🍕 Case Study #2: Pizza Runner

📜 Table of Contents

📄 Introduction

Danny Ma, inspired by the idea of 80s retro styling combined with the convenience of Uber-style deliveries, launched **Pizza Runner**. Starting from his home, Danny began a pizza delivery service where runners deliver pizzas from **Pizza Runner HQ** directly to customers. This case study focuses on assisting Danny with data analysis to streamline operations, manage runners effectively, and improve the customer experience.

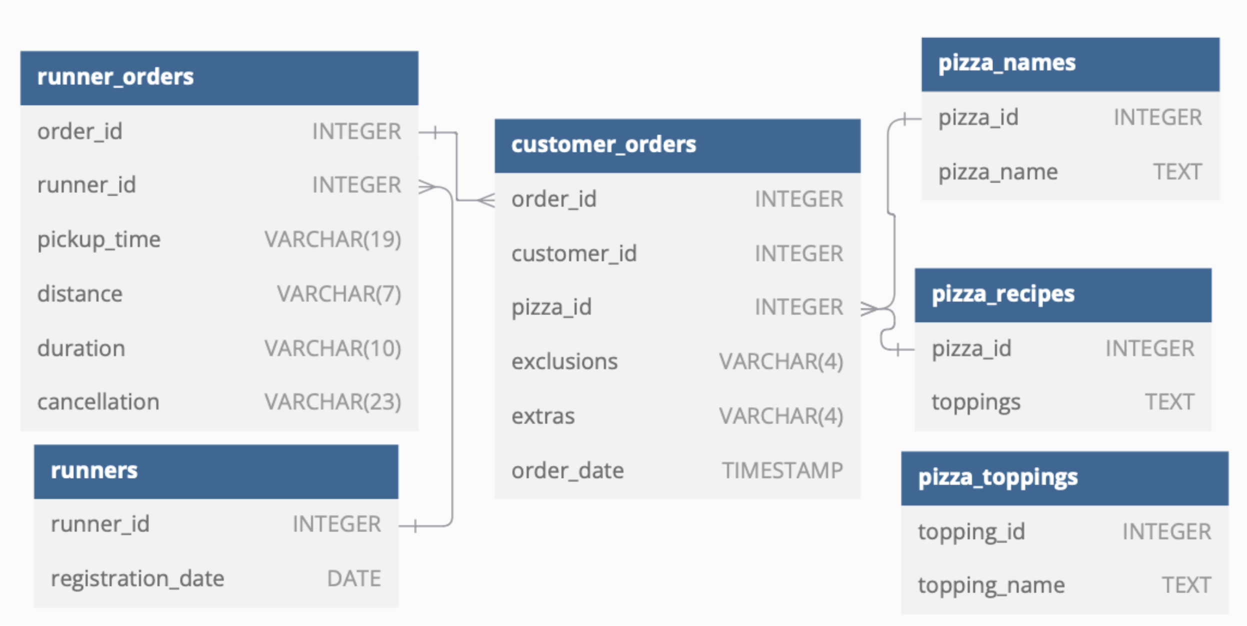
Note: All the information regarding the case study has been sourced from the following link: here. For executing the queries, I used PostgreSQL on DB Fiddle.

📃 Available Data & Entity Relationship Diagram

Danny collected various datasets to track every aspect of his business:

- **Runners**: Tracks each runner's registration date.
- **Customer Orders**: Details customer orders, including pizzas ordered, exclusions, and extras.
- **Runner Orders**: Assigns runners to customer orders, with data on pickup times, distances, durations, and cancellations.
- **Pizza Names**: Lists the two types of pizzas offered - Meat Lovers and Vegetarian.
- **Pizza Recipes**: Contains standard ingredients for each pizza type.
- **Pizza Toppings**: Maps topping IDs to topping names.

Entity Relationship Diagram: Shows relationships between the runners, customer orders, runner orders, pizza names, pizza recipes, and pizza toppings.



