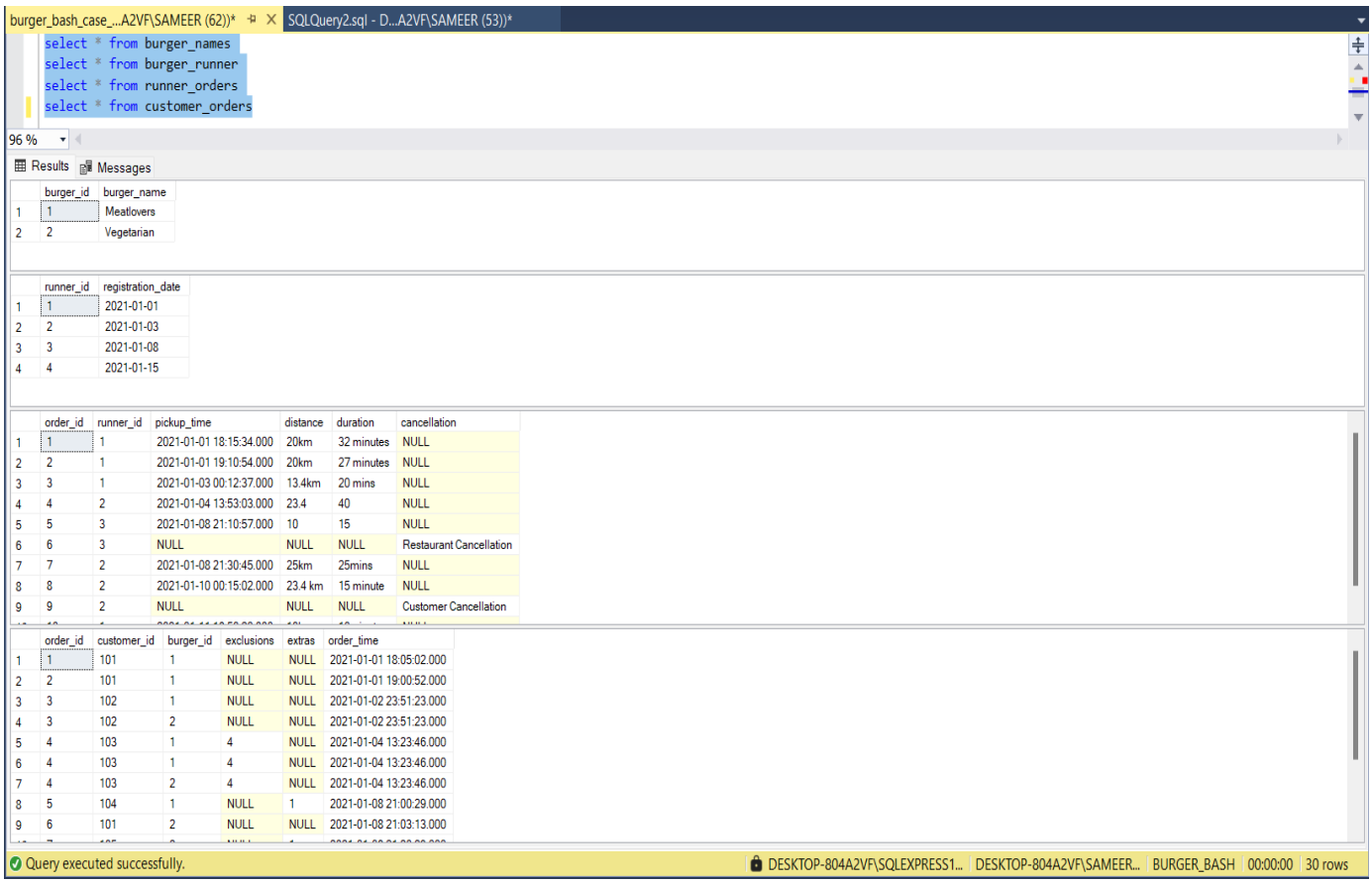
Coding Challenge

Burger Bash

DE131 Sameer Pal

Database Schema



```
select * from burger_names
select * from burger_runner
select * from runner_orders
select * from customer_orders
```

