<https://chatgpt.com/share/674cdfde-c2e4-8009-8959-7b5263243361>

A **Common Table Expression (CTE)** in SQL is a temporary result set defined within a WITH clause that can be referenced within the main SQL query. CTEs make complex queries easier to read and maintain by breaking them into logical parts.

### Key Characteristics of a CTE:

1. Declared with the WITH keyword.
2. It only exists for the duration of the query.
3. Improves query readability and maintainability.
4. Supports recursion.

### Syntax of a CTE:

sql

Copy code

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

AS (

-- CTE Query

SELECT column1, column2, ...

FROM table\_name

WHERE condition

)

-- Main query

SELECT column1, column2

FROM cte\_name;

### ****Example 1: Basic CTE****

Suppose we have a table employees:

| employee\_id | name | department | salary |
| --- | --- | --- | --- |
| 1 | Alice | HR | 5000 |
| 2 | Bob | IT | 6000 |
| 3 | Charlie | HR | 7000 |
| 4 | David | IT | 8000 |

Goal: Retrieve employees from the HR department with their salaries.

**Query using CTE:**

sql

Copy code

WITH hr\_employees AS (

SELECT employee\_id, name, salary

FROM employees

WHERE department = 'HR'

)

SELECT \*

FROM hr\_employees;

**Output:**

| employee\_id | name | salary |
| --- | --- | --- |
| 1 | Alice | 5000 |
| 3 | Charlie | 7000 |

### ****Example 2: Recursive CTE****

Recursive CTEs are useful for hierarchical or iterative data, such as finding all levels of a reporting structure.

Suppose we have a table organization:

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |
| 2 | Bob | 1 |
| 3 | Charlie | 1 |
| 4 | David | 2 |

Goal: Find the reporting hierarchy starting from Alice (employee\_id = 1).

**Query using Recursive CTE:**

sql

Copy code

WITH RECURSIVE hierarchy AS (

-- Anchor member

SELECT employee\_id, name, manager\_id

FROM organization

WHERE employee\_id = 1

UNION ALL

-- Recursive member

SELECT o.employee\_id, o.name, o.manager\_id

FROM organization o

INNER JOIN hierarchy h ON o.manager\_id = h.employee\_id

)

SELECT \*

FROM hierarchy;

**Output:**

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |
| 2 | Bob | 1 |
| 3 | Charlie | 1 |
| 4 | David | 2 |

### ****Example 3: Multiple CTEs****

You can define multiple CTEs in one query by separating them with commas.

Goal: Calculate the total salaries by department and list the departments with total salaries above 10,000.

**Query:**

sql

Copy code

WITH department\_salaries AS (

SELECT department, SUM(salary) AS total\_salary

FROM employees

GROUP BY department

),

high\_salary\_departments AS (

SELECT department, total\_salary

FROM department\_salaries

WHERE total\_salary > 10000

)

SELECT \*

FROM high\_salary\_departments;

**Output:**

| department | total\_salary |
| --- | --- |
| IT | 14000 |

### Benefits of Using CTE:

1. **Readability**: Easier to divide complex logic into smaller, logical sections.
2. **Reuse**: Helps avoid duplicate code.
3. **Recursive Logic**: Great for hierarchical data processing.

**Part2 – Recursive CTE**

**The goal is to find all employees reporting to Alice directly or indirectly. This includes both direct reports (like Bob and Charlie) and indirect reports (like David, who reports to Bob).**

### Key Concepts in the Query:

1. **Recursive CTE**:
   * Starts with an **anchor member** (base case).
   * Iteratively processes additional rows using the **recursive member** until no more rows can be added.
2. **Anchor Member**:

sql

Copy code

SELECT employee\_id, name, manager\_id

FROM organization

WHERE employee\_id = 1

* + This picks Alice, the starting point of the hierarchy (employee\_id = 1).

**Result of this step:**

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |

1. **Recursive Member**:

sql

Copy code

SELECT o.employee\_id, o.name, o.manager\_id

FROM organization o

INNER JOIN hierarchy h ON o.manager\_id = h.employee\_id

* + Finds employees (o) whose manager\_id matches the employee\_id in the current hierarchy (h).
  + These are Alice's direct reports and their reports, iterating step by step.

### Execution Flow:

The Recursive CTE processes data **iteratively**:

#### Iteration 1: Start with the Anchor Member

* **Anchor**: Select Alice (employee\_id = 1).
* **Hierarchy so far**:

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |

#### Iteration 2: Find Alice's Direct Reports

* Recursive step finds rows where manager\_id = 1 (Alice’s employee\_id).
* **Direct Reports**: Bob and Charlie.
* **Hierarchy so far**:

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |
| 2 | Bob | 1 |
| 3 | Charlie | 1 |

#### Iteration 3: Find Bob's Direct Reports

* Recursive step finds rows where manager\_id = 2 (Bob's employee\_id).
* **Direct Report**: David.
* **Hierarchy so far**:

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |
| 2 | Bob | 1 |
| 3 | Charlie | 1 |
| 4 | David | 2 |

#### Iteration 4: No More Reports

* No rows have manager\_id = 3 (Charlie’s employee\_id) or manager\_id = 4 (David’s employee\_id).
* The recursion stops here.

