Scenario-Based SQL Interview Questions

1. Write a Query to Find Duplicate Rows in a Table

Answer:

To find duplicate rows in a table, group by the columns that define a duplicate and use the `HAVING` clause to filter groups with more than one occurrence.

Example:

Suppose you have a table called `employees` with columns `first_name`, `last_name`, and `email`. To find duplicates based on `first_name` and `last_name`:

```
SELECT
    first_name,
    last_name,
    COUNT(*) AS duplicate_count
FROM
    employees
GROUP BY
    first_name,
    last_name
HAVING
    COUNT(*) > 1;
```

Explanation:

- **GROUP BY** groups rows with the same `first_name` and `last_name`.
- **COUNT(*)** counts the number of occurrences for each group.
- **HAVING COUNT(*) > 1** filters only those groups that have duplicates.

> **Tip:** Adjust the columns in the `GROUP BY` clause to match the definition of a duplicate in your specific table.

---

2. Explain the Difference Between INNER JOIN and OUTER JOIN with Examples

Answer:

INNER JOIN returns only the rows that have matching values in both tables.
OUTER JOIN returns all rows from one or both tables, filling in NULLs where there is no match.

Example:

Suppose you have two tables: `employees` and `departments`.

- `employees(employee_id, name, department_id)`
- `departments(department_id, department_name)`

INNER JOIN Example: Returns only employees who belong to a department.

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

OUTER JOIN Example (LEFT OUTER JOIN): Returns all employees, including those who do not belong to any department.

```
SELECT
    e.name,
    d.department_name
FROM
    employees e
LEFT OUTER JOIN
    departments d ON e.department_id = d.department_id;
```

Explanation:

- **INNER JOIN** includes only rows with matching `department_id` in both tables.
- **LEFT OUTER JOIN** includes all rows from `employees`, and fills `department_name` with NULL if there is no matching department.
- You can also use **RIGHT OUTER JOIN** or **FULL OUTER JOIN** to include all rows from the right table or both tables, respectively.

> **Tip:** Use `INNER JOIN` when you need only matching records, and `OUTER JOIN` when you want to include unmatched rows as well.

---

3. Write a Query to Fetch the Second-Highest Salary from an Employee Table

Answer:

To get the second-highest salary, you can use the `ORDER BY` and `LIMIT` clauses, or use a subquery to exclude the highest salary.

Example:

Suppose you have a table called `employees` with a column `salary`.

Using LIMIT/OFFSET (works in MySQL, PostgreSQL):

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

Using Subquery (works in most SQL dialects):

```
SELECT
    MAX(salary) AS second_highest_salary
FROM
    employees
WHERE
    salary &lt; (SELECT MAX(salary) FROM employees);
```

Explanation:

- The first query orders salaries in descending order, skips the highest, and fetches the next one.
- The second query finds the maximum salary that is less than the overall maximum, effectively giving the second-highest salary.
- **DISTINCT** ensures duplicate salaries are not counted multiple times.

> **Tip:** If there are multiple employees with the same second-highest salary, both queries will return that value. Adjust the query if you need all employees with the second-highest salary.

---

4. How Do You Use GROUP BY and HAVING Together? Provide an Example.

Answer:

The `GROUP BY` clause groups rows that have the same values in specified columns into summary rows. The `HAVING` clause is used to filter groups based on a condition, typically involving aggregate functions.

Example:

Suppose you have a table called `orders` with columns `customer_id` and `order_amount`. To find customers who have placed more than 2 orders:

```
SELECT
    customer_id,
    COUNT(*) AS total_orders
FROM
    orders
GROUP BY
    customer_id
```

```
HAVING
    COUNT(*) > 2;
```

Explanation:

- **GROUP BY** groups the rows by `customer_id`.
- **COUNT(*)** counts the number of orders for each customer.
- **HAVING COUNT(*) > 2** filters the groups to include only those customers with more than 2 orders.

> **Tip:** Use `HAVING` to filter groups after aggregation, while `WHERE` filters rows before grouping.

---

5. Write a Query to Find Employees Earning More Than Their Managers

Answer:

To find employees who earn more than their managers, you typically need a table where each employee has a `manager_id` referencing another employee's `employee_id`. You can use a self-join to compare each employee's salary with their manager's salary.

Example:

Suppose you have an `employees` table with columns `employee_id`, `name`, `salary`, and `manager_id`.

```
SELECT
    e.name AS employee_name,
    e.salary AS employee_salary,
    m.name AS manager_name,
    m.salary AS manager_salary
FROM
    employees e
JOIN
    employees m ON e.manager_id = m.employee_id
WHERE
    e.salary > m.salary;
```

Explanation:

- The table is joined to itself: `e` represents employees, `m` represents their managers.
- The `WHERE` clause filters for employees whose salary is greater than their manager's salary.

> **Tip:** Make sure `manager_id` is not NULL to avoid comparing employees without managers.

---

6. What is a Window Function in SQL? Provide Examples of ROW_NUMBER and RANK.

Answer:

A **window function** performs a calculation across a set of table rows that are somehow related to the current

row. Unlike aggregate functions, window functions do not collapse rows; they return a value for each row in the result set. Common window functions include `ROW_NUMBER`, `RANK`, `DENSE_RANK`, `SUM`, and `AVG` used with the `OVER` clause.

Example:
Suppose you have an `employees` table with columns `employee_id`, `name`, and `salary`.

ROW_NUMBER Example: Assigns a unique sequential number to each row within a partition, ordered by salary descending.

```
SELECT
    employee_id,
    name,
    salary,
    ROW_NUMBER() OVER (ORDER BY salary DESC) AS row_num
FROM
    employees;
```

RANK Example: Assigns a rank to each row within the result set, with gaps for ties.

```
SELECT
    employee_id,
    name,
    salary,
    RANK() OVER (ORDER BY salary DESC) AS salary_rank
FROM
    employees;
```

Explanation:

- **ROW_NUMBER()** gives each row a unique number based on the specified order.
- **RANK()** assigns the same rank to rows with equal values, but leaves gaps in the ranking sequence for ties.
- The `OVER (ORDER BY salary DESC)` clause defines the window for the function, ordering employees by salary from highest to lowest.

> **Tip:** Use window functions when you need to perform calculations across rows related to the current row, such as ranking, running totals, or moving averages.

---

7. Write a Query to Fetch the Top 3 Performing Products Based on Sales

Answer:
To find the top 3 performing products based on sales, you can aggregate the sales data by product, order the results by total sales in descending order, and then limit the output to the top 3 products.

Example:

Suppose you have a table called `sales` with columns `product_id` and `sale_amount`, and a `products` table with `product_id` and `product_name`.

```
SELECT
    p.product_name,
    SUM(s.sale_amount) AS total_sales
FROM
    sales s
JOIN
    products p ON s.product_id = p.product_id
GROUP BY
    p.product_name
ORDER BY
    total_sales DESC
LIMIT 3;
```

Explanation:

- **JOIN** combines the `sales` and `products` tables to get product names.
- **SUM(s.sale_amount)** calculates the total sales for each product.
- **GROUP BY** groups the results by product name.
- **ORDER BY total_sales DESC** sorts products from highest to lowest sales.
- **LIMIT 3** returns only the top 3 products.

> **Tip:** Adjust the `LIMIT` value to fetch a different number of top-performing products as needed.