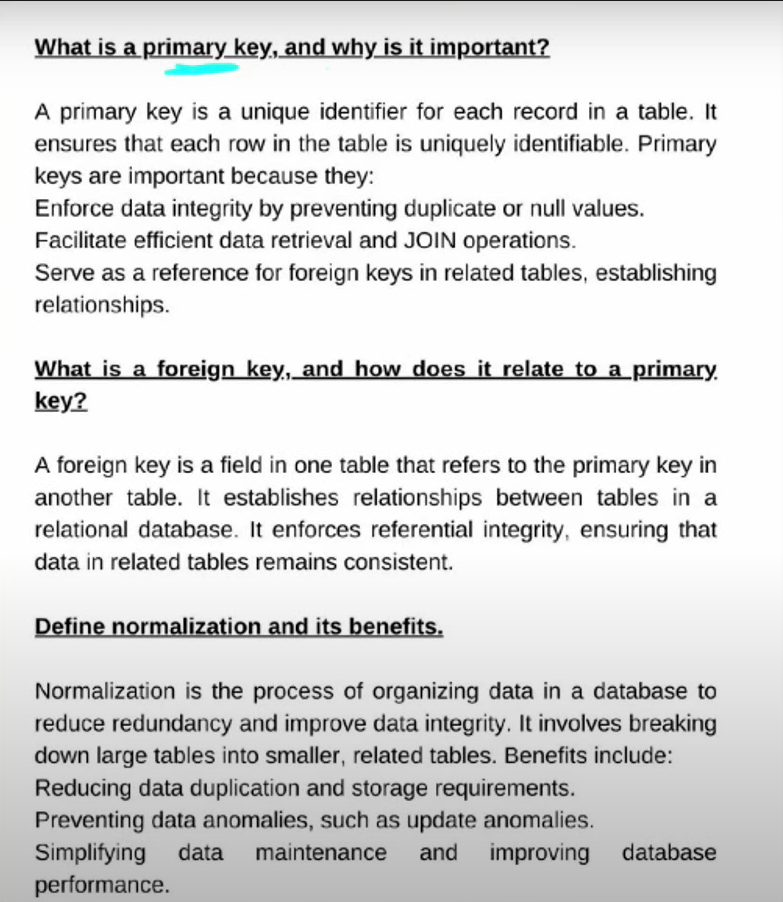
A Database Management System (DBMS) is software that enables users to create, manage, and interact with databases efficiently. It acts as an interface between the end-users and the database, handling the storage, retrieval, and modification of data while ensuring its security and integrity.

Types of DBMS:

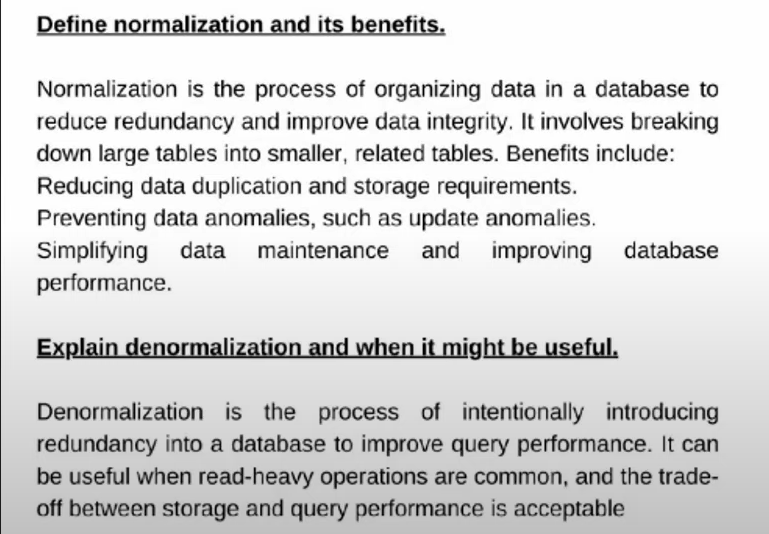
1. **Relational DBMS (RDBMS)**
2. **Hierarchical DBMS**
3. **Network DBMS**
4. **NoSQL DBMS**

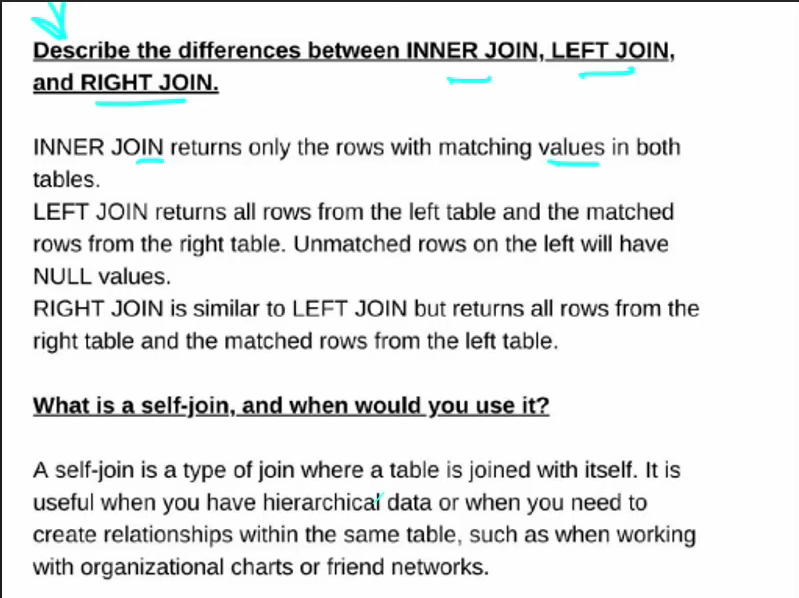
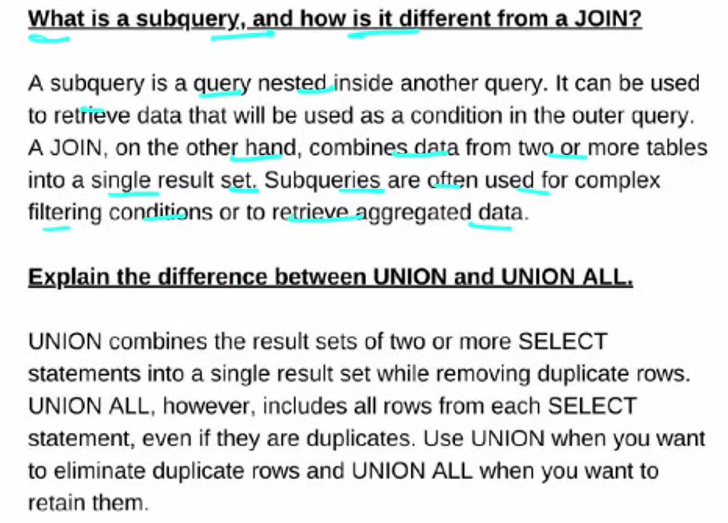


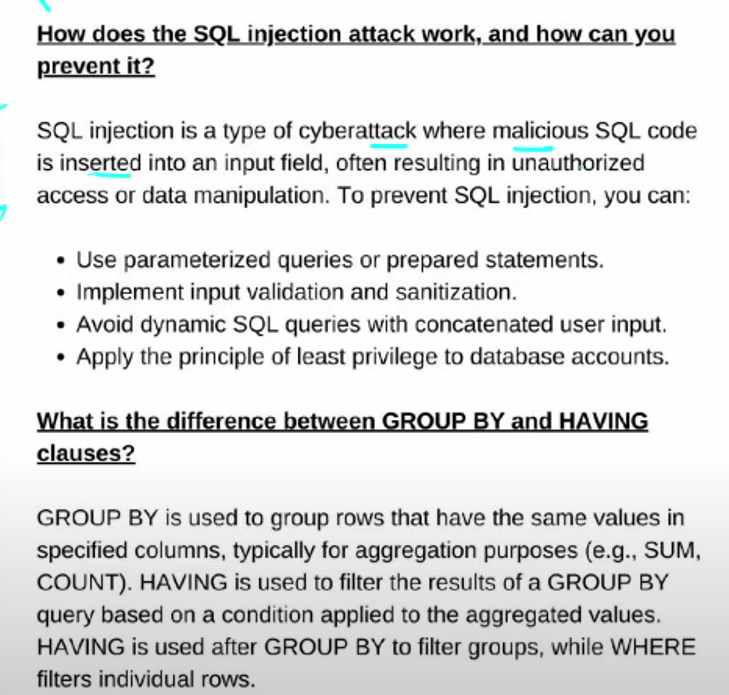
.

