

A common table expression, or CTE, is a temporary named result set created from a simple SELECT statement that can be used in a subsequent SELECT statement

We can define CTEs by adding a WITH clause directly

before SELECT, INSERT, UPDATE, DELETE, or MERGE

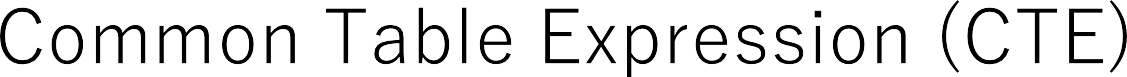
statement.

The **WITH** clause can include one or more CTEs separated by commas

96

##### customer csv file: https://bit.ly/3xGABBR payment csv file: https://bit.ly/3Zc0GUV

97



The name of this CTE is my\_cte, and the CTE query is SELECT a,b,c FROM Table1. The CTE starts with the **WITH** keyword, after which you specify the name of your CTE, then the content of the query in parentheses. The main query comes after the closing parenthesis and refers to the CTE. Here, the main query (also known as the outer query) is SELECT a,c FROM my\_cte



**Syntax**

**WITH my\_cte AS (**

SELECT a,b,c

FROM Table1 **)**

SELECT a,c

FROM **my\_cte**

Main query

CTE query

98

**1. Example EASY**

WITH my\_cte AS (

SELECT \*, AVG(amount) OVER(ORDER BY

p.customer\_id) AS "Average\_Price",

COUNT(address\_id) OVER(ORDER BY c.customer\_id) AS "Count"

FROM payment as p INNER JOIN customer AS c

ON p.customer\_id = c.customer\_id

)

SELECT first\_name, last\_name FROM my\_cte

**1. Example Multiple CTEs**

WITH my\_cp AS (

SELECT \*, AVG(amount) OVER(ORDER BY p.customer\_id) AS "Average\_Price",

COUNT(address\_id) OVER(ORDER BY c.customer\_id) AS

"Count"

FROM payment as p INNER JOIN customer AS c

ON p.customer\_id = c.customer\_id

),

my\_ca AS (

SELECT \*

FROM customer as c INNER JOIN address AS a

ON a.address\_id = c.address\_id INNER JOIN country as cc

ON cc.city\_id = a.city\_id

)

SELECT cp.first\_name, cp.last\_name, ca.city, ca.country, cp.amount FROM my\_ca as ca , my\_cp as cp

**2. Example Advance**

WITH my\_cte AS (

SELECT mode, MAX(amount) AS highest\_price, SUM(amount) AS total\_price

FROM payment GROUP BY mode

)

SELECT payment.\*, my.highest\_price, my.total\_price FROM payment

JOIN my\_cte my

ON payment.mode = my.mode ORDER BY payment.mode

99

CTEs are useful for breaking down complex queries, improving readability, and simplifying maintenance.

**Advantages of CTEs**

* **Improved Readability**: CTEs make complex queries more readable by breaking them into smaller, more manageable parts.
* **Reusable Query Logic**: CTEs can be referenced multiple times within the same query, avoiding duplication of query logic.
* **Easier Maintenance**: Updating a CTE query is simpler than modifying nested subqueries.
* **Recursive Queries**: CTEs support recursion, which can be useful for hierarchical data (e.g., organizational structures, family trees).

**Disadvantages of CTEs**

* **Performance**: In some cases, CTEs can be less efficient than equivalent subqueries or derived tables, especially if not indexed properly.
* **Scope**: CTEs are only valid within the execution scope of the query they are defined in and cannot be reused across different queries.

**Use Cases for CTEs**

* **Hierarchical Data**: Recursive CTEs are ideal for querying hierarchical data structures, such as organizational charts or tree structures.
* **Complex Aggregations**: Simplifying complex aggregation queries by breaking them into smaller steps.
* **Data Transformation**: Intermediate transformations in ETL (Extract, Transform, Load) processes.
* **Modular Queries**: Breaking down large and complex queries into modular parts for better readability and maintenance.

