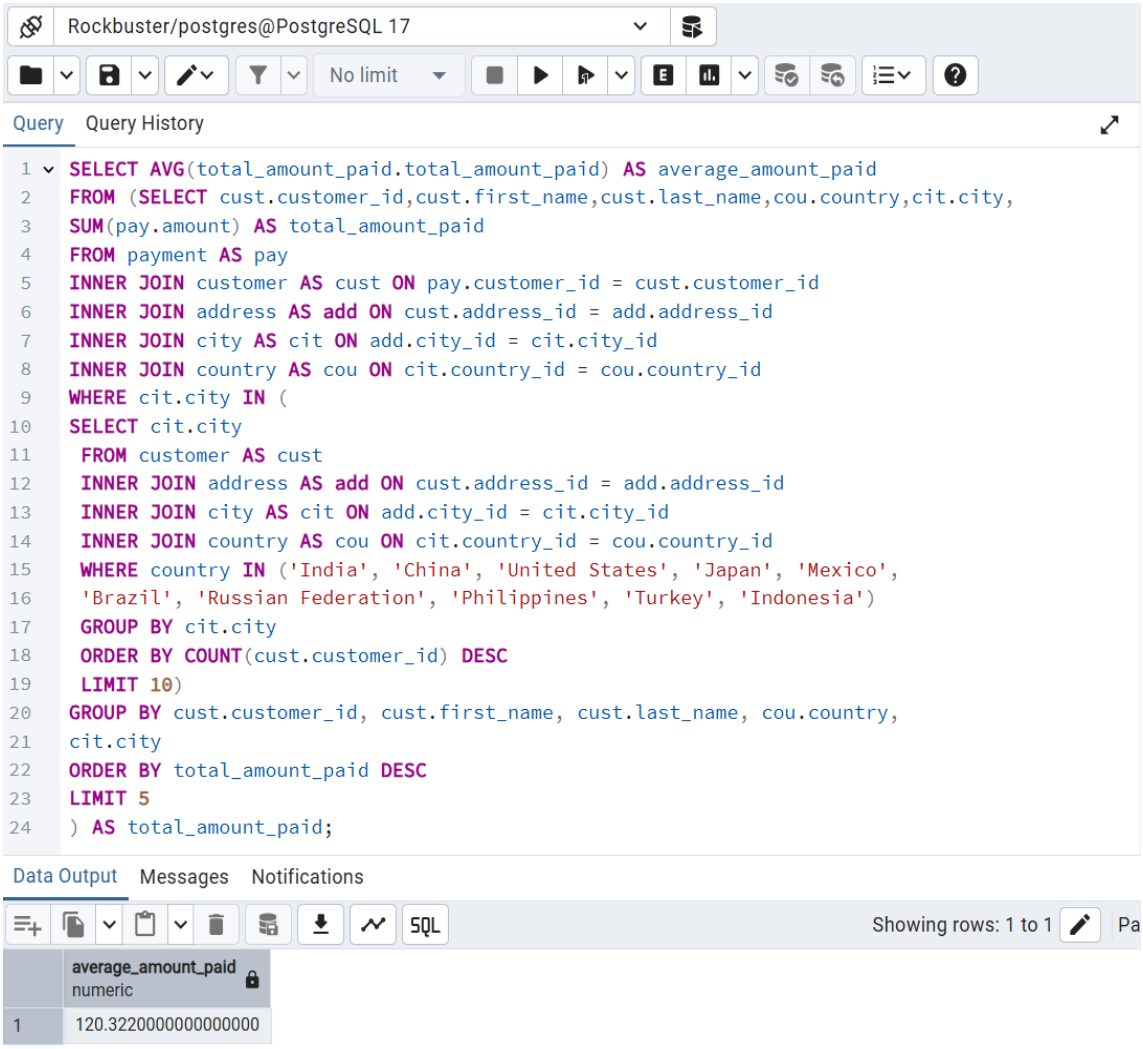
1. **Copy the query you wrote in step 3 of the task from Exercise 3.7: Joining Tables of Data into the Query Tool. This will be your subquery, so give it an alias, "total_amount_paid," and add parentheses around it.**

```
SELECT AVG(total_amount_paid.total_amount_paid) AS
average_amount_paid

FROM (SELECT
cust.customer_id,cust.first_name,cust.last_name,cou.country,cit.ci
ty,

SUM(pay.amount) AS total_amount_paid

FROM payment AS pay

INNER JOIN customer AS cust ON pay.customer_id = cust.customer_id

INNER JOIN address AS add ON cust.address_id = add.address_id

INNER JOIN city AS cit ON add.city_id = cit.city_id

INNER JOIN country AS cou ON cit.country_id = cou.country_id

WHERE cit.city IN (

SELECT cit.city

 FROM customer AS cust

 INNER JOIN address AS add ON cust.address_id = add.address_id

 INNER JOIN city AS cit ON add.city_id = cit.city_id

 INNER JOIN country AS cou ON cit.country_id = cou.country_id

 WHERE country IN ('India', 'China', 'United States', 'Japan',
'Mexico',

 'Brazil', 'Russian Federation', 'Philippines', 'Turkey',
'Indonesia')

 GROUP BY cit.city

 ORDER BY COUNT(cust.customer_id) DESC

 LIMIT 10)

GROUP BY cust.customer_id, cust.first_name, cust.last_name,
cou.country,

cit.city

ORDER BY total_amount_paid DESC

LIMIT 5

) AS total_amount_paid;
```

