Fictional E-commerce Database Schema

1. `Customers` Table

customer_id	name	country	join_date
1	Alice	USA	2021-01-15
2	Bob	UK	2020-12-01
3	Charlie	USA	2022-02-20
4	David	Canada	2021-06-11
5	Eve	Australia	2023-01-25

2. `Products` Table

product_id	product_name	price	category
101	Laptop	1200	Electronics
102	Smartphone	800	Electronics
103	Headphones	100	Accessories
104	Keyboard	50	Accessories
105	Monitor	300	Electronics

3. `Orders` Table

order_id	customer_id	order_date	total_amount
1001	1	2021-03-10	1300
1002	2	2020-12-15	2000
1003	3	2022-03-05	900
1004	4	2021-08-21	350
1005	5	2023-02-10	1250

4. `OrderDetails` Table

order_detail_id	order_id	product_id	quantity	price_at_time
1	1001	101	1	1200
2	1001	103	1	100
3	1002	102	2	800
4	1003	105	3	300
5	1004	104	2	50
6	1005	101	1	1200
7	1005	103	1	50

SQL Questions (Using Subqueries)

1. **Find customers who have placed an order in 2023.**
2. **Find all products that have never been ordered.**
3. **Get the name of the customer who spent the most on a single order.**
4. **List all orders where the total amount is greater than the average total amount of all orders.**
5. **Find the customer who has ordered the highest quantity of any single product.**
6. **Find the name of the customer who has never placed an order.**
7. **Find all customers who have ordered a product from the "Electronics" category.**
8. **List all customers who have purchased every product in the "Accessories" category.**
9. **Find the most expensive product that has been ordered.**
10. **List all orders that include more than 2 different products.**
11. **Find the customer who has ordered the most different products.**
12. **Find the total revenue generated from products in the "Electronics" category.**
13. **Find the customers who placed their first order in 2022.**
14. **Find the product that has generated the most revenue.**
15. **List all products that have been ordered by at least 3 different customers.**
16. **Find all orders with a total amount greater than the largest single product price.**
17. **Find the average quantity of products ordered per customer for products in the "Accessories" category.**

Answers, Query Explanations, and Why Direct Queries Won’t Work

1. Find customers who have placed an order in 2023.

Answer:

```sql
SELECT name
FROM Customers
WHERE customer_id IN (
    SELECT customer_id
    FROM Orders
    WHERE YEAR(order_date) = 2023
);
```

Query Explanation:

- The inner query retrieves `customer_id`s from the `Orders` table where the `order_date` is in 2023.
- The outer query uses this result to fetch the corresponding customer names from the `Customers` table.

Why a Direct Query Won’t Work:

- The `Customers` table does not contain order information, so we must query the `Orders` table to filter by the year. This requires a subquery to link the two tables.

---

2. Find all products that have never been ordered.

Answer:

```sql
SELECT product_name
FROM Products
WHERE product_id NOT IN (
    SELECT product_id
    FROM OrderDetails
);
```

Query Explanation:

- The subquery retrieves all `product_id`s from the `OrderDetails` table, representing products that have been ordered.
- The outer query selects products from the `Products` table whose `product_id` is not in that list.

Why a Direct Query Won't Work:

- The `Products` table doesn't indicate whether a product has been ordered, and the `OrderDetails` table doesn't contain all products. We need a subquery to find products not present in `OrderDetails`.

---

3. Get the name of the customer who spent the most on a single order.

Answer:

```sql
SELECT name
FROM Customers
WHERE customer_id = (
    SELECT customer_id
    FROM Orders
    ORDER BY total_amount DESC
    LIMIT 1
);
```

Query Explanation:

- The subquery finds the `customer_id` associated with the order having the highest `total_amount`.
- The outer query retrieves the name of that customer.

Why a Direct Query Won't Work:

- The `Customers` table doesn't contain order information. We need a subquery to find the customer who placed the largest order by querying the `Orders` table first.

---

4. List all orders where the total amount is greater than the average total amount of all orders.

Answer:

```sql
SELECT order_id, total_amount
FROM Orders
WHERE total_amount > (
    SELECT AVG(total_amount)
    FROM Orders
);
```

Query Explanation:

- The subquery calculates the average `total_amount` of all orders.
- The outer query selects all orders where the `total_amount` exceeds this average.

Why a Direct Query Won't Work:

- Direct queries can't compare individual rows to an aggregate value (the average total) without a subquery.

---

5. Find the customer who has ordered the highest quantity of any single product.

Answer:

```sql
SELECT name
FROM Customers
WHERE customer_id = (
    SELECT customer_id
    FROM Orders
    WHERE order_id = (
        SELECT order_id
        FROM OrderDetails
        ORDER BY quantity DESC
        LIMIT 1
    )
);
```

Query Explanation:

- The innermost subquery finds the `order_id` where the highest quantity of any product was ordered.
- The middle subquery finds the `customer_id` associated with that order.
- The outer query retrieves the customer's name.

