DATAWITHDANNY.COM

START YOUR SQL ENGINES

PIZZA RUNNER

CASE STUDY #2

8 WEEK SQL CHALLENGE

8WEEKSQLCHALLENGE.COM

INTRODUCTION

Did you know that over 115 million kilograms of pizza is consumed daily worldwide??? (Well according to Wikipedia anyway…)

Danny was scrolling through his Instagram feed when something really caught his eye - “80s Retro Styling and Pizza Is The Future!”

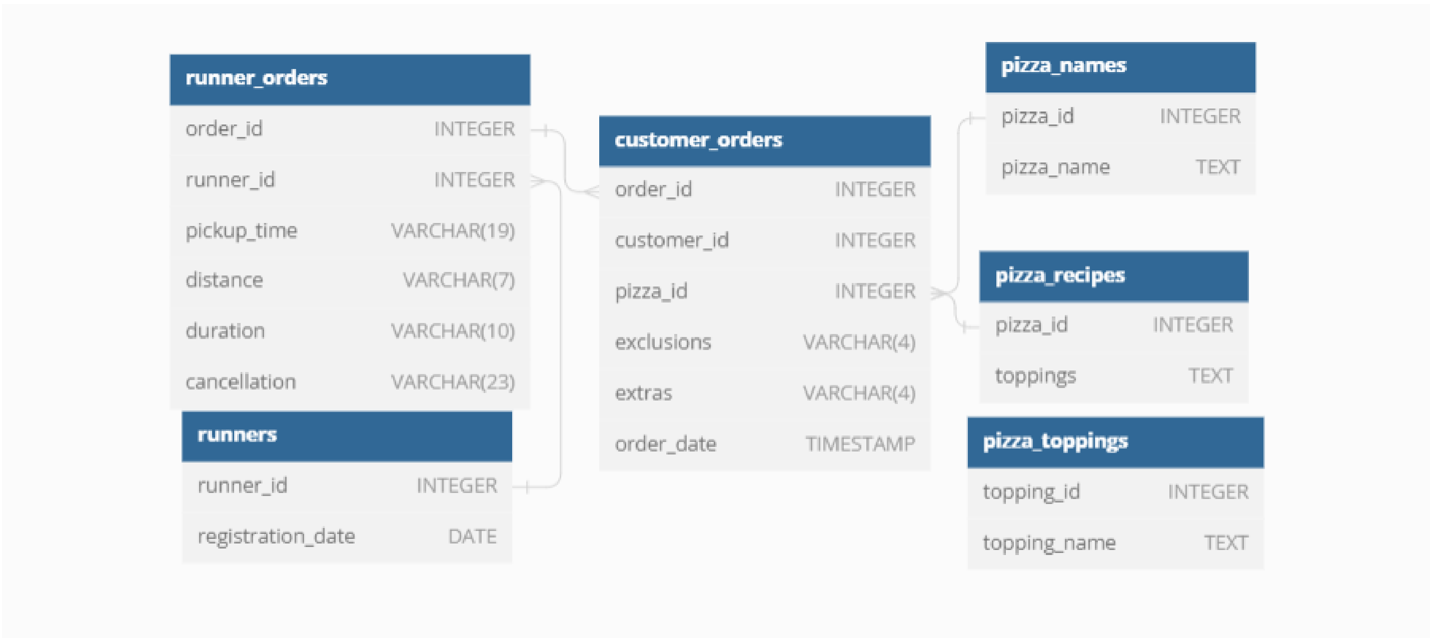
Danny was sold on the idea, but he knew that pizza alone was not going to help him get seed funding to expand his new Pizza Empire - so he had one more genius idea to combine with it - he was going to Uberize it - and so Pizza Runner was launched!

Danny started by recruiting “runners” to deliver fresh pizza from Pizza Runner Headquarters (otherwise known as Danny’s house) and also maxed out his credit card to pay freelance developers to build a mobile app to accept orders from customers.

Available Data

Because Danny had a few years of experience as a data scientist - he was very aware that data collection was going to be critical for his business' growth.
He has prepared for us an entity relationship diagram of his database design but requires further assistance to clean his data and apply some basic calculations so he can better direct his runners and optimise Pizza Runner's operations.
All datasets exist within the pizza_runner database schema - be sure to include this reference within your SQL scripts as you start exploring the data and answering the case study questions.

Entity Relationship Diagram



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

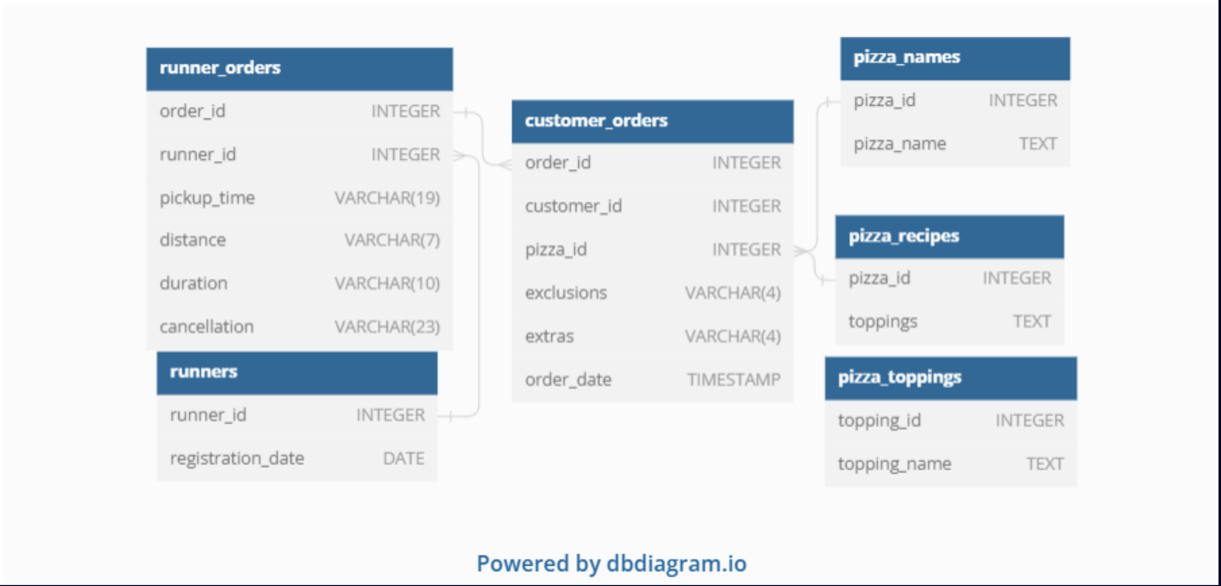
customer_orders	
order_id	INTEGER
customer_id	INTEGER
pizza_id	INTEGER
exclusions	VARCHAR(4)
extras	VARCHAR(4)
order_date	TIMESTAMP

