MySQLTUTORIAL

An Introduction to MySQL CTE

Summary: in this tutorial, you will learn how to use MySQL CTE or common table expression to construct complex queries in a more readable manner.

> MySQL introduced the common table expression or CTE feature since version 8.0 so you should have MySQL 8.0+ in order to practice with the statements in this tutorial.

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` (https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) , `INSERT` (https://www.mysqltutorial.org/mysql-insert-statement.aspx) , `UPDATE` (https://www.mysqltutorial.org/mysql-update-data.aspx) , or `DELETE` (https://www.mysqltutorial.org/mysql-delete-statement.aspx) .

Similar to a derived table (https://www.mysqltutorial.org/mysql-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 (https://www.mysqltutorial.org/mysql-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 (https://www.mysqltutorial.org/mysql-sample-database.aspx) .

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 (https://www.mysqltutorial.org/mysql-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.