Top SQL Interview Questions

from companies like Amazon, Google, Flipkart, eBay, Airbnb, Meta, Microsoft, Walmart, Deloitte, Zomato, Swiggy, PayPal, Stripe, and Shopify.

### Prepared by Narendiranath

### DSML Mentor

### Entri Elevate

### Question 1: Retrieve the Second Highest Salary

**Company**: Amazon, Google  
**Problem**: Write an SQL query to retrieve the second-highest salary from the Employee table.

Table: Employee

EmpID Name Salary

1 John 3000

2 Alice 5000

3 Bob 4000

**Solution**:

SELECT MAX(Salary) AS SecondHighestSalary

FROM Employee

WHERE Salary < (SELECT MAX(Salary) FROM Employee);

### Question 2: Find Customers with More Than One Order

**Company**: Flipkart, Walmart

**Problem**: Write an SQL query to find customers who have placed more than one order.

Table: Orders

OrderID CustomerID OrderDate

1 101 2023-01-01

2 102 2023-01-02

3 101 2023-01-03

**Solution**:

SELECT CustomerID

FROM Orders

GROUP BY CustomerID

HAVING COUNT(OrderID) > 1;

### Question 3: Find the Most Booked Listing

**Company**: Airbnb

**Problem**: Find the listing with the highest number of bookings.

Table: Bookings

BookingID ListingID BookingDate

1 201 2023-01-01

2 202 2023-01-02

3 201 2023-01-03

**Solution**:

SELECT ListingID, COUNT(\*) AS BookingCount

FROM Bookings

GROUP BY ListingID

ORDER BY BookingCount DESC

LIMIT 1;

### Question 4: Calculate the Total Transaction Amount

**Company**: PayPal, Stripe

**Problem**: Calculate the total transaction amount for each customer.

Table: Transactions

TransactionID CustomerID Amount

1 101 200

2 102 300

3 101 150

Sol:

SELECT CustomerID, SUM(Amount) AS TotalAmount

FROM Transactions

GROUP BY CustomerID;

### Question 5: Identify Top-Selling Products

**Company:** Shopify, eBay

**Problem**: Identify the top-selling products by total sales.

Table: Sales

ProductID QuantitySold Price

1 10 50

2 5 100

3 8 40

Sol:

SELECT ProductID, SUM(QuantitySold \* Price) AS TotalSales

FROM Sales

GROUP BY ProductID

ORDER BY TotalSales DESC;

### Question 6: Retrieve Customers Without Orders

**Company**: Walmart, Zomato

**Problem**: Find customers who have not placed any orders.

Tables:

Customers

CustomerID Name

101 John

102 Alice

103 Bob

Tables: Orders

OrderID CustomerID

1 101

2 102

Sol:

SELECT c.CustomerID, c.Name

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

WHERE o.OrderID IS NULL;

### Question 7: Rank Employees by Salary

**Company:** Microsoft, Deloitte

**Problem:** Rank employees by their salary.

Table: Employee

EmpID Name Salary

1 John 3000

2 Alice 5000

3 Bob 4000

SELECT EmpID, Name, Salary, RANK() OVER (ORDER BY Salary DESC) AS Rank

FROM Employee;

**Explanation:**

The RANK() function in SQL is a window function that assigns a rank to each row in a result set based on the order specified in the OVER clause. It is particularly useful when you need to rank rows based on a specific column while allowing for ties.

**RANK() Function:**

The RANK() function assigns a unique rank to each row within a partition of a result set.

If two or more rows have the same value in the column being ranked (in this case, Salary), they receive the same rank, but the next rank is skipped (i.e., a gap appears in the sequence).

OVER Clause:

The OVER clause specifies how the ranking should be calculated.

ORDER BY Salary DESC: This orders the rows in descending order of Salary before assigning ranks. Higher salaries get a higher priority.

AS Rank:

This renames the output column of the ranking result as Rank.

**Key Characteristics of RANK():**

Handles Ties:

Rows with identical values are given the same rank.

Gaps in Ranking:

If there are ties, the ranks skip the numbers that would otherwise have been assigned to the tied rows.

