50 Leetcode MySql Questions with Answers

Arun Prasanth Deenathayalan

Like Comment Share

Essential SQL: SELECT, WHERE, ORDER BY, and LIMIT

Problem 1: High-Earning Employees

Find all employees from the 'Sales' department who earn more than 60000.

```
SELECT name, salary
FROM Employees
WHERE department = 'Sales' AND salary > 60000;
```

This query selects the name and salary of employees from the 'Employees' table. It filters the results using the WHERE clause to include only those in the 'Sales' department with a salary greater than 60000.

Problem 2: Product Search by Category

Retrieve the names and prices of all products in the 'Electronics' category.

```
SELECT name, price
FROM Products
WHERE category = 'Electronics';
```

The query selects product names and prices from the 'Products' table, filtering by the 'Electronics' category using the WHERE clause.

Problem 3: Order by Price (Descending)

List all products, ordering them by price in descending order.

```
SELECT name, price
FROM Products
ORDER BY price DESC;
```

This query selects all product names and prices and then uses the ORDER BY clause to sort the results by price in descending order (highest price first).

Problem 4: Limiting Results

Find the top 5 highest-paid employees.

```
SELECT name, salary
FROM Employees
ORDER BY salary DESC
LIMIT 5;
```

The query selects employee names and salaries, orders the results by salary in descending order, and then limits the output to the top 5 employees.

Problem 5: Employees Not in Sales

Find all employees who are NOT in the 'Sales' or 'Marketing' departments.

```
SELECT name, department
FROM Employees
WHERE department NOT IN ('Sales', 'Marketing');
```

This query retrieves the names and departments of employees, excluding those in the 'Sales' or 'Marketing' departments using the NOT IN operator within the WHERE clause.

Problem 6: Products with Prices Between

List all products with a price between 20 and 100 (inclusive).

```
SELECT name, price
FROM Products
WHERE price BETWEEN 20 AND 100;
```

The query selects product names and prices from the 'Products' table where the price falls within the specified range (20 to 100), using the BETWEEN operator.

Problem 7: Find Products Containing 'Laptop'

Find the names of all products that contain the word 'Laptop'.

```
SELECT name
FROM Products
WHERE name LIKE '%Laptop%';
```

This SQL query uses the `LIKE` operator with wildcards (`%`) to find product names that contain the substring 'Laptop'.

Problem 8: Unique Departments

List all distinct departments from the Employees table.

```
SELECT DISTINCT department
FROM Employees;
```

This query uses the `DISTINCT` keyword to retrieve only the unique department names from the `Employees` table, eliminating any duplicates in the output.

Problem 9: Employees with Salaries Less Than or Equal To

Find employees with salaries less than or equal to 55000.

```
SELECT name, salary
FROM Employees
WHERE salary <= 55000;
```

The query selects the name and salary of all employees whose salary is less than or equal to 55000.

Problem 10: Products Starting with 'A'

Find the names of all products that start with the letter 'A'.

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

This query utilizes the `LIKE` operator to find product names that begin with 'A'. The '%' wildcard represents any characters following 'A'.

Navigating Data: Joins, Unions, and Set Operations

Problem 11: Customers and Their Orders (Including Those with No Orders)

Retrieve all customers and their orders, including customers who have not placed any orders.

```
SELECT Customers.name, Orders.order_id
FROM Customers
LEFT JOIN Orders ON Customers.id = Orders.customer_id;
```

This query uses a LEFT JOIN to include all customers from the `Customers` table, regardless of whether they have corresponding entries in the `Orders` table. If a customer has no orders, the `order_id` will be NULL.

Problem 12: Orders and Customer Names

Retrieve all orders along with the name of the customer who placed the order.

```
SELECT Orders.order_id, Customers.name
FROM Orders
INNER JOIN Customers ON Orders.customer_id = Customers.id;
```

This query uses an INNER JOIN to combine rows from the `Orders` and `Customers` tables where the `customer_id` in `Orders` matches the `id` in `Customers`. Only orders with a valid customer are returned.

Problem 13: All Employees and Their Department (Including Those with No Department)

Retrieve all employees and their department names, including employees who are not assigned to any department.

```
SELECT Employees.name, Departments.department_name
FROM Employees
LEFT JOIN Departments ON Employees.department_id = Departments.id;
```

A LEFT JOIN is used to ensure all employees are listed, even if they don't have a department assigned. The `department_name` will be NULL for employees without a department.

Problem 14: Combining Customer and Employee Names

Combine a list of customer names and employee names into a single result set.

```
SELECT name FROM Customers
UNION
SELECT name FROM Employees;
```

This query uses the UNION operator to combine the results of two SELECT statements. `UNION` removes duplicate names from the final result. If you want to keep duplicates, use `UNION ALL`.

