# An Introduction to MySQL CTE

## **What is a common table expression or CTE?**

A common table expression is a named temporary result set that exists only within the execution scope of a single SQL statement e.g., SELECT, INSERT, UPDATE, or DELETE.

Similar to a derived table, a CTE is not stored as an object and last only during the execution of a query.

Unlike a derived table, a CTE can be self-referencing (a recursive CTE) or can be referenced multiple times in the same query.

In addition, a CTE provides better readability and performance in comparison with a derived table.

## **MySQL CTE syntax**

The structure of a CTE includes the name, an optional column list, and a query that defines the CTE.

After the CTE is defined, you can use it as a view in a SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement.

The following illustrates the basic syntax of a CTE:

WITH cte\_name (column\_list) AS (

query

)

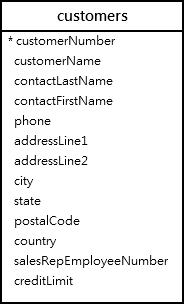
SELECT \* FROM cte\_name;

Notice that the number of columns in the query must be the same as the number of columns in the column\_list.

If you omit the column\_list, the CTE will use the column list of the query that defines the CTE

## **Simple MySQL CTE examples**

We’ll use the customers table from the sample database for demonstration:



The following example illustrates how to use a CTE for querying data from the customers table in the sample database.

Note that this example is only for the demonstration purpose to make it easy for you to understand the CTE concept.

WITH customers\_in\_usa AS (

SELECT

customerName, state

FROM

customers

WHERE

country = 'USA'

) SELECT

customerName

FROM

customers\_in\_usa

WHERE

state = 'CA'

ORDER BY customerName;



In this example, the name of the CTE is customers\_in\_usa, the query that defines the CTE returns two columns customerName and state. Hence, the customers\_in\_usa CTE returns all customers located in the USA.

After defining the customers\_in\_usa CTE, we referenced it in the SELECT statement to select only customers located in California.

See the following example:

WITH topsales2003 AS (

SELECT

salesRepEmployeeNumber employeeNumber,

SUM(quantityOrdered \* priceEach) sales

FROM

orders

INNER JOIN

orderdetails USING (orderNumber)

INNER JOIN

customers USING (customerNumber)

WHERE

YEAR(shippedDate) = 2003

AND status = 'Shipped'

GROUP BY salesRepEmployeeNumber

ORDER BY sales DESC

LIMIT 5

)

SELECT

employeeNumber,

firstName,

lastName,

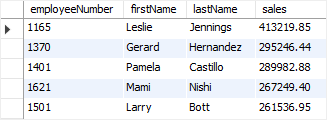
sales

FROM

employees

JOIN

topsales2003 USING (employeeNumber);



In this example, the CTE returns the top 5 sales rep in 2003. After that, we referenced to the topsales2003 CTE to get additional information about the sales rep including first name and last name.

## **A more advanced MySQL CTE example**

See the following example:

WITH salesrep AS (

SELECT

employeeNumber,

CONCAT(firstName, ' ', lastName) AS salesrepName

FROM

employees

WHERE

jobTitle = 'Sales Rep'

),

customer\_salesrep AS (

SELECT

customerName, salesrepName

FROM

customers

INNER JOIN

salesrep ON employeeNumber = salesrepEmployeeNumber

)

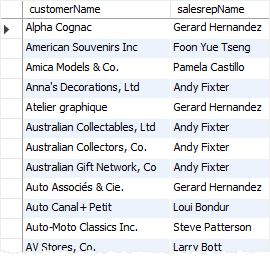
SELECT

\*

FROM

customer\_salesrep

ORDER BY customerName;



In this example, we have two CTEs in the same query. The first CTE ( salesrep) gets the employees whose job titles are the sales representative. The second CTE ( customer\_salesrep ) references the first CTE in the [INNER JOIN](https://www.mysqltutorial.org/mysql-inner-join.aspx) clause to get the sales rep and customers of whom each sales rep is in charge.

After having the second CTE, we query data from that CTE using a simple SELECT statement with the [ORDER BY](https://www.mysqltutorial.org/mysql-order-by/) clause.

## **The WITH clause usages**

There are some contexts that you can use the WITH clause to make common table expressions:

First, a WITH clause can be used at the beginning of SELECT, UPDATE, and DELETE statements:

WITH ... SELECT ...

WITH ... UPDATE ...

WITH ... DELETE ...

Second, a WITH clause can be used at the beginning of a subquery or a derived table subquery:

SELECT ... WHERE id IN (WITH ... SELECT ...);

SELECT \* FROM (WITH ... SELECT ...) AS derived\_table;

Third, a WITH clause can be used immediately preceding SELECT of the statements that include a SELECT clause:

CREATE TABLE ... WITH ... SELECT ...

CREATE VIEW ... WITH ... SELECT ...

INSERT ... WITH ... SELECT ...

REPLACE ... WITH ... SELECT ...

DECLARE CURSOR ... WITH ... SELECT ...

EXPLAIN ... WITH ... SELECT ...

In this tutorial, you have learned how to use MySQL CTE to simplify complex queries.

# A Definitive Guide To MySQL Recursive CTE

## **Introduction to MySQL recursive CTE**

A recursive common table expression (CTE) is a CTE that has a subquery which refers to the CTE name itself. The following illustrates the syntax of a recursive CTE

WITH RECURSIVE cte\_name AS (

initial\_query -- anchor member

UNION ALL

recursive\_query -- recursive member that references to the CTE name

)

SELECT \* FROM cte\_name;

A recursive CTE consists of three main parts:

* An initial query that forms the base result set of the CTE structure. The initial query part is referred to as an anchor member.
* A recursive query part is a query that references to the CTE name, therefore, it is called a recursive member. The recursive member is joined with the anchor member by a UNION ALL or UNION DISTINCT operator.
* A termination condition that ensures the recursion stops when the recursive member returns no row.

