What is a Subquery?

A **subquery** is a query nested inside another query. It is executed first, and the result of the subquery is used by the outer query to perform its operations. Subqueries are commonly used in the `WHERE`, `FROM`, and `SELECT` clauses of SQL queries.

Example 1: Simple Subquery in the `WHERE` Clause

Imagine we have two tables:

1. **Employees Table** (`employees`)

employee_id	name	department_id	salary
1	John Doe	101	5000
2	Jane Smith	102	6000
3	Alice Brown	101	7000
4	Bob White	103	5500

2. **Departments Table** (`departments`)

department_id	department_name
101	HR
102	Sales
103	Marketing

Now, let's say we want to **find the employees who work in the "HR" department**. Instead of manually finding the `department_id` for "HR", we can use a subquery to dynamically retrieve it.

```
SELECT employee_id, name
FROM employees
WHERE department_id = (SELECT department_id FROM departments WHERE
department_name = 'HR');
```

Explanation:

1. **Subquery**: `(SELECT department_id FROM departments WHERE department_name = 'HR')`
   - This subquery runs first and finds the `department_id` of the "HR" department.
   - In this case, it returns `101` (since HR has a `department_id` of `101`).
2. **Outer Query**:
   - The outer query selects employees from the `employees` table where `department_id = 101`.

- It will return the employees whose department is "HR", which are **John Doe** and **Alice Brown**.

Result:

employee_id	name
1	John Doe
3	Alice Brown

Example 2: Subquery in the `SELECT` Clause

Let's say we want to **find each employee's name and their salary compared to the highest salary in the company**.

```
SELECT name, salary,
       (SELECT MAX(salary) FROM employees) AS highest_salary
FROM employees;
```

Explanation:

1. **Subquery**: `(SELECT MAX(salary) FROM employees)`
   - This subquery calculates the maximum salary from the entire `employees` table.
   - The result of the subquery will be `7000` (which is the highest salary in the table).
2. **Outer Query**:
   - The outer query selects the `name` and `salary` for each employee, and for each row, it also adds the value of the highest salary in the company (which is `7000`).

Result:

name	salary	highest_salary
John Doe	5000	7000
Jane Smith	6000	7000
Alice Brown	7000	7000
Bob White	5500	7000

Example 3: Correlated Subquery

A **correlated subquery** is a subquery that refers to a column from the outer query. Unlike a normal subquery, the subquery is executed for **each row** processed by the outer query.

Let's say we want to find employees whose salary is greater than the **average salary** of their own department.

```
SELECT employee_id, name, salary, department_id
```

```
FROM employees e
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department_id =
e.department_id);
```

Explanation:

1. **Correlated Subquery**: `(SELECT AVG(salary) FROM employees WHERE department_id = e.department_id)`
    - The subquery calculates the **average salary** for the department of the employee currently being processed by the outer query.
    - For each employee, the subquery is executed to find the average salary for their specific department.
2. **Outer Query**:
    - The outer query selects employees whose salary is greater than the average salary for their department.

Result:

For example, let's assume the averages for each department are:

- HR (department_id = 101): Avg salary = `(5000 + 7000) / 2 = 6000`
- Sales (department_id = 102): Avg salary = `6000`
- Marketing (department_id = 103): Avg salary = `5500`

Then, the query will return employees whose salary is greater than their department's average salary.

employee_id	name	salary	department_id
3	Alice Brown	7000	101
2	Jane Smith	6000	102

- **John Doe** is excluded because his salary is equal to the average salary for his department (HR).
- **Bob White** is excluded because his salary is not greater than the average salary for Marketing.

Summary:

- A **subquery** is simply a query inside another query.
- It can be used in various clauses like `WHERE`, `SELECT`, or `FROM`.
- **Correlated subqueries** depend on the outer query, while **non-correlated subqueries** run independently.
- Subqueries help simplify queries by breaking down complex logic into smaller, more manageable parts.

EXERCISE:

New Table: orders

order_id	customer_id	product_id	order_amount	order_date
1	101	201	150	2025-01-10
2	102	202	250	2025-01-12
3	103	201	300	2025-01-15
4	104	203	450	2025-01-20
5	101	202	200	2025-01-22
6	105	204	500	2025-01-25

Additional Table: products

product_id	product_name	price
201	Product A	100
202	Product B	150
203	Product C	200
204	Product D	250

Sub query Questions

Question 1: Find All Orders with Order Amount Greater Than the Average Order Amount

Question 2: Find Products Purchased More Than Once

Question 3: Find Customers Who Have Placed Orders for Products with a Price Greater Than 200

Question 4: Find Orders Where the Total Amount of Orders by the Same Customer is Greater Than 500

Question 5: Find Orders Where the Product is More Expensive than the Average Price of All Products

ANSWER:

Sure! Let's go over the solutions to each of the subquery questions using detailed explanations.