### Why Does the Output Match the Original Data?

The table represents a complete organizational hierarchy with Alice at the top. Since the Recursive CTE starts with Alice and explores all her direct and indirect reports, the output includes **every row** from the organization table because all employees eventually report to Alice.

### Why Use Recursive CTE Then?

1. **Flexibility**: You can dynamically change the starting point (employee\_id) to explore hierarchies for any employee.
2. **Efficient for Subset Hierarchies**: If the table had more rows unrelated to Alice, the Recursive CTE would only fetch Alice's hierarchy, not unrelated rows.
3. **Hierarchical Analysis**: Recursive CTEs work for more complex hierarchies (e.g., multi-level trees, paths, or partial structures).

**Part-3**

**i cant understand because i dont the concepts you are talking what is hierarchy, anchor member, recursive member and this table contains only employee\_id who are all managers right?**

###### ChatGPT said:

Got it! Let’s clarify the concepts first and break it down from the very basics. I'll also explain what the table represents and what terms like "hierarchy," "anchor member," and "recursive member" mean in simpler terms.

### 1. ****What Does the Table Represent?****

The table contains information about employees and their managers:

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |
| 2 | Bob | 1 |
| 3 | Charlie | 1 |
| 4 | David | 2 |

* + **employee\_id**: A unique ID for each employee.
  + **name**: The name of the employee.
  + **manager\_id**: The employee\_id of the employee's manager.
    - For example, Bob (employee\_id = 2) has a manager\_id = 1, meaning Alice is Bob's manager.

**Special case**:

* + If manager\_id = NULL, the employee has no manager and is at the top of the hierarchy (in this case, Alice).

### 2. ****What is a Hierarchy?****

A **hierarchy** shows relationships between people or things, usually in levels. In this table:

* + Alice is at the top (she has no manager).
  + Bob and Charlie report to Alice.
  + David reports to Bob.

This is a **tree-like structure**:

Copy code

Alice

├── Bob

│ └── David

└── Charlie

The goal of the query is to find **all employees under Alice's management**, whether they report to her directly or indirectly.

### 3. ****What are Anchor Member and Recursive Member?****

#### ****Anchor Member****:

This is where the query starts. In our case, it begins with Alice because we want to find all employees reporting to her. The anchor query fetches Alice's details:

sql

Copy code

SELECT employee\_id, name, manager\_id

FROM organization

WHERE employee\_id = 1;

**Result:**

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |

#### ****Recursive Member****:

This is the part of the query that repeatedly looks for employees reporting to the current level of employees. It uses a JOIN to find:

* + Who reports to Alice (directly).
  + Then who reports to those employees (indirectly), and so on.

For example:

* + Alice (employee\_id = 1) has manager\_id = 1 for Bob and Charlie.
  + Bob (employee\_id = 2) has manager\_id = 2 for David.

The recursive query keeps running until no more employees are found.

### 4. ****Does the Table Only Contain Managers?****

No, the table contains **all employees**. However:

* + If an employee appears as a manager\_id, it means they are managing someone.
  + If an employee does not appear as a manager\_id, they are not a manager (e.g., David only has a manager\_id but is not listed as a manager\_id for anyone else).

### 5. ****How the Query Works Step-by-Step****

* + **Start with the Anchor Member:** Fetch Alice (employee\_id = 1).
  + **Recursive Step 1:** Find who reports to Alice (manager\_id = 1).
    - Result: Bob and Charlie.
  + **Recursive Step 2:** Find who reports to Bob (manager\_id = 2).
    - Result: David.
  + **Recursive Step 3:** Check if anyone reports to David (manager\_id = 4).
    - No result, so the recursion stops.

### Final Query Output:

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |
| 2 | Bob | 1 |
| 3 | Charlie | 1 |
| 4 | David | 2 |

This shows Alice and all employees under her hierarchy (both direct and indirect reports).

### Why Does It Match the Original Table?

* + In this example, Alice is the top-level manager, and every employee in the table is part of her hierarchy.
  + If there were other managers unrelated to Alice, they wouldn’t appear in the result.

**give me some other examples for recursive cte**

###### ChatGPT said:

Here are a few more examples to help you understand **Recursive CTEs** in different contexts. Each example focuses on solving a problem where recursion is necessary.

### ****1. Find All Subordinate Employees****

Suppose we have a table employees:

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 1 | Alice | NULL |
| 2 | Bob | 1 |
| 3 | Charlie | 1 |
| 4 | David | 2 |
| 5 | Eve | 3 |

**Goal:** Find all subordinates under Bob (employee\_id = 2).

**Query:**

sql

Copy code

WITH RECURSIVE subordinates AS (

-- Anchor: Start with Bob

SELECT employee\_id, name, manager\_id

FROM employees

WHERE employee\_id = 2

UNION ALL

-- Recursive: Find employees managed by current members

SELECT e.employee\_id, e.name, e.manager\_id

FROM employees e

INNER JOIN subordinates s ON e.manager\_id = s.employee\_id

)

SELECT \*

FROM subordinates;

**Output:**

| employee\_id | name | manager\_id |
| --- | --- | --- |
| 2 | Bob | 1 |
| 4 | David | 2 |

### ****2. Calculate Factorial Using Recursive CTE****

You can use a Recursive CTE to calculate a factorial of a number. For example, factorial of 5 (5! = 5 \* 4 \* 3 \* 2 \* 1).

