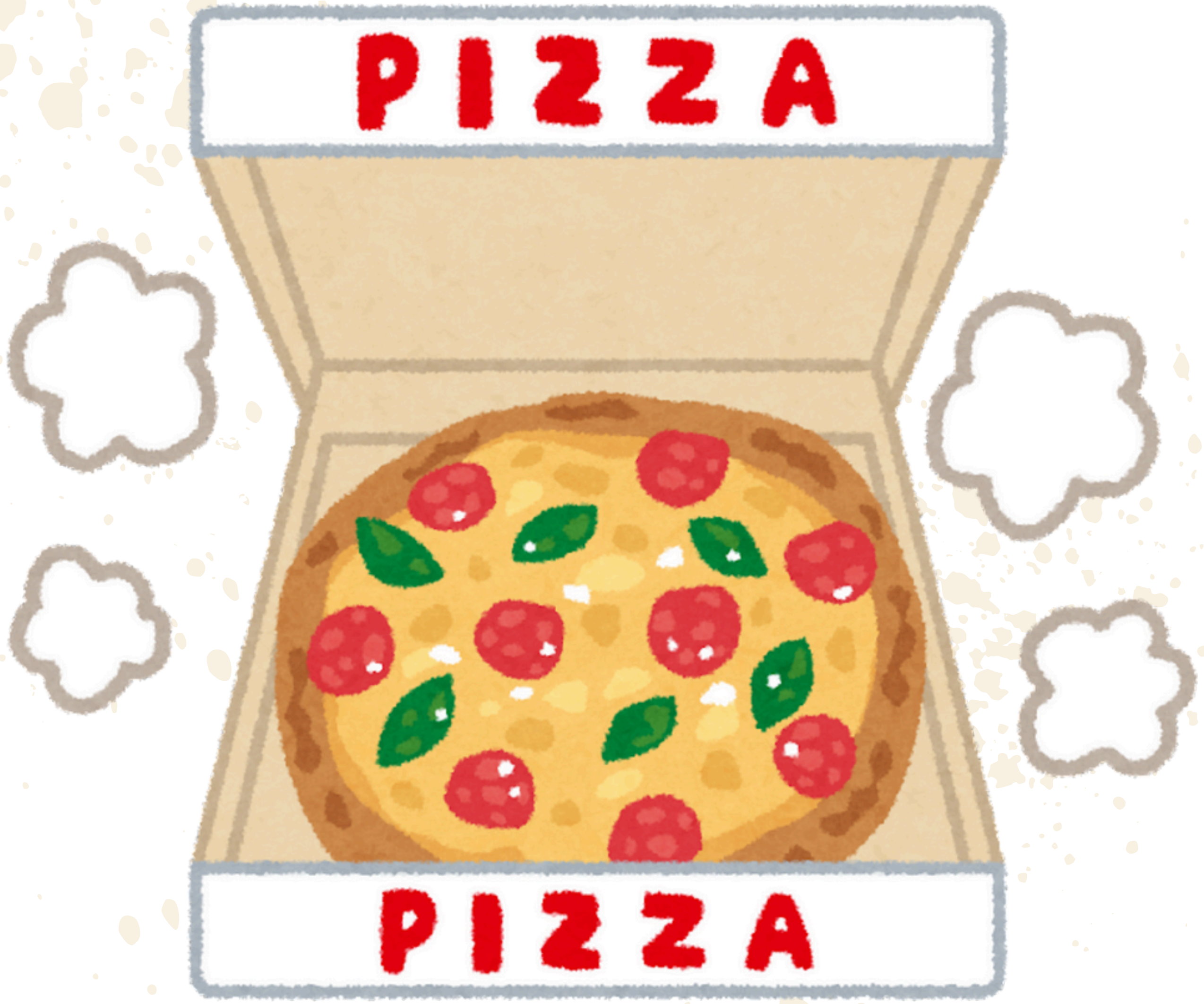
EIU

March 21, 2025

PIZZA RUNNER

8 Week SQL Challenge - Case Study 2



EIU

TEAM MEMBERS

Trần Phương Thảo - 2132300447

Thái Gia Huy - 2132300332

Trần Tiến Thảo Hiếu Ngân - 2032300513

TABLE OF CONTENT

A. Pizza Metrics

B. Runner and Customer Experience

C. Ingredient Optimisation

EIU

A. PIZZA METRICS

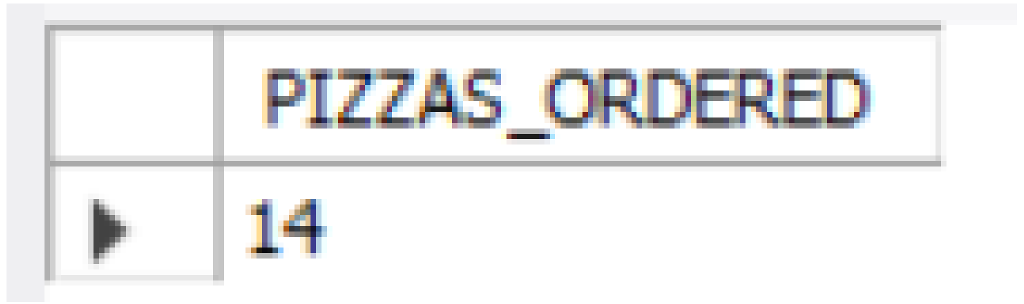
A. PIZZA METRICS

1. How many pizzas were ordered?

Objective: We need to find the total number of pizza orders recorded in the CUSTOMER_ORDERS table.

Count: All rows in the CUSTOMER_ORDERS table
Select: The total count is selected and displayed as PIZZAS_ORDERED.

Result:



	PIZZAS_ORDERED
▶	14

INSIGHT:

Pizza Runner received a total of 14 pizza orders.

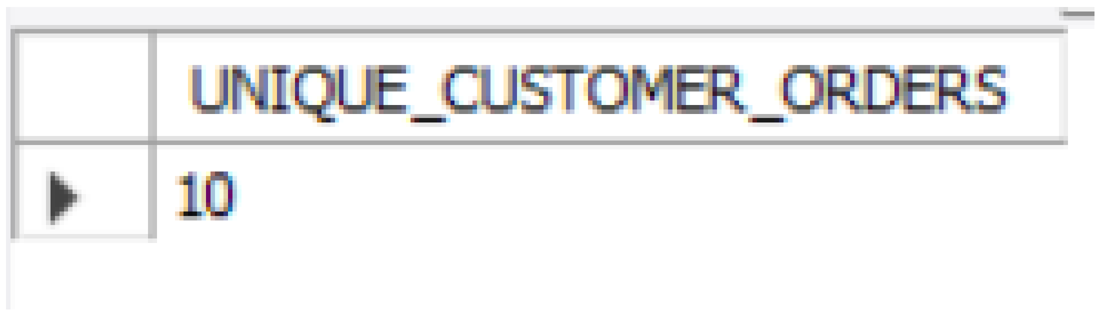
A. PIZZA METRICS

2. How many unique customer orders were made?

Objective: We need to find the number of unique order IDs in the CUSTOMER_ORDERS table

Distinct: Unique values of ORDER_ID are identified in the CUSTOMER_ORDERS table.
Count: The number of unique ORDER_ID values

Result:



	UNIQUE_CUSTOMER_ORDERS
▶	10

INSIGHT:

Pizza Runner received 10 unique customer orders. This means that while there were 14 pizzas ordered in total, some customers placed more than one order.