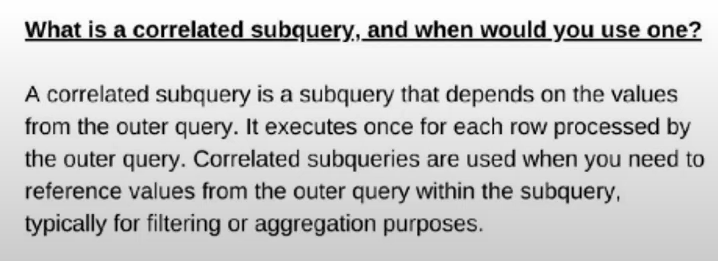


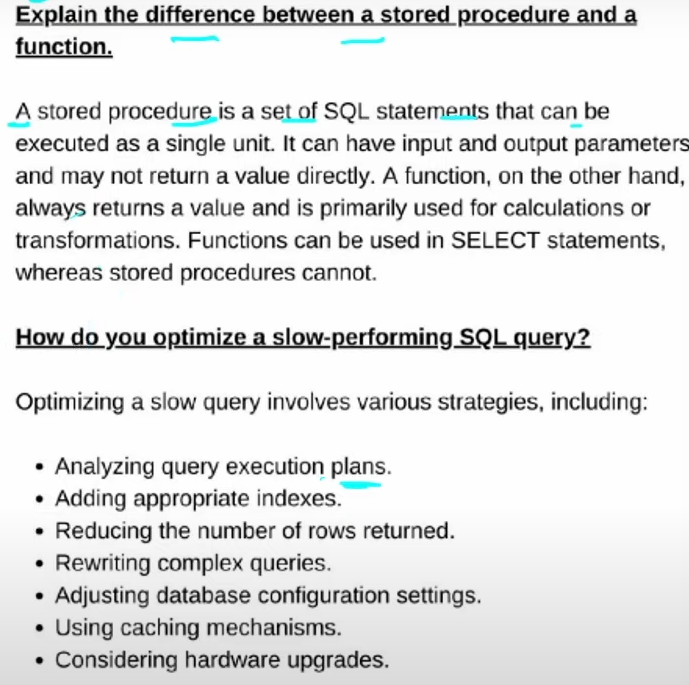
7 , 8 :