### Question 8: Find Restaurants with No Ratings

**Company:** Zomato, Swiggy

**Problem:** Identify restaurants that have not received any ratings.

Tables:

Restaurants

RestaurantID Name

1 Pizza Place

2 Burger Spot

3 Sushi House

**Ratings**

RatingID RestaurantID Rating

1 1 5

Sol:

SELECT r.RestaurantID, r.Name

FROM Restaurants r

LEFT JOIN Ratings rat ON r.RestaurantID = rat.RestaurantID

WHERE rat.RatingID IS NULL;

## Questions from Amazon, Google, Flipkart, PayPal

### Question 9. Find employees who earn more than the average salary of their department.

Table: Employee

EmpID Name Salary DeptID

1 John 3000 101

2 Alice 5000 102

3 Bob 4000 101

Sol:

SELECT EmpID, Name, Salary, DeptID

FROM Employee e

WHERE Salary > (SELECT AVG(Salary)

FROM Employee

WHERE DeptID = e.DeptID);

### Question 10. Retrieve a list of customers and their orders, including customers with no orders.

Tables:

Customers

CustomerID Name

1 John

2 Alice

3 Bob

Orders

OrderID CustomerID OrderAmount

101 1 200

102 1 300

103 2 150

Solution:

SELECT c.CustomerID, c.Name, o.OrderID, o.OrderAmount

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID;

### Question 11. Retrieve the top three salaries from the Employee tables

Sol:

SELECT DISTINCT Salary

FROM Employee

ORDER BY Salary DESC

LIMIT 3;

### Question 12. Write a stored procedure that takes a customer ID as input and returns all orders for that customer.

Sol:

DELIMITER $$

CREATE PROCEDURE GetCustomerOrders (IN customerID INT)

BEGIN

SELECT OrderID, OrderAmount

FROM Orders

WHERE CustomerID = customerID;

END$$

DELIMITER ;

CALL GetCustomerOrders(1);

### Question 13. Count the total number of orders placed by each customer.

SELECT CustomerID, COUNT(OrderID) AS TotalOrders

FROM Orders

GROUP BY CustomerID;

### Question 14. Calculate the total sales for each month

Table: Sales

SaleID SaleAmount SaleDate

1 100 2023-01-01

2 200 2023-01-15

3 150 2023-02-10

SELECT DATE\_FORMAT(SaleDate, '%Y-%m') AS SaleMonth, SUM(SaleAmount) AS TotalSales

FROM Sales

GROUP BY SaleMonth;

### Question 15. Identify duplicate customer names in the Customers table.

SELECT Name, COUNT(\*) AS Count

FROM Customers

GROUP BY Name

HAVING COUNT(\*) > 1;

### Question 16. Find orders where the total amount exceeds the average of all orders.

Table: Orders

OrderID OrderAmount

101 200

102 300

103 150

Sol:

SELECT OrderID, OrderAmount

FROM Orders

WHERE OrderAmount > (SELECT AVG(OrderAmount) FROM Orders);

### Question 17. Rank products based on their total sales

Table: ProductSales

ProductID QuantitySold Price

1 10 20

2 5 50

3 8 15

Sol:

SELECT ProductID, SUM(QuantitySold \* Price) AS TotalSales,

RANK() OVER (ORDER BY SUM(QuantitySold \* Price) DESC) AS Rank

FROM ProductSales

GROUP BY ProductID;

**Explanation:**

ProductID: The unique identifier for each product in the table.

SUM(QuantitySold \* Price): This calculates the total revenue (or sales) for each product by multiplying the quantity sold by the price for each row and summing it up for each ProductID.

GROUP BY ProductID: Aggregates the sales calculation (SUM) for each ProductID.

RANK() OVER (ORDER BY SUM(QuantitySold \* Price) DESC): Assigns a rank to each product based on its total sales, ordered from highest to lowest.

### Question 18. Create a view to retrieve customers with total orders greater than $500.

CREATE VIEW HighValueCustomers AS

SELECT c.CustomerID, c.Name, SUM(o.OrderAmount) AS TotalSpent

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.Name

HAVING TotalSpent > 500;

SELECT \* FROM HighValueCustomers;

What is a Window Function in MySQL?

