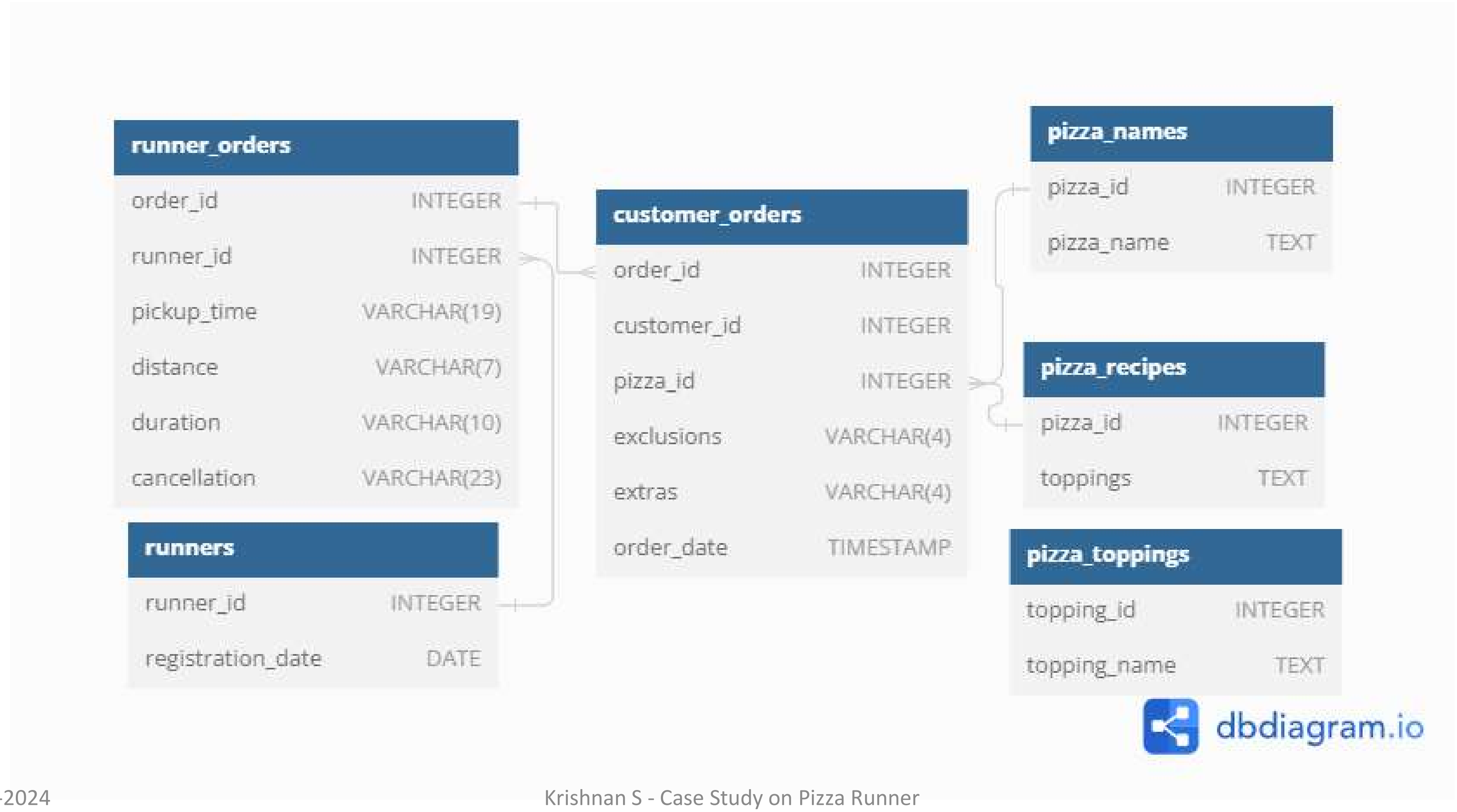
8 Week SQL Challenge

Case Study #2
Pizza Runner

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.

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

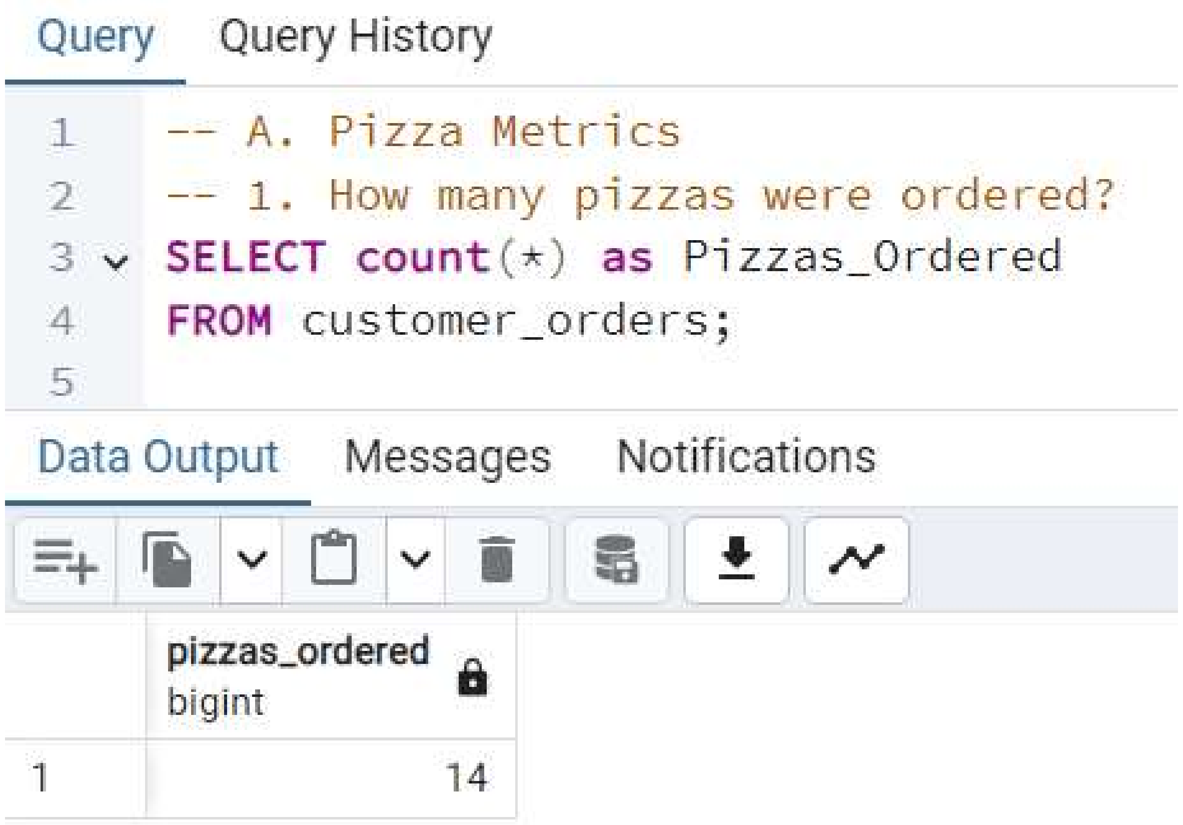
pizza_toppings	
topping_id	INTEGER
topping_name	TEXT

Case Study Questions

A. Pizza Metrics

1. How many pizzas were ordered?
2. How many unique customer orders were made?
3. How many successful orders were delivered by each runner?
4. How many of each type of pizza was delivered?
5. How many Vegetarian and Meatlovers were ordered by each customer?
6. What was the maximum number of pizzas delivered in a single order?
7. For each customer, how many delivered pizzas had at least 1 change and how many had no changes?
8. How many pizzas were delivered that had both exclusions and extras?
9. What was the total volume of pizzas ordered for each hour of the day?
10. What was the volume of orders for each day of the week?

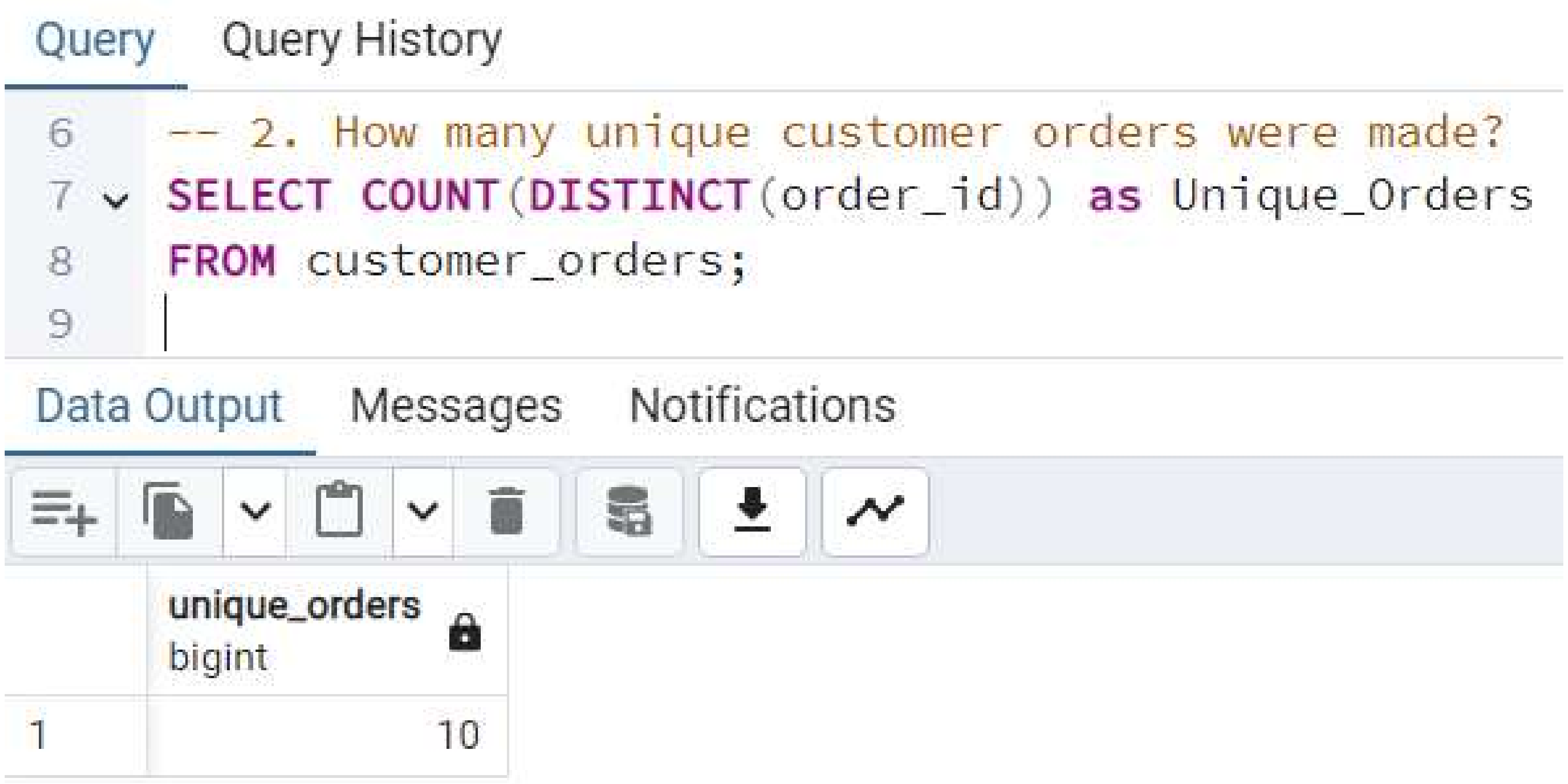
1. How many pizzas were ordered?



```sql
-- A. Pizza Metrics
-- 1. How many pizzas were ordered?
SELECT count(*) as Pizzas_Ordered
FROM customer_orders;
```

	pizzas_ordered bigint
1	14

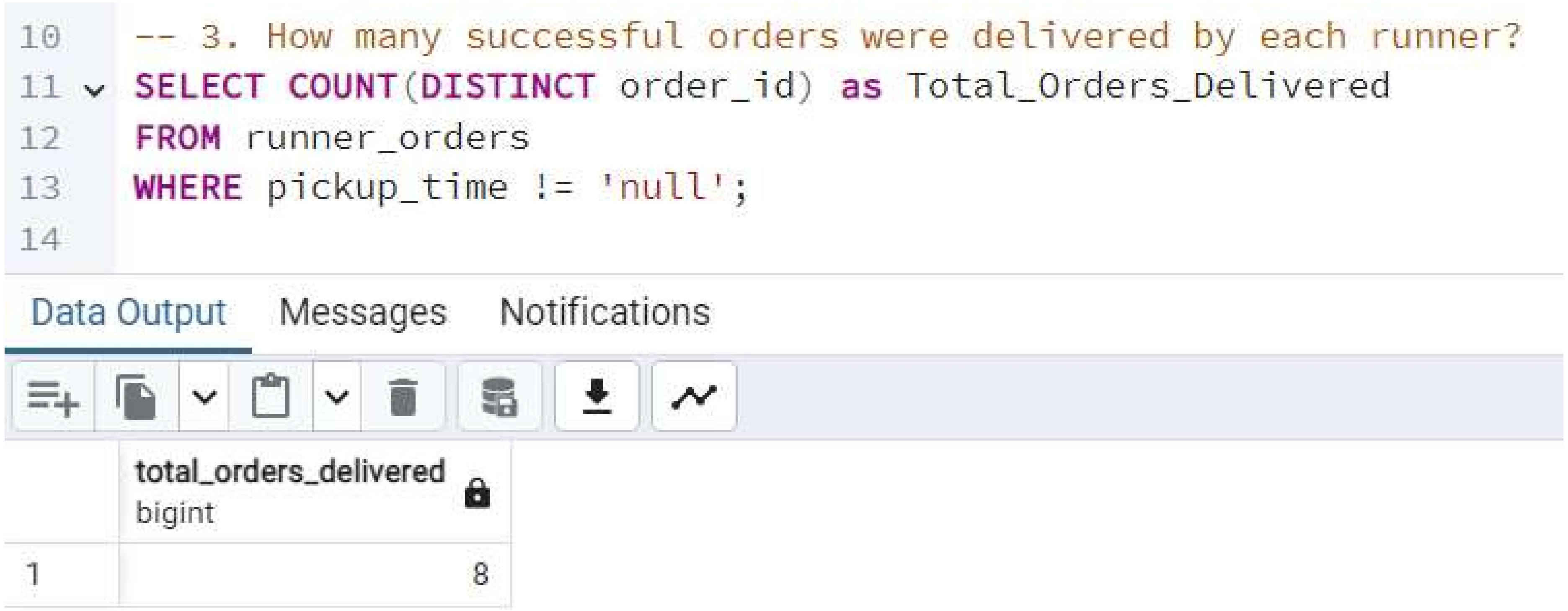
2. How many unique customer orders were made?



```sql
-- 2. How many unique customer orders were made?
SELECT COUNT(DISTINCT(order_id)) as Unique_Orders
FROM customer_orders;
```

```
  unique_orders
  bigint
1            10
```

