PLACEMENT PREP · @CHALKESAHIL

80 CS fundamentals questions on OS, DBMS, networks and OOP

This guide is for students and early-career engineers preparing for placement interviews. It covers the four subjects interviewers ask about most: operating systems, databases, computer networks and OOP. After working through it, you should be able to explain each topic in plain words with a small example, the way a good interview answer sounds.

How to use this PDF

1. Read one section a day. Read the question first and try to answer it aloud before you read the answer.
2. Redo the small examples by hand: address translation, scheduling charts, normal forms and SQL queries.
3. Use the tip line under each answer as your closing sentence in an interview.
4. At the end of every section, answer the practice questions without looking, then mark the items you still stumble on and revisit them.

INSIDE

01 Processes and threads
10 questions

02 Memory management
10 questions

03 Scheduling, deadlocks
10 questions

04 DBMS and normalization
10 questions

05 Transactions, indexing
10 questions

06 Computer networks
10 questions

07 OOP and SOLID
10 questions

08 Rapid fire
10 questions

SECTION 01 / 08

Processes and threads

Almost every OS interview starts here. Be ready to explain what a process is, how it differs from a thread, and what goes wrong when threads share data. Use small real examples like a browser or a bank transfer.

01 Program vs process

Easy

What is the difference between a program and a process?

A program is a passive file on disk: compiled instructions and data, such as chrome.exe. A process is a program in execution. When you launch it, the OS loads the code into memory and gives it its own address space (code, data, heap and stack), a process ID and CPU time. One program can create many processes: open the same app twice and you get two separate processes from one file. Each process is isolated, so one crashing does not normally corrupt another. Think of a program as a recipe in a book and a process as a cook actually following it in a kitchen right now. Two cooks can follow the same recipe at once.

> **Interview tip.** Say it in one line: a process is a program in execution with its own address space and resources.

02 Process vs thread

Easy

What is the difference between a process and a thread?

A process is an independent unit with its own memory space. A thread is a lightweight unit of execution inside a process. Threads of the same process share code, heap and open files, but each thread has its own stack, registers and program counter. Because they share memory, threads talk to each other cheaply and are quick to create and switch. The cost is safety: one buggy thread can corrupt shared data or crash the whole process. Processes are heavier but isolated, and they need IPC (pipes, sockets, shared memory) to communicate. A browser may run each tab as a separate process for isolation, and use threads inside each one for rendering and network work.

> **Interview tip.** Common follow-up: what do threads share and what is private? Heap is shared, stack is private.

03 Process states

Easy

Explain the states of a process and how it moves between them.

A process moves through states during its life. New: being created. Ready: waiting in a queue for the CPU. Running: its instructions are executing. Waiting (or blocked): paused until an event such as disk I/O or user input finishes. Terminated: finished, with resources being cleaned up. The scheduler picks a ready process (ready to running). A timer interrupt preempts it (running to ready). An I/O request blocks it (running to waiting). I/O completion moves it back to ready. A waiting process never goes straight to running; it must queue as ready first. Some systems add suspended states for processes swapped out of memory.

```
New -> Ready -> Running -> Terminated
Running -> Waiting (I/O request)
Waiting -> Ready (I/O done)
Running -> Ready (time slice over)
```

> **Interview tip.** Draw the diagram on paper. Interviewers like the ready queue vs waiting queue distinction.

04 Process control block

Medium

What is a PCB and what does it store?

The process control block is the data structure the OS keeps for every process. It is the process's identity card. It stores the process ID, current state, program counter, saved CPU registers, scheduling information (priority, queue pointers), memory management information (page tables, limits), accounting information (CPU time used) and I/O status (open files, devices). When the OS switches away from a process, it saves the CPU state into that process's PCB. When it switches back, it reloads that state, so the process resumes exactly where it stopped. PCBs live in kernel memory, so user code cannot tamper with them. In Linux the equivalent structure is called task_struct.

> **Interview tip.** Tie it to context switching: the PCB is where the saved state goes.

05 Context switch

Medium

What is a context switch and why is it expensive?

A context switch is when the CPU stops running one process or thread and starts another. The kernel saves the current state (registers, program counter, stack pointer) into its PCB, picks the next one, and loads that one's saved state. It is pure overhead because no useful user work happens during the switch. Process switches cost more than thread switches because memory mappings change, and the TLB and CPU caches often lose useful entries. Switches are triggered by timer interrupts, blocking I/O, or a higher priority process arriving. Too many switches waste CPU time, which is why a very small time slice hurts throughput.

> **Interview tip.** Why are thread switches cheaper? Same address space, so no page table swap.

06 Inter-process communication

Medium

How do two processes communicate with each other?

Processes have separate memory, so they need OS help to exchange data. Pipes are a one-way byte stream, like the output of one command feeding the next. Named pipes (FIFOs) work between unrelated processes. Message queues let the kernel store structured messages. Shared memory is a region both processes map; it is the fastest, because data is not copied through the kernel after setup, but it needs synchronization. Sockets also work across machines. Signals are tiny notifications such as SIGTERM. Message passing is safer and easier to reason about, shared memory is faster. Pick based on data size, speed needs and whether both processes run on one machine.

> **Sample answer**
>
> ls | grep .txt
>
> # the pipe sends the output of ls into grep as input

> **Interview tip.** Name at least four methods and say which is fastest (shared memory).

07 Why use multithreading

Easy

What are the benefits and risks of multithreading?

Multithreading lets one program do several things at once. Benefits: responsiveness (a UI thread stays alive while another thread downloads a file), resource sharing (threads share memory, so there is no copying), lower overhead than processes, and use of multiple CPU cores. A web server can handle each request on its own thread. Risks: race conditions, deadlocks, harder debugging, and synchronization costs. More threads do not always mean faster. On a CPU-bound task, more threads than cores just adds switching overhead. Concurrency means tasks make progress in overlapping time. Parallelism means they literally run at the same instant on different cores.

> **Interview tip.** Be ready to separate concurrency from parallelism. It is a favourite follow-up.

08 User vs kernel threads

Medium

What is the difference between user-level and kernel-level threads?

User-level threads are created and scheduled by a library in user space, and the kernel sees only one process. They are fast to create and switch, but if one thread makes a blocking system call the whole process blocks, and they cannot use multiple cores. Kernel-level threads are managed by the OS, so the scheduler can run them on different cores and one blocking does not stop the others. They cost more to create and switch because the kernel is involved. Mapping models connect the two: many-to-one, one-to-one and many-to-many. Mainstream systems like Linux and Windows mostly use one-to-one.

> **Interview tip.** Mention that one-to-one is the common model today.

09 Race condition

Medium

What is a race condition and a critical section?

A race condition happens when the result depends on the unpredictable order in which threads touch shared data. The classic case is count += 1, which is really three steps: read, add, write. Two threads can both read 5, both write 6, and one increment is lost. The code that touches the shared data is the critical section. A correct solution needs mutual exclusion (only one thread inside at a time), progress (a waiting thread gets in when nobody is inside) and bounded waiting (nobody waits forever). Fix it with locks, atomic operations, or by avoiding shared mutable state.

```
import threading

count = 0
lock = threading.Lock()

def work():
    global count
    for _ in range(100000):
        with lock:
            count += 1

threads = [threading.Thread(target=work) for _ in range(4)]
for t in threads: t.start()
for t in threads: t.join()
print(count)  # always 400000 with the lock
```

> **Interview tip.** Always give the counter example. It makes the idea clear in ten seconds.

10 Mutex vs semaphore

Medium

What is the difference between a mutex and a semaphore?