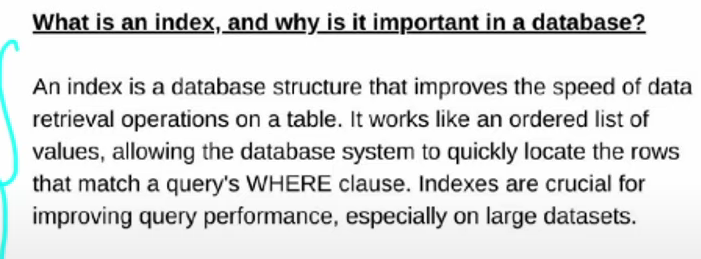


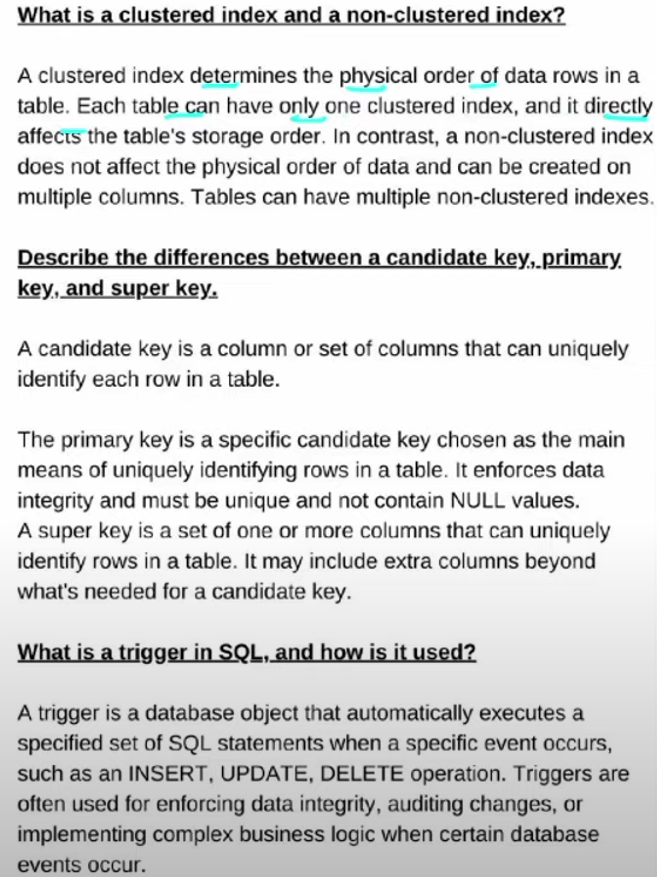
 

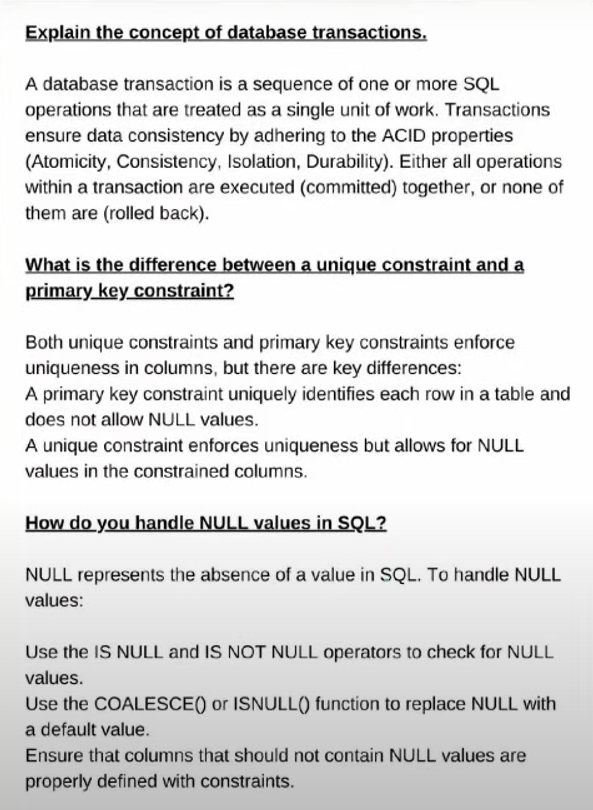


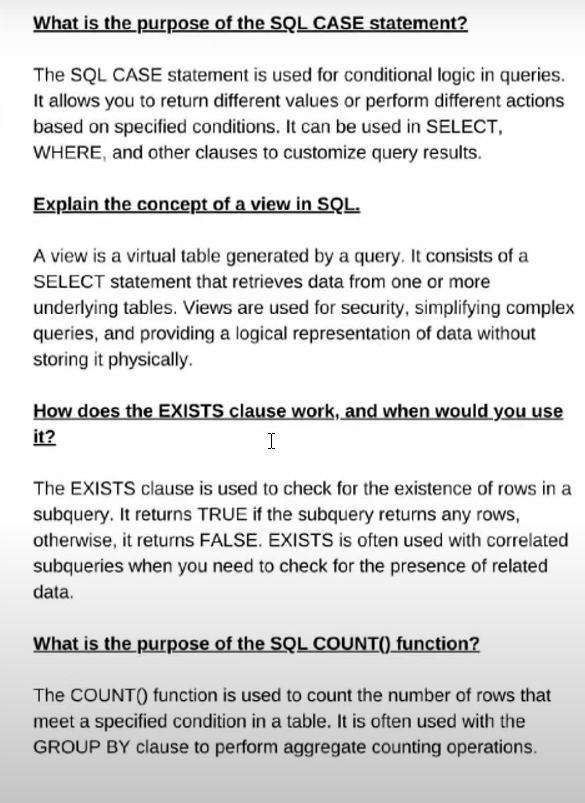


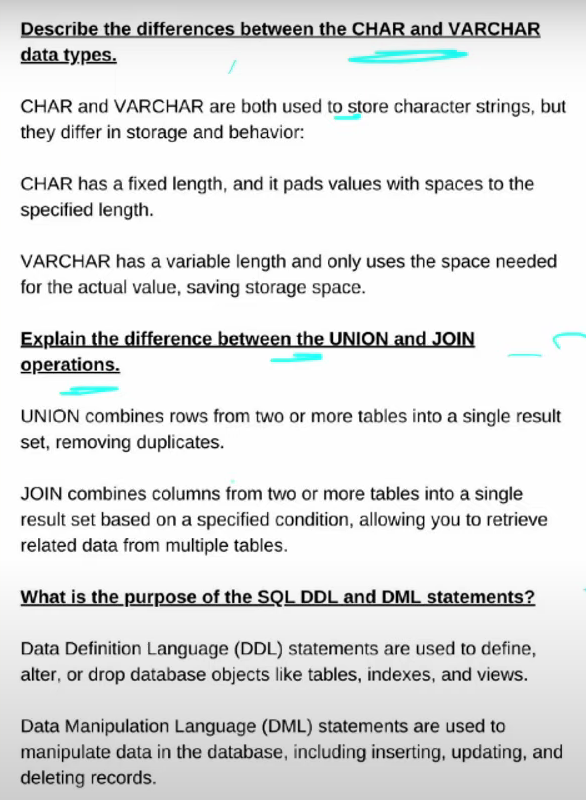


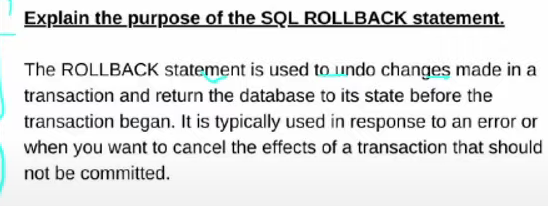


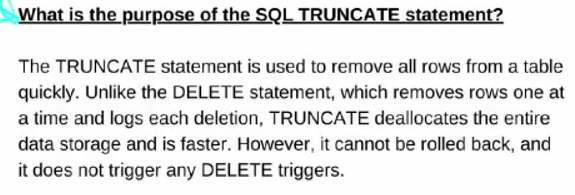






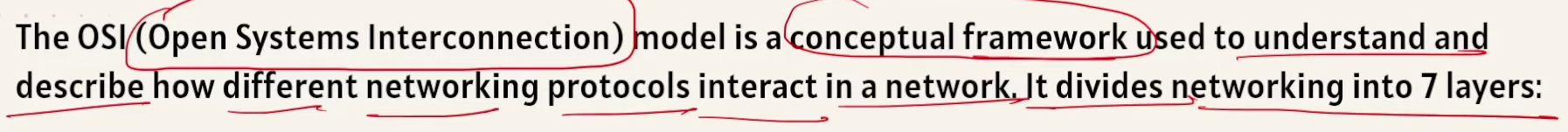






**Wipro:**

**OSI LAYER:**

****

**The OSI model is a theoretical framework to standardize network communication, while the TCP/IP model is a practical implementation used in the Internet.**

1. **Physical Layer**:  
   "This is the foundation of the OSI model, dealing with the actual hardware and physical connections like cables, switches, and electrical signals. It’s about how data is transmitted as bits (0s and 1s) over physical media."
2. **Data Link Layer**:  
   "This layer is responsible for ensuring error-free data transfer between devices within the same network. It uses MAC addresses to identify devices and organizes data into frames for transmission. For example, a switch works at this layer."
3. **Network Layer**:  
   "The Network Layer handles routing and IP addressing. It decides the best path for data to travel across different networks. Routers work at this layer, using IP addresses to forward packets."
4. **Transport Layer**:  
   "This layer ensures complete and reliable data transfer between devices. It uses protocols like TCP (for reliability) and UDP (for speed). For example, it ensures that all parts of a file download reach the device correctly."
5. **Session Layer**:  
   "This layer manages and controls communication sessions between devices. It establishes, maintains, and terminates sessions. For example, it keeps track of multiple tabs open in a web browser, ensuring they don’t mix data."
6. **Presentation Layer**:  
   "The Presentation Layer translates data into a format the application layer can understand. It handles encryption, compression, and data translation, like converting a file into a readable format."
7. **Application Layer**:  
   "This is where users interact with the network. It provides services like web browsing, email, and file transfer. Applications like Chrome, Outlook, and FTP clients work at this layer."

You can summarize like this:  
"The first three layers (Physical, Data Link, Network) deal with how data moves across the network. The next two (Transport, Session) manage reliable communication. The last two (Presentation, Application) ensure data is usable and interacts with users."

1. **Packets:**

When you send a message or a file over the internet, it’s too big to send all at once. So, it’s broken into small pieces, called packets. Each packet contains a piece of the message, the address it’s going to (like on a real envelope), and instructions on how to put all the pieces back together when they arrive.

**Explanation:**  
Packets make communication faster and more reliable. If one packet is lost, only that small part needs to be resent, not the entire message.

1. **Routers:**

Routers are devices that forward data packets between computer networks.

They examine the destination IP address of a packet and use routing tables to determine the best path for the packet to reach its destination.

1. **Gateways:**

* A **gateway** is like a translator or a bridge between two different networks.
* On the internet, different networks might use different "languages" or rules to send data. Gateways ensure packets can move smoothly between these networks, even if they speak different "languages."

1. **Protocols:**

* Protocols are like the rules of the road for internet communication. They tell packets how to behave, like when to stop, where to go, and how to say “hello.”
* Protocols like TCP/IP are rules that all devices follow to communicate on the internet. They ensure that packets are sent, received, and reassembled correctly, like following the same rules in a game.

1. **Switches:**

* A Switch operates at the data link layer and is used to connect device within a local area newtwork(LAN). It used MAC addresses to forward data to the correct destination device within the same network.
* In a local network (like at home), a switch connects devices like computers, printers, and phones. It ensures data goes to the right device.

1. **What is the difference between a switch and a router?**

* A switch connects devices in the same network and forwards data using MAC addresses.
* A router connects different networks and decides the best path for data using IP addresses.
* Example: A switch is like an office phone system; a router is like a travel agent for different cities.

1. **IP address and Port Number:**

An **IP address (Internet Protocol address)** is a unique numerical identifier assigned to every device connected to a network. It ensures that data sent over the internet reaches the correct device. IP addresses work like home addresses, allowing devices to locate and communicate with each other.

A **port number** is a numerical identifier within a device that specifies the service or application to which data should be directed. It works alongside the IP address to route data to the correct application.

1. **Routers Basics**

**IP Addressing**: Routers use IP addresses to decide where to send data.

**Route**: The best path for sending data.

**Gateway**: The device that connects your network to another network (e.g., your home router connecting to the internet).

**Protocols**: Rules like RIP or OSPF that routers follow to decide the best path.

1. **NAT (Network Address Translation):**

Router often perform NAT, which translates the Private IP address within a network, into a single public IP address, it allowing to connect multiple devices to share a single public IP address for internet access.

1. **OSPF (Open Shortest Path First) :**

OSPF is a link-state routing protocol used in larger and complex networks. It uses the Dijikstra algorithm to find the shortest path.

1. **RIP:**

**RIP (Routing Information Protocol)** is one of the simplest distance-vector routing protocols used in computer networks to help routers dynamically share and update routing tables.

1. **DHCP: (**Dynamic Host Configuration Protocol)

* DHCP is a network management protocol used to automatically assign IP addresses and other network configuration settings to devices on a network.
* This allows devices to communicate with each other and access resources like the internet without the need for manual IP configuration.

**Example:** When you connect to Wi-Fi, DHCP gives your device an IP address.

1. **DNS: (Domain Name System)**

* DNS is used to convert the domain name of the websites to their numerical IP address.
* DNS translates the IP address(like 172.197.255.0) into human readable domain names like ([www.google.com](http://www.google.com)), which computer use to identify each other on the network.
* DNS acts like a phonebook for the internet.
* **Example:** When you type www.amazon.com in your browser, DNS resolves it to an IP address to load the website.

1. **What is Network?**

A network is a system that connect two or more electronic devices such as mobile, computers is used to allowing them share resources and communicate with each other.

1. **CSMA/CD:( Carrier Sense Multiple Access / Collision Detection)**

It is used in wired networks like Ethernet, it detects and resolves the collisions after they happen.

**How it works:**

1. **Carrier Sense:** Before sending data, a device checks if the network is free (i.e., no other device is sending data).
2. **Multiple Access:** Many devices share the same communication medium (like an Ethernet cable).
3. **Collision Detection:** If two devices send data at the same time, their signals collide. The devices detect this collision, stop transmitting, and wait for a random amount of time before trying again.

**Real-World Example:**

Imagine a group of people trying to talk on a single phone line. Before speaking, they listen to ensure no one else is talking. If two people speak at the same time (collision), they stop, wait, and then retry.

1. **CSMA/CA: (Carrier Sense Multiple Access / Collision Avoidance)**

It is used in wireless networks like Wi-Fi , it avoids collision by reserving the network before sending data.

**How it works:**

1. **Carrier Sense:** A device listens to the network to check if it’s free before sending data.
2. **Collision Avoidance:** The device reserves the network by sending a small signal (called RTS – Request to Send). If the network responds with CTS (Clear to Send), the device sends its data.
3. **Acknowledgment:** After data transmission, the receiver sends an acknowledgment to confirm successful delivery.

