Top 100 SQL Interview Questions & Answers [2025 Updated]

Vista Academy — Freshers & Analytics Roles • SQL Query Interview Questions • Database Testing for Freshers

Generated on August 21, 2025

Table of Contents

Part 1 — Basics for Freshers (Q1–20)

1. What is SQL, and why is it important in Data Analytics?

SQL manages relational databases and is essential for extracting insights, cleaning data, and producing reports for data■driven decisions.

2. Difference between SELECT and SELECT DISTINCT?

SELECT returns all rows (including duplicates); SELECT DISTINCT returns only unique rows.

```
SELECT DISTINCT department FROM employees;
```

3. How do you retrieve all columns from a table?

Use the asterisk (*) with SELECT.

```
SELECT * FROM employees;
```

4. What is the purpose of the WHERE clause?

Filters rows before they are returned.

```
SELECT name FROM employees WHERE salary > 50000;
```

5. What are the main types of SQL JOINs?

INNER, LEFT (OUTER), RIGHT (OUTER), and FULL (OUTER). Use based on matching needs.

```
SELECT e.name, d.department_name
FROM employees e
INNER JOIN departments d ON e.dept_id = d.dept_id;
```

6. What is a Primary Key?

A column (or set) that uniquely identifies each row and cannot be NULL.

7. What is a Foreign Key?

A column that references a primary key in another table to enforce referential integrity.

8. How do you sort query results?

Use ORDER BY with ASC/DESC.

```
SELECT name, salary FROM employees ORDER BY salary DESC;
```

9. What does GROUP BY do?

Groups rows for aggregates (e.g., COUNT, SUM).

10. What is the HAVING clause?

Filters groups after aggregation (WHERE filters rows before grouping).

11. Difference between WHERE and HAVING?

WHERE filters rows pre-aggregation; HAVING filters groups post-aggregation.

12. How do you limit rows?

Use LIMIT (MySQL/Postgres) or TOP (SQL Server).

```
-- MySQL/Postgres
SELECT name FROM employees LIMIT 10;
-- SQL Server
SELECT TOP(10) name FROM employees;
```

13. What are aggregate functions?

COUNT, SUM, AVG, MIN, MAX.

14. How do you handle NULL values?

Use IS NULL / IS NOT NULL, or COALESCE / IFNULL to substitute defaults.

15. What does COALESCE do?

Returns the first non■NULL value from its arguments.

16. Difference between DELETE and TRUNCATE?

DELETE removes selected rows and can be rolled back; TRUNCATE removes all rows, is faster, and less granular.

17. How do you update data?

Use UPDATE with SET to modify columns based on a condition.

```
UPDATE employees SET salary = salary * 1.10 WHERE dept_id = 5;
```

18. What is a Subquery?

A query nested inside another to supply values or filter results.

```
SELECT name FROM employees
WHERE dept_id = (SELECT dept_id FROM departments WHERE dept_name='Sales');
```

19. Count paying customers by city

Aggregate with GROUP BY and filter with WHERE.

```
SELECT city, COUNT(*) AS customer_count
FROM customers
WHERE is_paying = 1
GROUP BY city;
```

20. Correlated vs non■correlated subqueries?

Non■correlated executes independently; correlated runs per row of the outer query and references it.

```
-- Correlated example
SELECT name FROM employees e
WHERE salary > (SELECT AVG(salary) FROM employees WHERE dept_id = e.dept_id);
```

Part 2 — Intermediate (Q21–40)

21. How do you create a table?

Use CREATE TABLE with column definitions and constraints.

```
CREATE TABLE customers (
  customer_id INT PRIMARY KEY,
  name VARCHAR(50),
  email VARCHAR(100)
);
```

22. What is the CASE statement used for?

Adds conditional logic within SELECT expressions.

```
SELECT name,
       CASE WHEN salary > 50000 THEN 'High' ELSE 'Low' END AS salary_category
FROM employees;
```

23. How do you find duplicate rows?

Use GROUP BY with HAVING COUNT(*) > 1.

```
SELECT email, COUNT(*) AS cnt
FROM customers
GROUP BY email
HAVING COUNT(*) > 1;
```

24. What is the UNION operator?

Combines results of two SELECTs and removes duplicates (schemas must align).