A mutex is a lock with ownership: the thread that locks it must be the one to unlock it. It protects a critical section so one thread enters at a time. A semaphore is an integer counter with two atomic operations: wait (decrement, block if zero) and signal (increment). A binary semaphore takes values 0 and 1. A counting semaphore allows N threads at once, which is useful for a pool of N database connections. Semaphores have no owner, so one thread can signal while another waits, which suits producer and consumer problems. Rule of thumb: mutex for mutual exclusion, semaphore for counting resources or signalling between threads.

> **Interview tip.** Is a binary semaphore the same as a mutex? No, ownership is the difference.

Test yourself

1. Why can threads of one process crash each other but processes usually cannot?
2. Two threads run count += 1 a thousand times each without a lock. What final values are possible and why?

SECTION 02 / 08

Memory management

Memory questions test whether you understand how the OS gives each program its own space. Learn paging and virtual memory first, then page faults and replacement. Work through the small numeric examples by hand.

11 Paging

Easy

What is paging and how does address translation work?

Paging splits a process's logical memory into fixed-size pages and physical memory into frames of the same size, commonly 4 KB. Any page can sit in any free frame, so memory need not be contiguous. A page table maps each page number to a frame number, and every process has its own table. A logical address is split into a page number and an offset. The page number indexes the table, and the offset is kept as is. Paging removes external fragmentation and makes allocation simple. The last page of a process may be partly empty, which is internal fragmentation.

```
Page size 4096 bytes. Logical address 8200.
Page number = 8200 / 4096 = 2, offset = 8200 mod 4096 = 8.
If page 2 maps to frame 5: physical = 5 * 4096 + 8 = 20488.
```

> **Interview tip.** Always show one worked address translation when you explain paging.

12 Segmentation

Medium

What is segmentation and how is it different from paging?

Segmentation divides memory by the logical units a programmer sees: code, data, stack and heap, each a segment of variable size. An address is a pair of segment number and offset. A segment table stores each segment's base and limit, and the hardware checks that the offset is below the limit, otherwise it raises a fault. Segments match how programs are organized, so sharing and protection are natural: a code segment can be read-only. The downside is external fragmentation, because variable-size chunks leave holes. Paging uses fixed sizes and has no external fragmentation but ignores program structure. Some systems combine them: segments first, then paging inside each segment.

> **Interview tip.** Paging is invisible to the programmer. Segmentation is visible.

13 Virtual memory

Medium

What is virtual memory and why do we need it?

Virtual memory gives each process the illusion of a large, private, contiguous address space, even when physical RAM is small. Only the pages being used need to be in RAM. The rest stay on disk in swap space. The OS and a hardware unit called the MMU translate virtual addresses to physical ones on every access. Benefits: programs bigger than RAM can run, processes are isolated from each other, more processes fit in memory at once, and programs start faster because pages load on demand. The price is that touching a page not in RAM needs a disk access, which is far slower than a memory access.

> **Interview tip.** Say demand paging is how virtual memory is usually implemented.

14 Page fault

Medium

What happens when a page fault occurs?

A page fault occurs when a process accesses a page that is valid but not currently in RAM. The hardware traps to the OS, which then checks the address is legal (otherwise it is a segmentation fault), finds a free frame or evicts a page using a replacement algorithm, reads the needed page from disk into that frame, updates the page table, and restarts the instruction that faulted. Page faults are normal under demand paging, but each one involves disk I/O, so a high fault rate slows the system badly. Effective access time is (1 - p) times memory time plus p times fault service time, where p is the fault rate.

> **Interview tip.** A page fault is an exception, not always an error. Do not confuse it with a segmentation fault.

15 Page replacement

Medium

Explain FIFO, LRU and Optimal page replacement.

When RAM is full and a new page is needed, the OS must evict one. FIFO evicts the oldest page. It is simple but can remove a page that is used heavily. Optimal evicts the page that will not be used for the longest time in the future. It gives the fewest faults but needs knowledge of the future, so it is only a benchmark. LRU evicts the page unused for the longest time, using the past as a guide. It performs well, but exact tracking is costly, so real systems use approximations such as the clock algorithm.

```
Reference string 7 0 1 2 0 3, 3 frames, FIFO:
7, 0, 1 load (3 faults)
2 evicts 7 (4th fault)
0 is a hit
3 evicts 0 (5th fault)
Total faults = 5
```

> **Interview tip.** Practice with a reference string and count faults. Interviewers often ask for exactly that.

16 Belady's anomaly

Hard

What is Belady's anomaly?

Normally, giving a process more frames reduces page faults. Belady's anomaly is the surprising case where more frames cause more faults. It happens with FIFO. For the reference string 1 2 3 4 1 2 5 1 2 3 4 5, FIFO gives 9 faults with 3 frames but 10 faults with 4 frames. LRU and Optimal never show it because they are stack algorithms: the set of pages kept with n frames is always a subset of the set kept with n + 1 frames. FIFO ignores recency, so the extra frame can change the eviction order in an unhelpful way.

> **Interview tip.** Remember the string and the numbers 9 and 10. It proves you understand it.

17 Thrashing

Medium

What is thrashing and how do you prevent it?

Thrashing is when the system spends more time swapping pages in and out than doing useful work. It happens when too many processes are loaded and each gets too few frames, so each one faults constantly. CPU utilization drops, the OS may see the low utilization and admit even more processes, and things get worse. The working set model helps: track the set of pages a process has used in a recent window, and run the process only if that set fits in memory. Other fixes: reduce the number of running processes, use local replacement, add RAM, or control the page fault frequency.

> **Interview tip.** Explain the vicious cycle: low CPU use leads to more processes, which leads to more faults.

18 Fragmentation types

Medium

What is internal and external fragmentation?

Internal fragmentation is wasted space inside an allocated block. You ask for 17 KB, get a 20 KB block, and 3 KB is wasted. It happens with fixed-size partitions and with paging (the last page). External fragmentation is free memory split into small scattered holes, so the total free space is enough but no single hole is big enough. It happens with variable-size allocation and segmentation. Paging removes external fragmentation. Compaction, which moves blocks together, also fixes it but is costly. First-fit, best-fit and worst-fit are strategies for choosing which hole to use.

> **Interview tip.** One line: internal means wasted inside a block, external means wasted between blocks.

19 TLB

Medium

What is the TLB and why does it speed things up?

Every memory access needs a page table lookup, and the page table itself lives in memory, so a naive system would read memory twice for each access. The Translation Lookaside Buffer is a small, very fast hardware cache that stores recent page-to-frame translations. On each access the MMU checks the TLB first. A hit gives the frame immediately. A miss means walking the page table, and then the entry is loaded into the TLB. Because programs show locality (they reuse nearby pages), hit rates are usually high, so most accesses avoid the extra memory read.

```
Memory access 100 ns, TLB lookup 10 ns, hit ratio 90%, single-level page table.
EAT = 0.9 * (10 + 100) + 0.1 * (10 + 100 + 100)
    = 99 + 21 = 120 ns
```

> **Interview tip.** Be ready to compute effective access time. Use the formula with hit ratio.

20 Stack vs heap

Easy

What is the difference between stack and heap memory?

The stack holds function call frames: local variables, parameters and return addresses. It is automatic, allocation just moves a pointer, memory is freed when the function returns, and it is small and fixed in size. Overflowing it, for example with infinite recursion, causes a stack overflow. The heap is for dynamic memory that must outlive a function call, such as objects created with new or malloc. It is larger and flexible but slower. In C and C++ you free heap memory manually, so leaks and dangling pointers are possible. Java and Python use garbage collection for the heap. Each thread has its own stack, while the heap is shared.

> **Interview tip.** Mention that each thread has its own stack but shares the heap.

Test yourself

1. A system has 4 KB pages. What are the page number and offset of logical address 10000?
2. Why does adding more RAM often fix thrashing but adding a faster CPU does not?

SECTION 03 / 08

Scheduling, deadlocks

Scheduling questions are mostly small calculations, and deadlock questions are mostly the four conditions plus one fix. Practice drawing Gantt charts and computing average waiting time. Know which algorithm can starve a process.

