

Date: 07.07.2025

SQL QUESTIONS:

1. What is a database? What are the different types of databases?

- **Database:** A structured collection of data stored and managed electronically, allowing easy access, management, and updating.
 - **Types of Databases:**
 - **Relational Databases (RDBMS):** Data stored in tables with relationships (e.g., MySQL, PostgreSQL).
 - **Non-Relational Databases (NoSQL):** Data stored as documents, key-value pairs, graphs, or wide-columns (e.g., MongoDB, Redis).
 - **Distributed Databases:** Data spread across multiple locations.
 - **Cloud Databases:** Hosted on cloud platforms.
 - **Hierarchical, Network Databases:** Older models with tree or graph structures.
-

2. What is a relational database? How is it different from non-relational databases?

- **Relational Database:** Organizes data into tables (rows and columns) with relationships defined by keys. Uses SQL for queries.
 - **Differences:**
 - Relational uses structured schema & SQL.
 - Non-relational uses flexible schema, often JSON-like or key-value, better for unstructured or rapidly changing data.
 - Relational prioritizes ACID transactions; NoSQL often prioritizes scalability and availability.
-

3. What is SQL? What are its major components?

- **SQL (Structured Query Language):** Language for managing and manipulating relational databases.
- **Major Components:**
 - **DDL (Data Definition Language):** Create, alter, drop tables.
 - **DML (Data Manipulation Language):** Insert, update, delete, select data.
 - **DCL (Data Control Language):** Grant, revoke permissions.
 - **TCL (Transaction Control Language):** Commit, rollback transactions.

4. What are the different types of SQL commands? Explain each briefly.

- **DDL:** Define schema.
 - CREATE, ALTER, DROP
- **DML:** Data operations.
 - SELECT, INSERT, UPDATE, DELETE
- **DCL:** Manage permissions.
 - GRANT, REVOKE
- **TCL:** Control transactions.
 - COMMIT, ROLLBACK, SAVEPOINT

5. What is the role of SQL in a DBMS?

- SQL is the standard language used by DBMS to define database structures, manipulate data, control access, and manage transactions, enabling communication between users/applications and the database.

6. How do you install and set up MySQL on a system?

- **Basic Steps:**
 1. Download MySQL installer from the official site (<https://dev.mysql.com/downloads/>).
 2. Run the installer and choose configuration type (Developer, Server, etc.).
 3. Set root password and create user accounts.
 4. Configure server options and start the MySQL service.
 5. Use MySQL Workbench or command line client to connect and manage databases.

7. What is the syntax for creating a database and tables in SQL?

sql

Copy code

-- Create a database

CREATE DATABASE database_name;

```
-- Use the database
USE database_name;

-- Create a table
CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints,
    ...
);
```

8. How do you insert a single and multiple records into a table?

sql

Copy code

-- Single record

```
INSERT INTO table_name (column1, column2) VALUES (value1, value2);
```

-- Multiple records

```
INSERT INTO table_name (column1, column2) VALUES
(value1, value2),
(value3, value4),
(value5, value6);
```

9. What are the key differences between DDL, DML, DCL, and TCL commands?

Command Type	Purpose	Examples	Effect
DDL	Define or modify schema	CREATE, ALTER, DROP	Changes database structure
DML	Manipulate data	SELECT, INSERT, UPDATE, DELETE	Modify data within tables
DCL	Control access/permissions	GRANT, REVOKE	Manage user privileges

Command Type	Purpose	Examples	Effect
TCL	Manage transactions	COMMIT, ROLLBACK	Control transaction integrity

10. What are some common SQL data types?

- **Numeric:** INT, FLOAT, DECIMAL, BIGINT
 - **Character:** CHAR, VARCHAR, TEXT
 - **Date/Time:** DATE, DATETIME, TIMESTAMP, TIME
 - **Boolean:** BOOLEAN or TINYINT(1)
 - **Binary:** BLOB
-

11. What is the purpose of the SELECT statement in SQL?

- The **SELECT** statement is used to **retrieve data** from one or more tables in a database.
 - It specifies which columns to return and can be combined with clauses to filter, sort, or group data.
-

12. How can you retrieve only specific columns from a table?

- By listing the column names after SELECT instead of using *.

sql

Copy code

```
SELECT column1, column2 FROM table_name;
```

This returns only the specified columns.