96 %

Results | Messages

	burger_id	burger_name
1	1	Meatlovers
2	2	Vegetarian

	runner_id	registration_date
1	1	2021-01-01
2	2	2021-01-03
3	3	2021-01-08
4	4	2021-01-15

	order_id	runner_id	pickup_time	distance	duration	cancellation
1	1	1	2021-01-01 18:15:34.000	20km	32 minutes	NULL
2	2	1	2021-01-01 19:10:54.000	20km	27 minutes	NULL
3	3	1	2021-01-03 00:12:37.000	13.4km	20 mins	NULL
4	4	2	2021-01-04 13:53:03.000	23.4	40	NULL
5	5	3	2021-01-08 21:10:57.000	10	15	NULL
6	6	3	NULL	NULL	NULL	Restaurant Cancellation
7	7	2	2021-01-08 21:30:45.000	25km	25mins	NULL
8	8	2	2021-01-10 00:15:02.000	23.4 km	15 minute	NULL
9	9	2	NULL	NULL	NULL	Customer Cancellation

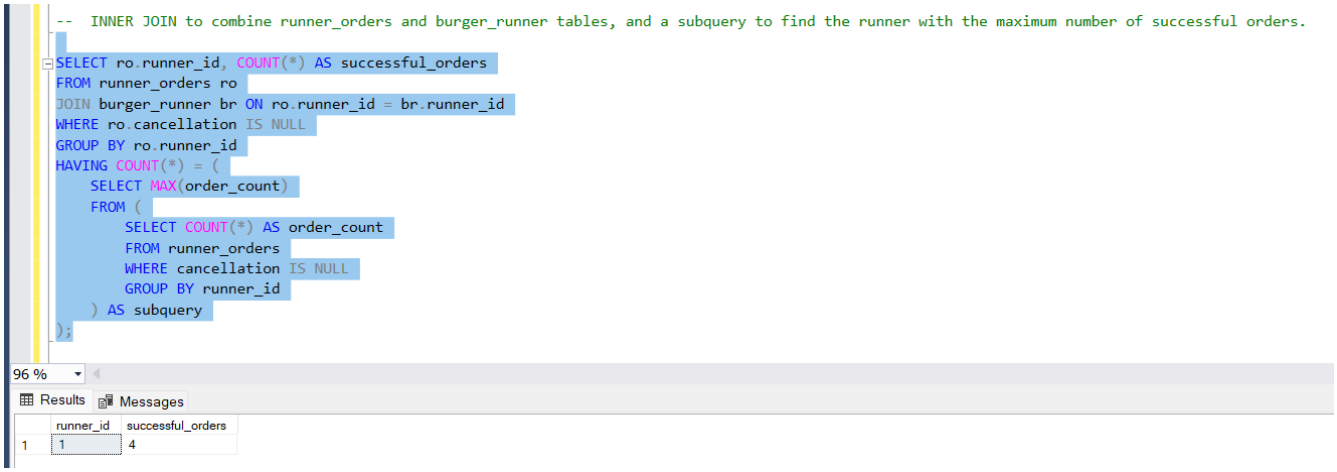
	order_id	customer_id	burger_id	exclusions	extras	order_time
1	1	101	1	NULL	NULL	2021-01-01 18:05:02.000
2	2	101	1	NULL	NULL	2021-01-01 19:00:52.000
3	3	102	1	NULL	NULL	2021-01-02 23:51:23.000
4	3	102	2	NULL	NULL	2021-01-02 23:51:23.000
5	4	103	1	4	NULL	2021-01-04 13:23:46.000
6	4	103	1	4	NULL	2021-01-04 13:23:46.000
7	4	103	2	4	NULL	2021-01-04 13:23:46.000
8	5	104	1	NULL	1	2021-01-08 21:00:29.000
9	6	101	2	NULL	NULL	2021-01-08 21:03:13.000

Query executed successfully. DESKTOP-804A2VF\SQLEXPRESS1... DESKTOP-804A2VF\SAMEER... BURGER_BASH 00:00:00 30 rows