21 Preemptive vs non-preemptive

Easy

What is CPU scheduling and what is the difference between preemptive and non-preemptive scheduling?

The scheduler decides which ready process gets the CPU next. In non-preemptive scheduling, a process keeps the CPU until it finishes or blocks. It is simple, but one long job can hold everyone up. In preemptive scheduling the OS can take the CPU away, for example when a timer expires or a higher priority process arrives. This gives better response time and fairness but needs more context switches and care with shared data. Common goals: high CPU utilization and throughput, and low turnaround time (completion minus arrival), waiting time (time spent in the ready queue) and response time (arrival to first run).

> **Interview tip.** Know the formulas: turnaround = completion - arrival, waiting = turnaround - burst.

22 FCFS and convoy effect

Easy

Explain First Come First Served scheduling and the convoy effect.

First Come First Served runs processes in the order they arrive, like a queue at a counter. It is non-preemptive and very simple, but average waiting time can be poor. The convoy effect is when one long CPU-bound process at the front makes many short processes wait behind it, and the I/O devices sit idle meanwhile. The order of arrival changes the result a lot, as the example shows. FCFS is fair in the sense that nobody starves, but it is a poor choice for interactive systems.

```
Bursts P1=24, P2=3, P3=3, all arrive at time 0.
Order P1, P2, P3: waits 0, 24, 27, average 17.
Order P2, P3, P1: waits 0, 3, 6, average 3.
```

> **Interview tip.** Show the same jobs in two orders to prove the convoy effect.

23 SJF and starvation

Medium

Explain Shortest Job First scheduling. What is its main problem?

Shortest Job First picks the process with the smallest next CPU burst. It is provably optimal for minimum average waiting time. The preemptive version is called Shortest Remaining Time First (SRTF): if a new process arrives with a shorter remaining time, it takes the CPU. There are two problems. First, the next burst length is not known in advance, so it is predicted from past bursts using exponential averaging. Second, long processes can starve if short jobs keep arriving. So SJF is excellent in theory but needs a good predictor and some fairness protection.

```
Bursts 6, 8, 7, 3, all arrive at time 0.
SJF order: 3, 6, 7, 8.
Waits: 0, 3, 9, 16. Average = 28 / 4 = 7.
```

> **Interview tip.** Say it is optimal for average waiting time but can starve long jobs.

24 Round robin

Medium

How does Round Robin scheduling work and how do you pick the time quantum?

Round Robin gives each ready process a fixed time slice, called the quantum, in turn. If the process is not finished, it goes to the back of the queue. It is preemptive and fair, with good response time, which suits time-sharing systems. The quantum is the key choice. If it is too large, Round Robin behaves like FCFS. If it is too small, context switch overhead dominates. A common rule of thumb is that most CPU bursts should finish within one quantum. Average waiting time is often higher than SJF, but no process starves.

> **Sample answer**
>
> P1 burst 5, P2 burst 3, quantum 2.
>
> Gantt: P1(0-2) P2(2-4) P1(4-6) P2(6-7) P1(7-8)

> **Interview tip.** Mention the quantum trade-off: too big becomes FCFS, too small wastes time switching.

25 Priority scheduling, aging

Medium

What is priority scheduling and how does aging solve starvation?

Each process has a priority, and the CPU goes to the highest priority ready process. It can be preemptive or non-preemptive. Low priority processes may starve if high priority work keeps arriving. Aging fixes this by slowly raising the priority of a process the longer it waits, so every process eventually runs. Priority can be internal (set by the OS from memory use or I/O behaviour) or external (set by the user or by importance). A multilevel feedback queue builds on this idea: new processes start in a high priority queue and CPU hogs sink to lower queues, which favours short and interactive jobs without knowing burst times.

> **Interview tip.** Starvation is the problem, aging is the answer. Say both together.

26 Deadlock and 4 conditions

Easy

What is a deadlock? What are the four necessary conditions?

A deadlock is when a set of processes are each waiting for a resource held by another process in the set, so none can proceed. Think of two cars meeting on a one-lane bridge, each waiting for the other to reverse. Four Coffman conditions must all hold together. Mutual exclusion: a resource is used by one process at a time. Hold and wait: a process holds some resources while waiting for more. No preemption: resources cannot be forcibly taken away. Circular wait: there is a cycle of processes, each waiting for a resource held by the next. Break any one condition and deadlock cannot happen.

```
Thread 1: lock(A); lock(B)
Thread 2: lock(B); lock(A)
If each gets its first lock, both wait forever.
```

> **Interview tip.** List all four by name. Missing one is the most common mistake.

27 Deadlock prevention

Medium

How can deadlocks be prevented?

Prevention means designing the system so at least one Coffman condition can never hold. Mutual exclusion: make resources sharable where possible, such as read-only files, though many cannot be. Hold and wait: make a process request all its resources at once, or release everything before asking again. This lowers utilization and can cause starvation. No preemption: if a request fails, the process releases what it holds and retries later. Circular wait: number every resource and require processes to request them in increasing order. This is the most practical, and in real code it appears as lock ordering.

> **Interview tip.** In code, say: always acquire locks in one fixed global order.

28 Banker's algorithm

Hard

Explain the Banker's algorithm and what a safe state is.

The Banker's algorithm is deadlock avoidance. Each process declares its maximum need in advance. The OS grants a request only if the system stays in a safe state, meaning there is some order in which all processes can finish. It tracks Available, Max, Allocation and Need (Need = Max - Allocation). To check safety, find a process whose Need fits in Available, assume it finishes and returns its Allocation, add that to Available, and repeat. If every process can finish, the state is safe. An unsafe state does not mean deadlock, only that deadlock is possible. The limit is that it needs maximum demands up front, so real systems rarely use it.

```
12 units total. Allocation / Max: P0 5/10, P1 2/4, P2 2/9. Available = 3.
Need: P0 5, P1 2, P2 7.
P1 fits (2 <= 3), finishes, Available = 5.
P0 fits (5 <= 5), finishes, Available = 10.
P2 fits (7 <= 10). Safe sequence: P1, P0, P2.
```

> **Interview tip.** Remember: unsafe does not equal deadlocked. It means deadlock is possible.

29 Detection and recovery

Medium

How are deadlocks detected and recovered from?

If prevention and avoidance are too restrictive, the system allows deadlocks and handles them later. Detection: build a resource allocation graph or a wait-for graph. With a single instance of each resource, a cycle means deadlock. With multiple instances, a Banker-like algorithm checks it. The detector runs periodically or when CPU utilization drops. Recovery options: terminate all deadlocked processes, terminate them one at a time until the cycle breaks (choosing by priority, work done and resources held), or preempt resources and roll a process back to a safe point. Many general-purpose systems simply ignore the problem, which is called the ostrich algorithm, since deadlocks are rare.

> **Interview tip.** Mention the ostrich algorithm. It shows you know what real systems do.

30 Deadlock vs starvation

Medium

What is the difference between deadlock, starvation and livelock?

In a deadlock, processes wait on each other in a cycle and none can ever proceed. In starvation, a process is ready but never gets the resource or the CPU because others keep getting preference, for example a low priority job under heavy high priority load. Other processes keep working, and the fix is aging. In a livelock, processes keep changing state in response to each other but make no progress, like two people in a corridor who both step aside the same way again and again. They are not blocked. They are busy but useless. Deadlock is stuck, starvation is ignored, livelock is busy without progress.

> **Interview tip.** Use the corridor example for livelock. It is easy to remember.

Test yourself

1. Three processes with bursts 5, 2 and 8 arrive at time 0. Find the average waiting time under FCFS and SJF.
2. Two bank transfer threads deadlock on two account locks. Give two different fixes.