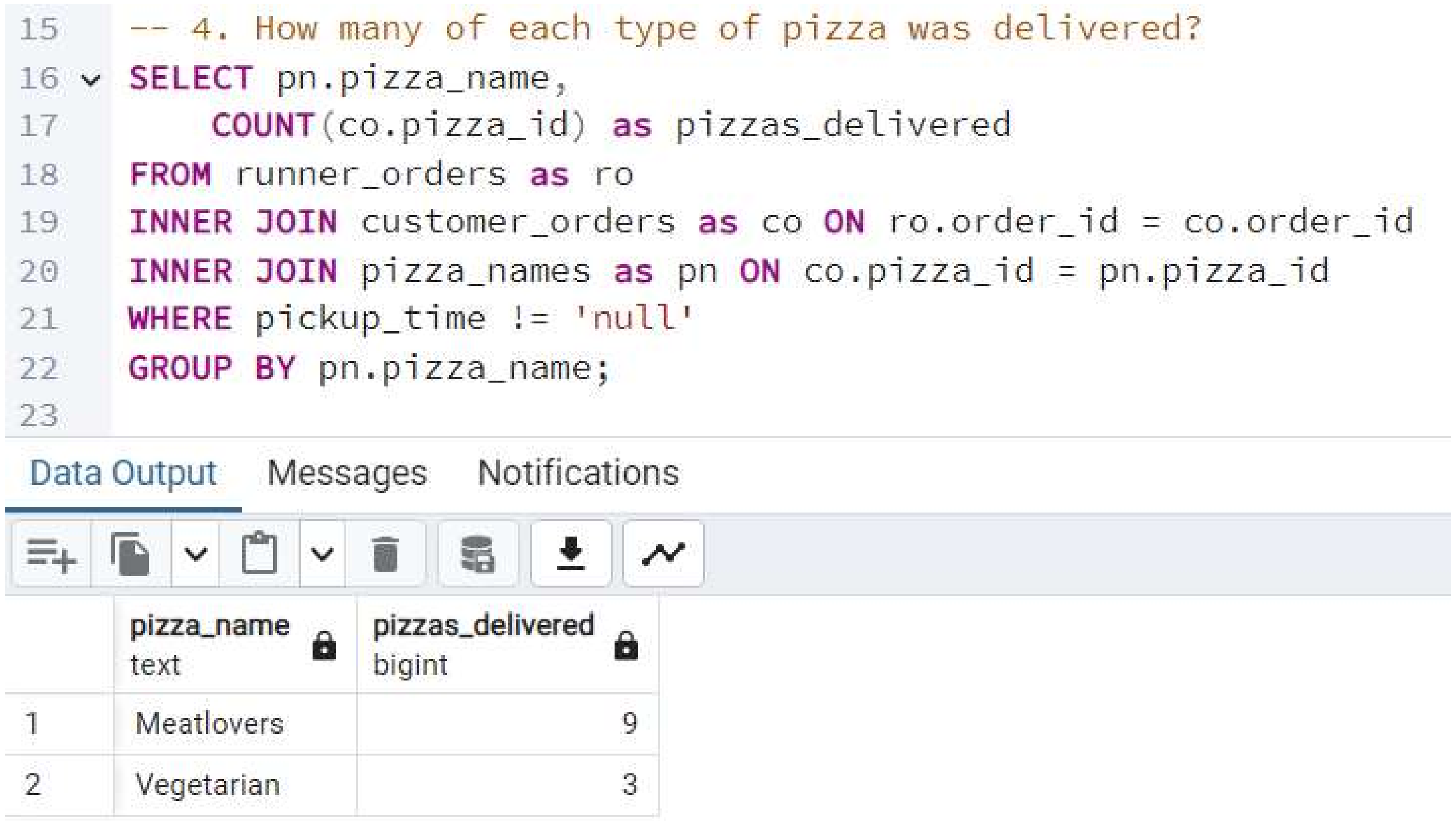
3. How many successful orders were delivered by each runner?



```sql
-- 3. How many successful orders were delivered by each runner?
SELECT COUNT(DISTINCT order_id) as Total_Orders_Delivered
FROM runner_orders
WHERE pickup_time != 'null';
```

	total_orders_delivered bigint
1	8

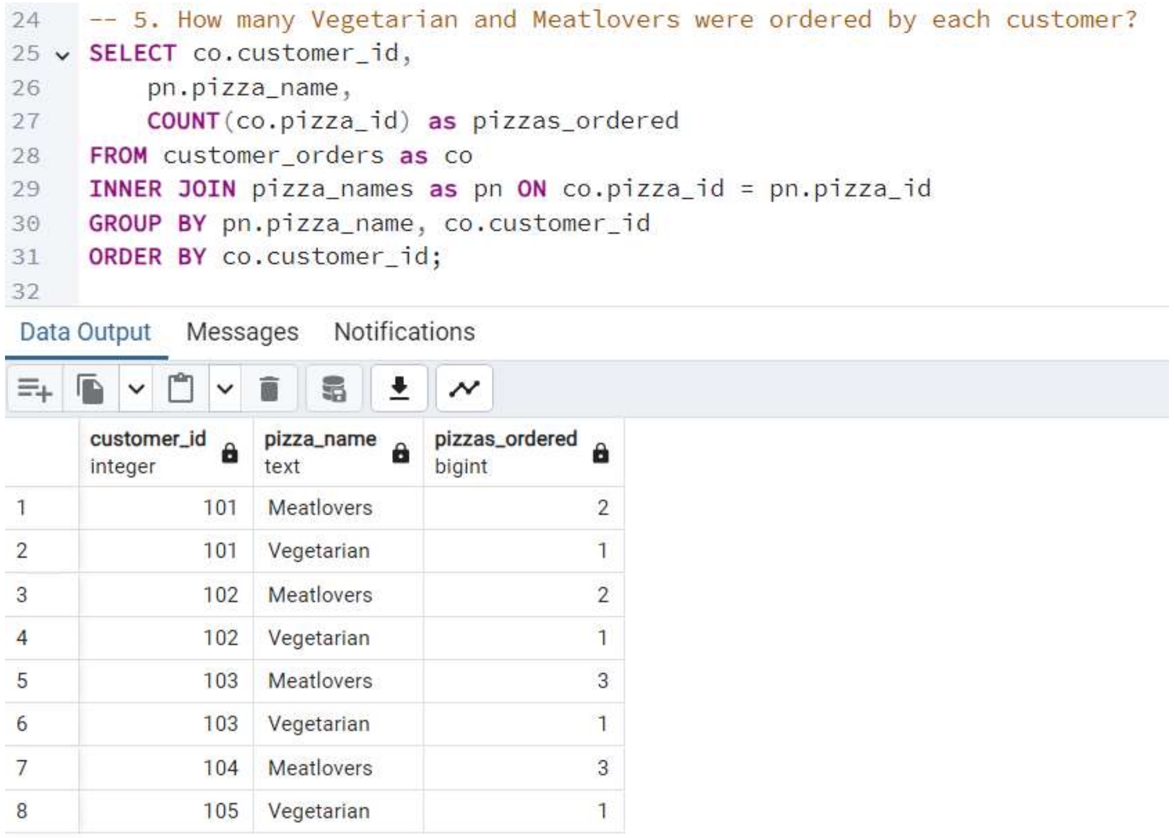
4. How many of each type of pizza was delivered?



```sql
-- 4. How many of each type of pizza was delivered?
SELECT pn.pizza_name,
    COUNT(co.pizza_id) as pizzas_delivered
FROM runner_orders as ro
INNER JOIN customer_orders as co ON ro.order_id = co.order_id
INNER JOIN pizza_names as pn ON co.pizza_id = pn.pizza_id
WHERE pickup_time != 'null'
GROUP BY pn.pizza_name;
```

	pizza_name text	pizzas_delivered bigint
1	Meatlovers	9
2	Vegetarian	3

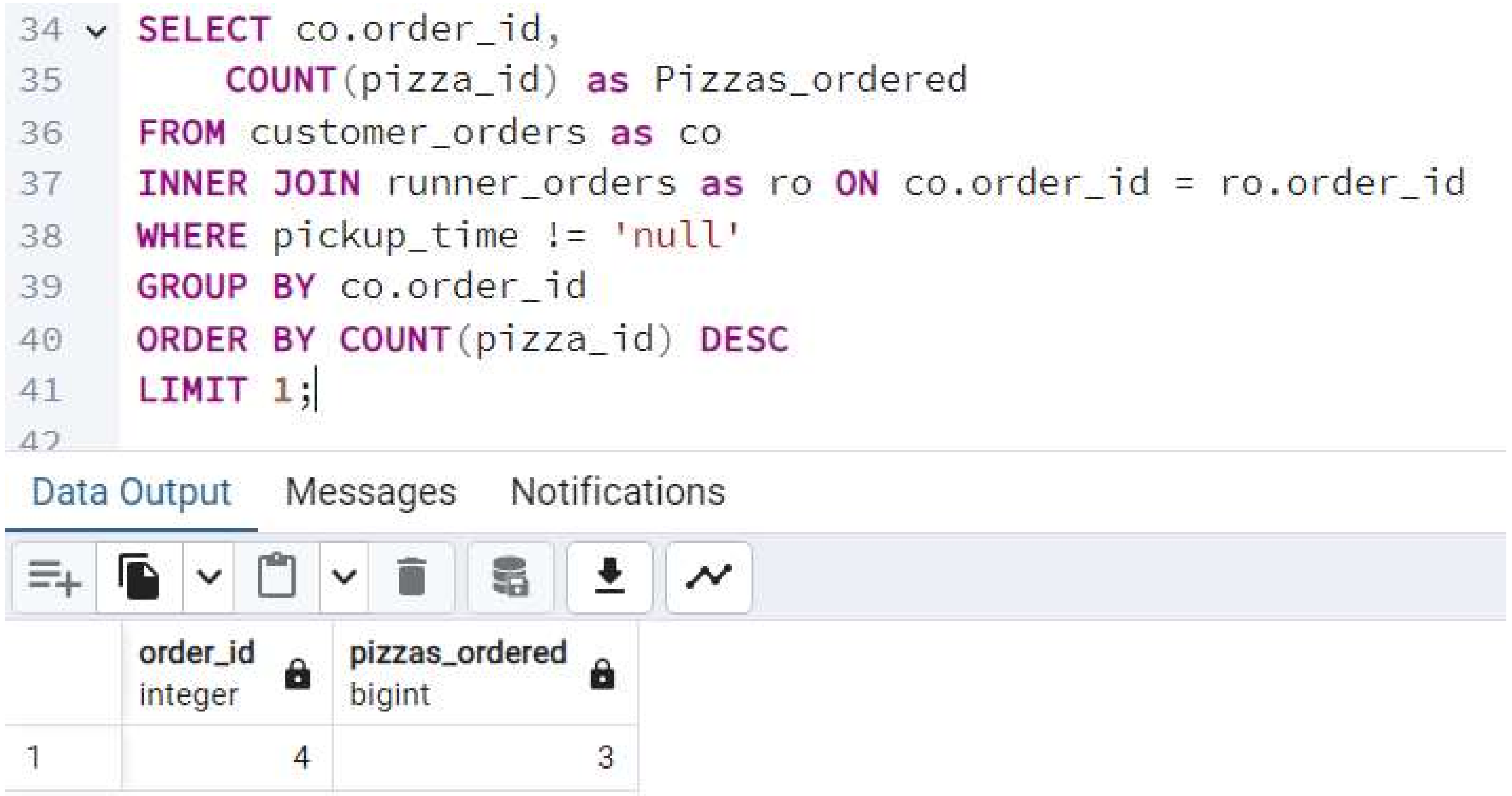
5. How many Vegetarian and Meat lovers were ordered by each customer?



```
-- 5. How many Vegetarian and Meatlovers were ordered by each customer?
SELECT co.customer_id,
    pn.pizza_name,
    COUNT(co.pizza_id) as pizzas_ordered
FROM customer_orders as co
INNER JOIN pizza_names as pn ON co.pizza_id = pn.pizza_id
GROUP BY pn.pizza_name, co.customer_id
ORDER BY co.customer_id;
```

	customer_id integer	pizza_name text	pizzas_ordered bigint
1	101	Meatlovers	2
2	101	Vegetarian	1
3	102	Meatlovers	2
4	102	Vegetarian	1
5	103	Meatlovers	3
6	103	Vegetarian	1
7	104	Meatlovers	3
8	105	Vegetarian	1