pizza_names	
pizza_id	INTEGER
pizza_name	TEXT

pizza_recipes	
pizza_id	INTEGER
toppings	TEXT

pizza_toppings	
topping_id	INTEGER
topping_name	TEXT

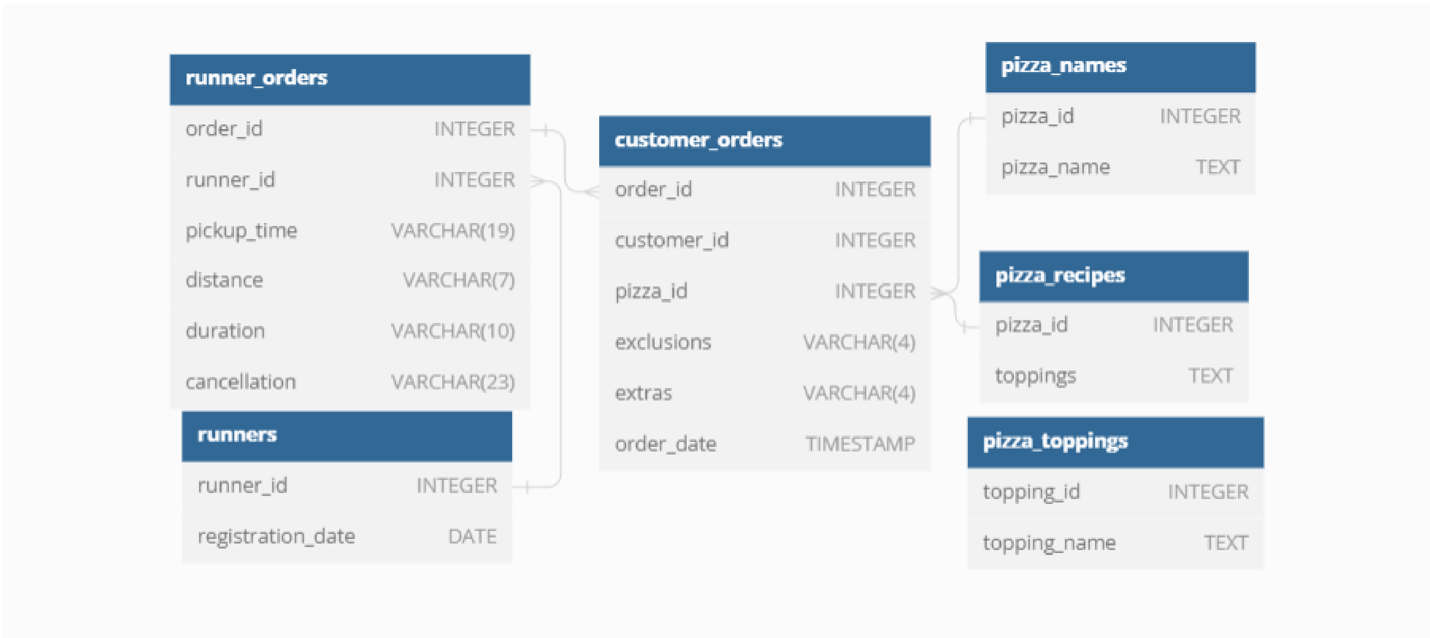
Entity Relationship Diagram



Available Data

Because Danny had a few years of experience as a data scientist - he was very aware that data collection was going to be critical for his business' growth.
He has prepared for us an entity relationship diagram of his database design but requires further assistance to clean his data and apply some basic calculations so he can better direct his runners and optimise Pizza Runner's operations.
All datasets exist within the pizza_runner database schema - be sure to include this reference within your SQL scripts as you start exploring the data and answering the case study questions.

Entity Relationship Diagram



Available Data

Table 1: runners

The runners table shows the registration_date for each new runner

runner_id	registration_date
1	2021-01-01
2	2021-01-03
3	2021-01-08
4	2021-01-15

Table 2: customer_orders

Customer pizza orders are captured in the customer_orders table with 1 row for each individual pizza that is part of the order.

The pizza_id relates to the type of pizza which was ordered whilst the exclusions are the ingredient_id values which should be removed from the pizza and the extras are the ingredient_id values which need to be added to the pizza.

Note that customers can order multiple pizzas in a single order with varying exclusions and extras values even if the pizza is the same type!

The exclusions and extras columns will need to be cleaned up before using them in your queries.

order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1			2021-01-01 18:05:02
2	101	1			2021-01-01 19:00:52
3	102	1			2021-01-02 23:51:23
3	102	2		NaN	2021-01-02 23:51:23
4	103	1	4		2021-01-04 13:23:46
4	103	1	4		2021-01-04 13:23:46
4	103	2	4		2021-01-04 13:23:46
5	104	1	null	1	2021-01-08 21:00:29
6	101	2	null	null	2021-01-08 21:03:13
7	105	2	null	1	2021-01-08 21:20:29
8	102	1	null	null	2021-01-09 23:54:33
9	103	1	4	1, 5	2021-01-10 11:22:59
10	104	1	null	null	2021-01-11 18:34:49
10	104	1	2, 6	1, 4	2021-01-11 18:34:49

Table 3: runner_orders

After each orders are received through the system - they are assigned to a runner - however not all orders are fully completed and can be cancelled by the restaurant or the customer.

The pickup_time is the timestamp at which the runner arrives at the Pizza Runner headquarters to pick up the freshly cooked pizzas. The distance and duration fields are related to how far and long the runner had to travel to deliver the order to the respective customer.

There are some known data issues with this table so be careful when using this in your queries - make sure to check the data types for each column in the schema SQL!

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	NaN
4	2	2021-01-04 13:53:03	23.4	40	NaN
5	3	2021-01-08 21:10:57	10	15	NaN
6	3	null	null	null	Restaurant Cancellation
7	2	2020-01-08 21:30:45	25km	25mins	null
8	2	2020-01-10 00:15:02	23.4 km	15 minute	null
9	2	null	null	null	Customer Cancellation
10	1	2020-01-11 18:50:20	10km	10minutes	null

Table 4: pizza_names
At the moment - Pizza Runner only has 2 pizzas available the Meat Lovers or Vegetarian!

pizza_id	**pizza_name**
1	Meat Lovers
2	Vegetarian

Table 5: pizza_recipes
Each pizza_id has a standard set of toppings which are used as part of the pizza recipe.

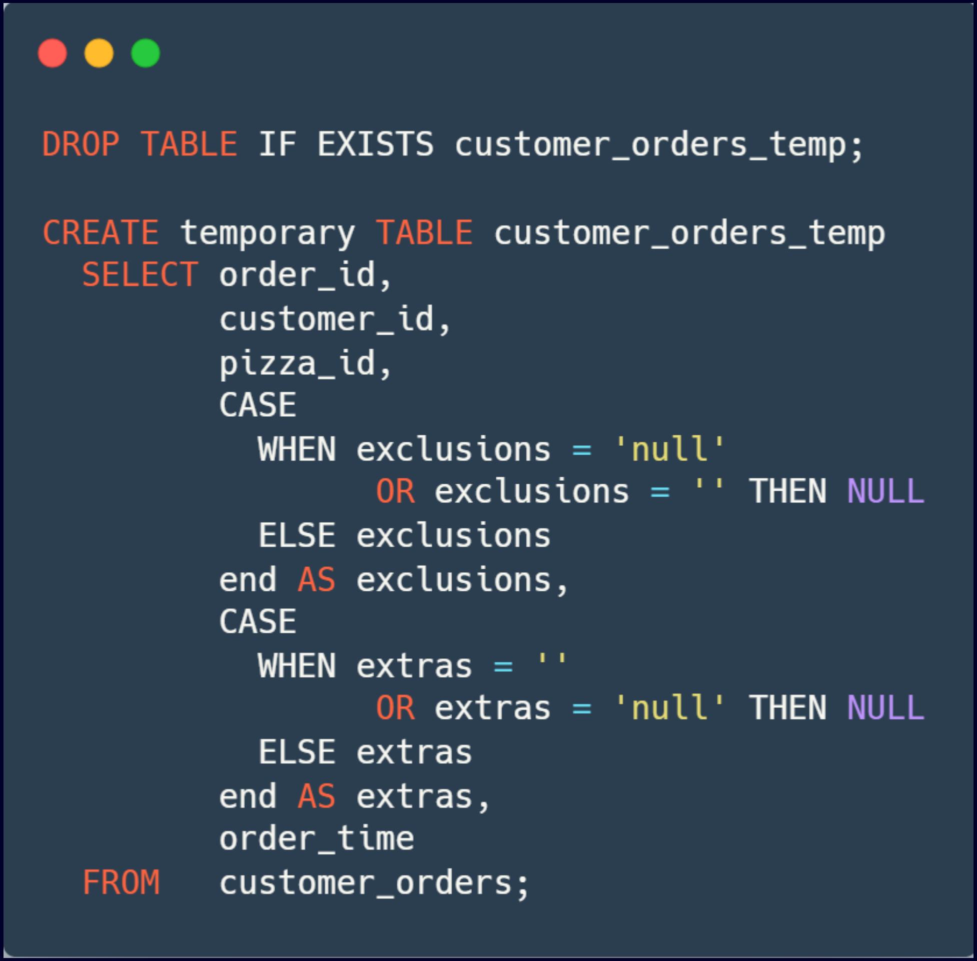
pizza_id	**toppings**
1	1, 2, 3, 4, 5, 6, 8, 10
2	4, 6, 7, 9, 11, 12