**Real-World Example:**

Imagine students in a classroom raising their hands before speaking. The teacher gives permission to one student (avoiding collision), and others wait for their turn.

1. **VPN:**

A VPN creates a secure and encrypted connection between your device and a remote server, masking your IP address and ensuring privacy.

1. **What is diff b/w IP address and MAC address:**

**IP ADDRESS:**

* IP Address Works at the Network Layer (Layer 3)
* It's like a **postal address** for a device on a network.
* It uses the **IP address** to find the destination network (global path).

**MAC ADDRESS:**

* MAC Address Works at the Data Link Layer (Layer 2)
* It's like the **name tag** or **permanent ID** for a device’s network interface card (NIC).
* Once the data reaches the local network, it uses the **MAC address** to identify the exact device.

1. **What is ARP: (Address Resolution Protocol)** Is used in only LAN:

* ARP is a communication protocol.
* It is essential for mapping an IP address to a MAC address.

**How Does ARP Work?**

1. When a device knows the destination's **IP address** but not its **MAC address**, it uses ARP to find the MAC address.
2. The device sends a broadcast ARP request across the local network, asking, "Who has this IP address?"
3. The device with the matching IP address responds with its MAC address.
4. The requesting device saves this MAC address in its ARP table for future use.
5. **What is IP address:**

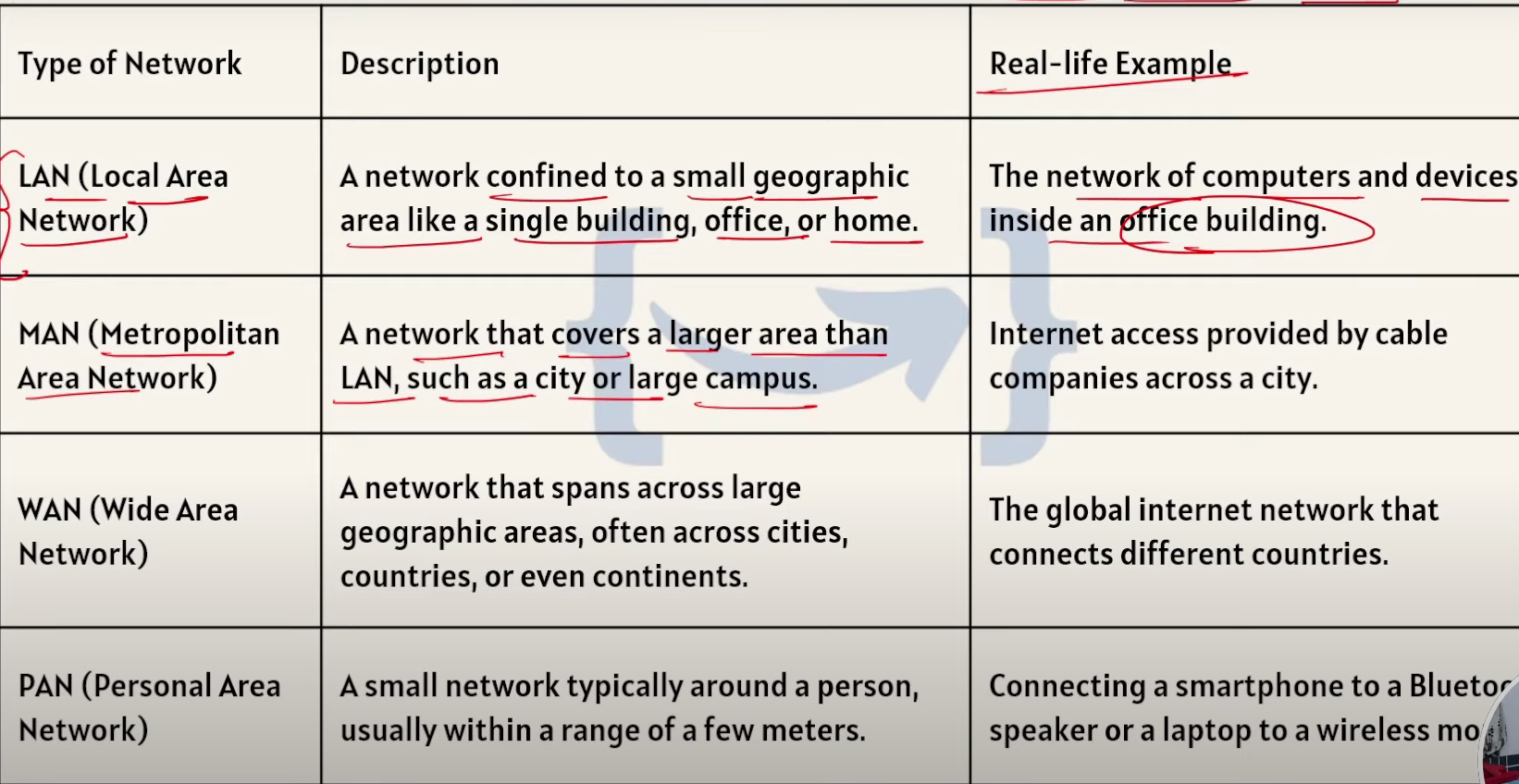
An **IP address** (Internet Protocol address) is a unique identifier assigned to every device connected to a network. It helps devices communicate with each other over the internet or local networks by identifying the source and destination of data packets.

**Types of IP Addresses**

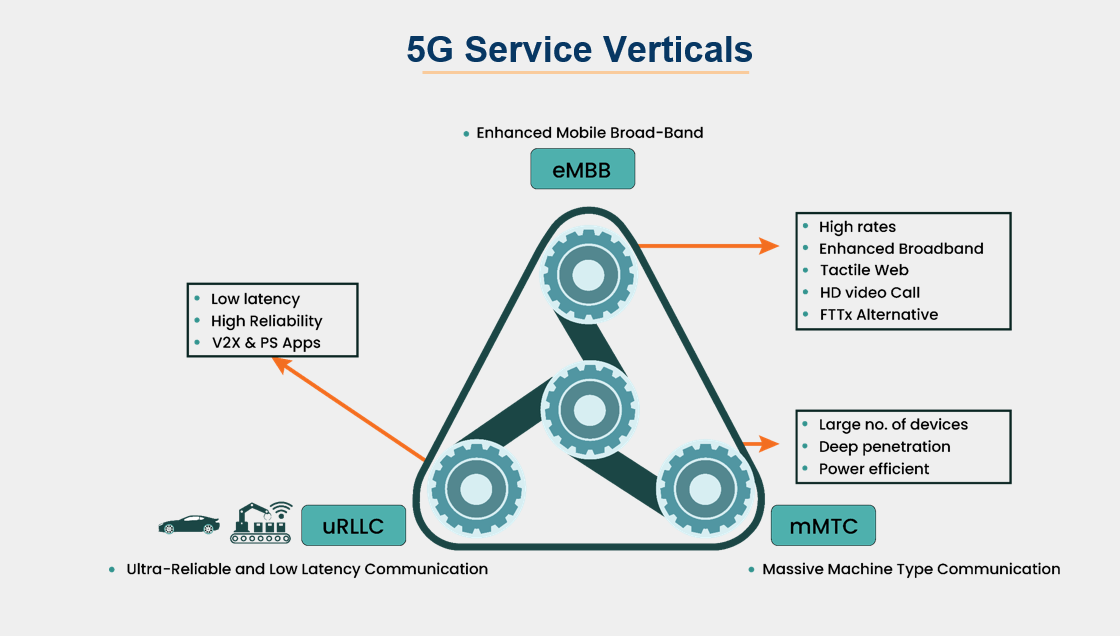
1. **IPv4 (Internet Protocol Version 4)**:
   * Format: Four numbers separated by dots (e.g., **192.168.1.1**).
   * Each number ranges from 0 to 255.
   * Common and widely used.
   * Example: A home router often has an IPv4 address like **192.168.0.1**.
2. **IPv6 (Internet Protocol Version 6)**:
   * Format: Eight groups of hexadecimal numbers separated by colons (e.g., **2001:0db8:85a3:0000:0000:8a2e:0370:7334**).
   * Developed because IPv4 addresses are limited and cannot accommodate the growing number of devices.
   * More efficient and secure.
3. **Subnet Mask:**

* A subnet mask is a 32-bit number used to define which portion of an IP address belongs to the network and which portion identifies a device (host) within that network.
* For example, if the IP address is 192.168.1.3 and the subnet mask is 255.255.255.0, it means the first three octets (192.168.1) represent the network, and the last octet (3) represents the device in that network.
* This helps in identifying devices on the same network and facilitates efficient communication within and across network.

