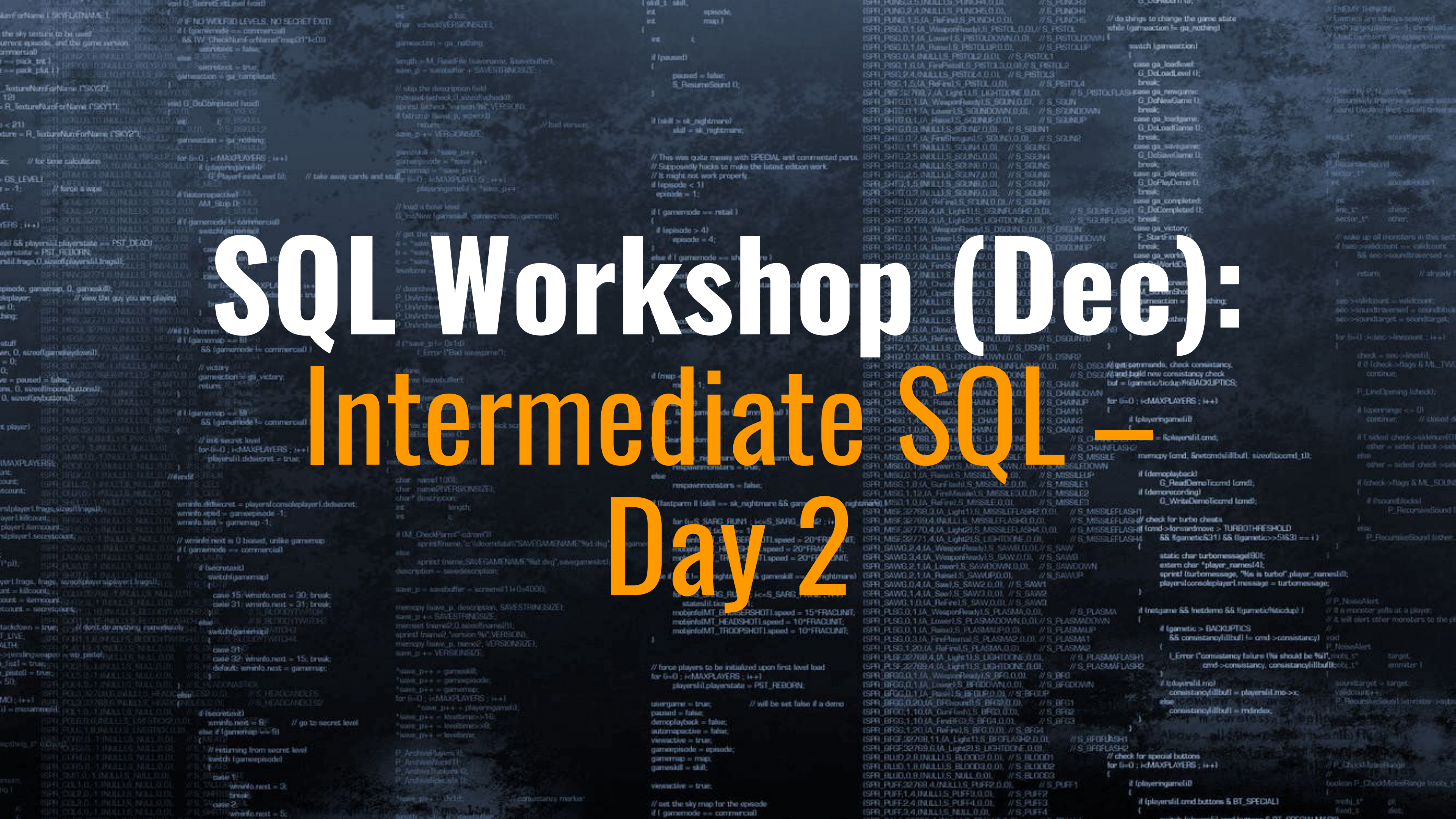


SQL Workshop (Dec):

Intermediate SQL – Day 2

SUBQUERY

- Also known as Inner Query, Nested Query is a form of SQL command that enables you to answer more complex questions within a database.

- These are highly relevant in most work setting, where the answers to various analytics questions aren't as simple as pulling data from a single table.

