TCS INTERVIEW PREPARATION

1.) MySQL

5 MOST IMPORTANT SQL QUERY interview questions

1️⃣ Find the 2nd Highest Salary from an Employee table

Table: `Employee`

id	name	salary

Query:

```
SELECTMAX(salary)AS SecondHighestSalary
FROM Employee
WHERE salary< (SELECTMAX(salary)FROM Employee);
```

Explanation:

- Inner query finds highest salary
- Outer query finds **max salary less than highest**

⭐ **Very common question**

2️⃣ Find Duplicate Records in a table

Table: `Employee`

| id | email |

Query:

```
SELECT email,COUNT(*)AS count
FROM Employee
GROUPBY email
HAVINGCOUNT(*)>1;
```

Explanation:

- `GROUP BY` groups same values
- `HAVING` filters groups

⭐ Asked in **almost every SQL interview**

3️⃣ Find Employees who earn more than average salary

Query:

```
SELECT*
FROM Employee
WHERE salary> (SELECTAVG(salary)FROM Employee);
```

Explanation:

- Subquery calculates average salary
- Main query compares each salary with average

⭐ Tests **subquery understanding**

4️⃣ Get Nth Highest Salary (General solution)

Example: 3rd Highest Salary

```
SELECT salary
FROM Employee
ORDERBY salaryDESC
LIMIT1OFFSET2;
```

OR (Interview-safe approach)

```
SELECTDISTINCT salary
FROM Employee e1
WHERE3= (
SELECTCOUNT(DISTINCT salary)
FROM Employee e2
WHERE e2.salary>= e1.salary
);
```

Explanation:

- Works even if duplicate salaries exist

⭐ Interviewers love this logic

5️⃣ Find Employees without matching records (LEFT JOIN concept)

Tables:

Employee(emp_id, dept_id)

Department(dept_id, dept_name)

Query:

```
SELECT e.*
FROM Employee e
LEFTJOIN Department d
ON e.dept_id= d.dept_id
WHERE d.dept_idISNULL;
```

Explanation:

- `LEFT JOIN` keeps all employees
- `NULL` means no matching department

⭐ Tests **JOIN + NULL logic**

🔥 BONUS (If interviewer asks more)

Find highest salary department-wise

```
SELECT dept_id,MAX(salary)
FROM Employee
GROUPBY dept_id;
```

🧠 Interview Tip (Important)

If interviewer asks:

> “Explain your query”

Always say:

> “First I use subquery / group by to calculate required value, then I filter the result using WHERE / HAVING.”

2.) Operating System

10 most important Operating System interview questions with concise, interview-ready solutions.

1. What is an Operating System and what are its main functions?

Answer:
An Operating System (OS) is system software that acts as an interface between the user and the computer hardware. It manages hardware resources and provides common services for computer programs.

- **Key Functions:**
  - **Process Management:** Creating, scheduling, and terminating processes.
  - **Memory Management:** Allocating and deallocating memory (RAM) to programs.
  - **File Management:** Managing files and directories on storage devices.
  - **Device Management:** Handling input/output devices via drivers.
  - **Security:** Protecting data and resources from unauthorized access.

2. What is the difference between a Process and a Thread?

Answer:
This is one of the most frequently asked questions.

- **Process:** A program in execution. It is an active entity that has its own memory space (code, data, stack, heap) and system resources. It is heavy-weight; context switching between processes is slow.
- **Thread:** A segment of a process, often called a "lightweight process." Threads within the same process share the same memory space (code, data, heap) but have their own stack and registers. Context switching between threads is much faster.

3. What is Deadlock and what are the necessary conditions for it to occur?

Answer:
A Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
4 Necessary Conditions (Coffman Conditions):

1. **Mutual Exclusion:** At least one resource must be non-sharable (only one process can use it at a time).
2. **Hold and Wait:** A process holding at least one resource is waiting to acquire additional resources held by others.
3. **No Preemption:** Resources cannot be forcibly taken from a process; they must be released voluntarily.
4. **Circular Wait:** A set of processes {P0,P1,...,Pn} exists such that P0 is waiting for P1, P1 for P2, ..., and Pn for P0.

