A Common Table Expression (CTE) can make it easier to manage and write complex queries by making them more readable and simple, like database views and derived tables. We can reuse or rewrite the query by breaking down the complex queries into simple blocks.

The SQL Common Table Expression

The **WITH** clause in MySQL is used to specify a Common Table Expression.

A Common Table Expression (CTE) in SQL is a one-time result set, i.e. it is a temporary table that exists only during the execution of a single query. It allows us to work with data specifically within that query, such as using it in **SELECT**, **UPDATE**, **INSERT**, **DELETE**, **CREATE**, **VIEW**, OR **MERGE** statements.

CTE is temporary because it cannot be stored anywhere for later use; once the query is executed, it is lost.

You cannot use the WITH clause in MySQL versions before 8.0.

### **Syntax**

Following is the syntax to create a CTE using WITH clause −

WITH CTE\_NAME (column\_name)

AS (query) SELECT \* FROM CTE\_NAME;

Where,

* **CTE\_NAME −** It is the name assigned to the CTE.
* **column\_name −** It is the column names for the CTE, which can be useful for improving query readability.
* **query −** It defines the CTE and it can be any valid SQL query.
* After defining the CTE, you can reference it in subsequent queries within the same session.

Ex-

The table created is as shown below −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | Kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | Hyderabad | 4500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |

1. we are creating a Common Table Expression (CTE) named **CUSTOMER\_AGE** that selects all customers with an age of 23. We are then retrieving the ID, NAME, and AGE of these customers from the CTE.

WITH CUSTOMER\_AGE AS ( SELECT \* FROM customers WHERE AGE = 23) SELECT ID, NAME, AGE FROM CUSTOMER\_AGE;

Following is the output of the above query −

|  |  |  |
| --- | --- | --- |
| **ID** | **NAME** | **AGE** |
| 3 | Kaushik | 23 |

## CTE from Multiple Tables

We can also create a Common Table Expression (CTE) that combines data from multiple tables by using JOIN operations within the CTE's subquery. To do this, we need to use the comma operator to separate each CTE definition, effectively merging them into a single statement.

### **Syntax**

Following is the basic syntax for multiple Common Table Expression (CTE) −

WITH

CTE\_NAME1 (column\_name) AS (query),

CTE\_NAME2 (column\_name) AS (query)

SELECT \* FROM CTE\_NAME1

UNION ALL

SELECT \* FROM CTE\_NAME2;

We can use multiple Common Table Expressions (CTEs) with various SQL operations, such as UNION, UNION ALL, JOIN, INTERSECT, or EXCEPT.

### **Example**

In here, we are defining two CTEs namely 'CUSTOMERS\_IN\_DELHI' and 'CUSTOMERS\_IN\_MUMBAI' to segregate customers based on their addresses in Delhi and Mumbai. Then, we are using the UNION ALL operator to combine the results from both CTEs into a single result set, retrieving customer information from both cities.

Open Compiler

WITH

CUSTOMERS\_IN\_DELHI AS (

SELECT \* FROM CUSTOMERS WHERE ADDRESS = 'Delhi'),

CUSTOMERS\_IN\_MUMBAI AS (

SELECT \* FROM CUSTOMERS WHERE ADDRESS = 'Mumbai')

SELECT ID, NAME, ADDRESS FROM CUSTOMERS\_IN\_DELHI

UNION ALL

SELECT ID, NAME, ADDRESS FROM CUSTOMERS\_IN\_MUMBAI;

### **Output**

Output of the above query is as shown below −

|  |  |  |
| --- | --- | --- |
| **ID** | **NAME** | **ADDRESS** |
| 2 | Khilan | Delhi |
| 4 | Chaitali | Mumbai |

## Recursive CTE

A common table expression is a query that keeps referring back to its own result in a loop repeatedly until it returns an empty result.

A recursive query continually iterates across a subset of the data during its execution, and defines itself in a self-referencing manner. This self-referencing mechanism allows it to repeatedly process and expand its results until a stopping condition is met.

To make a CTE recursive, it must include a UNION ALL statement and provide a second definition of the query that utilizes the CTE itself. This allows the CTE to repeatedly reference to its own results, creating a recursive behaviour in the query.

### **Example**

Now, we are using a recursive CTE named **recursive\_cust** to retrieve data from the 'CUSTOMERS' table created above. Initially, we are selecting customers with salaries above 3000 and then recursively appending customers older than 25 to the result set using the UNION ALL operator −

Open Compiler

WITH recursive\_cust (ID, NAME, ADDRESS, AGE) AS (

SELECT ID, NAME, ADDRESS, AGE

FROM CUSTOMERS

WHERE SALARY > 3000

UNION ALL

SELECT ID, NAME, ADDRESS, AGE

FROM CUSTOMERS

WHERE AGE > 25

)

SELECT \* FROM recursive\_cust;

### **Output**

When the above query is executed, all data from the customers table whose age is greater than 25 or salary is greater than 3000 will be displayed recursively as shown below −

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **NAME** | **ADDRESS** | **AGE** |
| 4 | Chaitali | Mumbai | 25 |
| 5 | Hardik | Bhopal | 27 |
| 6 | Komal | Hyderabad | 22 |
| 7 | Muffy | Indore | 24 |
| 1 | Ramesh | Ahmedabad | 32 |
| 5 | Hardik | Bhopal | 27 |

### **Example**

In the following query, we are using a recursive CTE named **Numbers** to generate and display numbers from 1 to 5. The recursive part continually adds 1 to the previous value until it reaches 5, creating a sequence −

Open Compiler

WITH RECURSIVE Numbers AS (

SELECT 1 AS N

UNION ALL

SELECT N + 1 FROM Numbers WHERE N < 5

)

SELECT n FROM Numbers;

### **Output**

After executing the above query, we get the following output −

|  |
| --- |
| **N** |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

## Advantages of CTE

Following are the advantages of the CTE −

* CTE makes the code maintenance easier.
* It increases the readability of the code.
* It increases the performance of the query.
* CTE allows for the simple implementation of recursive queries.

## Disadvantages of CTE

Following are the disadvantages of the CTE −

* CTE can only be referenced once by the recursive member.
* We cannot use the table variables and CTEs as parameters in a stored procedure.
* A CTE can be used in place of a view, but a CTE cannot be nested while views can.

Nested views-

#### Create the first (base) view:

sql

CopyEdit

CREATE VIEW it\_employees AS

SELECT emp\_id, emp\_name, salary

FROM employees

WHERE department = 'IT';

#### 3. Create the nested view (second-level view):

sql

CopyEdit

CREATE VIEW high\_paid\_it\_employees AS

SELECT emp\_id, emp\_name

FROM it\_employees

WHERE salary > 55000;

#### 4. Query the nested view:

sql

CopyEdit

SELECT \* FROM high\_paid\_it\_employees;

**Output:**

markdown

CopyEdit

emp\_id | emp\_name

-------------------

1 | John

### Notes:

* You can **nest views multiple levels**, but MySQL has a limit on view nesting depth (default is 62).
* Views must not contain certain clauses (like LIMIT, ORDER BY outside subqueries).
* Views are **not materialized** (unless using materialized views in other DBMS like PostgreSQL/Oracle).