13. What does the DISTINCT keyword do in a SQL query?

- **DISTINCT removes duplicate rows** from the result set, returning only unique records.

Example:

sql

Copy code

```
SELECT DISTINCT city FROM customers;
```

Returns unique cities without repetition.

14. How would you filter records using the WHERE clause?

- The **WHERE** clause filters rows based on a specified condition.

Example:

sql

Copy code

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

Only employees with salary greater than 50,000 are returned.

15. What are comparison operators in SQL? Give examples.

- Operators to compare values in a condition:
 - = (equal to)
 - <> or != (not equal to)
 - < (less than)
 - > (greater than)
 - <= (less than or equal to)
 - >= (greater than or equal to)

Example:

sql

Copy code

```
SELECT * FROM products WHERE price <= 100;
```

16. What are logical operators (AND, OR, NOT) and how are they used in SQL?

- Combine multiple conditions:
 - AND: Both conditions must be true.
 - OR: Either condition can be true.
 - NOT: Negates a condition.

Example:

sql

Copy code

```
SELECT * FROM employees WHERE department = 'Sales' AND salary > 50000;
```

Returns sales employees earning more than 50,000.

17. How does the LIKE operator work? What do % and _ mean?

- LIKE is used for pattern matching in strings.
 - % matches **zero or more characters**.
 - _ matches **exactly one character**.

Example:

sql

Copy code

```
SELECT * FROM customers WHERE name LIKE 'J%n'; -- Names starting with J and ending with n
```

```
SELECT * FROM products WHERE code LIKE 'A_1'; -- Code with 'A', any one char, then '1'
```

18. How do BETWEEN and IN work for filtering values?

- BETWEEN filters values **within a range** (inclusive):

sql

Copy code

```
SELECT * FROM orders WHERE order_date BETWEEN '2025-01-01' AND '2025-01-31';
```

- IN filters for values **matching any value** in a list:

sql

Copy code

```
SELECT * FROM employees WHERE department IN ('HR', 'Finance', 'Sales');
```

19. What is the difference between IS NULL and = NULL?

- IS NULL is used to check if a value **is null** (missing value).
- = NULL **does not work** because NULL represents unknown, so equality comparisons fail.

Correct usage:

sql

Copy code

```
SELECT * FROM users WHERE last_login IS NULL;
```

20. How do you sort query results using ORDER BY? How do you sort by multiple columns?

- ORDER BY sorts the result by specified columns in **ascending (ASC)** or **descending (DESC)** order.

Example sorting by one column:

sql

Copy code

```
SELECT * FROM products ORDER BY price DESC;
```

Sorting by multiple columns:

sql

Copy code

```
SELECT * FROM employees ORDER BY department ASC, salary DESC;
```

- First sorts by department ascending, then within each department by salary descending.

21. What are aggregate functions in SQL? List and explain five with examples.

- **Aggregate functions** perform calculations on multiple rows and return a single result.

Five common ones:

- COUNT(): Counts number of rows or non-null values.
- SUM(): Adds up numeric values.
- AVG(): Calculates average of numeric values.
- MAX(): Finds the maximum value.
- MIN(): Finds the minimum value.

Example:

sql

Copy code

```
SELECT COUNT(*) AS total_employees FROM employees;
```

```
SELECT SUM(salary) AS total_salary FROM employees;
```

```
SELECT AVG(salary) AS average_salary FROM employees;
```

```
SELECT MAX(salary) AS highest_salary FROM employees;
```

```
SELECT MIN(salary) AS lowest_salary FROM employees;
```

22. How does the COUNT() function behave with NULL values?

- COUNT(*) counts **all rows**, including rows with NULLs.
- COUNT(column) counts only **non-NULL values** in that column.

Example:

sql

Copy code

```
SELECT COUNT(phone_number) FROM employees; -- counts only non-null phone numbers
```

23. What is the difference between SUM() and COUNT()?

- **SUM()** adds numeric values (returns total).
- **COUNT()** counts rows or non-null values (returns count).

Example:

sql

Copy code

```
SELECT SUM(quantity) FROM sales; -- total quantity sold
```

```
SELECT COUNT(*) FROM sales; -- total number of sales records
```

24. How do you group data in SQL using the GROUP BY clause?

- GROUP BY groups rows sharing the same values in specified columns, allowing aggregate functions to operate on each group.