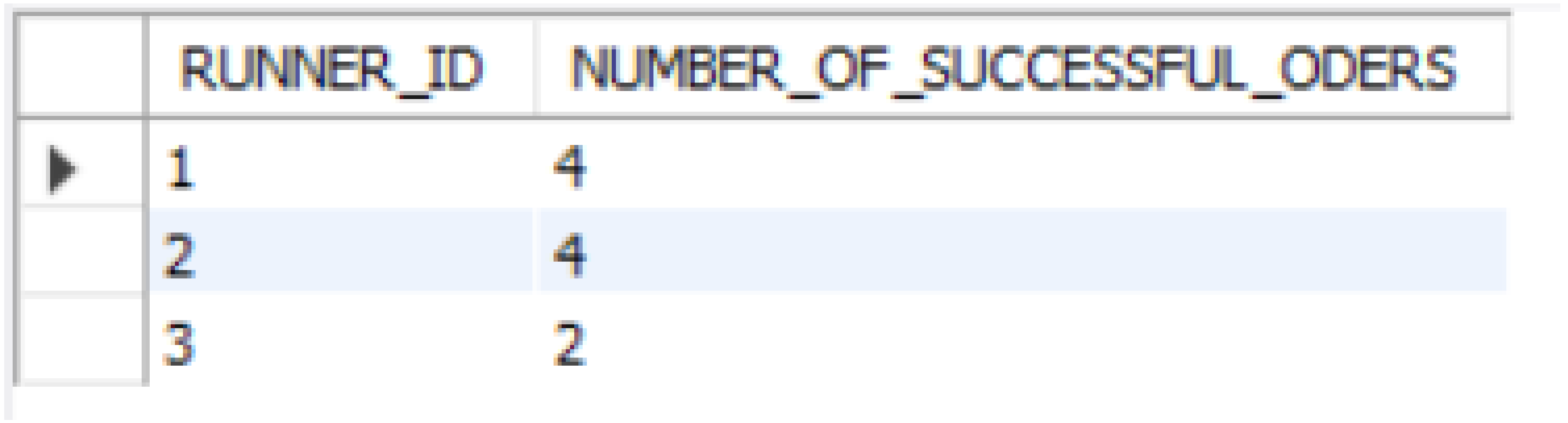
A. PIZZA METRICS

3. How many successful orders were delivered by each runner?

Objective: We'll use the DURATION as an indicator of a successful delivery.

Group the rows in the PIZZA_RUNNER.RUNNER_ORDERS table by RUNNER_ID.
Count the number of non-NULL DURATION values for each RUNNER_ID.
Order the results by RUNNER_ID in ascending order.

Result:



	RUNNER_ID	NUMBER_OF_SUCCESSFUL_ODERS
▶	1	4
	2	4
	3	2

INSIGHT:

Runners 1 and 2 each had 4 successful deliveries.

Runner 3 had 2 successful deliveries.

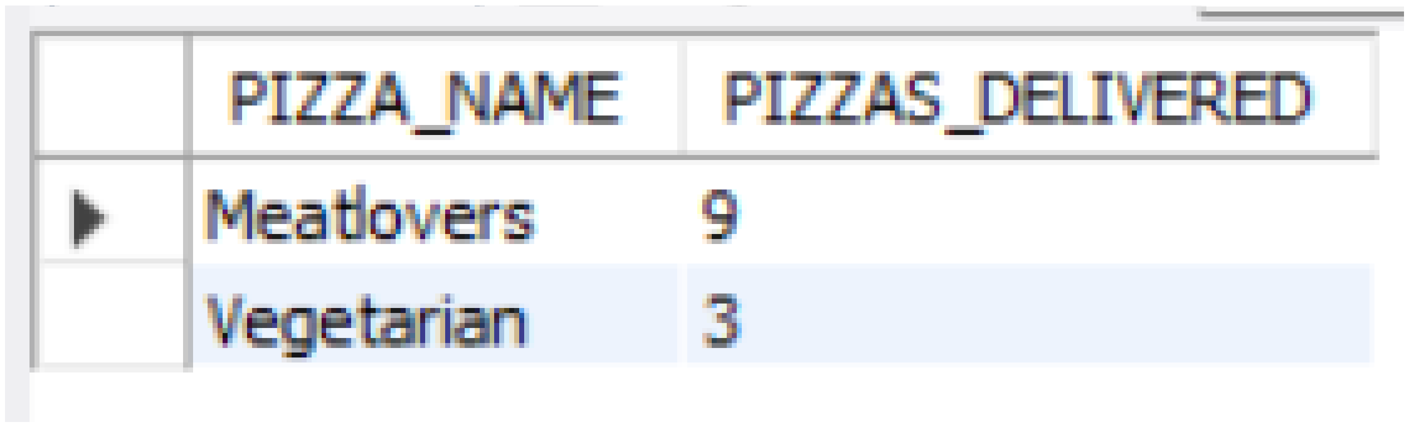
A. PIZZA METRICS

4. How many of each type of pizza was delivered?

Objective: We'll need to join multiple tables to get the pizza names and filter for successful deliveries.

Join RUNNER_ORDERS, CUSTOMER_ORDERS, and PIZZA_NAMES tables using ORDER_ID and PIZZA_ID.
Filter for successful deliveries by checking PICKUP_TIME is not 'null'.
Group: Group the results by PIZZA_NAME.
Count the number of PIZZA_ID values for each pizza name.

Result:

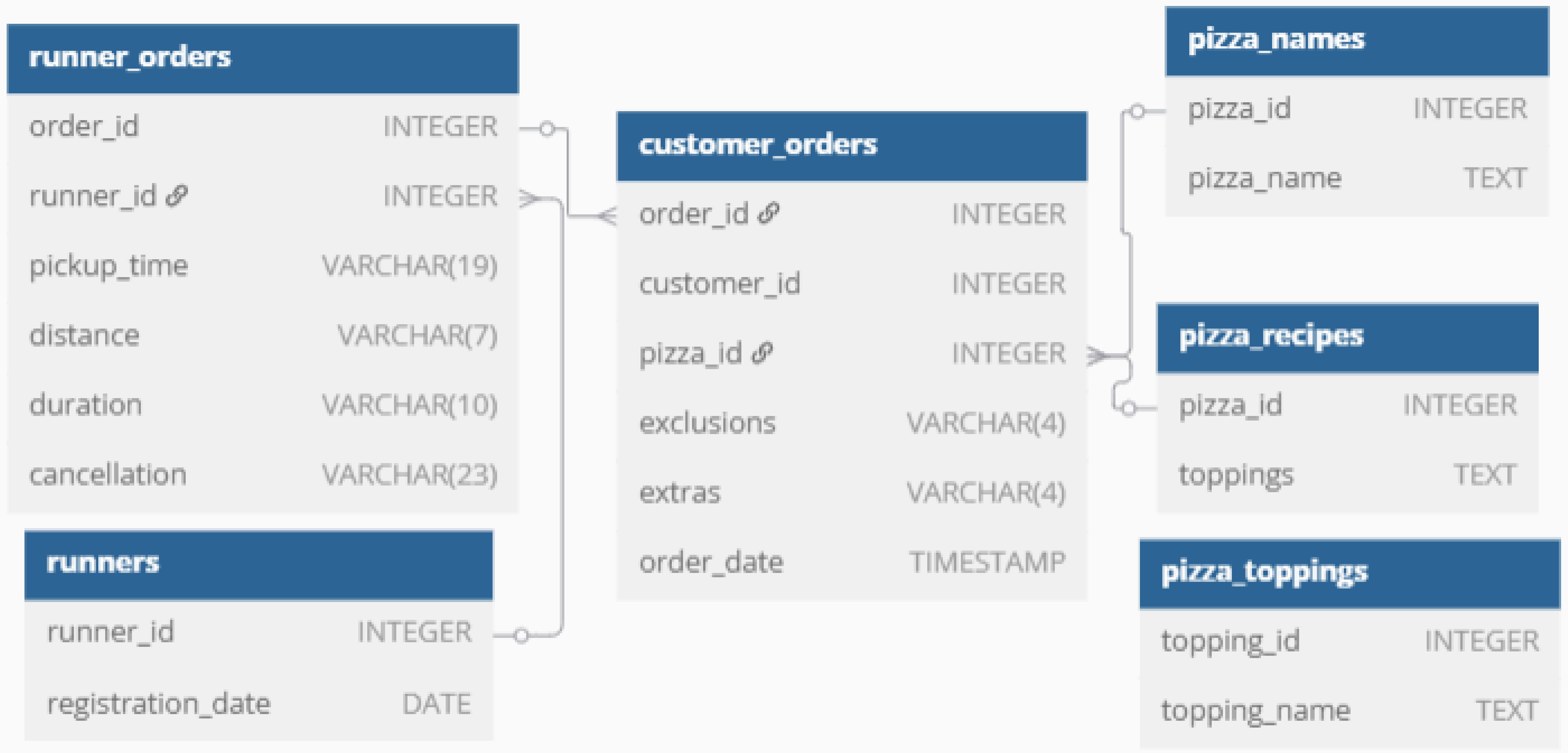


PIZZA_NAME	PIZZAS_DELIVERED
Meatlovers	9
Vegetarian	3

INSIGHT:

9 Meatlovers pizzas were delivered.
3 Vegetarian pizzas were delivered.

=> This indicates that Meatlovers are significantly more popular than Vegetarian among the delivered orders.


runner_orders
order_id INTEGER
runner_id INTEGER
pickup_time VARCHAR(19)
distance VARCHAR(7)
duration VARCHAR(10)
cancellation VARCHAR(23)
runners
runner_id INTEGER
registration_date DATE
customer_orders
order_id INTEGER
customer_id INTEGER
pizza_id INTEGER
exclusions VARCHAR(4)
extras VARCHAR(4)
order_date TIMESTAMP
pizza_names
pizza_id INTEGER
pizza_name TEXT
pizza_recipes
pizza_id INTEGER
toppings TEXT
pizza_toppings
topping_id INTEGER
topping_name TEXT

EIU

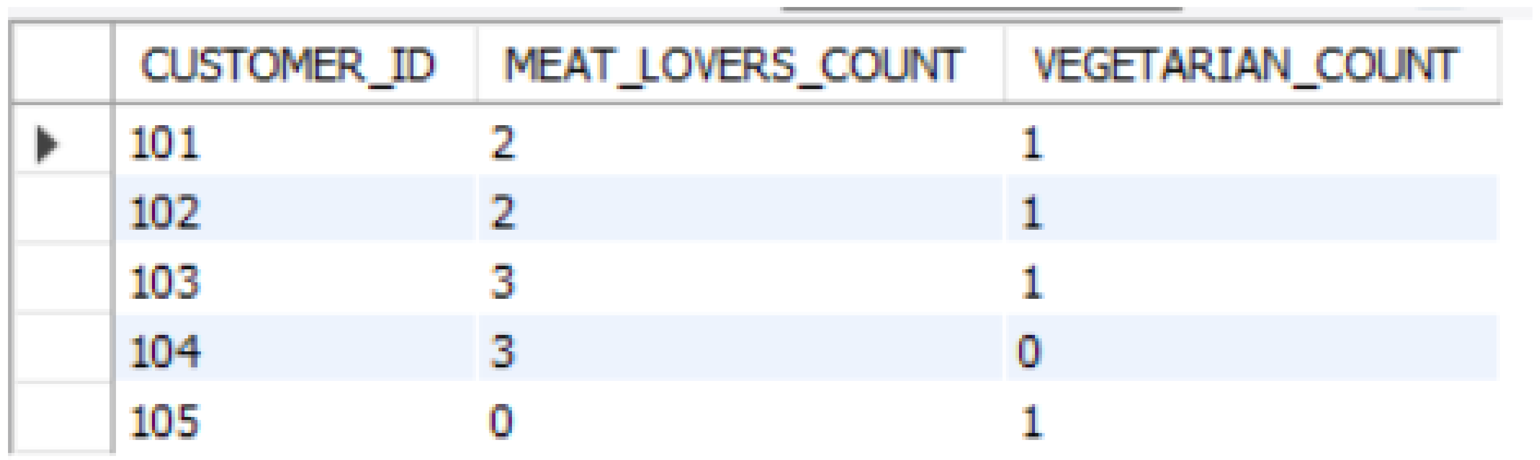
A. PIZZA METRICS

5. How many Vegetarian and Meatlovers were ordered by each customer?

Objective: We'll need to join the CUSTOMER_ORDERS and PIZZA_NAMES tables to get the pizza names.

Join CUSTOMER_ORDERS and PIZZA_NAMES tables using PIZZA_ID.
Conditional Sum: Use SUM(CASE WHEN ...) to count Meatlovers and Vegetarian pizzas for each row.
Group by CUSTOMER_ID.

Result:



CUSTOMER_ID	MEAT_LOVERS_COUNT	VEGETARIAN_COUNT
101	2	1
102	2	1
103	3	1
104	3	0
105	0	1

INSIGHT:

- Customers 101 and 102 each ordered 2 Meatlovers and 1 Vegetarian pizza.
- Customer 103 ordered 3 Meatlovers and 1 Vegetarian pizza.
- Customer 104 ordered 3 Meatlovers1 and 0 Vegetarian pizzas.

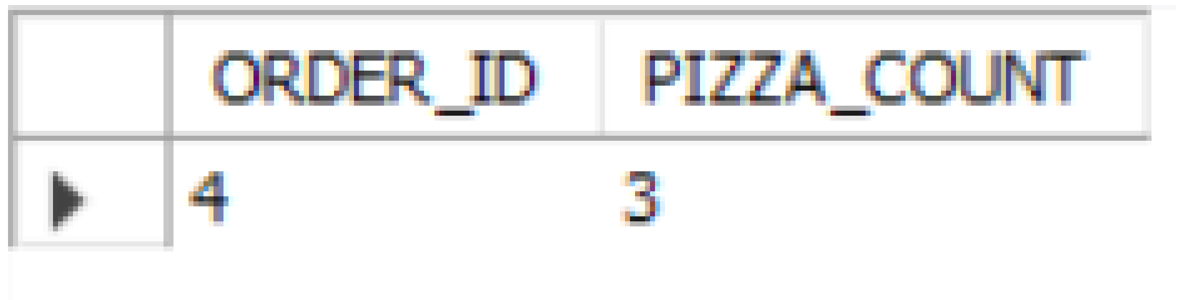
A. PIZZA METRICS

6. What was the maximum number of pizzas delivered in a single order?

Objective: We'll need to count the pizzas for each order and then find the maximum count.

Filter CUSTOMER_ORDERS table to exclude rows where PIZZA_ID is NULL.
Group the remaining rows by ORDER_ID.
Count the PIZZA_ID values for each ORDER_ID group.
Order the results in descending order based on the PIZZA_COUNT.
Limit the result to the first row (the row with the highest PIZZA_COUNT).

Result:



	ORDER_ID	PIZZA_COUNT
▶	4	3

INSIGHT:

Order ID 4 had the maximum number of pizzas ordered in a single order, with a total of 3 pizzas.