Part-1

Querying Data by Using Joins and Subqueries & subtotal

INNER JOIN to combine runner_orders and burger_runner tables, and a subquery to find the runner with the maximum number of successful orders.



```sql
--  INNER JOIN to combine runner_orders and burger_runner tables, and a subquery to find the runner with the maximum number of successful orders.

SELECT ro.runner_id, COUNT(*) AS successful_orders
FROM runner_orders ro
JOIN burger_runner br ON ro.runner_id = br.runner_id
WHERE ro.cancellation IS NULL
GROUP BY ro.runner_id
HAVING COUNT(*) = (
    SELECT MAX(order_count)
    FROM (
        SELECT COUNT(*) AS order_count
        FROM runner_orders
        WHERE cancellation IS NULL
        GROUP BY runner_id
    ) AS subquery
);
```

96 %

Results | Messages

	runner_id	successful_orders
1	1	4

LEFT JOIN is used to combine customer_orders with burger_names, and aggregates burger orders for each customer

```sql
---LEFT JOIN is used to combine customer_orders with burger_names, and aggregates burger orders for each customer

SELECT c.customer_id, b.burger_name, COUNT(*) AS burger_count
FROM customer_orders c
LEFT JOIN burger_names b ON c.burger_id = b.burger_id
GROUP BY c.customer_id, b.burger_name
ORDER BY c.customer_id;
```

96 %

Results | Messages

	customer_id	burger_name	burger_count
1	101	Meatlovers	2
2	101	Vegetarian	1
3	102	Meatlovers	2
4	102	Vegetarian	1
5	103	Meatlovers	3
6	103	Vegetarian	1
7	104	Meatlovers	3
8	105	Vegetarian	1

Uses a RIGHT JOIN between burger_names and customer_orders, and an INNER JOIN with runner_orders to combine burger and order data.

```sql
--Uses a RIGHT JOIN between burger_names and customer_orders, and an INNER JOIN with runner_orders to combine burger and order data.

SELECT b.burger_name, ro.order_id, ro.runner_id, ro.pickup_time, ro.cancellation
FROM burger_names b
RIGHT JOIN customer_orders c ON b.burger_id = c.burger_id
JOIN runner_orders ro ON c.order_id = ro.order_id;
```

96 %

Results | Messages

	burger_name	order_id	runner_id	pickup_time	cancellation
1	Meatlovers	1	1	2021-01-01 18:15:34.000	NULL
2	Meatlovers	2	1	2021-01-01 19:10:54.000	NULL
3	Meatlovers	3	1	2021-01-03 00:12:37.000	NULL
4	Vegetarian	3	1	2021-01-03 00:12:37.000	NULL
5	Meatlovers	4	2	2021-01-04 13:53:03.000	NULL
6	Meatlovers	4	2	2021-01-04 13:53:03.000	NULL
7	Vegetarian	4	2	2021-01-04 13:53:03.000	NULL
8	Meatlovers	5	3	2021-01-08 21:10:57.000	NULL
9	Vegetarian	6	3	NULL	Restaurant Cancellation
10	Vegetarian	7	2	2021-01-08 21:30:45.000	NULL
11	Meatlovers	8	2	2021-01-10 00:15:02.000	NULL
12	Meatlovers	9	2	NULL	Customer Cancellation
13	Meatlovers	10	1	2021-01-11 18:50:20.000	NULL
14	Meatlovers	10	1	2021-01-11 18:50:20.000	NULL

Self-join on customer_orders is used to find customers who have ordered more than one distinct burger in a single order.

```sql
--A self-join on customer_orders is used to find customers who have ordered more than one distinct burger in a single order.

SELECT a.order_id, a.customer_id, COUNT(*) AS burger_count
FROM customer_orders a
JOIN customer_orders b ON a.order_id = b.order_id
WHERE a.customer_id = b.customer_id
GROUP BY a.order_id, a.customer_id
HAVING COUNT(DISTINCT a.burger_id) > 1;
```

96 %

Results | Messages

	order_id	customer_id	burger_count
1	3	102	4
2	4	103	9

Uses a JOIN with a subquery that counts burgers per order, then finds the maximum burger count for each customer.