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

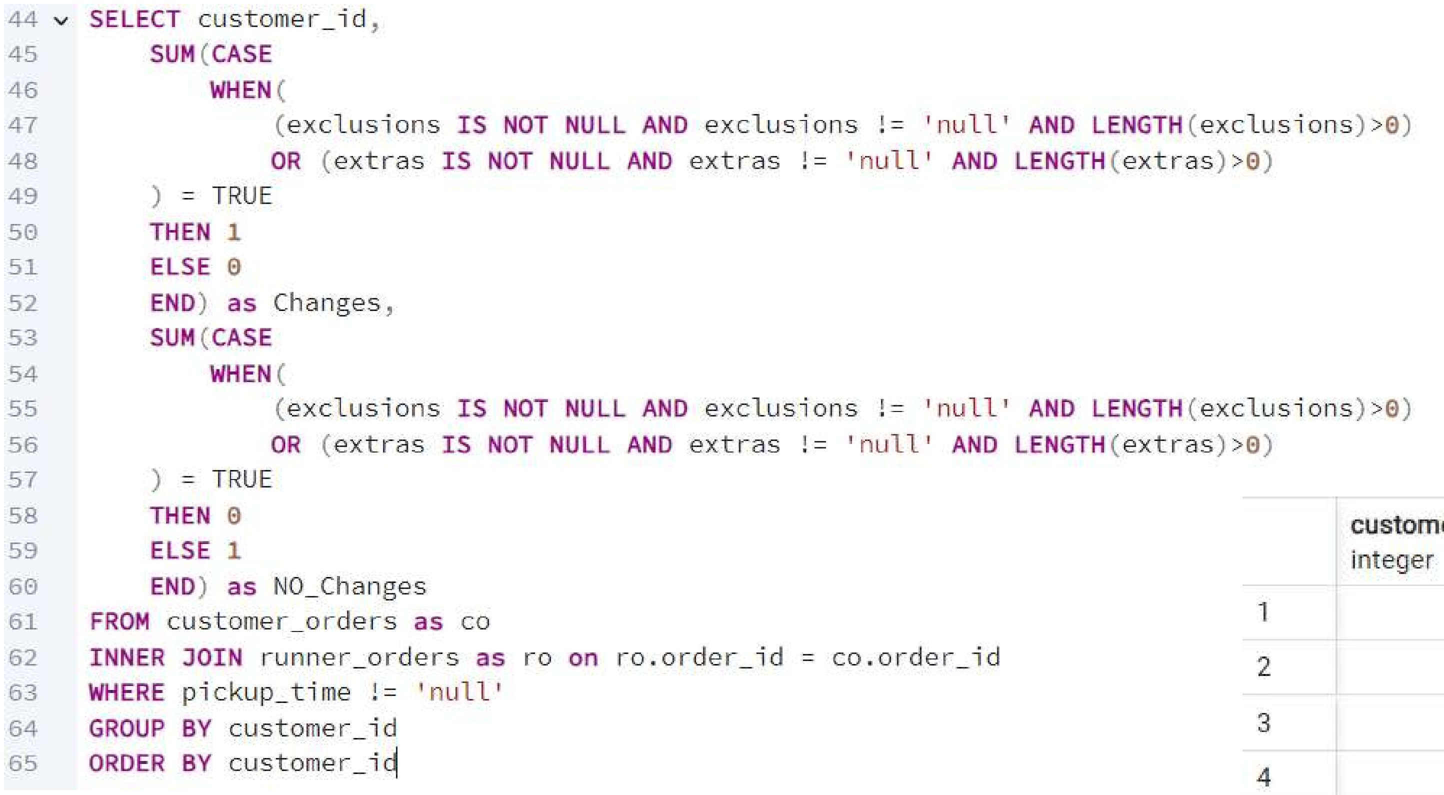


```
SELECT co.order_id,
    COUNT(pizza_id) as Pizzas_ordered
FROM customer_orders as co
INNER JOIN runner_orders as ro ON co.order_id = ro.order_id
WHERE pickup_time != 'null'
GROUP BY co.order_id
ORDER BY COUNT(pizza_id) DESC
LIMIT 1;
```

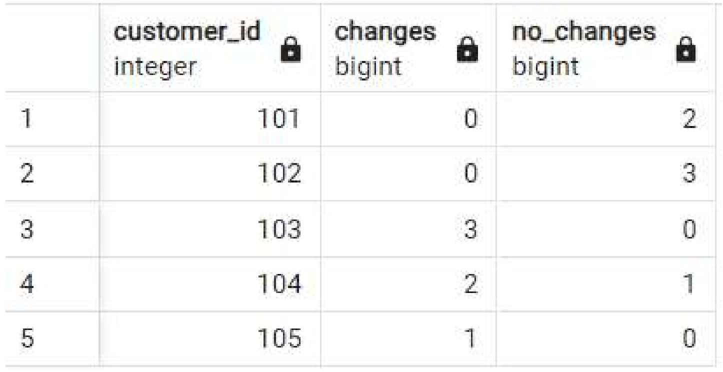
Data Output Messages Notifications

	order_id integer	pizzas_ordered bigint
1	4	3

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

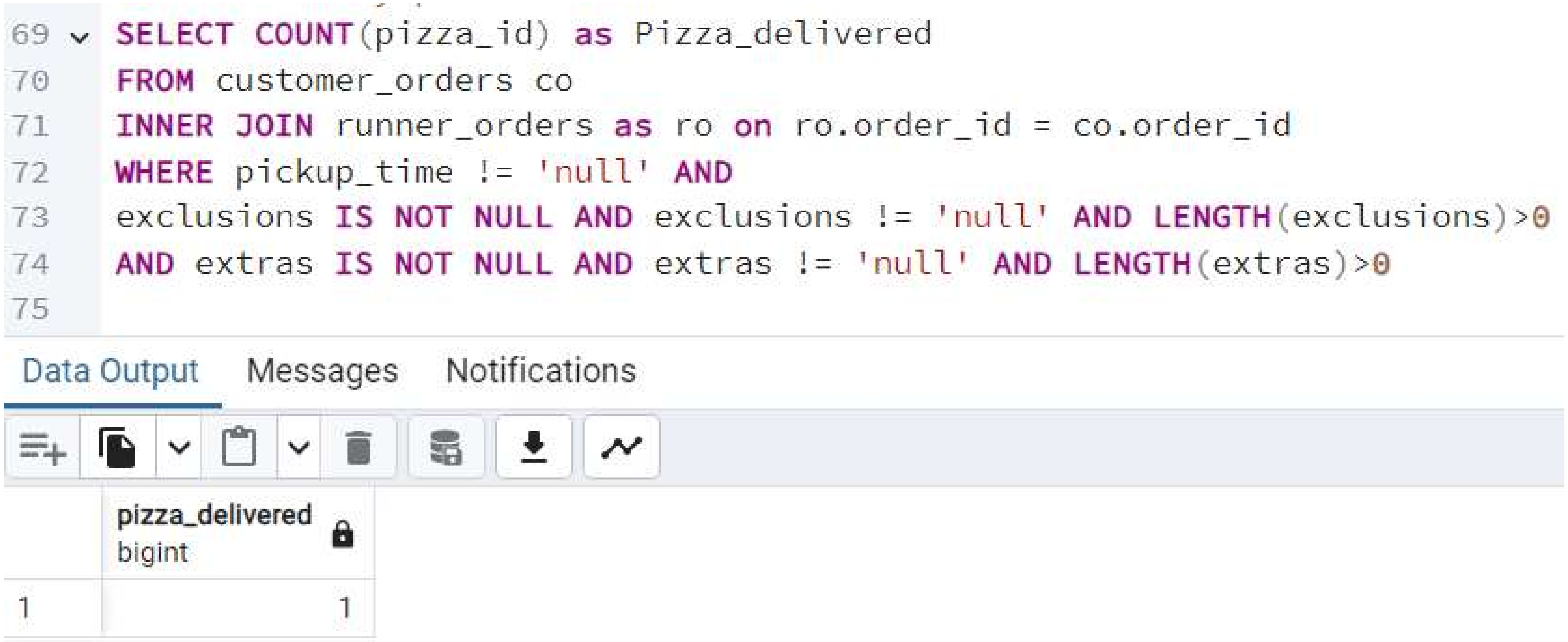


```sql
SELECT customer_id,
    SUM(CASE
        WHEN(
            (exclusions IS NOT NULL AND exclusions != 'null' AND LENGTH(exclusions)>0)
            OR (extras IS NOT NULL AND extras != 'null' AND LENGTH(extras)>0)
    ) = TRUE
    THEN 1
    ELSE 0
    END) as Changes,
    SUM(CASE
        WHEN(
            (exclusions IS NOT NULL AND exclusions != 'null' AND LENGTH(exclusions)>0)
            OR (extras IS NOT NULL AND extras != 'null' AND LENGTH(extras)>0)
    ) = TRUE
    THEN 0
    ELSE 1
    END) as NO_Changes
FROM customer_orders as co
INNER JOIN runner_orders as ro on ro.order_id = co.order_id
WHERE pickup_time != 'null'
GROUP BY customer_id
ORDER BY customer_id
```



	customer_id integer	changes bigint	no_changes bigint
1	101	0	2
2	102	0	3
3	103	3	0
4	104	2	1
5	105	1	0

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

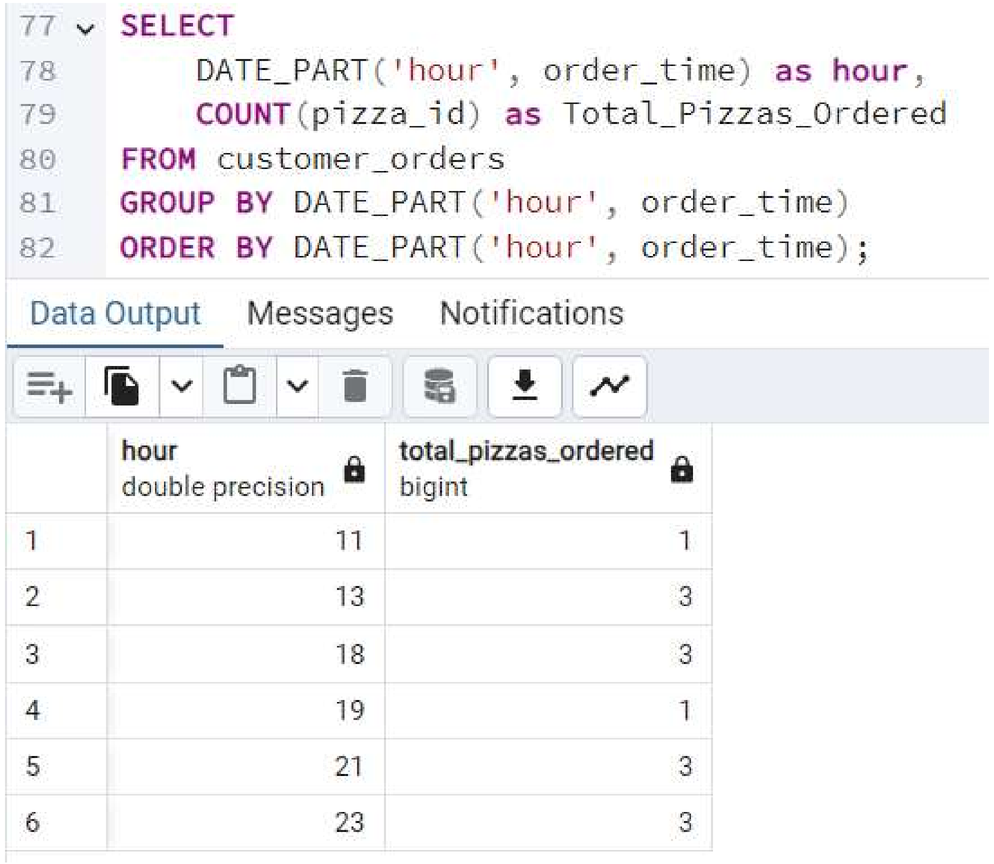


```
SELECT COUNT(pizza_id) as Pizza_delivered
FROM customer_orders co
INNER JOIN runner_orders as ro on ro.order_id = co.order_id
WHERE pickup_time != 'null' AND
exclusions IS NOT NULL AND exclusions != 'null' AND LENGTH(exclusions)>0
AND extras IS NOT NULL AND extras != 'null' AND LENGTH(extras)>0

