Coding Challenge – 1 (Sql)
Burger Bash

Name: Aathirainathan P

Date: 08-11-2024

I. Creating the Database & Tables:

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

97 %

Results | Messages

	burger_id	burger_name
1	1	Meatlovers
2	2	Vegetarian

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

	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	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

The four tables that were necessary were created.

II. Topics:

1. Querying Data by Using Joins and Subqueries & Subtotal:

- **SQL Joins:**

  - SQL Joins are used to combine rows from two or more tables based on related fields between them. When we work with multiple tables, joins allow us to retrieve related data across those tables.

  - ***Types:***

    1. Inner Join
    2. Left Join
    3. Right Join
    4. Outer Join
    5. Cross Join
    6. Self Join

Query 1: Get the list of orders with order id, customer id, runner id and burger names:

```sql
SELECT co.order_id, co.customer_id, bn.burger_name, br.runner_id
FROM customer_orders co
JOIN burger_names bn ON co.burger_id = bn.burger_id
JOIN runner_orders ro ON co.order_id = ro.order_id
JOIN burger_runner br ON ro.runner_id = br.runner_id;
```

97 %

Results | Messages

	order_id	customer_id	burger_name	runner_id
1	1	101	Meatlovers	1
2	2	101	Meatlovers	1
3	3	102	Meatlovers	1
4	3	102	Vegetarian	1
5	4	103	Meatlovers	2
6	4	103	Meatlovers	2
7	4	103	Vegetarian	2
8	5	104	Meatlovers	3
9	6	101	Vegetarian	3
10	7	105	Vegetarian	2
11	8	102	Meatlovers	2
12	9	103	Meatlovers	2
13	10	104	Meatlovers	1
14	10	104	Meatlovers	1

*This query joins four tables () to return the details of customer orders along with **burger names and runner IDs**.*

Query 2: Get a list of customers and their corresponding runner details along with order time and pickup time.

```sql
SELECT co.customer_id, co.order_id,  ro.runner_id,co.order_time, ro.pickup_time
FROM customer_orders co
JOIN runner_orders ro ON co.order_id = ro.order_id
ORDER BY co.order_time;
```

88 %

Results | Messages

	customer_id	order_id	runner_id	order_time	pickup_time
1	101	1	1	2021-01-01 18:05:02.000	2021-01-01 18:15:34.000
2	101	2	1	2021-01-01 19:00:52.000	2021-01-01 19:10:54.000
3	102	3	1	2021-01-02 23:51:23.000	2021-01-03 00:12:37.000
4	102	3	1	2021-01-02 23:51:23.000	2021-01-03 00:12:37.000
5	103	4	2	2021-01-04 13:23:46.000	2021-01-04 13:53:03.000
6	103	4	2	2021-01-04 13:23:46.000	2021-01-04 13:53:03.000
7	103	4	2	2021-01-04 13:23:46.000	2021-01-04 13:53:03.000
8	104	5	3	2021-01-08 21:00:29.000	2021-01-08 21:10:57.000
9	101	6	3	2021-01-08 21:03:13.000	NULL
10	105	7	2	2021-01-08 21:20:29.000	2021-01-08 21:30:45.000
11	102	8	2	2021-01-09 23:54:33.000	2021-01-10 00:15:02.000
12	103	9	2	2021-01-10 11:22:59.000	NULL
13	104	10	1	2021-01-11 18:34:49.000	2021-01-11 18:50:20.000
14	104	10	1	2021-01-11 18:34:49.000	2021-01-11 18:50:20.000

*This query returns customer order details along with the **runner ID and pickup time**, showing the chronological order of orders.*

- **SubQuery:**
  - A **subquery** is a query within another query. Subqueries can be used to perform operations that depend on the results of another query.
  - **A correlated subquery** is a subquery that refers to columns of the outer query, making it dependent on the outer query.

Query 3: Find the customers who have ordered both burgers:

```sql
SELECT DISTINCT customer_id
FROM customer_orders
WHERE customer_id IN (
    SELECT customer_id
    FROM customer_orders
    WHERE burger_id = 1
) AND customer_id IN (
    SELECT customer_id
    FROM customer_orders
    WHERE burger_id = 2
);
```

88 %

Results | Messages

	customer_id
1	101
2	102
3	103

*This query finds customers who have ordered **both burger ID 1 and burger ID 2** by using two subqueries. The outer query checks if a customer exists in both subquery results.*

Query 4: Find the runner who handled the most orders:

```sql
SELECT runner_id
FROM runner_orders
GROUP BY runner_id
HAVING COUNT(order_id) = (
    SELECT MAX(order_count)
    FROM (
        SELECT runner_id, COUNT(order_id) AS order_count
        FROM runner_orders
        GROUP BY runner_id
    ) AS runner_order_counts
);
```

97 %

Results | Messages

	runner_id
1	1
2	2

*This query uses a subquery to first calculate the order count for each runner, and then finds the runner with the **highest number of orders**.*

- **Subtotals:**
  - A **subtotal** is a partial sum, which is grouped by certain attributes. Subtotals can be calculated using **group by and aggregate functions** like SUM(), AVG(), COUNT(), etc.

Query 5: Calculate the total number of orders handled by each runner and show the subtotal for each runner:

```sql
SELECT br.runner_id, COUNT(ro.order_id) AS total_orders
FROM runner_orders ro
JOIN burger_runner br ON ro.runner_id = br.runner_id
GROUP BY br.runner_id WITH ROLLUP
ORDER BY br.runner_id;
```

88 %

Results | Messages

	runner_id	total_orders
1	NULL	10
2	1	4
3	2	4
4	3	2

This query returns the total number of orders handled by each runner by grouping the results by ***runner_id*** *and using the* ***COUNT()*** *function to calculate the number of orders per runner*

2. Manipulating Data Using SQL Commands: GROUP BY and HAVING Clause

- **Groupby and Having Clause:**
  - *The GROUP BY clause is used to arrange identical data into groups, usually with aggregate functions like* ***SUM(), COUNT(), AVG(),*** *etc. The* HAVING *clause is used to filter the results after the grouping, whereas WHERE filters the rows before grouping.*
  - *W*HERE is used to filter rows **before** aggregation.
  - HAVING is used to filter groups **after** aggregation.

Query 6: Get the number of orders per customer where the total is **greater than 1**.

```sql
SELECT customer_id, COUNT(order_id) AS total_orders
FROM customer_orders
GROUP BY customer_id
HAVING COUNT(order_id) > 1;
```

88 %

Results | Messages

	customer_id	total_orders
1	101	3
2	102	3
3	103	4
4	104	3

This query groups the ***customer_orders by customer_id*** *and filters the results using* ***HAVING*** *to only include customers who have placed more than one order.*

Query 7: Get the number of orders per burger type, only for those with more than 2 orders.

```sql
SELECT burger_id, COUNT(order_id) AS total_orders
FROM customer_orders
GROUP BY burger_id
HAVING COUNT(order_id) > 2;
```

88 %

Results | Messages

	burger_id	total_orders
1	1	10
2	2	4

*This query groups the orders by **burger_id** and filters them to only include burger types with **more than two orders**.*

Submitted by:
Aathirainathan P