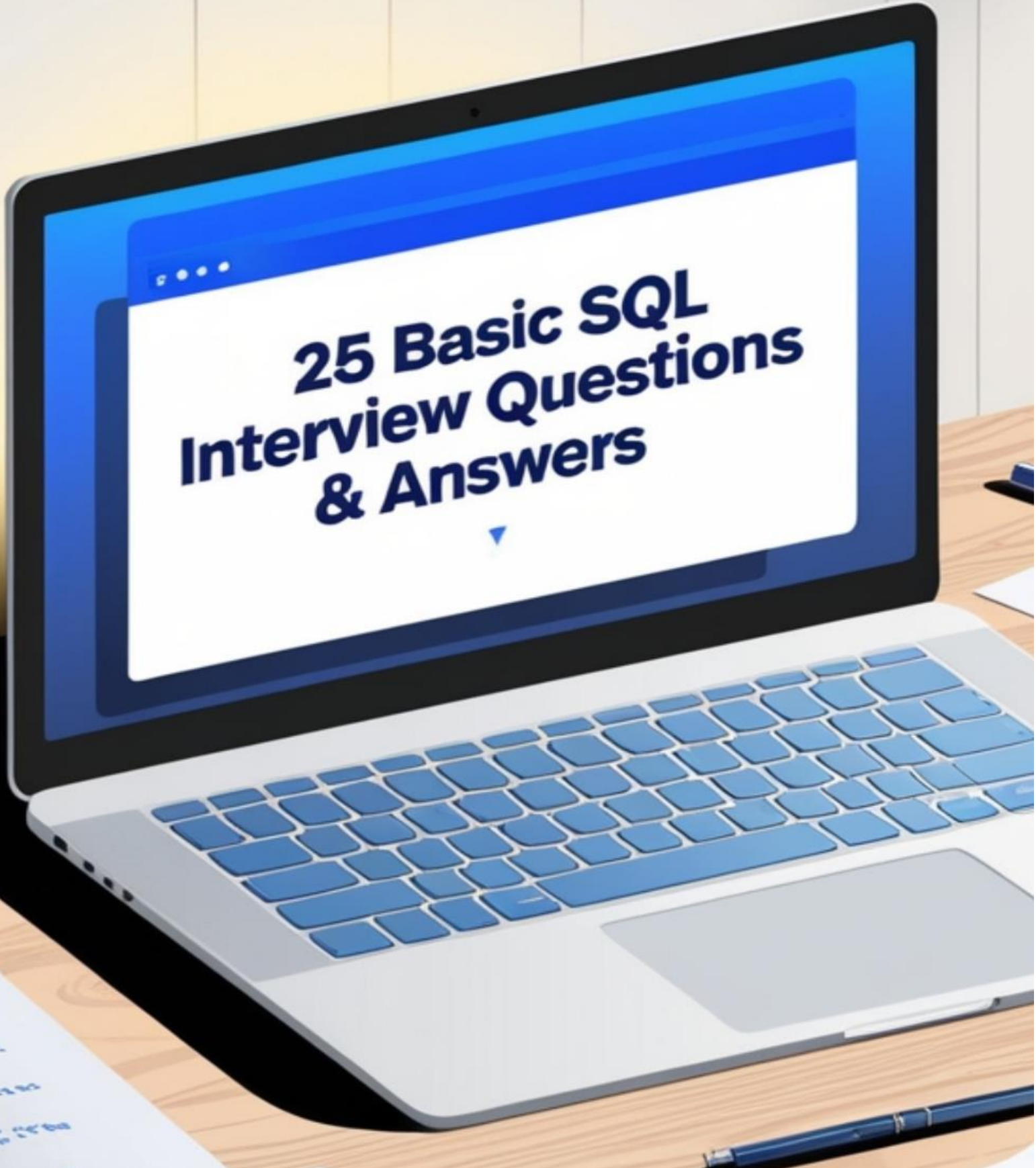


Best **SQL** Interview Questions with Answers



**25 Basic SQL
Interview Questions
& Answers**

The image shows a silver laptop on a wooden desk. The laptop screen displays a webpage with a blue header and a white content area. The text on the screen is '25 Basic SQL Interview Questions & Answers' in a bold, dark blue font. The laptop keyboard is visible, and a blue pen lies on the desk in the bottom right corner.

