70+ SQL INTERVIEW QUESTIONS:

FROM BEGINNER TO ADVANCED

WSDA
DATA NO DOUBT

WALTER SHIELDS
DATA
ACADEMY

Beginner-Level Questions

1. WHAT IS SQL, AND WHY IS IT IMPORTANT?

SQL (Structured Query Language) is used to interact with databases. It helps in retrieving, updating, and managing structured data efficiently.

2. WHAT ARE THE DIFFERENT TYPES OF SQL STATEMENTS?

SQL statements are categorized into:

- **DDL (Data Definition Language)** – CREATE, ALTER, DROP
- **DML (Data Manipulation Language)** – INSERT, UPDATE, DELETE
- **DCL (Data Control Language)** – GRANT, REVOKE
- **TCL (Transaction Control Language)** – COMMIT, ROLLBACK, SAVEPOINT
- **DQL (Data Query Language)** – SELECT

3. WHAT IS THE DIFFERENCE BETWEEN SQL AND MYSQL?

- **SQL** is a query language used to interact with databases.
- **MySQL** is a relational database management system (RDBMS) that uses SQL as its query language.

4. WHAT IS THE DIFFERENCE BETWEEN CHAR AND VARCHAR?

- **CHAR** is a fixed-length data type (e.g., CHAR(10) always takes 10 bytes).
- **VARCHAR** is a variable-length data type (e.g., VARCHAR(10) takes up only the required space).

5. WHAT IS THE DIFFERENCE BETWEEN DELETE, TRUNCATE, AND DROP?

- **DELETE** removes specific rows and can be rolled back.
- **TRUNCATE** removes all rows but retains the table structure.
- **DROP** deletes the table along with its structure.

6. WHAT IS NORMALIZATION IN SQL?

Normalization is the process of organizing a database to reduce redundancy and improve data integrity. It involves dividing large tables into smaller ones and establishing relationships.

7. WHAT ARE THE DIFFERENT NORMAL FORMS IN SQL?

1NF: Eliminate duplicate columns and ensure atomicity.
2NF: Meet 1NF and remove partial dependencies.
3NF: Meet 2NF and remove transitive dependencies.
BCNF: A stricter version of 3NF, ensuring all functional dependencies are handled.

8. WHAT IS DENORMALIZATION, AND WHEN IS IT USED?

Denormalization is the process of combining tables to improve read performance, often used in reporting or analytical applications where fast retrieval is more important than minimizing redundancy.

9. WHAT ARE PRIMARY AND FOREIGN KEYS?

- **Primary Key:** A unique identifier for a record in a table.
- **Foreign Key**: A reference to a primary key in another table to establish relationships between tables.

10. WHAT ARE THE DIFFERENT TYPES OF DATABASE RELATIONSHIPS?

- One-to-One (1:1)
- One-to-Many (1:M)
- Many-to-Many (M:M)

11. HOW DO YOU USE THE WHERE CLAUSE IN SQL?

The **WHERE** clause is used to filter records based on a specified condition.

12. WHAT IS THE ORDER BY CLAUSE?

The **ORDER BY** clause is used to sort the result set in ascending (ASC) or descending (DESC) order.

13. HOW DOES THE GROUP BY CLAUSE WORK?

GROUP BY is used to group rows with the same values in specified columns and apply aggregate functions.

14. WHAT IS THE HAVING CLAUSE, AND HOW IS IT DIFFERENT FROM WHERE?

The **WHERE** clause filters rows ***before*** aggregation, while **HAVING** filters ***after*** aggregation.

15. WHAT IS THE DIFFERENCE BETWEEN COUNT(), SUM(), AND AVG()?

- **COUNT():** Counts the number of rows.
- **SUM():** Adds up numerical values.
- **AVG():** Calculates the average of numerical values.

16. HOW CAN YOU USE DISTINCT IN A SQL QUERY?

DISTINCT eliminates duplicate values in a result set.

17. WHAT IS THE BETWEEN OPERATOR USED FOR?

BETWEEN filters values within a specified range.

18. HOW DOES THE IN OPERATOR WORK?

IN is used to filter rows matching a list of values

19. WHAT IS THE LIKE OPERATOR USED FOR?

LIKE is used for pattern matching in string values.

20. WHAT IS THE SELECT STATEMENT USED FOR?

The **SELECT** statement is used to retrieve data from a database table.

