

Data Immersion

Exercise 3.8

Performing Subqueries

Step 1: Find the average amount paid by the top 5 customers

Query →

```
SELECT country, AVG(total_amount_paid) AS average FROM (
Select
SUM (A.amount) AS total_amount_paid
FROM payment A
INNER JOIN customer B ON A.customer_id = B.customer_id
INNER JOIN address C ON B.address_id = C.address_id
INNER JOIN city D ON C.city_id = D.city_id
INNER JOIN country E ON D.country_id = E.country_id
WHERE (E.country,D.city) IN (
    SELECT D.country, C.city
    FROM customer A
    INNER JOIN address B ON A.address_id = B.address_id
    INNER JOIN city C ON B.city_id = C.city_id
    INNER JOIN country D ON C.country_id = D.country_id
    WHERE D.country IN (
        SELECT D.country
        FROM customer A
        JOIN address B ON A.address_id = B.address_id
        JOIN city C ON B.city_id = C.city_id
        JOIN country D ON C.country_id = D.country_id
        GROUP BY D.country
        ORDER BY COUNT(A.customer_id) DESC
        LIMIT 10 )
    GROUP BY D.country, C.city
    ORDER BY COUNT (A.customer_id) DESC
    LIMIT 10 )
    GROUP BY B.customer_id, B.first_name, B.last_name, D.city, E.country
    ORDER BY total_amount_paid DESC
    LIMIT 5) AS total_amount_paid;
```

Output:

The screenshot shows a PostgreSQL query editor interface. At the top, the connection is 'Rockbuster/postgres@PostgreSQL 17'. Below the toolbar, the 'Query' tab is active, displaying a SQL query. The query calculates the average total amount paid by customers, joined with their address and city information. The output is shown in the 'Data Output' tab, which displays a single row with the average value.

```
1 SELECT AVG(total_amount_paid) AS average FROM (  
2   Select  
3   SUM (A.amount) AS total_amount_paid  
4   FROM payment A  
5   INNER JOIN customer B ON A.customer_id = B.customer_id  
6   INNER JOIN address C ON B.address_id = C.address_id  
7   INNER JOIN city D ON C.city_id = D.city_id  
8   INNER JOIN country E ON D.country_id = E.country_id  
9   WHERE (E.country,D.city) IN (  
10      SELECT D.country, C.city  
11      FROM customer A  
12      INNER JOIN address B ON A.address_id = B.address_id  
13      INNER JOIN city C ON B.city_id = C.city_id  
14      INNER JOIN country D ON C.country_id = D.country_id  
15      WHERE D.country IN (  
16         SELECT D.country
```

Data Output Messages Notifications

	average numeric
1	105.5540000000000000

Step 2: Find out how many of the top 5 customers you identified in step 1 are based within each country

Query:

```
SELECT D.country, COUNT(DISTINCT A.customer_id) AS all_customer_count,  
COUNT(DISTINCT top_customers.customer_id) AS top_customers_count  
FROM customer A  
INNER JOIN address B ON A.address_id = B.address_id  
INNER JOIN city C ON B.city_id = C.city_id  
INNER JOIN country D ON C.country_id = D.country_id  
LEFT JOIN (  
    SELECT A.customer_id, A.first_name, A.last_name, c.city, d.country, SUM(amount) AS  
total_amount_paid  
    FROM customer A  
    INNER JOIN address B ON A.address_id = B.address_id  
    INNER JOIN city C ON b.city_id = c.city_id  
    INNER JOIN country D ON c.country_id = d.country_id  
    INNER JOIN payment E ON A.customer_id = E.customer_id  
    WHERE c.city IN(  
        SELECT C.city  
        FROM customer A  
        INNER JOIN address B ON A.address_id = B.address_id
```

```

INNER JOIN city C ON B.city_id = C.city_id
INNER JOIN country D on C.country_id = D.country_id
WHERE D.country IN(
    SELECT D.country
    FROM customer A
    INNER JOIN address B ON A.address_id = B.address_id
    INNER JOIN city C ON B.city_id = C.city_id
    INNER JOIN country D on C.country_id = D.country_id
    GROUP BY country
    ORDER BY COUNT (customer_id) DESC
    LIMIT 10)
GROUP BY D.country, C.city
ORDER BY COUNT (customer_id) DESC LIMIT 10)
GROUP BY A.customer_id, A.first_name, A.last_name, c.city, d.country
ORDER BY total_amount_paid DESC
LIMIT 5) AS top_customers ON top_customers.country = D.country
GROUP BY D.country
ORDER BY top_customers_count DESC limit 5;

```

Output:

Rockbuster/postgres@PostgreSQL 17

Do you think steps 1 and step 2 could be done without using subqueries?

→ Technically, steps 1 and 2 can be completed without subqueries. I could have used a **VIEW** table to handle these steps instead. The choice depends on how frequently these steps will be needed. If it's a one-time task, using subqueries might be more convenient, but if they are used repeatedly, a **VIEW** table would be a better option.

When do you think are subqueries useful?

→ I believe subqueries are more useful when working with smaller datasets since, with larger datasets, the query might need to run multiple times, slowing down the process. Subqueries also help simplify complex queries by breaking them into manageable parts, making them easier to read and understand.