```
/*
    ==============================
    == SubQueries Documentation ==
    ==============================

    What is a Subquery?
    - A subquery is a SQL query nested inside another query (outer query).
    - SQL first runs the inner query → gets a result → then uses that result in the
outer query.
    - Subqueries help you filter, calculate, and validate data before returning
results.

    Types of Subqueries:
    - Scalar Subquery: Returns a single value (e.g., AVG, MAX, MIN).
        Example: score > (SELECT AVG(score) FROM customers)
    - IN / NOT IN Subquery: Returns multiple values for filtering.
        Example: id IN (SELECT customer_id FROM orders)
    - EXISTS Subquery: Checks if at least one row exists.
        Example: WHERE EXISTS (SELECT 1 FROM orders WHERE orders.customer_id =
customers.id)
    - Subquery in SELECT: Adds a calculated value per row.
        Example: SELECT first_name, (SELECT AVG(score) FROM customers) AS avg_score
    - Subquery in FROM / Derived Table: Used as a temporary table for aggregation.
        Example: SELECT AVG(total_sales) FROM (SELECT SUM(sales) AS total_sales FROM
orders GROUP BY customer_id) AS temp
    - Nested Subquery: A subquery inside another subquery.
        Example: score > (SELECT AVG(score) FROM customers WHERE id IN (SELECT
customer_id FROM orders))

    Why use Subqueries?
    - Filtering based on aggregate calculations.
    - Data validation (existence / non-existence of records).
    - Comparing data across tables.
    - Preparing intermediate results for ETL pipelines.
    - Simplifying complex queries without writing multiple joins.

    Common Mistakes:
    - Forgetting parentheses around subqueries.
    - Using IN with a subquery that returns NULL (can give unexpected results).
    - Scalar subquery returning multiple rows (use LIMIT or aggregation to fix).
    - Not using EXISTS when checking for row existence (may return duplicates with IN).

    ETL / Data Engineering Use Cases:
    - Identify customers who never made a purchase.
    - Filter records based on aggregated metrics.
    - Detect missing or duplicate data before loading.
    - Create summary tables from transactional data.

    Quick Visual Guide:
    --------------------
    | Subquery Type | Returns       | Use Case Example |
    |---------------|---------------|------------------|
    | Scalar        | Single Value  | Compare score > AVG(score) |
    | IN / NOT IN   | Multiple      | Find customers with orders |
    | EXISTS        | Boolean       | Check if orders exist for customer |
    | SELECT        | Value per row | Add avg_score column |
    | FROM / Derived| Table         | Aggregate sales per customer |
    | Nested        | Depends       | Avg of customers who made orders |
```

```sql
*/

USE MyDatabase;

-- Find customers whose score is greater than average score
SELECT first_name, score
FROM customers
WHERE score > (
    SELECT AVG(score)       -- Inner query calculates average score
    FROM customers
);

-- Find customers who placed orders
SELECT first_name
FROM customers
WHERE id IN (
    SELECT customer_id
    FROM orders
);

-- Find customers who never placed any order
SELECT first_name
FROM customers
WHERE id NOT IN (
    SELECT customer_id
    FROM orders
);

-- Find customers who have orders
SELECT first_name
FROM customers c
WHERE EXISTS (
    SELECT 1
    FROM orders o
    WHERE o.customer_id = c.id
);

-- Show customers with their score AND the average score
SELECT
    first_name,
    score,
    (SELECT AVG(score) FROM customers) AS avg_score
FROM customers;

-- Find average sales per customer
SELECT AVG(total_sales)
FROM (
    SELECT SUM(sales) AS total_sales
    FROM orders
    GROUP BY customer_id
) AS temp;
```

```sql
-- Find customers whose score is higher than the average score of customers who
placed orders
SELECT first_name, score
FROM customers
WHERE score > (
    SELECT AVG(score)
    FROM customers
    WHERE id IN (
        SELECT customer_id
        FROM orders
    )
);

-- Find customers whose score is higher than the maximum sales value
SELECT first_name, score
FROM customers
WHERE score > (
    SELECT MAX(sales)
    FROM orders
);

-- ============================
-- Extra Practice Queries
-- 1. Customers whose score is lower than minimum sales
SELECT first_name, score
FROM customers
WHERE score < (
    SELECT MIN(sales)
    FROM orders
);

-- 2. Customers whose score is equal to their total sales
SELECT first_name, score
FROM customers
WHERE score = (
    SELECT SUM(sales)
    FROM orders
    WHERE orders.customer_id = customers.id
);

-- 3. Customers whose score is higher than the average score of German customers
SELECT first_name, score
FROM customers
WHERE score > (
    SELECT AVG(score)
    FROM customers
    WHERE country = 'Germany'
);
```