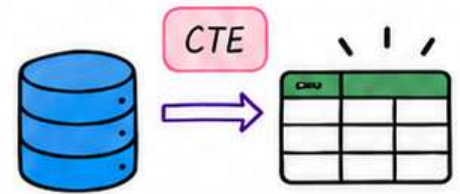


Chapter 23: MySQL Common Table Expressions

23.1 Introduction to CTEs

- * A **Common Table Expression (CTE)** is a temporary result set that can be referenced within a single SQL statement.
- * CTEs make complex queries more **readable** and help in breaking the query into logical parts.
- * CTEs are defined using the **WITH** clause.



Key Points :

- ✓ CTE is a named temporary result set.
- ✓ Improves readability and reusability of queries.
- ✓ Acts like a temporary table for the duration of the main query.



Example :

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

23.2 WITH Clause

- * The **WITH** clause is used to define one or more **CTEs**.
- * It must be placed at the **beginning** of the main SQL statement.
- * **Multiple CTEs** can be defined using commas.



Syntax :

```
WITH cte_name AS ( query )  
    [, cte_name AS ( query ) ... ]
```

Example :

```
WITH DepartmentCount AS (  
    SELECT dept_id, COUNT(*) AS total_employees  
    FROM employees GROUP BY dept_id  
)  
SELECT * FROM DepartmentCount ;
```

Key Takeaway :



CTEs (using **WITH** clause) make SQL queries **modular, readable** and maintainable by allowing temporary named result sets.

23.3 Single CTE

- * A **single CTE** contains **one temporary result set**.
- * It is useful for breaking down complex queries into simpler parts.
- * It can be referenced **once** or **multiple times** in the main query.

Syntax :

```
WITH cte_name AS (  
    SELECT column_list  
    FROM table_name  
    WHERE condition  
)  
SELECT column_list  
FROM cte_name  
WHERE condition ;
```

Example :

```
WITH high_salary AS (  
    SELECT emp_id, emp_name, salary  
    FROM employees  
    WHERE salary > 50000  
)  
SELECT emp_name, salary  
FROM high_salary  
ORDER BY salary DESC ;
```

emp_name	salary
Alice	60000
John	75000
Robert	55000

Key Takeaway:



A single CTE makes the query **easier to read** and **maintain** by creating a **temporary result set**.



23.4 Multiple CTEs

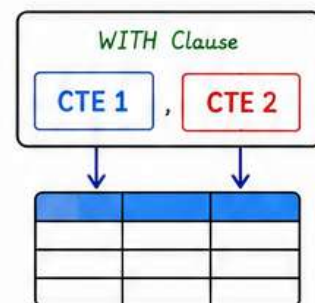
- * Multiple CTEs are defined in the **same WITH clause**, separated by commas.
- * They can **reference each other** in the order they are defined.
- * Useful for building **step-by-step** logic in complex queries.

Syntax :

```
WITH cte1 AS ( subquery1 ),  
     cte2 AS ( subquery2 ),  
     ...  
SELECT column_list  
FROM cte_name  
WHERE condition ;
```

Example :

```
WITH dept_count AS (  
    SELECT dept_id, COUNT(*) AS total_emp  
    FROM employees  
    GROUP BY dept_id  
,  
     high_earners AS (  
    SELECT emp_id, emp_name, salary, dept_id  
    FROM employees  
    WHERE salary > 60000  
)  
SELECT h.emp_name, h.salary, d.total_emp  
FROM high_earners h  
JOIN dept_count d ON h.dept_id = d.dept_id  
ORDER BY h.salary DESC ;
```



dept_count CTE (Step 1)	
dept_id	total_emp
10	3
20	2
30	4

high_earners CTE (Step 2)			
emp_id	emp_name	salary	dept_id
2	Alice	60000	10
4	John	75000	20
6	Eve	65000	30

emp_name	salary	total_emp
John	75000	2
Eve	65000	4
Alice	60000	3

Key Takeaway:



Multiple CTEs allow us to build complex queries in a **modular** and **readable** way by using intermediate result sets.



23.5 CTE with JOIN

- ✳ A CTE can be used with **JOIN** to combine data from multiple tables.
- ✳ The CTE is defined first and then used in the main query.
- ✳ It improves **readability** and **simplifies** complex **JOIN** operations.

Syntax :

```
WITH cte_name AS (  
    SELECT column_list  
    FROM table_name  
    WHERE condition  
)  
SELECT c.column1, t.column2, ...  
FROM cte_name c  
JOIN another_table t ON c.key = t.key  
WHERE condition ;
```

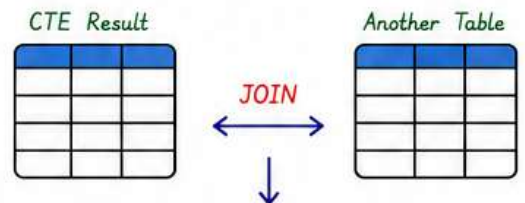
Example :

```
WITH emp_dept AS (  
    SELECT emp_id, emp_name, dept_id  
    FROM employees  
)  
SELECT e.emp_name, d.dept_name  
FROM emp_dept e  
JOIN departments d ON e.dept_id = d.dept_id  
ORDER BY e.emp_name ;
```

Key Takeaway :



CTEs with **JOIN** make complex queries **easier** to write, understand and maintain.