SUBQUERY

```
SELECT t3.sales_rep_name, t3.region_name, t3.total_amt
FROM
    -- SELECT max amount by region
    (SELECT region_name, MAX(total_amt) total_amt
    FROM
        -- SELECT rep and region, along with the total sale amt
        (SELECT s.name sale_rep_name, r.name region_name,
SUM(o.total_amt_usd) total_amt
            FROM accounts a
            LEFT JOIN sales_reps s
            ON a.sales_rep_id = s.id
            LEFT JOIN orders o
            ON a.id = o.id
            LEFT JOIN region r
            ON s.region_id = r.id
            GROUP BY 1, 2
            ORDER BY 3 DESC) t1
    GROUP BY 1) t2
LEFT JOIN
    (SELECT s.name sales_rep_name, r.name region_name, SUM(o.total_amt_usd)
total_amt
        FROM accounts a
        LEFT JOIN sales_reps s
            ON a.sales_rep_id = s.id
        LEFT JOIN orders o
            ON a.id = o.id
        LEFT JOIN region r
            ON s.region_id = r.id
        GROUP BY 1, 2
        ORDER BY 3 DESC) t3
    ON t2.region_name = t3.region_name AND t2.total_amt = t3.total_amt)
```

- On the left is an example of a subquery where we are trying to top selling sales rep based on the highest selling region.

- Taking a look at it, you would have it realise it is more complex that all the previous query we have wrote so far

SUBQUERY

So now, let’s begin writing a subquery based on the following question!

Considering that your boss just asked you to find the number of stores that have higher in-stock quantity for each product – which means that each store’s average stock qty (local by store) should be greater than the avg quantity across stores (global across all stores).

How you would tackle such a question?

SUBQUERY

To tackle a more complex SQL query, you often need to breakdown the question in smaller sub-question. For instance:

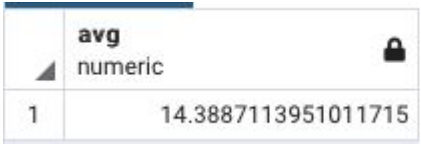
1. What is the average stock quantity across all stores and products? (as a whole)
2. What are the stores that have an average stock quantity greater than the amount in 1.?
3. How many stores fulfills such a condition?

By breaking down into smaller sub-questions, you can better query this one by one first, before combining them into a single big query.

SUBQUERY

To find the average across all stores and products,

```
SELECT
AVG(ps.quantity)
FROM
production.stocks
ps
```



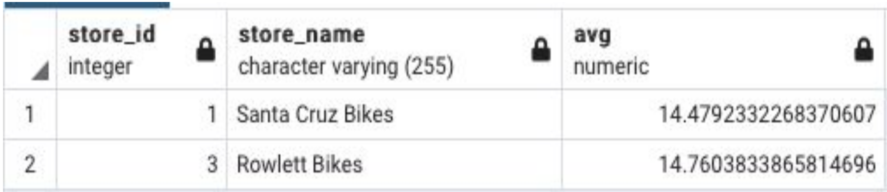
	avg numeric
1	14.3887113951011715

This returns us that average stock quantity of all products across store is ~14.388

SUBQUERY

To find the stores that has an average stock quantity greater than (~14.388),

```
SELECT ps.store_id, ss.store_name,
AVG(ps.quantity)
FROM production.stocks ps
JOIN sales.stores ss
ON ss.store_id = ps.store_id
GROUP BY 1, 2
HAVING AVG(ps.quantity) > (SELECT
AVG(quantity) FROM production.stocks)
```



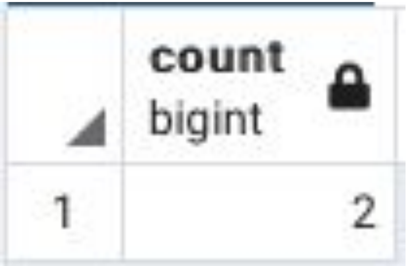
	store_id integer	store_name character varying (255)	avg numeric
1	1	Santa Cruz Bikes	14.4792332268370607
2	3	Rowlett Bikes	14.7603833865814696