1. What is SQL, and what are its different types?

Answer:

SQL (Structured Query Language) is a programming language used to communicate with and manage data in relational databases. It allows users to perform operations such as querying, updating, inserting, and deleting data. SQL is also used to define and manipulate database structures.

Types of SQL Commands:

1. **DDL (Data Definition Language):** Defines the database structure.
 - Example: CREATE, ALTER, DROP
 - CREATE TABLE employees (id INT, name VARCHAR(50));
 2. **DML (Data Manipulation Language):** Manipulates data within tables.
 - Example: INSERT, UPDATE, DELETE
 - INSERT INTO employees (id, name) VALUES (1, 'John');
 3. **DCL (Data Control Language):** Controls access to data.
 - Example: GRANT, REVOKE
 - GRANT SELECT ON employees TO user;
 4. **TCL (Transaction Control Language):** Manages transactions.
 - Example: COMMIT, ROLLBACK
 - BEGIN TRANSACTION; INSERT INTO employees (id, name) VALUES (2, 'Jane'); COMMIT;
 5. **DQL (Data Query Language):** Retrieves data from the database.
 - Example: SELECT
 - SELECT * FROM employees;
-

2. What is the difference between SQL and MySQL?

Answer:

Feature	SQL	MySQL
Definition	A language for managing data.	A relational database management system (RDBMS) that uses SQL.
Purpose	Used to write and execute queries.	A software tool that stores and manages databases.
Execution	Cannot store data by itself.	Stores data and processes SQL commands.
Developer	Standardized by ANSI.	Developed by Oracle Corporation.

Example:

- SQL command: `SELECT * FROM employees;`
- MySQL processes this command and retrieves the data.

3. What are the different SQL commands? Explain each briefly.

Answer:

Command Type	Purpose	Examples
DDL	Defines the structure of a database.	CREATE TABLE, ALTER TABLE, DROP TABLE.
DML	Manipulates data within tables.	INSERT INTO, UPDATE, DELETE.
DCL	Controls user access.	GRANT, REVOKE.
TCL	Manages database transactions.	COMMIT, ROLLBACK.
DQL	Retrieves data.	SELECT.

Example:

1. DDL: `CREATE TABLE employees (id INT, name VARCHAR(50));`
2. DML: `INSERT INTO employees (id, name) VALUES (1, 'John');`
3. DQL: `SELECT * FROM employees;`

4. What is a primary key? Can a table have more than one primary key?

Answer:

A primary key is a unique identifier for a record in a table. It ensures that no two rows in a table have the same value for the primary key column(s).

Features:

- Cannot contain NULL values.
- Must be unique.

Can a table have more than one primary key?

No, a table can have only one primary key, but it can consist of multiple columns (a composite key).

Example:

```
CREATE TABLE students (  
    student_id INT PRIMARY KEY,  
    name VARCHAR(50),  
    age INT  
);
```

Composite Key Example:

```
CREATE TABLE orders (  
    order_id INT,  
    product_id INT,  
    PRIMARY KEY (order_id, product_id)  
);
```

5. What is a foreign key, and how is it different from a primary key?

Answer:

A foreign key is a column (or a set of columns) in one table that references the primary key in another table. It creates a relationship between the two tables.

Differences between Primary Key and Foreign Key:

Feature	Primary Key	Foreign Key
Purpose	Uniquely identifies a record.	Establishes a link between two tables.
Uniqueness	Must be unique.	Can contain duplicate values.
NULL Values	Cannot have NULL values.	Can have NULL values (if allowed).