```

	pizza_delivered bigint
1	1

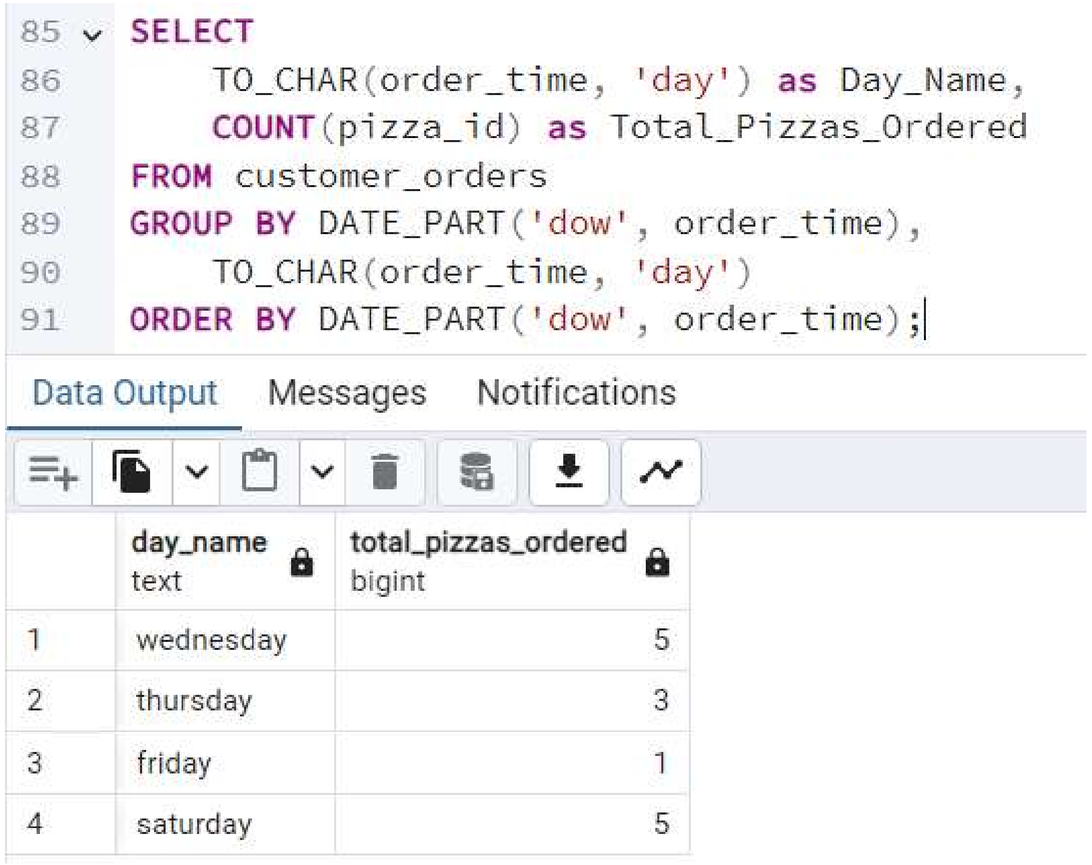
9. What was the total volume of pizzas ordered for each hour of the day?



```sql
SELECT
    DATE_PART('hour', order_time) as hour,
    COUNT(pizza_id) as Total_Pizzas_Ordered
FROM customer_orders
GROUP BY DATE_PART('hour', order_time)
ORDER BY DATE_PART('hour', order_time);
```

	hour double precision	total_pizzas_ordered bigint
1	11	1
2	13	3
3	18	3
4	19	1
5	21	3
6	23	3

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



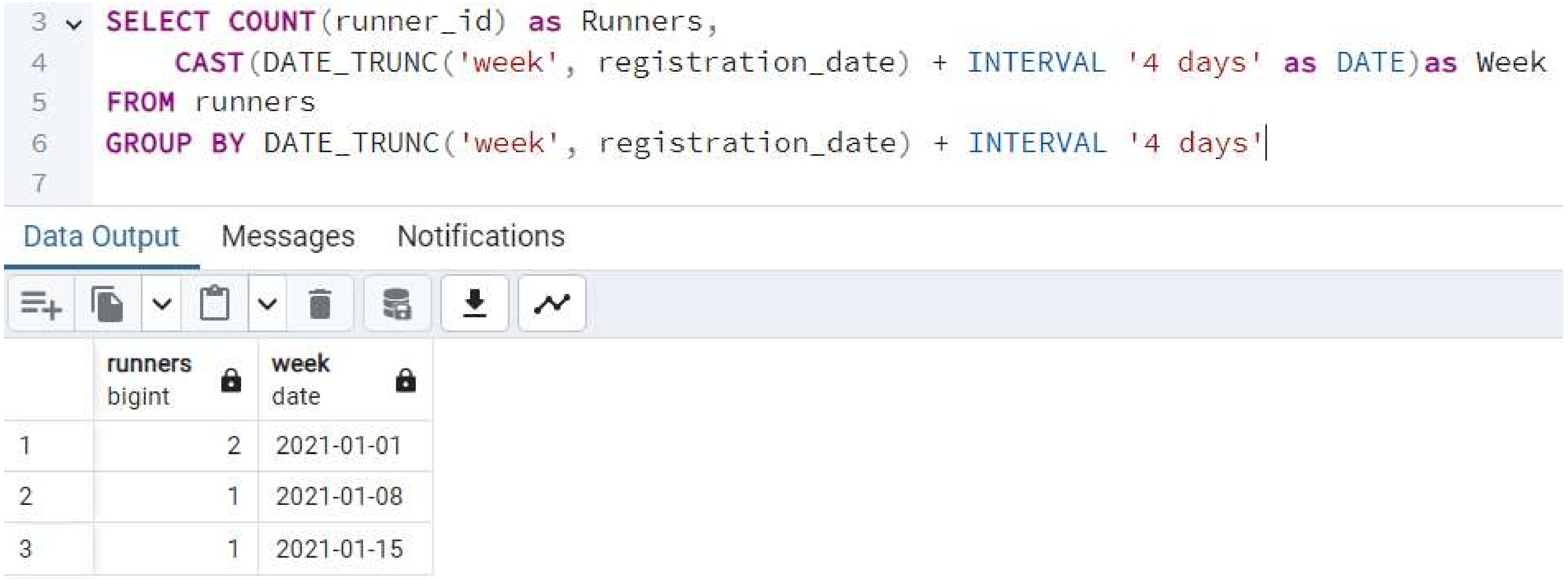
```
SELECT
    TO_CHAR(order_time, 'day') as Day_Name,
    COUNT(pizza_id) as Total_Pizzas_Ordered
FROM customer_orders
GROUP BY DATE_PART('dow', order_time),
    TO_CHAR(order_time, 'day')
ORDER BY DATE_PART('dow', order_time);
```

	day_name text	total_pizzas_ordered bigint
1	wednesday	5
2	thursday	3
3	friday	1
4	saturday	5

B. Runner and Customer Experience

1. How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)
2. What was the average time in minutes it took for each runner to arrive at the Pizza Runner HQ to pickup the order?
3. Is there any relationship between the number of pizzas and how long the order takes to prepare?
4. What was the average distance travelled for each customer?
5. What was the difference between the longest and shortest delivery times for all orders?
6. What was the average speed for each runner for each delivery and do you notice any trend for these values?
7. What is the successful delivery percentage for each runner?

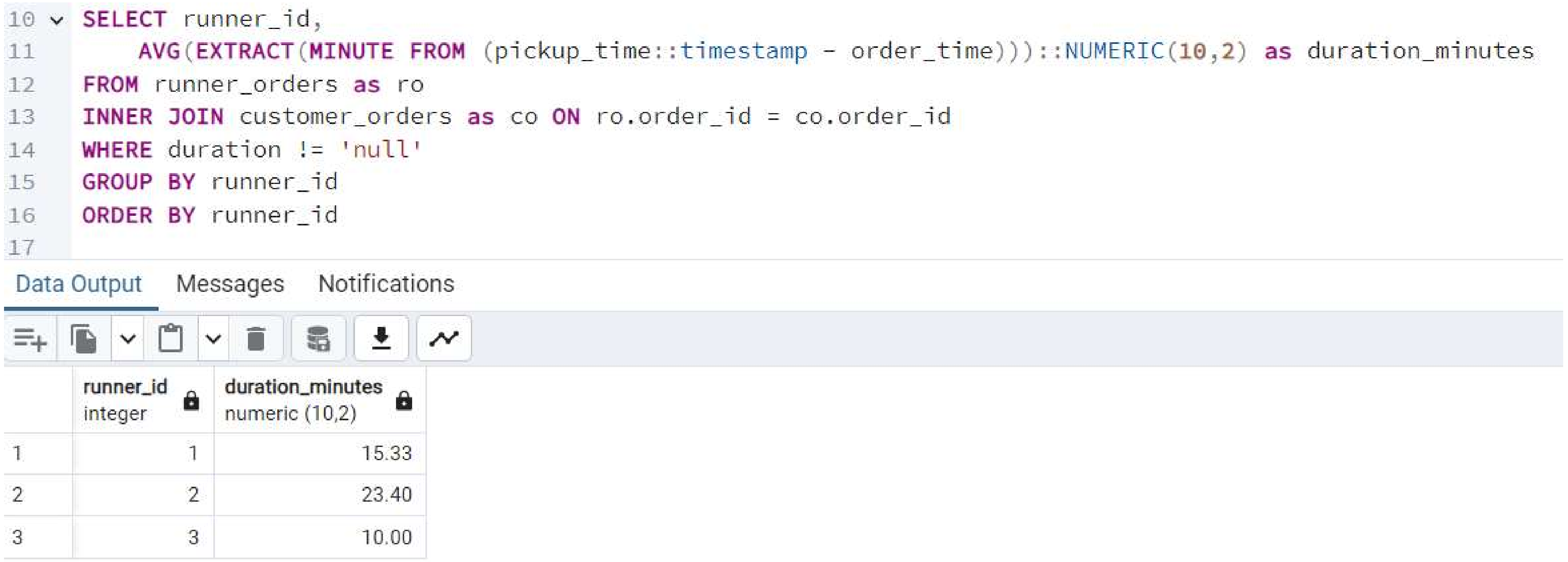
1. How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)



```sql
SELECT COUNT(runner_id) as Runners,
    CAST(DATE_TRUNC('week', registration_date) + INTERVAL '4 days' as DATE)as Week
FROM runners
GROUP BY DATE_TRUNC('week', registration_date) + INTERVAL '4 days'
```

	runners bigint	week date
1	2	2021-01-01
2	1	2021-01-08
3	1	2021-01-15

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

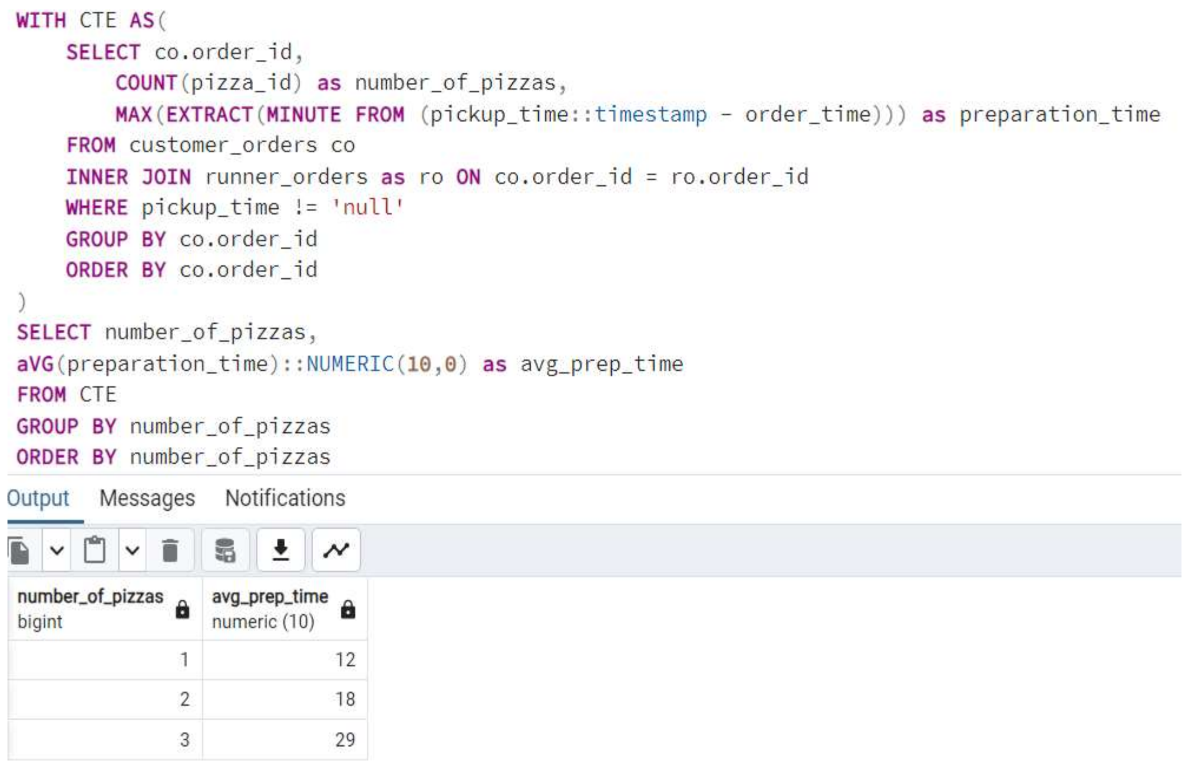


```sql
SELECT runner_id,
    AVG(EXTRACT(MINUTE FROM (pickup_time::timestamp - order_time)))::NUMERIC(10,2) as duration_minutes
FROM runner_orders as ro
INNER JOIN customer_orders as co ON ro.order_id = co.order_id
WHERE duration != 'null'
GROUP BY runner_id
ORDER BY runner_id
```