4. Explain Virtual Memory. Why do we need it?

Answer:
Virtual Memory is a memory management technique that creates an illusion for users that the computer has more memory (RAM) than it actually does. It allows the execution of processes that are larger than the physical memory by transferring data between the main memory (RAM) and secondary storage (Disk) using **Paging** or **Segmentation**.

- **Why we need it:** It increases the degree of multiprogramming (more apps run simultaneously) and allows running large applications that exceed physical RAM size.

5. What is the difference between Paging and Segmentation?

Answer:
Both are non-contiguous memory allocation techniques.

- **Paging:** Divides memory into fixed-size blocks called **Pages** (logical memory) and **Frames** (physical memory). It is managed by the OS and solves external fragmentation but may cause internal fragmentation.
- **Segmentation:** Divides memory into variable-size blocks called **Segments** based on logical units (e.g., function, stack, array). It is visible to the user/compiler and solves internal fragmentation but may suffer from external fragmentation.

6. What is Context Switching?

Answer:
Context Switching is the process of storing the state (context) of the currently running process or thread so that it can be resumed later, and loading the state of the next process to be run.

- **Context:** Includes the process's register values, program counter, and memory maps.
- **Overhead:** It is purely overhead because the CPU does no useful work while switching; therefore, OS designers try to minimize it.

7. Differentiate between Semaphore and Mutex.

Answer:
Both are synchronization tools used to solve the critical section problem.

- **Mutex (Mutual Exclusion Object):** A locking mechanism. Only the process that locks the mutex can unlock it. It is used for exclusive access to a resource (like a key to a toilet).
- **Semaphore:** A signaling mechanism. It uses an integer value to manage access. It can be a **Binary Semaphore** (value 0 or 1, similar to Mutex) or a **Counting Semaphore** (value > 1) to manage multiple instances of a resource (e.g., 5 printers).

8. What is Thrashing?

Answer:
Thrashing occurs when the system spends more time swapping pages in and out of memory (paging) than actually executing instructions.

- **Cause:** This usually happens when the degree of multiprogramming is too high, and the total memory required by the active processes exceeds the physical memory available.
- **Result:** CPU utilization drops drastically.

9. What is the difference between Monolithic Kernel and Microkernel?

Answer:

- **Monolithic Kernel:** The entire OS (process management, memory, device drivers, file system) runs in the kernel space. It is faster but if one service crashes, the whole system can crash (e.g., Early Unix, Linux).
- **Microkernel:** Only the essential services (IPC, basic scheduling, memory handling) run in the kernel space, while others (drivers, file systems) run in user space. It is more stable and secure but slightly slower due to increased message passing (e.g., Minix, QNX).

10. Explain the Process States (Process Lifecycle).

Answer:
A process goes through five main states during its execution:

1. **New:** The process is being created.
2. **Ready:** The process is waiting to be assigned to the processor (in the ready queue).
3. **Running:** Instructions are being executed by the CPU.
4. **Waiting (Blocked):** The process is waiting for some event (like I/O completion) to occur.
5. **Terminated:** The process has finished execution.

Top 15 OS Interview Questions | Placement Strategy

This video by Gate Smashers is highly relevant as it breaks down the top 15 most asked Operating System questions (including Process States, Deadlock, and Paging) specifically for placement interviews with clear explanations.

Top 15 OS Interview Questions | Placement Strategy - YouTubeGate Smashers · 208K views

3.) React js

1. What is React JS? What is the Virtual DOM and how does it work?

Answer:

→React is a **JavaScript library** used to build **user interfaces**, especially **single-page applications (SPA)**. It follows a **component-based architecture** and uses a **Virtual DOM** for better performance.

→The Virtual DOM (VDOM) is a lightweight copy of the actual DOM kept in memory.

- **How it works:** When the state of an object changes, React creates a new Virtual DOM tree. It then compares this new tree with the previous one (a process called **"Diffing"**).
- **Reconciliation:** Once the differences are identified, React updates **only** those specific objects in the real DOM, rather than re-rendering the entire page. This makes React highly efficient.

2. What is JSX? Why can't browsers read it directly?

Answer:
JSX (JavaScript XML) is a syntax extension for JavaScript that allows us to write HTML-like code within JavaScript.

- **Why browsers can't read it:** Browsers only understand standard JavaScript objects. JSX is not valid JS.
- **Solution:** We need transpilers like **Babel** to convert JSX into regular `React.createElement()` calls that the browser can parse.