A window function in MySQL is a powerful feature that allows you to perform calculations across a specific subset or "window" of rows within your dataset. Unlike aggregate functions (e.g., SUM, AVG), which collapse rows into a single result per group, window functions retain the individual rows while applying calculations over the defined window.

Introduced in MySQL 8.0, window functions enable operations like ranking, cumulative totals, moving averages, and percentages.

Syntax of Window Functions

<window\_function> OVER (

[PARTITION BY column\_name]

[ORDER BY column\_name]

)

**Components:**

<window\_function>: The function to apply (e.g., SUM, AVG, RANK, ROW\_NUMBER, etc.).

PARTITION BY (Optional): Divides the data into subsets (similar to GROUP BY) for the function to operate within.

ORDER BY (Optional): Specifies the order of rows within each partition for calculations.

**Types of Window Functions**

**Aggregate Functions:**

SUM(), AVG(), COUNT(), MAX(), MIN() applied over a window.

**Ranking Functions:**

ROW\_NUMBER(), RANK(), DENSE\_RANK(), NTILE().

**Value Functions:**

LEAD(), LAG(), FIRST\_VALUE(), LAST\_VALUE().

Key Use Cases of Window Functions

1. Ranking

SELECT EmployeeID, Name, Salary,

RANK() OVER (ORDER BY Salary DESC) AS Rank

FROM Employee;

Ranks employees based on salary.

Retains all rows, showing their relative rank.

2. Cumulative Sum

SELECT SalesID, Region, Amount,

SUM(Amount) OVER (PARTITION BY Region ORDER BY SalesID) AS CumulativeSales

FROM SalesData;

Calculates running totals (CumulativeSales) for each region.

3. Moving Average

SELECT SalesID, Region, Amount,

AVG(Amount) OVER (PARTITION BY Region ORDER BY SalesID ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS MovingAvg

FROM SalesData;

Computes a rolling average over the current row and the two preceding rows.

4. Accessing Row Values

SELECT SalesID, ProductID, Amount,

LAG(Amount) OVER (PARTITION BY ProductID ORDER BY SalesID) AS PreviousAmount,

LEAD(Amount) OVER (PARTITION BY ProductID ORDER BY SalesID) AS NextAmount

FROM SalesData;

LAG retrieves the value from the previous row.

LEAD retrieves the value from the next row.

How Window Functions Differ from Aggregate Functions

**Feature Window Function Aggregate Function**

Output Rows Retains all rows Collapses into one result

per group

Scope Operates on a "window" of rows Operates on entire groups

Order-sensitive Can use ORDER BY within the window Not inherently order-sensitive

Interview questions based on windows functions

### 1. Amazon

### Question: Rank products by total sales within each category.

SELECT Category, ProductID,

SUM(QuantitySold \* Price) AS TotalSales,

RANK() OVER (PARTITION BY Category ORDER BY SUM(QuantitySold \* Price) DESC) AS Rank

FROM ProductSales

GROUP BY Category, ProductID;

Window Function: RANK() OVER (PARTITION BY Category ORDER BY SUM(QuantitySold \* Price) DESC)

Explanation:

Calculates the total sales (SUM(QuantitySold \* Price)) for each product.

Partitions data by Category to rank products within their respective categories based on sales.

Use Case: Identify best-selling products in each category for inventory management.

### 2. Google

### Question: Find the top 3 employees with the highest salaries in each department.

SELECT DepartmentID, EmployeeID, Name, Salary,

RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS Rank

FROM Employees

WHERE RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) <= 3;

Solution:

Window Function: RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC)

Explanation:

The RANK() function ranks employees within each department (PARTITION BY DepartmentID) based on their salary in descending order.

The WHERE clause filters only the top 3 employees per department.

Use Case: Analyze top performers in different departments for bonus allocation.

### 3. Flipkart

### Question: Calculate the cumulative sales for each region.

SELECT Region, SalesID, SalesAmount,

SUM(SalesAmount) OVER (PARTITION BY Region ORDER BY SalesID) AS CumulativeSales

FROM RegionalSales;

Window Function: SUM(SalesAmount) OVER (PARTITION BY Region ORDER BY SalesID)

Explanation:

Partitions the data by Region.