	runner_id integer	duration_minutes numeric (10,2)
1	1	15.33
2	2	23.40
3	3	10.00

3. Is there any relationship between the number of pizzas and how long the order takes to prepare?

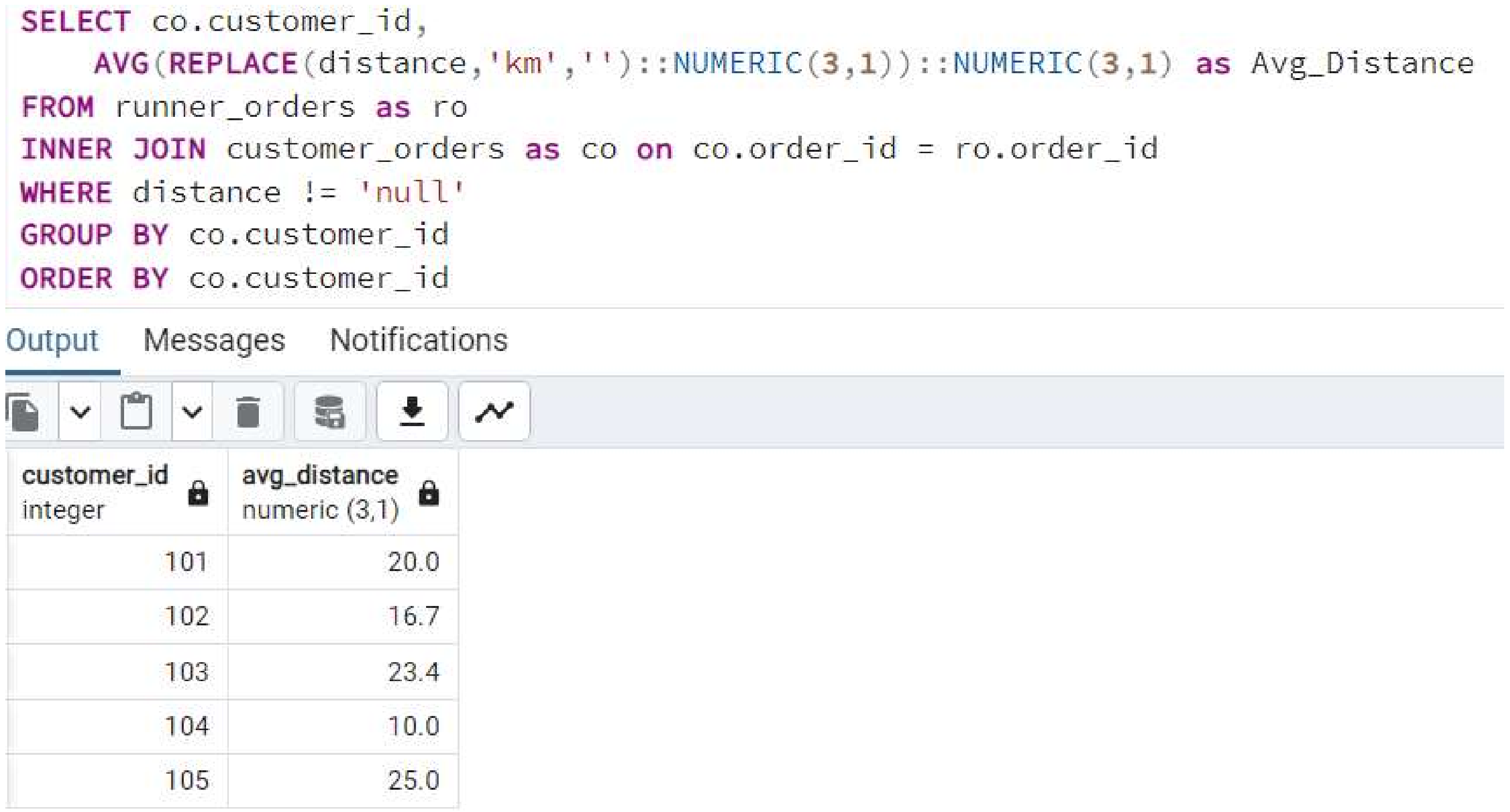
- Average Time taken to prepare
  - one pizza is 12minutes
  - Two pizza is 18minutes
  - Three pizza is 29minutes



```
WITH CTE AS(
    SELECT co.order_id,
        COUNT(pizza_id) as number_of_pizzas,
        MAX(EXTRACT(MINUTE FROM (pickup_time::timestamp - order_time))) as preparation_time
    FROM customer_orders co
    INNER JOIN runner_orders as ro ON co.order_id = ro.order_id
    WHERE pickup_time != 'null'
    GROUP BY co.order_id
    ORDER BY co.order_id
)
SELECT number_of_pizzas,
aVG(preparation_time)::NUMERIC(10,0) as avg_prep_time
FROM CTE
GROUP BY number_of_pizzas
ORDER BY number_of_pizzas
```

number_of_pizzas bigint	avg_prep_time numeric (10)
1	12
2	18
3	29

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

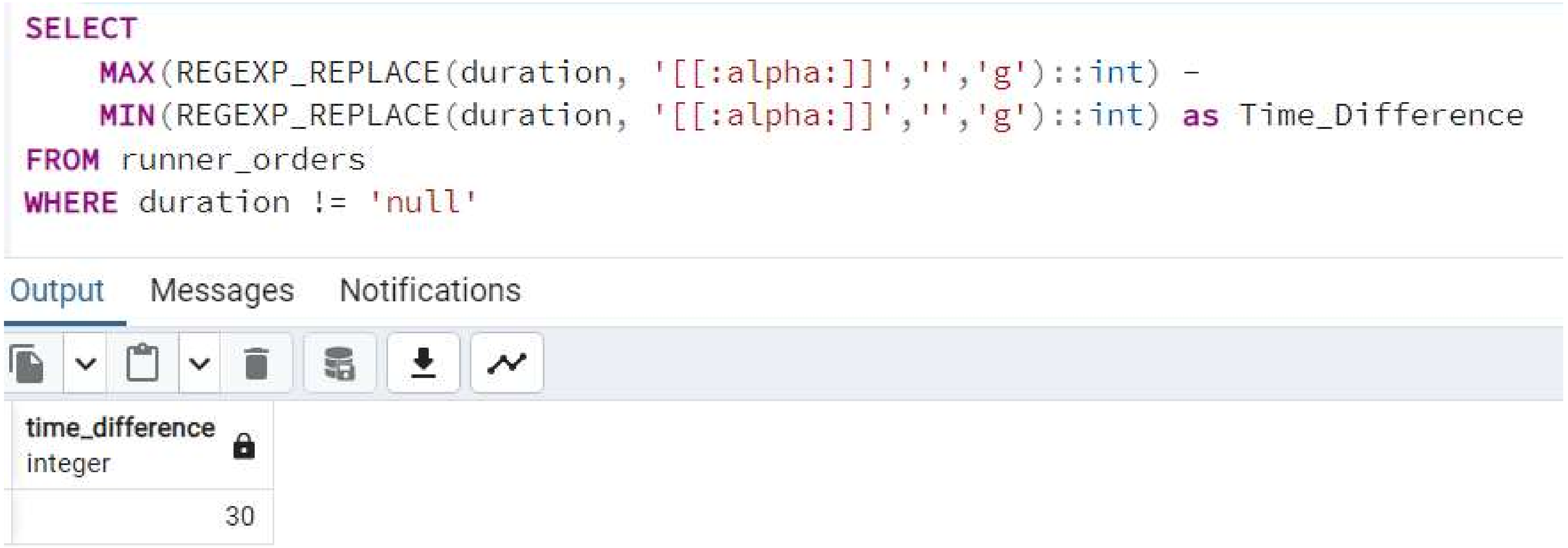


```
SELECT co.customer_id,
    AVG(REPLACE(distance,'km','')::NUMERIC(3,1))::NUMERIC(3,1) as Avg_Distance
FROM runner_orders as ro
INNER JOIN customer_orders as co on co.order_id = ro.order_id
WHERE distance != 'null'
GROUP BY co.customer_id
ORDER BY co.customer_id
```

Output Messages Notifications

customer_id integer	avg_distance numeric (3,1)
101	20.0
102	16.7
103	23.4
104	10.0
105	25.0

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



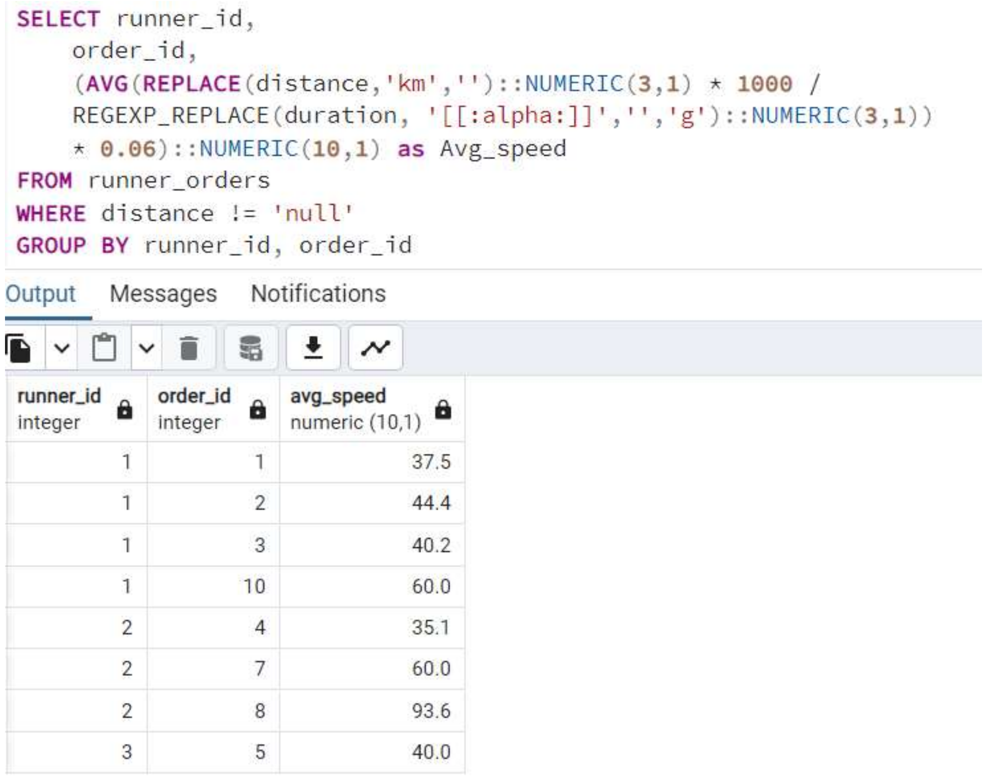
```
SELECT
    MAX(REGEXP_REPLACE(duration, '[[:alpha:]]','','g')::int) -
    MIN(REGEXP_REPLACE(duration, '[[:alpha:]]','','g')::int) as Time_Difference
FROM runner_orders
WHERE duration != 'null'
```

Output Messages Notifications

```
time_difference
integer
            30
```

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

- An observed trend is that as runners do more deliveries, they get faster.

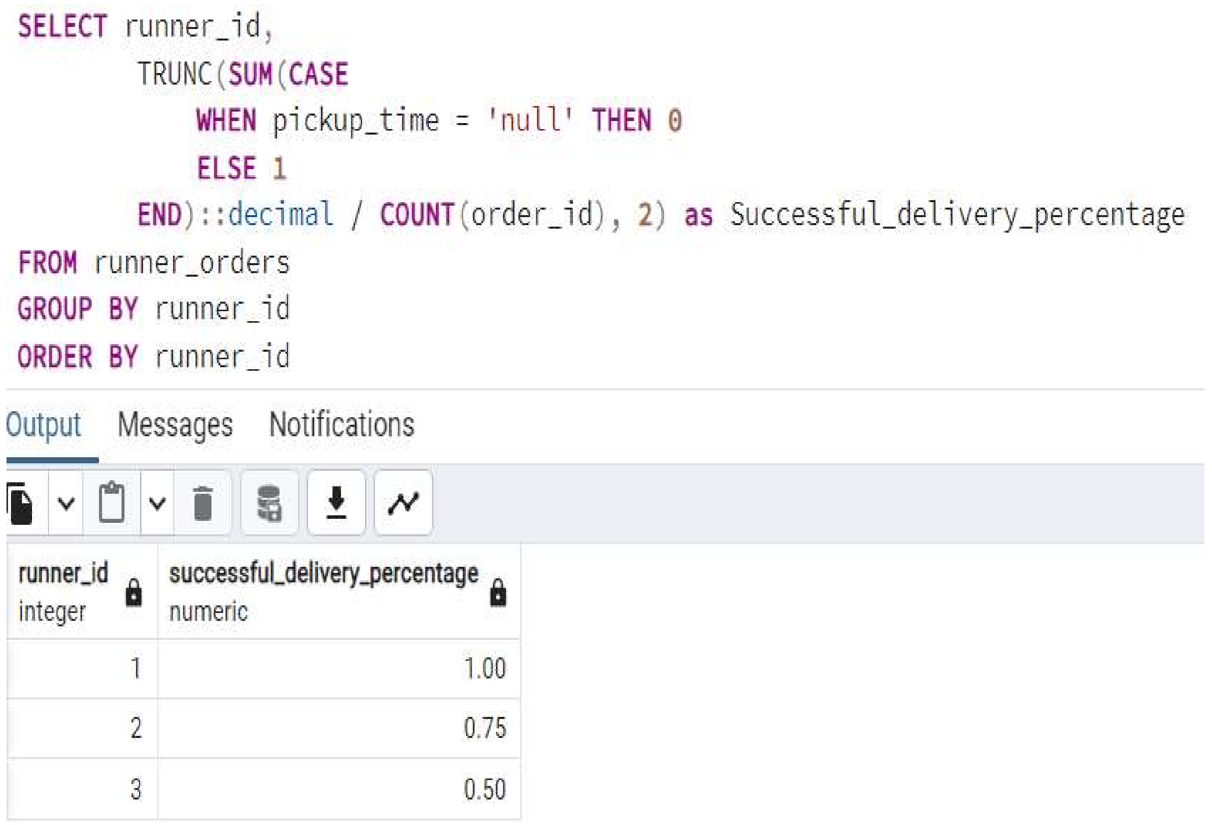


```
SELECT runner_id,
    order_id,
    (AVG(REPLACE(distance,'km','')::NUMERIC(3,1) * 1000 /
    REGEXP_REPLACE(duration, '[[:alpha:]]','','g')::NUMERIC(3,1))
    * 0.06)::NUMERIC(10,1) as Avg_speed
FROM runner_orders
WHERE distance != 'null'
GROUP BY runner_id, order_id
```

runner_id integer	order_id integer	avg_speed numeric (10,1)
1	1	37.5
1	2	44.4
1	3	40.2
1	10	60.0
2	4	35.1
2	7	60.0
2	8	93.6
3	5	40.0

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

1. Runner ID 1
   - Successful Delivery Percentage is 100%
2. Runner ID 2
   - Successful Delivery Percentage is 75%
3. Runner ID 3
   - Successful Delivery Percentage is 50%



```
SELECT runner_id,
       TRUNC(SUM(CASE
           WHEN pickup_time = 'null' THEN 0
           ELSE 1
       END)::decimal / COUNT(order_id), 2) as Successful_delivery_percentage
FROM runner_orders
GROUP BY runner_id
ORDER BY runner_id
```