Example:

-- Parent table

```
CREATE TABLE departments (  
    dept_id INT PRIMARY KEY,  
    dept_name VARCHAR(50)  
);
```

-- Child table

```
CREATE TABLE employees (  
    emp_id INT PRIMARY KEY,  
    emp_name VARCHAR(50),  
    dept_id INT,  
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id)  
);
```

In this example:

- dept_id is the **primary key** in the departments table.
 - dept_id in the employees table is a **foreign key** that references departments.
-

6. What is a join in SQL? Explain different types of joins.

Answer:

A **join** in SQL is used to combine rows from two or more tables based on a related column between them. It allows us to retrieve data from multiple tables in a relational database.

Types of Joins:

1. **INNER JOIN:** Returns records that have matching values in both tables.
2. **LEFT JOIN (LEFT OUTER JOIN):** Returns all records from the left table and matched records from the right table. Unmatched rows will contain NULL.
3. **RIGHT JOIN (RIGHT OUTER JOIN):** Returns all records from the right table and matched records from the left table. Unmatched rows will contain NULL.
4. **FULL OUTER JOIN:** Returns all records when there is a match in either table. Unmatched rows from both tables are filled with NULL.

Example Dataset:

-- Table: employees

id	name	deptId
1	Alice	101
2	Bob	102
3	Charlie	NULL

-- Table: departments

deptId	deptName
101	HR
102	IT
103	Finance



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

Answer:

Feature	INNER JOIN	LEFT JOIN
Definition	Returns only matching rows from both tables.	Returns all rows from the left table and matching rows from the right table.
Unmatched Rows	Not included in the result set.	Filled with NULL values in columns of the right table.

INNER JOIN Example:

```
SELECT employees.name, departments.deptName
FROM employees
INNER JOIN departments
ON employees.deptId = departments.deptId;
```

Result:

```
+-----+-----+
| name  | deptName|
+-----+-----+
| Alice | HR      |
| Bob   | IT      |
+-----+-----+
```

LEFT JOIN Example:

```
SELECT employees.name, departments.deptName
FROM employees
LEFT JOIN departments
ON employees.deptId = departments.deptId;
```

Result:

```
+-----+-----+
| name  | deptName|
+-----+-----+
| Alice  | HR      |
| Bob    | IT      |
| Charlie| NULL    |
+-----+-----+
```

8. How does the RIGHT JOIN work in SQL?

Answer:

The **RIGHT JOIN** (or **RIGHT OUTER JOIN**) returns all rows from the right table and matching rows from the left table. If there is no match, NULL values are filled in columns of the left table.

Example:

```
SELECT employees.name, departments.deptName
FROM employees
RIGHT JOIN departments
ON employees.deptId = departments.deptId;
```

Result:

name	deptName
Alice	HR
Bob	IT
NULL	Finance



9. What is a full outer join? How is it different from other joins?

Answer:

A **FULL OUTER JOIN** combines the results of both LEFT and RIGHT joins. It returns all rows when there is a match in either table and fills unmatched rows with NULL in respective columns.

Differences from Other Joins:

Join Type	Rows Returned
INNER	Only matching rows from both tables.
LEFT	All rows from the left table + matched rows.
RIGHT	All rows from the right table + matched rows.
FULL	All rows from both tables.

Example:

```
SELECT employees.name, departments.deptName
FROM employees
FULL OUTER JOIN departments
ON employees.deptId = departments.deptId;
```

Result:

```
+-----+-----+
| name   | deptName|
+-----+-----+
| Alice  | HR      |
| Bob    | IT      |
| Charlie| NULL    |
| NULL   | Finance |
+-----+-----+
```



10. What is the UNION operator in SQL, and how is it different from UNION ALL?

Answer:

The **UNION** operator combines the result sets of two or more SELECT statements and removes duplicate rows. **UNION ALL** also combines result sets but includes duplicates.

Key Differences:

Feature	UNION	UNION ALL
Duplicates	Removes duplicate rows.	Includes duplicate rows.
Performance	Slower due to duplicate removal.	Faster as no duplicate check.