**Query:**

sql

Copy code

WITH RECURSIVE factorial\_cte AS (

-- Anchor: Start with 1 as the base case

SELECT 1 AS n, 1 AS factorial

UNION ALL

-- Recursive: Multiply the current number by the previous factorial

SELECT n + 1, factorial \* (n + 1)

FROM factorial\_cte

WHERE n < 5

)

SELECT \*

FROM factorial\_cte;

**Output:**

| n | factorial |
| --- | --- |
| 1 | 1 |
| 2 | 2 |
| 3 | 6 |
| 4 | 24 |
| 5 | 120 |

### ****3. Parse a Tree Structure****

Suppose you have a categories table representing a product hierarchy:

| category\_id | category\_name | parent\_category |
| --- | --- | --- |
| 1 | Electronics | NULL |
| 2 | Computers | 1 |
| 3 | Laptops | 2 |
| 4 | Mobile Phones | 1 |
| 5 | Accessories | 3 |

**Goal:** Get all subcategories under Electronics (category\_id = 1).

**Query:**

sql

Copy code

WITH RECURSIVE category\_tree AS (

-- Anchor: Start with Electronics

SELECT category\_id, category\_name, parent\_category

FROM categories

WHERE category\_id = 1

UNION ALL

-- Recursive: Find child categories

SELECT c.category\_id, c.category\_name, c.parent\_category

FROM categories c

INNER JOIN category\_tree t ON c.parent\_category = t.category\_id

)

SELECT \*

FROM category\_tree;

**Output:**

| category\_id | category\_name | parent\_category |
| --- | --- | --- |
| 1 | Electronics | NULL |
| 2 | Computers | 1 |
| 3 | Laptops | 2 |
| 4 | Mobile Phones | 1 |
| 5 | Accessories | 3 |

### ****4. Generate a Sequence of Numbers****

Sometimes, you might need to generate a sequence of numbers dynamically.

**Goal:** Generate numbers from 1 to 10.

**Query:**

sql

Copy code

WITH RECURSIVE sequence AS (

-- Anchor: Start with 1

SELECT 1 AS number

UNION ALL

-- Recursive: Add 1 to the current number

SELECT number + 1

FROM sequence

WHERE number < 10

)

SELECT \*

FROM sequence;

**Output:**

| number |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

### ****5. Sum Numbers Recursively****

Suppose you want to calculate the sum of numbers from 1 to 5 (1 + 2 + 3 + 4 + 5).

**Query:**

sql

Copy code

WITH RECURSIVE sum\_cte AS (

-- Anchor: Start with 1

SELECT 1 AS n, 1 AS cumulative\_sum

UNION ALL

-- Recursive: Add the next number to the cumulative sum

SELECT n + 1, cumulative\_sum + (n + 1)

FROM sum\_cte

WHERE n < 5

)

SELECT \*

FROM sum\_cte;

**Output:**

| n | cumulative\_sum |
| --- | --- |
| 1 | 1 |
| 2 | 3 |
| 3 | 6 |
| 4 | 10 |
| 5 | 15 |

### Summary of Use Cases:

* 1. **Hierarchical Data**: Subordinates, categories, folders.
  2. **Mathematical Problems**: Factorials, sequences, sums.
  3. **Tree Parsing**: Parent-child relationships.
  4. **Dynamic Generation**: Number sequences or data on-the-fly.

Let me know if you’d like any of these examples explained further!

##### You said:

what is recursive means here say it crisply. I think it means running within the same table

###### ChatGPT said:

**Recursive** in this context means that the query references itself to process data iteratively, step by step, until a condition is met.

It doesn’t necessarily mean "running within the same table" but rather "building results layer by layer" (like finding children of children) using the results from the previous step.

##### You said:

so it will use its own table

###### ChatGPT said:

Yes, in **Recursive CTE**, the query uses its **own results** (from the previous step) as input for the next step. It starts with an initial result (anchor) and repeatedly applies the recursive logic to expand the result, often using the same table or dataset.

##### You said:

INNER JOIN hierarchy h ON o.manager\_id = h.employee\_id here heirarchry is a key word right ?

###### ChatGPT said:

No, hierarchy is not a keyword. It is the **name of the CTE** defined in the query.

In this case:

* 1. hierarchy refers to the CTE that is being built recursively.
  2. It acts like a temporary table that gets updated with new rows during each iteration of the recursion.

The INNER JOIN is simply using the current contents of the hierarchy CTE in that step of the recursion.