Varun Sagar Theegala

SIMPLIFY SQL QUERIES with these TOP 5 SQL CTE (Common Table Expression) PRACTICAL TIPS

Varun Sagar
Theegala

Common Table Expressions (CTEs) **simplify complex SQL queries** by breaking them into **manageable & readable parts.**

For analysts and developers, mastering CTEs is essential for writing **cleaner, more efficient code**, ultimately making SQL queries **more structured and easier to debug.**

Varun Sagar
Theegala

LET’S DIVE INTO 5 CTE TIPS TO SIMPLIFY COMPLEX QUERIES

#1

Use CTEs To Simplify Complex Queries

WHAT IT IS ?

Split complex queries into manageable parts using CTEs.

WHY IT MATTERS ?

Improves readability and simplifies debugging.

HOW TO IMPLEMENT ?

Define each logical step as a separate CTE.

LET'S WORK WITH AN EXAMPLE

BEFORE IMPLEMENTATION

```
SELECT name, total_sales
FROM (
  SELECT name, SUM(sales) AS total_sales
  FROM employees
  JOIN sales ON employees.id = sales.employee_id
  WHERE sales.date >= '2023-01-01'
  GROUP BY name
) AS sales_summary
WHERE total_sales > 1000;
```

AFTER IMPLEMENTATION

```
WITH sales_summary AS (
  SELECT name, SUM(sales) AS total_sales
  FROM employees
  JOIN sales ON employees.id = sales.employee_id
  WHERE sales.date >= '2023-01-01'
  GROUP BY name
)
SELECT name, total_sales
FROM sales_summary
WHERE total_sales > 1000;
```

Varun Sagar Theegala

NEXT

#2

Use Recursive CTEs for Hierarchical Data

WHAT IT IS ?

Create recursive CTEs to handle hierarchical or self-referential data.

WHY IT MATTERS ?

Efficiently traverses hierarchical structures like organizational charts.

HOW TO IMPLEMENT ?

Use UNION ALL with a base case and a recursive case.

Varun Sagar Theegala

NEXT

LET'S WORK WITH AN EXAMPLE

BEFORE IMPLEMENTATION

```sql
SELECT name, manager_id
FROM employees
WHERE manager_id IS NULL;

SELECT name, manager_id
FROM employees
WHERE manager_id = 1;

SELECT name, manager_id
FROM employees
WHERE manager_id = 2;
```

AFTER IMPLEMENTATION

```sql
WITH EmployeeHierarchy AS (
  SELECT id, name, manager_id
  FROM employees
  WHERE manager_id IS NULL
  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;
```

Varun Sagar Theegala

NEXT

#3
Improve Readability in Multi-JOIN Queries

WHAT IT IS ?

Simplify complex JOIN operations by breaking them into CTEs.

WHY IT MATTERS ?

Enhances readability and separates logical query parts.

HOW TO IMPLEMENT ?

Use separate CTEs for each JOIN operation.

LET'S WORK WITH AN EXAMPLE

BEFORE IMPLEMENTATION

```sql
SELECT c.name, o.order_id, p.product_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
JOIN products p ON od.product_id = p.product_id
WHERE c.city = 'New York';
```

AFTER IMPLEMENTATION

```sql
WITH CustomerOrders AS (
  SELECT c.name, o.order_id
  FROM customers c
  JOIN orders o ON c.customer_id = o.customer_id
  WHERE c.city = 'New York'
), ProductDetails AS (
  SELECT od.order_id, p.product_name
  FROM order_details od
  JOIN products p ON od.product_id = p.product_id
)
SELECT co.name, pd.product_name
FROM CustomerOrders co
JOIN ProductDetails pd ON co.order_id = pd.order_id;
```

#4

Use CTEs for Repeated Subqueries

WHAT IT IS ?

Avoid repeating subqueries by using CTEs to define them once.

WHY IT MATTERS ?

Reduces code duplication and makes maintenance easier.

HOW TO IMPLEMENT ?

Define the subquery as a CTE and reference it multiple times.

Varun Sagar Theegala

NEXT

LET'S WORK WITH AN EXAMPLE

BEFORE IMPLEMENTATION

```sql
SELECT name,
       (SELECT COUNT(*)
        FROM orders o
        WHERE o.customer_id = c.customer_id) AS order_count
FROM customers c
WHERE (SELECT COUNT(*)
       FROM orders o
       WHERE o.customer_id = c.customer_id) > 5;
```

AFTER IMPLEMENTATION

```sql
WITH OrderCounts AS (
  SELECT customer_id, COUNT(*) AS order_count
  FROM orders
  GROUP BY customer_id
)
SELECT c.name, oc.order_count
FROM customers c
JOIN OrderCounts oc ON c.customer_id = oc.customer_id
WHERE oc.order_count > 5;
```

Varun Sagar
Theegala

NEXT

#5

Use CTEs to Replace Temporary Tables

WHAT IT IS ?

Use CTEs instead of temporary tables for better performance.

WHY IT MATTERS ?

CTEs are easier to manage and don’t require explicit cleanup.

HOW TO OPTIMISE ?

Replace temporary table definitions with CTEs.

LET'S WORK WITH AN EXAMPLE

BEFORE IMPLEMENTATION

```
CREATE TEMP TABLE RecentSales AS
SELECT *
FROM sales
WHERE sale_date >= '2023-01-01';

SELECT product_id, SUM(amount)
FROM RecentSales
GROUP BY product_id;
```

BEFORE IMPLEMENTATION

```
WITH RecentSales AS (
  SELECT *
  FROM sales
  WHERE sale_date >= '2023-01-01'
)
SELECT product_id, SUM(amount)
FROM RecentSales
GROUP BY product_id;
```

Varun Sagar Theegala

NEXT

TL;DR:

- Break down complex queries logically.
- Use recursive CTEs for hierarchies.
- Simplify multi-join operations with CTEs.
- Replace repeated subqueries using CTEs.
- Use CTEs instead of temporary tables.

REMEMBER

CTEs are a **powerful tool** for writing **clear, maintainable SQL queries.**

By incorporating them into your workflow, you'll improve not just the quality of your code but also your **efficiency in solving complex problems.**

Varun Sagar Theegala

SHARE THIS

If you think **your network** would find this **valuable**

FOLLOW ME

I help you **GROW & SUSTAIN** as a **Data Analyst**