Table 6: pizza_toppings
This table contains all of the topping_name values with their corresponding topping_id value.

topping_id	**topping_name**
1	Bacon
2	BBQ Sauce
3	Beef
4	Cheese
5	Chicken
6	Mushrooms
7	Onions
8	Pepperoni
9	Peppers
10	Salami
11	Tomatoes
12	Tomato Sauce

DATA CLEANING QUERIES

customer_orders table cleaning:

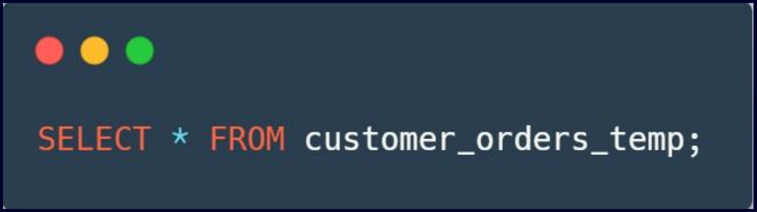


```
DROP TABLE IF EXISTS customer_orders_temp;

CREATE temporary TABLE customer_orders_temp
  SELECT order_id,
         customer_id,
         pizza_id,
         CASE
           WHEN exclusions = 'null'
                 OR exclusions = '' THEN NULL
           ELSE exclusions
         end AS exclusions,
         CASE
           WHEN extras = ''
                 OR extras = 'null' THEN NULL
           ELSE extras
         end AS extras,
         order_time
  FROM   customer_orders;
```