1. **LAN, WAN, MAN, PAN:**



1. **5G verticals:**

****

1. **What is 5G ?**

5G is the fifth generation of mobile network technology. It is much faster, more reliable, and supports more devices than previous generations. It enables advanced technologies like the Internet of Things, autonomous vehicles, and smart cities by providing high-speed data and low latency.

Why 5G:

High Speed, Low latency , Massive connectivity.

1. **What is AMF ?**

AMF is responsible for managing the access and mobility of the user equipment (UE), including tasks like authentication, authorization, and tracking the user's location in the network.

AMF is the entry point for UE, handling initial registration and mobility.

1. **What is and work of AMF in 5G?**

* The **Access and Mobility Management Function (AMF)** is a key network function in the 5G.
* It manages how a User device (UE) connects to 5G network and stay connected while moving.
* AMF ensures how a User device (UE) connects to 5G and stays connected while moving, like when you use video calling while travelling. It handles tasks like registration, Mobility, connectivity and reachability for making sure the connection is smooth and uninterrupted.

1. **Diff b/w handoff and AMF: (Access and Mobility Management Function)**

* "Handoff is the process where an active session, like a phone call or video streaming, is transferred between gNodeBs as a user moves between cells. The AMF is part of the 5G core and manages the broader process, including signaling, location tracking, and coordinating handoffs.
* For example, while handoff switches your call to the next tower, AMF ensures this transition is smooth, secure, and works across the network."

1. **What is base station:**

A base station is a fixed location that transmits and receives radio signals to and from wireless devices, such as mobile phones, tablets, and laptops. Base stations are a central point for wireless devices to connect to the cellular network and communicate with each other.

1. **What is GNODEB: ( g- Next Generation NodeB)**

* **gNodeB (Next Generation Node B)** is the base station in **5G networks.**
* It connects **User Equipment (UE)**, like smartphones or IoT devices, to the 5G core network, facilitating wireless communication for services like high-speed internet, voice, and data transfer.

Key functions:

* + - Radio communication
    - Connection setup and handover
    - Data transfer

Key features:

* + 5G connectivity
  + Massive MIMO

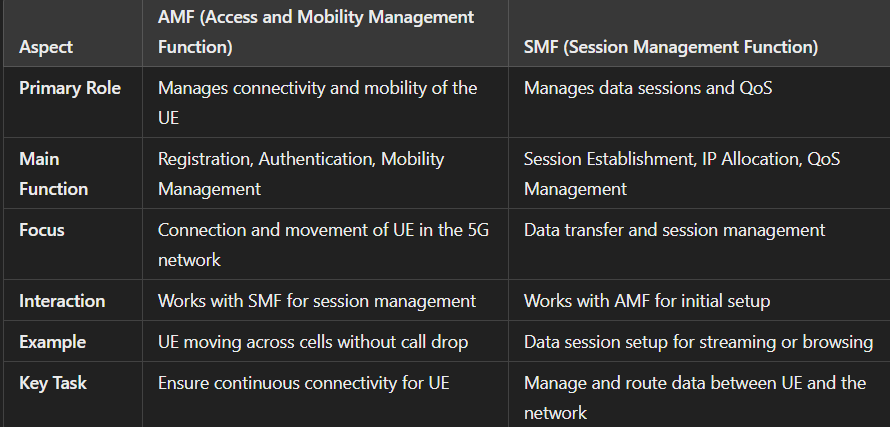
1. **SMF: (Session Management Function)** manages data session

* SMF is a key component or **critical part** of 5G network.
* It main job of SMF is maintain a **life cycle of user data connection** which includes session creation, modification and termination.
* It creates, modifies, and terminates sessions, assigns IP addresses, enforces policies, and works with other network components to provide seamless connectivity and data management.

**Real-World Example of SMF:**

* Let’s say you are using your phone to stream a movie.
  + **Session Creation**: When you connect, **SMF** creates the session, allocates an IP address, and sets up the path for streaming.
  + **Session Modification**: While watching, if you move from one area to another or increase the video quality, **SMF** modifies the session to accommodate these changes.
  + **Session Termination**: Once you finish watching the movie, **SMF** terminates the session, freeing up network resources.

1. **Diff b/w AMF and SMF:**

****

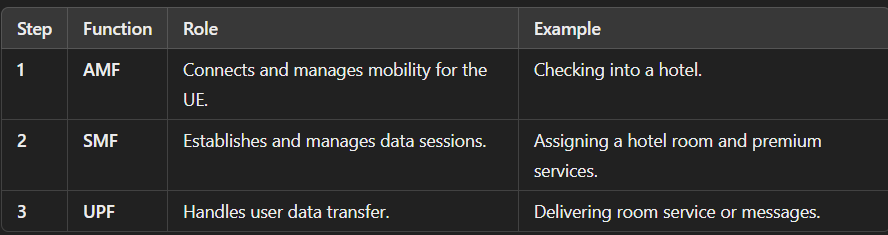
1. **UPF: (User Plane Function)**

It is Third step : Once AMF and SMF set up the control and session,

UPF forwards and routes user data packets between the UE and external networks. It ensures efficient delivery of data for applications like video streaming, browsing, or voice calls.

**E.g : UPF acts like the hotel’s delivery service that brings food, parcels, or messages to and from your room. It makes sure everything is routed efficiently without delays.**

1. **Summary for AMF, SMF, UPF:**



AMF is the gatekeeper, SMF is the session organizer, and UPF is the delivery system.

1. **How GnodeB and UE connect with each other:**

The gNodeB and UE connect through synchronization, random access, authentication, and resource allocation. This ensures the UE can securely access the 5G network and exchange data efficiently.

 **UE Searches for a Network**

* The UE powers on and starts scanning for available 5G networks in the area.
* It listens for signals from nearby gNodeBs broadcasting synchronization and reference signals.

 **Synchronization**

* The UE synchronizes its timing with the gNodeB by decoding the synchronization signals.
* This ensures that the UE and gNodeB can communicate on the same time and frequency.

 **Random Access Procedure**

* The UE sends a "Random Access Request" to the gNodeB, asking for permission to connect.
* The gNodeB responds with a "Random Access Response," assigning the UE resources for communication.

 **Authentication and Security**

* The UE and the 5G Core (via gNodeB) authenticate each other to ensure a secure connection.
* Authentication ensures that only authorized devices connect to the network.

 **Attach Procedure**

* Once authenticated, the UE attaches to the network via the gNodeB.
* The gNodeB forwards the request to the AMF (Access and Mobility Management Function) in the 5G Core to complete the process.

 **Connection Establishment**

* The gNodeB assigns specific resources (radio frequencies) to the UE for data transmission.
* The UE is now ready to send and receive data via the gNodeB.

1. **What is AUSF :**

* AUSF (Authentication Server Function) is a key network function in 5G responsible for verifying and authenticating a user's identity.
* AUSF acts like a security checkpoint in a network.
* The **AUSF is the security officer**, verifying your device (UE).
* Your **SIM card or digital credentials** act as your ID.
* If the credentials are valid, AUSF communicates with other functions (like the Unified Data Management, UDM) to grant network access.

1. **What is UDM:**

* **UDM (Unified Data Management)** is a key component in the 5G core network. It is responsible for managing and storing user-related data, such as user profiles, authentication details, subscription information.
* When your device (UE) wants to connect to the 5G network, it sends its **identity** (like a **SIM card ID**) to the network.
* The **AMF (Access and Mobility Management Function)** receives this identity and sends a **request to the UDM** to verify whether the user (UE) is allowed to connect.
* It acts like a database in this UDM they store the all the details like user profiles, subscription info and service plan.

1. **What is PCF:**

* The **PCF (Policy Control Function)** in 5G is a key network function that helps in the decision-making process regarding how network resources are allocated and used.
* It manages **policy rules** related to Quality of Service (QoS), charging, traffic prioritization, and other network management services based on the user's subscription and service needs.

**How PCF Works in 5G:**

1. When a **user equipment (UE)** requests access to the network or initiates a service (such as streaming video or making a voice call), the **AMF (Access and Mobility Management Function)** sends a request to the **PCF** to retrieve the applicable policy rules for that service.
2. The **PCF** checks the user’s subscription profile, service needs, and available network resources.
3. Based on this, the **PCF** makes decisions about the **QoS** for the user’s session, any **traffic prioritization**, and **charging rules**.
4. The **PCF** sends these decisions back to the AMF or other network functions like **UPF (User Plane Function)** to implement the policies for the user session.
5. **What is 3GPP:**

* **3GPP** (3rd Generation Partnership Project)
* It is a Global partnership which creates a technical specification and standards for mobile networks and future technology.
* It ensures compatability and interoperability of mobile systems world wide.
* Enabling device and networks to communicate seamlessly.