Example Tables:

-- Table: table1

```
+-----+
| col |
+-----+
| A   |
| B   |
| C   |
+-----+
```

-- Table: table2

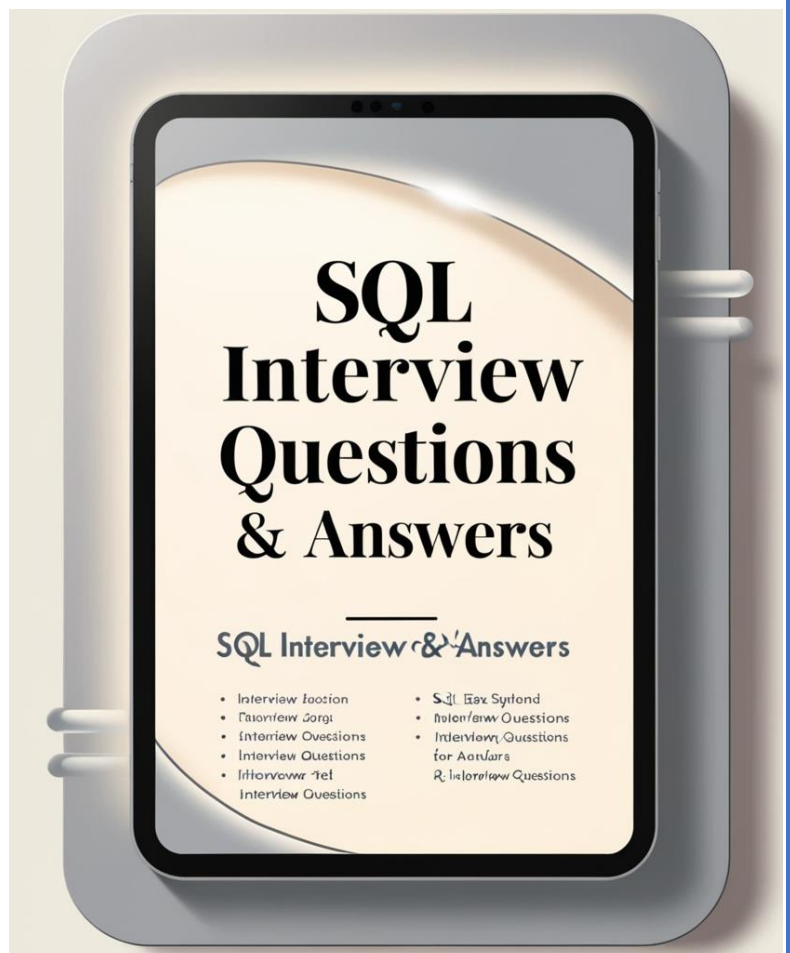
```
+-----+
| col |
+-----+
| B   |
| C   |
| D   |
+-----+
```

UNION Example:

```
SELECT col FROM table1
UNION
SELECT col FROM table2;
```

Result:

```
+-----+
| col |
+-----+
```



A
B
C
D

+-----+

UNION ALL Example:

SELECT col FROM table1

UNION ALL

SELECT col FROM table2;

Result:

+-----+

col

+-----+

A
B
C
B
C
D

+-----+

KHURSHID
MD ANWAR

WhatsApp: 91-9143407019

11. What is the GROUP BY clause used for?

Answer:

The **GROUP BY** clause is used to group rows that have the same values in specified columns into summary rows, like calculating aggregate functions such as SUM(), COUNT(), AVG(), etc.

Example:

We have a table sales:

-- Table: sales

category	product	sales
Electronics	TV	1000
Electronics	Laptop	1500
Furniture	Chair	300
Furniture	Table	700
Electronics	Phone	1200

Query:

```
SELECT category, SUM(sales) AS total_sales
FROM sales
GROUP BY category;
```

Result:

category	total_sales
Electronics	3700
Furniture	1000

12. How is the COUNT() function used in SQL?

Answer:

The **COUNT()** function is used to return the number of rows that match a specific condition or total rows in a table.

