#PostgreSQL#

Section 8. Common Table Expressions
PostgreSQL CTE – introduce you to PostgreSQL common table expressions or CTEs.
Recursive query using CTEs – discuss the recursive query and learn how to apply it in various contexts.

Can You Teach Me above all the topics with Realtime example (create a simple table and insert some values) and show output in Table Format and Also Explain Each topic why this needed, what happen, like syntax and so on.

PostgreSQL Common Table Expressions (CTEs)

What is a CTE?

A Common Table Expression (CTE) is a temporary result set defined within the execution scope of a single `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement. CTEs simplify complex queries, improve readability, and enable recursion.

Syntax for CTE:

```sql


WITH cte_name (optional_column_list) AS (
    cte_query
)
SELECT ... FROM cte_name;
```

- `WITH`: Introduces the CTE.
- `cte_name`: The name of the CTE.
- `cte_query`: The query that defines the temporary result set.

Example 1: Basic CTE

Problem:

We have a `sales` table, and we want to calculate the total revenue for each salesperson and then find the top salesperson.

Steps:

1. Create a `sales` table and insert sample data.

2. Use a CTE to calculate total revenue for each salesperson.
3. Select the top salesperson from the CTE.

SQL Script:

```sql
-- Create a sales table
CREATE TABLE sales (
    id SERIAL PRIMARY KEY,
    salesperson VARCHAR(50),
    amount NUMERIC(10, 2)
);

-- Insert data into sales
INSERT INTO sales (salesperson, amount) VALUES
('Alice', 1200.50),
('Bob', 800.00),
('Alice', 500.00),
('Charlie', 700.00),
('Bob', 900.00);

-- Use a CTE
WITH total_sales AS (
    SELECT
        salesperson,
        SUM(amount) AS total_revenue
    FROM sales
    GROUP BY salesperson
)
SELECT
    salesperson,
    total_revenue
FROM total_sales
ORDER BY total_revenue DESC
LIMIT 1;
```

Output:

salesperson	total_revenue
Alice	1700.50

Recursive Query Using CTEs

What is a Recursive Query?

A recursive query is a query that refers to itself. It's useful for querying hierarchical or tree-structured data like organizational charts or file systems.

Syntax for Recursive CTE:

```sql
WITH RECURSIVE cte_name AS (
    initial_query
    UNION ALL
    recursive_query
```

```
)
SELECT ... FROM cte_name;
```

- `RECURSIVE`: Allows the CTE to refer to itself.
- `initial_query`: The base case for recursion.
- `recursive_query`: The iterative step that builds on the base case.

Example 2: Recursive CTE

Problem:

We have an `employees` table that represents a hierarchical structure of a company. Find the reporting hierarchy starting from the CEO.

Steps:

1. Create an `employees` table and insert sample data.
2. Use a recursive CTE to traverse the hierarchy.

SQL Script:

```sql


-- Create an employees table
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    manager_id INT
);

-- Insert data into employees
INSERT INTO employees (name, manager_id) VALUES
('Alice', NULL),   -- CEO
('Bob', 1),        -- Reports to Alice
('Charlie', 1),    -- Reports to Alice
('David', 2),      -- Reports to Bob
('Eve', 2);        -- Reports to Bob

-- Recursive CTE to find hierarchy
WITH RECURSIVE employee_hierarchy AS (
    SELECT
        id,
        name,
        manager_id,
        1 AS level
    FROM employees
    WHERE manager_id IS NULL -- Start with the CEO
    UNION ALL
    SELECT
        e.id,
        e.name,
        e.manager_id,
        eh.level + 1
    FROM employees e
    INNER JOIN employee_hierarchy eh
    ON e.manager_id = eh.id
)
SELECT
    name,
    manager_id,
    level
```

```
FROM employee_hierarchy
ORDER BY level, manager_id;
```

Output:

name	manager_id	level
Alice	NULL	1
Bob	1	2
Charlie	1	2
David	2	3
Eve	2	3

Why Use CTEs?

1. **Readability**: Simplifies complex queries.
2. **Reuse**: Results from a CTE can be reused in the main query.
3. **Modularity**: Breaks down queries into smaller, understandable parts.
4. **Recursion**: Solves problems like hierarchies and paths.

Would you like further examples or deeper dives into any aspect?