

Step 1

The screenshot shows a PostgreSQL query editor interface. The top menu bar includes File, Object, Tools, Edit, View, Window, and Help. The title bar indicates the connection is 'Rockbuster/postgres@PostgreSQL 17*'. The main query editor contains the following SQL code:

```
1 SELECT AVG(total_amount_paid) AS avg_amount_paid_top_five_customers
2 FROM (SELECT B.customer_id,
3 B.first_name,
4 B.last_name,
5 E.country,
6 D.city,
7 SUM(A.amount) AS total_amount_paid
8 FROM payment A
9 INNER JOIN customer B ON A.customer_id = B.customer_id
10 INNER JOIN address C ON B.address_id = C.address_id
11 INNER JOIN city D ON C.city_id = D.city_id
12 INNER JOIN country E ON D.country_id = E.country_id
13 WHERE D.city IN ('London', 'Aurora', 'Tokat', 'Mukateve', 'Pontianak', 'Gatineau', 'Molodetno', 'Saint-Denis', 'Yingkou',
14 GROUP BY B.customer_id, B.first_name, B.last_name, E.country, D.city) AS total_amount_paid;
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with two columns: 'avg_amount_paid_top_five_customers' and 'numeric'. The first row shows the value '116.63916666666667'.

avg_amount_paid_top_five_customers	numeric
1	116.63916666666667

```
SELECT AVG(total_amount_paid) AS avg_amount_paid_top_five_customers
FROM (SELECT B.customer_id,
B.first_name,
B.last_name,
E.country,
D.city,
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 D.city IN ('London', 'Aurora', 'Tokat', 'Mukateve', 'Pontianak', 'Gatineau', 'Molodetno', 'Saint-Denis', 'Yingkou', 'Atlixco')
GROUP BY B.customer_id, B.first_name, B.last_name, E.country, D.city) AS total_amount_paid;
```

Step 2

The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```
1 SELECT D.country, COUNT(DISTINCT top_5_customers.customer_id) AS top_customer_count,
2 COUNT (DISTINCT A.customer_id) AS all_customer_count
3 FROM customer A
4 INNER JOIN address B ON A.address_id = B.address_id
5 INNER JOIN city C ON B.city_id = C.city_id
6 INNER JOIN country D ON C.country_id = D.country_id
7 LEFT JOIN
8 (SELECT B.customer_id,
9 B.first_name,
10 B.last_name,
11 E.country,
12 D.city,
13 SUM(A.amount) AS total_amount_paid
14 FROM payment A
15 INNER JOIN customer B ON A.customer_id = B.customer_id
16 INNER JOIN address C ON B.address_id = C.address_id
17 INNER JOIN city D ON C.city_id = D.city_id
18 INNER JOIN country E ON D.country_id = E.country_id
19 WHERE D.city IN ('London', 'Aurora', 'Tokat', 'Mukateve', 'Pontianak', 'Gatineau', 'Molodetno', 'Saint-Denis', 'Yingkou',
20 GROUP BY B.customer_id, B.first_name, B.last_name, E.country, D.city) AS top_5_customers
21 ON D.country = top_5_customers.country
22 GROUP BY D.country
23 ORDER BY all_customer_count;
```

The results pane shows the following columns:

country	top_customer_count	all_customer_count
7amhin	0	1

Summary: Total rows: 108, Query complete 00:00:00.186

SELECT D.country, COUNT(DISTINCT top_5_customers.customer_id) AS top_customer_count,

COUNT (DISTINCT A.customer_id) AS all_customer_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 B.customer_id,

B.first_name,

B.last_name,

E.country,

D.city,

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 D.city IN ('London', 'Aurora', 'Tokat', 'Mukateve', 'Pontianak', 'Gatineau', 'Molodetno', 'Saint-
Denis', 'Yingkou','Atlixco')

GROUP BY B.customer_id, B.first_name, B.last_name, E.country, D.city) AS top_5_customers

ON D.country = top_5_customers.country

GROUP BY D.country

ORDER BY all_customer_count;
```

Step 3

The approach can quickly become difficult to manage and understand, especially when dealing with large datasets or more advanced logic such as ranking, filtering, and aggregating. Subqueries help simplify the logic by isolating specific tasks, making the overall query easier to read, debug, and maintain.

Subqueries are especially useful when you need to perform an intermediate calculation, filter based on aggregated results or reuse a derived dataset multiple times within a larger query. They allow you to modularize your SQL code, making it clearer and more efficient by breaking down a large problem into smaller, logical steps.