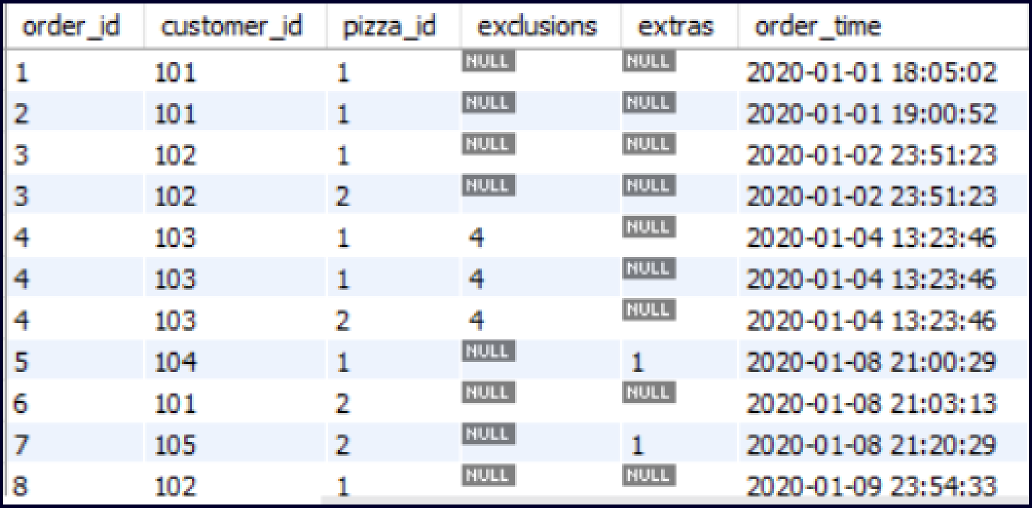
DATA CLEANING QUERIES

Customer_orders_temp table:



```
SELECT * FROM customer_orders_temp;
```

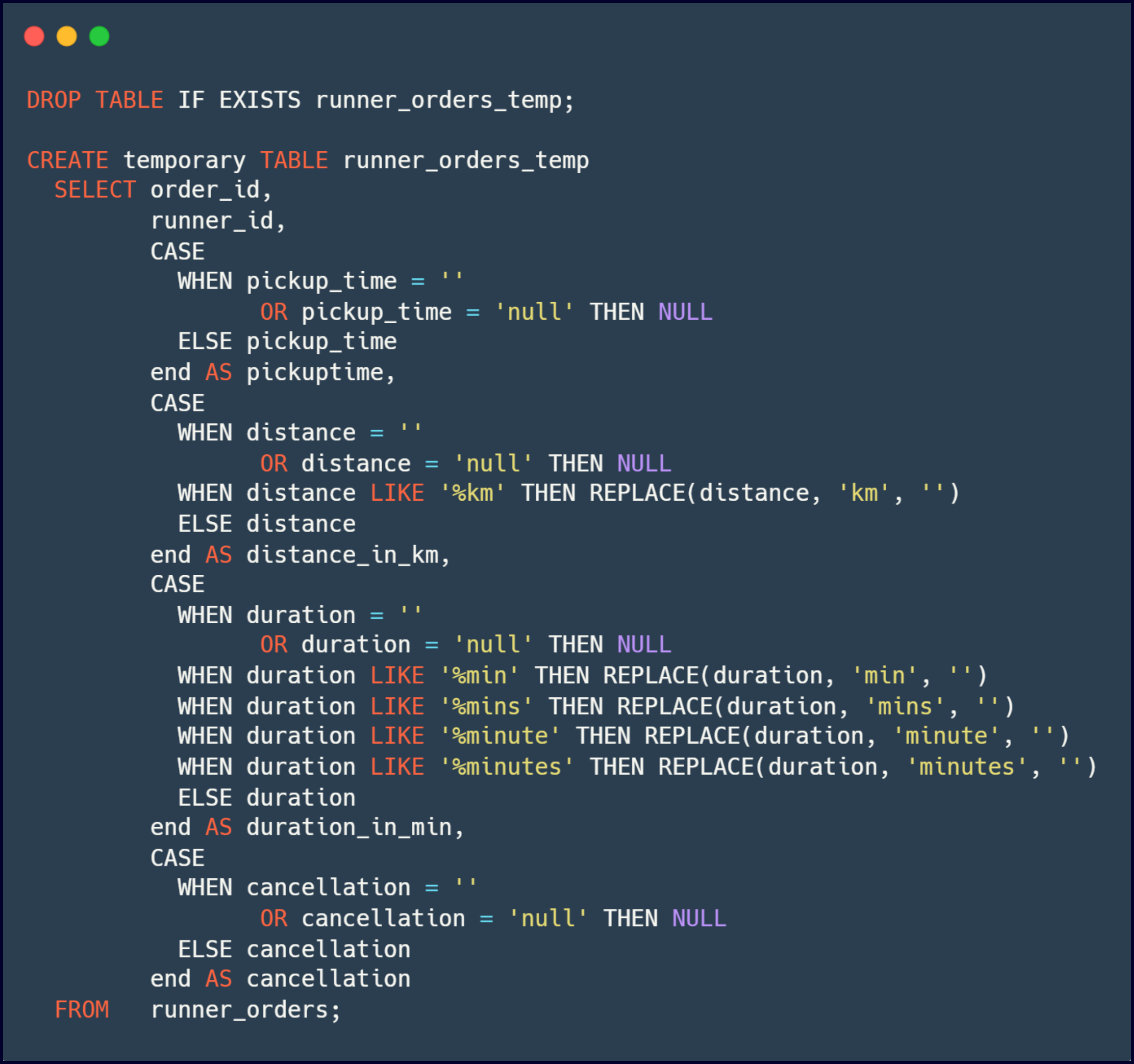
Output



order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1	NULL	NULL	2020-01-01 18:05:02
2	101	1	NULL	NULL	2020-01-01 19:00:52
3	102	1	NULL	NULL	2020-01-02 23:51:23
3	102	2	NULL	NULL	2020-01-02 23:51:23
4	103	1	4	NULL	2020-01-04 13:23:46
4	103	1	4	NULL	2020-01-04 13:23:46
4	103	2	4	NULL	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

DATA CLEANING QUERIES

runner_orders table cleaning:

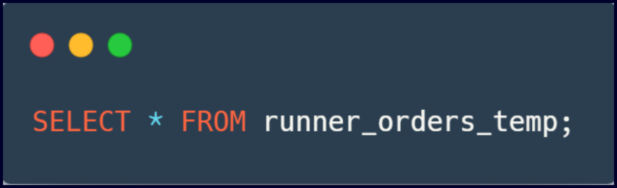


```sql
DROP TABLE IF EXISTS runner_orders_temp;

CREATE temporary TABLE runner_orders_temp
  SELECT order_id,
         runner_id,
         CASE
           WHEN pickup_time = ''
                 OR pickup_time = 'null' THEN NULL
           ELSE pickup_time
         end AS pickuptime,
         CASE
           WHEN distance = ''
                 OR distance = 'null' THEN NULL
           WHEN distance LIKE '%km' THEN REPLACE(distance, 'km', '')
           ELSE distance
         end AS distance_in_km,
         CASE
           WHEN duration = ''
                 OR duration = 'null' THEN NULL
           WHEN duration LIKE '%min' THEN REPLACE(duration, 'min', '')
           WHEN duration LIKE '%mins' THEN REPLACE(duration, 'mins', '')
           WHEN duration LIKE '%minute' THEN REPLACE(duration, 'minute', '')
           WHEN duration LIKE '%minutes' THEN REPLACE(duration, 'minutes', '')
           ELSE duration
         end AS duration_in_min,
         CASE
           WHEN cancellation = ''
                 OR cancellation = 'null' THEN NULL
           ELSE cancellation
         end AS cancellation
  FROM   runner_orders;
```

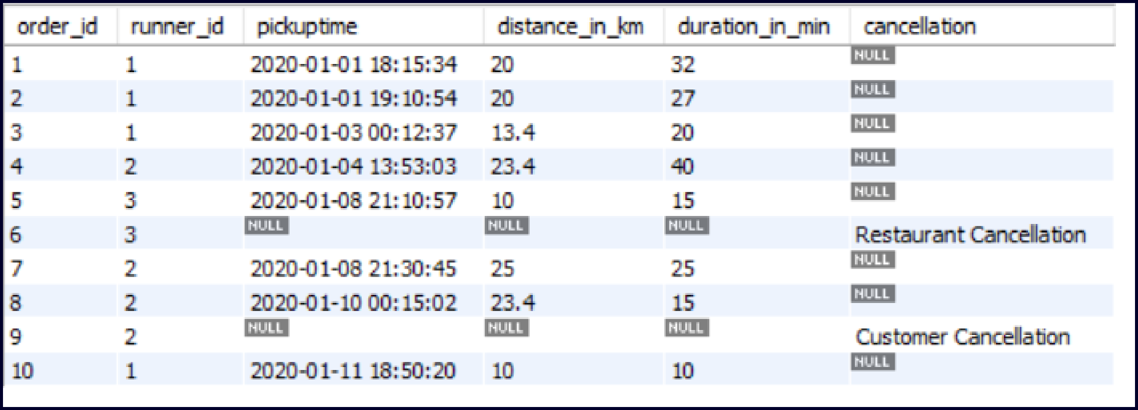
DATA CLEANING QUERIES

runner_orders_temp table:



```
SELECT * FROM runner_orders_temp;
```

Output

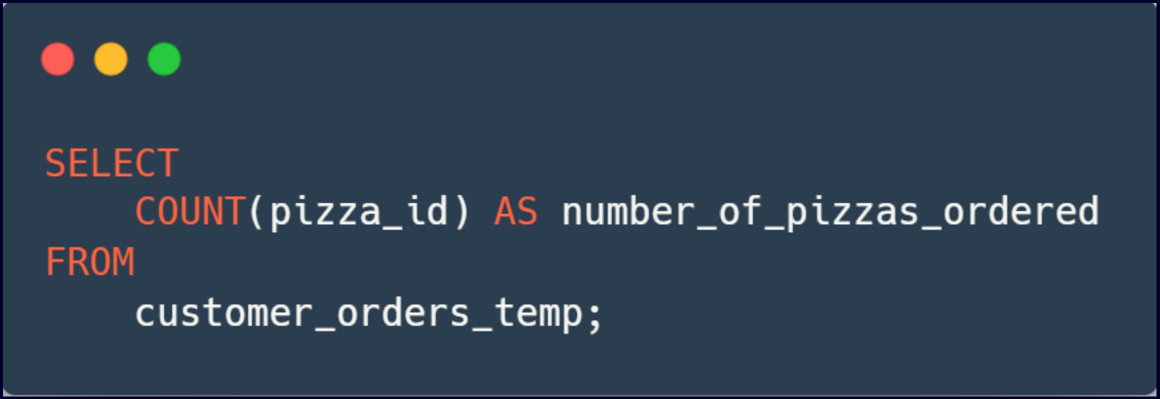


order_id	runner_id	pickuptime	distance_in_km	duration_in_min	cancellation
1	1	2020-01-01 18:15:34	20	32	NULL
2	1	2020-01-01 19:10:54	20	27	NULL
3	1	2020-01-03 00:12:37	13.4	20	NULL
4	2	2020-01-04 13:53:03	23.4	40	NULL
5	3	2020-01-08 21:10:57	10	15	NULL
6	3	NULL	NULL	NULL	Restaurant Cancellation
7	2	2020-01-08 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

PIZZA METRICS

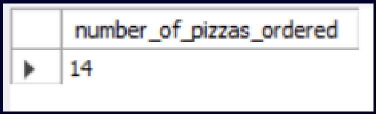
1. How many pizzas were ordered?

