MS-SQL Coding Challenge

-Sarthak Niranjan Kulkarni (Maverick)

- <u>sarthakkul2311@gmail.com</u> - (+91) 93256 02791

Day 5 - 8/11/2024 (Friday)

(1) Querying Data by Using Joins and Subqueries & subtotal: -

1. What is the total count of each type of burger ordered by customers, along with the total count of all burgers ordered?

 \rightarrow

- JOIN: This query uses an INNER JOIN between the customer_orders table (aliased as co) and the burger names table (aliased as b).
- Grouping: The GROUP BY b.burger_name groups the results by each unique burger_name.
- ROLLUP: with rollup adds an extra row at the end of the result set to show a subtotal (or grand total) across all groups.

```
-- 1. What is the total count of each type of burger ordered by customers, along with the total count of all burgers ordered?
    --- WITH ROLLUP: Creates a subtotal row at the end to show the total count of all burgers ordered.
    --- JOIN: Joins customer_orders with burger_names on burger_id to match burger names.
    --- GROUP BY: Groups by burger_name to count each burger type.
  SELECT b.burger_name, COUNT(co.order_id) AS burger_count
    FROM customer_orders co
    JOIN burger_names b ON co.burger_id = b.burger_id
    GROUP BY b.burger_name
    WITH ROLLUP;
100 % + 4
Results Messages
     burger_name burger_count
    Meatlovers
               10
     Vegetarian
                4
    NULL
```

2. Which customers ordered both 'Meatlovers' and 'Vegetarian' burgers?

 \rightarrow

• JOIN: This query uses an INNER JOIN between customer_orders (aliased as co) and burger names (aliased as b).

- Filtering: The WHERE clause filters the records to include only orders for burgers named "Meatlovers" and "Vegetarian".
- Grouping: The GROUP BY customer_id groups the results by each unique customer id.

3. For each runner, what is the total number of deliveries they completed, and the total number of orders they couldn't deliver due to cancellations?

 \rightarrow

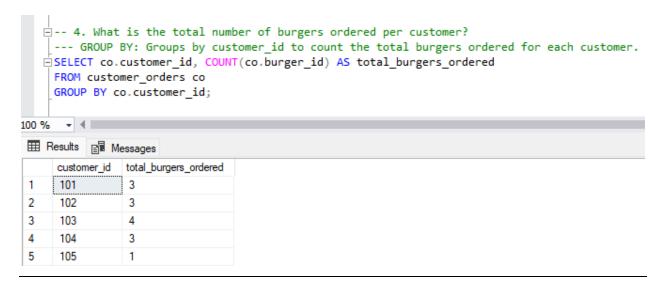
- SUM with CASE: The SUM function with CASE is used to count the completed and canceled deliveries separately.
- Grouping: The GROUP BY ro.runner_id groups the results by each unique runner_id.

4. What is the total number of burgers ordered per customer?

 \rightarrow

• COUNT: COUNT(co.burger_id) counts the number of burgers ordered by each customer.

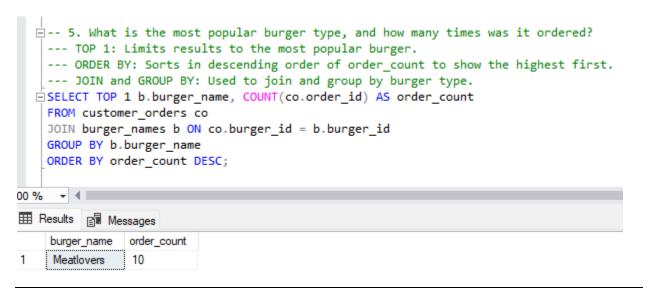
• Grouping: The GROUP BY co.customer_id groups the results by each unique customer id.



5. What is the most popular burger type, and how many times was it ordered?

 \rightarrow

- JOIN: An INNER JOIN is used between customer orders (co) and burger names (b).
- COUNT: COUNT(co.order_id) counts the number of times each burger type was ordered.
- Grouping: The GROUP BY b.burger_name groups the results by each unique burger type.

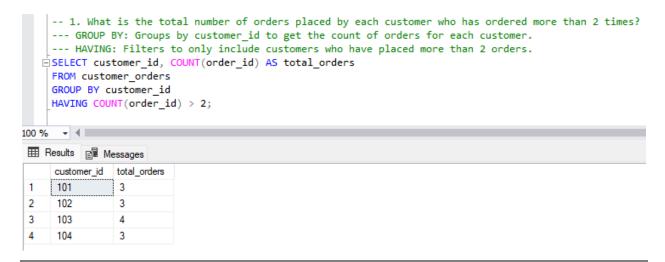


(2) Manipulate data by using sql commands using groupby and having clause: -

1. What is the total number of orders placed by each customer who has ordered more than 2 times?

 \rightarrow

- COUNT: COUNT(order id) counts the number of orders placed by each customer.
- Grouping: The GROUP BY customer_id groups the results by each unique customer id.
- HAVING: The HAVING clause filters the results to only include customers who have placed more than 2 orders.



2. Find the customers who have ordered more than one unique type of burger.

 \rightarrow

- COUNT: COUNT (DISTINCT burger_id) is calculated to determine how many unique types of burgers that customer has ordered.
- HAVING: The HAVING clause is used after the GROUP BY to filter the groups. It
 ensures that only those customers who ordered more than one unique type of burger
 are included in the result.

```
\dot{\exists} -- 2. Find the customers who have ordered more than one unique type of burger.
     --- COUNT(DISTINCT burger_id): Counts the unique types of burgers each customer ordered.
     --- HAVING: Filters to include only customers who ordered more than one unique type of burger.
  SELECT customer_id, COUNT(DISTINCT burger_id) AS unique_burgers_ordered
     FROM customer_orders
     GROUP BY customer_id
    HAVING COUNT(DISTINCT burger_id) > 1;
100 % - 4
Results Messages
     customer_id unique_burgers_ordered
                2
 2
                 2
      102
 3
                 2
     103
```