# 12. Derived Tables, CTEs, and Recursive CTEs (MS SQL)

Derived tables and CTEs (Common Table Expressions) let you build intermediate results in a readable, composable way. Recursive CTEs let you traverse hierarchies like org charts or folder trees.

---

## 12.1 Derived Tables — inline subqueries in FROM

A derived table is a subquery used in the `FROM` clause, treated like a temporary table for that query.

**Example — total spent per customer**

```
SELECT dt.CustomerID, dt.TotalSpent
FROM (
    SELECT o.CustomerID, SUM(oi.Quantity * oi.UnitPrice) AS TotalSpent
    FROM dbo.Orders o
    JOIN dbo.OrderItems oi ON o.OrderID = oi.OrderID
    GROUP BY o.CustomerID
) AS dt
WHERE dt.TotalSpent > 1000;
```

- `dt` is a derived table.
- Makes complex queries modular and readable.

**Tips:**

- Always give a derived table an alias.
- Useful when the same intermediate result is used only once.

---

## 12.2 CTEs — Common Table Expressions

A CTE is a named temporary result set defined **before** the main query using `WITH`.

**Syntax:**

```
WITH CTEName AS (
    SELECT ... FROM ... WHERE ...
)
SELECT * FROM CTEName;
```

**Example — top 3 orders per customer**

```sql
WITH TopOrders AS (
    SELECT o.CustomerID, o.OrderID, o.OrderDate,
           ROW_NUMBER() OVER (PARTITION BY o.CustomerID ORDER BY o.OrderDate DESC) AS rn
    FROM dbo.Orders o
)
SELECT CustomerID, OrderID, OrderDate
FROM TopOrders
WHERE rn <= 3;
```

- Improves readability compared to deeply nested derived tables.
- Can be referenced multiple times within the same query if needed.

---

## 12.3 Recursive CTEs — traversing hierarchies

Recursive CTEs reference themselves, useful for hierarchical data.

**Syntax:**

```sql
WITH RecursiveCTE AS (
    -- Anchor member
    SELECT EmployeeID, Name, ManagerID, 0 AS Level
    FROM dbo.Employees
    WHERE ManagerID IS NULL

    UNION ALL

    -- Recursive member
    SELECT e.EmployeeID, e.Name, e.ManagerID, rc.Level + 1
    FROM dbo.Employees e
    INNER JOIN RecursiveCTE rc ON e.ManagerID = rc.EmployeeID
)
SELECT * FROM RecursiveCTE;
```

- Anchor member defines the starting point (top-level managers).
- Recursive member joins to itself to find children.
- `Level` helps track hierarchy depth.
- Termination occurs naturally when no more rows match.

**Tips:**

- Be careful of infinite recursion. SQL Server defaults to 100 recursion levels; override with `OPTION (MAXRECURSION n)`.
- Useful for org charts, bill of materials, folder trees, or graph-like data.

---

## 12.4 When to Use Derived Tables vs CTEs

| Feature | Derived Table | CTE |
|---|---|---|
| Scope | One query | Entire query following WITH |
| Readability | Moderate | High for complex queries |
| Recursion | ❌ | ☑ |
| Reusability | Only in that FROM | Can be referenced multiple times in main query |

---

## 12.5 Practical Tips

- Use derived tables for quick, one-off modularization.
- Use CTEs when readability or multiple references are needed.
- Use recursive CTEs for hierarchies; ensure proper termination conditions.
- Combine CTEs with ROW_NUMBER(), RANK(), or aggregate functions for top-N per group or hierarchy-level calculations.

---

Mastering derived tables, CTEs, and recursive CTEs opens the door to modular, readable, and powerful queries. They form the backbone for analytic SQL patterns, hierarchies, and iterative computations.