SECTION 04 / 08

DBMS and normalization

DBMS rounds begin with keys, joins and normal forms. Normalization questions are almost always solved with one small table and a clear dependency. Learn to spot partial and transitive dependencies on sight.

31 DBMS vs file system

Easy

Why do we use a DBMS instead of storing data in files?

A file system stores data as plain files, and each application manages its own format. This leads to data redundancy and inconsistency (the same data copied in many files), difficult queries, no control over concurrent access, weak security and no crash recovery. A DBMS stores data centrally and provides a query language (SQL), integrity constraints, concurrency control, transactions, backup and recovery, and access control. It also gives data independence: you can change how data is stored without changing the applications that use it. The cost is more complexity, software overhead and expense. For a tiny single-user settings file, a plain file is fine. For shared, structured, important data, use a DBMS.

> **Interview tip.** Name at least four problems of file systems: redundancy, inconsistency, no concurrency control, no recovery.

32 Keys in DBMS

Easy

Explain super key, candidate key, primary key and foreign key.

A super key is any set of columns that uniquely identifies a row. A candidate key is a minimal super key, with no unnecessary column. The primary key is the candidate key you choose to identify rows. It cannot be NULL or duplicate. The other candidates are called alternate keys. A foreign key is a column that refers to the primary key of another table, and it enforces referential integrity, so you cannot point to a row that does not exist. A composite key uses more than one column together. Every candidate key is a super key, but not every super key is a candidate key.

> **Sample answer**
>
> students(student_id, email, name)
>
> student_id and email are both candidate keys.
>
> student_id is chosen as the primary key, email is an alternate key.
>
> enrollments(student_id, course_id) has student_id as a foreign key to students.

> **Interview tip.** Every candidate key is a super key, but not the other way round. Say that.

33 DDL, DML, DCL, TCL

Easy

What are DDL, DML, DCL and TCL commands?

SQL commands are grouped by purpose. DDL (Data Definition Language) changes structure: CREATE, ALTER, DROP, TRUNCATE. DML (Data Manipulation Language) changes data: INSERT, UPDATE, DELETE. SELECT is often placed in its own group, DQL, for querying. DCL (Data Control Language) manages permissions: GRANT and REVOKE. TCL (Transaction Control Language) manages transactions: COMMIT, ROLLBACK and SAVEPOINT. In many databases, such as MySQL and Oracle, a DDL statement commits automatically, so you cannot roll it back. DML can be rolled back until you commit. PostgreSQL is an exception and supports transactional DDL.

> **Interview tip.** In MySQL and Oracle DDL auto-commits. PostgreSQL lets you roll DDL back.

34 SQL joins

Easy

Explain the different types of SQL joins.

A join combines rows from two tables using a related column. INNER JOIN returns only the rows that match in both tables. LEFT JOIN returns all rows from the left table, plus matches from the right, with NULL where there is no match. RIGHT JOIN is the mirror image. FULL OUTER JOIN returns all rows from both tables, with NULL where either side has no match. CROSS JOIN returns every pairing of rows, called a Cartesian product, so 10 rows by 10 rows gives 100. A self join joins a table to itself, for example employees to their managers. Pick LEFT JOIN when you must keep rows that have no match.

```
SELECT s.name, c.title
FROM students s
LEFT JOIN courses c ON c.id = s.course_id;
-- students with no course still appear, with title NULL
```

> **Interview tip.** Draw two overlapping circles to explain. Then give one real use of LEFT JOIN.

35 Why normalization

Easy

What is normalization and what anomalies does it prevent?

Normalization organizes tables to reduce redundancy and avoid anomalies. Take one table orders(order_id, customer_name, customer_city, item). Customer details repeat on every row. Update anomaly: changing a city means editing many rows, and missing one leaves inconsistent data. Insert anomaly: you cannot add a customer who has not ordered anything yet. Delete anomaly: deleting a customer's last order also erases the customer's details. Normalization splits the data into customers and orders tables linked by a key. It works through functional dependencies (A determines B) and a series of normal forms: 1NF, 2NF, 3NF, BCNF and beyond.

> **Interview tip.** Always name the three anomalies: update, insert and delete.

36 First normal form

Easy

What is 1NF?

A table is in first normal form when every column holds atomic (indivisible) values, there are no repeating groups, and each row is unique. It is the minimum bar for a relational table. A cell holding a comma separated list breaks it, because you cannot index or search a single value cleanly and cannot put constraints on the parts. Repeating columns like phone1, phone2 and phone3 also break it, because they cap the count and make queries awkward. The fix is to move repeating data into its own table, with one row per value, linked back by a key. This keeps searches simple and removes any limit on how many values one record can have.

```
Bad:  student(id, name, phones)  with phones = '98760, 91230'
Good: student(id, name)
      student_phone(student_id, phone)  one row per number
```

Interview tip. Atomic means one value per cell. Say it that plainly.

37 Second normal form

Medium

What is 2NF? Give an example of a partial dependency.

A table is in second normal form if it is in 1NF and no non-key column depends on only part of a composite primary key. That unwanted situation is called a partial dependency. 2NF only matters when the key has more than one column. In the example, the key is (student_id, course_id). The column student_name depends only on student_id, which is part of the key, so it is repeated for every course the student takes. Splitting the table fixes it. The grade depends on the whole key, because a grade belongs to one student in one course, so it stays in the enrollment table.

```
Bad:  enrollment(student_id, course_id, student_name, grade)
Good: students(student_id, student_name)
      enrollment(student_id, course_id, grade)
```

Interview tip. If the primary key is a single column, a 1NF table is already in 2NF.

38 Third normal form

Medium

What is 3NF? Give an example of a transitive dependency.

A table is in third normal form if it is in 2NF and no non-key column depends on another non-key column. That unwanted case is called a transitive dependency: A determines B, and B determines C, so C depends on A only through B. A handy summary is that every non-key column must depend on the key, the whole key, and nothing but the key. In the example, dept_name depends on dept_id, and dept_id depends on emp_id. So the department name is repeated for each employee in that department. Move departments into their own table, and a rename happens in one place.

```
Bad:  employee(emp_id, name, dept_id, dept_name)
Good: employee(emp_id, name, dept_id)
      department(dept_id, dept_name)
```

Interview tip. Remember the phrase: the key, the whole key and nothing but the key.

39 BCNF

Hard

What is BCNF and how is it different from 3NF?

Boyce-Codd Normal Form is a stricter version of 3NF. A table is in BCNF if, for every functional dependency X determines Y, X is a super key. 3NF allows one exception: when Y is a prime attribute, meaning part of some candidate key. BCNF does not allow it. In the example, each teacher teaches one subject, so teacher determines subject, but teacher is not a super key, which violates BCNF. Decompose into (teacher, subject) and (student, teacher). The trade-off is that a BCNF decomposition may lose the ability to enforce some dependencies, whereas 3NF decomposition can always preserve them.

```
class(student, subject, teacher)
Rules: teacher -> subject. Each student has one teacher per subject.
Candidate keys: (student, subject) and (student, teacher).
teacher -> subject breaks BCNF.
Split: (teacher, subject) and (student, teacher).
```

> **Interview tip.** BCNF is a stricter 3NF. The difference only shows with overlapping candidate keys.

40 Denormalization

Medium

What is denormalization and when would you use it?

Denormalization deliberately adds redundancy back to make reads faster. Normalized data needs joins, and joins on huge tables cost time. By storing the customer name directly in the orders table, or keeping a precomputed order_total, reads can avoid joins. The trade-offs are more storage, more complex writes because the copies must be kept in sync, and a risk of inconsistency. It is common in reporting, analytics and read-heavy systems, and in NoSQL databases where data is modelled around the queries. A good rule is to normalize first, measure, and denormalize only the hot paths that really need it.

> **Interview tip.** Say you normalize first and denormalize only after measuring a real slow query.