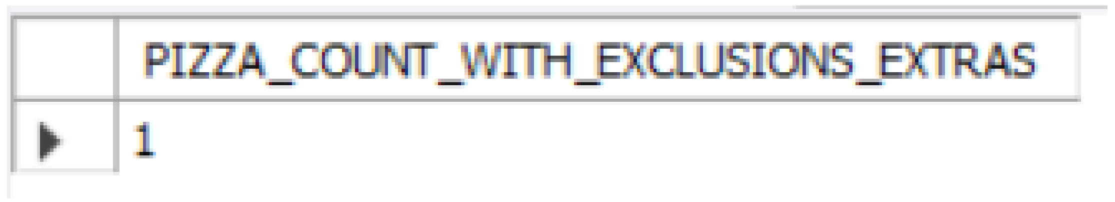
A. PIZZA METRICS

8. How many pizzas were delivered that had both exclusions and extras?

Objective: We'll need to join CUSTOMER_ORDERS and RUNNER_ORDERS, filter for delivered orders, and then filter further for rows where both exclusions and extras are not null and not empty.

Join CUSTOMER_ORDERS and RUNNER_ORDERS on order_id.
Filter (Delivery) by checking distance is greater than or equal to 1.
Filter for rows where exclusions are not null and not an empty string.
Filter for rows where extras are not null and not an empty string.
Conditional Sum: For each row, add 1 to the sum if both exclusions and extras pass the filters, otherwise add 0.

Result:



	PIZZA_COUNT_WITH_EXCLUSIONS_EXTRAS
▶	1

INSIGHT:

Only 1 pizza was delivered with both exclusions and extras.

EIU

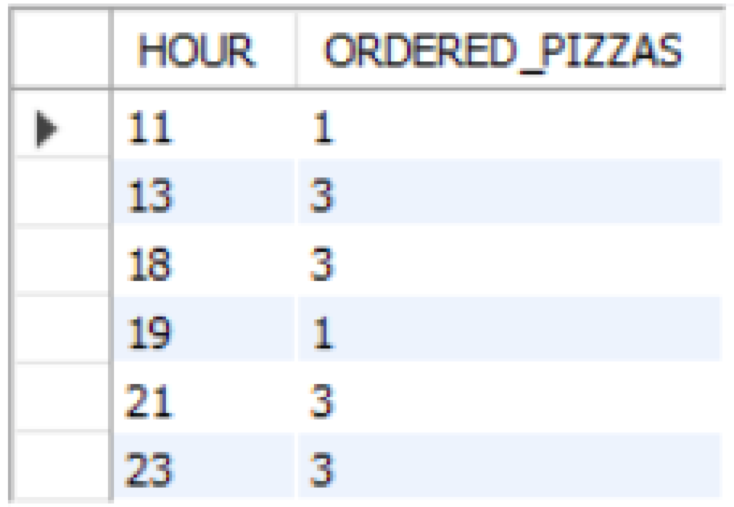
A. PIZZA METRICS

9. What was the total volume of pizzas ordered for each hour of the day?

Objective: We will need to extract the hour from the ORDER_TIME column and then count the number of orders for each hour.

Extract the hour from the ORDER_TIME column using DATE_PART('HOUR', ORDER_TIME).
Group: Group the rows by the extracted hour.
Count: Count the number of rows (pizzas) in each hour group.

Result:



HOUR	ORDERED_PIZZAS
11	1
13	3
18	3
19	1
21	3
23	3

INSIGHT:

- The highest order volumes occurred at hours 13, 18, 21, and 23, with 3 pizzas ordered each.
- The lowest order volumes occurred at hours 11 and 19, with 1 pizza ordered each.

EIU

A. PIZZA METRICS

9. What was the total volume of pizzas ordered for each hour of the day?

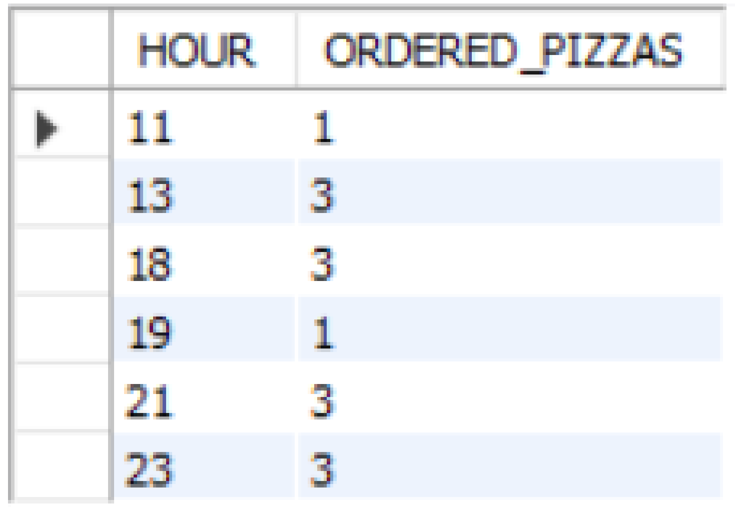
Objective: We will need to extract the hour from the ORDER_TIME column and then count the number of orders for each hour.

Extract the hour from the ORDER_TIME column using DATE_PART('HOUR', ORDER_TIME).

Group: Group the rows by the extracted hour.

Count: Count the number of rows (pizzas) in each hour group.

Result:



HOUR	ORDERED_PIZZAS
11	1
13	3
18	3
19	1
21	3
23	3

INSIGHT:

- The highest order volumes occurred at hours 13, 18, 21, and 23, with 3 pizzas ordered each.
- The lowest order volumes occurred at hours 11 and 19, with 1 pizza ordered each.

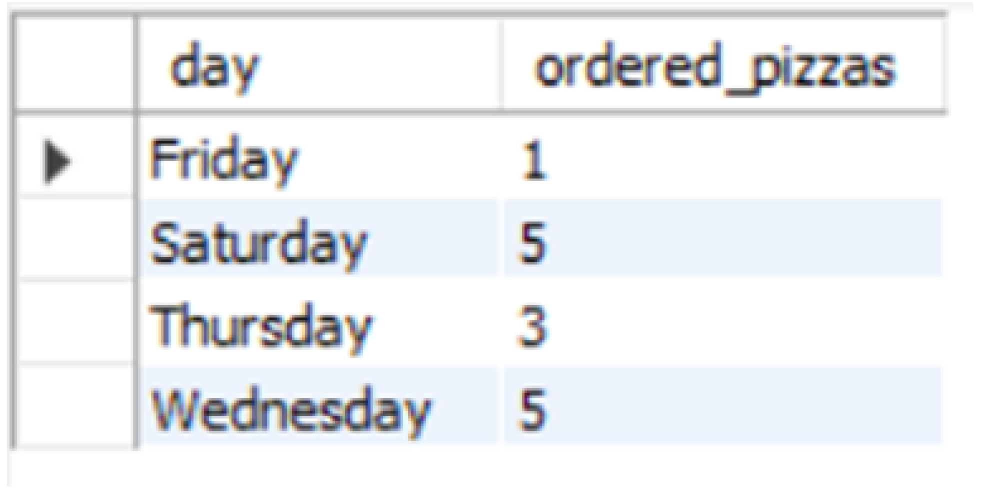
A. PIZZA METRICS

10. What was the volume of orders for each day of the week?

Objective: We will need to extract the day of the week from the ORDER_TIME column and then count the number of orders for each day.

Extract the day of the week from the ORDER_TIME column using DAYNAME(ORDER_TIME).
Group: Group the rows by the extracted day.
Count: Count the number of rows (pizzas) in each day group.