Problem 15: Customers with and without Orders

Create a list of all customers. If they have placed an order, show the order ID; otherwise, show 'No Order'.

```
SELECT c.name, COALESCE(CAST(o.order_id AS VARCHAR), 'No Order') AS order_id
FROM Customers c
LEFT JOIN Orders o ON c.id = o.customer_id;
```

This query uses a LEFT JOIN to include all customers. `COALESCE` is used to display 'No Order' if the customer has no corresponding order.

Problem 16: Finding Common Products

Find the names of products that are present in both the 'Electronics' and 'Home Goods' categories.

```
SELECT name
FROM Products
WHERE category = 'Electronics'
INTERSECT
SELECT name
FROM Products
WHERE category = 'Home Goods';
```

This query uses the `INTERSECT` operator to find the common product names between the two categories.

Problem 17: Products in One Category but Not Another

Find the names of products that are in the 'Electronics' category but not in the 'Home Goods' category.

```
SELECT name
FROM Products
WHERE category = 'Electronics'
EXCEPT
SELECT name
FROM Products
WHERE category = 'Home Goods';
```

The `EXCEPT` operator returns product names from the 'Electronics' category that are not present in the 'Home Goods' category.

Problem 18: Orders Placed by Specific Customers

Retrieve all orders placed by customers with IDs 1, 2, or 3, along with the customer's name.

```
SELECT Orders.order_id, Customers.name
FROM Orders
INNER JOIN Customers ON Orders.customer_id = Customers.id
WHERE Customers.id IN (1, 2, 3);
```

An INNER JOIN is used to retrieve orders and customer names, filtering by customer ID using the `IN` operator within the `WHERE` clause.

Problem 19: Union with Different Columns

Combine a list of customer IDs and order IDs into a single column (requires casting to a common data type).

```
SELECT CAST(id AS VARCHAR) AS identifier FROM Customers
UNION ALL
SELECT CAST(order_id AS VARCHAR) AS identifier FROM Orders;
```

This query uses `UNION ALL` to combine customer IDs and order IDs. Since the data types might be different, we cast them to a common VARCHAR type.

Problem 20: Joining Three Tables

Retrieve customer name, order ID, and product name for all orders.

```
SELECT c.name AS customer_name, o.order_id, p.name AS product_name
FROM Customers c
INNER JOIN Orders o ON c.id = o.customer_id
INNER JOIN Products p ON o.product_id = p.id;
```

This query joins three tables (`Customers`, `Orders`, and `Products`) to retrieve related information across all three. It uses INNER JOINs to combine rows based on matching IDs.

Summarizing Data: Aggregate Functions and GROUP BY

Problem 21: Total Sales

Calculate the total sales amount from the Sales table.

```
SELECT SUM(amount) AS total_sales
FROM Sales;
```

This query uses the SUM aggregate function to calculate the sum of the 'amount' column in the Sales table, providing the total sales.

Problem 22: Average Salary by Department

Calculate the average salary for each department in the Employees table.

```
SELECT department_id, AVG(salary) AS average_salary
FROM Employees
GROUP BY department_id;
```

This query uses the AVG aggregate function to calculate the average salary, grouping the results by department_id using the GROUP BY clause.

Problem 23: Number of Products in Each Category

Find the number of products in each category in the Products table.

```
SELECT category, COUNT(*) AS number_of_products
FROM Products
GROUP BY category;
```

This query employs the COUNT aggregate function to count the number of products, grouping the results by category.

Problem 24: Minimum and Maximum Salary

Find the minimum and maximum salary in the Employees table.

```
SELECT MIN(salary) AS minimum_salary, MAX(salary) AS maximum_salary
FROM Employees;
```

This query uses the MIN and MAX aggregate functions to find the lowest and highest salaries, respectively.

Problem 25: Total Sales for Each Category

Calculate the total sales for each product category, only including categories with total sales over 10000.

```
SELECT category, SUM(amount) AS total_sales
FROM Sales
GROUP BY category
HAVING SUM(amount) > 10000;
```

This query calculates total sales per category and uses the HAVING clause to filter out categories with total sales less than or equal to 10000.

Problem 26: Departments with More Than 5 Employees

Find all department IDs that have more than 5 employees.

```
SELECT department_id
FROM Employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

This query groups employees by department and then filters those groups to only include departments with more than 5 employees using the HAVING clause.

Problem 27: Average Order Value Above Threshold

Calculate the average order value for each customer, only including customers with an average order value greater than 50.

```
SELECT customer_id, AVG(amount) AS average_order_value
FROM Orders
GROUP BY customer_id
```

```
HAVING AVG(amount) > 50;
```

The query computes average order values per customer and uses the HAVING clause to filter for customers whose average order value exceeds 50.

Problem 28: Count of Orders Placed on Each Day

Find the number of orders placed on each day.