Intermediate-Level Questions

21. WHAT IS THE DIFFERENCE BETWEEN INNER JOIN, LEFT JOIN, RIGHT JOIN, AND FULL JOIN?

- **INNER JOIN**: Returns only matching records from both tables.
- **LEFT JOIN**: Returns all records from the left table and matching records from the right table. If no match is found, NULL values are returned for the right table columns.
- **RIGHT JOIN**: Returns all records from the right table and matching records from the left table. If no match is found, NULL values are returned for the left table columns.
- **FULL JOIN**: Returns all records from both tables, with NULLs in columns where there is no match.

22. HOW DOES A CROSS JOIN WORK?

A **CROSS JOIN** returns the Cartesian product of two tables, meaning it joins every row from the first table with every row from the second table.

23. WHAT IS A SELF JOIN?

A **SELF JOIN** is a join where a table is joined with itself, often using an alias to differentiate the instances of the table.

24. WHAT IS A NATURAL JOIN?

A **NATURAL JOIN** automatically joins tables on columns with the same name and compatible data types, eliminating duplicate columns in the result.

25. WHEN WOULD YOU USE A LEFT JOIN INSTEAD OF AN INNER JOIN?

You would use a **LEFT JOIN** when you need to retrieve all records from the left table regardless of whether there is a match in the right table. An INNER JOIN only returns matching rows from both tables.

26. WHAT HAPPENS WHEN THERE IS NO MATCH IN A RIGHT JOIN?

If there is no match in a RIGHT JOIN, NULL values are returned for columns from the left table.

27. WHAT ARE UNION AND UNION ALL, AND HOW DO THEY DIFFER?

- **UNION** combines the results of two queries and removes duplicate rows.
- **UNION ALL** combines results without removing duplicates.

28. WHAT IS THE DIFFERENCE BETWEEN EXISTS AND IN?

- **EXISTS**: Checks if a subquery returns any rows. It stops executing once it finds a match, making it more efficient in some cases.
- **IN**: Compares a value with a list of values and retrieves matching rows. It processes the entire list before returning results.

29. WHAT IS A COMPOSITE KEY?

A **composite key** is a primary key that consists of two or more columns that together uniquely identify a record in a table.

30. WHAT IS THE DIFFERENCE BETWEEN A PRIMARY KEY AND A UNIQUE KEY?

- **Primary Key:** Uniquely identifies each record and does not allow NULL values.
- **Unique Key:** Ensures uniqueness but allows one NULL value per column.

31. WHAT ARE AGGREGATE FUNCTIONS IN SQL?

Aggregate functions perform calculations on multiple rows and return a single value. Examples include:

- **SUM()** – Returns the sum of values.
- **AVG()** – Returns the average of values.
- **COUNT()** – Counts the number of rows.
- **MAX()** – Returns the maximum value.
- **MIN()** – Returns the minimum value.

32. WHAT ARE SCALAR FUNCTIONS?

Scalar functions return a single value based on the input. Examples include:

- **UPPER()** – Converts text to uppercase.
- **LOWER()** – Converts text to lowercase.
- **LEN()** – Returns the length of a string.

33. WHAT IS THE COALESCE() FUNCTION?

COALESCE() returns the first non-null value in a list.

34. WHAT DOES THE NULLIF() FUNCTION DO?

NULLIF() returns NULL if two values are equal; otherwise, it returns the first value.

35. WHAT ARE THE DIFFERENT STRING FUNCTIONS IN SQL?

- **LEN()** – Returns the length of a string.
- **SUBSTRING()** – Extracts a substring from a string.
- **REPLACE()** – Replaces part of a string.
- **TRIM()** – Removes leading/trailing spaces.

36. HOW CAN YOU USE THE CASE STATEMENT IN SQL?

The **CASE** statement is used to implement conditional logic.

37. WHAT IS THE CAST() FUNCTION USED FOR?

CAST() converts one data type to another.

38. HOW DOES THE CONVERT() FUNCTION DIFFER FROM CAST()?

CONVERT() is SQL Server-specific and allows formatting options, while **CAST()** is more standard and works across databases.

39. WHAT IS THE DIFFERENCE BETWEEN LEAD() AND LAG() FUNCTIONS?

- **LEAD()** retrieves the next row's value.
- **LAG()** retrieves the previous row's value.

40. WHAT IS THE DIFFERENCE BETWEEN RANK(), DENSE_RANK(), AND ROW_NUMBER()?

- **RANK()** assigns a rank but skips numbers for duplicate values.
- **DENSE_RANK()** assigns ranks sequentially without gaps.
- **ROW_NUMBER()** assigns a unique row number to each record.

41. WHAT IS A SUBQUERY IN SQL?

A **subquery** is a query inside another query, used to retrieve data for the main query.

42. WHAT IS A CORRELATED SUBQUERY?

A **correlated subquery** depends on the outer query and runs once per row processed by the outer query.