**Practice Example**

**-- Sample data for customers**

**CREATE TABLE customers (**

**customer\_id INT PRIMARY KEY,**

**first\_name VARCHAR(50),**

**last\_name VARCHAR(50)**

**);**

**INSERT INTO customers (customer\_id, first\_name, last\_name)**

**VALUES**

**(1, 'John', 'Doe'),**

**(2, 'Jane', 'Smith'),**

**(3, 'Michael', 'Brown');**

**-- Sample data for orders**

**CREATE TABLE orders (**

**order\_id INT PRIMARY KEY,**

**customer\_id INT,**

**order\_date DATE,**

**amount DECIMAL(10, 2),**

**FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)**

**);**

**INSERT INTO orders (order\_id, customer\_id, order\_date, amount)**

**VALUES**

**(1, 1, '2023-01-01', 100.00),**

**(2, 1, '2023-01-15', 200.00),**

**(3, 2, '2023-02-01', 150.00),**

**(4, 3, '2023-02-15', 300.00);**

Using the above tables, write a query to find the top spending customers and their total spend:

**Answer**

WITH TotalSpent AS (

SELECT customer\_id, SUM(amount) AS total\_amount

FROM orders

GROUP BY customer\_id

)

SELECT c.customer\_id, c.first\_name, c.last\_name, ts.total\_amount

FROM customers c

JOIN TotalSpent ts ON c.customer\_id = ts.customer\_id

ORDER BY ts.total\_amount DESC;

Questions

* 1. **main query is must to write.**

Yes, when using Common Table Expressions (CTEs), the main query that references the CTE is essential. Without the main query, the CTEs have no context or purpose. The CTEs are defined for use in the subsequent main query, and they don't exist independently of it.

* 1. **Can we use CTE table multiple times in same query file?**
     + Yes, you can reference a CTE multiple times within the same query file, but it must be within the scope of a single SQL statement. CTEs are defined for use within a single query and can't persist beyond that. If you need to use the same CTE results in multiple statements, you would **either need to redefine the CTE in each statement or use a temporary table.**

1. **Using a CTE Multiple Times in a Single Statement**

WITH my\_cte AS (

SELECT

c.customer\_id,

c.first\_name AS customer\_name,

o.order\_id,

o.order\_date,

o.amount AS order\_amount

FROM

customers c

LEFT JOIN

orders o ON c.customer\_id = o.customer\_id

)

SELECT

customer\_name,

order\_amount

FROM

my\_cte;

-- Use the CTE again in the same statement:

WITH my\_cte AS (

SELECT

c.customer\_id,

c.first\_name AS customer\_name,

o.order\_id,

o.order\_date,

o.amount AS order\_amount

FROM

customers c

LEFT JOIN

orders o ON c.customer\_id = o.customer\_id

)

SELECT

customer\_name,

COUNT(order\_id) AS total\_orders

FROM

my\_cte

GROUP BY

customer\_name;

1. **Using a Temporary Table**
   1. If you need to reuse the results in multiple queries, consider using a temporary table:

**-- Create a temporary table**

**CREATE TEMPORARY TABLE temp\_my\_cte AS**

**SELECT**

**c.customer\_id,**

**c.first\_name AS customer\_name,**

**o.order\_id,**

**o.order\_date,**

**o.amount AS order\_amount**

**FROM**

**customers c**

**LEFT JOIN**

**orders o ON c.customer\_id = o.customer\_id;**

**-- First usage**

**SELECT**

**customer\_name,**

**order\_amount**

**FROM**

**temp\_my\_cte;**

**-- Second usage**

**SELECT**

**customer\_name,**

**COUNT(order\_id) AS total\_orders**

**FROM**

**temp\_my\_cte**

**GROUP BY**

**customer\_name;**

**-- Optionally, drop the temporary table when done**

**DROP TEMPORARY TABLE temp\_my\_cte;**

1. **Using a View**
   1. Alternatively, if you want to reuse the same logic frequently, you can create a view:

-- Create a view

CREATE VIEW my\_cte\_view AS

SELECT

c.customer\_id,

c.first\_name AS customer\_name,

o.order\_id,

o.order\_date,

o.amount AS order\_amount

FROM

customers c

LEFT JOIN

orders o ON c.customer\_id = o.customer\_id;

-- First usage

SELECT

customer\_name,

order\_amount

FROM

my\_cte\_view;

-- Second usage

SELECT

customer\_name,

COUNT(order\_id) AS total\_orders

FROM

my\_cte\_view

GROUP BY

customer\_name;

By using a view or a temporary table, you can avoid redefining the CTE multiple times and make your queries more efficient and easier to maintain.