Test yourself

1. A table has columns (order_id, product_id, product_name, quantity) with key (order_id, product_id). Which normal form does it break and how do you fix it?
2. Write a query that lists all students and their course titles, including students with no course.

SECTION 05 / 08

Transactions, indexing

These topics decide whether a database stays correct and fast. Learn ACID and isolation levels with one bank example, then understand indexes as sorted lookup structures. Expect why and when questions, not just definitions.

41 ACID properties

Easy

What are the ACID properties of a transaction?

A transaction is a group of operations treated as one unit. Atomicity: all operations happen or none do, usually implemented with an undo log. Consistency: a transaction takes the database from one valid state to another, respecting all constraints. Isolation: concurrent transactions do not see each other's half-done work. Durability: once committed, the data survives crashes, usually thanks to a write-ahead log on disk. Together they make money transfers and bookings safe. Note that consistency in ACID is about data rules, and it is different from the consistency in the CAP theorem, which is about replicas agreeing.

```
Transfer 500 from A to B:
UPDATE accounts SET balance = balance - 500 WHERE id = 'A';
UPDATE accounts SET balance = balance + 500 WHERE id = 'B';
COMMIT;
If the second update fails, ROLLBACK undoes the first (atomicity).
```

> **Interview tip.** Use the bank transfer example for all four letters.

42 Isolation levels

Medium

Explain the four transaction isolation levels.

The SQL standard defines four levels, trading correctness for speed. Read Uncommitted: you can read changes that are not yet committed (dirty reads). Read Committed: you see only committed data, but reading a row twice may give different results. Repeatable Read: a row you read once returns the same values again within your transaction. Serializable: transactions behave as if they ran one after another, the safest and slowest. Higher isolation means fewer anomalies but more locking or conflicts. Defaults differ: PostgreSQL and Oracle default to Read Committed, while MySQL InnoDB defaults to Repeatable Read. Exact behaviour also varies by database.

> **Interview tip.** Say the default level of the database you use. It shows real experience.

43 Dirty and phantom reads

Medium

What are dirty reads, non-repeatable reads and phantom reads?

A dirty read means you read data that another transaction changed but has not committed. If that transaction rolls back, you have read something that never existed. A non-repeatable read means you read a row twice and it changed in between, because another transaction committed an update. A phantom read means you run the same range query twice and new rows appear, because another transaction inserted matching rows. Read Committed stops dirty reads. Repeatable Read also stops non-repeatable reads. Serializable stops all three. Some databases, such as PostgreSQL and MySQL InnoDB, also block phantoms at Repeatable Read.

```
T1: SELECT COUNT(*) FROM orders WHERE city = 'Pune';  -- 10
T2: INSERT a Pune order; COMMIT;
T1: SELECT COUNT(*) FROM orders WHERE city = 'Pune';  -- 11 (phantom)
```

> **Interview tip.** Phantoms are about new rows. Non-repeatable reads are about changed rows.

44 Locks and 2PL

Hard

What are shared and exclusive locks? What is two-phase locking?

Locks control concurrent access. A shared (read) lock lets many readers in at once. An exclusive (write) lock allows one writer and blocks everyone else. Two-phase locking has a growing phase, where a transaction only acquires locks, and a shrinking phase, where it only releases them. Once it releases any lock, it cannot acquire a new one. This rule guarantees serializable schedules. Strict 2PL holds exclusive locks until commit, which prevents cascading rollbacks. The downside is that 2PL can cause deadlocks. The database detects them with a wait-for graph and breaks them by aborting one transaction.

> **Interview tip.** 2PL guarantees serializability, but it can still deadlock.

45 Optimistic vs pessimistic

Medium

What is the difference between optimistic and pessimistic locking?

Pessimistic locking assumes conflicts are likely. You lock the row first (for example with SELECT ... FOR UPDATE), do your work, then commit. It is safe, but other users wait and deadlocks are possible. Optimistic locking assumes conflicts are rare. You read without locking, and at write time you check that nothing changed, usually with a version column. If the version changed, the update affects zero rows and the application retries. Optimistic suits read-heavy workloads and long user think time, like editing a form. Pessimistic suits high contention, such as stock of a very popular item.

```
UPDATE products
SET stock = stock - 1, version = version + 1
WHERE id = 7 AND version = 3;
-- 0 rows updated means someone else changed it first, so retry
```

> **Interview tip.** Give one example where each is the better choice.

46 How indexes work

Easy

What is a database index and how does it speed up queries?

An index is a separate sorted structure that lets the database find rows without scanning the whole table, like the index at the back of a book. Without one, WHERE email = 'a@b.com' is a full table scan that checks every row. With an index on email, the database jumps to the match in logarithmic time. Most indexes are B+ trees. The costs are extra disk space, and every INSERT, UPDATE or DELETE must also update the index, so writes get slower. Index columns that you use often in WHERE, JOIN and ORDER BY, and do not index everything.

```
CREATE INDEX idx_users_email ON users(email);
SELECT * FROM users WHERE email = 'a@b.com';  -- now uses the index
```

> **Interview tip.** Always mention the cost: indexes slow down writes and use space.

47 B-tree vs B+ tree

Hard

What is the difference between a B-tree and a B+ tree?

Both are balanced, sorted, multi-way trees designed for disks. Each node holds many keys, so the tree stays shallow, often only three or four levels for millions of rows. In a B-tree, keys and data pointers live in every node, internal ones too. In a B+ tree, internal nodes store only keys for routing, and all data pointers sit in the leaves, which are linked together in a list. Because of that leaf chain, range queries such as BETWEEN and ORDER BY are fast: find the start, then walk along the leaves. Internal nodes hold more keys, so the tree is shorter. Most relational databases use B+ trees.

> **Interview tip.** Key points: data only in leaves, and leaves are linked for range scans.

48 Clustered vs non-clustered

Medium

What is the difference between a clustered and a non-clustered index?

A clustered index decides the physical order of the rows: the table data is stored in the index's order, so there can be only one per table. In MySQL InnoDB, the primary key is the clustered index. A non-clustered (secondary) index is a separate structure whose entries point to the rows, either by row location or by primary key value, and a table can have many of them. Clustered is fast for range scans on its key. A secondary index lookup may need an extra hop to fetch the full row, unless the index covers the query. Terms vary by database: PostgreSQL tables are heaps and all of its indexes are secondary.

> **Interview tip.** Only one clustered index per table, because rows can be sorted only one way.

49 Composite index rule

Hard

What is a composite index and what is the leftmost prefix rule?

A composite index covers several columns in a fixed order. It works like a phone book sorted by last name, then first name. The leftmost prefix rule says the index helps queries that use the first column, or the first two, and so on, but not queries that skip the leading column. Column order therefore matters: put columns compared with equality first and range columns last. A covering index contains every column a query needs, so the database never touches the table itself. Too many composite indexes slow writes, so build them for your real queries.

```
CREATE INDEX idx_city_age ON users(city, age);
-- uses the index: WHERE city = 'Pune'
-- uses the index: WHERE city = 'Pune' AND age = 25
-- cannot use it well: WHERE age = 25
```

> **Interview tip.** Ask which column the query filters on first. That decides the order.

50 Write-ahead logging

Hard

What is write-ahead logging and how does it help recovery?

Write-ahead logging gives durability and atomicity. The rule is that before a changed data page is written to disk, a log record describing the change must be written to the log on disk first. On commit, only the log needs flushing, which is a fast sequential write, while the data pages can be written later. After a crash, recovery reads the log. It redoes committed changes that never reached the data files and undoes changes of transactions that never committed. A checkpoint records a point where the data files are known to be up to date, which limits how much log must be replayed. PostgreSQL and MySQL InnoDB both use this idea.

> **Interview tip.** Say: log first, data later. That one line is the whole idea.