Joined Result

employees

emp_id	emp_name	dept_id
1	Alice	10
2	John	20
3	Robert	10
4	Eve	30

departments

dept_id	dept_name
10	HR
20	IT
30	Finance

emp_name	dept_name
Alice	HR
John	IT
Eve	Finance
Robert	HR

23.6 CTE with Aggregation

- ✳ A CTE can be used with aggregate functions like **COUNT()**, **SUM()**, **AVG()**, **MAX()**, **MIN()** to summarize data.
- ✳ The aggregation is performed in the CTE and the result is used in the main query.
- ✳ It helps in **breaking down** complex aggregation logic.

Syntax :

```
WITH cte_name AS (  
    SELECT group_column, aggregate_function(column) AS alias  
    FROM table_name  
    GROUP BY group_column  
    HAVING condition  
)  
SELECT *  
FROM cte_name  
WHERE condition  
ORDER BY alias ;
```

Example :

```
WITH sales_summary AS (  
    SELECT emp_id, COUNT(*) AS total_orders,  
           SUM(amount) AS total_amount  
    FROM orders  
    GROUP BY emp_id  
)  
SELECT *  
FROM sales_summary  
WHERE total_amount > 3000  
ORDER BY total_amount DESC ;
```

Key Takeaway :



CTEs with aggregation help **summarize** data first and then use the summarized result in the main query for **better analysis**.



orders

order_id	emp_id	amount	order_date
101	1	2500	2024-01-01
102	1	1500	2024-01-05
103	2	3000	2024-01-03
104	2	2000	2024-01-06
105	3	4000	2024-01-07

emp_id	total_orders	total_amount
3	1	4000
2	2	5000
1	2	4000

23.7 Recursive CTE

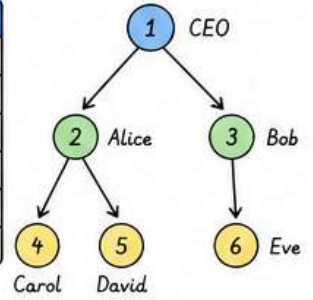
- * A Recursive CTE is used when a query needs to **reference itself**.
- * It is useful for **hierarchical or tree-like** data.
- * Recursive CTEs use **UNION ALL** to combine the anchor and recursive members.
- * It consists of an **anchor member** and a **recursive member**.

Syntax :

```
WITH RECURSIVE cte_name AS (
    -- Anchor member
    SELECT column_list
    FROM table_name
    WHERE condition
    UNION ALL
    -- Recursive member
    SELECT column_list
    FROM table_name t
    JOIN cte_name c ON t.key = c.key
    WHERE condition
)
SELECT *
FROM cte_name
ORDER BY column_name ;
```

Hierarchy Example (Employee Table)

emp_id	emp_name	manager_id
1	CEO	NULL
2	Alice	1
3	Bob	1
4	Carol	2
5	David	2
6	Eve	3



Example : Get all employees in the hierarchy starting from CEO.

```
WITH RECURSIVE emp_hierarchy AS (
    -- Anchor member: Top level (CEO)
    SELECT emp_id, emp_name, manager_id, 1 AS level
    FROM employees
    WHERE manager_id IS NULL
    UNION ALL
    -- Recursive member: Employees under the previous level
    SELECT e.emp_id, e.emp_name, e.manager_id, eh.level + 1
    FROM employees e
    JOIN emp_hierarchy eh ON e.manager_id = eh.emp_id
)
SELECT emp_id, emp_name, manager_id, level
FROM emp_hierarchy
ORDER BY level, emp_id ;
```

Key Takeaway :



Recursive CTEs are **powerful** for hierarchical data traversal such as organizational structures, file directories, bill of materials, etc.

Result

emp_id	emp_name	manager_id	level
1	CEO	NULL	1
2	Alice	1	2
3	Bob	1	2
4	Carol	2	3
5	David	2	3
6	Eve	3	3

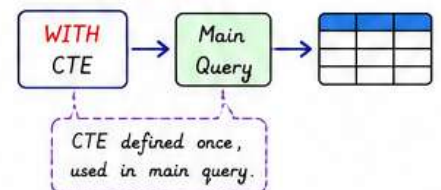


23.8 CTE vs Subquery

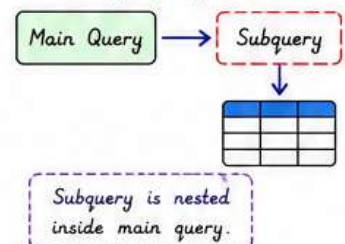
- * Both CTEs and Subqueries are used to break down complex queries, but they have some **key differences**.

Feature	CTE (Common Table Expression)	Subquery
Definition	A named temporary result set defined using WITH clause.	A query nested inside another query.
Readability	Improves readability by separating logic into meaningful steps.	Can be harder to read for complex queries.
Reusability	Can be referenced multiple times in the main query.	Typically used once within a query.
Performance	May improve performance by allowing the optimizer to work better.	Can sometimes lead to slower performance in complex scenarios.
Recursion	Supports recursive queries .	Cannot be recursive.
Syntax	Uses WITH clause.	Uses nested SELECT statements.
Use Case	Best for complex, multi-step, or recursive logic.	Best for simple, single-step filtering or calculations.

CTE Approach



Subquery Approach



Examples :

CTE Example

```
WITH high_salary AS (
    SELECT emp_id, emp_name, salary
    FROM employees
    WHERE salary > 50000
)
SELECT *
FROM high_salary
ORDER BY salary DESC ;
```

Subquery Example

```
SELECT emp_id, emp_name, salary
FROM employees
WHERE salary > (
    SELECT AVG(salary)
    FROM employees
);
```

Key Takeaway :



Use CTEs for **better readability, reusability, and recursion**.
Use Subqueries for **simple, one-time filtering**.