```
SELECT order_date, COUNT(*) AS number_of_orders
FROM Orders
GROUP BY order_date;
```

This SQL query groups orders by the order date and counts the number of orders in each group to determine the daily order count.

Problem 29: Sum of Salaries for Each Department and Gender

Calculate the sum of salaries for each department and gender combination.

```
SELECT department_id, gender, SUM(salary) AS total_salary
FROM Employees
GROUP BY department_id, gender;
```

This query uses GROUP BY with multiple columns (department_id and gender) to calculate the total salary for each unique combination.

Problem 30: Categories with Average Product Price Below 50

Identify product categories where the average product price is less than 50.

```
SELECT category, AVG(price) AS average_price
FROM Products
GROUP BY category
HAVING AVG(price) < 50;
```

This query calculates the average price for each category and then filters the results to show only those categories where the average price is below 50, using the HAVING clause.

Advanced Querying: Subqueries and Common Table Expressions (CTEs)

Problem 31: Employees with Salary Above Department Average (Subquery)

Find the names of employees who have a salary greater than the average salary in their respective departments.

```
SELECT e.name
FROM Employees e
WHERE e.salary > (SELECT AVG(salary) FROM Employees WHERE department_id =
e.department_id);
```

This query uses a correlated subquery in the WHERE clause. For each employee, the subquery calculates the average salary of their department. The employee's salary is then compared to this average.

Problem 32: Departments with Highest Salary (CTE)

Find the names of departments where at least one employee has the highest salary.

```
WITH MaxSalary AS (
    SELECT MAX(salary) AS max_salary
    FROM Employees
)
SELECT DISTINCT d.department_name
FROM Employees e
JOIN Departments d ON e.department_id = d.id
JOIN MaxSalary ms ON e.salary = ms.max_salary;
```

This solution utilizes a CTE called `MaxSalary` to determine the maximum salary across all employees. It then joins this CTE with the `Employees` and `Departments` tables to find departments with employees earning the maximum salary.

Problem 33: Employees in Departments with More Than 5 Employees (Subquery)

List the names of employees who work in departments that have more than 5 employees.

```
SELECT name
FROM Employees
WHERE department_id IN (SELECT department_id FROM Employees GROUP BY
department_id HAVING COUNT(*) > 5);
```

This query uses a subquery in the WHERE clause with the `IN` operator. The subquery identifies department IDs with more than 5 employees, and the main query selects the names of employees belonging to those departments.

Problem 34: Top 3 Salaries in Each Department (CTE)

Find the top 3 highest salaries in each department.

```
WITH RankedSalaries AS (
    SELECT department_id, name, salary,
           DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary
DESC) as salary_rank
    FROM Employees
)
```

```
SELECT department_id, name, salary
FROM RankedSalaries
WHERE salary_rank <= 3;
```

This query uses a CTE called `RankedSalaries` to assign a rank to each employee's salary within their department. It then selects employees whose salary rank is less than or equal to 3.

Problem 35: Departments with No Employees (Subquery)

Find the names of departments that have no employees.

```
SELECT department_name
FROM Departments
WHERE id NOT IN (SELECT DISTINCT department_id FROM Employees WHERE
department_id IS NOT NULL);
```

This query uses a subquery with the `NOT IN` operator to find departments whose IDs are not present in the `department_id` column of the `Employees` table.

Problem 36: Average Salary Compared to Overall Average (CTE)

Find employees whose salary is higher than the overall average salary.

```
WITH AvgSalary AS (
    SELECT AVG(salary) AS avg_salary FROM Employees
)
SELECT name, salary
FROM Employees, AvgSalary
WHERE salary > AvgSalary.avg_salary;
```

A CTE, `AvgSalary`, pre-calculates the average salary. The main query then selects employees whose salaries exceed this average. This is a non-correlated subquery effectively used as a CTE.

Problem 37: Employees Reporting to Managers Earning More (Subquery)

Find employees who report to managers who earn more than 70000.

```
SELECT e.name
FROM Employees e
WHERE EXISTS (SELECT 1 FROM Employees m WHERE m.id = e.manager_id AND
m.salary > 70000);
```

This query uses the `EXISTS` operator with a correlated subquery to check if there is a manager with a salary greater than 70000 for each employee.

Problem 38: Total Salary by Department (CTE and Subquery)

Calculate the total salary for each department and then select the departments where the total salary is greater than the average total salary across all departments.