Example:

We have a table employees:

-- Table: employees

id	name	deptId
1	Alice	101
2	Bob	102
3	Charlie	NULL
4	David	101
5	Eve	102

Query 1: Count all employees:

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

Result:

total_employees
5

Query 2: Count employees in department 101:

```
SELECT COUNT(*) AS employees_in_dept_101  
FROM employees  
WHERE deptId = 101;
```

Result:

employees_in_dept_101
2

13. What is the difference between WHERE and HAVING clauses?

Answer:

Feature	WHERE	HAVING
Purpose	Filters rows before grouping.	Filters groups after GROUP BY.
Usage	Can't use aggregate functions.	Can use aggregate functions.
Query Level	Works on raw data rows.	Works on aggregated data after grouping.

Example:

We have the same sales table.

-- Table: sales

```
+-----+-----+-----+
| category | product | sales |
+-----+-----+-----+
| Electronics | TV      | 1000  |
| Electronics | Laptop  | 1500  |
| Furniture   | Chair   | 300   |
| Furniture   | Table   | 700   |
| Electronics | Phone   | 1200  |
+-----+-----+-----+
```

Query 1: Using WHERE:

```
SELECT product, sales
FROM sales
WHERE sales > 1000;
```

Result:

```
+-----+-----+
| product | sales |
+-----+-----+
```


| Laptop | 1500 |

| Phone | 1200 |

+-----+-----+

Query 2: Using HAVING:

SELECT category, SUM(sales) AS total_sales

FROM sales

GROUP BY category

HAVING total_sales > 1500;

Result:

+-----+-----+

| category | total_sales |

+-----+-----+

| Electronics | 3700 |

+-----+-----+

Khurshid MD Anwar

WhatsApp: 91-9143407019

14. What is the purpose of the DISTINCT keyword in SQL?

Answer:

The **DISTINCT** keyword is used to remove duplicate values from the result set, ensuring each value is unique.

Example:

We have a table orders:

-- Table: orders

id	customer	status
1	Alice	Shipped
2	Bob	Pending
3	Charlie	Shipped
4	Alice	Pending
5	Eve	Shipped

Query: Get distinct statuses:

```
SELECT DISTINCT status  
FROM orders;
```

Result:

status
Shipped
Pending

15. How can you use the LIKE operator in SQL?

Answer:

The **LIKE** operator is used to search for a specified pattern in a column. It often uses two wildcards:

- % to represent zero, one, or multiple characters.
- _ to represent a single character.

Example:

We have a table employees:

-- Table: employees

id	name	deptId
1	Alice	101
2	Bob	102
3	Charlie	NULL
4	David	101
5	Eve	102

Query 1: Names starting with "A":

```
SELECT name
FROM employees
WHERE name LIKE 'A%';
```

Result:

Alice

Query 2: Names ending with "e":

```
SELECT name  
FROM employees  
WHERE name LIKE '%e';
```

Result:

```
+-----+  
| name |  
+-----+  
| Alice |  
| Eve   |  
+-----+
```

Query 3: Names with second letter "o":

```
SELECT name  
FROM employees  
WHERE name LIKE '_o%';
```

Result:

```
+-----+  
| name |  
+-----+  
| Bob   |  
+-----+
```

Khurshid MD Anwar

WhatsApp: 91-9143407019

16. Explain the difference between the AND and OR operators in SQL.

Answer:

The **AND** and **OR** operators are used in SQL to combine multiple conditions in a query.

Feature	AND	OR
Functionality	All conditions must be true for the row to be included.	At least one condition must be true for the row to be included.
Use Case	Used for narrowing results (more restrictive).	Used for broadening results (less restrictive).
Effect	Results are fewer due to stricter criteria.	Results are more due to relaxed criteria.