Test yourself

1. Two users book the last seat at the same time. How would you stop a double booking using locking, and using a version column?
2. A table has an index on (country, city). Which of these queries can use it well: country only, city only, both?

SECTION 06 / 08

Computer networks

Networking questions follow the path of a request: layers, then TCP, then DNS, then HTTP. The most common closer is what happens when you type a URL. Learn it as one story, and be ready to go deeper at any step.

51 OSI model

Easy

Explain the seven layers of the OSI model.

The OSI model splits networking into seven layers, each with one job. Layer 7 Application: user-facing protocols like HTTP and SMTP. Layer 6 Presentation: encoding, encryption and compression. Layer 5 Session: opening, managing and closing conversations. Layer 4 Transport: end-to-end delivery, ports and reliability (TCP, UDP). Layer 3 Network: addressing and routing between networks (IP, routers). Layer 2 Data Link: delivery on one local link using MAC addresses (Ethernet, switches). Layer 1 Physical: bits as signals over cable or radio. On the way down, each layer wraps data with its own header, called encapsulation, and the receiver unwraps it on the way up.

> **Interview tip.** Memory trick from layer 1 up: Please Do Not Throw Sausage Pizza Away.

52 OSI vs TCP/IP

Medium

What is the difference between the OSI and TCP/IP models?

OSI is a seven-layer reference model. TCP/IP is the four-layer model the internet actually runs on. The TCP/IP layers are Application (covering OSI's top three layers), Transport, Internet (OSI's Network layer, where IP lives), and Network Access, also called Link (OSI's Data Link and Physical layers). OSI was designed as a general theory first and protocols came later. TCP/IP grew out of working protocols. In practice, engineers use OSI layer numbers as a shared vocabulary. A layer 4 load balancer works on TCP or UDP ports. A layer 7 load balancer looks inside the HTTP request, such as the URL or headers.

> **Interview tip.** Use the layer numbers casually: layer 4 means transport, layer 7 means application.

53 TCP vs UDP

Easy

What is the difference between TCP and UDP? When do you use each?

Both are transport layer protocols. TCP is connection-oriented and reliable. It sets up a connection, numbers bytes, acknowledges them, retransmits lost segments, delivers data in order, and controls flow and congestion. That makes it slower, with more overhead. UDP is connectionless. It just sends datagrams with no handshake, no ordering and no retransmission. It is light and fast, but packets can be lost, duplicated or reordered. Use TCP for web pages, email and file transfer, where correctness matters. Use UDP for live video, voice calls, games and DNS queries, where late data is useless and speed matters more. QUIC, used by HTTP/3, builds reliability on top of UDP.

> **Interview tip.** Give one example for each. Live video on UDP is the classic one.

54 TCP three-way handshake

Medium

How does the TCP three-way handshake work?

Before sending data, TCP sets up a connection in three steps. First, SYN: the client sends a segment with a starting sequence number. Second, SYN-ACK: the server acknowledges it and sends its own sequence number. Third, ACK: the client acknowledges the server's number. Now both sides know the other is reachable and agree on starting numbers, and data can flow both ways. Closing uses four steps (FIN, ACK, FIN, ACK), because each direction is closed separately. Why three and not two? The server needs proof that the client can receive too, and old duplicate SYN packets must not create phantom connections.

```
Client -> Server: SYN seq=100
Server -> Client: SYN-ACK seq=300 ack=101
Client -> Server: ACK ack=301
```

> **Interview tip.** Be ready for: why three steps and not two?

55 How DNS works

Medium

How does DNS resolve a domain name?

DNS translates names like example.com into IP addresses. Your browser checks its cache, then the OS cache, then asks a recursive resolver, usually run by your ISP or a public provider. If the resolver has no cached answer, it asks a root server, which points to the .com servers. Those point to the domain's authoritative name server, which returns the IP. The resolver sends the answer back and caches it for the record's TTL (time to live). Common record types: A (IPv4 address), AAAA (IPv6), CNAME (alias), MX (mail) and NS (name servers). DNS mostly uses UDP on port 53, and falls back to TCP for large answers.

> **Interview tip.** Name the chain: cache, resolver, root, TLD, authoritative.

56 HTTP vs HTTPS

Easy

What is the difference between HTTP and HTTPS?

HTTP sends everything in plain text, so anyone on the path, such as someone on public WiFi, can read or modify it. HTTPS is HTTP carried inside TLS, which gives encryption (privacy), integrity (tampering is detected) and authentication (the server proves its identity with a certificate signed by a trusted authority). During the TLS handshake, the client and server agree on a cipher, the server presents its certificate, and they establish a shared session key. After that, the data is encrypted with fast symmetric encryption. HTTP uses port 80 and HTTPS uses 443. A certificate only proves the connection is private to that domain. It does not prove the site is trustworthy.

> **Interview tip.** Say TLS, not SSL. SSL is the old name and is no longer used.

57 HTTP methods, status codes

Easy

Explain common HTTP methods and status codes.

Methods say what to do. GET reads data and is safe and idempotent. POST creates a resource or triggers an action and is not idempotent. PUT replaces a resource and is idempotent. PATCH updates part of a resource. DELETE removes one and is idempotent. Idempotent means repeating the call leaves the same end state. Status codes are grouped: 1xx informational, 2xx success (200 OK, 201 Created, 204 No Content), 3xx redirect (301 permanent, 302 temporary, 304 not modified), 4xx client error (400 bad request, 401 not authenticated, 403 forbidden, 404 not found, 429 too many requests), and 5xx server error (500 internal error, 502 bad gateway, 503 unavailable).

> **Interview tip.** 401 means who are you, 403 means I know you but no. Interviewers test this.

58 What happens on typing a URL

Hard

What happens when you type a URL in the browser and press Enter?

The browser parses the URL and checks its caches. Then DNS turns the domain into an IP address. The browser opens a TCP connection with the three-way handshake, usually to port 443. For HTTPS, a TLS handshake follows. The browser sends an HTTP request, for example GET /, with headers and cookies. The request may pass through a CDN, load balancer and reverse proxy to an application server, which may query a database. The server returns a status code and the HTML. The browser parses the HTML into the DOM, fetches CSS, JavaScript and images with more requests, builds the layout and paints the page. Connections are often reused for those extra requests.

> **Interview tip.** Tell it in order, then go deeper on whichever step the interviewer picks.

59 IP, MAC and port

Easy

What is the difference between an IP address, a MAC address and a port?

An IP address identifies a device's place on a network and can change when you move to a different network. Routers use it to forward packets between networks (layer 3). A MAC address is a hardware address stamped on the network card, 48 bits written like 00:1A:2B:3C:4D:5E. Switches use it inside a local network (layer 2). A port is a 16-bit number, 0 to 65535, that identifies one application or service on a device, so one machine can run many network services (layer 4). Think of IP as the postal address, the port as the flat number in the building, and the MAC as the serial number of your network card. Well-known ports: HTTP 80, HTTPS 443, SSH 22, DNS 53.

> **Interview tip.** Remember the pairs: IP with routers, MAC with switches, port with applications.

60 NAT and DHCP

Medium

What are NAT and DHCP?

DHCP automatically gives a device its network settings when it joins a network: an IP address, subnet mask, default gateway and DNS server, usually for a limited time called a lease. Nobody configures anything by hand. NAT (Network Address Translation) lets many devices on a private network share one public IP. Your home router rewrites the source address and port of outgoing packets to its public IP and keeps a table, so replies reach the right device. Private ranges are 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16. NAT helped IPv4 last longer, since its 32-bit space has only about 4.3 billion addresses. IPv6 uses 128-bit addresses, so every device can have a public one.

> **Interview tip.** NAT blocks unsolicited inbound traffic by default, but it is not a firewall by design.

Test yourself

1. Why does a video call use UDP while a bank website uses TCP?
2. Your site loads over HTTP but not HTTPS. Which layers and steps would you check first?

