Interview Q&A for Freshers

Databases (SQL & NoSQL), OOP, Operating Systems, Computer Networks

Databases (SQL and NoSQL)

Key comparisons, design trade-offs, and practical examples.

What's the difference between SQL and NoSQL? When to use each?

Answer: SQL databases are relational, schema-on-write, and ACID-compliant—best for transactional apps needing strict consistency and complex joins (e.g., banking). NoSQL databases are non-relational, flexible schema, distributed, and often favor horizontal scalability and availability—best for large-scale, evolving data and high write throughput.

- **SQL examples**: PostgreSQL, MySQL, SQL Server.
- **NoSQL examples**: MongoDB (document), Redis (key-value), Cassandra (wide-column), Neo4j (graph).

Normalization (1NF, 2NF, 3NF, BCNF)—what and why?

Answer: Normalization reduces redundancy and update anomalies.

- **1NF**: Atomic columns; no repeating groups.
- **2NF**: 1NF + no partial dependency on a composite key.
- **3NF**: 2NF + no transitive dependency on the key.
- **BCNF**: For every dependency X→Y, X is a super key.

Normalization vs denormalization—trade-offs?

Answer: Normalization improves integrity and reduces duplication but may increase join complexity (slower reads). Denormalization duplicates data to speed reads at the cost of slower writes and potential inconsistencies.

ACID properties and why they matter

Answer: Atomicity (all-or-nothing), **Consistency** (valid state transitions), **Isolation** (concurrent transactions don't interfere), **Durability** (committed data persists even after crashes). They ensure reliable transactions.

Indexes: types, pros/cons, and when to use

Answer: Common types: B-Tree (range queries), Hash (equality lookups), Full-text, GiST/GIN (Postgres). Indexes speed reads and filters but slow writes and use space.

```
CREATE INDEX idx_users_email ON users(email);
EXPLAIN ANALYZE SELECT * FROM users WHERE email = 'a@b.com';
```

Primary key vs Unique key vs Foreign key

Answer: Primary key uniquely identifies rows (not null). **Unique key** enforces uniqueness (nullable in many DBs). **Foreign key** enforces referential integrity to another table, optionally with `ON DELETE/UPDATE` actions.

JOIN types and use-cases

Answer: INNER (matches), LEFT/RIGHT (preserve one side), FULL (preserve both), CROSS (cartesian), SELF (same table).

```
SELECT u.name, o.total
FROM users u
LEFT JOIN orders o ON o.user_id = u.id;
```

WHERE vs HAVING

Answer: WHERE filters rows before aggregation; **HAVING** filters after aggregation.

```
SELECT dept, COUNT(*) c
FROM employees
WHERE active = true
GROUP BY dept
HAVING COUNT(*) > 5;
```

Clustered vs non-clustered index

Answer: Clustered defines physical row order (often one per table, e.g., InnoDB PK). **Non-clustered** is a separate structure with pointers (many allowed).

Transactions and isolation levels—anomalies

Answer: Isolation levels: Read Uncommitted (dirty reads), Read Committed (no dirty reads, possible non-repeatable), Repeatable Read (no non-repeatable, phantom possible), Serializable (no anomalies, lowest concurrency). Choose based on correctness vs throughput.

SQL injection and prevention

Answer: Avoid string concatenation; use parameterized queries or prepared statements.

```
-- BAD: "SELECT * FROM users WHERE email='" + input + "'"
-- GOOD:
PREPARE stmt FROM 'SELECT * FROM users WHERE email = ?';
EXECUTE stmt USING @email;
```

CAP theorem and distributed databases

Answer: Under a network partition, a system must choose **Consistency** or **Availability**, not both. CP systems (e.g., majority writes) favor consistency; AP systems (Dynamo-style) favor availability.

Sharding vs replication

Answer: Sharding splits data across nodes (scales writes/size). **Replication** copies data for reliability and read scaling. Often combined.