3. Differentiate between State and Props.

Answer:

Feature	State	Props (Properties)
Ownership	Owned and managed locally within the component.	Passed down from a parent component.
Mutability	Mutable (can be changed using `setState` or `useState`).	Immutable (cannot be changed by the receiving component).
Role	Holds data that might change over time (e.g., user input).	Configures a component (e.g., passing a label to a button).

4. What are Components in React? Explain the Component Lifecycle (and how Hooks replace it).

Answer:

→ Components are **reusable pieces of UI**.

Types:

- **Functional Components** (mostly used)
- **Class Components** (older)

→ In Class components, there are three main phases:

1. **Mounting:** `componentDidMount()` – Runs after the component is rendered. Good for API calls.
2. **Updating:** `componentDidUpdate()` – Runs after state/props change.
3. **Unmounting:** `componentWillUnmount()` – Runs before removal. Good for cleanup.

In Functional Components (Modern React), we use the `useEffect` hook to achieve the same:

JavaScript

```
useEffect(() => {
 // Code here runs on Mount and Update
 return () => {
   // Code here runs on Unmount (Cleanup)
 };
}, [dependencies]);
```

5. What is "Prop Drilling" and how do you avoid it?

Answer:

Prop Drilling is the process of passing data from a parent component down to a deeply nested child component through intermediate components that don't need the data themselves.

- **Solution:**
  1. **Context API:** Allows you to "teleport" data directly to the components that need it without passing props manually at every level.
  2. **State Management Libraries:** Tools like Redux or Zustand.

6. What are React Hooks? Name the rules of using them.

Answer:
Hooks are functions that let you "hook into" React state and lifecycle features from functional components (introduced in React 16.8). Common hooks include `useState`, `useEffect`, and `useContext`.
Rules of Hooks:

1. **Only call Hooks at the top level:** Don’t call them inside loops, conditions, or nested functions.
2. **Only call Hooks from React functions:** Call them from React functional components or custom hooks, not regular JS functions.

7. Why do we need "Keys" in React Lists?

Answer:
Keys are special string attributes you need to include when creating lists of elements.

- **Purpose:** They help React identify which items have changed, been added, or been removed.
- **Importance:** Without unique keys, React might re-render the entire list instead of just the changed item, leading to performance issues and potential bugs with component state.
- **Note:** Avoid using array indexes as keys if the list order can change.

8. Controlled vs. Uncontrolled Components.

Answer:

- **Controlled Component:** The form data is handled by the React component (via State). The input value is controlled by React.
  - *Example:* `<input value={name} onChange={handleChange} />`
- **Uncontrolled Component:** The form data is handled by the DOM itself. You access the values using **Refs**.
  - *Example:* `<input ref={inputRef} />`

9. What is the difference between `useMemo` and `useCallback`?

Answer:
Both are used for performance optimization (Memoization).

- **`useMemo`:** Returns a **memoized value**. It only recomputes the value when dependencies change. (Useful for expensive calculations).
- **`useCallback`:** Returns a **memoized function**. It prevents a function from being recreated on every render unless dependencies change. (Useful when passing functions to child components to prevent unnecessary re-renders).

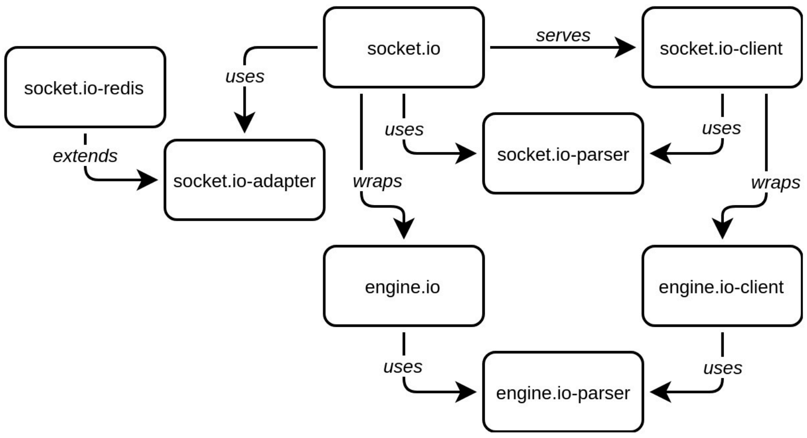
10. What are Higher-Order Components (HOC)?