Query



```
SELECT
    COUNT(pizza_id) AS number_of_pizzas_ordered
FROM
    customer_orders_temp;
```

Output



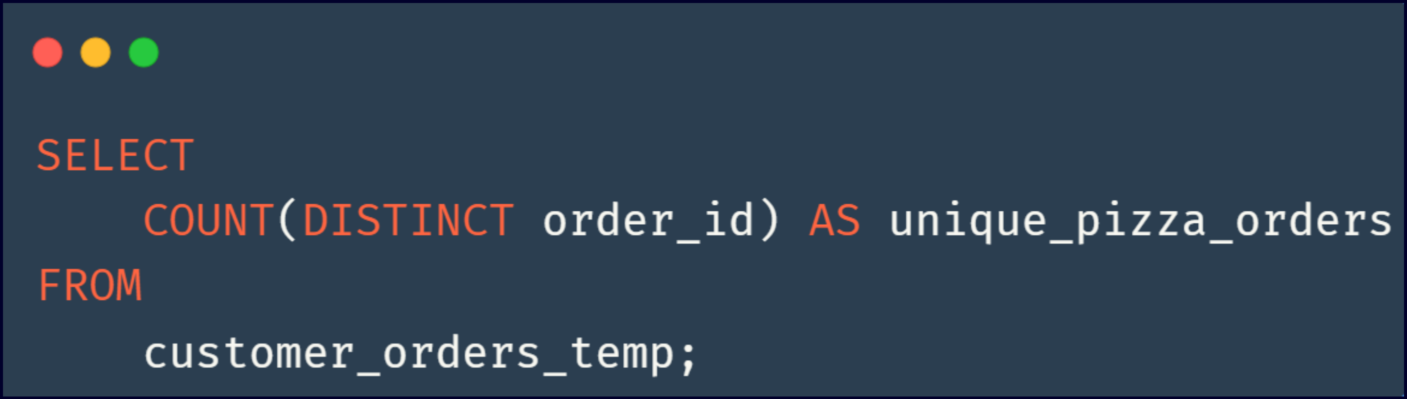
```
number_of_pizzas_ordered
14
```

Analysis

1. There have been a total of 14 pizzas ordered.

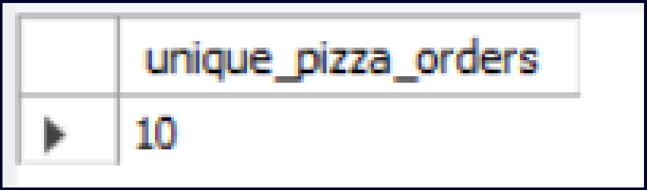
2. How many unique customer orders were made?

Query



```sql
SELECT
    COUNT(DISTINCT order_id) AS unique_pizza_orders
FROM
    customer_orders_temp;
```

Output



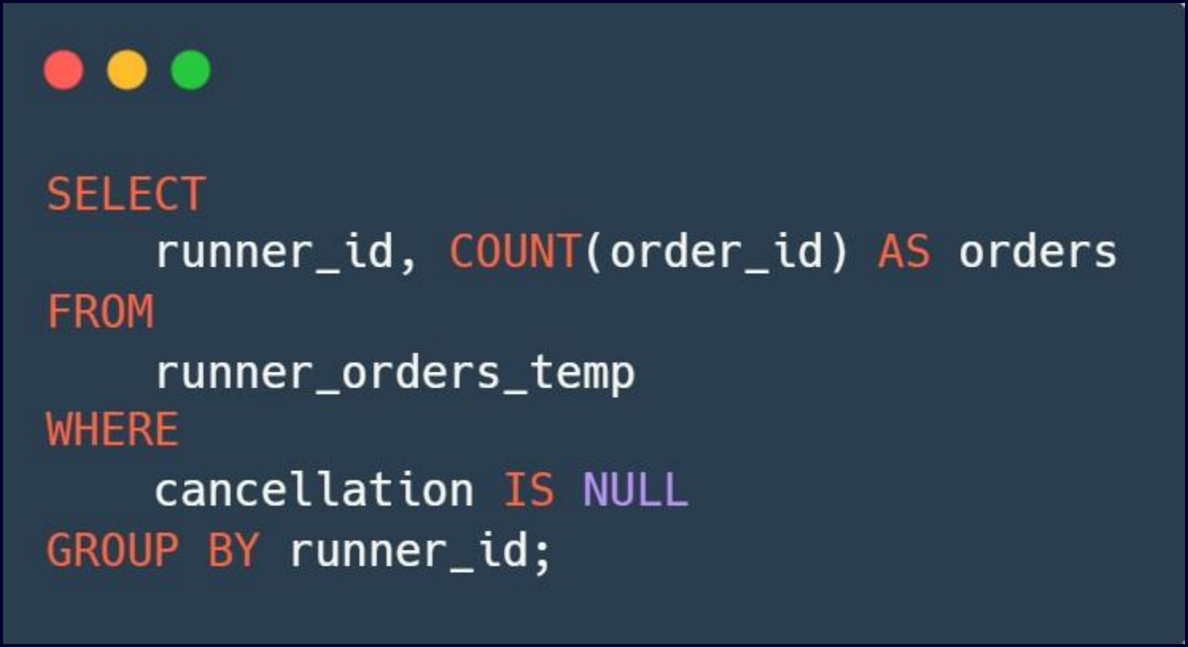
	unique_pizza_orders
▶	10

Analysis

1. There have been a total of 10 unique orders.

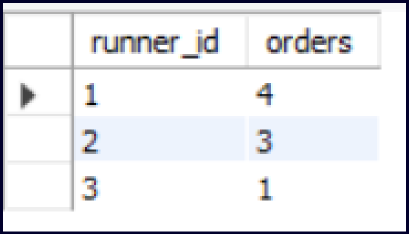
3. How many successful orders were delivered by each runner?

Query



```
SELECT
     runner_id, COUNT(order_id) AS orders
FROM
     runner_orders_temp
WHERE
     cancellation IS NULL
GROUP BY runner_id;
```

Output



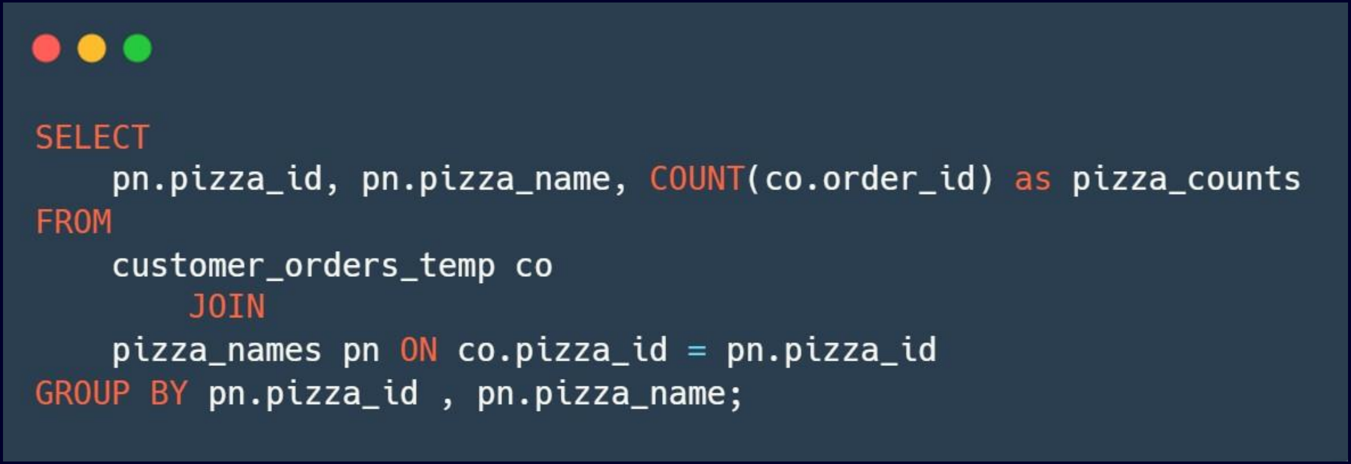
	runner_id	orders
▶	1	4
	2	3
	3	1

Analysis

1. Runner 1 has delivered the most number of orders with 4 orders followed by runner 2 with 3 orders and runner 3 with only one order.