Example:

We have a table employees:

-- Table: employees

```
+----+-----+-----+-----+
| id | name   | deptId | age |
+----+-----+-----+-----+
| 1 | Alice  | 101    | 30  |
| 2 | Bob    | 102    | 35  |
| 3 | Charlie| 103    | 40  |
| 4 | David  | 101    | 45  |
| 5 | Eve    | 102    | 50  |
+----+-----+-----+-----+
```

Query using AND: Find employees in department 101 and older than 40.

```
SELECT name, deptId, age FROM employees WHERE deptId = 101
AND age > 40;
```

Result:

```
+-----+-----+-----+
| name | deptId | age |
+-----+-----+-----+
| David | 101    | 45  |
+-----+-----+-----+
```

Query using OR: Find employees in department 101 or older than 40.

SELECT name, deptId, age

FROM employees

WHERE deptId = 101 OR age > 40;

Result:

```
+-----+-----+-----+
| name | deptId | age |
+-----+-----+-----+
| Alice | 101    | 30  |
| Charlie | 103   | 40  |
| David | 101    | 45  |
| Eve   | 102    | 50  |
+-----+-----+-----+
```

Khurshid MD Anwar

WhatsApp: 91-9143407019

17. What is a subquery in SQL? How is it used?

Answer:

A **subquery** is a query nested inside another query. It can be used to perform intermediate calculations or fetch data to be used in the outer query.

Features:

- A subquery is enclosed in parentheses.
- It can return a single value, multiple values, or a table.
- Commonly used with SELECT, INSERT, UPDATE, or DELETE.

Example:

We have tables employees and departments:

-- Table: employees

id	name	deptId
1	Alice	101
2	Bob	102
3	Charlie	103

-- Table: departments

deptId	deptName
101	HR
102	Sales

Query: Find names of employees who work in the "Sales" department.

SELECT name

```
FROM employees
WHERE deptId = (
    SELECT deptId
    FROM departments
    WHERE deptName = 'Sales'
);
```

Result:

```
+-----+
| name |
+-----+
| Bob  |
+-----+
```

Khurshid MD Anwar

WhatsApp: 91-9143407019

18. What is the difference between a correlated subquery and a non-correlated subquery?

Feature	Correlated Subquery	Non-Correlated Subquery
Dependency	Depends on the outer query for its values.	Independent of the outer query.
Execution	Executed repeatedly for each row of the outer query.	Executed once and the result is used in the outer query.
Performance	Slower due to repeated execution.	Faster as it is executed only once.

Example:

Correlated Subquery:

Find employees whose salary is greater than the average salary in their department:

```
SELECT e1.name, e1.salary
FROM employees e1
WHERE e1.salary > ( SELECT AVG(e2.salary) FROM employees e2
  WHERE e1.deptId = e2.deptId);
```

Non-Correlated Subquery:

Find employees who belong to department 101:

```
SELECT name
FROM employees
WHERE deptId IN (SELECT deptId FROM departments WHERE
deptName = 'HR');
```

Khurshid MD Anwar

WhatsApp: 91-9143407019

19. How does the ORDER BY clause work in SQL?

Answer:

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

Features:

- Defaults to ascending order if not specified.
- Multiple columns can be used for sorting.

Example:

We have a table students:

-- Table: students

```
+----+-----+----+
| id | name  | age |
+----+-----+----+
| 1  | Alice | 20  |
| 2  | Bob   | 22  |
| 3  | Charlie | 19  |
+----+-----+----+
```

Query 1: Sort students by age in ascending order:

```
SELECT name, age
FROM students
ORDER BY age ASC;
```

Result:

```
+-----+----+
| name  | age |
+-----+----+
| Charlie | 19  |
| Alice   | 20  |
```

Bob	22	
-----	----	--

+-----+	+-----+
---------	---------

Query 2: Sort students by age in descending order:

```
SELECT name, age
```

```
FROM students
```

```
ORDER BY age DESC;
```

Result:

+-----+	+-----+
---------	---------

name	age	
------	-----	--

+-----+	+-----+
---------	---------

Bob	22	
-----	----	--

Alice	20	
-------	----	--

Charlie	19	
---------	----	--

+-----+	+-----+
---------	---------

KHURSHID MD ANWAR
CONTACT FOR LIVE CLASSES
DATA ANALYTICS



9143407019



learnwithkhurshid.com



20. What are aggregate functions in SQL? List a few common ones.

Answer:

Aggregate functions perform calculations on multiple rows of data and return a single value.