New Table: orders

order_id	customer_id	product_id	order_amount	order_date
1	101	201	150	2025-01-10
2	102	202	250	2025-01-12
3	103	201	300	2025-01-15
4	104	203	450	2025-01-20
5	101	202	200	2025-01-22
6	105	204	500	2025-01-25

Additional Table: products

product_id	product_name	price
201	Product A	100
202	Product B	150
203	Product C	200
204	Product D	250

Answer 1: Find All Orders with Order Amount Greater Than the Average Order Amount

Query:

```
SELECT order_id, customer_id, order_amount, order_date
FROM orders
WHERE order_amount >
   (SELECT AVG(order_amount)
    FROM orders);
```

Explanation:

- **Inner Query**: The inner query (SELECT AVG(order_amount) FROM orders) calculates the **average order amount** from the entire orders table.
- **Outer Query**: The outer query selects all columns from the orders table where the order_amount is greater than the average calculated by the inner query.
- **Result**: It returns all orders where the amount is above the overall average.

Answer 2: Find Products Purchased More Than Once

Query:

```
SELECT p.product_id, p.product_name
FROM products p
WHERE p.product_id IN
  (SELECT o.product_id
   FROM orders o
   GROUP BY o.product_id
   HAVING COUNT(o.order_id) > 1);
```

Explanation:

- **Inner Query**: The inner query (SELECT o.product_id FROM orders o GROUP BY o.product_id HAVING COUNT(o.order_id) > 1) finds all product_ids that have been purchased more than once (i.e., there are more than 1 order for the same product).
    - The GROUP BY o.product_id groups the orders by product_id, and HAVING COUNT(o.order_id) > 1 filters those groups where the count of orders is greater than 1.
- **Outer Query**: The outer query selects all products where the product_id matches any of those from the inner query (i.e., products that were purchased more than once).
- **Result**: It returns the products that have been purchased more than once.

Answer 3: Find Customers Who Have Placed Orders for Products with a Price Greater Than 200

Query:

```
SELECT DISTINCT o.customer_id
FROM orders o
WHERE o.product_id IN
  (SELECT p.product_id
```

```
FROM products p
WHERE p.price > 200);
```

Explanation:

- **Inner Query**: The inner query (SELECT p.product_id FROM products p WHERE p.price > 200) selects the product_ids of products that have a price greater than 200.
- **Outer Query**: The outer query selects customer_id from the orders table where the product_id is in the list of products with a price greater than 200 (as determined by the inner query).
- **Result**: It returns a list of **unique** customers who have ordered products priced above 200.

Answer 4: Find Orders Where the Total Amount of Orders by the Same Customer is Greater Than 500

Query:

```
SELECT o.order_id, o.customer_id, o.order_amount, o.order_date
FROM orders o
WHERE o.customer_id IN
  (SELECT o1.customer_id
   FROM orders o1
   GROUP BY o1.customer_id
   HAVING SUM(o1.order_amount) > 500);
```

Explanation:

- **Inner Query**: The inner query (SELECT o1.customer_id FROM orders o1 GROUP BY o1.customer_id HAVING SUM(o1.order_amount) > 500) groups the orders by customer_id and calculates the **total order amount** for each customer using SUM(o1.order_amount). It then filters to include only customers whose total order amount is greater than 500.
- **Outer Query**: The outer query selects all the orders (order_id, customer_id, order_amount, and order_date) where the customer_id is in the list of customers whose total order amount is greater than 500 (from the inner query).
- **Result**: It returns all orders placed by customers who have spent more than 500 in total.

Answer 5: Find Orders Where the Product is More Expensive Than the Average Price of All Products

Query:

```
SELECT o.order_id, o.customer_id, o.product_id, o.order_amount, o.order_date
FROM orders o
WHERE o.product_id IN
  (SELECT p.product_id
   FROM products p
   WHERE p.price >
     (SELECT AVG(price) FROM products));
```

Explanation:

- **Inner Query**: The innermost query (SELECT AVG(price) FROM products) calculates the **average price** of all products in the products table.
- **Outer Inner Query**: The next query (SELECT p.product_id FROM products p WHERE p.price > (SELECT AVG(price) FROM products)) selects product_ids where the product price is greater than the calculated average price.
- **Outer Query**: The outer query then selects all the orders where the product_id matches any of the product_ids that are more expensive than the average price.
- **Result**: It returns all orders where the product ordered has a price higher than the average price of all products.