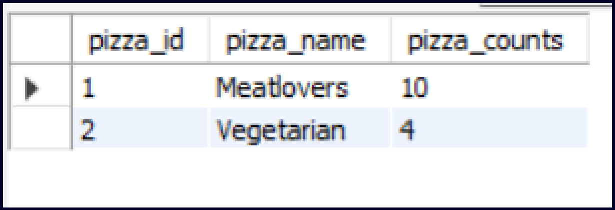
4. How many of each type of pizza was delivered?

Query



```
SELECT
    pn.pizza_id, pn.pizza_name, COUNT(co.order_id) as pizza_counts
FROM
    customer_orders_temp co
        JOIN
    pizza_names pn ON co.pizza_id = pn.pizza_id
GROUP BY pn.pizza_id , pn.pizza_name;
```

Output



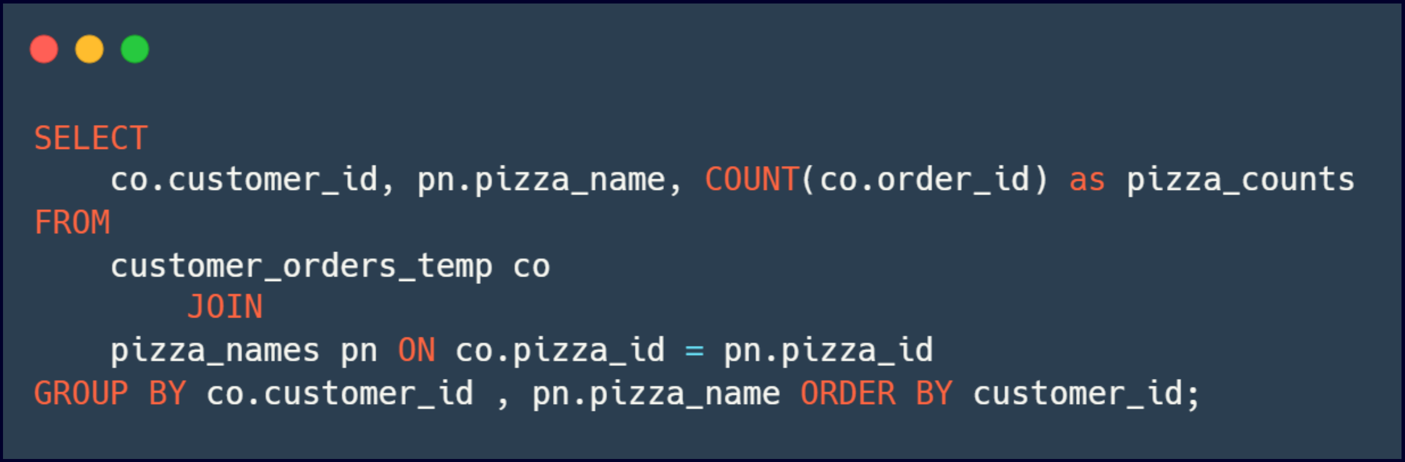
	pizza_id	pizza_name	pizza_counts
▶	1	Meatlovers	10
	2	Vegetarian	4

Analysis

1. Meatlovers pizza is the most ordered pizza with 10 orders.
2. Vegetarian Pizza has been ordered 4 times.

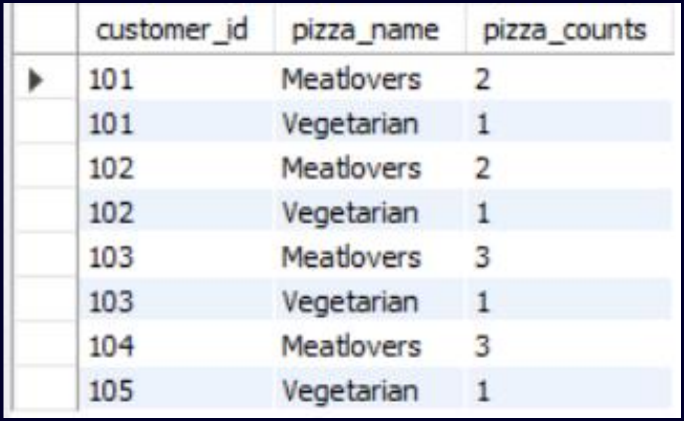
5. How many Vegetarian and Meatlovers were ordered by each customer?

Query



```
SELECT
    co.customer_id, pn.pizza_name, COUNT(co.order_id) as pizza_counts
FROM
    customer_orders_temp co
        JOIN
    pizza_names pn ON co.pizza_id = pn.pizza_id
GROUP BY co.customer_id , pn.pizza_name ORDER BY customer_id;
```

Output



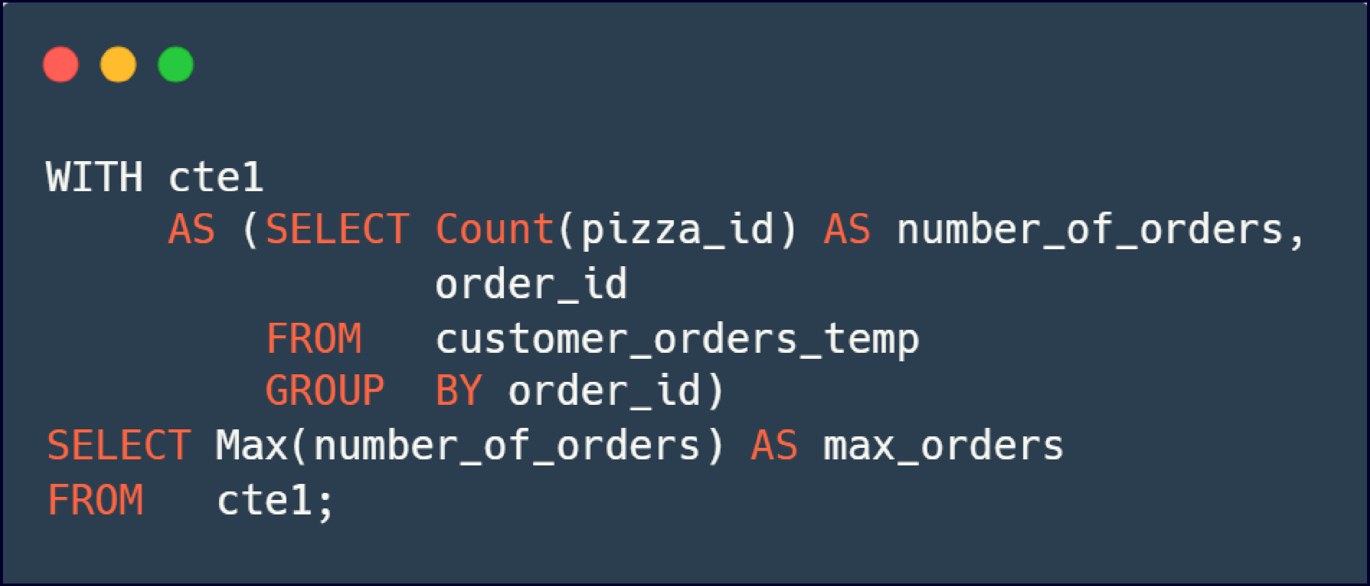
customer_id	pizza_name	pizza_counts
101	Meatlovers	2
101	Vegetarian	1
102	Meatlovers	2
102	Vegetarian	1
103	Meatlovers	3
103	Vegetarian	1
104	Meatlovers	3
105	Vegetarian	1

Analysis

1. Customers with id 104 and 103 have ordered the maximum number of meatlovers pizza.
2. Customers with id 101,102,103,105 have ordered one veg pizza each.