Why a Direct Query Won't Work:

- The `Customers` table doesn't include information about product quantities, so we need multiple subqueries to join data from `OrderDetails` and `Orders`.

---

6. Find the name of the customer who has never placed an order.

Answer:

```sql
SELECT name
FROM Customers
WHERE customer_id NOT IN (
    SELECT customer_id
    FROM Orders
);
```

Query Explanation:

- The subquery retrieves all `customer_id`s from the `Orders` table who have placed orders.
- The outer query selects customers from the `Customers` table whose `customer_id` is not in that list.

Why a Direct Query Won't Work:

- The `Customers` table doesn't contain order information. A subquery is needed to check which customers haven't appeared in the `Orders` table.

---

7. Find all customers who have ordered a product from the "Electronics" category.

Answer:

```
SELECT name
FROM Customers
WHERE customer_id IN (
    SELECT customer_id
    FROM Orders
    WHERE order_id IN (
        SELECT order_id
        FROM OrderDetails
        WHERE product_id IN (
            SELECT product_id
            FROM Products
            WHERE category = 'Electronics'
        )
    )
);
```

Query Explanation:

- The innermost subquery retrieves `product_id`s from the `Products` table in the "Electronics" category.
- The next subquery retrieves `order_id`s from `OrderDetails` where those products were ordered.
- The final subquery retrieves `customer_id`s corresponding to those orders, and the outer query fetches their names.

Why a Direct Query Won’t Work:

- The `Customers` table has no information about product categories or orders. The query requires multiple joins across `Orders`, `OrderDetails`, and `Products`.

---

8. List all customers who have purchased every product in the "Accessories" category.

Answer:

```sql
SELECT name
FROM Customers
WHERE customer_id NOT IN (
    SELECT customer_id
    FROM (
        SELECT customer_id, COUNT(DISTINCT product_id) AS product_count
        FROM Orders
        JOIN OrderDetails USING(order_id)
        WHERE product_id IN (
            SELECT product_id
            FROM Products
            WHERE category = 'Accessories'
        )
        GROUP BY customer_id
    ) AS customer_products
    WHERE customer_products.product_count < (
        SELECT COUNT(product_id)
        FROM Products
        WHERE category = 'Accessories'
    )
);
```

Query Explanation:

- The inner subquery counts the number of distinct products in the "Accessories" category purchased by each customer.

- The outer query selects customers who have not purchased all products in that category by excluding those who have a product count less than the total number of "Accessories" products.

Why a Direct Query Won't Work:

- You need to compare the number of products a customer has purchased to the total number of products in the "Accessories" category, which requires a subquery.

---

9. Find the most expensive product that has been ordered.

Answer:

```
SELECT product_name
FROM Products
WHERE product_id = (
    SELECT product_id
    FROM OrderDetails
    ORDER BY price_at_time DESC
    LIMIT 1
);
```

Query Explanation:

- The subquery retrieves the `product_id` of the product that was sold at the highest price.
- The outer query retrieves the name of that product from the `Products` table.

Why a Direct Query Won't Work:

- The `Products` table doesn't store the price at the time of the order, so a subquery is necessary to retrieve the relevant `product_id` from `OrderDetails`.

---

10. List all orders that include more than 2 different products.

Answer:

```sql
SELECT order_id
FROM Orders
WHERE order_id IN (
    SELECT order_id
    FROM OrderDetails
    GROUP BY order_id
    HAVING COUNT(DISTINCT product_id) > 2
);
```

Query Explanation:

- The subquery counts distinct products in each order and filters orders with more than 2 different products.
- The outer query retrieves those order IDs.

Why a Direct Query Won't Work:

- The `Orders` table only contains total order amounts, not details about the number of distinct products. A subquery is required to count products for each order in `OrderDetails`.

---

11. Find the customer who has ordered the most different products.

Answer:

```sql
SELECT name
FROM Customers
WHERE customer_id = (
    SELECT customer_id
    FROM Orders
    WHERE order_id IN (
        SELECT order_id
        FROM OrderDetails
        GROUP BY order_id
    )
    GROUP BY customer_id
    ORDER BY COUNT(DISTINCT product_id) DESC
    LIMIT 1
);
```

Query Explanation:

- The innermost subquery counts distinct products ordered in each order.
- The middle subquery groups orders by `customer_id` and orders them by the count of distinct products in descending order.
- The outer query retrieves the name of the customer with the highest count.

Why a Direct Query Won’t Work:

- The `Customers` table doesn’t contain product ordering information. We need a subquery to count products ordered by each customer.