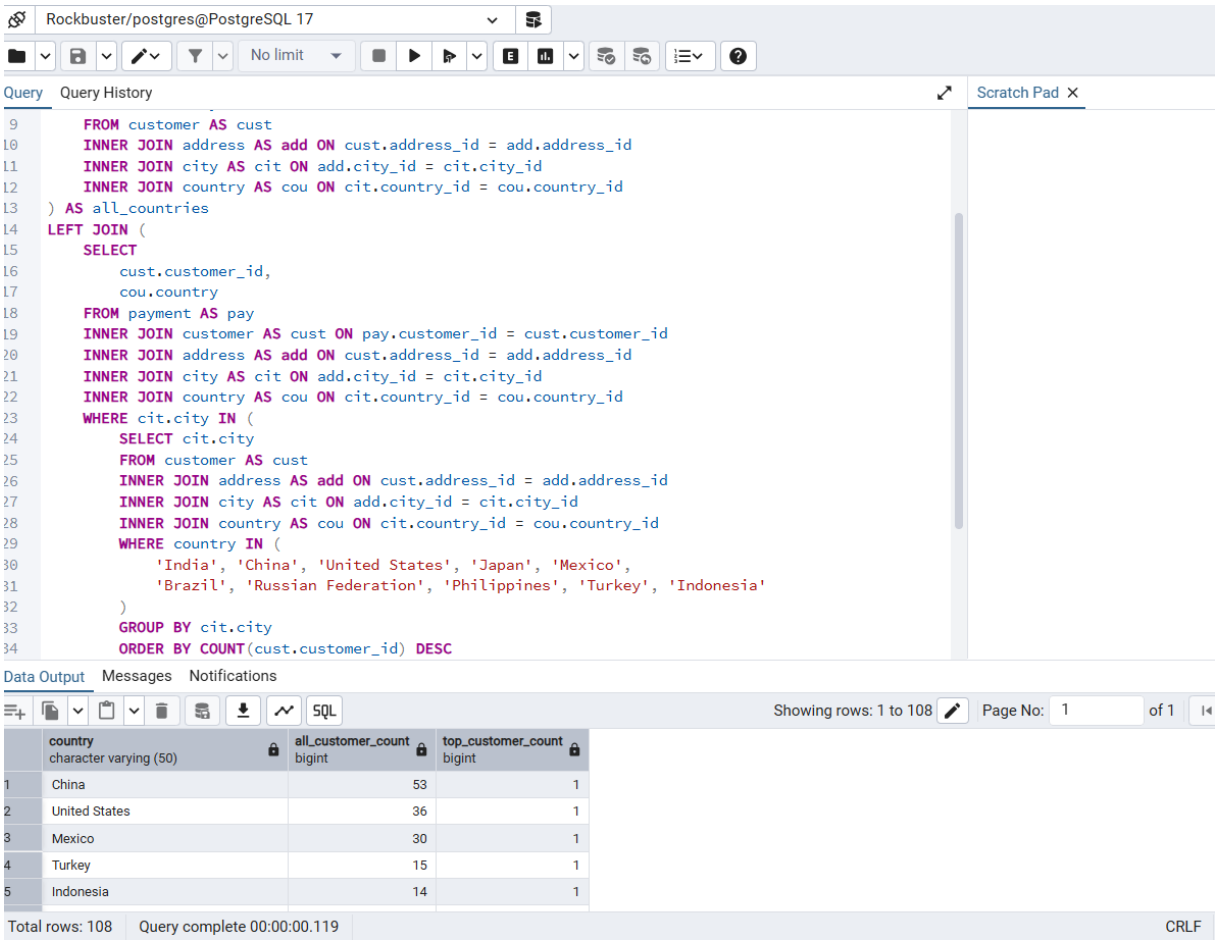


2. **Find out how many of the top 5 customers you identified in step 1 are based within each country.**

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**

```sql
SELECT
    all_countries.country,
    COUNT(DISTINCT all_countries.customer_id) AS all_customer_count,
    COUNT(DISTINCT top_customers.customer_id) AS top_customer_count
FROM (
    SELECT
        cust.customer_id,
        cou.country
    FROM customer AS cust
    INNER JOIN address AS add ON cust.address_id = add.address_id
    INNER JOIN city AS cit ON add.city_id = cit.city_id
    INNER JOIN country AS cou ON cit.country_id = cou.country_id
) AS all_countries
LEFT JOIN (
    SELECT
        cust.customer_id,
        cou.country
    FROM payment AS pay
    INNER JOIN customer AS cust ON pay.customer_id = cust.customer_id
    INNER JOIN address AS add ON cust.address_id = add.address_id
    INNER JOIN city AS cit ON add.city_id = cit.city_id
    INNER JOIN country AS cou ON cit.country_id = cou.country_id
    WHERE cit.city IN (
        SELECT cit.city
        FROM customer AS cust
        INNER JOIN address AS add ON cust.address_id = add.address_id
        INNER JOIN city AS cit ON add.city_id = cit.city_id
        INNER JOIN country AS cou ON cit.country_id = cou.country_id
        WHERE country IN (
            'India', 'China', 'United States', 'Japan', 'Mexico',
            'Brazil', 'Russian Federation', 'Philippines', 'Turkey',
'Indonesia'
        )
        GROUP BY cit.city
        ORDER BY COUNT(cust.customer_id) DESC
        LIMIT 10
    )
    GROUP BY cust.customer_id, cou.country
    ORDER BY SUM(pay.amount) DESC
    LIMIT 5
) AS top_customers
ON all_countries.customer_id = top_customers.customer_id
GROUP BY all_countries.country
ORDER BY top_customer_count DESC, all_customer_count DESC;
```



3. Steps 1 and 2 could technically be done without subqueries by using temporary tables or common table expressions (CTEs). However, subqueries offer a more efficient and readable solution when filtering or aggregating data within the same query. In our case, identifying the top 10 cities within the top 10 countries and then narrowing down to the top 5 customers requires multi-level filtering. Without subqueries, the logic would be more complex and harder to maintain.
   Subqueries are especially useful when we need to isolate specific data sets temporarily for use in filtering, grouping, or comparison. They allow us to nest logic directly within the main query, avoiding unnecessary joins or duplication. In scenarios like ranking, filtering top values, or breaking down data step-by-step, subqueries help keep the SQL concise and structured.