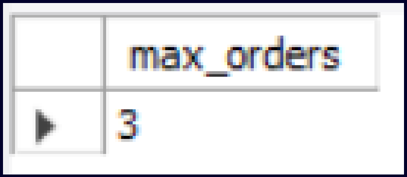
6. What was the maximum number of pizzas delivered in a single order?

Query



```
WITH cte1
     AS (SELECT Count(pizza_id) AS number_of_orders,
                order_id
         FROM   customer_orders_temp
         GROUP  BY order_id)
SELECT Max(number_of_orders) AS max_orders
FROM   cte1;
```

Output



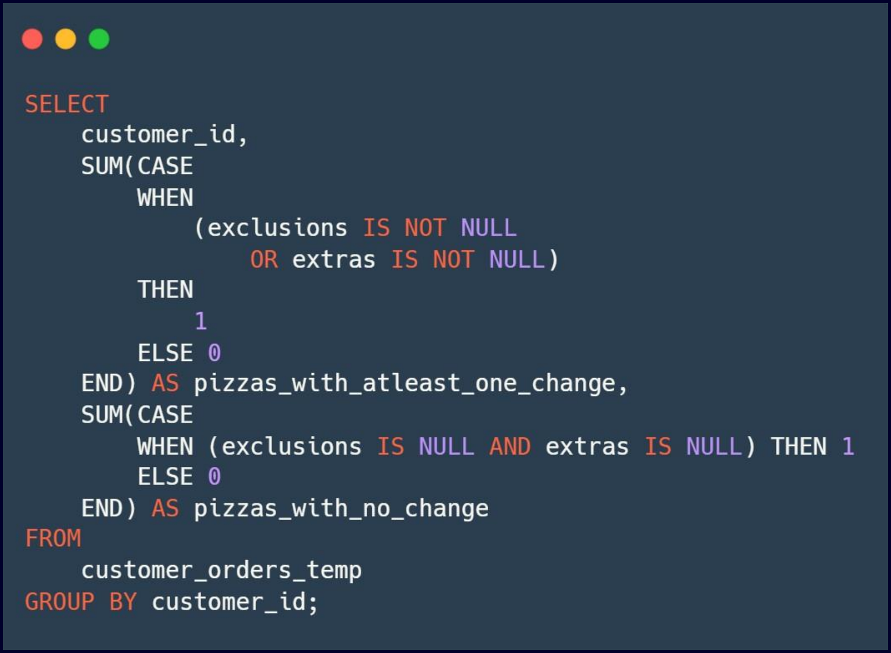
	max_orders
▶	3

Analysis

1. The maximum number of orders were 3.

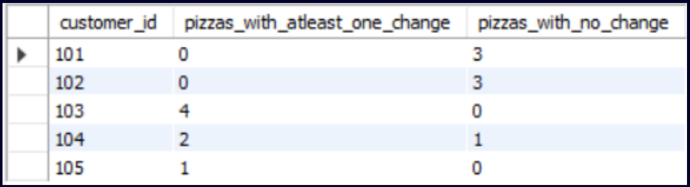
7. For each customer, how many delivered pizzas had at least 1 change and how many had no changes?

Query



```
SELECT
    customer_id,
    SUM(CASE
        WHEN
            (exclusions IS NOT NULL
                OR extras IS NOT NULL)
        THEN
            1
        ELSE 0
    END) AS pizzas_with_atleast_one_change,
    SUM(CASE
        WHEN (exclusions IS NULL AND extras IS NULL) THEN 1
        ELSE 0
    END) AS pizzas_with_no_change
FROM
    customer_orders_temp
GROUP BY customer_id;
```

Output



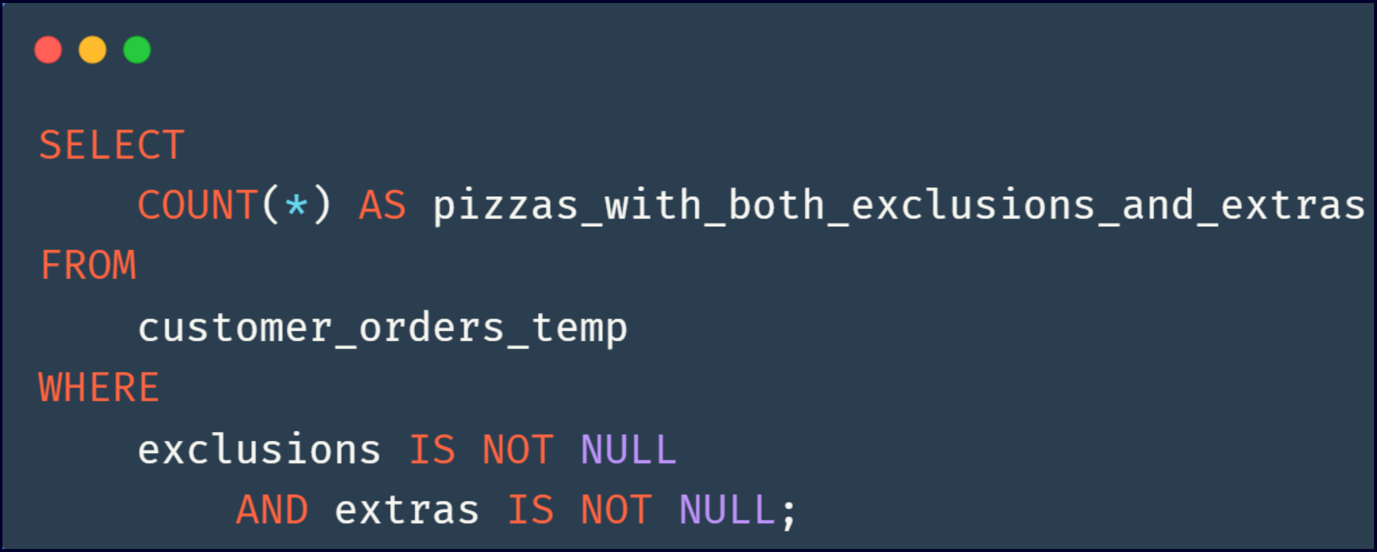
customer_id	pizzas_with_atleast_one_change	pizzas_with_no_change
101	0	3
102	0	3
103	4	0
104	2	1
105	1	0

Analysis

1. Customer with ID 101 and 102 has ordered 3 pizzas each and all of them are without changes.
2. Customers with ID 103 have ordered 4 pizzas and all of them had at least one change.
3. Customer with ID 104 has ordered a total 3 pizzas of which 2 were with changes and one without change.
4. Customer with ID 105 has ordered only one pizza and it had at least one change.of

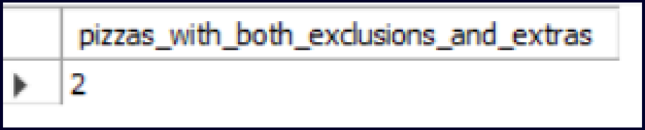
8. How many pizzas were delivered that had both exclusions and extras?