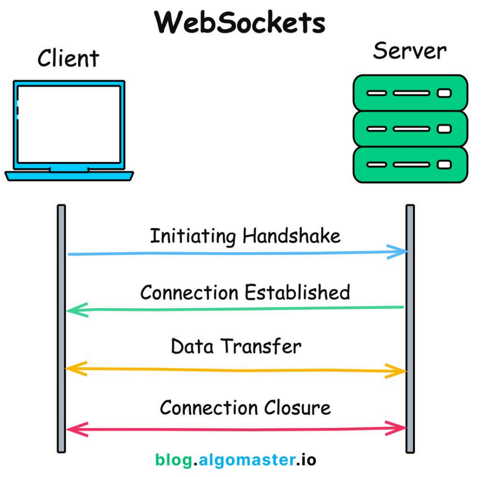
Answer:
A Higher-Order Component is a function that takes a component and returns a new component. It is an advanced technique for reusing component logic.

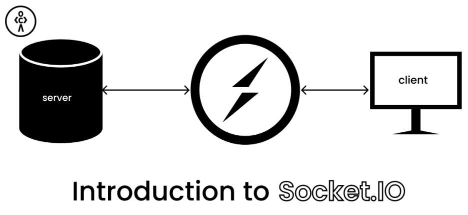
- **Pattern:** `const EnhancedComponent = higherOrderComponent(WrappedComponent);`
- **Use Case:** Authentication (wrapping a protected route), logging, or adding common styles/props to multiple components.

Socket.io & WebRTC

What is Socket.IO?







Socket.IO is a **JavaScript library** used for **real-time, bidirectional communication** between a client (browser/app) and a server.

✅ What problem does it solve?

Normally, web apps use HTTP, which is **request–response based**.

Socket.IO allows the server to **push data instantly** to clients without waiting for a request.

🧠 How it works

- Built on top of **WebSockets** (with fallbacks like HTTP long polling)
- Maintains a **persistent connection** between client and server
- Uses an **event-based** system

✨ Key Features

- Real-time messaging

- Automatic reconnection
- Event-based communication (`emit` , `on`)
- Rooms & namespaces (group users easily)
- Works even if WebSockets are blocked

📌 Common Use Cases

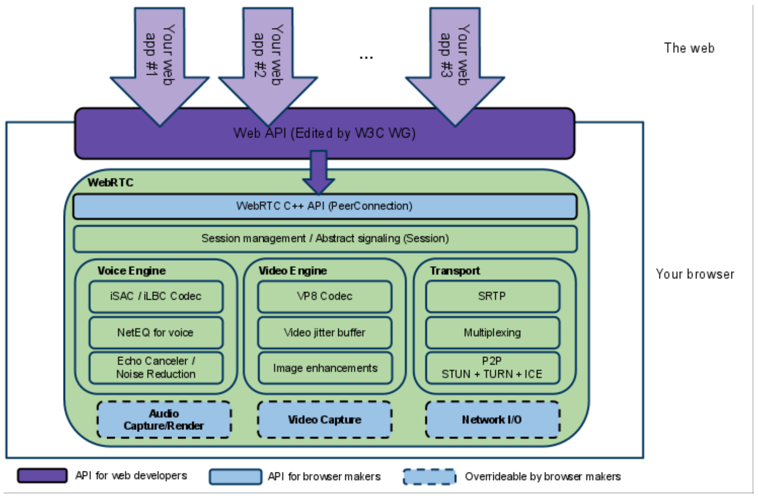
- Chat applications 💬
- Live notifications
- Multiplayer online games 🎮
- Live dashboards (stocks, analytics)
- Collaborative apps (Google Docs–like)