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

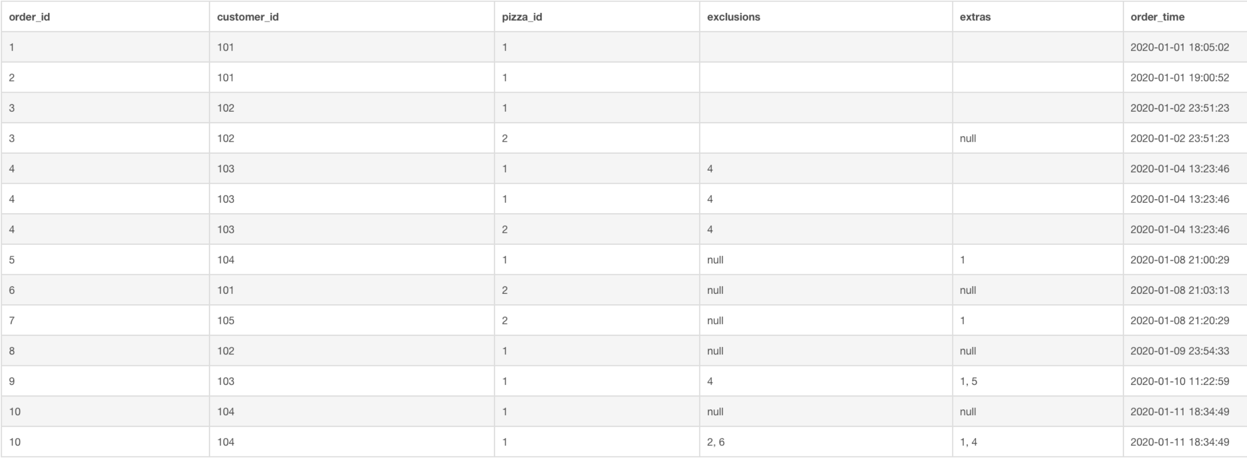
pizza_toppings	
topping_id	INTEGER
topping_name	TEXT

🧼 Data Cleaning & Transformation

🔨 Table: customer_orders

Observations:

- The `exclusions` and `extras` columns contain null values and blank spaces.



order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1			2020-01-01 18:05:02
2	101	1			2020-01-01 19:00:52
3	102	1			2020-01-02 23:51:23
3	102	2		null	2020-01-02 23:51:23
4	103	1	4		2020-01-04 13:23:46
4	103	1	4		2020-01-04 13:23:46
4	103	2	4		2020-01-04 13:23:46
5	104	1	null	1	2020-01-08 21:00:29
6	101	2	null	null	2020-01-08 21:03:13
7	105	2	null	1	2020-01-08 21:20:29
8	102	1	null	null	2020-01-09 23:54:33
9	103	1	4	1, 5	2020-01-10 11:22:59
10	104	1	null	null	2020-01-11 18:34:49
10	104	1	2, 6	1, 4	2020-01-11 18:34:49

Cleaning Strategy:

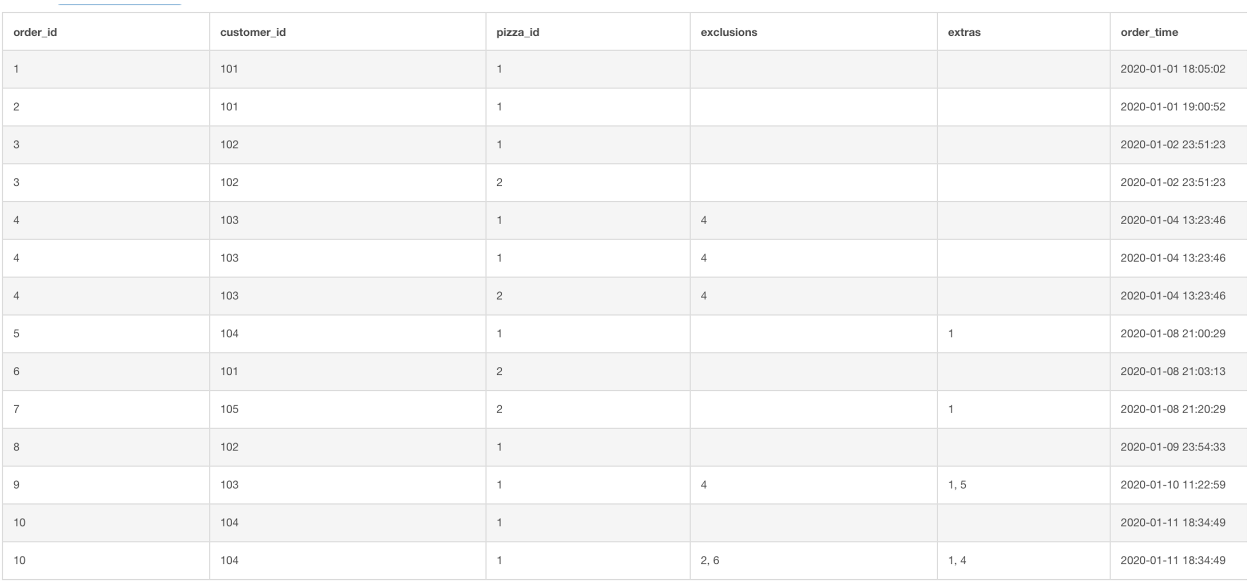
- Replace all null values or blank spaces in the `exclusions` and `extras` columns with a single blank space (`' '`).