Query



```
SELECT
    COUNT(*) AS pizzas_with_both_exclusions_and_extras
FROM
    customer_orders_temp
WHERE
    exclusions IS NOT NULL
        AND extras IS NOT NULL;
```

Output



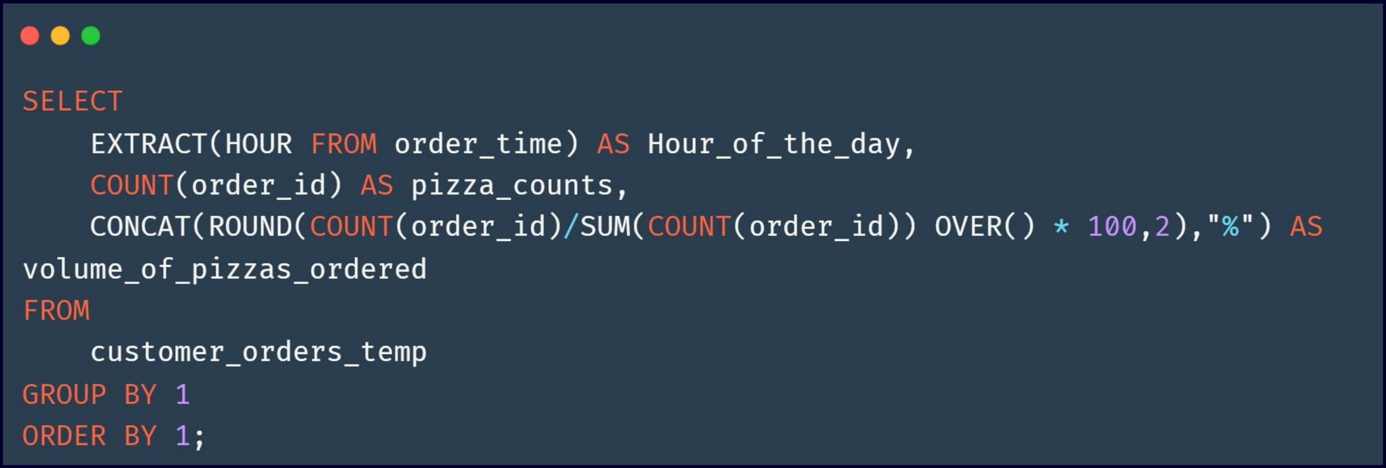
```
pizzas_with_both_exclusions_and_extras
2
```

Analysis

1. Of the total orders of 14 pizzas, two pizzas had both exclusions and inclusions.

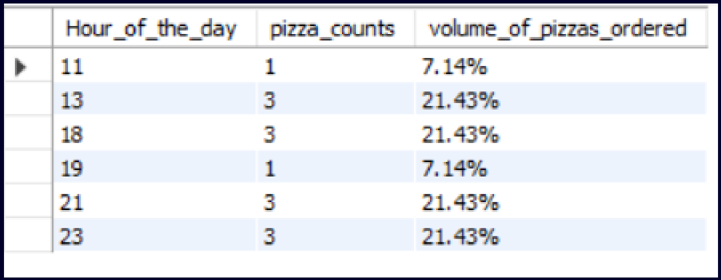
9. What was the total volume of pizzas ordered for each hour of the day?

Query



```
SELECT
    EXTRACT(HOUR FROM order_time) AS Hour_of_the_day,
    COUNT(order_id) AS pizza_counts,
    CONCAT(ROUND(COUNT(order_id)/SUM(COUNT(order_id)) OVER() * 100,2),"%") AS
volume_of_pizzas_ordered
FROM
    customer_orders_temp
GROUP BY 1
ORDER BY 1;
```

Output



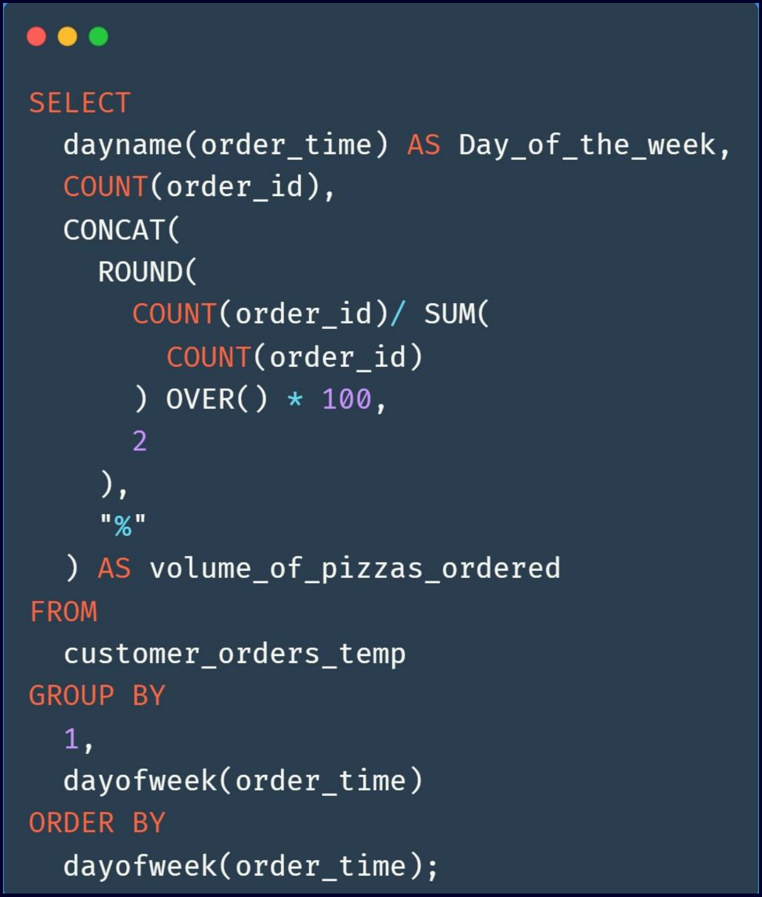
Hour_of_the_day	pizza_counts	volume_of_pizzas_ordered
11	1	7.14%
13	3	21.43%
18	3	21.43%
19	1	7.14%
21	3	21.43%
23	3	21.43%

Analysis

1. The pizza volume is highest during afternoon 13:00 hrs, 18:00 hrs, 21:00 hrs, 23:00 hrs indicating high volume during lunch and dinner.

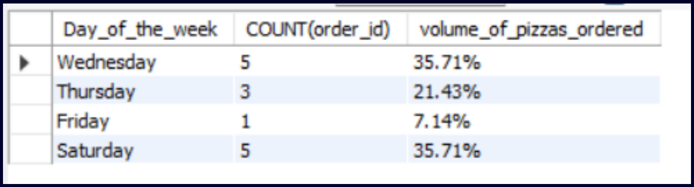
10. What was the volume of orders for each day of the week?

Query



```
SELECT
  dayname(order_time) AS Day_of_the_week,
  COUNT(order_id),
  CONCAT(
    ROUND(
      COUNT(order_id)/ SUM(
        COUNT(order_id)
      ) OVER() * 100,
      2
    ),
    "%"
  ) AS volume_of_pizzas_ordered
FROM
  customer_orders_temp
GROUP BY
  1,
  dayofweek(order_time)
ORDER BY
  dayofweek(order_time);
```

Output



Day_of_the_week	COUNT(order_id)	volume_of_pizzas_ordered
Wednesday	5	35.71%
Thursday	3	21.43%
Friday	1	7.14%
Saturday	5	35.71%

Analysis

1. The pizza volume is highest on Wednesday and Saturday.