Aggregate Function	Description	Example Query
COUNT()	Counts the number of rows.	SELECT COUNT(*) FROM employees;
SUM()	Calculates the total sum of a column.	SELECT SUM(salary) FROM employees;
AVG()	Calculates the average value of a column.	SELECT AVG(salary) FROM employees;
MAX()	Finds the maximum value in a column.	SELECT MAX(salary) FROM employees;
MIN()	Finds the minimum value in a column.	SELECT MIN(salary) FROM employees;

Example:

We have a table employees:

-- Table: employees

```
+----+-----+-----+
| id | name  | salary |
+----+-----+-----+
| 1  | Alice | 5000   |
| 2  | Bob   | 7000   |
| 3  | Charlie | 6000  |
+----+-----+-----+
```

Query: Calculate aggregate values for salaries:

```
SELECT
    COUNT(*) AS total_employees,
    SUM(salary) AS total_salary,
```


AVG(salary) AS average_salary,

MAX(salary) AS max_salary,

MIN(salary) AS min_salary

FROM employees;

Result:

total_employees	total_salary	average_salary	max_salary	min_salary	
3	18000	6000	7000	5000	

KHURSHID MD ANWAR

CONTACT FOR LIVE CLASSES

DATA ANALYTICS



9143407019



learnwithkhurshid.com



21. Explain the use of the LIMIT clause in SQL.

Answer:

The **LIMIT** clause in SQL is used to restrict the number of rows returned by a query.

- **Purpose:**

- Fetches a subset of rows from a result set.
- Useful for pagination or testing queries.

- **Syntax:**

SELECT columns

FROM table

LIMIT number_of_rows;

- **Optional OFFSET:**

- Used with LIMIT to skip a specified number of rows before starting to return results.
- Syntax:

SELECT columns

FROM table

LIMIT number_of_rows OFFSET offset_value;

Example:

We have a table students:

-- Table: students

+----+-----+----+

| id | name | age |

+----+-----+----+

| 1 | Alice | 20 |

| 2 | Bob | 22 |

| 3 | Charlie| 21 |

| 4 | David | 23 |

```
| 5 | Eve   | 19 |
```

```
+----+-----+----+
```

Query: Fetch the first 3 rows:

```
SELECT name, age
```

```
FROM students
```

```
LIMIT 3;
```

Result:

```
+-----+-----+
```

```
| name   | age |
```

```
+-----+-----+
```

```
| Alice  | 20 |
```

```
| Bob    | 22 |
```

```
| Charlie | 21 |
```

```
+-----+-----+
```

Query with OFFSET: Skip the first 2 rows and fetch the next 2 rows:

```
SELECT name, age
```

```
FROM students
```

```
LIMIT 2 OFFSET 2;
```

Result:

```
+-----+-----+
```

```
| name   | age |
```

```
+-----+-----+
```

```
| Charlie | 21 |
```

```
| David  | 23 |
```

```
+-----+-----+
```

22. What is a view in SQL? How is it different from a table?

Answer:

A **view** in SQL is a virtual table based on the result set of a query. It does not store data physically but fetches it dynamically from the underlying tables.

Feature	View	Table
Definition	A saved SQL query treated as a table.	A database object that stores data physically.
Storage	No physical storage for data.	Physically stores data in rows and columns.
Modifiability	Limited (depends on the SQL implementation).	Fully modifiable (INSERT, UPDATE, DELETE).
Use Case	Simplifies complex queries and enhances security.	Stores actual data.

