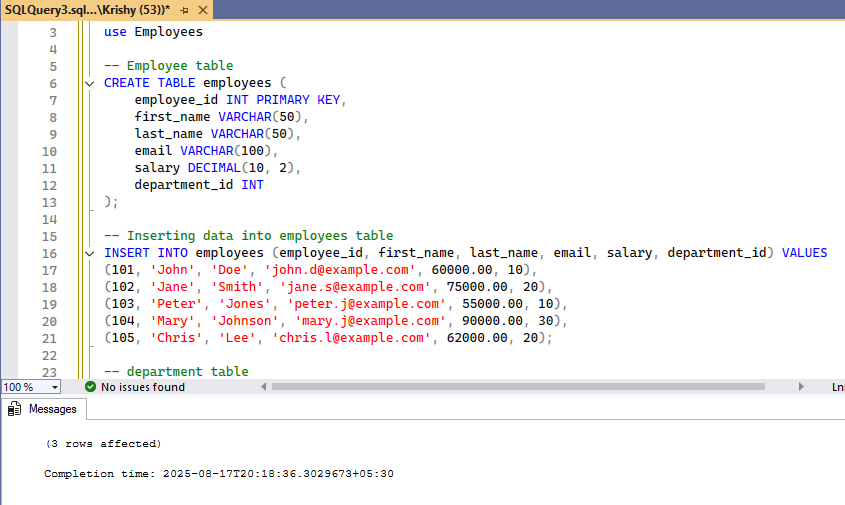
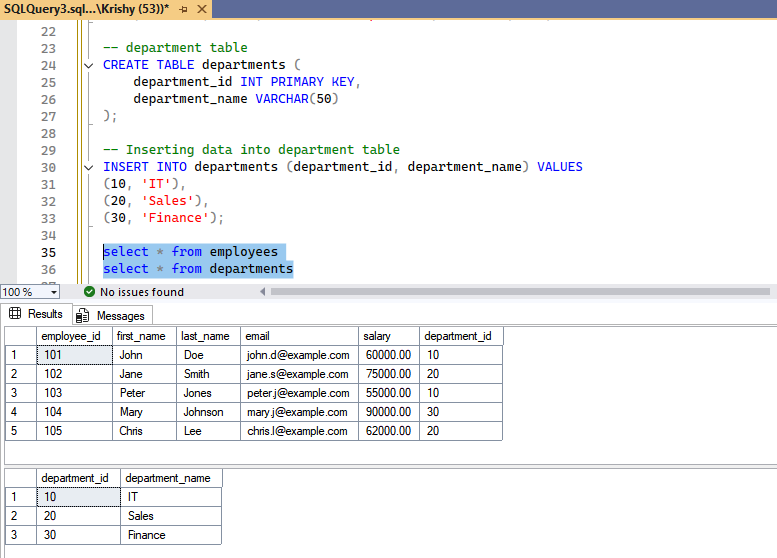
**SUBQUERIES AND NESTED QUERIES**

**1. Create table and insert values**

Firstly, Created the table Employees and Departments as we are going to see about Subqueries and Nested queries..



