20 Common SQL Interview Questions & Answers

Suitable for ~5 years experienced candidates

---

1. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN and FULL OUTER JOIN?

INNER JOIN returns rows with matching keys in both tables. LEFT JOIN returns all rows from the left table and matched rows from the right; unmatched right rows are NULL. RIGHT JOIN returns all rows from the right table and matched rows from the left; unmatched left rows are NULL. FULL OUTER JOIN returns all rows when there is a match in one of the tables; unmatched columns are NULL.

2. How would you find duplicate rows in a table?

Use GROUP BY on the columns that define duplicates and HAVING COUNT(*) > 1. Example: SELECT col1, col2, COUNT(*) FROM table GROUP BY col1, col2 HAVING COUNT(*) > 1.

3. Explain the difference between WHERE and HAVING.

WHERE filters rows before aggregation; HAVING filters groups after aggregation. Use WHERE for row-level conditions and HAVING for conditions on aggregates (SUM, COUNT, AVG, etc.).

4. How do you optimize a slow query?

Common steps: check execution plan, add appropriate indexes, avoid SELECT *, limit returned rows, rewrite subqueries as joins (or vice versa), denormalize if necessary, use proper statistics and query hints only when justified, and batch large DML operations.

5. What are indexes? Types of indexes and when to use them.

Indexes are data structures that speed up data retrieval. Types: B-tree (general purpose), hash (exact match lookups), bitmap (low-cardinality columns), composite (multi-column), unique (enforce uniqueness). Use indexes on columns used in WHERE, JOIN, ORDER BY; avoid on frequently-updated columns without benefit.

6. Explain ACID properties in databases.

ACID: Atomicity (transactions complete fully or not at all), Consistency (database moves from one valid state to another), Isolation (concurrent transactions don't interfere; isolation levels control visibility), Durability (committed transactions persist despite failures).

7. What is normalization? Explain up to 3NF with an example.

Normalization organizes data to reduce redundancy. 1NF: atomic values. 2NF: 1NF + no partial dependence on part of a composite key. 3NF: 2NF + no transitive dependencies. Example: split customer orders into Customers and Orders tables so customer details aren’t repeated in each order.

8. How do you handle many-to-many relationships in SQL?

Use a junction (bridge) table containing foreign keys to both tables. Example: Students, Courses, and StudentCourses(student_id, course_id). Optionally include additional attributes (grade, enrolled_date).

9. What is a correlated subquery? Provide an example.

A correlated subquery references columns from the outer query and is evaluated per outer row. Example: SELECT e.* FROM employees e WHERE salary > (SELECT AVG(salary) FROM employees WHERE department_id = e.department_id).

10. Difference between UNION and UNION ALL.

UNION removes duplicates (performs distinct) and can be slower; UNION ALL returns all rows including duplicates and is faster. Use UNION ALL if duplicates are acceptable or impossible by design.

11. How does a transaction isolation level affect concurrency?

Isolation levels (Read Uncommitted, Read Committed, Repeatable Read, Serializable) trade off between consistency and concurrency. Lower levels allow higher concurrency but can cause phenomena like dirty reads, non-repeatable reads, or phantom reads.

12. What is a window function? Give a use-case.

Window functions perform calculations across a set of rows related to the current row without collapsing rows. Examples: ROW_NUMBER(), RANK(), SUM() OVER(PARTITION BY ... ORDER BY ...). Use-case: compute running totals or rank rows within partitions.

13. How to perform pagination in SQL efficiently?

Use keyset pagination (seek method) for large datasets: SELECT ... WHERE (key > last_key) ORDER BY key LIMIT n. OFFSET ... LIMIT is simpler but can be inefficient for large offsets because the DB must skip rows.

14. What is an execution plan and how do you read it?

An execution plan shows how the database will execute a query: index scans vs full table scans, join algorithms (nested loop, hash, merge), estimated costs and row counts. Look for expensive operations, large row estimates, and missing index usage to optimize queries.

15. Explain foreign keys and cascades.

Foreign keys enforce referential integrity between tables. Cascades define behavior on parent changes: ON DELETE CASCADE removes child rows when parent deleted; ON UPDATE CASCADE updates child FK values when parent PK changes. Use cascades carefully.

16. How do you implement soft delete? Pros and cons.

Add a boolean (is_deleted) or deleted_at timestamp column and filter it out in queries. Pros: recoverability, audit history. Cons: requires every query to filter, can bloat indexes and tables, needs maintenance (archiving).

17. How to store hierarchical data in SQL? Discuss methods.

Options: adjacency list (parent_id column), nested sets (left/right values), materialized path (path string), and recursive CTEs for querying. Choose based on read/write patterns: adjacency + recursive CTEs is flexible; nested sets are good for fast read-heavy subtree queries.

18. What are stored procedures vs functions? When to use which?

Stored procedures can perform actions and may not return a value (can return result sets); functions return a value and can be used in expressions. Procedures are better for complex operations and control flow; functions are useful for reusable computations inside queries. Note DB-specific differences (permissions, side-effects).

19. How do you migrate a large table with minimal downtime?

Use techniques like online schema changes (pt-online-schema-change, gh-ost), create new table with desired schema, copy data in batches while keeping sync with triggers or change data capture, swap tables/rename, and backfill remaining rows. Test and ensure transactional integrity and backups.

20. Explain common types of JOIN algorithms (nested loop, hash, merge).

Nested loop: iterates outer rows and probes inner—good for small outer sets or indexed inner. Hash join: builds hash table on smaller input and probes with the other—good for large, unsorted data. Merge join: requires both inputs sorted on join key; streams through both—efficient for pre-sorted or large datasets.