Syntax:

```
CREATE VIEW view_name AS
SELECT columns
FROM table
WHERE condition;
```

Example:

We have a table employees:

-- Table: employees

```
+----+-----+-----+
| id | name   | deptId |
+----+-----+-----+
| 1  | Alice  | 101    |
| 2  | Bob    | 102    |
| 3  | Charlie| 101    |
```

+----+-----+-----+

Create a View: View to fetch employees from department 101:

```
CREATE VIEW dept101_employees AS
```

```
SELECT name
```

```
FROM employees
```

```
WHERE deptId = 101;
```

Query the View:

```
SELECT * FROM dept101_employees;
```

Result:

+-----+

| name |

+-----+

| Alice |

| Charlie |

+-----+

23. What is an index in SQL? How does it improve query performance?

Answer:

An **index** in SQL is a data structure that enhances the speed of data retrieval operations on a database table.

- **Key Features:**

- Works like an index in a book to locate data quickly.
- Does not affect the data itself.
- Improves read performance but may slow down INSERT, UPDATE, and DELETE operations due to index maintenance.

- **Types of Indexes:**

- **Unique Index:** Ensures all values in a column are unique.
- **Composite Index:** Built on multiple columns.

- **Full-Text Index:** Used for text search.
- **Clustered Index:** Physically reorders table data (only one per table).

Syntax:

CREATE INDEX index_name ON table_name(column_name);

Example:

We have a table products:

-- Table: products

```
+----+-----+-----+
| id | product  | price |
+----+-----+-----+
| 1  | Laptop   | 1000  |
| 2  | Phone    | 800   |
| 3  | Tablet   | 600   |
+----+-----+-----+
```

Create an Index on Price:

CREATE INDEX idx_price ON products(price);

Query without Index: Takes more time to find rows with price = 800.

Query with Index: Locates rows faster.

24. What is normalization, and why is it important?

Answer:

Normalization is a database design process that organizes data to reduce redundancy and improve data integrity.

- **Goals of Normalization:**

- Minimize redundancy.
- Ensure data consistency.
- Optimize storage space.
- Simplify maintenance.

- **Steps:**

- Divide tables into smaller tables.
- Define relationships between them using foreign keys.

Example:

A table before normalization:

-- Table: sales

id	customer	product	price
1	Alice	Laptop	1000
2	Bob	Phone	800
3	Alice	Tablet	600

After normalization:

Table 1: customers

id	customer
1	Alice
2	Bob

+----+-----+

Table 2: products

+----+-----+-----+

| id | product | price |

+----+-----+-----+

| 1 | Laptop | 1000 |

| 2 | Phone | 800 |

| 3 | Tablet | 600 |

+----+-----+-----+

SQL INTERVIEW QUESTIONS & ANSWERS



25. Explain the different normal forms in database design.

Answer:

Normal Form	Description	Example (Before and After)
1NF	Ensures all columns have atomic values.	A table where columns have multiple values (split into rows).
2NF	Ensures no partial dependency exists (relates to composite keys).	Split table into smaller tables, removing partial dependencies.
3NF	Ensures no transitive dependency (non-key columns should depend only on the primary key).	Remove indirect relationships and create separate tables.
BCNF	A stricter version of 3NF, ensures every determinant is a candidate key.	Further decomposition to remove anomalies.
4NF	Ensures no multi-valued dependencies.	Separate tables where a column has multiple independent values.

Example for 3NF:

Before:

```
+----+-----+-----+-----+
| id | customer | deptId | deptName |
+----+-----+-----+-----+
| 1 | Alice   | 101   | HR       |
| 2 | Bob     | 102   | Sales    |
+----+-----+-----+-----+
```

After:

Table 1: customers

```
+----+-----+-----+
```

id	customer	deptId
----	----------	--------

+----	+-----+	-----+
-------	---------	--------

1	Alice	101
---	-------	-----

2	Bob	102
---	-----	-----

+----	+-----+	-----+
-------	---------	--------

Table 2: departments

+-----+	-----+
---------	--------

deptId	deptName
--------	----------

+-----+	-----+
---------	--------

101	HR
-----	----

102	Sales
-----	-------

+-----+	-----+
---------	--------



Disclaimer:

The information shared in this EBook is based on my knowledge and experience. While I strive for accuracy, errors or mistakes may occur. Some details may vary depending on individual use cases or evolving practices. Please verify independently before making decisions based on this content. Thank you for your understanding!