43. HOW DO YOU FIND THE SECOND-HIGHEST SALARY IN A TABLE?

To find the **second-highest salary**, you can use the **LIMIT** and **OFFSET** or a subquery:

```
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET 1;
```

Or using a **subquery**:

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

44. HOW DO YOU GET THE NTH HIGHEST SALARY IN SQL?

You can find the nth highest salary using a subquery with **LIMIT** and **OFFSET**:

```
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET (n-1);
```

Or using the **DENSE_RANK()** function:

```
SELECT salary
FROM (
   SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) as rnk
   FROM employees
) ranked
WHERE rnk = n;
```

45. WHAT IS A COMMON TABLE EXPRESSION (CTE)?

A **CTE** is a temporary result set that can be referenced multiple times in a query.

46. HOW DOES A RECURSIVE CTE WORK?

A **recursive CTE** calls itself until a termination condition is met.

47. HOW DO YOU PERFORM A PIVOT IN SQL?

A **pivot** converts row values into columns. In SQL Server, you can use the PIVOT operator:

```
SELECT *
FROM (
    SELECT department, employee, salary
    FROM employees
) SourceTable
PIVOT (
    SUM(salary)
    FOR department IN (Sales, IT, HR)
) AS PivotTable;
```

In databases like **PostgreSQL** or **MySQL**, you can use **CASE WHEN**:

```
SELECT
    employee,
    SUM(CASE WHEN department = 'Sales' THEN salary ELSE 0 END) AS Sales,
    SUM(CASE WHEN department = 'IT' THEN salary ELSE 0 END) AS IT,
    SUM(CASE WHEN department = 'HR' THEN salary ELSE 0 END) AS HR
FROM employees
GROUP BY employee;
```

48. WHAT IS AN UNPIVOT OPERATION IN SQL?

An **unpivot** operation converts columns into rows.

Example in **SQL Server** using UNPIVOT:

```
SELECT department, category, salary
FROM (
    SELECT department, Sales, IT, HR
    FROM employees
) SourceTable
UNPIVOT (
    salary FOR category IN (Sales, IT, HR)
) AS UnpivotTable;
```

In **other SQL versions**, you can achieve the same result using **UNION ALL**:

```
SELECT department, 'Sales' AS category, Sales AS salary FROM employees
UNION ALL
SELECT department, 'IT', IT FROM employees
UNION ALL
SELECT department, 'HR', HR FROM employees;
```

49. HOW DO YOU CHECK FOR DUPLICATE RECORDS IN A TABLE?

To check for duplicates, use the **GROUP BY** clause with **HAVING COUNT(*) > 1:**

```
SELECT name, COUNT(*)
FROM employees
GROUP BY name
HAVING COUNT(*) > 1;
```

50. HOW DO YOU DELETE DUPLICATE RECORDS WHILE KEEPING ONE COPY?

To delete duplicates while keeping one record, you can use **ROW_NUMBER():**

```
WITH DuplicateCTE AS (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY name, salary ORDER BY id) AS
row_num
    FROM employees
)
DELETE FROM employees
WHERE id IN (
    SELECT id FROM DuplicateCTE WHERE row_num > 1
);
```

In **MySQL**:

```
DELETE e1 FROM employees e1
INNER JOIN employees e2
ON e1.name = e2.name AND e1.salary = e2.salary
WHERE e1.id > e2.id;
```

Advanced-Level Questions

51. WHAT IS AN INDEX IN SQL?

An **index** is a database object that improves the speed of data retrieval operations by allowing the database engine to find rows more quickly.

52. WHAT ARE THE DIFFERENT TYPES OF INDEXES?

- **Clustered Index** – Sorts and stores data physically in the table.
- **Non-Clustered Index** – Stores pointers to the data instead of sorting it physically.
- **Unique Index** – Ensures that all values in a column are unique.
- **Composite Index** – Created on multiple columns to optimize specific queries.
- **Filtered Index** – Created with a WHERE clause to index a subset of data.

53. HOW DOES A CLUSTERED INDEX DIFFER FROM A NON-CLUSTERED INDEX?

- A **clustered index** determines the physical order of data and there can be only one per table.
- A **non-clustered index** stores pointers to data and allows multiple indexes per table.

54. WHAT IS A UNIQUE INDEX?

A **unique index** prevents duplicate values in a column, enforcing uniqueness.

55. WHOW DOES INDEXING IMPROVE PERFORMANCE?

Indexing speeds up queries by reducing the number of rows that need to be scanned, leading to **faster retrieval times**.