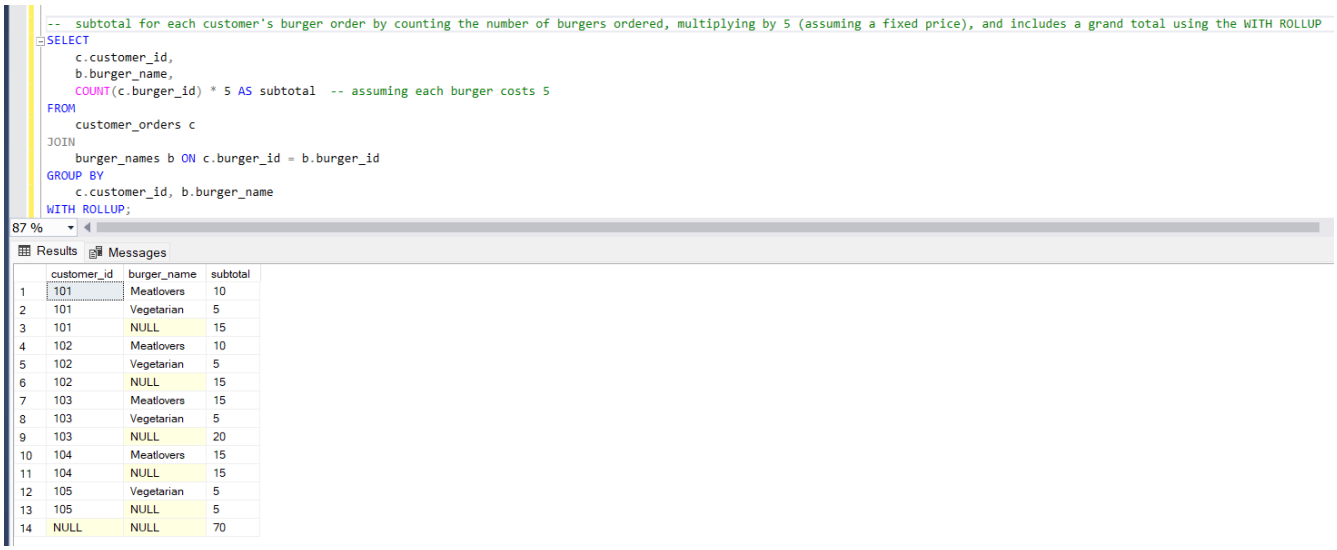
```sql
--Uses a JOIN with a subquery that counts burgers per order, then finds the maximum burger count for each customer.

SELECT co.customer_id, MAX(order_counts.burger_count) AS max_burgers
FROM customer_orders co
JOIN (
    SELECT order_id, COUNT(*) AS burger_count
    FROM customer_orders
    GROUP BY order_id
) AS order_counts ON co.order_id = order_counts.order_id
GROUP BY co.customer_id;
```

96 %

Results | Messages

	customer_id	max_burgers
1	101	1
2	102	2
3	103	3
4	104	2
5	105	1

Subtotal for each customer's burger order by counting the number of burgers ordered, multiplying by 5 (assuming a fixed price), and includes a grand total using the WITH ROLLUP



```sql
--  subtotal for each customer's burger order by counting the number of burgers ordered, multiplying by 5 (assuming a fixed price), and includes a grand total using the WITH ROLLUP
SELECT
    c.customer_id,
    b.burger_name,
    COUNT(c.burger_id) * 5 AS subtotal  -- assuming each burger costs 5
FROM
    customer_orders c
JOIN
    burger_names b ON c.burger_id = b.burger_id
GROUP BY
    c.customer_id, b.burger_name
WITH ROLLUP;
```