Implementation:

A temporary table, `customer_orders_temp`, was created with the following adjustments:

- Null values or entries marked as `'null'` in the `exclusions` and `extras` columns were replaced with `' '`.

```
CREATE TEMP TABLE customer_orders_temp AS
SELECT
  order_id,
  customer_id,
  pizza_id,
  CASE
      WHEN exclusions IS NULL OR exclusions LIKE 'null' THEN '
      ELSE exclusions
  END AS exclusions,
  CASE
      WHEN extras IS NULL OR extras LIKE 'null' THEN ' '
      ELSE extras
  END AS extras,
  order_time
FROM pizza_runner.customer_orders;
```

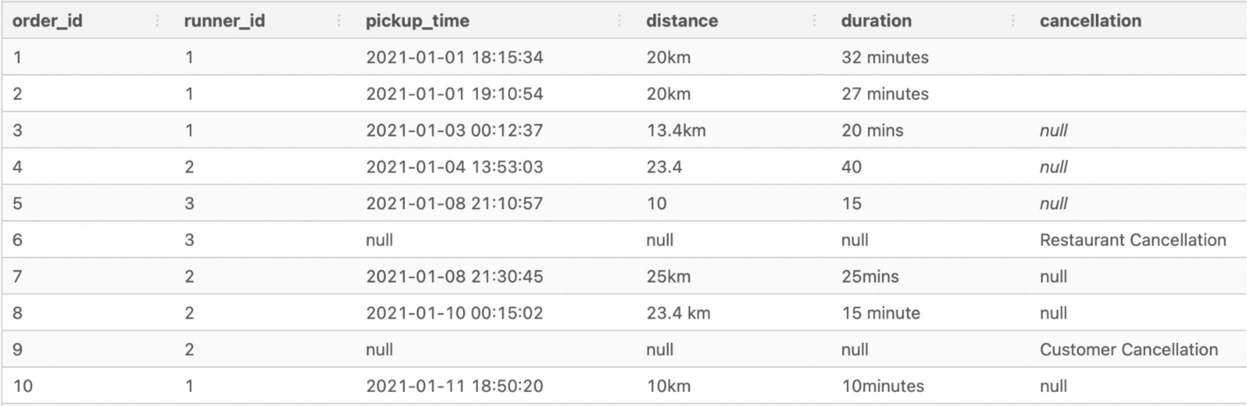


order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1			2020-01-01 18:05:02
2	101	1			2020-01-01 19:00:52
3	102	1			2020-01-02 23:51:23
3	102	2			2020-01-02 23:51:23
4	103	1	4		2020-01-04 13:23:46
4	103	1	4		2020-01-04 13:23:46
4	103	2	4		2020-01-04 13:23:46
5	104	1		1	2020-01-08 21:00:29
6	101	2			2020-01-08 21:03:13
7	105	2		1	2020-01-08 21:20:29
8	102	1			2020-01-09 23:54:33
9	103	1	4	1, 5	2020-01-10 11:22:59
10	104	1			2020-01-11 18:34:49
10	104	1	2, 6	1, 4	2020-01-11 18:34:49

🔨 Table: `runner_orders`

Observations:

- The following columns contain inconsistent data:
  - `pickup_time`: Null values and blank spaces.
  - `distance`: Values include the text "km" and null values.
  - `duration`: Values include "minutes," "minute," and nulls.
  - `cancellation`: Null values and blank spaces.



order_id	runner_id	pickup_time	distance	duration	cancellation
1	1	2021-01-01 18:15:34	20km	32 minutes	
2	1	2021-01-01 19:10:54	20km	27 minutes	
3	1	2021-01-03 00:12:37	13.4km	20 mins	*null*
4	2	2021-01-04 13:53:03	23.4	40	*null*
5	3	2021-01-08 21:10:57	10	15	*null*
6	3	null	null	null	Restaurant Cancellation
7	2	2021-01-08 21:30:45	25km	25mins	null
8	2	2021-01-10 00:15:02	23.4 km	15 minute	null
9	2	null	null	null	Customer Cancellation
10	1	2021-01-11 18:50:20	10km	10minutes	null

Cleaning Strategy:

1. Replace null values and blank spaces in relevant columns with a single blank space (`' '`).
2. Standardize formats by removing units (e.g., "km" from `distance`, "minutes" from `duration`).
3. Convert `pickup_time`, `distance`, and `duration` columns to appropriate data types.