This returns us two stores with an average stock quantity that is > ~14.388

SUBQUERY

To count the previous number of stores, you will need to use the previous query,

```
SELECT COUNT(t1.store_id) -- * means all cols
FROM (SELECT ps.store_id, ss.store_name,
AVG(ps.quantity)
       FROM production.stocks ps
       JOIN sales.stores ss
       ON ss.store_id = ps.store_id
       GROUP BY 1, 2
       HAVING AVG(ps.quantity) > (SELECT
AVG(quantity) FROM production.stocks)) t1
```



	count bigint
1	2

If you put a subquery under **FROM**, you will need to remember to have an alias as it needs to be in a table form.

As shown previously, using a simple count can aggregate and return the number of stores that fulfills our conditions.

SUBQUERY

Tips for writing subquery:

- Subquery gets complex easily, hence the need to emphasize on code readability with proper formatting to help others to understand what you are writing.

- Subquery formatting is crucial for you and others to better understand the logic you are writing and how each inner query contributes to running the outer query.

- In work setting, we can avoid subquery by writing in CTE which is similar to subquery but presents a more readable way to write complex query.

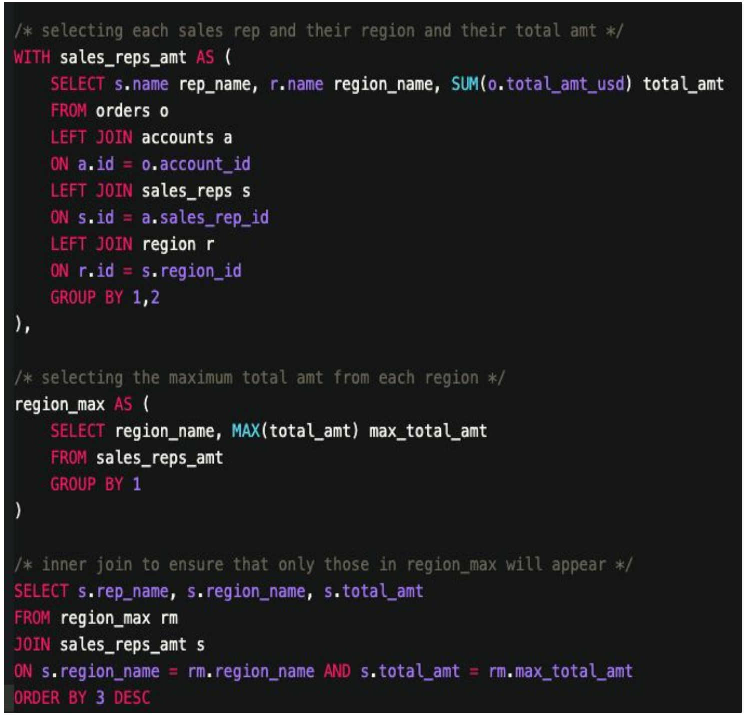
Common Table Expression (CTE)

After looking at subquery, you can see how if you are faced with a more complex question, the subquery can start to get messier and unreadable at times.

This is where we introduce CTE, where we can built *temporary* results set and use the results like a **TABLE** and build our query on those tables that we constructs

It may sound abstract now, but let’s take it a look on how we can use CTE to solve the two questions previously.

COMMON TABLE EXPRESSIONS (CTE)



```sql
/* selecting each sales rep and their region and their total amt */
WITH sales_reps_amt AS (
    SELECT s.name rep_name, r.name region_name, SUM(o.total_amt_usd) total_amt
    FROM orders o
    LEFT JOIN accounts a
    ON a.id = o.account_id
    LEFT JOIN sales_reps s
    ON s.id = a.sales_rep_id
    LEFT JOIN region r
    ON r.id = s.region_id
    GROUP BY 1,2
),

/* selecting the maximum total amt from each region */
region_max AS (
    SELECT region_name, MAX(total_amt) max_total_amt
    FROM sales_reps_amt
    GROUP BY 1
)

/* inner join to ensure that only those in region_max will appear */
SELECT s.rep_name, s.region_name, s.total_amt
FROM region_max rm
JOIN sales_reps_amt s
ON s.region_name = rm.region_name AND s.total_amt = rm.max_total_amt
ORDER BY 3 DESC
```

CTE, is often defined by the **WITH** operator, an example of a CTE can look at this:

If you can recall, this query serves the same purpose as the initial complex subquery that we introduced previously.

