**1. What is a CTE (Common Table Expression)?**

A **CTE (Common Table Expression)** is a **temporary, named result set** that exists **only during the execution** of a single SQL statement.

It’s like creating a **temporary view** inside your query — you can use it to simplify complex logic.

**🧠 Basic Idea**

Instead of writing:

SELECT ...

FROM (SELECT ... FROM ...) AS subquery

WHERE ...

You can write:

WITH cte\_name AS (

SELECT ... FROM ...

)

SELECT \* FROM cte\_name WHERE ...;

It improves **readability**, **maintainability**, and allows **recursion** (self-referencing).

**🧾 2. CTE Syntax**

WITH cte\_name (optional\_column\_list) AS (

-- CTE definition (a SELECT statement)

SELECT column1, column2, ...

FROM table\_name

WHERE condition

)

-- Main query using the CTE

SELECT \* FROM cte\_name;

**🧱 3. Example 1 — Simple CTE**

WITH HighSalary AS (

SELECT EmpID, Name, Salary

FROM Employee

WHERE Salary > 50000

)

SELECT Name

FROM HighSalary

WHERE Salary > 70000;

**🔍 Explanation:**

* HighSalary is a **CTE name**.
* It selects employees with salary > 50000.
* The **main query** further filters those with salary > 70000.

**🧩 4. Example 2 — Using Multiple CTEs**

You can define **more than one CTE** by separating them with commas.

WITH

Sales AS (

SELECT EmpID, SUM(SaleAmount) AS TotalSales

FROM Orders

GROUP BY EmpID

),

HighPerformers AS (

SELECT EmpID, TotalSales

FROM Sales

WHERE TotalSales > 100000

)

SELECT E.Name, H.TotalSales

FROM HighPerformers H

JOIN Employee E ON H.EmpID = E.EmpID;

✅ Helps you **break complex logic into steps** — much cleaner than nested subqueries.

**🔁 5. Recursive CTE (Advanced)**

A **Recursive CTE** calls itself — used for **hierarchical data**, like organization trees, categories, or parent-child relationships.

**Example: Employee Hierarchy**

WITH EmployeeHierarchy AS (

-- Anchor part (Base)

SELECT EmpID, Name, ManagerID, 1 AS Level

FROM Employee

WHERE ManagerID IS NULL

UNION ALL

-- Recursive part

SELECT E.EmpID, E.Name, E.ManagerID, H.Level + 1

FROM Employee E

INNER JOIN EmployeeHierarchy H

ON E.ManagerID = H.EmpID

)

SELECT \* FROM EmployeeHierarchy;

**🔍 Explanation:**

* The **anchor part** selects top-level managers (ManagerID = NULL).
* The **recursive part** repeatedly finds subordinates for each manager.
* The process repeats until no more rows are found.

**⚙️ 6. Key Features of CTE**

| **Feature** | **Description** |
| --- | --- |
| **Temporary** | Exists only during that query execution |
| **Readable** | Simplifies nested subqueries |
| **Recursive Capable** | Can reference itself |
| **Reusable** | Can be referenced multiple times in the same query |
| **No Index Needed** | Works on logical data, not physical storage |

**🧮 7. CTE vs Subquery vs View**

| **Feature** | **CTE** | **Subquery** | **View** |
| --- | --- | --- | --- |
| Scope | Only within that query | Only within its parent query | Persistent (stored in DB) |
| Recursive | ✅ Yes | ❌ No | ✅ (in some DBs) |
| Performance | Similar | Similar | Cached if materialized |
| Readability | High | Low (nested) | High |
| Storage | None | None | Stored |

**📊 8. Example — CTE for Aggregation**

WITH DeptSalary AS (

SELECT DeptID, AVG(Salary) AS AvgSalary

FROM Employee

GROUP BY DeptID

)

SELECT D.DeptName, DS.AvgSalary

FROM DeptSalary DS

JOIN Department D ON DS.DeptID = D.DeptID

ORDER BY DS.AvgSalary DESC;

✅ Calculates average salary per department, then joins with department names.

**🧠 9. Example — Recursive CTE for Numbers**

Generate a sequence of numbers (no numbers table needed):

WITH Numbers AS (

SELECT 1 AS n

UNION ALL

SELECT n + 1

FROM Numbers

WHERE n < 10

)

SELECT \* FROM Numbers;

Output:

n

--

1

2

3

...

10

**🔐 10. Restrictions on CTE**

* CTE must be followed by a **SELECT / INSERT / UPDATE / DELETE**.
* A CTE **cannot be indexed** directly.
* Recursive CTE must have:
  + **Anchor part**
  + **Recursive part**
  + **Termination condition** (e.g., WHERE n < 10)
* Some DBMS (like MySQL) support CTEs **from version 8.0** onward.

**🧰 11. Real-Life Use Case**

**Problem:**

Find all employees under a particular manager — including indirect subordinates.

WITH AllSubordinates AS (

SELECT EmpID, Name, ManagerID

FROM Employee

WHERE EmpID = 1 -- Manager ID

UNION ALL

SELECT E.EmpID, E.Name, E.ManagerID

FROM Employee E

JOIN AllSubordinates S ON E.ManagerID = S.EmpID

)

SELECT \* FROM AllSubordinates;

This recursively lists everyone reporting to Manager 1, directly or indirectly.

**💬 12. Interview Questions on CTE**

| **#** | **Question** | **Answer** |
| --- | --- | --- |
| 1 | What is a CTE? | A temporary named result set defined within a query. |
| 2 | Difference between CTE and view? | View is permanent, CTE exists only for one query. |
| 3 | Can a CTE be recursive? | Yes, that’s one of its main advantages. |
| 4 | Can you use multiple CTEs in one query? | Yes, separated by commas. |
| 5 | Can CTEs improve performance? | Not always — they mainly improve readability. |
| 6 | What is the base case in a recursive CTE? | The first query (anchor member). |
| 7 | What is the termination condition in a recursive CTE? | A WHERE clause in the recursive part. |
| 8 | Can we use CTE in an UPDATE or DELETE statement? | Yes, supported in most databases. |
| 9 | Are CTEs stored physically? | No, they exist only in memory during query execution. |
| 10 | Which DBs support CTEs? | SQL Server, PostgreSQL, Oracle, MySQL 8+, SQLite 3.8+. |

**🧭 13. Summary Mind Map**

CTE (Common Table Expression)

│

├── Definition → Temporary named result set

├── Syntax → WITH cte\_name AS (SELECT ...)

├── Types

│ ├─ Non-Recursive (simple)

│ └─ Recursive (self-referencing)

├── Benefits

│ ├─ Readable queries

│ ├─ Simplifies subqueries

│ ├─ Supports recursion

│ └─ Reusable logic

├── Limitations

│ ├─ Exists only during query

│ ├─ No indexes

│ └─ Needs termination for recursion

└── Used in

├─ Hierarchies

├─ Tree structures

├─ Aggregation steps

└─ Reusable temporary sets