25. Difference between UNION and UNION ALL?

UNION removes duplicates; UNION ALL includes all rows (often faster).

26. How to add a new column?

ALTER TABLE ... ADD column.

```
ALTER TABLE employees ADD hire_date DATE;
```

27. What is the IN operator?

Checks if a value matches any in a list or subquery.

28. What is the BETWEEN operator?

Selects values within an inclusive range.

29. How do you rename a table?

ALTER TABLE old_name RENAME TO new_name; (syntax differs by DB).

30. What is a View?

A virtual table based on a SELECT; simplifies access/security for complex queries.

31. How do you drop a table?

DROP TABLE table_name; permanently deletes the table and data.

32. What is the LIKE operator?

Pattern matching with % (any chars) and _ (single char).

33. How do you concatenate strings?

Use CONCAT() or || depending on the database (DB■specific).

34. Purpose of an Index?

Speeds up reads at the cost of storage and slower writes.

35. Find the second highest salary

Use a subquery or ranking.

```
SELECT MAX(salary)
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

36. What is Normalization?

Organizing data to reduce redundancy (1NF, 2NF, 3NF…).

37. What is Denormalization?

Reintroducing redundancy for faster reads (typical in analytics).

38. Total sales by month?

Group by month and sum amounts.

```
SELECT DATE_TRUNC('month', order_date) AS month, SUM(amount) AS total_sales
FROM orders
GROUP BY 1;
```

39. What is a Self Join?

Joining a table to itself (e.g., employees with managers).

```
SELECT e1.name, e2.name AS manager
FROM employees e1 LEFT JOIN employees e2 ON e1.manager_id = e2.emp_id;
```

40. Case■insensitive comparison?

Normalize case with LOWER/UPPER while comparing.

```
SELECT name FROM employees WHERE LOWER(name) = 'john';
```

Part 3 — Advanced & Analytics (Q41–60)

41. How does COUNT handle NULLs?

COUNT(column) ignores NULLs; COUNT(*) counts all rows.

42. How to pivot data?

Use conditional aggregation or DB■specific PIVOT.

```
SELECT department,
  SUM(CASE WHEN year=2023 THEN sales ELSE 0 END) AS sales_2023,
  SUM(CASE WHEN year=2024 THEN sales ELSE 0 END) AS sales_2024
FROM sales_data
GROUP BY department;
```

43. INNER JOIN vs LEFT JOIN?

INNER returns matches only; LEFT returns all left rows plus matches from right (NULL when no match).

44. Running total?

Use SUM() OVER with an ORDER BY.

```
SELECT order_id, amount,
       SUM(amount) OVER (ORDER BY order_date) AS running_total
FROM orders;
```

45. What is a Window Function?

Calculations across related rows without collapsing (ROW_NUMBER, RANK, SUM OVER…).

46. Remove duplicate rows?

Identify with ROW_NUMBER() in a CTE and delete where rn > 1.

```
WITH d AS (
  SELECT *, ROW_NUMBER() OVER (PARTITION BY email ORDER BY customer_id) rn
  FROM customers
)
DELETE FROM d WHERE rn > 1;
```

47. What is EXISTS?

Checks if a subquery returns rows; efficient for semi/anti joins.

48. Top 5 employees by salary?

ORDER BY salary DESC with LIMIT/TOP.

```
SELECT name, salary FROM employees ORDER BY salary DESC LIMIT 5;
```

49. RANK vs DENSE_RANK?

RANK leaves gaps after ties; DENSE_RANK doesn’t.

50. % contribution by department?

Divide dept sum by overall sum using a window.