COMMON TABLE EXPRESSIONS (CTE)

So to better understand, let's tackle the two subquery questions we have previously, but instead we shall form query using CTE.

To recall the first question,

Considering that your boss just asked you to find the number of stores that are higher in stock for each product – which means that each store's average stock qty (local by store) should be greater than the avg quantity across stores (global across all stores).

COMMON TABLE EXPRESSIONS (CTE)

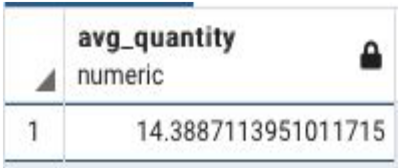
We will first build a temporary results to find the average stock quantity across all stores and products.

```
WITH avg_stock AS (
SELECT AVG(quantity) as avg_quantity
FROM production.stocks
)
```

Temp results set using **WITH** operator

```
—- to see what we building in avg_stock
SELECT *
FROM avg_stock
```

This is to check what you query in the temp results set



	avg_quantity numeric
1	14.3887113951011715

We are achieving a same number as previously in the subquery breakdown

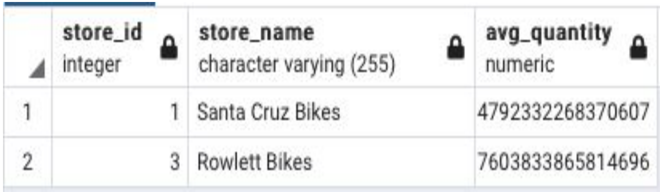
COMMON TABLE EXPRESSIONS (CTE)

Let’s continue building the temp results to show the stores that fulfills our conditions.

```
WITH avg_stock AS (
SELECT AVG(quantity) as avg_quantity
FROM production.stocks
),

more_than_avg AS (
SELECT ps.store_id, ss.store_name, AVG(ps.quantity) avg_quantity
FROM production.stocks ps
JOIN sales.stores ss
ON ss.store_id = ps.store_id
GROUP BY 1,2
HAVING AVG(ps.quantity) > (SELECT avg_quantity FROM avg_stock)
)

SELECT store_id, store_name, avg_quantity
FROM more_than_avg
```



	store_id integer	store_name character varying (255)	avg_quantity numeric
1	1	Santa Cruz Bikes	4792332268370607
2	3	Rowlett Bikes	7603833865814696

Seeing something familiar? We are achieving similar results without subquery

We added in a 2nd temp result set to filter for stores that satisfy our conditions

COMMON TABLE EXPRESSIONS (CTE)

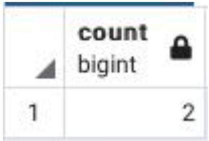
Now to simply find the number of stores, you just need count the final temp results set (*more_than_avg*)

```
WITH avg_stock AS (
SELECT AVG(quantity) as avg_quantity
FROM production.stocks
),

more_than_avg AS (
SELECT ps.store_id, ss.store_name, AVG(ps.quantity) avg_quantity
FROM production.stocks ps
JOIN sales.stores ss
ON ss.store_id = ps.store_id
GROUP BY 1,2
HAVING AVG(ps.quantity) > (SELECT avg_quantity FROM avg_stock)
)

SELECT COUNT(*)
FROM more_than_avg
```

Based on the CTE you built, now you query from them to find the number of stores.



	count bigint
1	2

Wow! Now we achieved the same results. Can you tell me how does this code look like as compared to before?