Calculates a running total (SUM) of SalesAmount for each SalesID in the region, ordered by SalesID.

Use Case: Track sales progress over time in each region.

### 4. Microsoft

### Question: Identify employees who earn more than the department average.

SELECT EmployeeID, Name, DepartmentID, Salary,

AVG(Salary) OVER (PARTITION BY DepartmentID) AS AvgDepartmentSalary

FROM Employees

WHERE Salary > AVG(Salary) OVER (PARTITION BY DepartmentID);

Window Function: AVG(Salary) OVER (PARTITION BY DepartmentID)

Explanation:

Calculates the average salary for each department (PARTITION BY DepartmentID).

Filters employees whose salaries are above the department average.

Use Case: Highlight employees eligible for promotion or leadership roles.

### 5. Meta (Facebook)

### Question: Calculate the percentage contribution of each post to total likes per user.

SELECT UserID, PostID, Likes,

SUM(Likes) OVER (PARTITION BY UserID) AS TotalLikesPerUser,

ROUND((Likes \* 100.0 / SUM(Likes) OVER (PARTITION BY UserID)), 2) AS PercentageContribution

FROM UserPosts;

Window Function:

SUM(Likes) OVER (PARTITION BY UserID)

ROUND((Likes \* 100.0 / SUM(Likes) OVER (PARTITION BY UserID)), 2)

Explanation:

Calculates the total likes for each user.

Determines the percentage contribution of each post to the user's total likes.

Use Case: Analyze user engagement and content effectiveness.

### 6. Airbnb

### Question: Find the first and last booking dates for each host.

SELECT HostID, GuestID, BookingDate,

FIRST\_VALUE(BookingDate) OVER (PARTITION BY HostID ORDER BY BookingDate ASC) AS FirstBooking,

LAST\_VALUE(BookingDate) OVER (PARTITION BY HostID ORDER BY BookingDate ASC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS LastBooking

FROM Bookings;

Window Function:

FIRST\_VALUE(BookingDate) OVER (PARTITION BY HostID ORDER BY BookingDate ASC)

LAST\_VALUE(BookingDate) OVER (PARTITION BY HostID ORDER BY BookingDate ASC ROWS...)

Explanation:

FIRST\_VALUE retrieves the earliest booking date for each host.

LAST\_VALUE retrieves the latest booking date, but the range clause ensures all rows are considered.

Use Case: Track hosting history.

### 7. Walmart

### Question: Calculate the moving average of weekly sales for each store.

SELECT StoreID, Week, Sales,

AVG(Sales) OVER (PARTITION BY StoreID ORDER BY Week ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS MovingAvg

FROM WeeklySales;

Window Function: AVG(Sales) OVER (PARTITION BY StoreID ORDER BY Week ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)

Explanation:

Partitions by StoreID.

Computes a rolling average of Sales for the current week and the two preceding weeks.

Use Case: Monitor weekly sales trends for operational decisions.

### 8. Stripe

### Question: Identify the last transaction amount for each user.

SELECT UserID, TransactionID, TransactionAmount,

LAG(TransactionAmount) OVER (PARTITION BY UserID ORDER BY TransactionID) AS PreviousTransaction

FROM Transactions;

Window Function: LAG(TransactionAmount) OVER (PARTITION BY UserID ORDER BY TransactionID)

Explanation:

Retrieves the amount of the previous transaction for each user.

Use Case: Detect spending patterns or anomalies.

### 9. Deloitte

### Question: Calculate the percentage growth in sales for each quarter.

SELECT Quarter, Sales,

(Sales - LAG(Sales) OVER (ORDER BY Quarter)) \* 100.0 / LAG(Sales) OVER (ORDER BY Quarter) AS GrowthPercentage

FROM QuarterlySales;

Window Function:

LAG(Sales) OVER (ORDER BY Quarter)

Explanation:

Calculates the sales difference from the previous quarter and divides it by the previous quarter's sales to get the growth percentage.

Use Case: Analyze sales growth trends.

