Introduction to CTE (Common Table Expression)

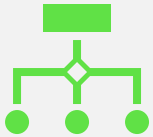
CTE is a temporary result set used within a SQL query.

Helps improve code readability and simplifies complex queries.

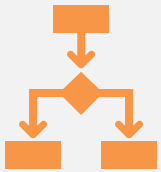
Defined using the WITH clause.

Why Use CTE?

Makes SQL queries easier to read and maintain.



Useful for recursive queries (hierarchical data like org charts).



Eliminates the need for multiple subqueries.

Basic Syntax of CTE

```
WITH CTE_Name AS (
   SELECT column1, column2 FROM table_name WHERE condition
)
SELECT * FROM CTE_Name WHERE another_condition;
```

WITH clause declares the start of a CTE.

CTE is referenced like a table in the main query.

Example - Employees Earning Above Dept Average

```
WITH DepartmentAverage AS (
   SELECT DepartmentID, AVG(Salary) AS AvgSalary FROM Employees
GROUP BY DepartmentID
)
SELECT e.* FROM Employees e
INNER JOIN DepartmentAverage d ON e.DepartmentID = d.DepartmentID
WHERE e.Salary > d.AvgSalary;
```

- CTE calculates the department's average salary.
- Main query filters employees earning above average.

Types of CTEs in SQL

1. Recursive CTE: Used for hierarchical data (e.g., employee hierarchy).

2. Non-Recursive CTE: Simplifies complex queries.

3. Inline CTE: Acts as an inline view for modular queries.

4. Multiple CTEs: Chains multiple CTEs together.

5. Materialized CTE (SQL Server 2022+): Improves performance by storing results.

Recursive CTE Example (Employee Hierarchy)

```
WITH EmployeeHierarchy AS (
   SELECT EmployeeID, EmployeeName, ManagerID, 0 AS Level FROM
Employees WHERE ManagerID IS NULL
   UNION ALL
   SELECT e.EmployeeID, e.EmployeeName, e.ManagerID, eh.Level + 1 FROM
Employees e
   INNER JOIN EmployeeHierarchy eh ON e.ManagerID = eh.EmployeeID
)
SELECT * FROM EmployeeHierarchy ORDER BY Level;
```

- Recursively finds employees reporting to a manager.

Non-Recursive CTE Example

```
WITH TotalSales AS (
   SELECT SUM(SaleAmount) AS TotalAmount FROM Sales WHERE
YEAR(SaleDate) = 2024
)
SELECT * FROM TotalSales WHERE TotalAmount > 500;
```

- Simplifies total sales calculation without using a subquery.

Benefits of Using CTEs

Improves SQL query readability.

Allows reusing query results within the same execution.

Helps manage hierarchical data efficiently.

Final Thoughts

CTEs are a powerful tool in SQL for organizing queries.

Can be used for both simple and complex data manipulations.

Enhances performance and maintainability.