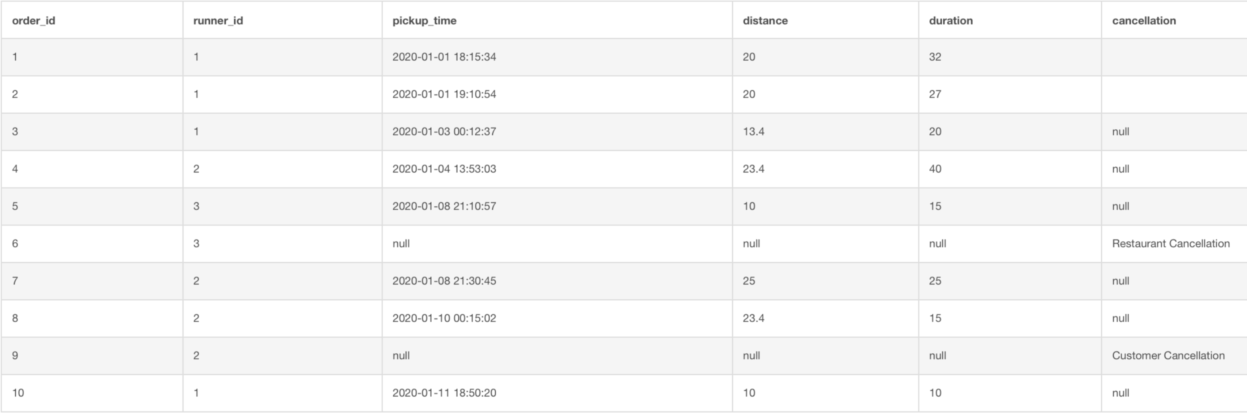
Implementation:

A temporary table, `runner_orders_temp`, was created with the following adjustments:

- Standardized data formats in the `pickup_time`, `distance`, `duration`, and `cancellation` columns.

```
CREATE TEMP TABLE runner_orders_temp AS
SELECT
  order_id,
  runner_id,
  CASE
      WHEN pickup_time LIKE 'null' THEN NULL
      ELSE pickup_time
  END AS pickup_time,
  CASE
      WHEN distance LIKE 'null' THEN NULL
      WHEN distance LIKE '%km' THEN CAST(TRIM('km' FROM distanc
      ELSE CAST(distance AS NUMERIC)
  END AS distance,
  CASE
      WHEN duration LIKE 'null' THEN NULL
      WHEN duration LIKE '%mins' THEN CAST(TRIM('mins' FROM dura
      WHEN duration LIKE '%minute' THEN CAST(TRIM('minute' FROM
      WHEN duration LIKE '%minutes' THEN CAST(TRIM('minutes' FRO
      ELSE CAST(duration AS NUMERIC)
  END AS duration,
  CASE
      WHEN cancellation IS NULL OR cancellation LIKE 'null' THEN
      ELSE cancellation
  END AS cancellation
FROM pizza_runner.runner_orders;


SELECT * FROM runner_orders_temp
```



order_id	runner_id	pickup_time	distance	duration	cancellation
1	1	2020-01-01 18:15:34	20	32	
2	1	2020-01-01 19:10:54	20	27	
3	1	2020-01-03 00:12:37	13.4	20	null
4	2	2020-01-04 13:53:03	23.4	40	null
5	3	2020-01-08 21:10:57	10	15	null
6	3	null	null	null	Restaurant Cancellation
7	2	2020-01-08 21:30:45	25	25	null
8	2	2020-01-10 00:15:02	23.4	15	null
9	2	null	null	null	Customer Cancellation
10	1	2020-01-11 18:50:20	10	10	null

❓ Case Study Questions & Analysis

This case study has four major areas of focus, each with its own questions. Each question can be answered using a single SQL statement.

A. Pizza Metrics

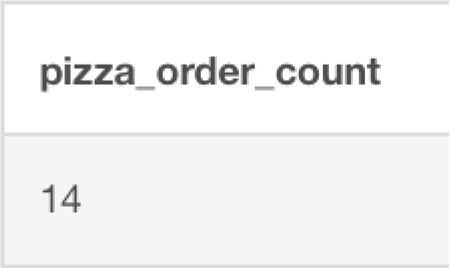
1. How many pizzas were ordered?