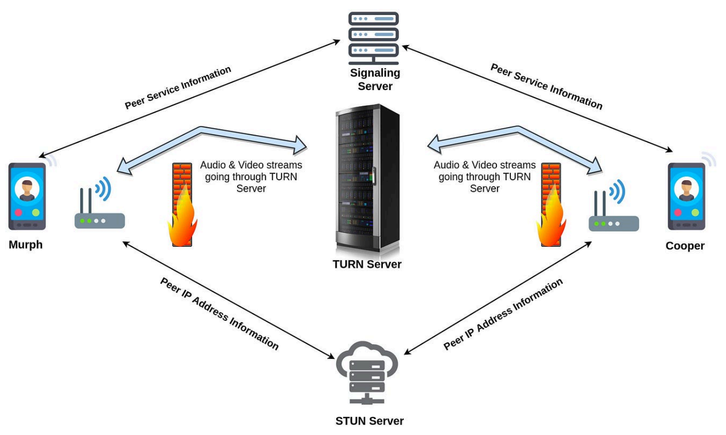
🧩 Simple Example

```
// client
socket.emit("message","Hello Server");

// server
socket.on("message",data => {
console.log(data);
});
```

📡 What is WebRTC?




Signaling Server
Peer Service Information
Peer Service Information
Audio & Video streams going through TURN Server
Audio & Video streams going through TURN Server
Murph
Cooper
TURN Server
Peer IP Address Information
Peer IP Address Information
STUN Server

WebRTC (Web Real-Time Communication) is a **browser technology** that enables **peer-to-peer (P2P)** communication of **audio, video, and data** directly between users.

✅ What problem does it solve?

It allows **real-time media streaming** (video/audio/screen sharing) **without plugins or external software**.

🧠 How it works

- Establishes **direct peer-to-peer connections**
- Uses:
  - **STUN** → find public IP
  - **TURN** → relay data if P2P fails
- Needs a **signaling server** (often Socket.IO) to exchange connection info

✨ Key Features

- Low latency
- Direct browser-to-browser communication
- Supports:
  - Video 📹
  - Audio 🎤
  - Screen sharing 🖥️
  - Data channels

📌 Common Use Cases

- Video calling apps (Zoom, Google Meet)
- Voice chat
- Screen sharing
- File transfer between users
- Live streaming

⚖️ Socket.IO vs WebRTC (Quick Comparison)

Feature	Socket.IO	WebRTC
Communication	Client ↔ Server	Peer ↔ Peer
Main Purpose	Real-time data	Audio / Video / Data
Latency	Low	Ultra-low
Media Support	❌ No	✅ Yes

Feature	Socket.IO	WebRTC
Server Required	✅ Always	❌ Only for signaling
Best For	Chats, notifications	Video/audio calls

🤝 How They Work Together

Very often, **both are used together**:

1. **Socket.IO** → signaling (exchange offer, answer, ICE candidates)
2. **WebRTC** → actual video/audio/data transfer

👉 Example:

Zoom / Google Meet

- Socket.IO (or similar): connection setup
- WebRTC: live video & audio

🧠 Simple Analogy

- **Socket.IO** → 📬 *WhatsApp text messages*
- **WebRTC** → 📞 *Phone/video calls*

Part 1: Socket.io (Real-Time Event-Based Communication)

1. How is Socket.io different from standard WebSockets?

Answer:

- **WebSockets** is a standard communication **protocol** (ws://) that provides a full-duplex communication channel over a single TCP connection.
- **Socket.io** is a **library** (not a protocol) built on top of WebSockets.
- **Key Differences:**
  - **Fallbacks:** Socket.io automatically falls back to HTTP Long Polling if WebSockets are not supported by the browser or network (firewalls).
  - **Auto-Reconnection:** Socket.io has built-in logic to keep trying to reconnect if the connection drops.
  - **Packet Metadata:** Socket.io sends extra metadata (packet type, namespace, ack id), making it incompatible with a raw WebSocket server.

2. Explain the difference between "Namespaces" and "Rooms" in Socket.io.

Answer:

- **Namespaces (`/admin`, `/chat`):** Essentially different endpoints. They allow you to split the logic of your application over a single shared connection. A client connects to a specific namespace.
- **Rooms (`room-101`, `lobby`):** Server-side groupings *within* a namespace. A socket can `join()` or `leave()` a room. It is useful for broadcasting data to a subset of users (e.g., users in a specific chat group).
  - *Note:* Rooms are a server-only concept; clients do not know which rooms they are in.

3. How do you scale Socket.io to multiple servers? (The Redis Adapter)

Answer:
By default, Socket.io only works on a single server instance because connections are stateful. If Client A is connected to Server 1 and Client B to Server 2, Server 1 cannot send a message to Client B directly.