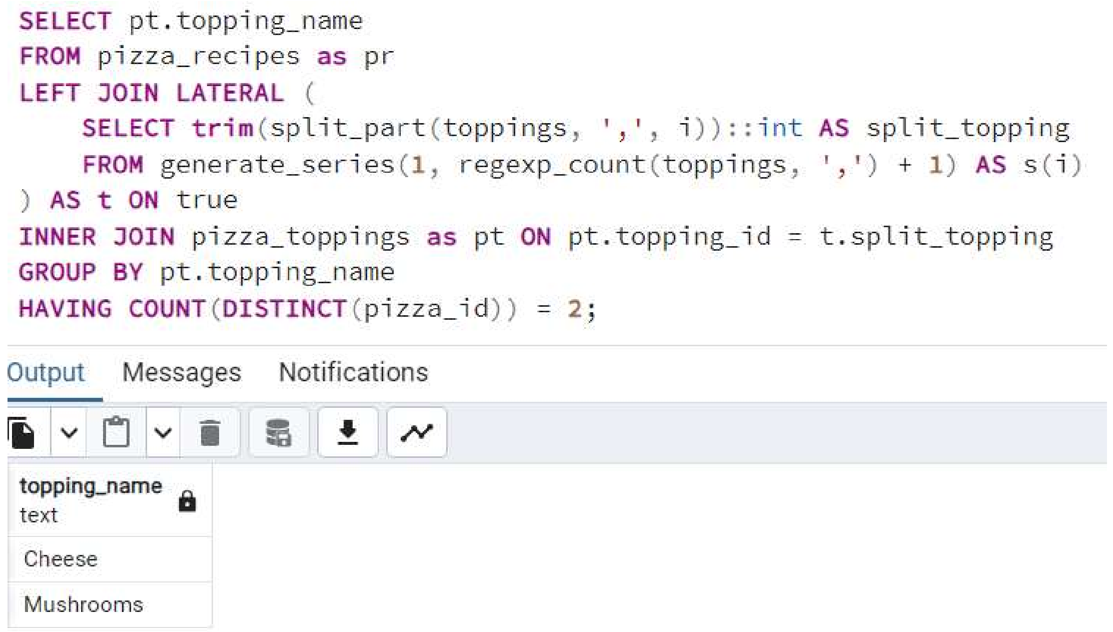
runner_id integer	successful_delivery_percentage numeric
1	1.00
2	0.75
3	0.50

C. Ingredient Optimisation

1. What are the standard ingredients for each pizza?
2. What was the most commonly added extra?
3. What was the most common exclusion?
4. Generate an order item for each record in the customers_orders table in the format of one of the following:
    - Meat Lovers
    - Meat Lovers - Exclude Beef
    - Meat Lovers - Extra Bacon
    - Meat Lovers - Exclude Cheese, Bacon - Extra Mushroom, Peppers
5. Generate an alphabetically ordered comma separated ingredient list for each pizza order from the customer_orders table and add a 2x in front of any relevant ingredients
    - For example: "Meat Lovers: 2xBacon, Beef, ... , Salami"
6. What is the total quantity of each ingredient used in all delivered pizzas sorted by most frequent first?

1. What are the standard ingredients for each pizza?

- The most standard ingredients used in both pizzas are
  1. Cheese
  2. Mushrooms

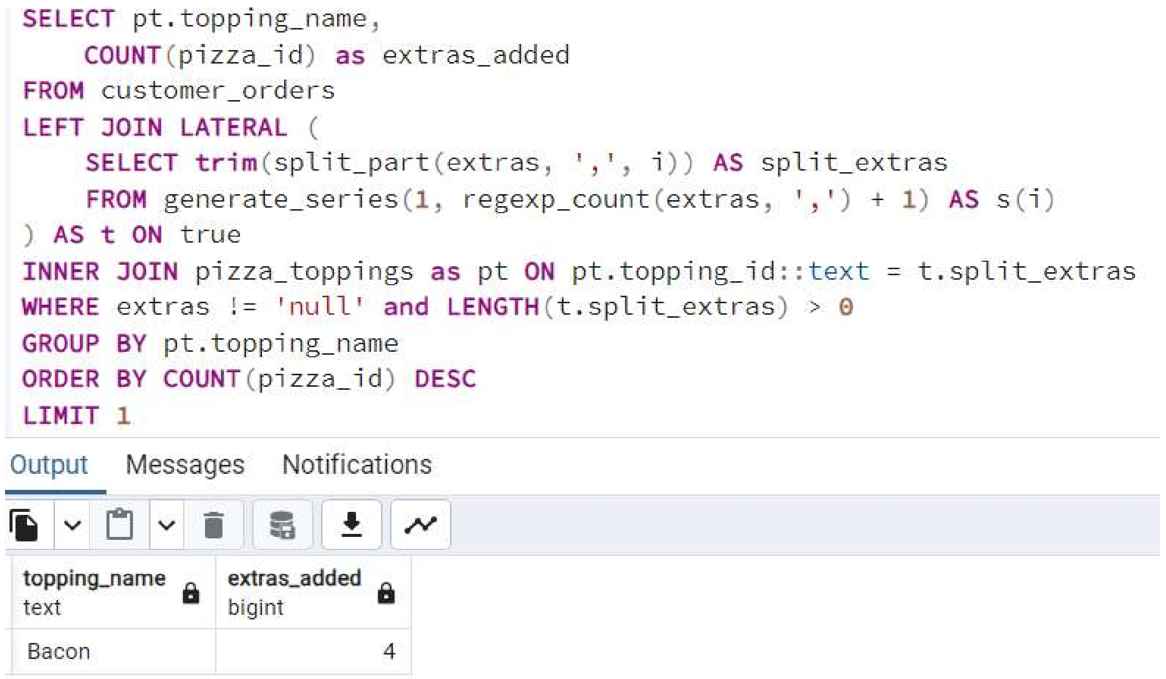


```
SELECT pt.topping_name
FROM pizza_recipes as pr
LEFT JOIN LATERAL (
    SELECT trim(split_part(toppings, ',', i))::int AS split_topping
    FROM generate_series(1, regexp_count(toppings, ',') + 1) AS s(i)
) AS t ON true
INNER JOIN pizza_toppings as pt ON pt.topping_id = t.split_topping
GROUP BY pt.topping_name
HAVING COUNT(DISTINCT(pizza_id)) = 2;
```

```
topping_name
text
Cheese
Mushrooms
```

2. What was the most commonly added extra?

- Bacon is the most commonly added extra.

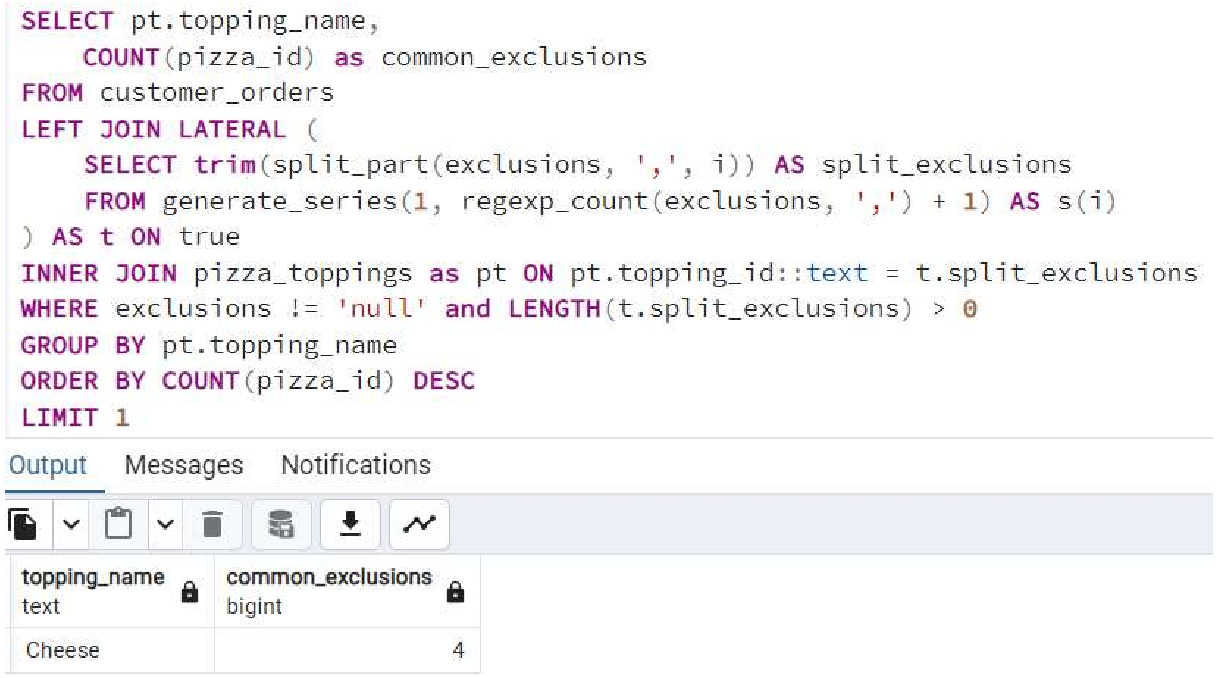


```
SELECT pt.topping_name,
    COUNT(pizza_id) as extras_added
FROM customer_orders
LEFT JOIN LATERAL (
    SELECT trim(split_part(extras, ',', i)) AS split_extras
    FROM generate_series(1, regexp_count(extras, ',') + 1) AS s(i)
) AS t ON true
INNER JOIN pizza_toppings as pt ON pt.topping_id::text = t.split_extras
WHERE extras != 'null' and LENGTH(t.split_extras) > 0
GROUP BY pt.topping_name
ORDER BY COUNT(pizza_id) DESC
LIMIT 1
```

topping_name text	extras_added bigint
Bacon	4

3. What was the most common exclusion?

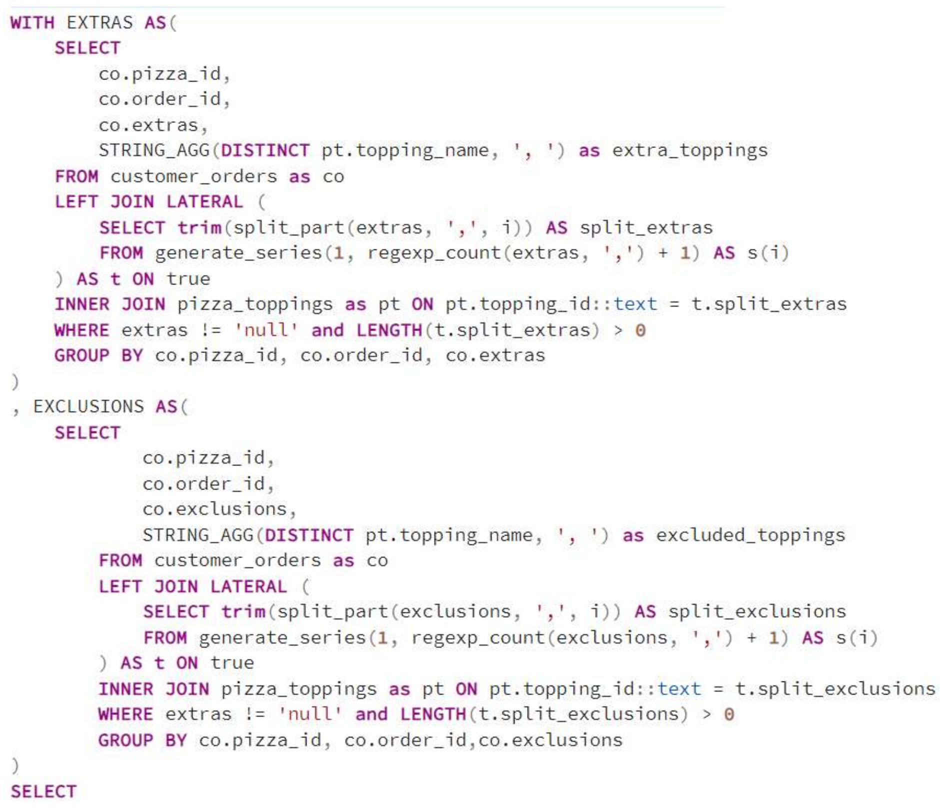
- Most Common Exclusion is Cheese



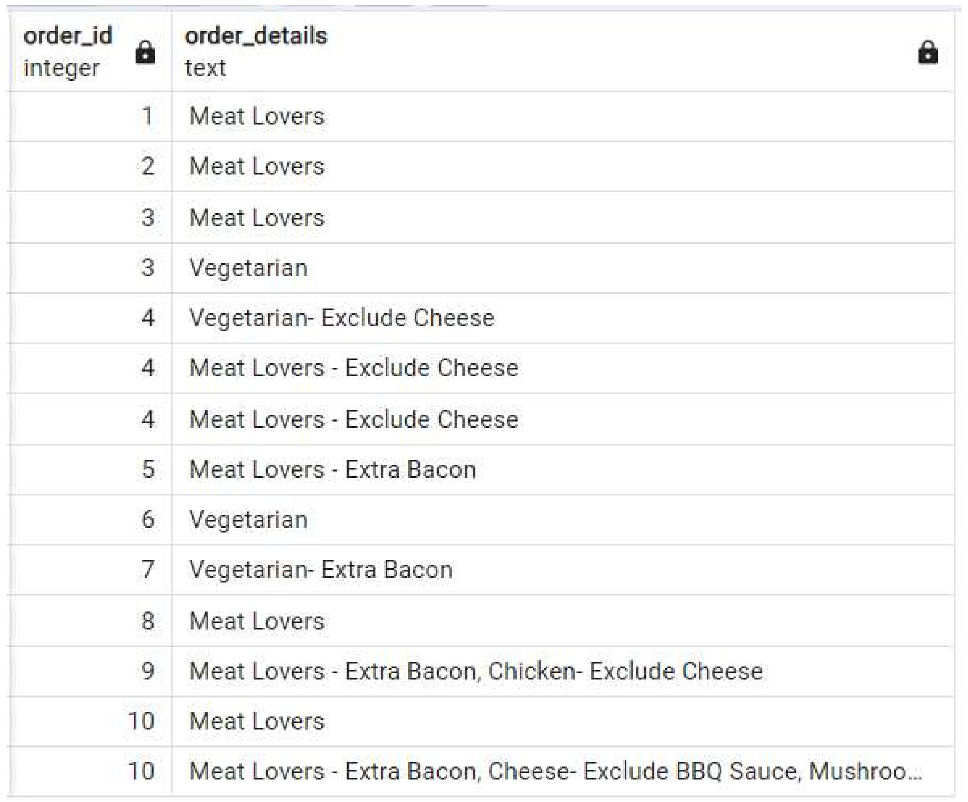
```
SELECT pt.topping_name,
    COUNT(pizza_id) as common_exclusions
FROM customer_orders
LEFT JOIN LATERAL (
    SELECT trim(split_part(exclusions, ',', i)) AS split_exclusions
    FROM generate_series(1, regexp_count(exclusions, ',') + 1) AS s(i)
) AS t ON true
INNER JOIN pizza_toppings as pt ON pt.topping_id::text = t.split_exclusions
WHERE exclusions != 'null' and LENGTH(t.split_exclusions) > 0
GROUP BY pt.topping_name
ORDER BY COUNT(pizza_id) DESC
LIMIT 1
```

topping_name text	common_exclusions bigint
Cheese	4

4. Generate an order item for each record in the customers_orders table in the format of one of the following:
Meat Lovers
Meat Lovers - Exclude Beef
Meat Lovers - Extra Bacon
Meat Lovers - Exclude Cheese, Bacon - Extra Mushroom, Peppers