SECTION 07 / 08

OOP and SOLID

OOP questions look easy but interviewers push for real examples. Prepare one small example per concept, ideally in the language you code in. The Java examples here work in most other languages with small changes.

61 Four pillars of OOP

Easy

What are the four pillars of object-oriented programming?

Encapsulation bundles data with the methods that work on it and hides the internals behind a public interface, using private fields and public methods. Abstraction exposes only what is necessary and hides the complexity: you drive a car with a steering wheel without knowing the engine. Inheritance lets a class reuse and extend another class, such as Dog extending Animal, and models an is-a relationship. Polymorphism means one interface with many forms: the same method call behaves differently depending on the object, so speak() barks for a Dog and meows for a Cat. Together they make code modular, reusable and easier to change.

```
class Animal { void speak() { System.out.println("..."); } }
class Dog extends Animal { void speak() { System.out.println("Woof"); } }

public class Main {
    public static void main(String[] args) {
        Animal a = new Dog();
        a.speak();  // prints Woof
    }
}
```

Interview tip. Give one real example per pillar, not just the definition.

62 Encapsulation in practice

Easy

What is encapsulation and why does it matter?

Encapsulation means keeping a class's fields private and letting the outside world touch them only through methods. This lets the class protect its own rules. A BankAccount should not let anyone set the balance to a negative number directly, so a withdraw method checks the amount first. Benefits: validation lives in one place, you can change the internals without breaking users, you can make fields read-only by leaving out setters, and accidental changes cause fewer bugs. Encapsulation hides data and uses access modifiers. Abstraction hides implementation detail and uses interfaces and abstract classes. Interviewers often ask for exactly that difference.

```
class Account {
    private double balance;

    void withdraw(double amount) {
        if (amount <= 0 || amount > balance) {
            throw new IllegalArgumentException("Invalid amount");
        }
        balance -= amount;
    }
}
```

Interview tip. Encapsulation protects data. Abstraction hides complexity. Keep them apart.

63 Inheritance types

Medium

What are the types of inheritance and what is the diamond problem?

Inheritance lets a child class take the fields and methods of a parent. Single: one parent. Multilevel: A to B to C. Hierarchical: many children of one parent. Multiple: a class has more than one parent. C++ and Python support it, but Java does not allow it for classes and offers multiple interfaces instead. Hybrid is a mix. The diamond problem appears when B and C both inherit from A, and D inherits from both B and C, so it is unclear which version of an A method D gets. C++ uses virtual inheritance to resolve it, Python uses method resolution order, and Java avoids the problem. Use inheritance only for true is-a relations.

> **Interview tip.** Know how your language handles multiple inheritance. Java uses interfaces.

64 Static vs dynamic polymorphism

Medium

What is the difference between compile-time and runtime polymorphism?

Compile-time (static) polymorphism means the compiler decides which method runs, based on the method signature. Method overloading and operator overloading are examples. Runtime (dynamic) polymorphism means the choice is made while the program runs, based on the actual type of the object. Method overriding, with a parent reference pointing to a child object, is the example. It works through a virtual method table in C++ and dynamic dispatch in Java. Runtime polymorphism lets you loop over a list of shapes and call area() on each one without knowing the concrete types, so you can add a new shape without touching the loop.

```
Shape s = new Circle();
s.area();  // calls Circle's area(), decided at runtime
```

> **Interview tip.** Overloading is compile time. Overriding is runtime. A very common question.

65 Overloading vs overriding

Easy

What is the difference between method overloading and overriding?

Overloading means the same method name with different parameter lists (number, type or order) in the same class. It is resolved at compile time. A different return type alone is not enough to make an overload. Overriding means a subclass provides its own implementation of a method that already exists in its parent, with the same name and parameters. It is resolved at runtime. In Java, the access level cannot be more restrictive, and you cannot override static, final or private methods. Use the @Override annotation so the compiler catches typos in the method name.

```
class Calc {
    int add(int a, int b) { return a + b; }            // overloading
    double add(double a, double b) { return a + b; }
}
class A { void hi() { System.out.println("A"); } }
class B extends A {
    @Override void hi() { System.out.println("B"); }  // overriding
}
```

> **Interview tip.** Overloading is same class, different parameters. Overriding is child class, same signature.

66 Interface vs abstract class

Medium

What is the difference between an interface and an abstract class?

Both define a contract that other classes complete, and neither can be instantiated directly. An abstract class can have fields, constructors, concrete methods and abstract methods, and a class can extend only one. It suits closely related classes that share state and code, like Vehicle with wheels. An interface describes a capability, like Comparable or Runnable, and a class can implement many of them. Traditionally interfaces had only abstract methods and constants. Since Java 8 they can also have default and static methods, but still no instance state. Rule of thumb: use an interface for can-do, and an abstract class for is-a with shared code.

> **Interview tip.** Mention Java 8 default methods to show you know modern behaviour.

67 Composition vs inheritance

Medium

What is the difference between composition and inheritance? Which do you prefer?

Inheritance models is-a: a Dog is an Animal. Composition models has-a: a Car has an Engine, stored as a field. Inheritance is tightly coupled. A change in the parent can break every child, and deep hierarchies are rigid. Composition is flexible: you can swap parts at runtime, combine behaviour from several objects, and test parts separately. The common guideline is to favour composition over inheritance. Use inheritance only when the child is truly a specialised version of the parent and honours its contract. If you only want to reuse code, compose instead of extending.

```
class Car {
    private final Engine engine;
    Car(Engine engine) { this.engine = engine; }
    void start() { engine.start(); }
}
// pass a PetrolEngine or an ElectricEngine, Car does not change
```

> **Interview tip.** Say: favour composition over inheritance, then give the Car and Engine example.

68 SRP and OCP

Medium

Explain the Single Responsibility and Open/Closed principles.

SOLID is a set of five design principles. S, Single Responsibility: a class should have one reason to change. An Invoice class that calculates totals, prints PDFs and saves to the database has three reasons, so split it. O, Open/Closed: software should be open for extension but closed for modification. Add new behaviour by adding new code, not by editing tested code. Instead of an if-else chain on payment type inside one method, define a PaymentMethod interface and add new classes such as UpiPayment and CardPayment. SRP keeps classes small and focused. OCP lets features grow without breaking what already works.

```
interface PaymentMethod { void pay(double amount); }
class UpiPayment implements PaymentMethod {
    public void pay(double amount) { /* UPI logic */ }
}
class CardPayment implements PaymentMethod {
    public void pay(double amount) { /* card logic */ }
}
```

> **Interview tip.** Use the payment example for OCP. It is short and clear.

69 LSP, ISP and DIP

Hard

Explain the Liskov, Interface Segregation and Dependency Inversion principles.

L, Liskov Substitution: a subclass must work anywhere its parent is expected without breaking behaviour. A Square that extends Rectangle and forces width to equal height breaks code that expects setWidth to change only the width. I, Interface Segregation: do not force classes to implement methods they do not use. Prefer several small interfaces to one fat one, so a Robot does not have to implement eat(). D, Dependency Inversion: high-level code should depend on abstractions, not on concrete classes. OrderService takes a PaymentMethod interface and the concrete one is passed in, which is called dependency injection. It also makes unit testing with fakes easy.

> **Interview tip.** The Square and Rectangle example is the standard answer for Liskov.

70 Constructors and destructors

Easy

What are constructors and destructors?

A constructor is a special method that runs when an object is created. It has the class name, no return type, and sets up the object's initial state. Types: default (no arguments), parameterized, and in C++ a copy constructor that builds an object from another. If you define none, the compiler provides a default one. Constructors can be overloaded, and in inheritance the parent constructor runs first (super() in Java). A destructor, in C++, runs when an object is destroyed and releases resources such as memory or files. Java has no destructor. Garbage collection reclaims memory, and resources like files are closed with try-with-resources. The finalize() method is deprecated.

