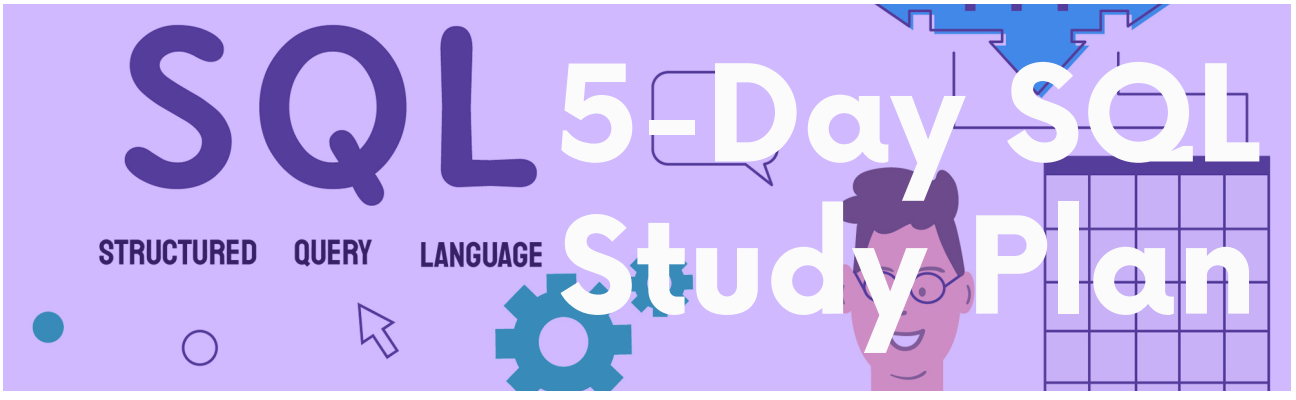

5 Day SQL Study Plan

Have an upcoming interview? Or looking to get out of the dreaded tutorial hell?

- 5 Short Lessons
- 5 Case Studies

Day 5: Advanced SQL Queries with Common Table Expressions (CTEs)

Objective

Learn about Common Table Expressions (CTEs)

Learn about Common Table Expressions (CTEs), a powerful feature in SQL that allows us to create temporary named result sets. Understand how to use CTEs to write complex and efficient SQL queries. Explore concepts such as recursive CTEs, multiple CTEs, and inline views.

Part 1: Introduction to Common Table Expressions (CTEs)

In this part, we will explore Common Table Expressions (CTEs), a powerful feature in SQL that allows us to create temporary named result sets. CTEs provide a way to break down complex queries into smaller, more manageable parts, improving query readability and maintainability. We will learn the syntax of CTEs and understand how to use them to write efficient and concise SQL queries.

```
WITH cte_name (column1, column2, ...)
AS (
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition
)
SELECT column1, column2, ...
FROM cte_name;
```

Part 2: Recursive CTEs

Recursive CTEs are a fascinating aspect of CTEs that enable us to perform hierarchical or graph-based queries. In this part, we will delve into the world of recursive CTEs and learn how they can be used to traverse tree structures, find connected components in graphs, and perform other iterative operations. We will understand the recursive CTE syntax, including the anchor member and recursive member, and see practical examples of their usage.

```
WITH RECURSIVE cte_name (column1, column2, ...)
AS (
  -- Anchor member
```

```
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition
  UNION ALL
  -- Recursive member
  SELECT column1, column2, ...
  FROM cte_name
  WHERE condition
)
SELECT column1, column2, ...
FROM cte_name;
```

Part 3: Multiple CTEs

Multiple CTEs allow us to define and use multiple CTEs within a single SQL statement. In this part, we will explore the concept of multiple CTEs and understand how they can help break down complex queries into smaller logical parts. We will learn how to define and reference multiple CTEs in a query and discover their usefulness in improving query organization and reusability.

```
WITH cte1 (column1, column2, ...)
AS (
  SELECT column1, column2, ...
  FROM table1
),
cte2 (column3, column4, ...)
AS (
  SELECT column3, column4, ...
  FROM table2
)
SELECT column1, column2, column3, column4, ...
FROM cte1
JOIN cte2 ON condition;
```

Part 4: Inline Views

Inline views, also known as derived tables, provide a powerful way to treat the result of a subquery as a temporary table within a SQL statement. In this part, we will explore the concept of inline views and understand how they can be used to manipulate and reference intermediate result sets in the main query. We will learn the syntax of inline views and see practical examples of their application.

```
SELECT column1, column2, ...
FROM (
```

```
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition
) AS inline_view_name;
```

