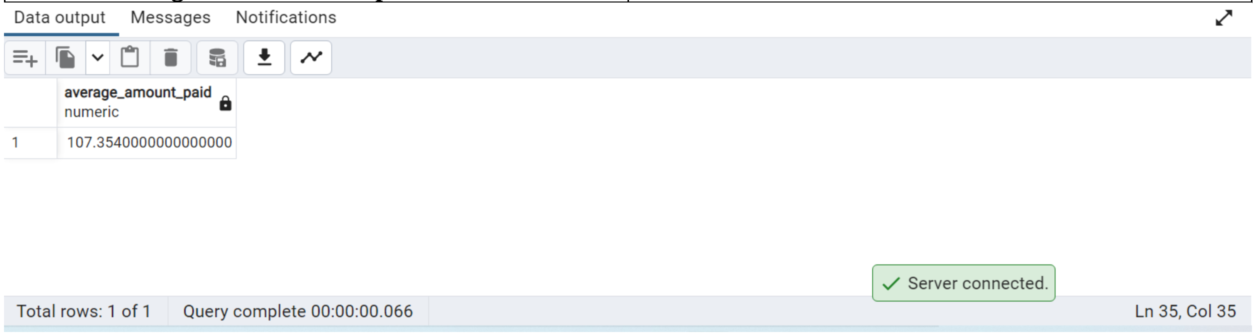
Juliana Lee

Task 3.9: Common Table Expressions

1) **Step 1:**

CTE:	**Explanation:**
WITH **average_total_amount_paid_cte(customer_id, first_name, last_name, city, country, total_amount_paid) AS** **(SELECT A.customer_id,** **A.first_name,** **A.last_name,** **D.city,** **E.country,** **SUM(B.amount) AS total_amount_paid** **FROM customer A** **INNER JOIN payment B on A.customer_id = B.customer_id** **INNER JOIN address C on A.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 (‘Aurora’, ‘Atlixco’, ‘Xintai’, ‘Adoni’, ‘Dhule (Dhulia)’, ‘Kurashiki’, ‘Pingxiang’, ‘Sivas’, ‘Celaya’, ‘So Leopoldo’)** **GROUP BY A.customer_id, A.first_name, A.last_name, D.city, E.country** **ORDER BY total_amount_paid** **DESC limit 5)** **SELECT AVG(total_amount_paid) AS average_amount_paid** **FROM average_total_amount_paid_cte**	**First, I copied and pasted the query from step in in 3.8. Then, I removed the outer query and rewrote it as CTE by using the WITH and naming it AS the inner query. Lastly, I added the SELECT AVG(total_amount_paid) AS average_amount_paid FROM average_total_amount_paid_cte.**



```
Data output    Messages    Notifications

     average_amount_paid
     numeric
1    107.3540000000000000

Total rows: 1 of 1    Query complete 00:00:00.066    Server connected.    Ln 35, Col 35
```

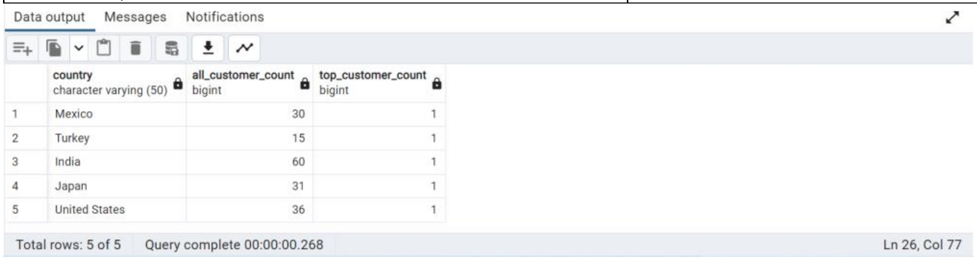
Step 2:

CTE:	**Explanation:**

Juliana Lee

```
WITH top_customer_count_cte(amount,
customer_id, first_name, last_name, city, country,
total_amount_paid) AS
(SELECT A.amount, B.customer_id, B.first_name,
B.last_name, D.city, E.country),
SUM(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 city IN (‘Aurora’, ‘Atlixco’, ‘Xintai’,
‘Adoni’, ‘Dhule (Dhulia)’, ‘Kurashiki’, ‘Pingxiang’,
‘Sivas’, ‘Celaya’, ‘So Leopoldo’)
GROUP BY A.amount, B.customer_id,
B.first_name, B.last_name, D.city, E.country
ORDER BY SUM (amount) DESC limit 5),
customer_count_cte AS (SELECT D.country,
COUNT(DISTINCT A.customer_id) AS
all_customer_count,
COUNT(DISTINCT D.country) AS
top_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
GROUP BY D.country)
SELECT D.country, COUNT(DISTINCT
A.customer_id) AS all_customer_count,
COUNT(DISTINCT
top_customer_count_cte.customer_id) AS
top_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 top_customer_count_cte ON
D.country=top_customer_count_cte.country
GROUP BY D.country
ORDER BY top_customer_count DESC
LIMIT 5;
```

The steps for step 2 was similar to step 1 except this time I added 2 CTE’s: one for all customer count and another for top customer count. Also, I used LEFT JOIN to combine the payment and customer tables.



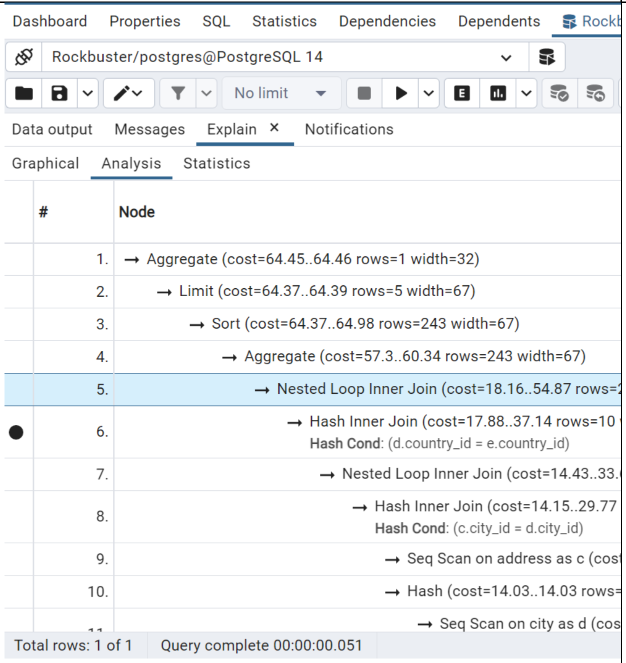
	country character varying (50)	all_customer_count bigint	top_customer_count bigint
1	Mexico	30	1
2	Turkey	15	1
3	India	60	1
4	Japan	31	1
5	United States	36	1

Total rows: 5 of 5   Query complete 00:00:00.268   Ln 26, Col 77

Juliana Lee

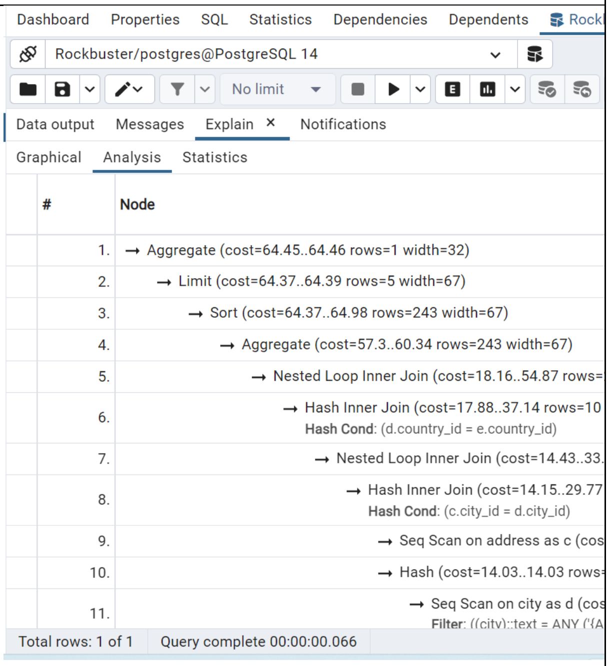
2)