Example:

sql

Copy code

```
SELECT department, COUNT(*) AS employee_count
```

```
FROM employees
```

```
GROUP BY department;
```

25. What is the purpose of the HAVING clause? How is it different from WHERE?

- **HAVING** filters groups after aggregation (e.g., filtering departments with >5 employees).

- **WHERE** filters rows **before** grouping and aggregation.

Example:

sql

Copy code

```
SELECT department, COUNT(*) AS employee_count  
FROM employees  
GROUP BY department  
HAVING COUNT(*) > 5;
```

26. Can you use aggregate functions in the WHERE clause? Why or why not?

- **No**, aggregate functions cannot be used in WHERE because WHERE filters rows before aggregation.
 - Use HAVING instead to filter based on aggregate values.
-

27. Write a query to find departments with more than 5 employees using GROUP BY and HAVING.

sql

Copy code

```
SELECT department, COUNT(*) AS num_employees  
FROM employees  
GROUP BY department  
HAVING COUNT(*) > 5;
```

28. How do you find the maximum and minimum salary from an employee table?

sql

Copy code

```
SELECT MAX(salary) AS max_salary, MIN(salary) AS min_salary  
FROM employees;
```

29. Write a query to calculate the average salary by department.

sql

Copy code

```
SELECT department, AVG(salary) AS average_salary  
FROM employees  
GROUP BY department;
```

30. What happens if you use GROUP BY without any aggregate functions?

- The query will **group the rows**, but if no aggregate functions are used in SELECT, it may return unpredictable results or cause errors depending on the SQL mode.
- Generally, GROUP BY without aggregation is uncommon because grouping implies summarization.

Example of unexpected behavior (some DBMS may allow):

sql

Copy code

```
SELECT department  
FROM employees  
GROUP BY department;
```

This just lists distinct departments, similar to SELECT DISTINCT department.

31. What are joins in SQL? Why are they used?

- **Joins** combine rows from two or more tables based on related columns.
 - Used to **retrieve related data spread across multiple tables** in a relational database.
-

32. What is the difference between INNER JOIN and LEFT JOIN?

- **INNER JOIN**: Returns only rows with matching keys in **both** tables.
 - **LEFT JOIN** (or LEFT OUTER JOIN): Returns all rows from the **left** table, plus matching rows from the right table; if no match, right columns show NULL.
-

33. Explain RIGHT JOIN and give a use case.

- **RIGHT JOIN** returns all rows from the **right** table and matching rows from the left table.
- If no match, left columns are NULL.

Use case: When you want all records from the “right” table regardless of matches in the left.

Example: List all departments and their employees; include departments even if they have no employees.

34. What is a FULL OUTER JOIN? Is it supported in MySQL?

- **FULL OUTER JOIN** returns rows when there is a match in **either** left or right table.
- Returns all rows from both tables, with NULLs where no matches exist.

MySQL does **not** natively support FULL OUTER JOIN. It can be emulated using UNION of LEFT and RIGHT joins.

35. What is a self join? How and why would you use it?

- A **self join** joins a table to itself, treating one copy as the “left” table and the other as the “right” table.
- Used to compare rows within the same table.

Example: To find employees who report to the same manager or to find hierarchical relationships.

36. Write a query using a self join to find employees who report to the same manager.

sql

Copy code

```
SELECT e1.employee_id, e1.name, e2.employee_id, e2.name, e1.manager_id
FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.manager_id
WHERE e1.employee_id <> e2.employee_id;
```

This lists pairs of employees sharing the same manager.

37. What is the purpose of using aliases in joins, especially in self joins?

- Aliases simplify **table referencing** and make queries more readable.
- In **self joins**, aliases distinguish between the two instances of the same table.

Example: employees e1 and employees e2

38. What is the difference between an equi join and a non-equi join?

- **Equi join:** Join condition uses **equality (=)** operator on columns.

Example:

sql

Copy code

```
SELECT * FROM A JOIN B ON A.id = B.id;
```

- **Non-equi join:** Join condition uses operators other than =, like <, >, BETWEEN.

Example:

sql

Copy code

```
SELECT * FROM sales s JOIN targets t ON s.amount BETWEEN t.min_amount AND t.max_amount;
```

39. What is a subquery? What are the types of subqueries in SQL?

- A **subquery** is a query nested inside another SQL query.
- Types of subqueries:
 - **Non-correlated subquery:** Independent query, runs once.
 - **Correlated subquery:** Depends on outer query values, runs for each row.