1. **RAN:**

**RAN (Radio Access Network)** It is a core component of 5G network which is used to enable high speed , low latency , wireless communication b/w user device (UE) and the core network.

In under the RAN **UE, GNODEB, core network.**

1. **SBA : (Service Based Architecture):**

SBA in 5G is a modular and flexible network design where functions like AMF and SMF operate independently and communicate via APIs. It enables dynamic scaling, efficient upgrades, and smoother network management.

1. **REST API and HTTP/2:**

**REST API**: "A REST API defines how systems communicate over HTTP using methods like GET, POST, PUT, and DELETE. For example, in an online shopping app, REST API allows the client to fetch product details or update the cart by making HTTP requests."  
**HTTP/2**: "HTTP/2 improves the performance of these communications by using multiplexing, where multiple requests and responses are handled over a single connection. This speeds up processes like loading web pages with many images."  
**Why Important**: "REST API provides the structure for communication, while HTTP/2 ensures it's fast and efficient, especially in 5G networks or cloud-based systems."

1. **NR (New Radio):**

**New Radio**, is the global standard for 5G wireless technology developed by **3GPP (3rd Generation Partnership Project)**. It defines the air interface between user equipment (UE) and the gNB (5G base station) in 5G networks. NR is designed to deliver enhanced data rates, ultra-low latency, and massive connectivity for IoT and other advanced applications.

**Definition**: *"In NR, devices like the User Equipment (UE) and base station (gNB) connect via specific interfaces such as Uu for wireless communication and NG for connecting to the 5G core.*

1. **Network Slicing:**

Network slicing is a key feature in 5G that allows a single physical network to be divided into multiple virtual networks, each tailored to meet specific service requirements. Each slice operates as an independent network, optimized for a particular use case or customer.

**Types of Slices**:

* + Empp
  + Urllc
  + Mmtc

**Real-World Example**

**Scenario**: A 5G-enabled smart city with three applications:

1. **Smart Traffic Management**:
   * Uses the **URLLC slice** for real-time data from sensors to control traffic lights and prevent accidents.
2. **Public Wi-Fi**:
   * Utilizes the **eMBB slice** to provide high-speed internet to users in parks and public spaces.
3. **IoT Sensors for Waste Management**:
   * Relies on the **mMTC slice** to connect thousands of low-power IoT sensors that monitor waste bins across the city.
4. **How UE and 5G network connection work:**

If UE wants to connect with 5G network , it can first establish the connection with GnodeB, once UE and GnodeB connected after it forwards the registration to the AMF, which manages the authentication and mobility management. It ensures to verify the identity with the help of AUSF and UDM. Once authenticated, the SMF allocates an IP address and interacts with the PCF for policy enforcement.

Finally, the UPF routes the data between UE and external networks , ensuring smooth communication.

1. **Network Slicing :**

Imagine a big highway that serves all kinds of vehicles—bikes, cars, trucks, and buses. What if each vehicle type had its own separate lane?

In the same way, Network Slicing divides a single physical 5G network into multiple virtual networks (slices). Each slice is customized for specific needs:

* 🚀 High-speed slices for gamers
* 📶 Reliable slices for emergency services
* 📊 Data-heavy slices for businesses

Why do this?

* Improves efficiency 💡
* Ensures better performance for different services 🎯
* Enables faster data transfer for specific users 🚀

**Key Functions of Network Slicing**

1. **Isolation** – Each slice works independently without interfering with others.
2. **Customization** – Each slice can be customized for speed, security, or performance.
3. **Resource Allocation** – Resources like bandwidth and processing power are assigned specifically to each slice.

**1. Emergency Services (Ambulance, Police, Firefighters) – URLLC Slice**

**Ultra-Reliable Low-Latency Communications (URLLC)**

**Scenario:**  
A person has an accident, and the ambulance system needs to communicate instantly with the hospital.

* A special **emergency slice** guarantees:
  + 📡 **Instant connection** (No delays)
  + 🔒 **High reliability** (No signal drops)
  + 🛡️ **Secured data transmission**

**Real-world impact:**

* Faster response times
* Remote doctors can assist during ambulance rides using live data

**2. Gaming and Virtual Reality – eMBB Slice**

**Enhanced Mobile Broadband (eMBB)**

**Scenario:**  
Gamers playing competitive online games need **ultra-fast** data speeds with **low latency**.

* A separate **gaming slice** ensures:
  + ⚡ **High-speed data** for smooth gameplay
  + 🎧 **Low latency** for quick responses
  + 🚫 **No interruptions** from other network traffic

**Real-world impact:**

* Seamless virtual reality experiences
* Cloud gaming services without lags

**3. Smart Cities – mMTC Slice**

**Massive Machine-Type Communications (mMTC)**

**Scenario:**  
Thousands of smart devices (traffic lights, sensors, surveillance cameras) are connected in a smart city.

* A **smart device slice** manages:
  + 🔗 Connection for **millions of devices**
  + 🔋 **Low power usage**
  + ⚙️ Efficient data handling

**Real-world impact:**

* Smart traffic control (reducing congestion 🚦)
* Automated energy-saving systems in buildings

**4. Industrial Automation – Private 5G Slice**

**Scenario:**  
A factory automates its machines using a private 5G network slice.

* A **private slice** provides:
  + 🏭 Reliable and secure communications between machines
  + ⚡ Real-time monitoring for efficiency
  + 🔐 Higher security to prevent cyberattacks

**Real-world impact:**

* Fewer human errors
* Real-time production adjustments

**5. Airline In-Flight Services – Custom Slice**

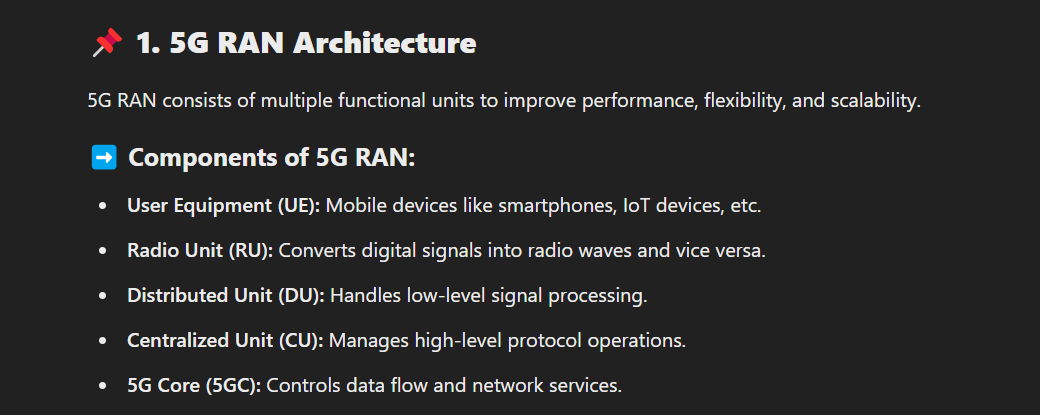
**Scenario:**  
Airlines offer different network services onboard for passengers and crew.

* A **passenger slice** → High-speed internet and entertainment
* A **crew slice** → Reliable communication with ground control

**Real-world impact:**

* Passengers stream videos without affecting flight operations
* Crew maintains safe communication channels

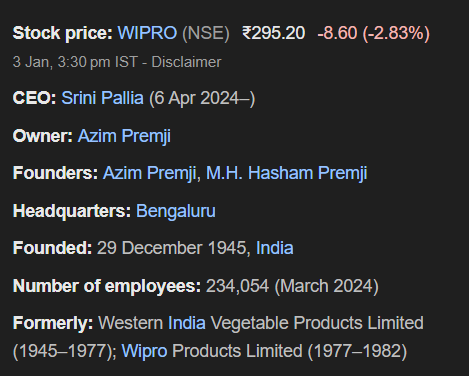
**5G RAN ARCHITECTURE ORDER WISE:**



**HR INTERVIEW:**

**Question ask to interviewer:**

* As a fresher, what Wipro expects from me?
* What are the learning and development opportunities available at Wipro for someone in the 5G Datacom role?

****

**ABOUT WIPRO:**

"Wipro is a leading global IT, consulting, and business process services company, founded in 1945. It operates in over 60 countries and provides innovative solutions in areas like cloud computing, cybersecurity, AI, and 5G technologies.

The company focuses on digital transformation and has a strong commitment to sustainability, diversity, and continuous learning. Wipro’s investments in emerging technologies like 5G align with my interest in networking and telecommunications, and I am excited to be part of such an innovative company."