### 10. Swiggy

### Question: Rank delivery partners by number of orders handled in each city.

SELECT City, PartnerID, COUNT(OrderID) AS OrdersHandled,

DENSE\_RANK() OVER (PARTITION BY City ORDER BY COUNT(OrderID) DESC) AS Rank

FROM Deliveries

GROUP BY City, PartnerID;

Window Function: DENSE\_RANK() OVER (PARTITION BY City ORDER BY COUNT(OrderID) DESC)

Explanation:

Ranks delivery partners in each city based on the total number of orders handled.

DENSE\_RANK ensures no gaps in ranking.

Use Case: Evaluate delivery efficiency.

HR round SQL interview questions

### 1. Amazon

### Question: Explain the difference between WHERE and HAVING with an example.

Answer:

WHERE is used to filter rows before any groupings are made.

HAVING is used to filter groups after aggregations are performed.

Example:

-- Using WHERE

SELECT ProductID, Quantity, Price

FROM Products

WHERE Price > 100; -- Filters rows with price > 100 before grouping

-- Using HAVING

SELECT CategoryID, AVG(Price) AS AvgPrice

FROM Products

GROUP BY CategoryID

HAVING AVG(Price) > 100; -- Filters groups where average price > 100

Explanation:

Use WHERE for raw data filtering and HAVING for aggregate filtering.

Real-world scenario: Analyze products with high prices vs. categories with high average prices.

### 2. Google

### Question: What are the different types of JOINs in SQL? Can you give examples?

Answer:

INNER JOIN: Returns rows with matching values in both tables.

LEFT JOIN: Returns all rows from the left table and matching rows from the right table.

RIGHT JOIN: Returns all rows from the right table and matching rows from the left table.

FULL OUTER JOIN: Returns all rows when there's a match in either table.

Example:

-- INNER JOIN

SELECT A.EmployeeID, B.DepartmentName

FROM Employees A

INNER JOIN Departments B ON A.DepartmentID = B.DepartmentID;

-- LEFT JOIN

SELECT A.EmployeeID, B.DepartmentName

FROM Employees A

LEFT JOIN Departments B ON A.DepartmentID = B.DepartmentID;

-- RIGHT JOIN

SELECT A.EmployeeID, B.DepartmentName

FROM Employees A

RIGHT JOIN Departments B ON A.DepartmentID = B.DepartmentID;

-- FULL OUTER JOIN

SELECT A.EmployeeID, B.DepartmentName

FROM Employees A

FULL OUTER JOIN Departments B ON A.DepartmentID = B.DepartmentID;

Explanation:

INNER JOIN: Focuses on common data.

LEFT JOIN/RIGHT JOIN: Focuses on one table's complete data with partial matches.

FULL OUTER JOIN: Combines unmatched rows from both sides.

### 3. Flipkart

### Question: How would you find duplicate records in a table?

Answer:

SELECT Column1, Column2, COUNT(\*)

FROM TableName

GROUP BY Column1, Column2

HAVING COUNT(\*) > 1;

Explanation:

Groups rows based on specific columns and counts occurrences.

Filters only those groups where COUNT(\*) > 1 (duplicates).

Use case: Identify duplicate orders in an e-commerce database.

### 4. Deloitte

### Question: What are indexes in SQL, and why are they used?

Answer:

Index: A database object that improves query performance by allowing faster retrieval of records.

Types:

Clustered Index: Alters the physical order of the table.

Non-clustered Index: Does not affect the physical order of the table.

Example:

-- Create an index on a column

CREATE INDEX idx\_employee\_name ON Employees(Name);

-- Query uses the index for faster retrieval

SELECT \* FROM Employees WHERE Name = 'John Doe';

Explanation:

Indexes are like book indexes—quickly locate information without scanning the entire book.

Use case: Faster lookups for customer details in a database.

### 5. Airbnb

### Question: What is a subquery? Can you provide an example?

Answer:

A subquery is a query nested inside another query.

Types:

Single-row subquery: Returns one value.

Multi-row subquery: Returns multiple rows.

Correlated subquery: Depends on the outer query.

