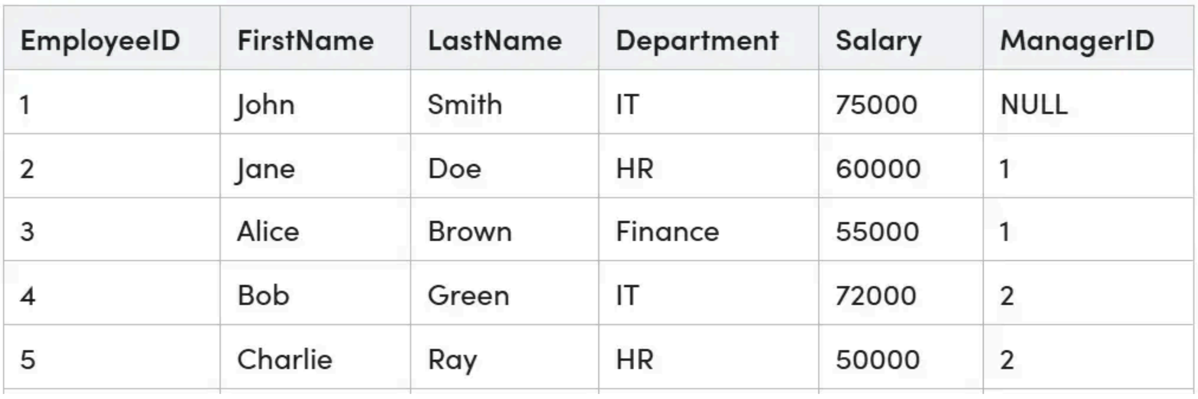
CTE in SQL

Last Updated : 17 Nov, 2025

In SQL, a Common Table Expression (CTE) is an essential tool for simplifying complex queries and making them more readable. By defining temporary result sets that can be referenced multiple times, a CTE in SQL allows developers to break down complicated logic into manageable parts.

Let’s consider an Employees table that contains employee details such as EmployeeID, Name, Department, Salary, and ManagerID. This table is used to demonstrate how to use a Common Table Expression (CTE) to simplify SQL queries, particularly when aggregating or filtering data.



EmployeeID	FirstName	LastName	Department	Salary	ManagerID
1	John	Smith	IT	75000	NULL
2	Jane	Doe	HR	60000	1
3	Alice	Brown	Finance	55000	1
4	Bob	Green	IT	72000	2
5	Charlie	Ray	HR	50000	2

Employees Table

This table represents the hierarchical structure of employees within an organization, based on a recursive Common Table Expression (CTE) query. The table displays employees, their respective levels in the hierarchy, and the managers who supervise them.

FullName	EmpLevel	Manager
John Smith	1	NULL

FullName	EmpLevel	Manager
Jane Doe	2	John Smith
Alice Brown	2	John Smith
Bob Green	3	Jane Doe
Charlie Ray	3	Jane Doe

Example: Calculate Average Salary by Department

In this example, we will use a Common Table Expression (CTE) to calculate the average salary for each department in the Employees table. The CTE simplifies the query by breaking it into a manageable part that can be referenced in the main query.

Query:

```
WITH AvgSalaryByDept AS (
    SELECT Department, AVG(Salary) AS AvgSalary
    FROM Employees
    GROUP BY Department
)
SELECT *
FROM AvgSalaryByDept;
```

Output:

Department	AvgSalary
IT	73500
HR	55000
Finance	55000

Explanation:

- The WITH clause defines a CTE named `AvgSalaryByDept`.
- The main query references this CTE to retrieve the average salary for each department.

Syntax

```
WITH cte_name AS (
    SELECT query
)
SELECT *
FROM cte_name;
```

In the above syntax:

- **cte_name**: A unique name for the CTE expression.
- **query**: A valid SQL query that returns a result set, which will be treated as a virtual table within the main query.
- **SELECT**: The main query that can reference the CTE by its name.

Uses of CTEs

- Breaking down complex queries into smaller, reusable components.
- Improving readability and modularity by separating the logic.
- Enabling recursive operations for hierarchical data.

Recursive Common Table Expression

A recursive CTE references itself and is useful for querying hierarchical data, such as employees and their managers stored in the same table. It repeatedly executes until the full hierarchy is returned. To avoid infinite loops from incorrect definitions, use the MAXRECURSION hint in the query's `OPTION` clause.

Recursive CTEs consist of two parts:

1. **Anchor member**: The initial query that selects the base case (e.g., top-level managers).
2. **Recursive member**: The query that references the CTE itself, pulling the next level of data.

Example: Hierarchical Employee Data

```
;WITH
  cteReports (EmpID, FirstName, LastName, MgrID, EmpLevel)
  AS
  (
    SELECT EmployeeID, FirstName, LastName, ManagerID, 1
    FROM Employees
    WHERE ManagerID IS NULL
    UNION ALL
    SELECT e.EmployeeID, e.FirstName, e.LastName,
e.ManagerID,
      r.EmpLevel + 1
    FROM Employees e
      INNER JOIN cteReports r
        ON e.ManagerID = r.EmpID
  )

SELECT
  FirstName + ' ' + LastName AS FullName,
  EmpLevel,
  (SELECT FirstName + ' ' + LastName FROM Employees
    WHERE EmployeeID = cteReports.MgrID) AS Manager
FROM cteReports
ORDER BY EmpLevel, MgrID
```

Output:

FullName	EmpLevel	Manager
John Smith	1	NULL
Jane Doe	2	John Smith
Alice Brown	2	John Smith
Bob Green	3	Jane Doe

FullName	EmpLevel	Manager
Charlie Ray	3	Jane Doe

Explanation:

- John Smith is at level 1 and has no manager (Top-level employee).
- Jane Doe and Alice Brown are at level 2, reporting to John Smith.
- Bob Green and Charlie Ray are at level 3, reporting to Jane Doe.

Limitations of CTEs in SQL

- **Temporary Scope**: A CTE exists only during the execution of the query. Once the query completes, the CTE is discarded.
- **Performance Issues**: For very large datasets, CTEs can sometimes lead to performance degradation due to multiple references to the same CTE.
- **Not Allowed in All Database Operations**: Some operations, such as INSERT and UPDATE, may have restrictions when using CTEs in certain databases.

CTE Vs Subqueries

CTE	Subquery
Can be referenced multiple times.	Typically used once.
Improves readability for complex queries.	Can become difficult to read when nested.
Optimized for multiple references.	May be less efficient for repeated operations.