```
WITH DepartmentSalaries AS (
    SELECT department_id, SUM(salary) AS total_salary
    FROM Employees
    GROUP BY department_id
)
SELECT d.department_name
FROM Departments d
JOIN DepartmentSalaries ds ON d.id = ds.department_id
WHERE ds.total_salary > (SELECT AVG(total_salary) FROM DepartmentSalaries);
```

The CTE `DepartmentSalaries` calculates the total salary per department. The main query then filters these departments based on whether their total salary exceeds the average total salary calculated using a subquery on the CTE.

Problem 39: Finding Duplicate Salaries (Subquery)

List all salaries that appear more than once in the Employees table.

```
SELECT salary
FROM Employees
GROUP BY salary
HAVING COUNT(*) > 1;
```

A subquery groups the salaries and returns only those salaries where the count is greater than 1, indicating duplicates.

Problem 40: Departments with Salary Greater Than X

Find all departments that have at least one employee whose salary is greater than X.

```
SELECT DISTINCT department_id
FROM Employees
WHERE salary > X;
```

The query selects the distinct department IDs that have at least one employee with a salary greater than X.

Mastering SQL: Window Functions and Complex Scenarios

Problem 41: Rank Students by Score

Rank students by their total score, allowing for ties. Use the `RANK()` window function.

```
SELECT name, score, RANK() OVER (ORDER BY score DESC) AS rank_number
FROM Students;
```

This query ranks students based on their scores. The `RANK()` function assigns the same rank to students with the same score, resulting in rank gaps.

Problem 42: Calculate Running Total

Calculate the running total of sales over time.

```
SELECT sale_date, sale_amount, SUM(sale_amount) OVER (ORDER BY sale_date) AS
running_total
FROM Sales;
```

The query calculates a running total of sales amounts, ordered by the sale date. The `SUM() OVER (ORDER BY)` syntax cumulatively sums the `sale_amount` for each row.

Problem 43: Compare Current and Previous Row Values

Compare the current salary with the previous salary for each employee.

```
SELECT name, salary, LAG(salary, 1, 0) OVER (ORDER BY hire_date) AS
previous_salary
FROM Employees;
```

This query uses the `LAG()` function to access the salary from the previous row (ordered by hire date). The third argument of `LAG()` provides a default value (0) if there is no previous row.

Problem 44: Calculate the Difference Between Two Dates

Calculate the difference in days between each order date and the previous order date for each customer.

```
SELECT customer_id, order_date, DATEDIFF(order_date, LAG(order_date, 1,
order_date) OVER (PARTITION BY customer_id ORDER BY order_date)) AS
days_since_last_order
FROM Orders;
```

This query partitions by customer ID and calculates the difference between consecutive order dates using `DATEDIFF` and `LAG`. If there is no previous date the current `order_date` is used.

Problem 45: Conditional Aggregation using CASE

Count the number of orders above and below an average order value.

```
SELECT
    SUM(CASE WHEN amount > (SELECT AVG(amount) FROM Orders) THEN 1 ELSE 0
END) AS orders_above_average,
    SUM(CASE WHEN amount <= (SELECT AVG(amount) FROM Orders) THEN 1 ELSE 0
END) AS orders_below_average
FROM Orders;
```

This query uses `CASE` statements inside `SUM` to conditionally count orders based on whether they are above or below the average order value.

Problem 46: Divide into Groups Using NTILE

Divide employees into 4 groups based on their salary.

```
SELECT name, salary, NTILE(4) OVER (ORDER BY salary DESC) AS salary_quartile
FROM Employees;
```

The `NTILE(4)` function divides the employees into four groups (quartiles) based on their salary, with the highest salaries in the first quartile.

Problem 47: Find the Second Highest Salary

Find the second highest salary in the Employees table using `DENSE_RANK()`.

```
WITH SalaryRanked AS (
    SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rank_num
    FROM Employees
)
SELECT salary
FROM SalaryRanked
WHERE rank_num = 2
LIMIT 1;
```

This query ranks salaries using `DENSE_RANK()` and then selects the salary with rank 2. `DENSE_RANK()` assigns consecutive ranks without gaps.

Problem 48: Concatenate Strings

Combine the first name and last name of employees into a full name.

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM Employees;
```

This query uses the `CONCAT()` function to combine the first name and last name, separated by a space, into a single full name column.

Problem 49: Extract Year and Month

Extract the year and month from the order date.

```
SELECT order_id, YEAR(order_date) AS order_year, MONTH(order_date) AS
order_month
FROM Orders;
```

This query uses the `YEAR()` and `MONTH()` functions to extract the year and month components from the `order_date` column.

Problem 50: Get the Length of Strings

Find the average length of product names for each category.

```
SELECT category, AVG(LENGTH(name)) AS average_name_length
FROM Products
GROUP BY category;
```

This query calculates the average length of product names for each category. It uses the `LENGTH()` function to determine the length of each name.

Thank you for reading!
If you found this document helpful,
feel free to share it with others.

http://www.linkedin.com/in/arun-prasanth-deenathayalan-b143351ab