40. How is a correlated subquery different from a non-correlated subquery?

- **Non-correlated:** Runs once and provides a result for the outer query.
- **Correlated:** References columns from the outer query and runs repeatedly for each row.

41. Give an example of a subquery in a SELECT clause.

sql

Copy code

```
SELECT employee_id,  
       salary,  
       (SELECT AVG(salary) FROM employees) AS avg_salary  
FROM employees;
```

Here, the subquery calculates average salary once and shows it for each employee.

42. Write a query to find employees who earn more than the average salary using a subquery.

sql

Copy code

```
SELECT employee_id, name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

43. What is the difference between UNION and UNION ALL?

- **UNION:** Combines result sets and **removes duplicates**.
 - **UNION ALL:** Combines result sets **including duplicates**.
-

44. What is the INTERSECT operator used for? Is it available in MySQL?

- **INTERSECT** returns rows common to both queries (set intersection).
 - **MySQL** does **not support INTERSECT** natively but it can be emulated with INNER JOIN or EXISTS.
-

45. How can you combine JOIN, GROUP BY, and HAVING in a single query?


Example: Find departments with more than 10 employees and show their total salary.

sql

Copy code

```
SELECT d.department_name, COUNT(e.employee_id) AS employee_count,
SUM(e.salary) AS total_salary
FROM departments d
JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_name
HAVING COUNT(e.employee_id) > 10;
```

- JOIN combines tables.
- GROUP BY aggregates data by department.
- HAVING filters groups with more than 10 employees.

 **F. CASE Statements, Dates, and Conditional Logic (46–47)**

46. What is a CASE WHEN statement in SQL? Provide an example.

- CASE WHEN is used for conditional logic inside SQL queries, similar to if-else in programming.
- It returns values based on conditions.

Example:

sql

Copy code

```
SELECT employee_id, salary,
CASE
    WHEN salary > 80000 THEN 'High'
    WHEN salary BETWEEN 50000 AND 80000 THEN 'Medium'
    ELSE 'Low'
END AS salary_category
FROM employees;
```

This categorizes salaries into High, Medium, and Low.

47. How would you retrieve all employees who joined in the last 6 months using date functions?

Using the JOIN_DATE column and CURRENT_DATE function:

sql

Copy code

```
SELECT employee_id, name, join_date
FROM employees
WHERE join_date >= DATE_SUB(CURRENT_DATE, INTERVAL 6 MONTH);
```

This fetches employees who joined within the last 6 months from today.

G. Data Modification and Constraints (48–49)

48. Explain the purpose and syntax of INSERT, UPDATE, and DELETE commands.

- **INSERT:** Adds new rows to a table.

sql

Copy code

```
INSERT INTO employees (employee_id, name, salary)
```

```
VALUES (101, 'John Doe', 70000);
```

- **UPDATE:** Modifies existing rows.

```
sql
```

Copy code

```
UPDATE employees
```

```
SET salary = 75000
```

```
WHERE employee_id = 101;
```

- **DELETE:** Removes rows from a table.

```
sql
```

Copy code

```
DELETE FROM employees
```

```
WHERE employee_id = 101;
```

49. What are the different types of constraints in SQL? How do they ensure data integrity?

- **NOT NULL:** Ensures a column cannot have NULL values.
- **UNIQUE:** Ensures all values in a column are unique.
- **PRIMARY KEY:** Uniquely identifies each row, combination of NOT NULL + UNIQUE.
- **FOREIGN KEY:** Enforces referential integrity between tables.
- **CHECK:** Ensures values satisfy a condition.
- **DEFAULT:** Provides a default value when none is specified.

They **prevent invalid data** entry and maintain **consistency** across tables.

H. Transactions and ACID Properties (50)

What are transactions in SQL? Explain COMMIT, ROLLBACK, SAVEPOINT, and the ACID properties.

- A **transaction** is a sequence of SQL operations treated as a single unit of work. It ensures data integrity even in case of failure.
- **COMMIT:** Saves all changes made in the current transaction permanently.
- **ROLLBACK:** Undoes all changes since the last COMMIT.

- **SAVEPOINT:** Sets a point within a transaction to which you can roll back without undoing the entire transaction.
-

ACID Properties:

- **Atomicity:** All operations succeed or none do.
- **Consistency:** Data moves from one valid state to another.
- **Isolation:** Transactions run independently without interference.
- **Durability:** Once committed, changes persist even after failures.