Part 5: Combining CTEs and Inline Views

Combining CTEs and inline views allows us to create even more complex SQL queries. In this part, we will learn how to leverage the power of both CTEs and inline views together to break down complex queries, create temporary result sets, and perform advanced analysis. We will explore examples demonstrating how CTEs and inline views can work harmoniously to enhance query readability and maintainability.

```
WITH cte_name (column1, column2, ...)
AS (
    SELECT column1, column2, ...
    FROM table1
),
inline_view_name AS (
    SELECT column3, column4, ...
    FROM table2
    WHERE condition
)
SELECT column1, column2, column3, column4, ...
FROM cte_name
JOIN inline_view_name ON condition;
```

Case Study: Employee Management System

Questions

1. Calculate the total salary expenditure for each department and display the departments in descending order of the total salary expenditure.
2. Retrieve the employees who have at least two subordinates.
3. Calculate the average salary for each department, considering only employees with a salary greater than the department average.
4. Find the employees who have the highest salary in their respective departments.
5. Calculate the running total of salaries for each department, ordered by department ID and employee name.

Complete the case study here

Calculate the total salary expenditure for each department and display the departments in descending order of the total salary expenditure.

```
WITH department_salary AS
(
SELECT
    department_id,
    SUM(salary) as total_salary
FROM employees
GROUP BY department_id
)
SELECT
     d1.department_name,
     d2.total_salary as total_salary_expenditure
FROM departments d1
JOIN department_salary d2
ON d1.department_id = d2.department_id
```

Sure, let's break down the SQL query step by step:

Step 1:

```
WITH department_salary AS
(
  SELECT
      department_id,
      SUM(salary) as total_salary
  FROM employees
  GROUP BY department_id
)
```

In this step, we are using a Common Table Expression (CTE) named department_salary. This CTE calculates the total salary expenditure for each department. It does so by selecting the department_id from the employees table and using the SUM function to sum up the salary for each department. The result is grouped by department_id, so you get the total salary for each department.

Step 2:

```
SELECT
     d1.department_name,
     d2.total_salary as total_salary_expenditure
FROM departments d1
```

```
JOIN department_salary d2
ON d1.department_id = d2.department_id
```

In this step, we are querying the result of the CTE department_salary. We want to retrieve the department names and their respective total salary expenditures.

- We select d1.department_name from the departments table as the department name.
- We select d2.total_salary from the CTE result department_salary as the total salary expenditure.
- We use a JOIN clause to join the departments table (d1) with the CTE result (d2) using the department_id as the common column.

The result is a list of department names along with their total salary expenditures, which is calculated using the CTE. This gives you the total salary expenditure for each department, and it is displayed as a result set.

Another solution

```
SELECT
  d.department_name,
  SUM(e.salary) AS total_salary_expenditure
FROM
  departments d
LEFT JOIN
  employees e ON d.department_id = e.department_id
GROUP BY
  d.department_name
ORDER BY
  total_salary_expenditure DESC;
```

Your SQL query is written to retrieve the total salary expenditure for each department, sorting the results in descending order of total salary expenditure. Here's an explanation of your query:

1. You are selecting two columns in your result set:
   - department_name from the departments table.
   - The sum of salary from the employees table, aliased as total_salary_expenditure.
2. You are using a LEFT JOIN to combine data from the departments and employees tables based on the department_id column. This allows you to associate each department with its employees.

3. You are grouping the results by department_name using the GROUP BY clause. This groups all the employees within each department together.
4. Finally, you are ordering the results in descending order of total_salary_expenditure using the ORDER BY clause.

The result of this query will give you a list of departments along with the total salary expenditure for each department, sorted from the highest expenditure to the lowest.

Query #1

```
WITH department_salary AS
(
SELECT
    department_id,
    SUM(salary) as total_salary
FROM employees
GROUP BY department_id
)
SELECT
     d1.department_name,
     d2.total_salary as total_salary_expenditure
FROM departments d1
JOIN department_salary d2
ON d1.department_id = d2.department_id
ORDER BY total_salary_expenditure DESC;
```

department_name	total_salary_expenditure
Sales	15200.00
Finance	11500.00
IT	9900.00
Marketing	8800.00
HR	5000.00

Query #2

```
SELECT
   d.department_name,
   SUM(e.salary) AS total_salary_expenditure
FROM
   departments d
```

```
  LEFT JOIN
    employees e ON d.department_id = e.department_id
  GROUP BY
    d.department_name
  ORDER BY
    total_salary_expenditure DESC;
```

department_name	total_salary_expenditure
Sales	15200.00
Finance	11500.00
IT	9900.00
Marketing	8800.00
HR	5000.00

