ECOLE SUPÉRIEURE DE LA STATISTIQUE
ET DE L'ANALYSE DE L'INFORMATION

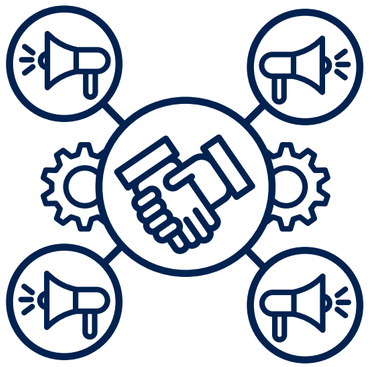
Common Table Expressions (CTEs)

Yahya Chammami

SOMMAIRE

1. What is a CTE?

Common Table Expressions (CTEs) are temporary result sets that can be referenced within a **SELECT**, **INSERT**, **UPDATE**, or **DELETE** statement. They allow for modular and organized SQL queries, helping to break down complex queries into simpler components.



2. Syntax of CTE.

The basic syntax for creating a CTE is as follows:

```
WITH cte_name AS (
    -- CTE query definition
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition
)
SELECT *
FROM cte_name;
```

Consider a table named employees with columns id, name, and salary. To fetch employees with salaries over $80,000:

```
with high_salary_employee as (
  select name,salary
  from employees
  where salary>80000
  )
  select*from high_salary_employee;
```

esult Grid | Filter Rows: | Export: | Wrap Cell

name	salary
David	85000.00
Eve	90000.00

2. Syntax of CTE.

You can also chain multiple CTEs together, separated by commas:

```
WITH cte1 AS (
    SELECT column1, column2
    FROM table_name1
),
cte2 AS (
    SELECT column1, column2
    FROM table_name2
)
SELECT *
FROM cte1
JOIN cte2 ON cte1.column1 = cte2.column1;
```

```
8  WITH cte_high AS (
9      SELECT name, salary FROM employees WHERE salary > 80000
0  ),
1  cte_low AS (
2      SELECT name, salary FROM employees WHERE salary <= 70000
3  )
4  SELECT *
5  FROM cte_high
6  UNION ALL
7  SELECT *
8  FROM cte_low;
```

sult Grid | Filter Rows: | Export: | Wrap Cell Content:

name	salary
David	85000.00
Eve	90000.00
Alice	70000.00
Charlie	60000.00

3. Practical Examples.

Recursive CTE Example

```sql
WITH RECURSIVE employee_hierarchy AS (
    -- Anchor member: Select top-level managers (those without managers)
    SELECT
        id,
        name,
        manager_id
    FROM employees
    WHERE manager_id IS NULL  -- Starting point (top-level manager)

    UNION ALL

    -- Recursive member: Select employees who report to the managers in the previous step
    SELECT
        e.id,
        e.name,
        e.manager_id
    FROM employees e
    INNER JOIN employee_hierarchy eh ON e.manager_id = eh.id
)
SELECT *
FROM employee_hierarchy;
```

3. Practical Examples.

Aggregating Data with CTEs

```
WITH total_sales AS (
    SELECT p.category_id, SUM(s.amount) AS total_amount
    FROM sales s
    JOIN products p ON s.product_id = p.id
    GROUP BY p.category_id
)
SELECT c.category_name, ts.total_amount
FROM categories c
JOIN total_sales ts ON c.id = ts.category_id;
```

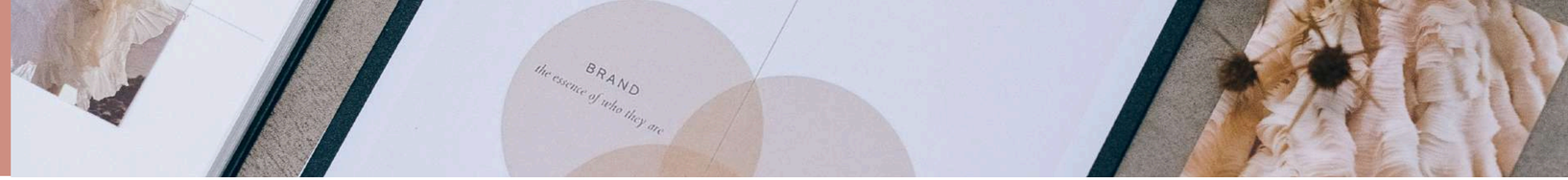
Filtering with a CTE

```
WITH high_performers AS (
    SELECT employee_id, performance_rating
    FROM performance_reviews
    WHERE performance_rating >= 90
)
SELECT e.name, e.email
FROM employees e
JOIN high_performers hp ON e.id = hp.employee_id;
```

3. Practical Examples.

Using CTEs for Data Transformation

```
WITH order_summary AS (
    SELECT order_id,
           SUM(item_price * quantity) AS total_price,
           SUM(shipping_cost) AS total_shipping,
           SUM(tax) AS total_tax,
           SUM(discount) AS total_discount
    FROM order_items
    JOIN orders o ON order_items.order_id = o.id
    GROUP BY order_id
)
SELECT os.order_id,
       os.total_price,
       os.total_shipping,
       os.total_tax,
       os.total_discount,
       (os.total_price + os.total_shipping + os.total_tax -
os.total_discount) AS grand_total
FROM order_summary os;
```



4. Benefits of using CTEs.

1

Improved Readability

CTEs make SQL queries more understandable by breaking them into smaller, logical units.

2

Enhanced Maintainability

CTEs simplify code updates and modifications, as changes only need to be made in one place.

3

Increased Efficiency

In some cases, using a CTE can improve query performance by reducing unnecessary data processing.

4

Logical Organization

They allow you to define logical subsections of the query upfront, making the main query clearer.

5. Limitations.

Scope

CTEs are temporary and only exist for the duration of the query that follows them. They cannot be reused outside that context.

Recursion Depth

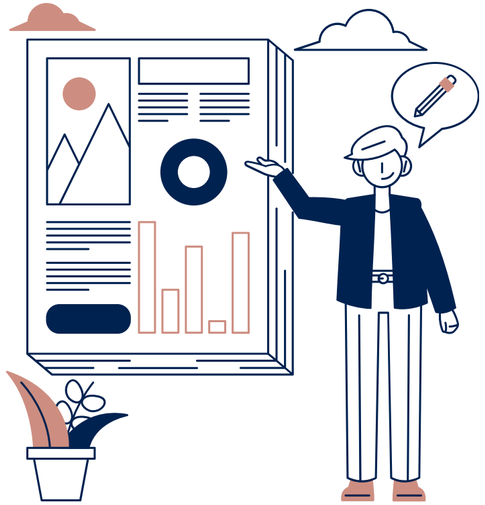
Recursive queries may lead to performance issues or infinite looping if not correctly structured, and you may hit system limits on recursion depth.

No Indexing

Since CTEs are not materialized, they do not allow for indexing, which can sometimes affect performance in large datasets.

MySQL Version

CTEs are only supported in MySQL version 8.0 and later. They are not available in earlier releases.



Conclusion.

CTEs provide a powerful mechanism for structuring and simplifying SQL queries in MySQL, making them easier to understand and maintain. However, like any tool, they should be used judiciously, considering their limitations and the specific requirements of your queries.