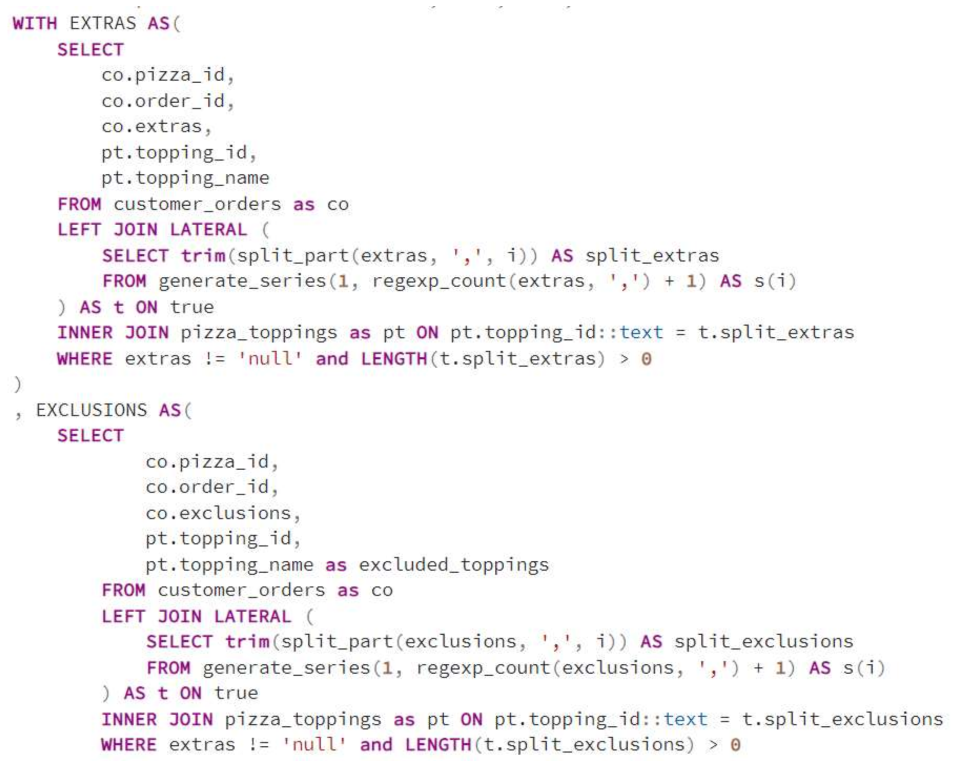


```
WITH EXTRAS AS(
    SELECT
        co.pizza_id,
        co.order_id,
        co.extras,
        STRING_AGG(DISTINCT pt.topping_name, ', ') as extra_toppings
    FROM customer_orders as co
    LEFT JOIN LATERAL (
        SELECT trim(split_part(extras, ',', i)) AS split_extras
        FROM generate_series(1, regexp_count(extras, ',') + 1) AS s(i)
    ) AS t ON true
    INNER JOIN pizza_toppings as pt ON pt.topping_id::text = t.split_extras
    WHERE extras != 'null' and LENGTH(t.split_extras) > 0
    GROUP BY co.pizza_id, co.order_id, co.extras
)
, EXCLUSIONS AS(
    SELECT
            co.pizza_id,
            co.order_id,
            co.exclusions,
            STRING_AGG(DISTINCT pt.topping_name, ', ') as excluded_toppings
        FROM customer_orders as co
        LEFT JOIN LATERAL (
            SELECT trim(split_part(exclusions, ',', i)) AS split_exclusions
            FROM generate_series(1, regexp_count(exclusions, ',') + 1) AS s(i)
        ) AS t ON true
        INNER JOIN pizza_toppings as pt ON pt.topping_id::text = t.split_exclusions
        WHERE extras != 'null' and LENGTH(t.split_exclusions) > 0
        GROUP BY co.pizza_id, co.order_id,co.exclusions
)
SELECT
```

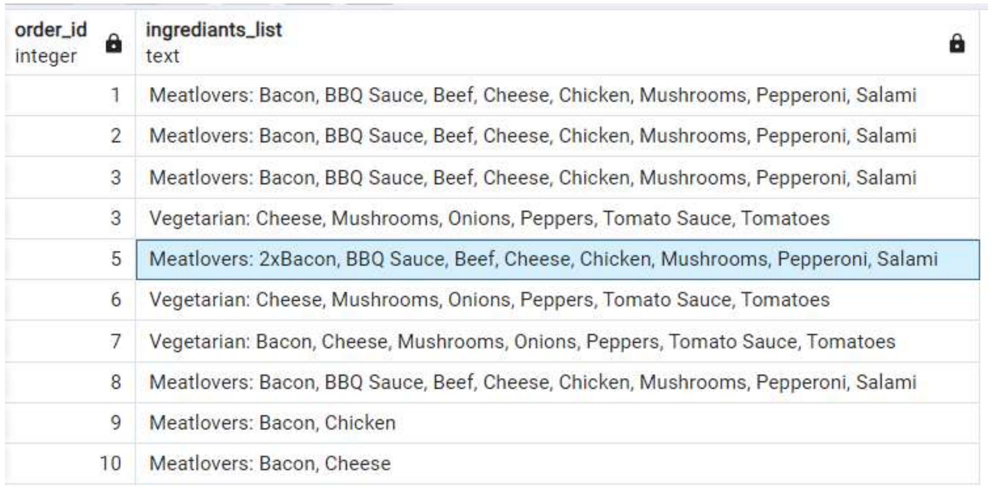


order_id integer	order_details text
1	Meat Lovers
2	Meat Lovers
3	Meat Lovers
3	Vegetarian
4	Vegetarian- Exclude Cheese
4	Meat Lovers - Exclude Cheese
4	Meat Lovers - Exclude Cheese
5	Meat Lovers - Extra Bacon
6	Vegetarian
7	Vegetarian- Extra Bacon
8	Meat Lovers
9	Meat Lovers - Extra Bacon, Chicken- Exclude Cheese
10	Meat Lovers
10	Meat Lovers - Extra Bacon, Cheese- Exclude BBQ Sauce, Mushroo...

5. Generate an alphabetically ordered comma separated ingredient list for each pizza order from the customer_orders table and add a 2x in front of any relevant ingredients
For example: "Meat Lovers: 2xBacon, Beef, ... , Salami"

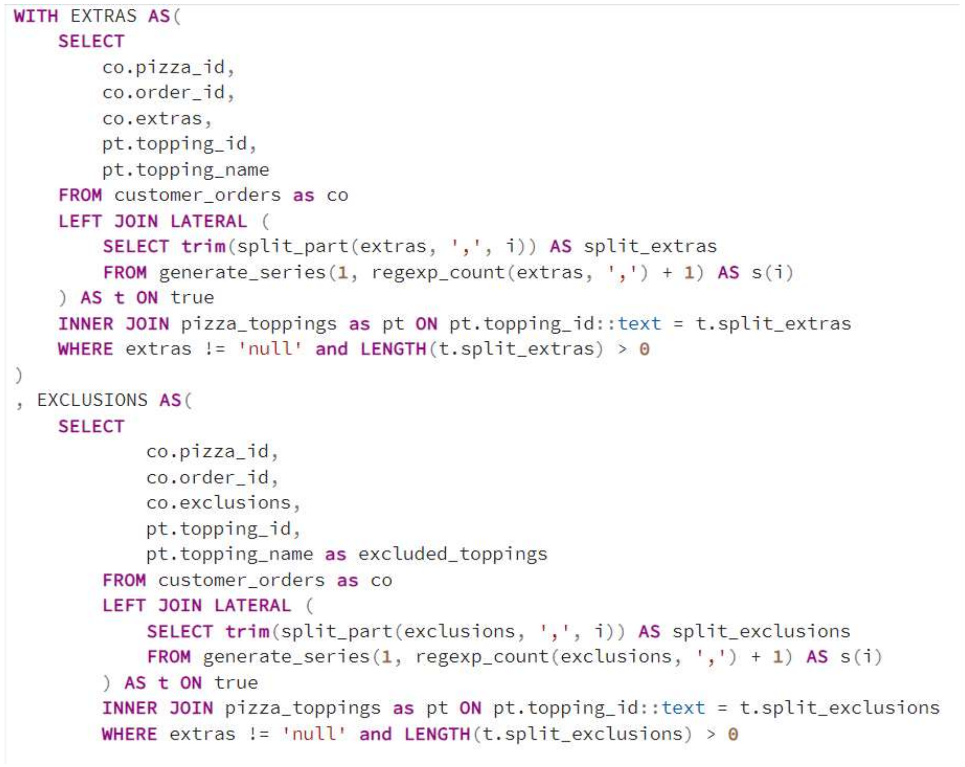


```
WITH EXTRAS AS(
    SELECT
        co.pizza_id,
        co.order_id,
        co.extras,
        pt.topping_id,
        pt.topping_name
    FROM customer_orders as co
    LEFT JOIN LATERAL (
        SELECT trim(split_part(extras, ',', i)) AS split_extras
        FROM generate_series(1, regexp_count(extras, ',') + 1) AS s(i)
    ) AS t ON true
    INNER JOIN pizza_toppings as pt ON pt.topping_id::text = t.split_extras
    WHERE extras != 'null' and LENGTH(t.split_extras) > 0
)
, EXCLUSIONS AS(
    SELECT
            co.pizza_id,
            co.order_id,
            co.exclusions,
            pt.topping_id,
            pt.topping_name as excluded_toppings
        FROM customer_orders as co
        LEFT JOIN LATERAL (
            SELECT trim(split_part(exclusions, ',', i)) AS split_exclusions
            FROM generate_series(1, regexp_count(exclusions, ',') + 1) AS s(i)
        ) AS t ON true
        INNER JOIN pizza_toppings as pt ON pt.topping_id::text = t.split_exclusions
        WHERE extras != 'null' and LENGTH(t.split_exclusions) > 0
```

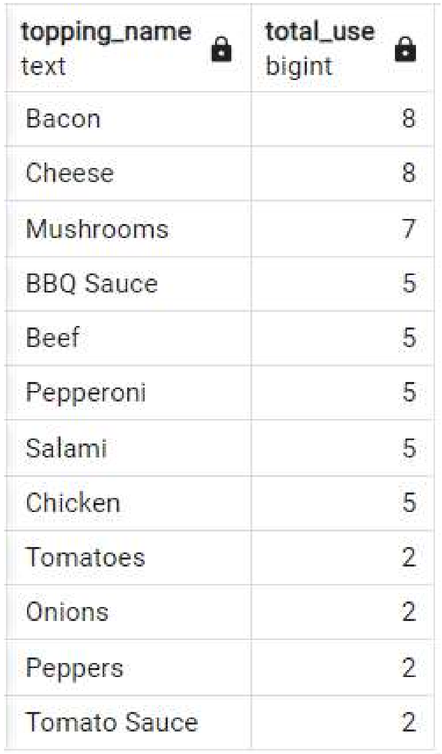


order_id integer	ingrediants_list text
1	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
2	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
3	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
3	Vegetarian: Cheese, Mushrooms, Onions, Peppers, Tomato Sauce, Tomatoes
5	Meatlovers: 2xBacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
6	Vegetarian: Cheese, Mushrooms, Onions, Peppers, Tomato Sauce, Tomatoes
7	Vegetarian: Bacon, Cheese, Mushrooms, Onions, Peppers, Tomato Sauce, Tomatoes
8	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
9	Meatlovers: Bacon, Chicken
10	Meatlovers: Bacon, Cheese

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



```
WITH EXTRAS AS(
    SELECT
        co.pizza_id,
        co.order_id,
        co.extras,
        pt.topping_id,
        pt.topping_name
    FROM customer_orders as co
    LEFT JOIN LATERAL (
        SELECT trim(split_part(extras, ',', i)) AS split_extras
        FROM generate_series(1, regexp_count(extras, ',') + 1) AS s(i)
    ) AS t ON true
    INNER JOIN pizza_toppings as pt ON pt.topping_id::text = t.split_extras
    WHERE extras != 'null' and LENGTH(t.split_extras) > 0
)
, EXCLUSIONS AS(
    SELECT
            co.pizza_id,
            co.order_id,
            co.exclusions,
            pt.topping_id,
            pt.topping_name as excluded_toppings
        FROM customer_orders as co
        LEFT JOIN LATERAL (
            SELECT trim(split_part(exclusions, ',', i)) AS split_exclusions
            FROM generate_series(1, regexp_count(exclusions, ',') + 1) AS s(i)
        ) AS t ON true
        INNER JOIN pizza_toppings as pt ON pt.topping_id::text = t.split_exclusions
        WHERE extras != 'null' and LENGTH(t.split_exclusions) > 0
```