Steps:

- Use the `COUNT(*)` function to calculate the total number of rows in the `customer_orders_temp` table, representing the total number of pizzas ordered.

```
SELECT COUNT(*) AS pizza_order_count
FROM customer_orders;
```

Answer: A total of 14 pizzas were ordered.



pizza_order_count
14

---

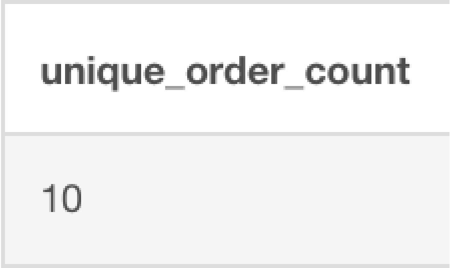
2. How many unique customer orders were made?

Steps:

- Use the `COUNT` function with the `DISTINCT` keyword on the `order_id` column to count the number of unique orders in the `customer_orders_temp` table.

```
SELECT COUNT(DISTINCT order_id) AS unique_order_count
FROM customer_orders;
```

Answer: There are 10 unique customer orders.



unique_order_count
10

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

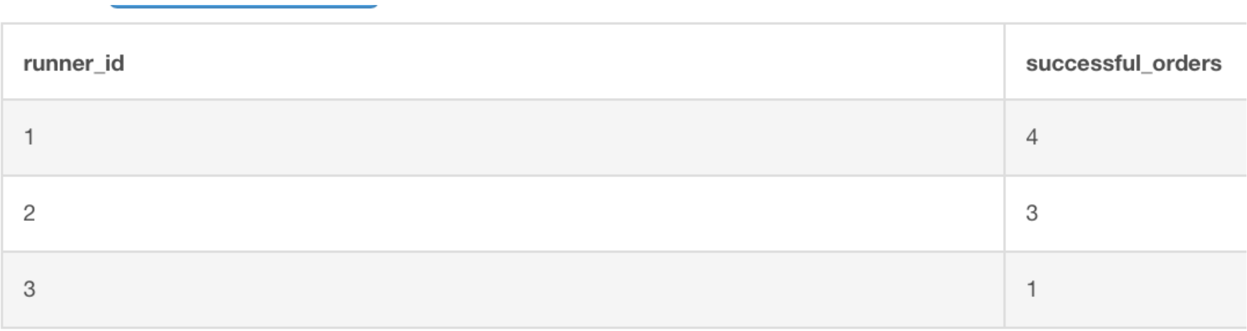
Steps:

- Select the `runner_id` column and count the number of `order_id` entries where the `distance` is greater than 0, indicating successful deliveries.
- Group results by `runner_id` to calculate the number of successful orders per runner.

```
SELECT
  runner_id,
  COUNT(order_id) AS successful_orders
FROM runner_orders
WHERE distance > 0
GROUP BY runner_id;
```

Answer:

- Runner 1: 4 successful deliveries
- Runner 2: 3 successful deliveries
- Runner 3: 1 successful delivery



runner_id	successful_orders
1	4
2	3
3	1

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

Steps:

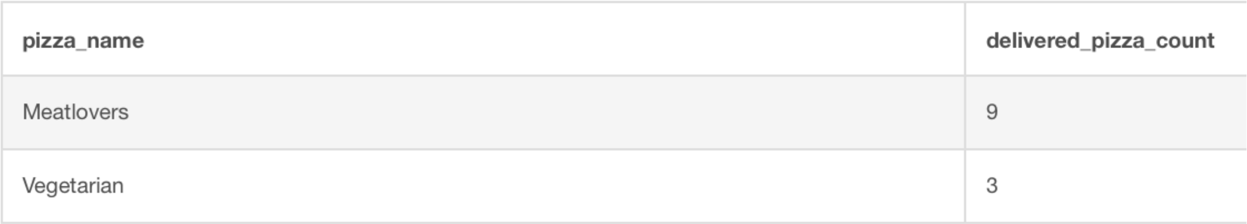
- Join the `customer_orders` and `runner_orders` tables on the `order_id` column to filter for delivered orders (`distance > 0`).