Result:



day	ordered_pizzas
Friday	1
Saturday	5
Thursday	3
Wednesday	5

INSIGHT:

- Saturday and Wednesday had the highest order volumes, with 5 pizzas ordered each.
- Thursday had a moderate order volume of 3 pizzas.
- Friday had the lowest order volume, with only 1 pizza ordered.

B. RUNNER AND CUSTOMER EXPERIENCE

EIU

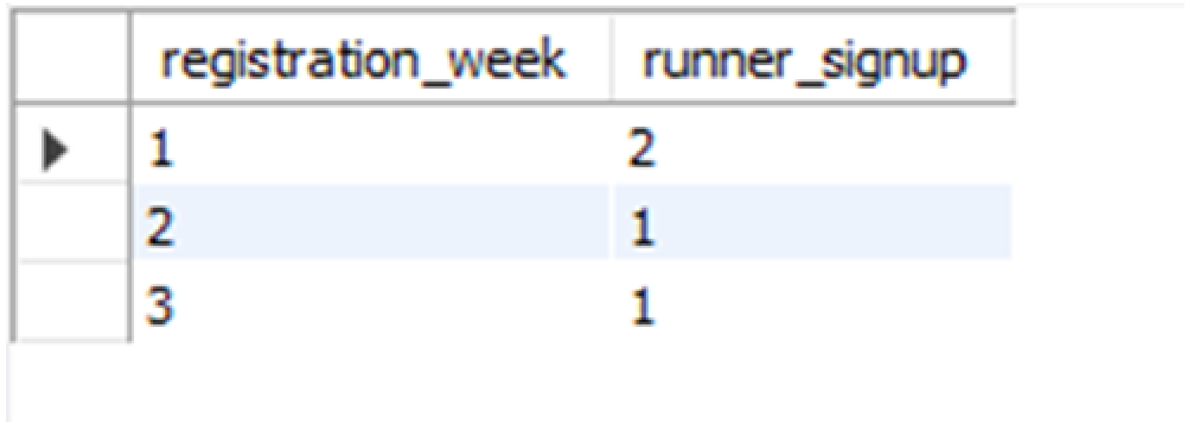
B. RUNNER AND CUSTOMER EXPERIENCE

1. How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)

Objective: Use the WEEK() function to extract the week number from the registration date

Filter the runners table to include only rows where registration_date is on or after '2021-01-01'. Extract week number using WEEK().
Group: Group the results by the extracted registration_week
Count: Count the number of runner_id within each week group.

Result:



registration_week	runner_signup
1	2
2	1
3	1

INSIGHT:

- Week 01: Highest sign-ups (2 runners)
- Weeks 02 & 03: 1 runner each
- There appears to be a decline after in registrations after Week 01.

EIU

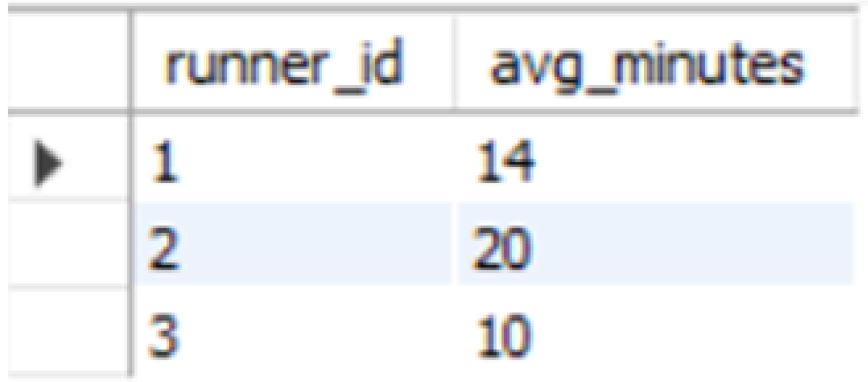
B. RUNNER AND CUSTOMER EXPERIENCE

2. What was the average time in minutes it took for each runner to arrive at the Pizza Runner HQ to pick up the order?

Objective: We want to calculate the average pickup time in minutes for each runner, measuring the time from when a customer places an order.

This will be done using functions like TO_TIMESTAMP(), EXTRACT(), AVG(), ROUND(), and CASE WHEN THEN ELSE END, LEFT JOIN () runner_orders and customer_orders
Group the final result by runner_id to get the average pickup time for each runner.

Result:



runner_id	avg_minutes
1	14
2	20
3	10

INSIGHT:

- Runner 2: Longest pickup time (20 min), indicating delays or inefficiencies.
- Runner 3: Shortest pickup time (10 min), suggesting efficiency.
- Runner 1: Moderate pickup time (14 min), showing consistency.

EIU

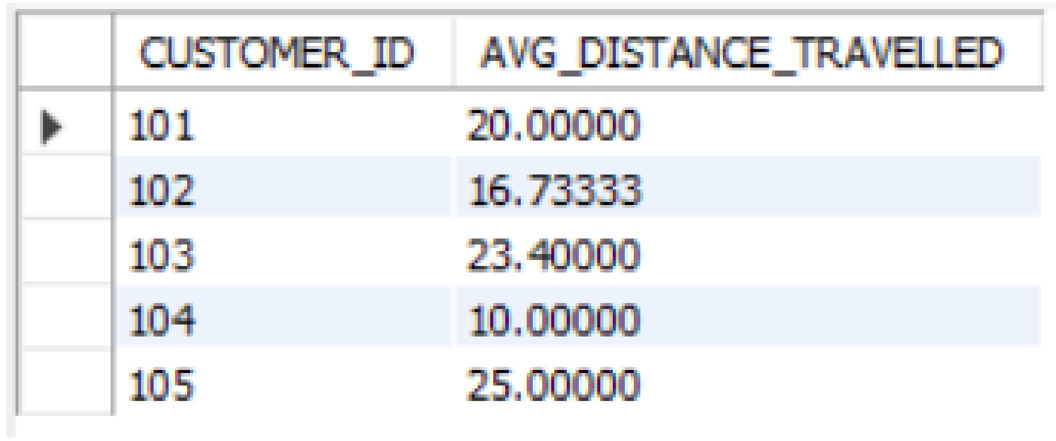
B. RUNNER AND CUSTOMER EXPERIENCE

3. What was the average distance travelled for each customer?

Objective: We need to find the average distance travelled by each customer, using the runner_orders and customer_orders tables, and the AVG(), REPLACE(), and type casting functions.

Join the runner_orders and customer_orders tables on order_id.
Remove the 'KM' from the DISTANCE column and convert the remaining value to a numeric type.
Filter out rows where the distance is 'NULL'.
Calculate the average distance for each customer.
Group the results by customer_id.

Result:



CUSTOMER_ID	AVG_DISTANCE_TRAVELLED
101	20.00000
102	16.73333
103	23.40000
104	10.00000
105	25.00000

INSIGHT:

Customer 105 traveled an average distance of 25.0 kilometers.
Customer 103 traveled an average distance of 23.4 kilometers.
Customer 101 traveled an average distance of 20.0 kilometers.
Customer 102 traveled an average distance of 16.73 kilometers.
Customer 104 traveled an average distance of 10.0 kilometers.