COMMON TABLE EXPRESSIONS (CTE)

Let’s look at the anatomy of a CTE!

```
      WITH avg_stock AS (
      SELECT AVG(quantity) as avg_quantity
      FROM production.stocks
      ),
```

Temp result 1:
finding the average
across all stores and
products

```
      more_than_avg AS (
      SELECT ps.store_id, ss.store_name, AVG(ps.quantity)
avg_quantity
      FROM production.stocks ps
      JOIN sales.stores ss
      ON ss.store_id = ps.store_id
      GROUP BY 1,2
      HAVING AVG(ps.quantity) > (SELECT avg_quantity FROM avg_stock)
      )
```

Temp results 2:
finding all the stores
that fulfills the
condition

```
      SELECT COUNT(*)
      FROM more_than_avg
```

Query using the CTE you
built above without touching
the tables in the database

COMMON TABLE EXPRESSION (CTE)

Let’s do a comparison between Subquery and CTE!

```
3
SELECT COUNT(*)
FROM (SELECT ps.store_id, ss.store_name,
AVG(ps.quantity)
       FROM production.stocks ps          2
       JOIN sales.stores ss
       ON ss.store_id = ps.store_id
       GROUP BY 1, 2
       HAVING AVG(ps.quantity) > (SELECT
AVG(quantity) FROM production.stocks)) t1

                               1
```

VS

```
WITH avg_stock AS (                           1
SELECT AVG(quantity) as avg_quantity
FROM production.stocks
),

more_than_avg AS (                                2
SELECT ps.store_id, ss.store_name, AVG(ps.quantity) avg_quantity
FROM production.stocks ps
JOIN sales.stores ss
ON ss.store_id = ps.store_id
GROUP BY 1,2
HAVING AVG(ps.quantity) > (SELECT avg_quantity FROM avg_stock)
)

SELECT COUNT(*)
FROM more_than_avg                        3
```

SUBQUERY

Now that you have a feel of how to approach a complex query, let's try another subquery!

Find the customers with the most number of orders and find the information about the customer (first and last name, email, number of orders). In the case where there are several customers with the same amount of orders, you will have to take the customer with the smallest customer id.

SUBQUERY

To breakdown the question,

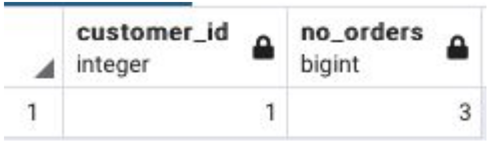
1. We need count each customer and their total orders, while including to sort them first by the number of order then their customer id
2. We need to pull the customer information from **CUSTOMERS** table as that consist of all the relevant information we need

Now, that seems slightly more simple than the previous one!

SUBQUERY

To find the one single customer, we will have to group by the customer id and count the instances in **ORDERS** table.

```
SELECT o.customer_id,
COUNT(*) as no_orders
FROM sales.orders o
GROUP BY 1
ORDER BY 2 DESC, 1 ASC
LIMIT 1
```



	customer_id integer	no_orders bigint
1	1	3

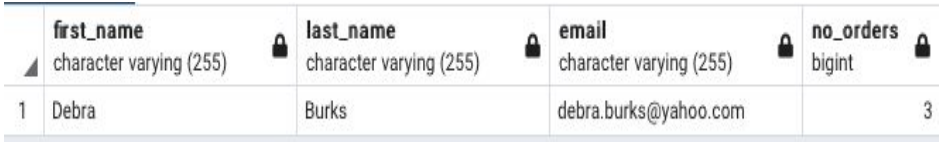
Seems like customer id 1 is the one we are finding!

SUBQUERY

To simply find the customer information, we just need to join the customer id in the previous query with the customer id in **CUSTOMERS** table.

```
SELECT c.first_name, c.last_name, c.email, t1.no_orders
FROM (
       SELECT o.customer_id, COUNT(*) as no_orders
       FROM sales.orders o
       GROUP BY 1
       ORDER BY 2 DESC, 1 ASC
       LIMIT 1
) t1
JOIN sales.customers c
ON c.customer_id = t1.customer_id
```

To find the customer id with the most orders



	first_name	last_name	email	no_orders
	character varying (255)	character varying (255)	character varying (255)	bigint
1	Debra	Burks	debra.burks@yahoo.com	3

To retrieve their details, we need to obtain it from **CUSTOMERS** table while joining on customer_id

COMMON TABLE EXPRESSIONS (CTE)

Let’s get another hands-on experience with querying using CTE. Let’s try to transform the 2nd subquery question using CTE.