87 %

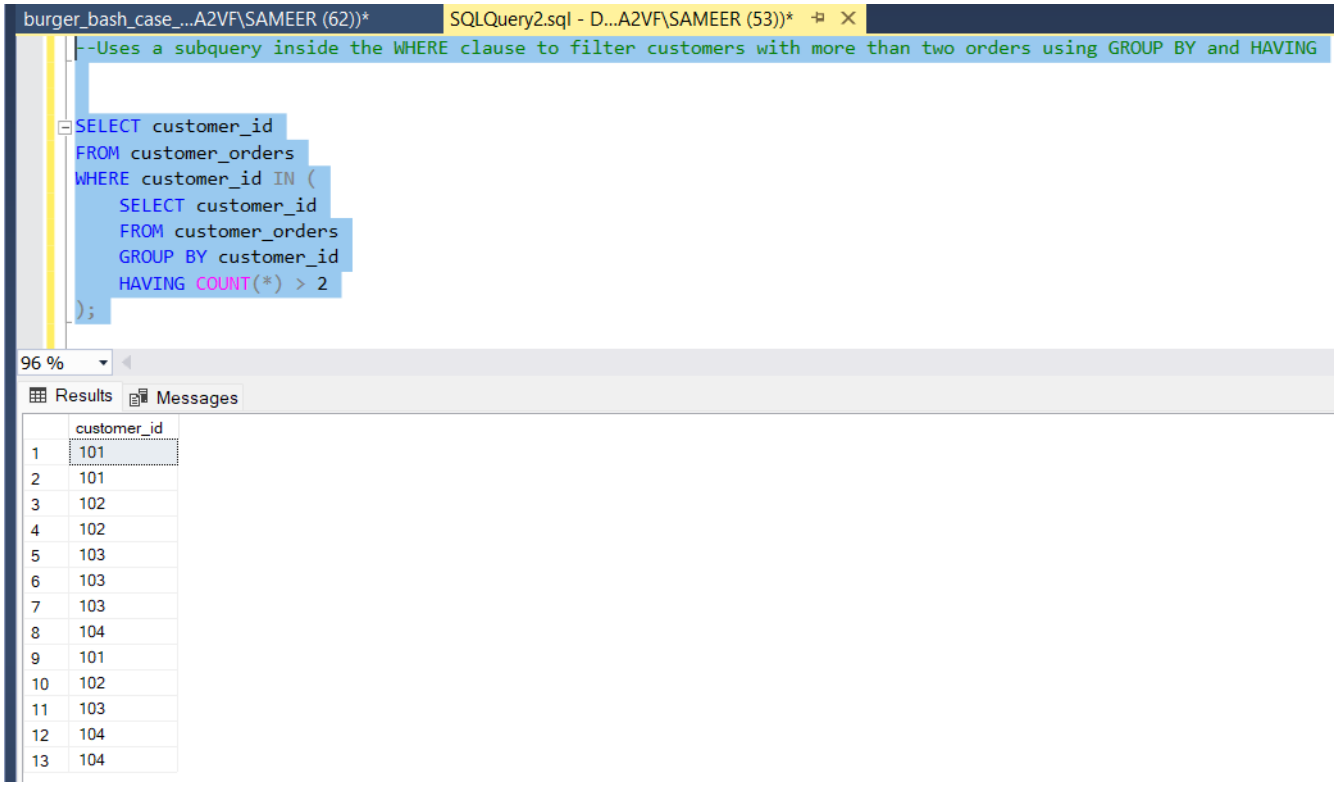
Results | Messages

	customer_id	burger_name	subtotal
1	101	Meatlovers	10
2	101	Vegetarian	5
3	101	NULL	15
4	102	Meatlovers	10
5	102	Vegetarian	5
6	102	NULL	15
7	103	Meatlovers	15
8	103	Vegetarian	5
9	103	NULL	20
10	104	Meatlovers	15
11	104	NULL	15
12	105	Vegetarian	5
13	105	NULL	5
14	NULL	NULL	70

Part-2

Manipulate data by using sql commands using groupby and having clause

Uses a subquery inside the WHERE clause to filter customers with more than two orders using GROUP BY and HAVING



Uses GROUP BY and HAVING to filter customers who have modified their orders (with exclusions or extras) more than twice.

```sql
-- Uses GROUP BY and HAVING to filter customers who have modified their orders (with exclusions or extras) more than twice.

SELECT customer_id,
       COUNT(*) AS modified_orders
FROM customer_orders
WHERE exclusions IS NOT NULL OR extras IS NOT NULL
GROUP BY customer_id
HAVING COUNT(*) > 2;
```

96 %

Results | Messages

	customer_id	modified_orders
1	103	4

Uses GROUP BY to calculate the average delivery distance per customer, ordering by the highest value and returning the top (TOP-1)result.