B. RUNNER AND CUSTOMER EXPERIENCE

4. What was the difference between the longest and shortest delivery times for all orders?

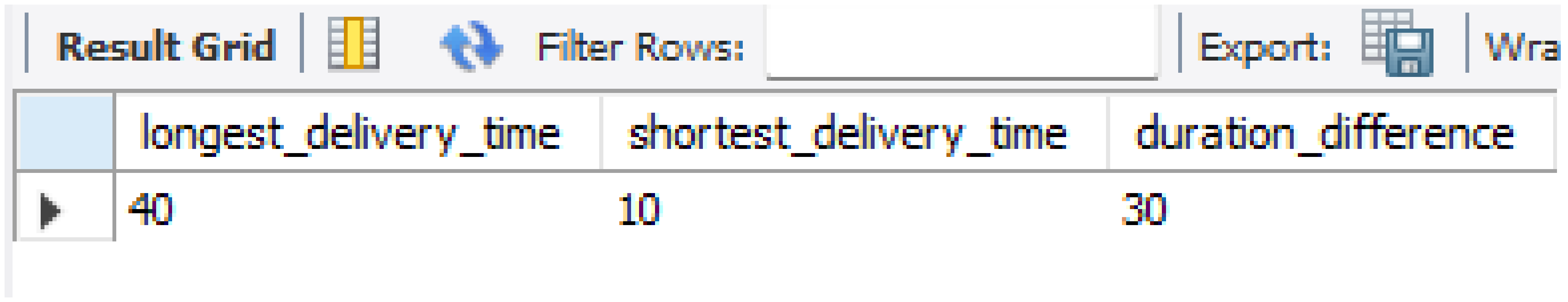
Objective: We will calculate the longest, shortest, and difference in delivery durations for all completed orders, using the runner_orders table, MAX(duration) and MIN(duration)

Use MAX() to find the longest delivery time.

Use MIN() to find the shortest delivery time.

Calculate the difference between the longest and shortest delivery times.

Result:



longest_delivery_time	shortest_delivery_time	duration_difference
40	10	30

INSIGHTS

The **highest** delivery time is 40 minutes

The **lowest** delivery time is 10 minutes

The **difference** between the longest and shortest delivery times is 30 minutes

EIU

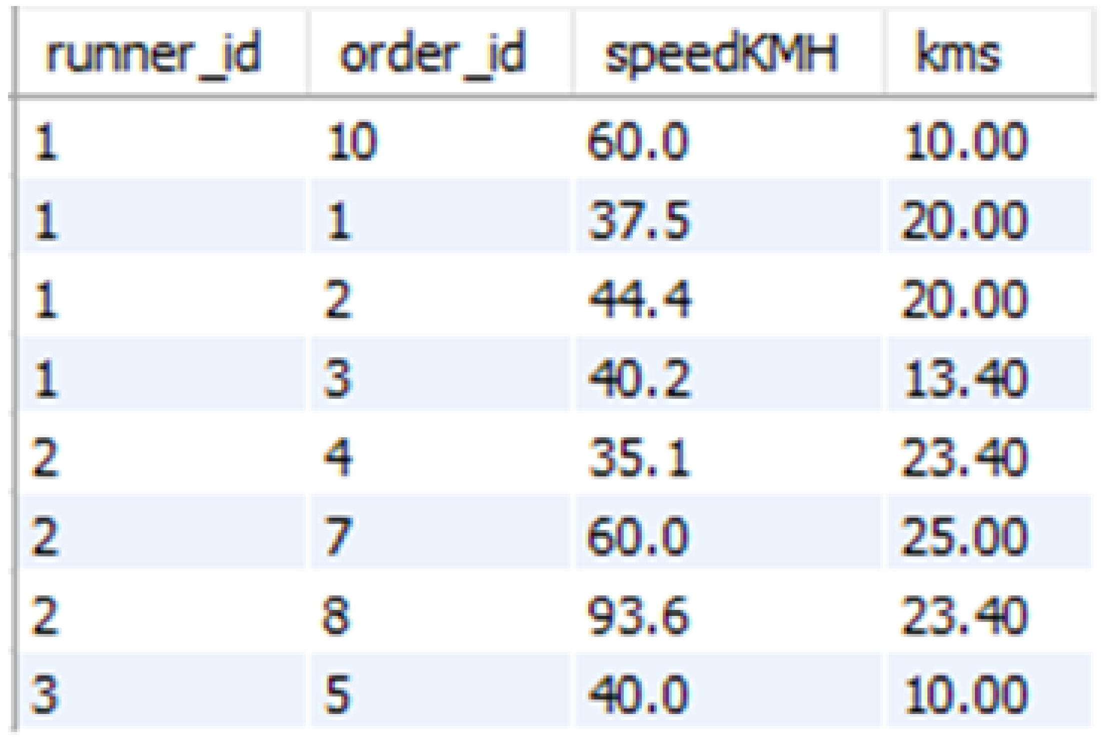
B. RUNNER AND CUSTOMER EXPERIENCE

5. What was the average speed for each runner for each delivery and do you notice any trend for these values?

Objective: We want to calculate the average speed for each runner for each delivery, using the runner_orders table using ROUND(), SUM().

Filters out invalid/canceled orders by excluding rows where distance or duration is NULL.
Selects runner_id, order_id, distance, and duration from runner_orders.

Result:



runner_id	order_id	speedKMH	kms
1	10	60.0	10.00
1	1	37.5	20.00
1	2	44.4	20.00
1	3	40.2	13.40
2	4	35.1	23.40
2	7	60.0	25.00
2	8	93.6	23.40
3	5	40.0	10.00

INSIGHT:

Runner 1: Shows consistent delivery speeds, indicating steady performance and predictable delivery conditions.
Runner 3: Only one delivery recorded, making trend analysis inconclusive; more data is needed.

EIU

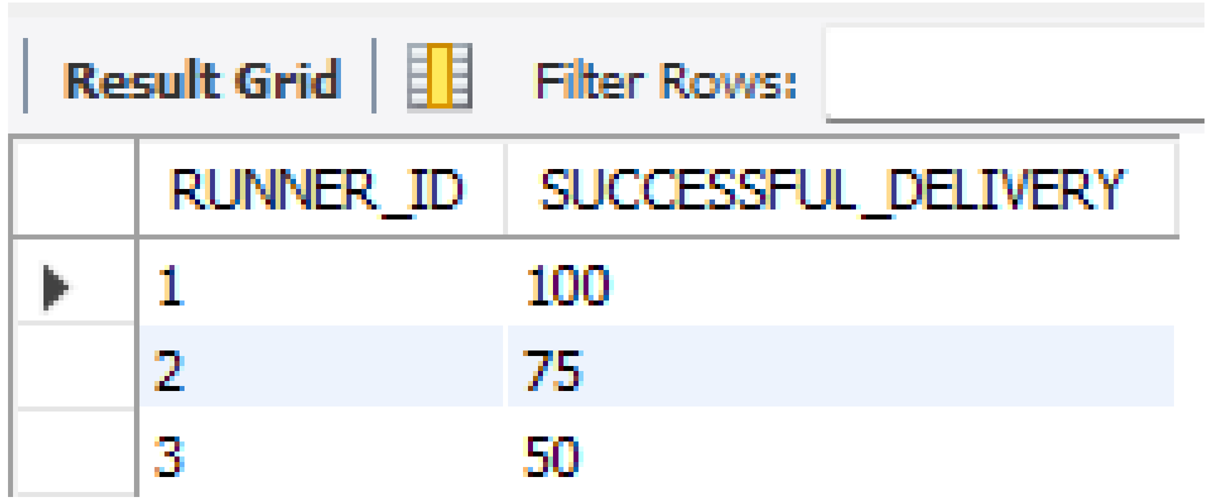
B. RUNNER AND CUSTOMER EXPERIENCE

6. What is the successful delivery percentage for each runner?

We will use data from the runner_orders table. We will use SQL functions such as COUNT(), SUM(), ROUND(), and CASE WHEN THEN ELSE END to achieve this.