```
SELECT department, SUM(sales) AS dept_sales,
       (SUM(sales) / SUM(SUM(sales)) OVER()) * 100 AS pct
FROM sales_data
GROUP BY department;
```

51. What is a Database?

An organized collection of data, managed by a DBMS.

52. What is a Table?

Rows and columns representing an entity’s data.

53. What is a Column?

An attribute/field of a table (e.g., salary).

54. What is a Row?

A single record (tuple) in a table.

55. What is a Query?

A statement to retrieve/modify data written in SQL.

56. What is a DBMS?

Software that stores, manages, and secures databases.

57. What is a Relational Database?

Data organized in related tables via keys.

58. Primary Key definition?

Unique, non■NULL identifier for table rows.

59. Foreign Key definition?

Column referencing another table’s PK to enforce relationships.

60. What is Data Integrity?

Accuracy/consistency of data enforced by constraints and rules.

Part 4 — Constraints & Optimization (Q61–80)

61. What are SQL Constraints?

Rules to maintain integrity: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT.

62. UNIQUE vs PRIMARY KEY?

PK = unique + not null (one per table). UNIQUE allows multiple NULLs in some DBs (not Oracle).

63. What is a Composite Key?

Multiple columns forming a unique identifier.

64. What is an Index?

Speeds up SELECTs; may slow writes and use storage.

65. Types of Indexes?

Clustered (defines physical order) and Non■clustered (separate structure).

66. What is a Transaction?

A unit of work following ACID properties.

67. ACID properties?

Atomicity, Consistency, Isolation, Durability.

68. COMMIT vs ROLLBACK?

COMMIT saves changes; ROLLBACK undoes uncommitted changes.

69. What is SAVEPOINT?

A marker to roll back part of a transaction.

70. What is a Deadlock?

Two transactions waiting on each other’s locks; both block until resolved.

71. What is a Self Join?

Join a table to itself to relate rows.

72. CROSS JOIN vs INNER JOIN?

CROSS = Cartesian product; INNER = matches by condition.

73. What is Referential Integrity?

Ensures FKs point to existing PKs, preserving relationships.

74. DELETE vs TRUNCATE vs DROP?

DELETE removes selected rows; TRUNCATE removes all rows; DROP removes the table definition.

75. What is Normalization?

Structuring data to reduce redundancy (1NF→5NF).

76. What is Denormalization?

Introducing redundancy for performance (common in OLAP).

77. How to optimize SQL queries?

Right indexes, avoid SELECT *, proper joins/filters, check execution plans.

78. What is a Stored Procedure?

Precompiled SQL logic stored in DB; reusable and secure.

79. What is a Trigger?

Automatic action on INSERT/UPDATE/DELETE events.

80. What is a Cursor?

Row■by■row iterator for procedural operations (use sparingly).

Part 5 — CTEs & Real■World Scenarios (Q81–100)

81. What is a CTE and why use it?

WITH clause defines a named temporary result set for readability and recursion.

```
WITH dept_avg AS (
  SELECT dept_id, AVG(salary) AS avg_sal FROM employees GROUP BY dept_id
)
SELECT e.name, e.salary, d.avg_sal
FROM employees e JOIN dept_avg d ON e.dept_id = d.dept_id;
```

82. What is a recursive CTE?

CTE that references itself to traverse hierarchies (org charts/categories).

```
WITH RECURSIVE chain AS (
  SELECT emp_id, manager_id, 0 AS lvl FROM employees WHERE manager_id IS NULL
  UNION ALL
  SELECT e.emp_id, e.manager_id, c.lvl+1
  FROM employees e JOIN chain c ON e.manager_id = c.emp_id
)
SELECT * FROM chain;
```

83. CTE vs Subquery?

CTEs are named, reusable within the statement, can be recursive; subqueries are inline and not reusable.

84. LAG/LEAD to compare rows

Access previous/next row values in window frames without self■joins.

```
SELECT order_id, order_date, amount,
       LAG(amount) OVER (ORDER BY order_date) AS prev_amt,
       LEAD(amount) OVER (ORDER BY order_date) AS next_amt
FROM orders;
```

85. Nth highest salary pattern

Use DENSE_RANK over salaries and filter by the desired rank.