- **Solution:** Use the **Redis Adapter**.
- **How it works:** It uses Redis **Pub/Sub** (Publish/Subscribe) mechanism. When Server 1 emits an event to a "Room," it publishes it to Redis. Redis then forwards this event to all other Socket.io servers (Server 2, Server 3), which then emit the message to their own connected clients.

4. How does Socket.io handle Disconnections and Reconnections?

Answer:

- **Heartbeat Mechanism:** Socket.io sends "Ping" and "Pong" packets at regular intervals. If the server doesn't receive a Pong within the `pingTimeout`, it considers the client disconnected.
- **Buffering:** When a client disconnects, Socket.io (client-side) buffers packets until it reconnects.
- **Connection State Recovery:** (Newer feature) The server temporarily stores the session state (rooms, missed packets) so that if a client reconnects quickly (e.g., switching from WiFi to Data), the session is restored without losing data.

5. What are "Acknowledgements" in Socket.io?

Answer:
Standard WebSockets are "fire and forget." Socket.io allows you to send a callback function as the last argument of `emit()`.

- **Example:** `socket.emit('update', data, (response) => { console.log(response); })`
- **Mechanism:** The server receives the event, processes it, and calls the callback. This sends a special acknowledgement packet back to the client, confirming the operation was successful.

Part 2: WebRTC (Peer-to-Peer Audio/Video)

6. What are the three main APIs of WebRTC?

Answer:

1. **`MediaStream` (aka `getUserMedia`):** Accesses the device's camera and microphone.
2. **`RTCPeerConnection`:** The core component. It handles the connection between peers, maintains the session, and handles bandwidth/media info (codecs, NAT traversal).
3. **`RTCDataChannel`:** A bidirectional channel for sending arbitrary data (text, files, gaming state) with very low latency, independent of audio/video.

7. What is "Signaling" in WebRTC and why isn't it part of the standard?

Answer:

- **What it is:** The process of exchanging connection information (IP addresses, ports, media codecs) *before* the peer-to-peer connection can start.
- **Why it's needed:** Two browsers don't know each other's IP or capabilities. They need a middleman to introduce them.
- **Not Standardized:** WebRTC specifies *how* to communicate once connected, but not *how* to connect. Developers must build their own signaling mechanism using technologies like **Socket.io**, WebSockets, or HTTP.

8. Explain STUN, TURN, and ICE. Why do we need them?

Answer:
These are required for **NAT Traversal** (connecting devices behind routers/firewalls).

- **ICE (Interactive Connectivity Establishment):** The framework/protocol that tries to find the best path to connect peers. It tries paths in this order: Direct → STUN → TURN.
- **STUN (Session Traversal Utilities for NAT):** A lightweight server that tells a client "Your Public IP is X and Port is Y." It helps peers connect directly (works ~80% of the time).
- **TURN (Traversal Using Relays around NAT):** A relay server used when STUN fails (e.g., Symmetric NAT or strict corporate firewalls). It receives data from Peer A and forwards it to Peer B. It is resource-intensive and costs money to run.

9. What is SDP (Session Description Protocol)?

Answer:
SDP is a standardized format (a string of text) that describes the multimedia communication session.

- **Content:** It contains details like: "I support VP8 and H.264 video codecs," "I am sending Audio and Video," and "My IP address is X."
- **Process:** This is what is exchanged during Signaling.
  1. Peer A creates an **Offer** (SDP).
  2. Peer B receives it and creates an **Answer** (SDP).

10. What are "ICE Candidates"?

Answer:
An ICE Candidate is a piece of network information that describes a possible method for one peer to connect to another.

- **Types:**
  - **Host:** The device's local IP (LAN).
  - **Srflx (Server Reflexive):** The public IP derived from a STUN server.
  - **Relay:** The IP address of the TURN server.
- **Trickle ICE:** Instead of waiting to gather *all* candidates (which is slow), WebRTC sends them one by one via the signaling server as soon as they are discovered.

Recommended Resource

... WebRTC Crash Course - STUN, TURN, ICE, SDP ...

This video is highly relevant because it visually breaks down the complex "Handshake" process of WebRTC (Signaling, ICE, STUN/TURN) which is the most difficult part of these interviews.

WebRTC Crash Course - YouTubeHussein Nasser · 280K views