Step 1 Subquery Query Plan:



```
#    Node
1.   → Aggregate (cost=64.45..64.46 rows=1 width=32)
2.     → Limit (cost=64.37..64.39 rows=5 width=67)
3.       → Sort (cost=64.37..64.98 rows=243 width=67)
4.         → Aggregate (cost=57.3..60.34 rows=243 width=67)
5.           → Nested Loop Inner Join (cost=18.16..54.87 rows=2
6.             → Hash Inner Join (cost=17.88..37.14 rows=10
               Hash Cond: (d.country_id = e.country_id)
7.               → Nested Loop Inner Join (cost=14.43..33.
8.                 → Hash Inner Join (cost=14.15..29.77
                   Hash Cond: (c.city_id = d.city_id)
9.                   → Seq Scan on address as c (cost
10.                  → Hash (cost=14.03..14.03 rows=
11.                    → Seq Scan on city as d (cos
Total rows: 1 of 1    Query complete 00:00:00.051
```

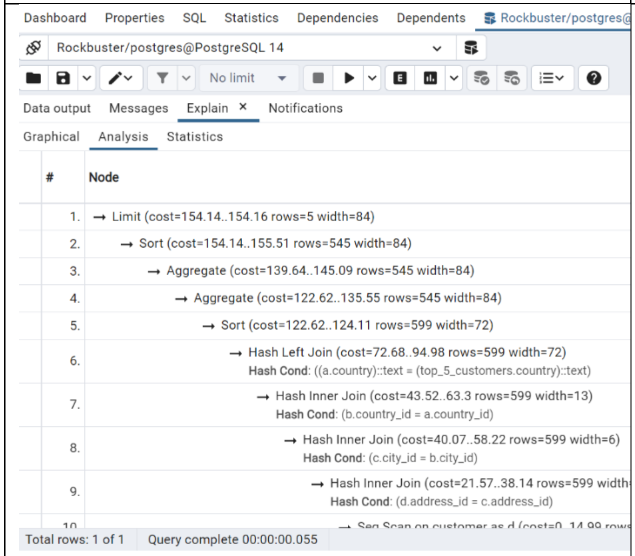
Step 1 CTE Query Plan:



```
#    Node
1.   → Aggregate (cost=64.45..64.46 rows=1 width=32)
2.     → Limit (cost=64.37..64.39 rows=5 width=67)
3.       → Sort (cost=64.37..64.98 rows=243 width=67)
4.         → Aggregate (cost=57.3..60.34 rows=243 width=67)
5.           → Nested Loop Inner Join (cost=18.16..54.87 rows=
6.             → Hash Inner Join (cost=17.88..37.14 rows=10
               Hash Cond: (d.country_id = e.country_id)
7.               → Nested Loop Inner Join (cost=14.43..33.
8.                 → Hash Inner Join (cost=14.15..29.77
                   Hash Cond: (c.city_id = d.city_id)
9.                   → Seq Scan on address as c (cos
10.                  → Hash (cost=14.03..14.03 rows=
11.                    → Seq Scan on city as d (cos
                       Filter: ((city)::text = ANY ('{A
Total rows: 1 of 1    Query complete 00:00:00.066
```