And Wipro follows 5 values:

Wipro follows 5 habits to increase their values:

* + 1. Being Respectful
    2. Being Responsive
    3. Always Communicating
    4. Being Stewardship
    5. Being trust

*What are Wipro’s core values?*

**Answer:**  
"According to Wipro's official website, the company's five core values are Being Respectful, Being Responsive, Always Communicating, Demonstrating Stewardship, and Building Trust. These values form the foundation of Wipro’s work culture and guide their interactions with clients, partners, and employees."

**How long will u work for thr company ku**

I expect continous learning and growth in my career so As long as I have continous growth I will work for this company

I wish this company provides me these requirements of mine

**Where do see urself in 5 years ?**

In 5 years I see myself as an important part of this organisation and I will be in the position to lead others and Ill share my knowledge to freshers

**Why should I hire u?**

"If you hire me, it will be a great platform for me to showcase my skills. I'm in the learning phase of my career, and I'm eager to gain more knowledge. I'm open to learning new technologies and working across different technical stacks. I believe that by continuously improving myself, I can contribute to the company's growth while achieving my personal career goals."

**Why do you want work at WIPRO:**

I want to work at Wipro because it is a highly reputed MNC that provides a huge platform to showcase my skills and gain new knowledge. Being selected for the 5G Datacom role makes me particularly curious and motivated. I believe this role will help me grow both professionally and personally, making a positive impact on my career.

**Strengths:**  
"My strengths are that I am a **quick learner** and a very **punctual person**. I always complete tasks on time, and I am eager to **explore new technologies** to improve my skills."

**Weaknesses:**  
"My weakness is that I tend to **overthink** certain situations. I also have a habit of **taking on too many tasks at once**, but I’m working on **improving my time management** and **prioritizing tasks** more effectively."

**Situation:**  
During my time at **L&T Defence**, my project supervisor assigned me a challenging task to collect and organize data from the **ERP LN portal**. The task involved tracking parts that were arriving, processed, and cleared to be sent to vendors. It was a critical task for ensuring smooth operations.

**Task:**  
The challenge was that I had to complete this task **within one night**, and there were no freshers on my team to assist. It was my responsibility to gather accurate data and create a comprehensive report.

**Action:**  
I started by accessing the ERP LN portal and comparing the data with our **reference workbook**, which was maintained by project and store team members. I filtered the product details **part-wise** and prepared a new workbook based on **priority parts** that needed immediate attention. I ensured that all the data was accurate and up to date.

**Result:**  
By working efficiently and prioritizing tasks, I successfully collected and organized all the necessary data **within the given timeframe**. My report helped the team process high-priority parts quickly.

**Tell me anything that not in your resume:**

Apart from what's in my resume, I'm someone who loves exploring new technologies and learning new skills. I enjoy solving real-world problems and working in a team environment. In my free time, I listen to music and stay updated on tech trends to keep myself motivated and creative.

**Can you work under pressure?**

Yes, I can work under pressure. For example, at L&T, I once had to collect and validate data from the ERP system within a tight deadline. I stayed focused, prioritized tasks, and completed it on time. It taught me how to handle pressure calmly and deliver results.

**LINUX:**

**What is Linux?**

Linux is a powerful, open-source operating system that is widely used in server environments due to its stability and security. It’s also popular in cloud infrastructure and development environments.

It comes in different **distributions (distros)** like **Ubuntu, CentOS, Fedora, Red Hat,** etc.

**What is Shell Scripting?**

Shell scripting allows users to automate tasks in Linux. For example, we can write scripts to automate backups, system monitoring, or file management tasks.

**What is the Difference Between Linux and Windows?**

Linux is open-source, secure, and highly customizable. Unlike Windows, it offers strong terminal support and is widely used for servers and development environments.

**What are File Permissions in Linux?**

* Linux has **three types of permissions**: **read (r), write (w), execute (x)**.
* **User types**:
  + **Owner**: The user who owns the file
  + **Group**: Users in the same group as the owner
  + **Others**: All other users

**What is Process Management in Linux?**

Linux provides various commands for process management like **ps** to view running processes, **top** to monitor system resources, and **kill** to terminate processes.

**What is a Package Manager in Linux?**

* A **package manager** is a tool that handles the installation, upgrade, and removal of software packages.
* Popular package managers:
  + **APT** (Ubuntu, Debian): sudo apt install package-name
  + **YUM** (CentOS, Red Hat): sudo yum install package-name.

***GIT and GITHUB:***

1) **What is Git?**

* Git is a version control system used to track changes in source code.
* Allows multiple people to work on a project without overwriting each other's work.

**Basic Git Setup:**

git config --global user.name "YourName"

git config --global user.email [your@email.com](mailto:your@email.com)

**Repository Commands:**

**Commands Purpose**

|  |  |
| --- | --- |
| git init | Initialize a new Git repo |
| git clone <url> | Clone an existing repo |
| git remote -v | Check remote URLs |
| git remote add origin <url> | Connect local repo to remote |
| **File Tracking:**   | **Command** | **Purpose** | | --- | --- | | git status | Show changes and status | | git add <file> | Stage a file | | git add . | Stage all changes | | git commit -m "Message" | Commit staged files | | git log | View commit history | | git diff | See unstaged changes | | git diff --staged | See staged changes | | git rm <file> | Delete a file and stage the deletion | |  |
|  |  |
|  |  |
|  |  |

**Branching:**

| **Command** | **Purpose** |
| --- | --- |
| git branch | List branches |
| git branch <name> | Create a new branch |
| git checkout <name> | Switch branches |
| git checkout -b <name> | Create + switch branch |
| git merge <branch> | Merge branch into current |
| git branch -d <name> | Delete branch |
| git log --oneline --graph --all | View branch structure |

**Pushing & Pulling:**

**Commands Purpose**

git push origin <branch> Push to remote

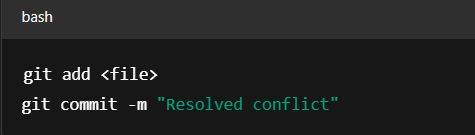
git pull origin <branch> Pull latest changes

git fetch Download changes (but   
 not merge)

**Merge Conflicts**

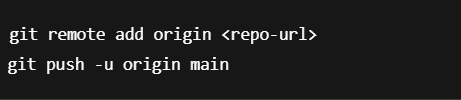
When two branches change the same lines:

* Git marks the conflict
* You manually edit and resolve
* Then:



**🚀 GitHub Workflow**

1. **Create a repository** on GitHub
2. **Connect your local repo**:



1. **Create pull requests (PR)** to review and merge changes
2. **Use Issues** to track bugs/tasks
3. **Clone, fork, and contribute** to open source

**🧰 Advanced Topics (Optional for later)**

* git stash — temporarily save changes
* git rebase — cleaner history for merging
* .gitignore — ignore certain files from tracking
* git tag — mark specific versions (e.g., v1.0.0)
* GitHub Actions — automate workflows

**📝 Pro Tips**

* Always use git status to understand what's going on.
* Commit frequently with meaningful messages.
* Use branches to avoid breaking the main project.
* Use .gitignore to avoid committing build or sensitive files.

**Resources**

* <https://git-scm.com/docs>
* <https://learngitbranching.js.org>
* <https://github.com>

**C++:**

🧰 Part 2: Must-Know C++ Inbuilt Functions

Here’s a shortlist of C++ STL functions you should know (mostly from <algorithm>):

| **Function** | **Use** |
| --- | --- |
| sort(v.begin(), v.end()) | Sort elements |
| reverse(v.begin(), v.end()) | Reverse elements |
| max(a, b) / min(a, b) | Get max or min |
| accumulate(v.begin(), v.end(), 0) | Sum of elements |
| count(v.begin(), v.end(), x) | Count occurrences of x |
| find(v.begin(), v.end(), x) | Find element x |
| binary\_search(v.begin(), v.end(), x) | Binary search in sorted vector |
| lower\_bound() / upper\_bound() | Find range positions |
| all\_of() / any\_of() | Check conditions on all or any elements |
| remove(v.begin(), v.end(), x) | Remove all x (use with erase) |
| next\_permutation() | Next lexicographical permutation |
| set / map / unordered\_set / unordered\_map | STL containers |
| queue, stack, deque, priority\_queue | Data structures |

🧪 STL Header Files to Remember:

| **Header** | **Contains** |
| --- | --- |
| <vector> | vector class |
| <algorithm> | sort, find, reverse, etc. |
| <numeric> | accumulate |
| <map> | map, unordered\_map |
| <set> | set, unordered\_set |
| <queue> | queue, priority\_queue, deque |
| <stack> | stack |

***DOCKER***

**What is Docker?**

* Docker is an open-source platform to automate the **deployment, scaling, and management of applications** using **containerization**.
* A **container** packages the application code along with dependencies, libraries, and runtime environment—ensuring it runs the same everywhere.
  + **Key Concepts:**

| **Term** | **Description** |
| --- | --- |
| Image | A lightweight, standalone, and executable software package that includes everything to run a piece of software. |
| Container | A running instance of a Docker image. |
| Dockerfile | A text file with step-by-step instructions to build a Docker image. |
| Docker Hub | A cloud-based registry to find and share container images. |
| Volume | A persistent data storage mechanism for containers. |
| Docker Engine | The core client-server technology behind Docker. |

⚙️ **Docker Architecture**

1. Docker Client – Interface to communicate with the Docker daemon.
2. Docker Daemon (dockerd) – Runs on host, manages images, containers, volumes, etc.
3. Docker Images – Read-only templates.
4. Docker Containers – Instances of images with isolation.
5. Docker Registry – Stores images (e.g., Docker Hub, GitHub Container Registry).