Partitioning strategies (RDBMS)

Answer: Range, list, hash, composite. Benefits: pruning, parallelism, manageability.

```
-- Postgres declarative partitioning (range)
CREATE TABLE events (id bigserial, ts date) PARTITION BY RANGE (ts);
```

OLTP vs OLAP

Answer: OLTP: many small transactions, normalized. **OLAP**: analytics on large data; denormalized star/snowflake schemas; columnar stores common.

Query planning and performance

Answer: Use `EXPLAIN`, ensure selective indexes, avoid N+1 queries, watch for table scans, keep stats updated, and rewrite queries to leverage indexes.

Views vs materialized views

Answer: Views are virtual (computed on read). **Materialized views** store results; faster reads, need refresh policies.

Stored procedures and triggers—pros/cons

Answer: Centralize logic near data, enforce rules; can complicate versioning, portability, and debugging—use judiciously.

NoSQL data models (with examples)

- **Document**: Flexible JSON per entity; embed vs reference trade-offs.
- **Key-value**: Ultra-fast lookups; limited querying.
- **Wide-column**: Time-series/huge sparse datasets.
- **Graph**: Relationship-heavy queries (shortest path, recommendations).

Object-Oriented Programming (OOP)

Four OOP pillars with short examples

Answer: Encapsulation (hide state via accessors), **Abstraction** (expose essential behavior via interfaces), **Inheritance** (reuse/extend base), **Polymorphism** (same interface, different implementations).

Overloading vs overriding

Answer: Overloading: same name, different parameters (compile-time polymorphism). **Overriding**: subclass changes base method behavior (runtime via dynamic dispatch).

Interface vs abstract class—how to choose?

Answer: Use **interfaces** for capability contracts and multiple implementations; use an **abstract class** for shared base state/logic and constrained extension.

Composition over inheritance—why?

Answer: Composition (*has-a*) avoids tight coupling and fragile base class issues; lets you swap behaviors at runtime (Strategy pattern).

SOLID principles (quick)

- **Single Responsibility**: one reason to change.
- **Open/Closed**: open to extension, closed to modification.
- **Liskov Substitution**: subtypes must be usable as base types.
- **Interface Segregation**: small, specific interfaces.
- **Dependency Inversion**: depend on abstractions; inject dependencies.

Shallow vs deep copy

Answer: Shallow copies references; deep copy recursively duplicates nested objects to avoid shared mutable state.

Immutability—benefits

Answer: Thread-safety by design, easier reasoning, safer sharing. Use builders or copy-on-write patterns.

Exceptions—checked vs unchecked

Answer: Checked must be declared/handled (recoverable conditions). **Unchecked** represent programming errors (null pointer, illegal state).

Common patterns and when to use

- **Singleton**: shared instance (watch testability).
- **Factory**: decouple creation logic.
- **Strategy**: swap algorithms at runtime.

- **Observer**: events/callbacks.
- **Adapter**: interface compatibility.

Operating Systems

Process vs thread; PCB/TCB

Answer: A **process** has its own address space; PCB stores its state. A **thread** shares the process memory; TCB stores thread context. Threads improve concurrency within a process.

Context switching—costs

Answer: Save/restore registers and memory maps; cache/TLB effects cause overhead. Excessive switching reduces performance.

CPU-bound vs I/O-bound tasks

Answer: CPU-bound saturate CPU—schedule to maximize CPU utilization. I/O-bound often waits—use async I/O and more concurrency.

Scheduling algorithms and trade-offs

Answer: FCFS (simple, convoy effect), SJF/SRTF (great avg wait; needs prediction), Priority (starvation risk), Round Robin (fairness, time slice choice), Multilevel queues (class-based policies).

Deadlock—conditions and handling

Answer: Conditions: mutual exclusion, hold-and-wait, no preemption, circular wait. Prevent with ordering, timeouts; detect via wait-for graphs; recover by killing/rollback.

Mutex vs semaphore

Answer: Mutex is owned by the locker; mutual exclusion. **Semaphore** is counting-based signaling (e.g., resource pools).

```
// Pseudocode
wait(mutex);  // lock
// critical section
signal(mutex); // unlock
```

Paging, segmentation, and virtual memory

Answer: Paging uses fixed-size pages/frames (no external fragmentation). Segmentation is variable-sized logical units. Virtual memory maps virtual pages to physical frames with on-demand paging.