Example:

-- Single-row subquery

SELECT Name, Salary

FROM Employees

WHERE Salary > (SELECT AVG(Salary) FROM Employees);

-- Correlated subquery

SELECT E1.Name, E1.Salary

FROM Employees E1

WHERE E1.Salary > (SELECT AVG(E2.Salary) FROM Employees E2 WHERE E1.DepartmentID = E2.DepartmentID);

Explanation:

Single-row: Filters based on overall salary average.

Correlated: Filters based on department-specific averages.

Use case: Identify above-average performers in different scenarios.

### 6. Meta (Facebook)

### Question: How would you calculate the total number of posts and the average likes per user?

Answer:

SELECT UserID, COUNT(PostID) AS TotalPosts, AVG(Likes) AS AvgLikes

FROM Posts

GROUP BY UserID;

Explanation:

COUNT(PostID): Total posts by each user.

AVG(Likes): Average likes received by each user.

Use case: Analyze user engagement on a platform.

### 7. Walmart

### Question: How can you delete duplicate rows but keep one instance in SQL?

Answer:

WITH CTE AS (

SELECT \*, ROW\_NUMBER() OVER (PARTITION BY Column1, Column2 ORDER BY ID) AS RowNum

FROM TableName

)

DELETE FROM TableName

WHERE ID IN (SELECT ID FROM CTE WHERE RowNum > 1);

Explanation:

Assigns a unique RowNum to duplicates using ROW\_NUMBER().

Deletes rows where RowNum > 1 to retain the first instance.

Use case: Clean up redundant records in a sales database.

### 8. Stripe

### Question: Explain the difference between RANK() and DENSE\_RANK() with an example.

Answer:

RANK(): Skips ranks when there’s a tie.

DENSE\_RANK(): Does not skip ranks.

Example:

SELECT ProductID, Price,

RANK() OVER (ORDER BY Price DESC) AS Rank,

DENSE\_RANK() OVER (ORDER BY Price DESC) AS DenseRank

FROM Products;

Explanation:

Tied prices (e.g., $100 and $100) will have the same rank.

RANK(): Next rank skips (e.g., 1, 2, 2, 4).

DENSE\_RANK(): Consecutive ranks (e.g., 1, 2, 2, 3).

Use case: Different ranking systems for competition.

### 9. Swiggy

### Question: How can you find the second-highest salary in a table?

Answer:

SELECT MAX(Salary) AS SecondHighestSalary

FROM Employees

WHERE Salary < (SELECT MAX(Salary) FROM Employees);

Explanation:

Subquery fetches the highest salary.

Filters out the highest salary to find the second-highest.

Use case: Highlight the runner-up in salary competitions.

### 10. PayPal

### Question: How do you handle NULL values in SQL?

Answer:

Use functions like COALESCE or ISNULL to replace NULL values.

Example:

SELECT OrderID, COALESCE(Discount, 0) AS Discount

FROM Orders;

Explanation:

COALESCE(Discount, 0): Replaces NULL with 0.

Use case: Ensure accurate calculations when dealing with missing data.

Most frequently asked SQL multiple-choice questions (MCQs)

1. Which SQL keyword is used to retrieve unique values from a column?

A) SELECT DISTINCT

B) SELECT UNIQUE

C) SELECT DISTINCTROW

D) SELECT ONLY

Answer: A) SELECT DISTINCT

Explanation: The DISTINCT keyword removes duplicate rows from the result set.

2. What will be the result of the following query?

SELECT COUNT(\*)

FROM Employees

WHERE DepartmentID IS NULL;

A) Counts all rows in the Employees table

B) Counts rows where DepartmentID is NULL

C) Throws an error

D) Counts rows where DepartmentID is NOT NULL

Answer: B) Counts rows where DepartmentID is NULL

Explanation: The WHERE clause filters rows where the value in the DepartmentID column is NULL.

3. Which clause is used to sort the rows returned by a query?

A) ORDER BY

B) GROUP BY

C) SORT BY

D) HAVING

Answer: A) ORDER BY

Explanation: ORDER BY is used to sort rows either in ascending (default) or descending order.

