**Day 11-12: Subqueries & Common Table Expressions (CTEs)**

**Subtopic 1: Subqueries (Nested Queries)**

* **Definition:** A subquery is a complete SELECT query that is nested inside another SQL query. The outer query uses the result of the inner query (the subquery) to determine its own results.
* **Use/Importance:** Subqueries allow you to break down a complex problem into sequential steps within a single query. They are powerful for filtering data based on the results of an aggregate calculation or another table's data. They can be used in WHERE, FROM, and SELECT clauses.
* **Syntax/Structure:**
  + In a WHERE clause: ... WHERE column\_name IN (SELECT column\_name FROM ...)
  + In a FROM clause (as a derived table): ... FROM (SELECT column1, column2 FROM ...) AS alias
  + In a SELECT clause (as a scalar subquery): SELECT column1, (SELECT MAX(column2) FROM ...) AS max\_val FROM ...
* **Example:** Find all employees who work in the same department as an employee named 'Priya'.

SELECT employee\_name, department\_id

FROM employees

WHERE department\_id IN (

SELECT department\_id

FROM employees

WHERE employee\_name = 'Priya'

);

**Subtopic 2: Common Table Expressions (CTEs)**

* **Definition:** A Common Table Expression, or CTE, is a temporary, named result set that you can reference within a subsequent SELECT, INSERT, UPDATE, or DELETE statement. It is defined using a WITH clause.
* **Use/Importance:** CTEs are the modern standard for writing complex queries. Their primary benefit is **readability**. They break down long, nested queries into logical, sequential, and reusable blocks, making them far easier to write, debug, and understand later.
* **Syntax/Structure:**

WITH CteName AS (

-- The query that defines the CTE

SELECT column1, column2

FROM some\_table

WHERE condition

)

-- The main query that uses the CTE

SELECT \*

FROM CteName;

* **Example:** Define a CTE that contains all sales from 'India', and then query it.

WITH IndiaSales AS (

SELECT order\_id, sale\_amount

FROM sales

WHERE country = 'India'

)

SELECT AVG(sale\_amount) as average\_india\_sale

FROM IndiaSales;

**Subtopic 3: Applying CTEs to Complex Problems**

* **Definition:** This involves using a CTE to pre-calculate an aggregate value (like a group average) and then joining that result back to the original table to perform row-by-row comparisons.
* **Use/Importance:** This pattern is extremely common and powerful. It cleanly solves problems like "find items above the average" or "compare an individual's performance to their team's average". Trying to solve this with only a subquery can be complex and less readable.
* **Syntax/Structure:**

SQL

WITH Aggregates AS (

SELECT grouping\_column, AVG(value\_column) as avg\_value

FROM table\_name

GROUP BY grouping\_column

)

SELECT t.\*

FROM table\_name AS t

JOIN Aggregates AS a ON t.grouping\_column = a.grouping\_column

WHERE t.value\_column > a.avg\_value;

* **Example:** Find all employees who earn a salary above the average salary of their respective departments.

SQL

WITH DepartmentAverages AS (

SELECT department\_id, AVG(salary) as avg\_dept\_salary

FROM employees

GROUP BY department\_id

)

SELECT

e.employee\_name,

e.salary,

da.avg\_dept\_salary

FROM

employees AS e

JOIN

DepartmentAverages AS da ON e.department\_id = da.department\_id

WHERE

e.salary > da.avg\_dept\_salary;

**Quick Recap:**

* A **Subquery** is a query inside another query; they can make SQL hard to read if nested too deeply.
* A **CTE** (WITH clause) is a named, temporary result set that acts like a temporary table.
* CTEs are the preferred method for complex queries because they are highly **readable** and allow you to structure your logic in clear, sequential steps.
* A common and powerful pattern is to create a CTE with aggregated data and then JOIN it back to the main table for comparisons.

**Practice Tasks:**

1. **Task 1 (Rewrite):** The following query finds the names of customers who have placed orders totaling more than $1,000. Rewrite it using a CTE.

SQL

SELECT customer\_name

FROM customers

WHERE customer\_id IN (

SELECT customer\_id

FROM orders

GROUP BY customer\_id

HAVING SUM(order\_total) > 1000

);

1. **Task 2 (Application):** From a products table with category and price columns, find all products that have a price higher than the average price of all products in their same category. Use a CTE.
2. **Task 3 (Derived Table vs CTE):** You have a sales table with salesperson\_id and sale\_amount. Write a query to first calculate the total sales for each salesperson, and then from that result, find the average of those total sales amounts. Solve it once using a subquery in the FROM clause, and then rewrite it using a CTE.
3. **Task 4 (Challenge - Chaining CTEs):** Using an orders table (with customer\_id, order\_date, order\_total), create a multi-step query with CTEs:
   * **CTE 1 (HighValueCustomers):** Find the customer\_id of all customers whose total spending (sum of order\_total) is over $5,000.
   * **CTE 2 (RecentOrders):** Find the details of all orders placed after September 1st, 2025.
   * **Final Query:** JOIN the two CTEs to find the recent orders that were placed by high-value customers.