Question 2: Retrieve the employees who have at least two subordinates.

```
-- Question 2: Retrieve the employees who have at least two subordinates.
-- Query 1
SELECT
  e1.employee_name AS employee_with_subordinates,
  COUNT(*) AS num_subordinates
FROM
  employees e1
INNER JOIN
  employees e2 ON e1.employee_id = e2.manager_id
GROUP BY
  e1.employee_name
HAVING
  COUNT(*) >= 2;


-- --- -----
--  Query 2

WITH subordinate_count AS (
  SELECT manager_id, COUNT(*) AS num_subordinates
  FROM employees
  GROUP BY manager_id
)
SELECT e.employee_name, sc.num_subordinates
FROM employees e
```

```
JOIN subordinate_count sc ON e.employee_id = sc.manager_id
WHERE sc.num_subordinates >= 2;
```

Query 1

- In this query, you are joining the employees table with itself using aliases e1 and e2 to create a self-join.
- You are counting the number of subordinates for each employee (employee_with_subordinates) by counting the matching rows in the self-join.
- The GROUP BY clause groups the results by the names of employees.
- The HAVING clause filters the results to only include employees with at least two subordinates (COUNT(*) >= 2).

Query2

- In this query, you first create a Common Table Expression (CTE) named subordinate_count that calculates the number of subordinates for each manager.
- Then, you select the employee_name and the num_subordinates from the employees table, joining it with the subordinate_count CTE on the employee_id and manager_id.
- The WHERE clause filters the results to only include employees with at least two subordinates (sc.num_subordinates >= 2).

Both queries achieve the same result of identifying employees with at least two subordinates, but they use slightly different approaches.

Question 3: Calculate the average salary for each department, considering only employees with a salary greater than the department average.

```
-- Question 3: Calculate the average salary for each department, considering only employees with a salary greater than the department average.

WITH DepartmentAvgSalary AS (
  SELECT
    d.department_id,
    AVG(e.salary) AS avg_salary
  FROM
    departments d
  LEFT JOIN
    employees e ON d.department_id = e.department_id
  GROUP BY
```

```
      d.department_id
)
SELECT
   d.department_name,
   ROUND(AVG(e.salary),2) AS average_salary
FROM
   departments d
LEFT JOIN
   employees e ON d.department_id = e.department_id
JOIN
   DepartmentAvgSalary a ON d.department_id = a.department_id
WHERE
   e.salary > a.avg_salary
GROUP BY
   d.department_name;


-- ---

WITH department_avg AS (
   SELECT
      department_id,
      AVG(salary) AS avg_salary
   FROM employees
   GROUP BY department_id
)
SELECT
     d.department_name,
     ROUND(AVG(e.salary),2) AS avg_salary
FROM employees e
JOIN department_avg da ON e.department_id = da.department_id
JOIN departments d ON e.department_id = d.department_id
WHERE e.salary > da.avg_salary
GROUP BY  d.department_name;


---
**Query #7**

   WITH DepartmentAvgSalary AS (
      SELECT
         d.department_id,
         AVG(e.salary) AS avg_salary
      FROM
         departments d
      LEFT JOIN
         employees e ON d.department_id = e.department_id
      GROUP BY
```

```
        d.department_id
    )
    SELECT
        d.department_name,
        ROUND(AVG(e.salary),2) AS average_salary
    FROM
        departments d
    LEFT JOIN
        employees e ON d.department_id = e.department_id
    JOIN
        DepartmentAvgSalary a ON d.department_id = a.department_id
    WHERE
        e.salary > a.avg_salary
    GROUP BY
        d.department_name;
```

department_name	average_salary
Finance	6000.00
Sales	5350.00
Marketing	4800.00
IT	5200.00

Query #8

```
    WITH department_avg AS (
        SELECT department_id, AVG(salary) AS avg_salary
        FROM employees
        GROUP BY department_id
    )
    SELECT  d.department_name, ROUND(AVG(e.salary),2) AS avg_salary
    FROM employees e
    JOIN department_avg da ON e.department_id = da.department_id
    JOIN departments d ON e.department_id = d.department_id
    WHERE e.salary > da.avg_salary
    GROUP BY  d.department_name;
```

department_name	avg_salary
Finance	6000.00
Sales	5350.00
Marketing	4800.00
IT	5200.00

Question 4: Find the employees who have the highest salary in their respective departments.