4. Which of the following is NOT a type of JOIN in SQL?

A) INNER JOIN

B) FULL JOIN

C) OUTER JOIN

D) PARTIAL JOIN

Answer: D) PARTIAL JOIN

Explanation: PARTIAL JOIN is not a valid SQL keyword. The valid types are INNER, LEFT, RIGHT, and FULL.

5. What does the following query do?

SELECT AVG(Salary)

FROM Employees

GROUP BY DepartmentID;

A) Finds the average salary for each department

B) Finds the average salary of all employees

C) Finds the sum of salaries for each department

D) Throws an error

Answer: A) Finds the average salary for each department

Explanation: The GROUP BY clause groups rows by DepartmentID, and the AVG function calculates the average for each group.

6. Which SQL function is used to return the number of characters in a string?

A) LENGTH()

B) CHAR\_COUNT()

C) LEN()

D) CHAR\_LENGTH()

Answer: A) LENGTH() (MySQL) or C) LEN() (SQL Server)

Explanation:

MySQL uses LENGTH() to count characters.

SQL Server uses LEN().

7. What is the default order of sorting in the ORDER BY clause?

A) Ascending

B) Descending

C) Random

D) None

Answer: A) Ascending

Explanation: By default, ORDER BY arranges rows in ascending order.

8. What is the output of the following query?

SELECT COALESCE(NULL, 2, 3);

A) NULL

B) 2

C) 3

D) 5

Answer: B) 2

Explanation: COALESCE returns the first non-NULL value from the list.

9. Which operator is used to search for a specified pattern in a column?

A) LIKE

B) MATCH

C) IN

D) PATTERN

Answer: A) LIKE

Explanation: The LIKE operator is used with wildcards (% or \_) to search for patterns in strings.

10. What is the purpose of the GROUP BY clause in SQL?

A) To filter rows based on a condition

B) To aggregate data based on one or more columns

C) To sort data

D) To return only distinct values

Answer: B) To aggregate data based on one or more columns

Explanation: GROUP BY groups rows sharing a property and applies aggregate functions like COUNT, SUM, AVG, etc.

11. Which SQL statement is used to remove a table from a database?

A) DELETE TABLE

B) DROP TABLE

C) REMOVE TABLE

D) TRUNCATE TABLE

Answer: B) DROP TABLE

Explanation: DROP TABLE deletes the table and its structure permanently.

12. What is the difference between TRUNCATE and DELETE in SQL?

A) DELETE removes rows but keeps the structure; TRUNCATE removes both.

B) TRUNCATE removes rows faster than DELETE but keeps the structure.

C) Both remove rows but cannot be used on tables with foreign keys.

D) TRUNCATE allows filtering rows, whereas DELETE does not.

Answer: B) TRUNCATE removes rows faster than DELETE but keeps the structure.

Explanation:

TRUNCATE is faster because it doesn’t log individual row deletions.

Both keep the table structure intact.

13. Which SQL clause is used to limit the number of rows returned by a query?

A) LIMIT

B) ROWS

C) TOP

D) OFFSET

Answer: A) LIMIT (MySQL, PostgreSQL) or C) TOP (SQL Server)

Explanation:

MySQL and PostgreSQL use LIMIT.

SQL Server uses TOP.

14. What will this query do?

SELECT \*

FROM Orders

WHERE OrderDate BETWEEN '2024-01-01' AND '2024-12-31';

A) Selects rows with OrderDate in 2024

B) Selects rows with OrderDate before 2024

C) Selects rows with OrderDate in January 2024

D) Throws an error

Answer: A) Selects rows with OrderDate in 2024

Explanation: The BETWEEN operator filters rows with OrderDate inclusively within the specified range.

15. What does the following query return?

SELECT EmployeeID, COUNT(\*)

FROM Tasks

GROUP BY EmployeeID

HAVING COUNT(\*) > 5;

A) All employees with more than 5 tasks assigned

B) All employees with 5 or fewer tasks assigned

C) All employees and their task count

D) Throws an error

Answer: A) All employees with more than 5 tasks assigned

Explanation: The HAVING clause filters groups based on the result of aggregate functions like COUNT.

Website to practice: <https://datalemur.com/sql-tutorial>

<https://datalemur.com/sql-tutorial/advanced-sql-tutorial-intro>