11. Subqueries — Scalar, Correlated, and Lateral (APPLY) (MS SQL)

Subqueries are queries nested inside other queries. They are tiny worlds embedded in larger ones, letting you compute values on the fly, filter based on dynamic results, or join in unconventional ways.

11.1 Scalar Subquery — one value, one place

A scalar subquery returns exactly **one value**. You can use it in the `SELECT` list, `WHERE`, or even `HAVING` clauses.

Example — latest order date per customer

```sql
SELECT c.CustomerID,
       c.Name,
       (SELECT MAX(OrderDate)
        FROM dbo.Orders o
        WHERE o.CustomerID = c.CustomerID) AS LastOrderDate
FROM dbo.Customers c;
```

- Returns a single value per outer row.
- Good for metrics like MAX, MIN, SUM, COUNT.

Tip: Scalar subqueries **must return only one row**. More than one row throws an error.

11.2 Correlated Subquery — outer-dependent

A correlated subquery references the outer query. It is evaluated **once per outer row**.

Example — customers who spent more than average

```sql
SELECT c.CustomerID, c.Name
FROM dbo.Customers c
WHERE (SELECT SUM(oi.Quantity * oi.UnitPrice)
       FROM dbo.Orders o
       JOIN dbo.OrderItems oi ON oi.OrderID = o.OrderID
       WHERE o.CustomerID = c.CustomerID) > 1000;
```

- The subquery calculates the total per customer.
- If the total exceeds 1000, the row survives.
- Correlated subqueries are often less efficient than joins but are readable for per-row calculations.

11.3 APPLY — lateral join for per-row computations

`APPLY` is like a join but allows the right-side query to reference the left-side row. Think of it as a flexible lateral join.

CROSS APPLY — only matching rows

```sql
SELECT c.CustomerID, c.Name, o.LastOrderDate
FROM dbo.Customers c
CROSS APPLY (
    SELECT TOP 1 OrderDate AS LastOrderDate
    FROM dbo.Orders o
    WHERE o.CustomerID = c.CustomerID
    ORDER BY OrderDate DESC
) o;
```

- Returns customers with at least one order.
- Great for picking top-N per group.

OUTER APPLY — include rows even if no match

```sql
SELECT c.CustomerID, c.Name, o.LastOrderDate
FROM dbo.Customers c
OUTER APPLY (
    SELECT TOP 1 OrderDate AS LastOrderDate
    FROM dbo.Orders o
    WHERE o.CustomerID = c.CustomerID
    ORDER BY OrderDate DESC
) o;
```

- Preserves customers without orders (LastOrderDate = NULL).
- Much like a LEFT JOIN but with the flexibility of per-row computation.

11.4 Practical Comparison

Type	Uses	Returns	When to Use
Scalar Subquery	SELECT, WHERE	One value	Quick computation of single metric per row
Correlated Subquery	WHERE, SELECT	Depends on outer row	Per-row calculation, dynamic filter
CROSS APPLY	SELECT	Rows only if match exists	Top-N per group, inline computations
OUTER APPLY	SELECT	All outer rows	Top-N per group, preserving unmatched rows

---

11.5 Tips and Pitfalls

- Scalar subqueries must be limited to one row — otherwise you get an error.
- Correlated subqueries can be slow on large datasets; consider joins or APPLY if performance suffers.
- APPLY shines when you need the “per-row subquery” power of a correlated subquery **but want join-like syntax**.
- Always check execution plans: APPLY can sometimes be more efficient than multiple correlated subqueries.

Subqueries allow SQL to be expressive and per-row aware without leaving the set-based mindset. They are your toolkit for embedding logic, dynamic filtering, and computing derived metrics on-the-fly.

Next up: **12. Derived tables, CTEs, and recursive CTEs**.