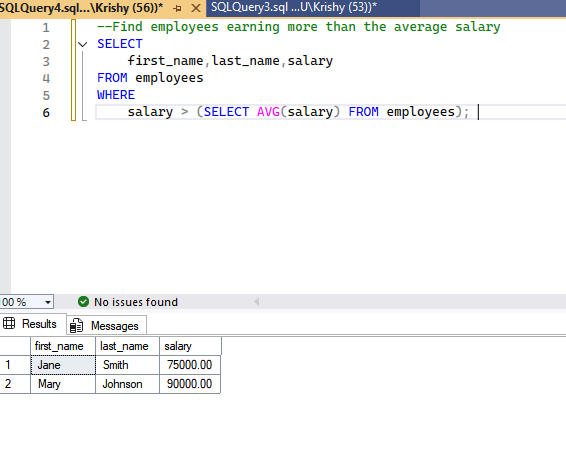


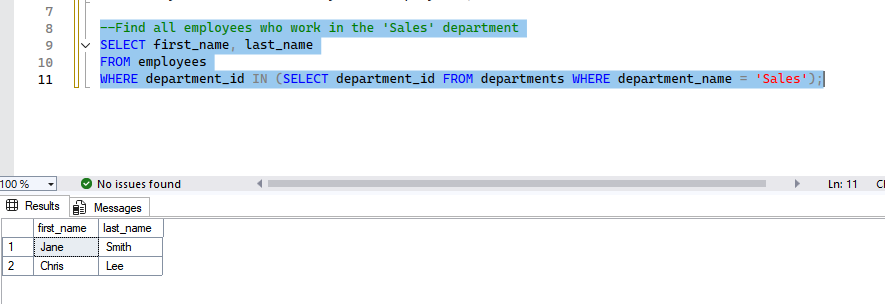
**SubQuery**

A subquery is a SELECT statement that is embedded within another SQL statement (e.g., SELECT, INSERT, UPDATE, DELETE). It is executed first, and its result is then used by the outer query. Subqueries can be used in the SELECT clause, WHERE clause, and FROM clause.

**Uses**

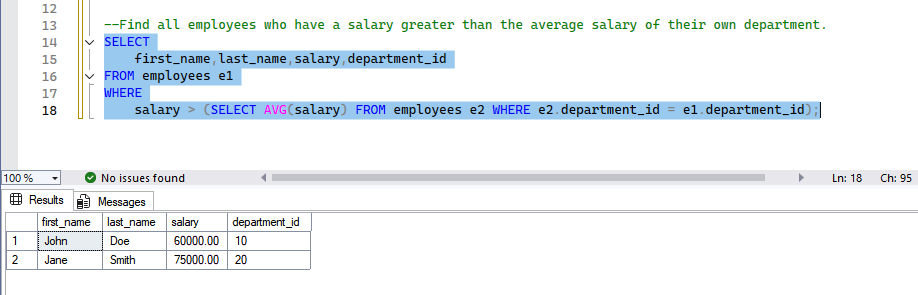
* Filter results
* Compute values
* Create virtual tables





**Correlated subquery**

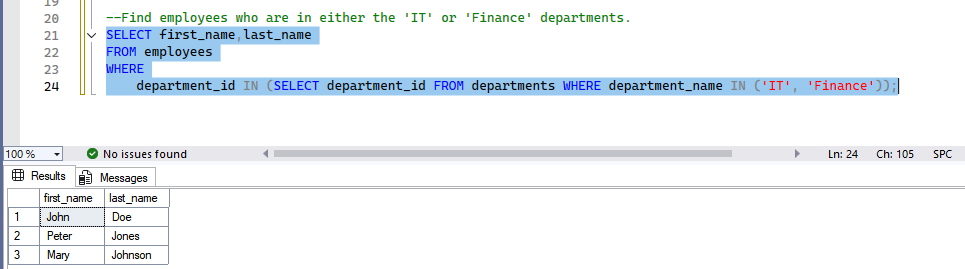
A correlated subquery is a subquery that references a column from the outer query. It is executed once for each row processed by the outer query.



For each employee (e1), the inner query calculates the average salary only for that employee's department (e2.department\_id = e1.department\_id).

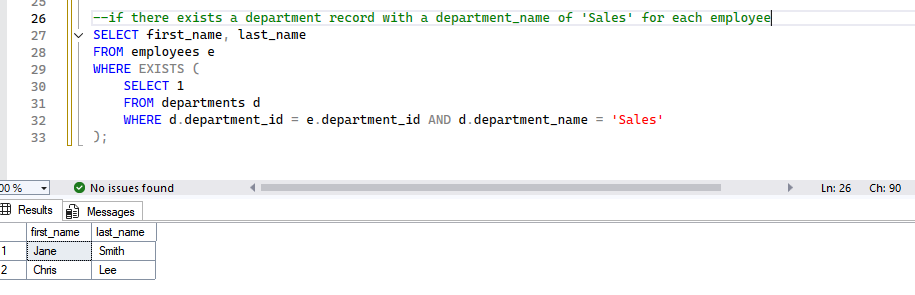
**Subquery returning multiple rows**

Subqueries can return multiple rows. When they do, they must be used with operators that can handle a set of values, such as IN, NOT IN, ANY, or ALL.