> **Interview tip.** Know the difference between C++ destructors and Java garbage collection.

Test yourself

1. Design a Notification system that can send email, SMS and push without changing existing code when a new channel is added. Which SOLID principle guides you?
2. When is inheritance a bad choice even though two classes seem related?

SECTION 08 / 08

Rapid fire

A mixed round across OS, DBMS, networks and OOP, the way interviewers jump between topics. Each answer is short enough to say aloud in under a minute. Practise saying them out loud, not just reading them.

71 Zombie and orphan process

Medium

What are zombie and orphan processes?

A zombie process has finished running, but its entry stays in the process table because the parent has not yet read its exit status with wait(). It uses no CPU and almost no memory, but too many of them can exhaust process IDs. The fix is for the parent to call wait(), or for the parent to exit so the init process adopts and cleans them up. An orphan process is one that is still running after its parent has exited. It is not harmful: init or systemd adopts it and cleans up when it ends. A zombie is dead but not cleaned up. An orphan is alive without a parent. On Linux, ps shows zombies with state Z.

> **Interview tip.** Zombie means dead but not reaped. Orphan means alive without a parent.

72 Monolithic vs microkernel

Medium

What is the difference between a monolithic kernel and a microkernel?

A monolithic kernel runs all OS services, such as file systems, drivers, memory management and networking, in kernel space as one big program. It is fast because services call each other directly, but a bug in one driver can crash the whole system. Linux is monolithic, though it can load modules. A microkernel keeps only the bare minimum in kernel space, such as scheduling, basic memory handling and IPC, and runs the rest as user-space servers. It is more stable and secure, since a failing driver does not take down the kernel, but it is slower because components talk by message passing. Minix and QNX are microkernels. Windows and macOS use hybrid designs.

> **Interview tip.** Trade-off: monolithic is faster, microkernel is safer.

73 DELETE, TRUNCATE, DROP

Easy

What is the difference between DELETE, TRUNCATE and DROP?

DELETE removes rows. It can use a WHERE clause, it is a DML command, it fires row triggers, and it can be rolled back. It logs each row, so it is slower on big tables. TRUNCATE removes all rows quickly. It is a DDL command, has no WHERE clause, usually resets auto-increment counters, and in many databases cannot be rolled back, though PostgreSQL allows it inside a transaction. DROP removes the entire table, including its structure, indexes and constraints, not just the data. Use DELETE for some rows, TRUNCATE to empty a table, and DROP when the table should not exist anymore.

```
DELETE FROM users WHERE id = 5;  -- some rows
TRUNCATE TABLE users;            -- all rows, table stays
DROP TABLE users;                -- table is gone
```

> **Interview tip.** Order of power: DELETE removes rows, TRUNCATE empties the table, DROP removes the table.

74 WHERE vs HAVING

Easy

What is the difference between WHERE and HAVING?

WHERE filters individual rows before grouping. HAVING filters groups after GROUP BY, so it can use aggregate functions like COUNT, SUM and AVG. WHERE cannot use aggregates. Filtering early with WHERE is cheaper, because fewer rows get grouped. The logical order of execution is FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY, which also explains why you cannot use a SELECT alias inside WHERE in standard SQL. Use WHERE for conditions on single rows and HAVING only for conditions on aggregated results. The example counts active users per city and keeps only cities with more than 100.

```
SELECT city, COUNT(*) AS n
FROM users
WHERE active = 1
GROUP BY city
HAVING COUNT(*) > 100;
```

> **Interview tip.** Learn the execution order. It answers many follow-up questions at once.

75 SQL vs NoSQL

Medium

What is the difference between SQL and NoSQL databases?

SQL databases such as MySQL and PostgreSQL store data in tables with a fixed schema, support joins and strong ACID transactions, and traditionally scale up with a bigger machine, though replication and sharding exist. NoSQL is a family: document (MongoDB), key-value (Redis), wide-column (Cassandra) and graph (Neo4j). Schemas are flexible, they often scale out across many machines more easily, and many trade strict consistency for availability and speed, a model called BASE (basically available, soft state, eventually consistent). Some NoSQL databases do offer transactions. Choose SQL for structured data with relationships, such as payments. Choose NoSQL for flexible data, huge scale or simple key-based access.

> **Interview tip.** Never say NoSQL is always faster. It depends on the access pattern.

76 Hub, switch and router

Easy

What is the difference between a hub, a switch and a router?

A hub is a simple device that repeats incoming data to every port. This causes collisions and gives no privacy. It works at layer 1 and is now obsolete. A switch works at layer 2. It learns which MAC address sits on which port and forwards frames only to the right port, so devices inside one network talk efficiently. A router works at layer 3. It connects different networks, reads IP addresses and picks the best path using a routing table. Your home router connects your local network to the internet. In short, a switch connects devices inside a network, and a router connects networks to each other.

> **Interview tip.** Switch uses MAC addresses, router uses IP addresses.

77 GET vs POST

Easy

What is the difference between GET and POST?

GET requests data. Its parameters travel in the URL, so they appear in browser history and server logs and are limited in length. Never send passwords with GET. GET is safe and idempotent, and responses can be cached and bookmarked. POST sends data in the request body and is used to create resources or submit forms. It is not idempotent, so refreshing may submit twice, and responses are generally not cached. POST is not more secure by itself: without HTTPS, both can be read on the wire. The real difference is meaning. GET reads, POST changes something.

> **Interview tip.** POST is not automatically secure. HTTPS is what protects the data.

78 Cookies vs sessions

Medium

What is the difference between cookies, sessions and tokens?

HTTP is stateless, so the server needs a way to remember you between requests. A cookie is a small piece of data stored in the browser and sent with every request to that site. A session stores user data on the server, and the browser holds only a session ID, usually inside a cookie. So sessions keep sensitive data off the client, while cookies are limited to about 4 KB. Token-based authentication, such as JWT, puts signed claims in a token the client sends, so the server needs no session store, but revoking a token early is harder. Useful cookie flags: HttpOnly, Secure and SameSite.

> **Interview tip.** Mention the cookie flags. It shows you know basic web security.

79 Garbage collection

Medium

What is garbage collection and how does it work?

Garbage collection automatically frees the memory of objects the program can no longer reach. Java, Python, Go and JavaScript use it. A common approach is mark and sweep. Starting from roots such as stack variables and static fields, the collector marks everything reachable, then reclaims the rest. Generational collectors assume most objects die young, so they clean the young area often and cheaply. Python mainly uses reference counting plus a collector for cycles. The benefit is no manual free, with fewer leaks and dangling pointers. The cost is pauses and less control. Leaks are still possible if you keep references you no longer need, like objects stuck in a static list.

> **Interview tip.** A garbage collector frees unreachable objects, not unused ones. Leaks still happen.

80 Shallow vs deep copy

Medium

What is the difference between a shallow copy and a deep copy?

A shallow copy duplicates the object but copies only the references of nested objects, so the original and the copy share the inner objects. Changing nested data through one shows up in the other. A deep copy recursively duplicates everything, so the two are fully independent. Deep copies cost more time and memory, and very large or circular structures need care, though libraries usually handle cycles. Choose shallow when the nested data is immutable or sharing is intended. Choose deep when you need a fully independent snapshot, such as keeping an undo state of a complex document.

```
import copy
a = [[1, 2], [3, 4]]
b = copy.copy(a)       # shallow
c = copy.deepcopy(a)   # deep
a[0][0] = 99
print(b[0][0])  # 99, shared inner list
print(c[0][0])  # 1, independent
```

Interview tip. Test it with a nested list. Mutate the inner list and see who changes.

Test yourself

1. Explain in under a minute why a switch cannot connect your home network to the internet by itself.
2. A query uses WHERE on an aggregate and fails. Explain why and rewrite it.