---

12. Find the total revenue generated from products in the "Electronics" category.

Answer:

```sql
SELECT SUM(price_at_time * quantity) AS total_revenue
FROM OrderDetails
WHERE product_id IN (
    SELECT product_id
    FROM Products
    WHERE category = 'Electronics'
);
```

Query Explanation:

- The subquery retrieves the `product_id`s for "Electronics" products from the `Products` table.
- The outer query calculates the total revenue for those products by multiplying the `price_at_time` by the `quantity` in the `OrderDetails` table.

Why a Direct Query Won't Work:

- The `OrderDetails` table doesn't include product categories. A subquery is required to filter products by category from the `Products` table.

---

13. Find the customers who placed their first order in 2022.

Answer:

```sql
SELECT name
FROM Customers
WHERE customer_id IN (
    SELECT customer_id
    FROM Orders
    GROUP BY customer_id
    HAVING MIN(order_date) >= '2022-01-01' AND MIN(order_date) <= '2022-12-31'
);
```

Query Explanation:

- The subquery finds the earliest (`MIN`) order date for each customer and filters those whose first order was in 2022.
- The outer query retrieves the names of those customers.

Why a Direct Query Won't Work:

- The `Customers` table doesn't track the date of the first order. You need a subquery to find the first order date from the `Orders` table.

---

14. Find the product that has generated the most revenue.

Answer:

```sql
SELECT product_name
FROM Products
WHERE product_id = (
    SELECT product_id
    FROM OrderDetails
    GROUP BY product_id
    ORDER BY SUM(price_at_time * quantity) DESC
    LIMIT 1
);
```

Query Explanation:

- The subquery calculates the total revenue for each product in the `OrderDetails` table and finds the product with the highest revenue.
- The outer query retrieves the name of that product.

Why a Direct Query Won't Work:

- The `Products` table doesn't store details about quantities sold or total revenue. A subquery is essential to aggregate this information from `OrderDetails`.

---

15. List all products that have been ordered by at least 3 different customers.

Answer:

```sql
SELECT product_name
FROM Products
WHERE product_id IN (
    SELECT product_id
    FROM OrderDetails
    JOIN Orders USING(order_id)
    GROUP BY product_id
    HAVING COUNT(DISTINCT customer_id) >= 3
);
```

Note: Added a `JOIN` with `Orders` to access `customer_id` in `OrderDetails`.

Query Explanation:

- The subquery joins `OrderDetails` with `Orders` to associate each `product_id` with a `customer_id`.
- It then counts distinct customers for each product and filters products ordered by at least 3 customers.
- The outer query retrieves the names of those products.

Why a Direct Query Won't Work:

- The `Products` table doesn't have customer information, and the `OrderDetails` table doesn't directly link to customers without joining with `Orders`. We need a subquery to count distinct customers per product.

---

16. Find all orders with a total amount greater than the largest single product price.

Answer:

```sql
SELECT order_id, total_amount
FROM Orders
WHERE total_amount > (
    SELECT MAX(price)
    FROM Products
);
```

Query Explanation:

- The subquery retrieves the highest product price from the `Products` table.
- The outer query filters orders whose `total_amount` is greater than this price.

Why a Direct Query Won't Work:

- The `Orders` table doesn't store product prices, and the `Products` table doesn't store order totals. A subquery is needed to compare these values.

---

17. Find the average quantity of products ordered per customer for products in the "Accessories" category.

Answer:

```sql
SELECT AVG(quantity) AS avg_quantity
FROM OrderDetails
WHERE product_id IN (
    SELECT product_id
    FROM Products
    WHERE category = 'Accessories'
);
```

Query Explanation:

- The subquery retrieves the `product_id`s for products in the "Accessories" category.
- The outer query calculates the average quantity ordered for those products across all orders.

Why a Direct Query Won't Work:

- The `OrderDetails` table doesn't include product categories. We need a subquery to filter products by category from the `Products` table.

Conclusion

These SQL problems demonstrate the power of **subqueries** in MySQL, especially for tasks involving multiple tables, aggregations, and complex filtering. Subqueries allow us to break down a problem into smaller, more manageable parts by first querying one dataset and then using the result in a larger query. This approach is especially useful when:

- **Data is spread across multiple tables** (e.g., customer information, order details, and product categories).
- **Aggregations** (e.g., total revenue, average quantities) and **comparisons** (e.g., comparing order totals to aggregate values) need to be performed.
- **Negation** or **exclusion logic** (e.g., finding products that have never been ordered) cannot be handled by a simple query.

In many cases, direct queries are insufficient because they cannot simultaneously handle data retrieval, aggregation, and filtering across different tables. Subqueries solve this problem by enabling more sophisticated query logic, making them an essential tool for intermediate and advanced SQL users.