SQL

Common Table Expressions (CTE)

https://www.linkedin.com/in/nivetha-anand/

What is a CTE and what does it do?

- CTE is a temporary result set that is defined and used within the execution scope of SELECT, INSERT, UPDATE or DELETE.
- CTE's are defined using **WITH** clause.
- A CTE can be referenced multiple times within the main SQL query.

https://www.linkedin.com/in/nivetha-anand/

Why use CTEs in SQL?

CTEs simplify query writing and maintenance by:

- Breaking down complex queries into smaller, reusable components.
- Improving code readability and modularity.
- Enabling recursive operations for hierarchical data.

03

Syntax

```
WITH cte_name AS (
 SELECT query
)
SELECT *
FROM cte_name;
```

https://www.linkedin.com/in/nivetha-anand/

04

🔹 Easy: Find Employees with Salary Greater Than 50,000

```
WITH HighSalary AS (
    SELECT name, salary
    FROM employees
    WHERE salary > 50000
)
SELECT * FROM HighSalary;
```

https://www.linkedin.com/in/nivetha-anand/

Explanation

- The HighSalary CTE filters employees with salaries above $50,000.
- The final SELECT statement retrieves the filtered results.

https://www.linkedin.com/in/nivetha-anand/

🔶 Medium: Find Departments with Average Salary Above 60,000

📝 **Problem: Write a query using a CTE to find departments where the average salary is above $60,000.**

```
WITH DeptSalary AS (
    SELECT department, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department
)
SELECT department
FROM DeptSalary
WHERE avg_salary > 60000;
```

https://www.linkedin.com/in/nivetha-anand/

07

Explanation:

- The DeptSalary CTE calculates the average salary per department.
- The final query filters departments where the average salary exceeds $60,000.

https://www.linkedin.com/in/nivetha-anand/

08

🔥 Hard: Find Employees in a Recursive Hierarchy (Manager-Employee Structure)

📝 **Problem: Given an employees table with id, name, and manager_id, find all employees reporting under a specific manager (e.g., manager_id = 1).**

```sql
WITH RECURSIVE EmployeeHierarchy AS (
   SELECT id, name, manager_id
   FROM employees
   WHERE manager_id = 1

   UNION ALL

   SELECT e.id, e.name, e.manager_id
   FROM employees e
   JOIN EmployeeHierarchy eh
   ON e.manager_id = eh.id
)
SELECT * FROM EmployeeHierarchy;
```

https://www.linkedin.com/in/nivetha-anand/

Explanation:

- The base case selects employees directly reporting to manager 1.
- The recursive step fetches employees who report to those managers, forming a hierarchy.
- This continues until all indirect reports are found.

https://www.linkedin.com/in/nivetha-anand/

Was this post helpful to you?

Please like, share, save, comment and repost!

https://www.linkedin.com/in/nivetha-anand/ →