Page faults and replacement algorithms

Answer: On fault, OS loads from disk; select victim via FIFO, LRU, Optimal (theoretical), or Clock (approx LRU). Thrashing occurs when the working set exceeds RAM.

User vs kernel mode; system calls

Answer: Kernel mode has full privileges; user mode is restricted. System calls are controlled entry points (open, read, write, fork, exec, wait).

File systems—inode and journaling

Answer: Inodes store metadata and block pointers. Journaling logs intent to maintain consistency after crashes (e.g., ext4, NTFS).

Interrupts and traps

Answer: Interrupts are async hardware signals. **Traps** are synchronous exceptions or system call entries.

Computer Networks

OSI vs TCP/IP—mapping

Answer: OSI (7 layers) vs TCP/IP (4–5 layers). Mapping: Application ↔ OSI 5–7; Transport ↔ OSI 4; Internet ↔ OSI 3; Link/Physical ↔ OSI 1–2.

TCP vs UDP—details

Answer: TCP: connection-oriented, reliable, ordered; flow control (`rwnd`), congestion control (AIMD), retransmissions (RTO). **UDP**: connectionless, best-effort datagrams; used for DNS, VoIP, streaming.

TCP three-way handshake and teardown

Answer: Handshake: SYN → SYN-ACK → ACK. Teardown: FIN/ACK exchange; TIME_WAIT avoids interference from stray segments.

HTTP/1.1 vs HTTP/2 vs HTTP/3

Answer: 1.1: keep-alive, HoL blocking at app level. 2: multiplexing over one TCP, HPACK. 3: QUIC over UDP, 0-RTT, avoids TCP HoL blocking.

REST vs RPC (gRPC)

Answer: REST uses resources and HTTP verbs; loosely coupled. **RPC/gRPC** defines service contracts and binary protocols (Protobuf) for high performance and streaming.

DNS resolution flow

Answer: Stub resolver → recursive resolver → root → TLD → authoritative; caching reduces latency. Records: A/AAAA, CNAME, MX, TXT, NS.

IP addressing and CIDR—quick example

Answer: `192.168.1.0/24` has 256 addresses (254 usable). Subnetting /26 yields 4 subnets of 64 addresses each.

NAT and port translation

Answer: Maps many private IPs to a public IP via port mapping (PAT). Conserves IPv4 and adds a basic isolation layer.

Routing basics (intra vs inter-domain)

Answer: Intra: OSPF/IS-IS (link-state). Inter: BGP (path-vector, policy-driven). Routers forward using longest prefix match.

TLS handshake (simplified)

Answer: ClientHello/ServerHello, certificate exchange/verification, key exchange (ECDHE), Finished messages; then encrypted application data.

Latency, bandwidth, jitter—impacts

Answer: Latency = delay; **bandwidth** = capacity; **jitter** = delay variation. Real-time apps are latency/jitter-sensitive; bulk transfer is bandwidth-sensitive.

Practice Snippets

SQL: 2nd highest salary per department

```
SELECT dept, MAX(salary) AS second_highest
FROM (
  SELECT dept, salary,
         DENSE_RANK() OVER (PARTITION BY dept ORDER BY salary DESC) rnk
  FROM employees
) t
WHERE rnk = 2
GROUP BY dept;
```

MongoDB: users created in last 7 days with verified email

```
db.users.find({
  emailVerified: true,
  createdAt: { $gte: new Date(Date.now() - 7*24*60*60*1000) }
});
```

Prevent race condition on balance update (SQL)

```
BEGIN;
SELECT balance FROM accounts WHERE id = 1 FOR UPDATE;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
COMMIT;
```

Semaphore for a connection pool (pseudocode)

```
wait(sem);   // acquire slot
conn = getConn();
// use conn
releaseConn(conn);
signal(sem); // release slot
```

Tip: Bring up trade-offs (consistency vs availability, composition vs inheritance, isolation levels vs anomalies) to stand out in interviews.