SQL Notes: Subqueries, CTEs, and Window Functions

# 1. Subqueries and Their Types

🔹 What is a Subquery?  
A subquery is a query nested inside another query. It is used to return data that will be used by the main (outer) query. Also called inner query or nested query. Enclosed within parentheses `()`. Can be used with SELECT, INSERT, UPDATE, or DELETE.

🔸 Types of Subqueries

1. Single-Row Subquery  
Returns only one row of data.  
Example:  
SELECT name, salary   
FROM employees   
WHERE salary > (  
 SELECT AVG(salary)   
 FROM employees  
);

2. Multiple-Row Subquery  
Returns more than one row.  
Example:  
SELECT name   
FROM employees   
WHERE department\_id IN (  
 SELECT department\_id   
 FROM departments   
 WHERE location = 'New York'  
);

3. Correlated Subquery  
A subquery that references columns from the outer query. Executed once for every row of the outer query.  
Example:  
SELECT name   
FROM employees e  
WHERE salary > (  
 SELECT AVG(salary)   
 FROM employees   
 WHERE department\_id = e.department\_id  
);

## ****What does "references columns from the outer query" mean?****

In a **correlated subquery**, the **inner query uses values from the current row of the outer query**. So, the **inner query is dependent on the outer query** — it runs **once for each row** of the outer query.

## Example:

SELECT name

FROM employees e

WHERE salary > (

SELECT AVG(salary)

FROM employees

WHERE department\_id = e.department\_id

);

### Explanation step-by-step:

Let’s say we have this employees table:

| **name** | **salary** | **department\_id** |
| --- | --- | --- |
| Alice | 90000 | 10 |
| Bob | 60000 | 10 |
| Carol | 95000 | 20 |
| Dave | 70000 | 20 |

Now let’s walk through the query **row by row**:

### 🔹 Step 1: Outer query picks a row, say Alice (department\_id = 10)

SELECT AVG(salary) FROM employees WHERE department\_id = 10

This inner query will return:  
AVG(90000, 60000) = 75000

Then, outer query checks:  
**Is Alice’s salary (90000) > 75000?** ✅ Yes → Include Alice

### 🔹 Step 2: Next row is Bob (also department\_id = 10)

Inner query again runs:

SELECT AVG(salary) FROM employees WHERE department\_id = 10

Still 75000  
**Is Bob’s salary (60000) > 75000?** ❌ No → Exclude Bob

### 🔹 Step 3: Next row is Carol (department\_id = 20)

Inner query:

SELECT AVG(salary) FROM employees WHERE department\_id = 20

AVG(95000, 70000) = 82500  
**Is Carol’s salary (95000) > 82500?** ✅ Yes → Include Carol

### 🔹 Step 4: Next row is Dave (department\_id = 20)

Same inner query as above.  
**Is Dave’s salary (70000) > 82500?** ❌ No → Exclude Dave

## So, what’s happening?

* The **inner query uses e.department\_id**, which is from the **outer query row**.
* Because of this **dependency**, the inner query has to run for **each outer row**.
* This is why it's called a **correlated subquery** – it’s **linked** (or "correlated") to each row of the outer query.

**4. Nested Subquery**  
A subquery inside another subquery.  
Example:  
SELECT name   
FROM employees   
WHERE department\_id = (  
 SELECT department\_id   
 FROM departments   
 WHERE location = (  
 SELECT location   
 FROM locations   
 WHERE city = 'Berlin'  
 )  
);

# 2. Common Table Expressions (CTEs)

🔹 What is a CTE?  
A Common Table Expression is a temporary result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. Defined using WITH clause. Improves readability and reusability. Especially useful for recursive queries.

🔸 Syntax:  
WITH CTE\_name AS (  
 SELECT column1, column2  
 FROM table\_name  
 WHERE condition  
)  
SELECT \*  
FROM CTE\_name  
WHERE column1 = 'value';

🔸 Example:  
WITH high\_earners AS (  
 SELECT name, salary   
 FROM employees   
 WHERE salary > 80000  
)  
SELECT name   
FROM high\_earners  
WHERE name LIKE 'A%';

🔸 Recursive CTE Example:  
To generate numbers from 1 to 5:  
WITH RECURSIVE numbers AS (  
 SELECT 1 AS num  
 UNION ALL  
 SELECT num + 1 FROM numbers WHERE num < 5  
)  
SELECT \* FROM numbers;

# 3. Important Window Functions and Their Usage

🔹 What is a Window Function?  
A window function performs a calculation across a set of rows related to the current row — without collapsing rows into a group. Used with OVER() clause. Different from aggregation functions like SUM() or AVG() used with GROUP BY.

🔸 Common Window Functions

ROW\_NUMBER() - Assigns unique row numbers  
RANK() - Gives rank with gaps in case of tie  
DENSE\_RANK() - Gives rank without gaps  
NTILE(n) - Divides rows into n equal parts  
LAG() - Accesses data of the previous row  
LEAD() - Accesses data of the next row  
SUM()/AVG() - Performs running totals or moving averages

🔸 Syntax:  
SELECT name, salary,  
 RANK() OVER (ORDER BY salary DESC) AS salary\_rank  
FROM employees;

🔸 Partitioning and Ordering  
PARTITION BY: Divides the dataset into groups  
ORDER BY: Specifies the order within each partition

Example:  
SELECT name, department\_id, salary,  
 RANK() OVER (  
 PARTITION BY department\_id  
 ORDER BY salary DESC  
 ) AS dept\_rank  
FROM employees;