**Exists**

EXISTS is a logical operator that checks for the existence of any rows returned by a subquery. It returns TRUE if the subquery returns at least one row, and FALSE if it returns none. It is often more performant than IN because it stops searching as soon as it finds the first match.



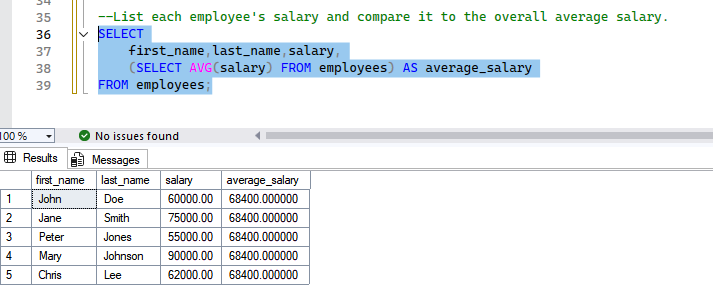
This query checks if there exists a department record with a department\_name of 'Sales' for each employee. The SELECT 1 is just a convention; the EXISTS operator doesn't care about the returned columns, only whether a row exists.

### Performance affected by subqueries?

* **Non-Correlated Subqueries:** The inner query runs once and its result is cached for the outer query. Performance is generally good.
* **Correlated Subqueries:** These can be very slow because the inner query runs for every single row of the outer query. For large datasets, a JOIN is almost always a better, more performant alternative. Modern database optimizers can sometimes rewrite a correlated subquery into a join, but this is not guaranteed.

**Scalar subquery**

A scalar subquery is a subquery that returns a single value (one row and one column). They can be used anywhere a single value is expected, such as in the SELECT list, WHERE clause, or a SET clause of an UPDATE statement.



The subquery in the SELECT list calculates the average salary once and applies it to every row in the result set.

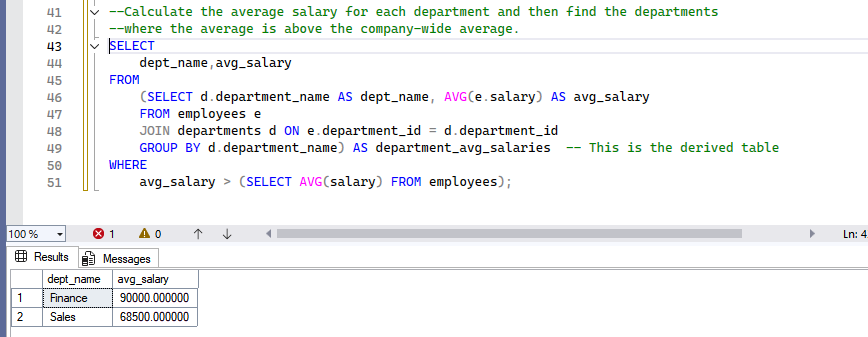
**How can be Subqueries used**

Subqueries can be used in several places within an SQL statement:

* **SELECT clause:** As a scalar value.
* **FROM clause:** To create a "derived table" (a temporary result set).
* **WHERE clause:** To filter data using operators like IN, NOT IN, =, <, >, EXISTS, NOT EXISTS.
* **HAVING clause:** To filter groups based on an aggregate condition.

**Subqueries in FROM clause**

Subquery can be used in the FROM clause. The result of the subquery is treated as a temporary, inline table, also known as a **derived table** or **inline view**.



**Derived table**

A derived table is the name given to a subquery that is used in the FROM clause. It is a temporary, in-memory table created on the fly during the execution of a query. The derived table must have an alias, as shown in the example above (AS department\_avg\_salaries).

**Key Characteristics:**

* It's a temporary, non-permanent result set.
* It must have an alias.
* It is useful for breaking down complex problems into smaller, more manageable steps.
* Can sometimes be a good alternative to a CTE (Common Table Expression).

**Summary (When to Use What):**

* **Scalar Subquery** → return single value (SELECT, WHERE).
* **Correlated Subquery** → when filtering based on outer query row.
* **IN / EXISTS** → when multiple rows possible.
* **FROM (Derived Table)** → treat subquery as a temporary table.

Thank you!