To recall,

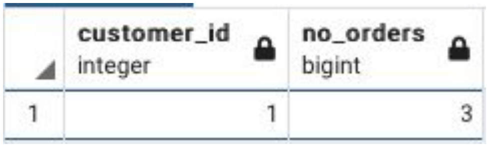
Find the customers with the most number of orders and find the information about the customer (first and last name, email, number of orders). In the case where there are several customers with the same amount of orders, you will have to take the customer with the smallest customer id.

COMMON TABLE EXPRESSIONS (CTE)

To begin, we shall figure out which are the customer that ordered the most and we sort them by their customer id.

```
WITH most_orders AS (
     SELECT o.customer_id, COUNT(*) as no_orders
     FROM sales.orders o
     GROUP BY 1
     ORDER BY 2 DESC, 1 ASC
     LIMIT 1
     )

SELECT customer_id, no_orders
FROM most_orders
```



	customer_id integer	no_orders bigint
1	1	3

Temp result set:
To find the customer that satisfy our condition

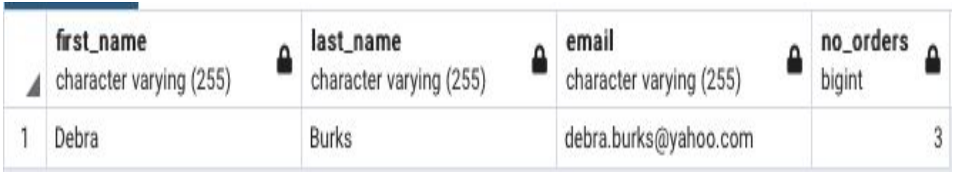
Querying from the CTE

COMMON TABLE EXPRESSIONS (CTE)

To find the customer information, we can actually join the data from **CUSTOMERS** table to the CTE itself!

```
WITH most_orders AS (
SELECT o.customer_id, COUNT(*) as no_orders
FROM sales.orders o
GROUP BY 1
ORDER BY 2 DESC, 1 ASC
LIMIT 1
)

SELECT c.first_name, c.last_name, c.email, mo.no_orders
FROM sales.customers c
JOIN most_orders mo
ON mo.customer_id = c.customer_id
```



	first_name character varying (255)	last_name character varying (255)	email character varying (255)	no_orders bigint
1	Debra	Burks	debra.burks@yahoo.com	3

You join the CTE with **CUSTOMERS** table using the common key = *customer_id*

COMMON TABLE EXPRESSIONS (CTE)

Let’s compare the subquery and CTE once again,

2

```
SELECT c.first_name, c.last_name, c.email, t1.no_orders
FROM (
      SELECT o.customer_id, COUNT(*) as no_orders
      FROM sales.orders o
      GROUP BY 1
      ORDER BY 2 DESC, 1 ASC
      LIMIT 1
) t1
JOIN sales.customers c
ON c.customer_id = t1.customer_id
```

1

VS

```
WITH most_orders AS (
SELECT o.customer_id, COUNT(*) as no_orders
FROM sales.orders o
GROUP BY 1
ORDER BY 2 DESC, 1 ASC
LIMIT 1
)
```

1

```
SELECT c.first_name, c.last_name, c.email, mo.no_orders
FROM sales.customers c
JOIN most_orders mo
ON mo.customer_id = c.customer_id
```

2

HANDS-ON

Using CTE, find the store with the most number of staffs, then calculate the number of orders the store has sold. [Do it with Subquery as well]

TO RECAP

So what exactly have you learn today?

- You have moved on from just simple querying by learning new querying method such as subquery and CTE to address complex analysis
- Subquery involves having another query nested within that provides you a table or value that you require to obtain and run your main query
- CTE is similar to subquery and often, provides a more readable code to better understand the flow of logic

ADDITIONAL RESOURCES

Here are some resources that have personally helped me to learn subqueries and CTES:

Subquery:

https://www.w3resource.com/sql/subqueries/understanding-sql-subqueries.php

CTE:

https://www.essentialsql.com/introduction-common-table-expressions-ctes/

END OF DAY 2!

Any questions? Feel free to clarify now.

Or you can reach us at:

Wei Teck (Jensen): wtlow003@suss.edu.sg / https://www.linkedin.com/in/weitecklow/

Jeanette: jeanettekoh001@suss.edu.sg / linkedin.com/in/jeanette-koh-872b64192