CTE Creation (ORDER_STATS): Calculate total orders and delivered orders for each runner, conditional sum: Use SUM(CASE WHEN DISTANCE != 0 THEN 1 ELSE 0 END) to count successfully delivered orders (i.e., deliveries where distance is not zero).
Group By: Group results by RUNNER_ID.

Result:



RUNNER_ID	SUCCESSFUL_DELIVERY
1	100
2	75
3	50

INSIGHTS

Runner 1 has a perfect successful delivery rate at **100%.**

Runners 2 achieved **75%** successful deliveries, placing it in the middle.

Runner 3 has the fewest successful deliveries at **50%.**

C. INGREDIENT OPTIMISATION

EIU

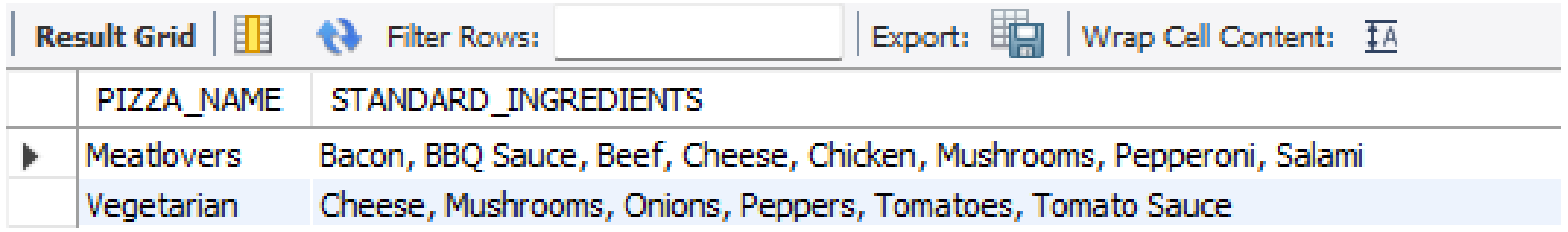
C. INGREDIENT OPTIMISATION

1. What are the standard ingredients for each pizza?

Objective: Find the standard ingredients for each pizza in order to see what ingredients are they made of.

INNER JOIN to connect PIZZA_TOPPINGS and PIZZA_RECIPES and PIZZA_NAMES
GROUP_CONCAT: Takes multiple rows of TOPPING_NAME column in PIZZA_TOPPINGS table and concatenates them into a single string.

Result:



PIZZA_NAME	STANDARD_INGREDIENTS
Meatlovers	Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
Vegetarian	Cheese, Mushrooms, Onions, Peppers, Tomatoes, Tomato Sauce

INSIGHTS

Both food items have unique ingredients as meatlovers are meant for meat-enjoyers and vegetarian is for plant-eater

Cheese and Mushrooms are included in both food items

C. INGREDIENT OPTIMISATION

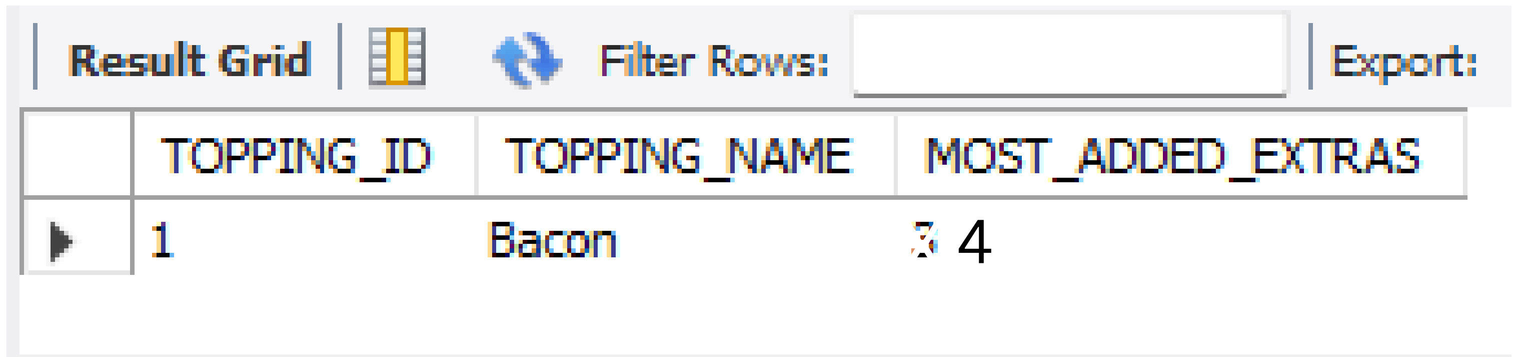
2. What was the most commonly added extra and exclusion?

Objective: Find the extra topping that is most added in the pizza to see which topping is ordered more than the others.

INNER JOIN to connect PIZZA_TOPPINGS, PIZZA_RECIPES, CUSTOMER_ORDERS

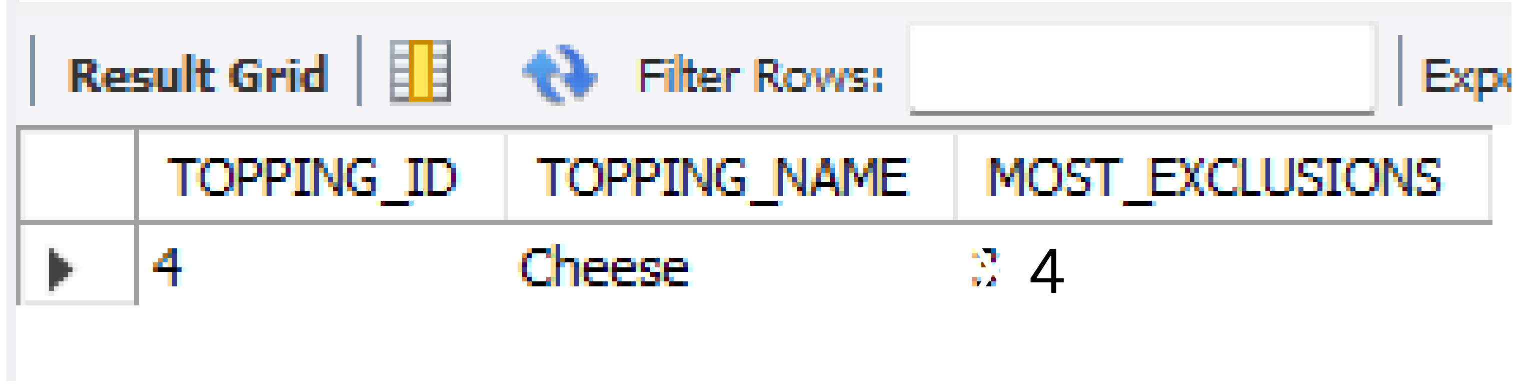
COUNT the distincts of extras and exclusions to see the total number of each

Result:



Result Grid | Filter Rows: | Export:

TOPPING_ID	TOPPING_NAME	MOST_ADDED_EXTRAS
1	Bacon	4



Result Grid | Filter Rows: | Exp

TOPPING_ID	TOPPING_NAME	MOST_EXCLUSIONS
4	Cheese	4

INSIGHTS

Bacon is the **most commonly added extra topping**, with 4 occurrences, indicating its popularity among customers.

Cheese is the **most commonly excluded topping**, also with 4 exclusions, suggesting some customers prefer pizzas without cheese.