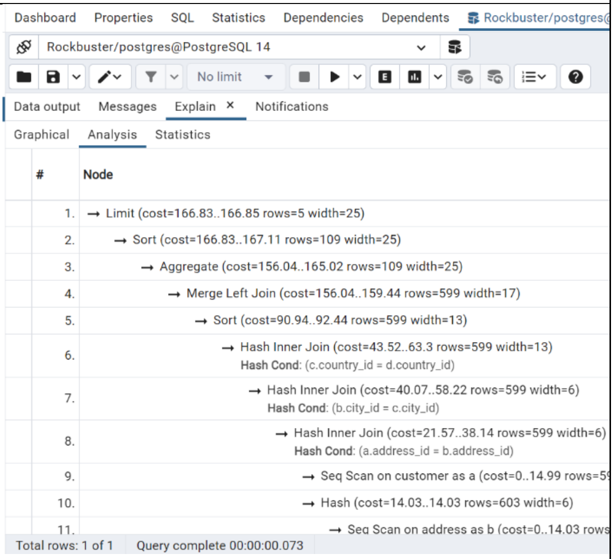
The cost for Step 1 was the same. However, the speed of the subquery was faster than the speed of the CTE by 15 milliseconds.

Step 2 Subquery Query Plan:



```
#    Node
1.   → Limit (cost=154.14..154.16 rows=5 width=84)
2.     → Sort (cost=154.14..155.51 rows=545 width=84)
3.       → Aggregate (cost=139.64..145.09 rows=545 width=84)
4.         → Aggregate (cost=122.62..135.55 rows=545 width=84)
5.           → Sort (cost=122.62..124.11 rows=599 width=72)
6.             → Hash Left Join (cost=72.68..94.98 rows=599 width=72)
               Hash Cond: ((a.country)::text = (top_5_customers.country)::text)
7.               → Hash Inner Join (cost=43.52..63.3 rows=599 width=13)
                 Hash Cond: (b.country_id = a.country_id)
8.                 → Hash Inner Join (cost=40.07..58.22 rows=599 width=6)
                   Hash Cond: (c.city_id = b.city_id)
9.                   → Hash Inner Join (cost=21.57..38.14 rows=599 width
                     Hash Cond: (d.address_id = c.address_id)
10.                    → Seq Scan on customer as d (cost=0..14.99 rows
Total rows: 1 of 1    Query complete 00:00:00.055
```

Step 2 CTE Query Plan:



```
#    Node
1.   → Limit (cost=166.83..166.85 rows=5 width=25)
2.     → Sort (cost=166.83..167.11 rows=109 width=25)
3.       → Aggregate (cost=156.04..165.02 rows=109 width=25)
4.         → Merge Left Join (cost=156.04..159.44 rows=599 width=17)
5.           → Sort (cost=90.94..92.44 rows=599 width=13)
6.             → Hash Inner Join (cost=43.52..63.3 rows=599 width=13)
               Hash Cond: (c.country_id = d.country_id)
7.               → Hash Inner Join (cost=40.07..58.22 rows=599 width=6)
                 Hash Cond: (b.city_id = c.city_id)
8.                 → Hash Inner Join (cost=21.57..38.14 rows=599 width=6)
                   Hash Cond: (a.address_id = b.address_id)
9.                   → Seq Scan on customer as a (cost=0..14.99 rows=5
10.                  → Hash (cost=14.03..14.03 rows=603 width=6)
11.                    → Seq Scan on address as b (cost=0..14.03 rows
Total rows: 1 of 1    Query complete 00:00:00.073
```

The cost for Step 2 were different. The subquery cost was lower than the cost of the CTE. Also, the subquery was 18 milliseconds faster than the CTE.

Juliana Lee

I thought the CTE approach would perform better because it was easier to read. However, the results surprised me because for both steps 1 and 2, the subquery was faster. It might be because the subquery is shorter than the CTE since it doesn’t have as many clauses or inner statements.

Step 3:

For step 1, replacing the subquery with a CTE was manageable. It was simple and easy to understand. It might be because for step 1 only one table was utilized. However, replacing the subquery with a CTE for step 2 was extremely difficult. Because step 2 dealt with two tables to combine I needed to be meticulous and accurate with which columns needed to be extracted and combined. I also had to write 2 CTE’s.