MS-SQL Coding Challenge

-Sarthak Niranjan Kulkarni (Maverick)

- sarthakkul2311@gmail.com - (+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?**

→

- 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 %

Results | Messages

	burger_name	burger_count
1	Meatlovers	10
2	Vegetarian	4
3	NULL	14

---

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

→

- 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.

```sql
-- 2. Which customers ordered both 'Meatlovers' and 'Vegetarian' burgers?
--- JOIN: Joins customer_orders with burger_names to get the burger names.
--- WHERE: Filters for only "Meatlovers" and "Vegetarian" burgers.
--- GROUP BY and HAVING: Ensures only customers who ordered both types of burgers are included by counting distinct burger names.
SELECT customer_id
FROM customer_orders co
JOIN burger_names b ON co.burger_id = b.burger_id
WHERE b.burger_name IN ('Meatlovers', 'Vegetarian')
GROUP BY customer_id
HAVING COUNT(DISTINCT b.burger_name) = 2;
```

100 %

Results | Messages

	customer_id
1	101
2	102
3	103

---

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?

→

- 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.

```sql
-- 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?
--- SUM with CASE: Uses CASE to count completed and canceled deliveries separately.
--- GROUP BY: Groups by runner_id to get totals per runner.
SELECT ro.runner_id,
       SUM(CASE WHEN ro.cancellation IS NULL THEN 1 ELSE 0 END) AS completed_deliveries,
       SUM(CASE WHEN ro.cancellation IS NOT NULL THEN 1 ELSE 0 END) AS cancelled_deliveries
FROM runner_orders ro
GROUP BY ro.runner_id;
```

)0 %

Results | Messages

	runner_id	completed_deliveries	cancelled_deliveries
1	1	4	0
2	2	3	1
3	3	1	1

---

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

→

- 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.

```sql
-- 4. What is the total number of burgers ordered per customer?
--- GROUP BY: Groups by customer_id to count the total burgers ordered for each customer.
SELECT co.customer_id, COUNT(co.burger_id) AS total_burgers_ordered
FROM customer_orders co
GROUP BY co.customer_id;
```

100 %

Results | Messages

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

---

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

→

- 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.

```sql
-- 5. What is the most popular burger type, and how many times was it ordered?
--- TOP 1: Limits results to the most popular burger.
--- ORDER BY: Sorts in descending order of order_count to show the highest first.
--- JOIN and GROUP BY: Used to join and group by burger type.
SELECT TOP 1 b.burger_name, COUNT(co.order_id) AS order_count
FROM customer_orders co
JOIN burger_names b ON co.burger_id = b.burger_id
GROUP BY b.burger_name
ORDER BY order_count DESC;
```

00 %

Results | Messages

	burger_name	order_count
1	Meatlovers	10

---

(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?**

→

- 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.

```
-- 1. What is the total number of orders placed by each customer who has ordered more than 2 times?
--- GROUP BY: Groups by customer_id to get the count of orders for each customer.
--- HAVING: Filters to only include customers who have placed more than 2 orders.
SELECT customer_id, COUNT(order_id) AS total_orders
FROM customer_orders
GROUP BY customer_id
HAVING COUNT(order_id) > 2;
```

100 %

Results | Messages

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

---

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

→

- 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.

```sql
-- 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 %

Results | Messages

	customer_id	unique_burgers_ordered
1	101	2
2	102	2
3	103	2