- Join the resulting table with `pizza_names` on the `pizza_id` column to retrieve pizza names.
- Count the occurrences of each `pizza_name` and group the results.

```
SELECT
  p.pizza_name,
  COUNT(c.pizza_id) AS delivered_pizza_count
FROM customer_orders_temp AS c
JOIN runner_orders_temp AS r
  ON c.order_id = r.order_id
JOIN pizza_names AS p
  ON c.pizza_id = p.pizza_id
WHERE r.distance > 0
GROUP BY p.pizza_name;
```

Answer:

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



pizza_name	delivered_pizza_count
Meatlovers	9
Vegetarian	3

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

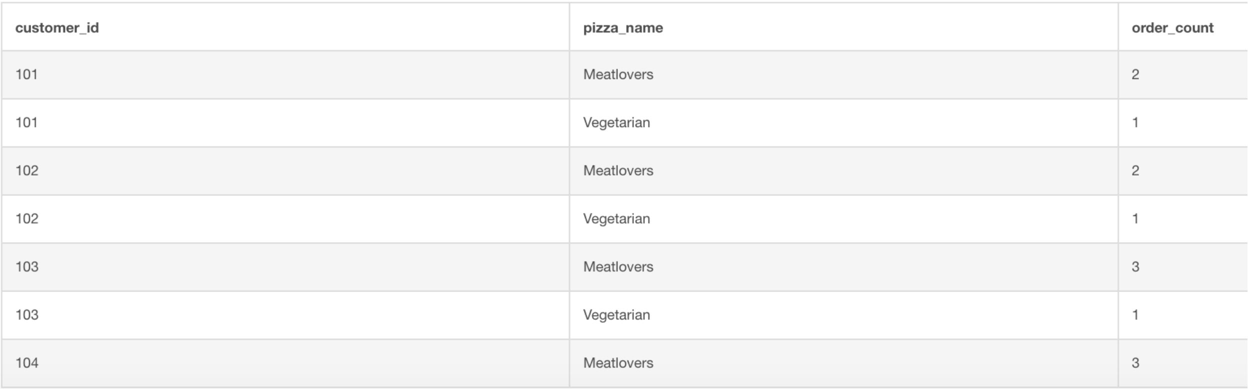
Steps:

- Join the `customer_orders` table with `pizza_names` on the `pizza_id` column.
- Group results by `customer_id` and `pizza_name`, and count the occurrences of each pizza type for each customer.

```
SELECT
  c.customer_id,
  p.pizza_name,
  COUNT(p.pizza_name) AS order_count
FROM customer_orders_temp AS c
JOIN pizza_names AS p
  ON c.pizza_id = p.pizza_id
GROUP BY c.customer_id, p.pizza_name
ORDER BY c.customer_id;
```

Answer:

- Customer 101 ordered 2 Meatlovers pizzas and 1 Vegetarian pizza.
- Customer 102 ordered 2 Meatlovers pizzas and 2 Vegetarian pizzas.
- Customer 103 ordered 3 Meatlovers pizzas and 1 Vegetarian pizza.
- Customer 104 ordered 1 Meatlovers pizza.
- Customer 105 ordered 1 Vegetarian pizza.



customer_id	pizza_name	order_count
101	Meatlovers	2
101	Vegetarian	1
102	Meatlovers	2
102	Vegetarian	1
103	Meatlovers	3
103	Vegetarian	1
104	Meatlovers	3

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

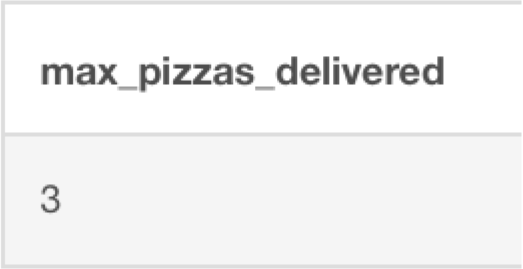
Steps:

- Create a CTE named `pizza_count_cte` to calculate the number of pizzas delivered for each order (`distance > 0`).

- Use the `MAX` function to find the highest pizza count from the CTE.

```
WITH pizza_count_cte AS (
  SELECT
    c.order_id,
    COUNT(c.pizza_id) AS pizza_per_order
  FROM customer_orders_temp AS c
  JOIN runner_orders_temp AS r
    ON c.order_id = r.order_id
  WHERE r.distance > 0
  GROUP BY c.order_id
)
SELECT
  MAX(pizza_per_order) AS max_pizzas_delivered
FROM pizza_count_cte
```

Answer: The maximum number of pizzas delivered in a single order is 3.



max_pizzas_delivered
3

7. For each customer, how many delivered pizzas had at least one change and how many had no changes?

Steps:

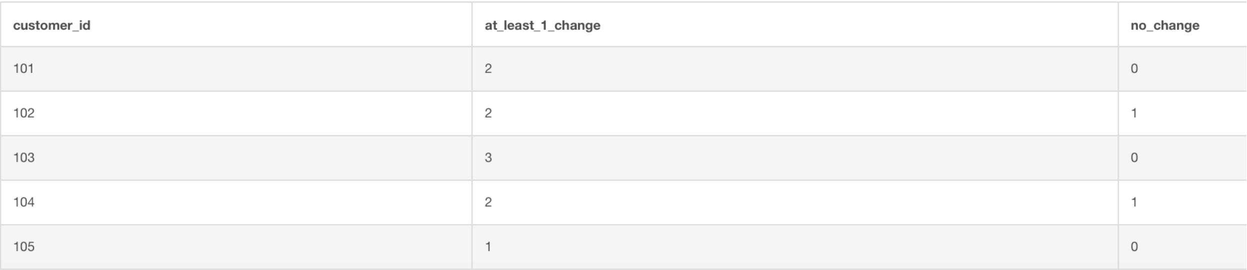
- Join the `customer_orders` and `runner_orders` tables on the `order_id` column to filter for delivered orders (`distance > 0`).
- Use conditional aggregation with `SUM` and `CASE` to count pizzas with at least one change (`exclusions` or `extras` not empty) and pizzas with no changes (`exclusions` and `extras` empty).

- Group results by `customer_id`.

```
SELECT
  c.customer_id,
  SUM(
    CASE WHEN c.exclusions <> ' ' OR c.extras <> ' ' THEN 1
    ELSE 0
    END) AS at_least_1_change,
  SUM(
    CASE WHEN c.exclusions = ' ' AND c.extras = ' ' THEN 1
    ELSE 0
    END) AS no_change
FROM customer_orders_temp AS c
JOIN runner_orders_temp AS r
  ON c.order_id = r.order_id
WHERE r.distance > 0
GROUP BY c.customer_id
ORDER BY c.customer_id;
```

Answer:

- Customers 101, 102and 105 made no changes to their pizzas.
- Customers 103 and 104 requested at least one change.



customer_id	at_least_1_change	no_change
101	2	0
102	2	1
103	3	0
104	2	1
105	1	0

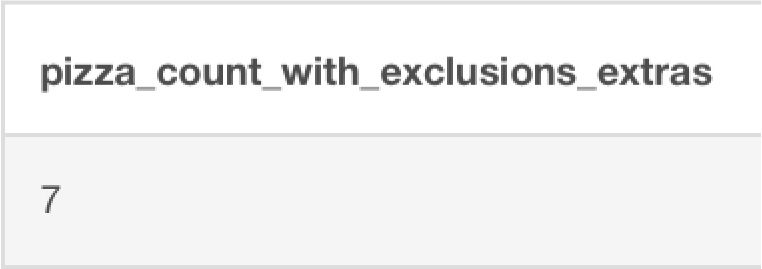
8. How many pizzas were delivered that had both exclusions and extras?

Steps:

- Join the `customer_orders` and `runner_orders` tables on the `order_id` column to filter for delivered orders (`distance > 0`).
- Use `SUM` and a `CASE` statement to count pizzas where both `exclusions` and `extras` are not empty.
- Filter the results to ensure only relevant pizzas are counted.

```
SELECT
  SUM(
    CASE WHEN exclusions IS NOT NULL AND extras IS NOT NULL T
HEN 1
    ELSE 0
    END) AS pizza_count_with_exclusions_extras
FROM customer_orders_temp AS c
JOIN runner_orders_temp AS r
  ON c.order_id = r.order_id
WHERE r.distance > 0
  AND exclusions <> ' '
  AND extras <> ' ';
```

Answer: Only 7 pizza was delivered with both exclusions and extras.



pizza_count_with_exclusions_extras
7

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

Steps:

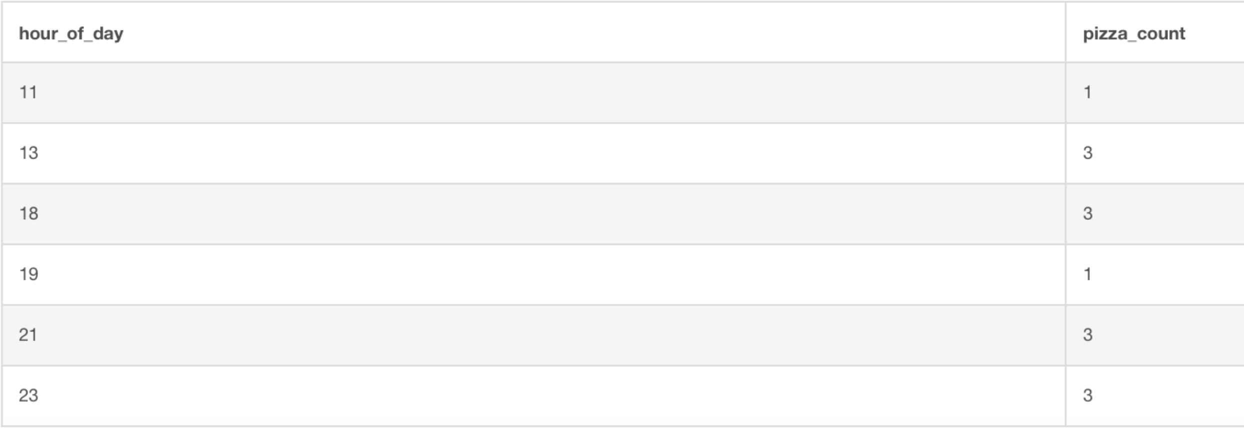
- Extract the hour from the `order_time` column using `EXTRACT(HOUR FROM order_time)`.

- Count the number of orders for each hour and group by the extracted hour.

```
SELECT
  EXTRACT(HOUR FROM order_time) AS hour_of_day,
  COUNT(order_id) AS pizza_count
FROM customer_orders_temp
GROUP BY EXTRACT(HOUR FROM order_time)
ORDER BY hour_of_day;
```

Answer:

- The highest volume of orders occurred at 13 (1:00 pm), 18 (6:00 pm), 21 (9:00 pm), and 23 (11:00 pm).
- The lowest volume of orders occurred at 11 (11:00 am) and 19 (7:00 pm)



hour_of_day	pizza_count
11	1
13	3
18	3
19	1
21	3
23	3

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

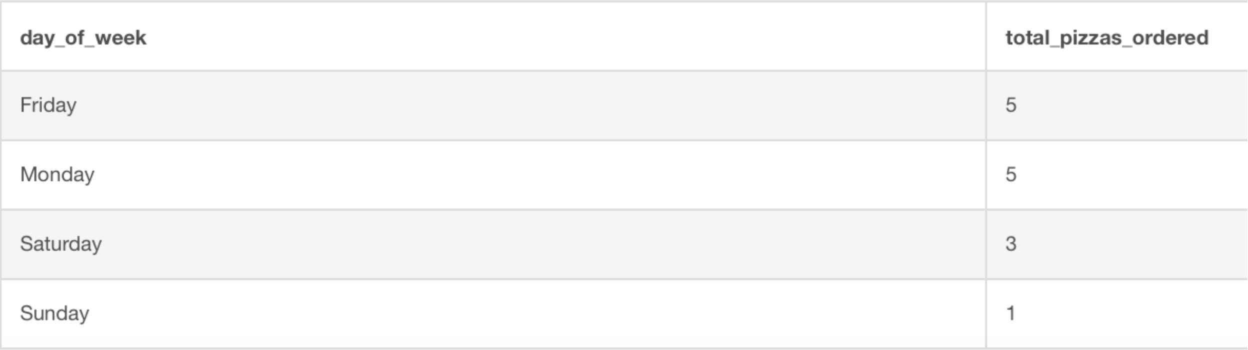
Steps:

- Use the `FORMAT` function with `TO_CHAR` to adjust the `order_time` so that the first day of the week starts on Monday.
- Group results by the formatted day of the week and count the total orders for each day.

```
SELECT
  TO_CHAR(order_time + interval '2 days', 'Day') AS day_of_week,
  COUNT(order_id) AS total_pizzas_ordered
FROM customer_orders_temp
GROUP BY TO_CHAR(order_time + interval '2 days', 'Day')
ORDER BY day_of_week;
```

Answer:

- 5 pizzas were ordered on both Friday and Monday.
- 3 pizzas were ordered on Saturday.
- 1 pizza was ordered on Sunday.



day_of_week	total_pizzas_ordered
Friday	5
Monday	5
Saturday	3
Sunday	1

🛠️ Tools Used

- **Languages**: SQL (PostgreSQL dialect)
- **Interactive Environment**: DB Fiddle for running SQL queries
- **Database Management**: PostgreSQL for structured queries and analysis

🌐 Additional Links

- GitHub Repository: Access the complete codebase and SQL queries.

- LinkedIn: Connect with me for further discussion or opportunities.

🍕 Case Study #2B: Pizza Runner

🍕 Case Study #2C: Pizza Ingredients

🍕 Case Study #2D: Pizza Pricing