The execution order of a recursive CTE is as follows:

1. First, separate the members into two: anchor and recursive members.
2. Next, execute the anchor member to form the base result set ( R0) and use this base result set for the next iteration.
3. Then, execute the recursive member with Ri result set as an input and make Ri+1 as an output.
4. After that, repeat the third step until the recursive member returns an empty result set, in other words, the termination condition is met.
5. Finally, combine result sets from R0 to Rn using UNION ALL operator.

## **Recursive member restrictions**

The recursive member must not contain the following constructs:

* Aggregate functions e.g., MAX, MIN, SUM, AVG, COUNT, etc.
* GROUP BY clause
* ORDER BY clause
* LIMIT clause
* DISTINCT

Note that the above constraint does not apply to the anchor member. Also, the prohibition on DISTINCT applies only when you use UNION operator. In case you use the UNION DISTINCT operator, the DISTINCT is permitted.

In addition, the recursive member can only reference the CTE name once and in its FROM clause, not in any subquery.

## **Simple MySQL recursive CTE example**

See the following simple recursive CTE example:

WITH RECURSIVE cte\_count (n)

AS (

SELECT 1

UNION ALL

SELECT n + 1

FROM cte\_count

WHERE n < 3

)

SELECT n

FROM cte\_count;

In this example, the following query:

SELECT 1

is the anchor member that returns 1 as the base result set.

The following query

SELECT n + 1

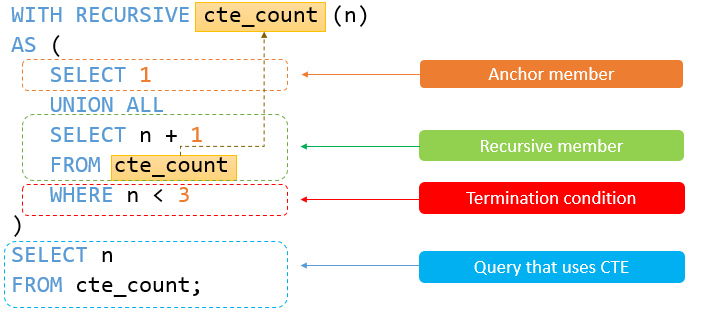
FROM cte\_count

WHERE n < 3

is the recursive member because it references to the name of the CTE which is cte\_count.

The expressionn < 3 in the recursive member is the termination condition. Once n equals 3, the recursive member returns an empty set that will stop the recursion.

The following picture illustrates the elements of the CTE above:



The recursive CTE returns the following output:

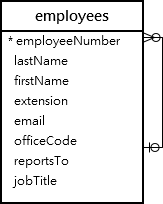


The execution steps of the recursive CTE is as follows:

1. First, separate the anchor and recursive members.
2. Next, the anchor member forms the initial row ( SELECT 1) therefore the first iteration produces 1 + 1 = 2 with n = 1.
3. Then, the second iteration operates on the output of the first iteration (2) and produces 2 + 1 = 3 with n = 2.
4. After that, before the third operation ( n = 3), the termination condition ( n < 3) is met therefore the query stops.
5. Finally, combine all result sets 1, 2 and 3 using the UNION ALL operator

## **Using MySQL recursive CTE to traverse the hierarchical data**

We will use the  employees table in the classicmodels sample database for the demonstration.



The  employees table has the reportsTo column that references to the employeeNumber column. The reportsTo column stores the ids of managers. The top manager does not report to anyone in the company’s organization structure, therefore, the value in the reportsTo column is NULL.

You can apply the recursive CTE to query the whole organization structure in the top-down manner as follows:

WITH RECURSIVE employee\_paths AS

( SELECT employeeNumber,

reportsTo managerNumber,

officeCode,

1 lvl

FROM employees

WHERE reportsTo IS NULL

UNION ALL

SELECT e.employeeNumber,

e.reportsTo,

e.officeCode,

lvl+1

FROM employees e

INNER JOIN employee\_paths ep ON ep.employeeNumber = e.reportsTo )

SELECT employeeNumber,

managerNumber,

lvl,

city

FROM employee\_paths ep

INNER JOIN offices o USING (officeCode)

ORDER BY lvl, city;

Let’s break the query into smaller parts to make it easier to understand.

First, form the anchor member by using the following query:

SELECT

employeeNumber,

reportsTo managerNumber,

officeCode

FROM

employees

WHERE

reportsTo IS NULL;

This query (anchor member) returns the top manager whose reportsTo is NULL.

Second, make the recursive member by reference to the CTE name, which is employee\_paths in this case:

SELECT

e.employeeNumber,

e.reportsTo,

e.officeCode

FROM

employees e

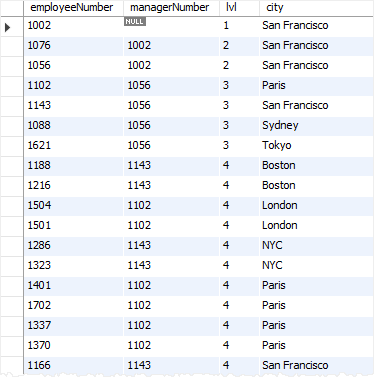
INNER JOIN employee\_paths ep

ON ep.employeeNumber = e.reportsTo;

This query ( recursive member) returns all direct reports of the manager(s) until there are no more direct reports. The if the recursive member returns no direct reports, the recursion stops.

Third, the query that uses the employee\_paths CTE joins the result set returned by the CTE with the offices table to make the final result set.

The following is the output of the query:



In this tutorial, you have learned about MySQL recursive CTE and how to use it to traverse hierarchical data.