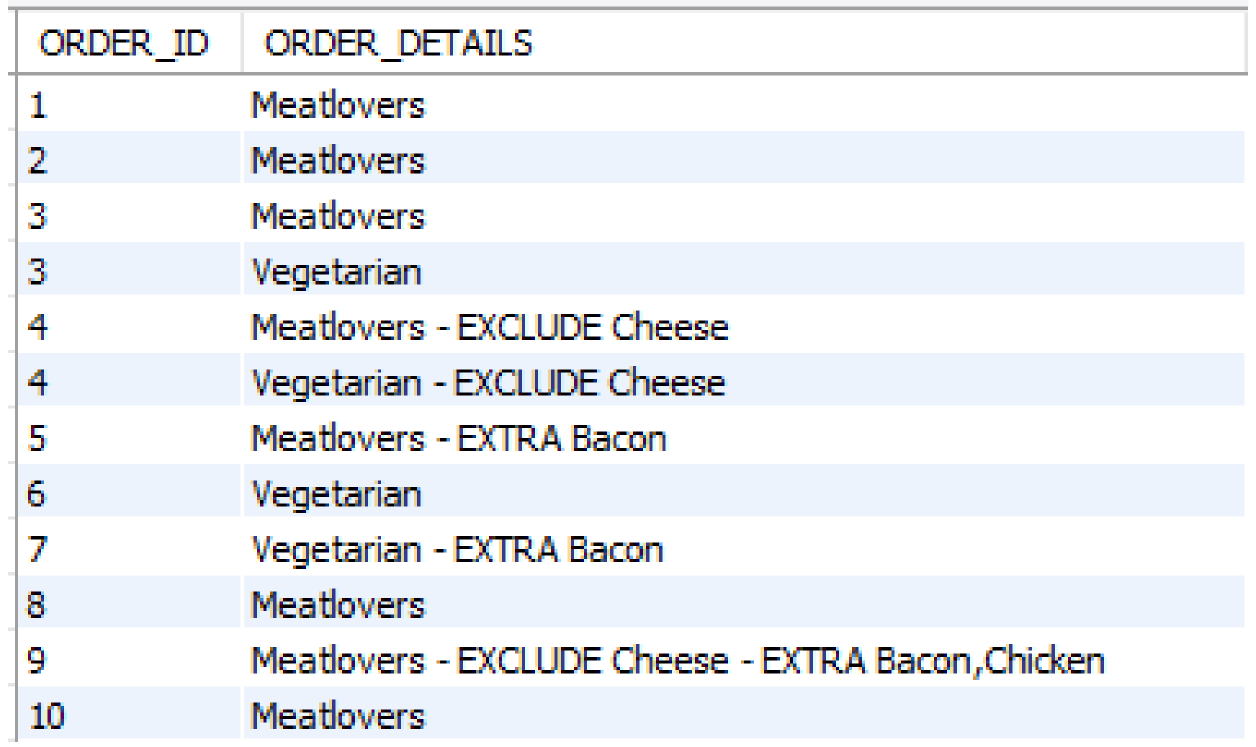
C. INGREDIENT OPTIMISATION

3. Generate an order item for each record in the customers_orders table

Objective: Make a list showing order id along with its order's details. The details include which type of pizza was ordered following with the extras and exclusions made by customers

INNER JOIN to connect CUSTOMER_ORDERS, PIZZA_NAMES, PIZZA_TOPPINGS
CONCAT Concatenates excluded toppings (EXCLUSIONS) and extra toppings (EXTRAS) for each order using GROUP_CONCAT, ensuring distinct values and sorting them alphabetically

Result:



ORDER_ID	ORDER_DETAILS
1	Meatlovers
2	Meatlovers
3	Meatlovers
3	Vegetarian
4	Meatlovers - EXCLUDE Cheese
4	Vegetarian - EXCLUDE Cheese
5	Meatlovers - EXTRA Bacon
6	Vegetarian
7	Vegetarian - EXTRA Bacon
8	Meatlovers
9	Meatlovers - EXCLUDE Cheese - EXTRA Bacon,Chicken
10	Meatlovers

INSIGHTS

About **half** of the total orders involve exclusions or extra toppings.

Cheese is the **only topping excluded** in both the Meatlovers and Vegetarian pizzas.

Bacon was added even to Vegetarian pizzas (Order ID 8), showing its popularity among customers as an extra topping.

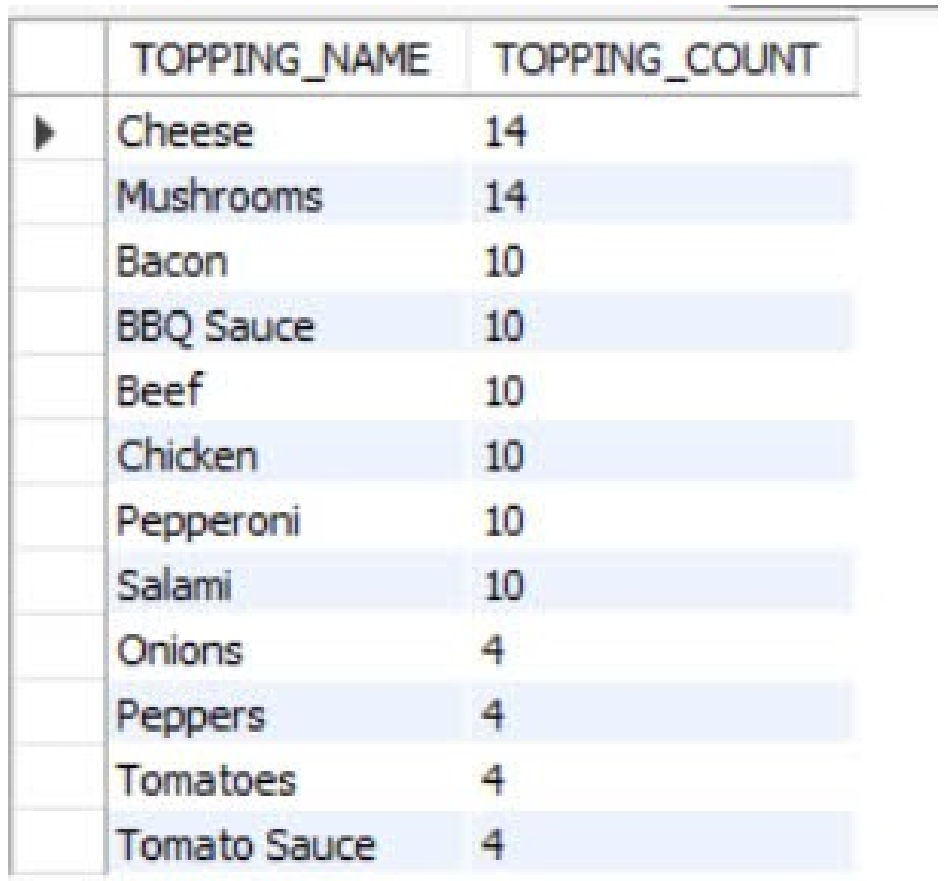
C. INGREDIENT OPTIMISATION

4. What is the total quantity of each ingredient used in all delivered pizzas sorted by most frequent first?

Objective: Calculate the total usage of each ingredient in all delivered pizzas sorting by the most frequently used ingredient first

Join 4 tables which are PIZZA_TOPPINGS, PIZZA_RECIPES, CUSTOMER_ORDERS, and RUNNER_ORDERS to trace toppings to specific orders.

Result:



TOPPING_NAME	TOPPING_COUNT
Cheese	14
Mushrooms	14
Bacon	10
BBQ Sauce	10
Beef	10
Chicken	10
Pepperoni	10
Salami	10
Onions	4
Peppers	4
Tomatoes	4
Tomato Sauce	4

INSIGHTS

Cheese & mushrooms are the most used toppings (14 times), followed by popular meats (10 times each), while vegetables and sauces are less frequent (4 times)

THANK YOU