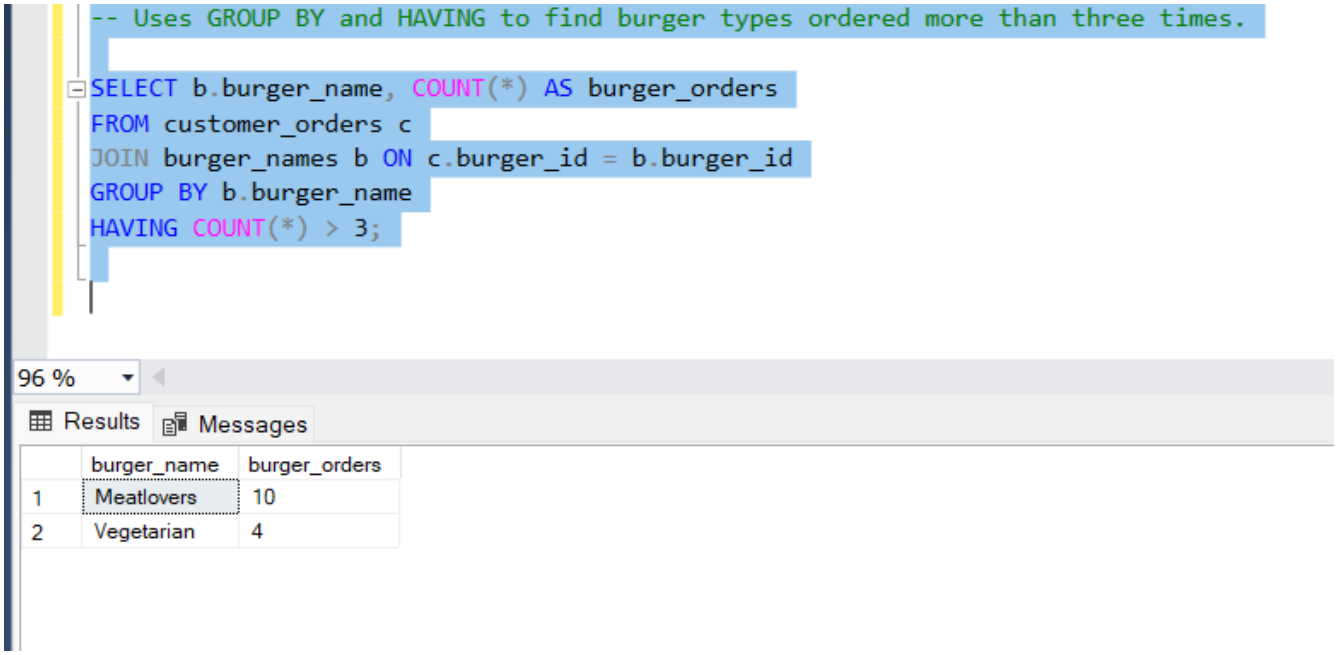
```sql
--Uses GROUP BY to calculate the average delivery distance per customer, ordering by the highest value and returning the top (TOP-1)result.
SELECT TOP 1 co.customer_id,
             AVG(CAST(REPLACE(ro.distance, 'km', '') AS NUMERIC)) AS avg_distance
FROM customer_orders co
JOIN runner_orders ro ON co.order_id = ro.order_id
WHERE ro.cancellation IS NULL
GROUP BY co.customer_id
ORDER BY avg_distance DESC;
```

96 %

Results | Messages

	customer_id	avg_distance
1	105	25.000000

Uses GROUP BY and HAVING to find burger types ordered more than three times.



```sql
-- Uses GROUP BY and HAVING to find burger types ordered more than three times.

SELECT b.burger_name, COUNT(*) AS burger_orders
FROM customer_orders c
JOIN burger_names b ON c.burger_id = b.burger_id
GROUP BY b.burger_name
HAVING COUNT(*) > 3;
```

96 %

Results | Messages

	burger_name	burger_orders
1	Meatlovers	10
2	Vegetarian	4