**⚙️🡪 Basic Docker Commands**

| **Task** | **Command** |
| --- | --- |
| Check Docker version | docker --version |
| List running containers | docker ps |
| List all containers (including stopped) | docker ps -a |
| Pull an image | docker pull <image\_name> |
| Run a container | docker run <image\_name> |
| Run with interactive terminal | docker run -it <image\_name> |
| Build an image | docker build -t <image\_name> . |
| Stop a container | docker stop <container\_id> |
| Remove a container | docker rm <container\_id> |
| Remove an image | docker rmi <image\_name> |

* + **Docker Lifecycle**

1. Write a **Dockerfile**.
2. Build image using docker build.
3. Run container using docker run.
4. Manage container lifecycle (start, stop, rm, etc.).
   * **Dockerfile – Building Custom Images**

A Dockerfile is a text file with instructions to create an image.

**Example:**

Dockerfile

*FROM node:18*

*WORKDIR /app*

*COPY . .*

*RUN npm install*

*CMD ["node", "index.js"]*

**🧪 Important Docker Commands**

**🗂️ Image Commands:**

*docker pull <image> # Download image from Docker Hub*

*docker build -t name . # Build image from Dockerfile*

*docker images # List all images*

*docker rmi <image\_id> # Remove image*

🚀 **Container Commands:**

*docker run <image> # Run container from image*

*docker run -it <image> # Run container in interactive mode*

*docker run -d <image> # Detached mode (background)*

*docker ps # List running containers*

*docker ps -a # List all containers*

*docker stop <id> # Stop container*

*docker rm <id> # Remove container*

🔁 **Volume & Storage:**

*docker volume create myvol # Create volume*

*docker run -v myvol:/data ... # Mount volume to container*

*docker volume ls # List volumes*

🧠 **Possible Interview Questions---**

**✅ 1. What is Docker and why is it used?**

**Answer:**

Docker is a containerization platform that allows me to package an application along with all its dependencies, libraries, and configurations into a single container. This ensures the application runs consistently across different environments—whether it's development, testing, or production. It helps solve the “It works on my machine” problem.

**✅ 2. What is the difference between an image and a container?**

**Answer:**

An image is like a blueprint—it contains the application code, libraries, dependencies, and the OS runtime. A container is a running instance of that image. So, I can create multiple containers from a single image, each running independently.

**✅ 3. How do you build and run a Docker container?**

**Answer:**

First, I write a Dockerfile with instructions to build the image. Then I use docker build -t <image-name> . to build it. To run the container, I use docker run <image-name>. If I need interactive mode, I add -it, and to map ports, I use -p.

**✅ 4. What is a Dockerfile?**

**Answer:**

A Dockerfile is a script that contains a set of instructions to build a Docker image. It defines things like the base image, working directory, files to copy, dependencies to install, and the command to run when the container starts.

**✅ 5. How is Docker different from a virtual machine?**

**Answer:**

Docker containers share the host OS kernel and are lightweight, fast to start, and use fewer resources. In contrast, virtual machines include a full OS, which makes them heavier and slower to boot. Docker is more efficient for running isolated applications or microservices.

**✅ 6. Have you used Docker Compose? Explain it.**

**Answer:**

Yes, I’ve used Docker Compose to manage multi-container applications. It uses a YAML file (docker-compose.yml) to define services like a web app and a database. I can bring them up together with docker-compose up, which simplifies running complex setups locally.

**✅ 7. How do you persist data in Docker?**

**Answer:**

I use **Docker volumes** to persist data. Volumes are stored outside the container’s writable layer, so even if the container is deleted, the data remains. I can use the -v flag in docker run or define volumes in a Compose file.

**✅ 8. What is the difference between COPY and ADD in Dockerfile?**

**Answer:**

Both COPY and ADD are used to move files into the image. COPY is straightforward and only copies files. ADD does everything COPY does, but also supports extracting .tar files and downloading files from URLs. I use COPY when I just need to transfer files and prefer ADD only when I need the extra features.

***Kubernetes:***

📁 **Additional Useful Commands:**

| **Task** | **Command** |
| --- | --- |
| List all Pods | kubectl get pods |
| View Pod details | kubectl describe pod <pod-name> |
| Delete Pod | kubectl delete pod <pod-name> |
| Apply YAML file | kubectl apply -f <file>.yaml |
| Delete resource from YAML | kubectl delete -f <file>.yaml |
| View Logs | kubectl logs <pod-name> |
| Exec inside container | kubectl exec -it <pod-name> -- /bin/sh |
| List Deployments | kubectl get deployments |
| Get Nodes | kubectl get nodes |
| Set current namespace | kubectl config set-context --current --namespace=<ns> |

**1. What is Kubernetes?**

**Answer:**  
Kubernetes is an open-source container orchestration platform used to automate the deployment, scaling, and management of containerized applications.

**✅ 2. Why do we need Kubernetes?**

**Answer:**  
Kubernetes helps manage containers in production by:

* Automatically restarting failed containers (self-healing)
* Load balancing and service discovery
* Automating rollouts and rollbacks
* Horizontal scaling of apps
* Managing container configuration using declarative YAML files

**✅ 3. What is a Pod?**

**Answer:**  
A Pod is the smallest and simplest unit in Kubernetes. It wraps one or more containers that share storage, network, and a specification for how to run them.

**✅ 4. What is a Node?**

**Answer:**  
A Node is a physical or virtual machine on which Kubernetes runs and deploys Pods. There are two types:

* **Master Node**: Controls and manages the cluster
* **Worker Node**: Where application workloads (Pods) run

**✅ 5. What is a Cluster?**

**Answer:**  
A Cluster is a group of nodes managed by Kubernetes, consisting of at least one master node and multiple worker nodes.

**✅ 6. What is the Control Plane?**

**Answer:**  
The Control Plane is responsible for managing the Kubernetes cluster. It includes:

* **API Server**: Frontend for the Kubernetes control plane
* **etcd**: Key-value store for all cluster data
* **Scheduler**: Assigns pods to nodes
* **Controller Manager**: Maintains desired state (replica, job, node controllers)

**✅ 7. What are the main components of a Worker Node?**

**Answer:**

* **kubelet**: Ensures containers are running in a pod
* **kube-proxy**: Handles networking and communication
* **Container Runtime**: Software used to run containers (like Docker or containerd)

**✅ 8. What is a Deployment?**

**Answer:**  
A Deployment is a Kubernetes object that manages a ReplicaSet and Pods. It ensures the specified number of pod replicas are running and can perform rolling updates.

**✅ 11. What is etcd?**

**Answer:**  
etcd is a consistent and distributed key-value store used by Kubernetes to store all cluster configuration data and state.

**✅ 12. What is a ConfigMap?**

**Answer:**  
A ConfigMap is a key-value pair storage used to store configuration data separately from the container image.

**✅ 15. What is kubectl?**

**Answer:**  
kubectl is the command-line tool used to interact with the Kubernetes cluster, like creating, viewing, and managing resources.

**Common commands:**

bash

CopyEdit

kubectl get pods

kubectl create -f pod.yaml

kubectl delete pod mypod

kubectl describe pod mypod

kubectl logs mypod

**✅ 17. What is the difference between a Pod and a Deployment?**

**Answer:**

* **Pod** is a single instance of a running process.
* **Deployment** manages Pods and ensures they are up and running in the desired count. It also handles updates and rollbacks.

**✅ 18. What is the lifecycle of a Pod?**

**Answer:**

1. **Pending**: Pod is accepted but not running
2. **Running**: Containers are executing
3. **Succeeded**: All containers completed successfully
4. **Failed**: At least one container failed
5. **Unknown**: State cannot be determined