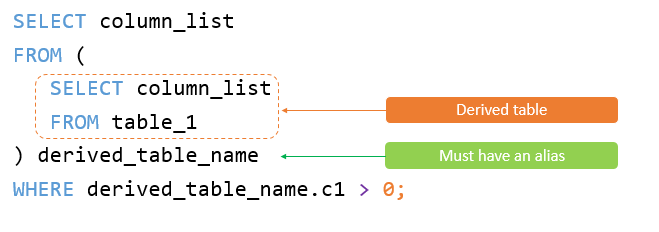
**Introduction to the common table expression (CTE)**

A common table expression is a named temporary result set that exists solely within the execution scope of a single SQL statement, such as [SELECT](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/), [INSERT](https://www.mysqltutorial.org/mysql-basics/mysql-insert/), [UPDATE](https://www.mysqltutorial.org/mysql-basics/mysql-update/), or [DELETE](https://www.mysqltutorial.org/mysql-basics/mysql-delete/).

Similar to a [derived table](https://www.mysqltutorial.org/mysql-basics/mysql-derived-table/), a common table expression (CTE) is not stored as an object and lasts only during the query execution.



Unlike a derived table, a common table expression (CTE) can be self-referencing (in the case of a [recursive CTE](https://www.mysqltutorial.org/mysql-basics/mysql-recursive-cte/)) or referenced multiple times within the same query. Moreover, a CTE offers enhanced readability and performance compared to a derived table.

A CTE is defined using the WITH keyword and can be referenced multiple times within the same query. CTEs are similar to subqueries, but they allow for better readability and can also be recursive.

CREATE DATABASE mysqlsampledatabase;

USE mysqlsampledatabase;

Syntax for CTE

**WITH cte\_name (column1, column2, ...)**

**AS (**

**SELECT query**

**)**

**SELECT \* FROM cte\_name;**

* cte\_name: The name you give the temporary result set.
* The SELECT query inside the CTE generates the temporary result set.
* You can reference the CTE in the main SELECT, UPDATE, INSERT, or DELETE queries.

Advantages of Using CTEs:

1. Improved Readability: Makes complex queries easier to read and understand by breaking them down into parts.
2. Reusability: You can reuse the CTE multiple times in the same query, avoiding redundant code.
3. Recursive Queries: CTEs support recursion, which allows querying hierarchical data like organizational structures.

**Recursive CTE Syntax:**

**WITH RECURSIVE cte\_name (column1, column2, ...)**

**AS (**

**-- Anchor member (base case)**

**SELECT query**

**UNION ALL**

**-- Recursive member**

**SELECT query referencing cte\_name**

**)**

**SELECT \* FROM cte\_name;**

**Problem 1: Retrieve the Top 5 Highest Salaries**

WITH TopSalaries AS (

SELECT employee\_id, salary

FROM employees

ORDER BY salary DESC

LIMIT 5

)

SELECT salary FROM TopSalaries;

**Why use CTE?**

In this example, we need to extract the top 5 highest salaries, and a CTE allows us to create a temporary result set (TopSalaries) that can be reused within the query. This makes the query modular and easier to read, especially if you need to perform more operations on the top salaries in a complex query. Using a CTE here helps to separate the logic and allows for potential reuse of the result set without repeating the same subquery multiple times.

**Problem: Retrieve the Top 5 Highest Salaries and Calculate Their Total Sum**

WITH TopSalaries AS (

SELECT employee\_id, salary

FROM employees

ORDER BY salary DESC

LIMIT 5

)

SELECT employee\_id, salary FROM TopSalaries;

**-- Additional operation: Calculate the total salary of the top 5 employees**

WITH TopSalaries AS (

SELECT employee\_id, salary

FROM employees

ORDER BY salary DESC

LIMIT 5

)

SELECT SUM(salary) AS total\_top\_salaries

FROM TopSalaries;

**Problem 2: Calculate the Average Salary of Each Department**

**WITH AvgSalary AS (**

**SELECT department\_id, AVG(salary) AS avg\_salary**

**FROM employees**

**GROUP BY department\_id**

**)**

**SELECT departments.department\_name, AvgSalary.avg\_salary**

**FROM departments**

**JOIN AvgSalary ON departments.department\_id = AvgSalary.department\_id;**

**Why use CTE?**

We are using a CTE to calculate the average salary for each department and store it temporarily. By using a CTE, we separate the salary calculation logic from the JOIN logic that fetches the department name. This keeps the query modular, easy to understand, and allows reusing AvgSalary as a reusable table in other parts of the query if needed.

**Problem 3: List Employees with Salaries Above the Average Salary**

WITH AvgSalary AS (

SELECT AVG(salary) AS company\_avg\_salary

FROM employees

)

SELECT employee\_id, first\_name, salary

FROM employees, AvgSalary

WHERE salary > AvgSalary.company\_avg\_salary;

**Why use CTE?**

The CTE calculates the average salary once and makes it available for the main query. Without a CTE, you’d have to compute the average salary separately for each row, leading to performance inefficiencies. The CTE enables us to cleanly encapsulate this logic, avoid repeated calculations, and improve the readability of the query.

WITH DepartmentSalaries AS (

SELECT department\_id, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department\_id

)

-- First operation: Retrieve the average salary for each department

SELECT department\_id, avg\_salary FROM DepartmentSalaries;

-- Additional operation: Calculate the total number of departments

SELECT COUNT(department\_id) AS total\_departments

FROM DepartmentSalaries;

**Using a CTE with a Window Function**

You can use a **window function** to calculate the total number of departments along with the average salary for each department in the same query. This avoids running two separate queries.

WITH DepartmentSalaries AS (

SELECT

department\_id,

AVG(salary) AS avg\_salary,

COUNT(department\_id) OVER () AS total\_departments -- Window function to count all departments

FROM employees

GROUP BY department\_id

)

SELECT department\_id, avg\_salary, total\_departments FROM DepartmentSalaries;

**Problem 4 (Extended): Ranking Employees by Salary and Calculate Median Rank Salary**

WITH RankedSalaries AS (

SELECT EMPLOYEE\_ID, SALARY, RANK() OVER (ORDER BY SALARY DESC) AS `rank`

FROM employees

)

SELECT EMPLOYEE\_ID, SALARY, `rank` FROM RankedSalaries;

-- Additional operation: Calculate the median rank salary (for simplicity, using rank 3)

WITH RankedSalaries AS (

SELECT EMPLOYEE\_ID, SALARY, DENSE\_RANK() OVER (ORDER BY SALARY DESC) AS `rank`

FROM employees

)

SELECT SALARY AS median\_salary

FROM RankedSalaries

WHERE `rank` = 3;