56. WHAT ARE SOME DOWNSIDES OF USING INDEXES?

- Indexes require **additional storage space**.
- They slow down **INSERT**, **UPDATE**, and **DELETE** operations because indexes must be updated when the data changes.

57. WHAT IS A COVERING INDEX?

A **covering index** contains all columns needed for a query, reducing the need to access the actual table.

58. WHAT IS AN INDEX SCAN VERSUS AN INDEX SEEK?

- **Index Scan** – Scans the entire index (less efficient).
- **Index Seek** – Directly searches for relevant data (more efficient).

59. WHAT IS A FILTERED INDEX?

A **filtered index** is created on a subset of data based on a specific condition.

60. WHAT IS A COMPOSITE INDEX?

A **composite index** is an index on multiple columns, improving performance for queries filtering by those columns.

61. WHAT IS A TRANSACTION IN SQL?

A **transaction** is a sequence of operations performed as a single unit of work.

62. WHAT ARE THE ACID PROPERTIES?

- **Atomicity** – All operations in a transaction are completed, or none are.
- **Consistency** – Ensures database integrity before and after transactions.
- **Isolation** – Prevents interference from other transactions.
- **Durability** – Ensures committed transactions remain saved.

63. WHAT ARE THE DIFFERENT ISOLATION LEVELS IN SQL?

- **Read Uncommitted** – Allows dirty reads.
- **Read Committed** – Prevents dirty reads.
- **Repeatable Read** – Prevents dirty and non-repeatable reads.
- **Serializable** – Ensures full isolation.

64. WHAT IS A DEADLOCK IN SQL?

A **deadlock** occurs when two transactions wait for each other to release locks, causing them to halt indefinitely.

65. HOW DO YOU PREVENT DEADLOCKS?

Use proper indexing, short transactions, and consistent locking orders.

66. WHAT IS OPTIMISTIC VS. PESSIMISTIC LOCKING?

- **Optimistic Locking** – Assumes conflicts are rare and checks before committing changes.
- **Pessimistic Locking** – Locks data to prevent conflicts.

67. WHAT IS A SAVEPOINT IN SQL TRANSACTIONS?

A **savepoint** allows rolling back part of a transaction.

68. WHAT HAPPENS WHEN A TRANSACTION IS ROLLED BACK?

All changes made in the transaction are **undone**, restoring the database to its previous state.

69. WHAT IS AN IMPLICIT TRANSACTION?

Automatically starts a transaction before a SQL statement is executed.

70. WHAT IS AN EXPLICIT TRANSACTION?

A transaction **manually defined** using BEGIN TRANSACTION and COMMIT/ROLLBACK.

71. HOW DO YOU OPTIMIZE SQL QUERIES FOR PERFORMANCE?

Use **indexes**, **optimize joins**, limit **SELECT** columns, analyze execution plans, and avoid unnecessary calculations.

72. WHAT IS A QUERY EXECUTION PLAN, AND HOW DO YOU ANALYZE IT?

A **query execution plan** shows how SQL Server executes a query. Use EXPLAIN PLAN or EXPLAIN ANALYZE to analyze it.

73. WHAT ARE SQL STORED PROCEDURES?

Stored procedures are precompiled SQL queries stored in the database, improving performance and security.

74. WHAT ARE SQL TRIGGERS, AND WHEN SHOULD THEY BE USED?

Triggers automatically execute SQL code in response to events (INSERT, UPDATE, DELETE).

75. WHAT ARE VIEWS IN SQL, AND HOW DO THEY IMPROVE QUERY PERFORMANCE?

A **view** is a virtual table based on a query. It simplifies complex queries and improves security.

76. WHAT IS THE DIFFERENCE BETWEEN MATERIALIZED AND NON-MATERIALIZED VIEWS?

- Materialized views store query results.
- Non-materialized views execute queries dynamically.

77. WHAT ARE USER-DEFINED FUNCTIONS (UDFS)?

Functions created by users to encapsulate reusable SQL logic.

78. WHAT IS THE DIFFERENCE BETWEEN DETERMINISTIC AND NON-DETERMINISTIC FUNCTIONS?

- **Deterministic functions** return the same output for the same input.
- **Non-deterministic functions** return varying results (e.g., **GETDATE()**).

79. WHAT ARE SOME COMMON SECURITY VULNERABILITIES IN SQL?

SQL injection, unauthorized access, improper privilege handling.

80. WHAT ARE SQL INJECTIONS, AND HOW DO YOU PREVENT THEM?

- **SQL injections** exploit input vulnerabilities to manipulate queries.
- Use **prepared statements, parameterized queries**, and **input validation** to prevent them.