V

Schema (MySQL v8.0)

Query #1

```
WITH max_salary_per_department AS
(SELECT department_id, MAX(salary) as max_salary
FROM employees
GROUP BY department_id)
SELECT
    employee_name,
    salary,
    department_name
FROM employees e
JOIN max_salary_per_department d1
ON e.department_id = d1.department_id
JOIN departments d2
ON d1.department_id = d2.department_id
WHERE e.salary = d1.max_salary;
```

employee_name	salary	department_name
John Doe	5000.00	HR
Jane Smith	6000.00	Finance
Emily Brown	5500.00	Sales
Sarah Davis	4800.00	Marketing
Robert Anderson	5200.00	IT

Query #2

```
SELECT
    d.department_name,
    e.employee_name,
    e.salary
FROM
    departments d
LEFT JOIN
    employees e ON d.department_id = e.department_id
```

```
  WHERE
    (e.department_id, e.salary) IN (
      SELECT
        department_id,
        MAX(salary)
      FROM
        employees
      GROUP BY
        department_id
    );
```

department_name	employee_name	salary
HR	John Doe	5000.00
Finance	Jane Smith	6000.00
Sales	Emily Brown	5500.00
Marketing	Sarah Davis	4800.00
IT	Robert Anderson	5200.00

Question 5: Calculate the running total of salaries for each department, ordered by department ID and employee name.

```
-- Question 5: Calculate the running total of salaries for each department, ordered by department ID and employee name.

WITH running_total AS (
  SELECT department_id, employee_name, salary,
      SUM(salary) OVER (PARTITION BY department_id ORDER BY employee_name) AS total_salary
  FROM employees
)
SELECT department_id, employee_name, salary, total_salary
FROM running_total
ORDER BY department_id, employee_name;
```

```
WITH DepartmentSalaries AS (
  SELECT
    d.department_id,
    e.employee_name,
    e.salary,
    ROW_NUMBER() OVER (PARTITION BY d.department_id ORDER BY e.employee_name) AS row_num
  FROM
    departments d
```

```
    LEFT JOIN
        employees e ON d.department_id = e.department_id
)
SELECT
    d.department_name,
    ds.employee_name,
    ds.salary,
    SUM(ds.salary) OVER (PARTITION BY ds.department_id ORDER BY ds.row_num) AS running_total
FROM
    departments d
LEFT JOIN
    DepartmentSalaries ds ON d.department_id = ds.department_id
ORDER BY
    ds.department_id,
    ds.row_num;
```

Output

Schema (MySQL v8.0)

Query #1

```
    WITH running_total AS (
        SELECT department_id, employee_name, salary,
            SUM(salary) OVER (PARTITION BY department_id ORDER BY employee_name) AS total_salary
        FROM employees
    )
    SELECT department_id, employee_name, salary, total_salary
    FROM running_total
    ORDER BY department_id, employee_name;
```

department_id	employee_name	salary	total_salary
1	John Doe	5000.00	5000.00
2	Jane Smith	6000.00	6000.00
2	Olivia Harris	5500.00	11500.00
3	Daniel Turner	5200.00	5200.00
3	Emily Brown	5500.00	10700.00
3	Michael Johnson	4500.00	15200.00
4	David Wilson	4000.00	4000.00
4	Sarah Davis	4800.00	8800.00
5	Laura Clark	4700.00	4700.00
5	Robert Anderson	5200.00	9900.00

Query #2

```
WITH DepartmentSalaries AS (
    SELECT
        d.department_id,
        e.employee_name,
        e.salary,
        ROW_NUMBER() OVER (PARTITION BY d.department_id ORDER BY e.employee_name) AS row_num
    FROM
        departments d
    LEFT JOIN
        employees e ON d.department_id = e.department_id
)
SELECT
    d.department_name,
    ds.employee_name,
    ds.salary,
    SUM(ds.salary) OVER (PARTITION BY ds.department_id ORDER BY ds.row_num) AS running_total
FROM
    departments d
LEFT JOIN
    DepartmentSalaries ds ON d.department_id = ds.department_id
ORDER BY
    ds.department_id,
    ds.row_num;
```

department_name	employee_name	salary	running_total
HR	John Doe	5000.00	5000.00
Finance	Jane Smith	6000.00	6000.00
Finance	Olivia Harris	5500.00	11500.00
Sales	Daniel Turner	5200.00	5200.00
Sales	Emily Brown	5500.00	10700.00
Sales	Michael Johnson	4500.00	15200.00
Marketing	David Wilson	4000.00	4000.00
Marketing	Sarah Davis	4800.00	8800.00
IT	Laura Clark	4700.00	4700.00
IT	Robert Anderson	5200.00	9900.00