```
WITH r AS (
  SELECT name, salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rnk
  FROM employees
)
SELECT name, salary FROM r WHERE rnk = 3;
```

86. Customers with no orders (anti■join)

Use LEFT JOIN ... IS NULL or NOT EXISTS.

```
SELECT c.customer_id, c.name
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.customer_id
WHERE o.customer_id IS NULL;
```

87. Window frames (RANGE/ROWS)

Control which rows are included in the calculation (e.g., rolling windows).

```
SELECT dt, amount,
  SUM(amount) OVER (ORDER BY dt ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling_7
FROM daily_sales;
```

88. Star vs Snowflake schema

Star = denormalized dimensions (simple/fast); Snowflake = normalized dimensions (less redundancy, more joins).

89. OLTP vs OLAP

OLTP = transactional, many small writes; OLAP = analytical, large read queries over history.

90. Materialized view vs View

View is virtual; materialized view stores results for faster reads and needs refresh.

91. Monthly Active Users (MAU) query

Count distinct users grouped by month.

```
SELECT DATE_TRUNC('month', activity_at) AS mth,
       COUNT(DISTINCT user_id) AS mau
FROM user_activity
GROUP BY 1
ORDER BY 1;
```

92. Cohort retention pattern

Build cohorts by first activity month and measure returns by later months.

```
WITH first_seen AS (
  SELECT user_id, DATE_TRUNC('month', MIN(activity_at)) AS cohort
  FROM user_activity GROUP BY 1
), activity AS (
  SELECT user_id, DATE_TRUNC('month', activity_at) AS act_month
  FROM user_activity
)
SELECT f.cohort, a.act_month, COUNT(DISTINCT a.user_id) AS retained
FROM first_seen f JOIN activity a USING (user_id)
GROUP BY 1,2
ORDER BY 1,2;
```

93. JSON handling (example)

Use DB■specific JSON operators/functions to extract fields.

```
-- Postgres
SELECT payload->>'event' AS event_type,
       (payload->>'amount')::NUMERIC AS amount
FROM events;
```

94. Date/time pitfalls

Handle time zones, DST, inclusive/exclusive ranges; store UTC, display local.

95. Database testing checks for freshers

Validate row counts, duplicates, referential integrity, NULLs, and boundary rules.

```
SELECT 'orders' AS tbl, COUNT(*) FROM orders
UNION ALL
SELECT 'order_items', COUNT(DISTINCT order_id) FROM order_items;
```

96. SQL queries for employee data — examples

Common tasks: averages by dept, recent hires.

```
SELECT dept_id, AVG(salary) AS avg_sal FROM employees GROUP BY dept_id;
SELECT * FROM employees WHERE hire_date >= CURRENT_DATE - INTERVAL '30 day';
```

97. Transaction isolation levels

Read Uncommitted, Read Committed, Repeatable Read, Serializable — control read phenomena.

98. Why check the execution plan?

To see access paths (scan/seek), join types, and index usage to remove bottlenecks.

99. UPSERT (MERGE/ON CONFLICT)

Insert if not exists; else update — keeps data consistent.

```
-- Postgres
INSERT INTO customers (customer_id, name, email)
VALUES (1, 'Asha', 'asha@x.com')
ON CONFLICT (customer_id)
DO UPDATE SET email = EXCLUDED.email;
```

100. Top product by revenue per month

Aggregate then rank per month to pick the winner.

```
WITH monthly AS (
  SELECT DATE_TRUNC('month', o.order_date) AS mth,
         oi.product_id,
         SUM(oi.qty * oi.price) AS revenue
  FROM orders o JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY 1,2
), ranked AS (
  SELECT *, RANK() OVER (PARTITION BY mth ORDER BY revenue DESC) AS rnk
  FROM monthly
)
SELECT mth, product_id, revenue
FROM ranked
WHERE rnk = 1
ORDER BY mth;
```