

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

1. Copy the query you wrote in step 3 of the task from [Exercise 3.7: Joining Tables of Data](#) into the Query Tool. This will be your subquery, so give it an alias, “total_amount_paid,” and add parentheses around it.

The screenshot shows the Query Tool interface for a PostgreSQL database. The top bar indicates the connection is 'Rockbuster/postgres@PostgreSQL 15'. Below the toolbar, the 'Query' tab is active, displaying a SQL query. The query is a complex SELECT statement that calculates the average amount paid by the top 5 customers. It uses a subquery to find the top 5 customers based on the total amount paid, then calculates the average of those amounts. The 'Data Output' tab at the bottom shows the result of the query, which is a single row with the value 107.35 for the 'average_amount_paid' column.

```
1 SELECT ROUND(AVG(total_amount_paid),2) AS average_amount_paid
2 FROM (SELECT
3 A.customer_id, A.first_name, A.last_name, C.city, D.country, SUM(F.amount) AS total_amount_paid
4 FROM customer A
5 JOIN rental E ON A.customer_id = E.customer_id
6 JOIN payment F ON E.rental_id = F.rental_id
7 JOIN address B ON A.address_id = B.address_id
8 JOIN city C ON B.city_id = C.city_id
9 JOIN country D ON C.country_id = D.country_id
10 WHERE city IN ('Aurora', 'Atlixco','Xintai', 'Adoni','Dhule(Dhulia)','Kurashiki', 'Pingxiang','Sivas','Celaya','So Leopoldo')
11 GROUP BY a.customer_id, A.first_name, A.last_name, C.city, D.country
12 ORDER BY total_amount_paid DESC
13 LIMIT 5) AS total_amount_paid
```

The 'Data Output' tab shows the following result:

average_amount_paid
107.35

Step 2: Find out how many of the top 5 customers are based within each country.

Your final output should include 3 columns:

- “country”
- “all_customer_count” with the total number of customers in each country
- “top_customer_count” showing how many of the top 5 customers live in each country

```
1 SELECT DISTINCT(D.country),
2 COUNT (DISTINCT A.customer_id) as total_customers,
3 COUNT (DISTINCT D.country) as count_top_five_customers
4 FROM country D
5 INNER JOIN city C on D.country_id = C.country_id
6 INNER JOIN address B ON C.city_id = B.city_id
7 INNER JOIN customer A ON B.address_id = A.address_id
8 LEFT JOIN
9 (SELECT
10 A.customer_id, A.first_name, A.last_name, C.city, D.country, SUM(F.amount) AS total_amount_paid
11 FROM customer A
12 JOIN rental E ON A.customer_id = E.customer_id
13 JOIN payment F ON E.rental_id = F.rental_id
14 JOIN address B ON A.address_id = B.address_id
15 JOIN city C ON B.city_id = C.city_id
16 JOIN country D ON C.country_ID = D.country_id
17 WHERE country IN ('India', 'China','United States', 'Japan','Mexico','Brazil', 'Russian Federation','Philippines','Turkey','Indonesia')
18 GROUP BY a.customer_id, A.first_name, A.last_name, C.city, D.country
19 ORDER BY total_amount_paid DESC
20 LIMIT 5) AS top_five_customers
21 ON D.country = top_five_customers.Country
22 GROUP BY D.country, top_five_customers
23 ORDER BY total_customers DESC
24 LIMIT 5;
```

	country character varying (50)	total_customers bigint	count_top_five_customers bigint
1	India	60	1
2	China	53	1
3	United States	36	1
4	Japan	31	1
5	Mexico	30	1

Step 3

Write 1 to 2 short paragraphs on the following:

- Do you think steps 1 and 2 could be done without using subqueries?
 1. Yes, I believe we could have completed steps 1 and 2 without using subqueries. We could have created multiple queries for each stage and then joined them together, but that would have required a significant amount of extra coding. The subqueries enabled us to build complex queries by simply conducting a query on the result of another query, making the syntax more straightforward.
- When do you think subqueries are useful?
 1. Subqueries can be used when data must be processed in multiple steps or when data is regularly modified. Using a subquery allows us to perform several queries in a single step and avoids the need to change the query each time the data is updated because it will run correctly every time.