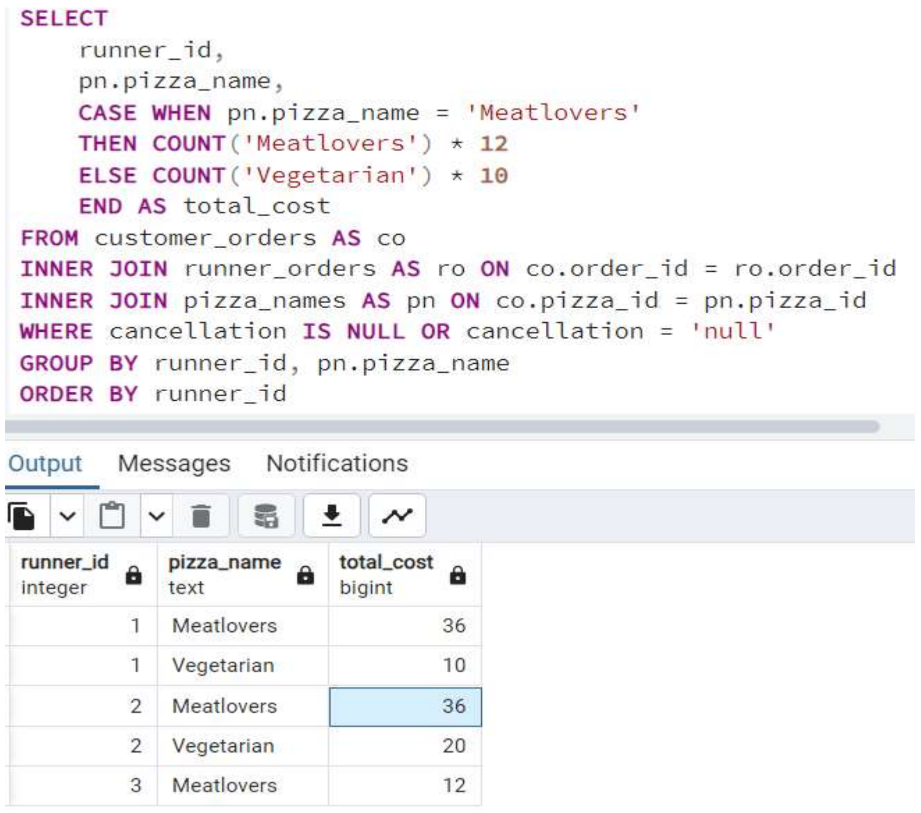
topping_name text	total_use bigint
Bacon	8
Cheese	8
Mushrooms	7
BBQ Sauce	5
Beef	5
Pepperoni	5
Salami	5
Chicken	5
Tomatoes	2
Onions	2
Peppers	2
Tomato Sauce	2

D. Pricing and Ratings

1. If a Meat Lovers pizza costs $12 and Vegetarian costs $10 and there were no charges for changes - how much money has Pizza Runner made so far if there are no delivery fees?
2. What if there was an additional $1 charge for any pizza extras?
   - Add cheese is $1 extra
3. The Pizza Runner team now wants to add an additional ratings system that allows customers to rate their runner, how would you design an additional table for this new dataset - generate a schema for this new table and insert your own data for ratings for each successful customer order between 1 to 5.
4. Using your newly generated table - can you join all of the information together to form a table which has the following information for successful deliveries?
   - customer_id
   - order_id
   - runner_id
   - rating
   - order_time
   - pickup_time
   - Time between order and pickup
   - Delivery duration
   - Average speed
   - Total number of pizzas
5. If a Meat Lovers pizza was $12 and Vegetarian $10 fixed prices with no cost for extras and each runner is paid $0.30 per kilometre traveled - how much money does Pizza Runner have left over after these deliveries?

1. If a Meat Lovers pizza costs $12 and Vegetarian costs $10 and there were no charges for changes how much money has Pizza Runner made so far if there are no delivery fees?

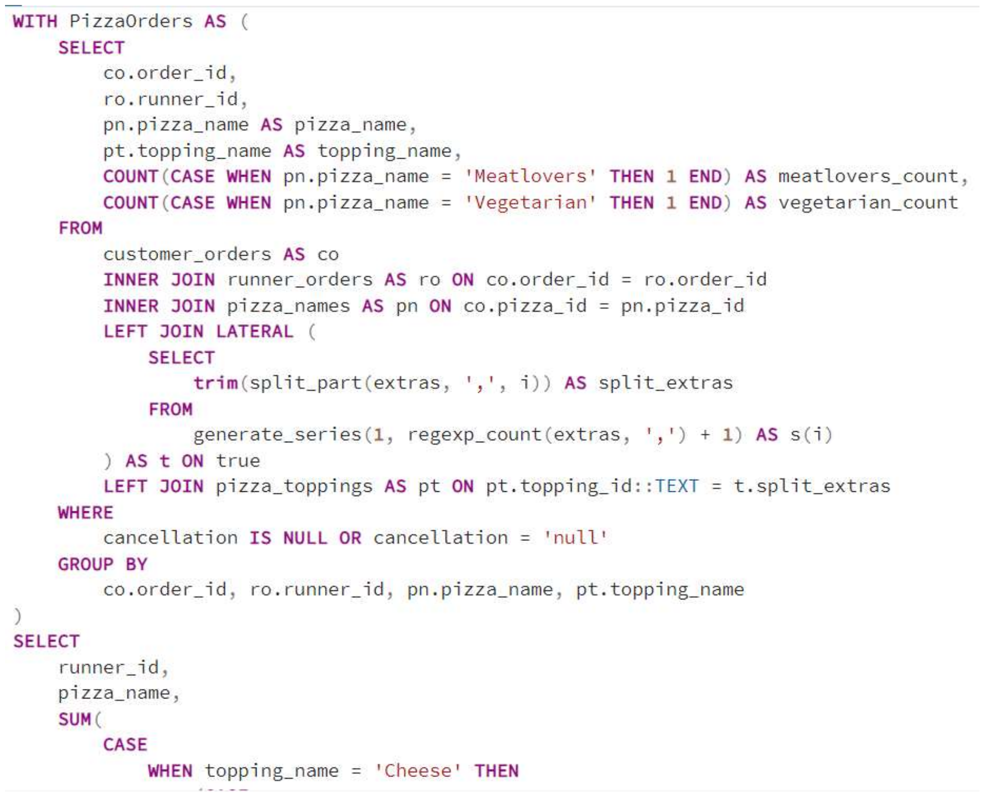
- Runner id 1 totally made $46
- Runner id 2 totally made $56
- Runner id 3 totally made $12



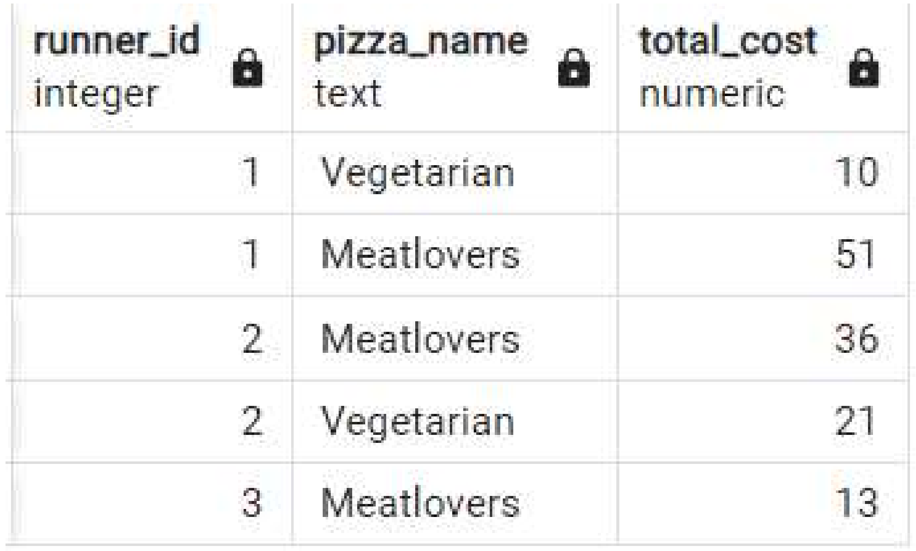
```sql
SELECT
    runner_id,
    pn.pizza_name,
    CASE WHEN pn.pizza_name = 'Meatlovers'
    THEN COUNT('Meatlovers') * 12
    ELSE COUNT('Vegetarian') * 10
    END AS total_cost
FROM customer_orders AS co
INNER JOIN runner_orders AS ro ON co.order_id = ro.order_id
INNER JOIN pizza_names AS pn ON co.pizza_id = pn.pizza_id
WHERE cancellation IS NULL OR cancellation = 'null'
GROUP BY runner_id, pn.pizza_name
ORDER BY runner_id
```

runner_id integer	pizza_name text	total_cost bigint
1	Meatlovers	36
1	Vegetarian	10
2	Meatlovers	36
2	Vegetarian	20
3	Meatlovers	12

2. What if there was an additional $1 charge for any pizza extras? Add cheese is $1 extra

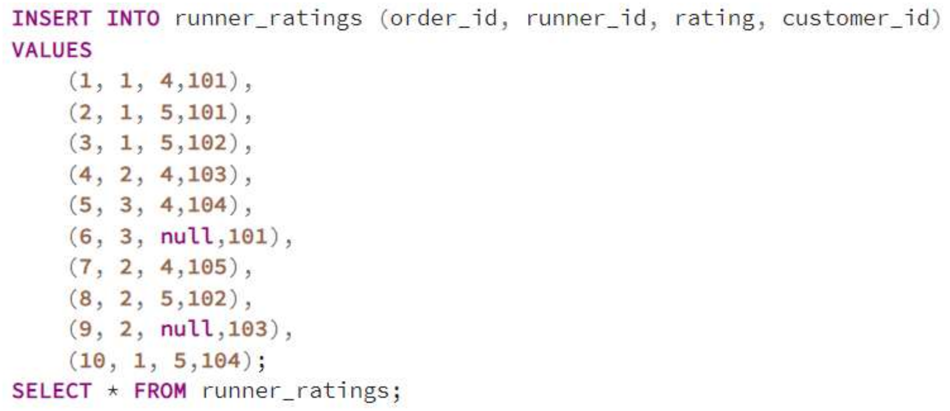


```
WITH PizzaOrders AS (
    SELECT
        co.order_id,
        ro.runner_id,
        pn.pizza_name AS pizza_name,
        pt.topping_name AS topping_name,
        COUNT(CASE WHEN pn.pizza_name = 'Meatlovers' THEN 1 END) AS meatlovers_count,
        COUNT(CASE WHEN pn.pizza_name = 'Vegetarian' THEN 1 END) AS vegetarian_count
    FROM
        customer_orders AS co
        INNER JOIN runner_orders AS ro ON co.order_id = ro.order_id
        INNER JOIN pizza_names AS pn ON co.pizza_id = pn.pizza_id
        LEFT JOIN LATERAL (
            SELECT
                trim(split_part(extras, ',', i)) AS split_extras
            FROM
                generate_series(1, regexp_count(extras, ',') + 1) AS s(i)
        ) AS t ON true
        LEFT JOIN pizza_toppings AS pt ON pt.topping_id::TEXT = t.split_extras
    WHERE
        cancellation IS NULL OR cancellation = 'null'
    GROUP BY
        co.order_id, ro.runner_id, pn.pizza_name, pt.topping_name
)
SELECT
    runner_id,
    pizza_name,
    SUM(
        CASE
            WHEN topping_name = 'Cheese' THEN
```

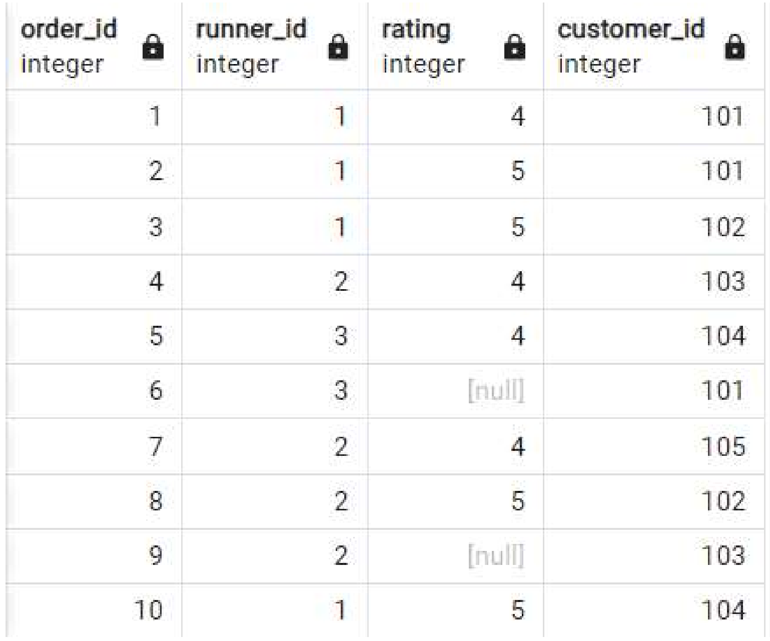


runner_id integer	pizza_name text	total_cost numeric
1	Vegetarian	10
1	Meatlovers	51
2	Meatlovers	36
2	Vegetarian	21
3	Meatlovers	13

3. The Pizza Runner team now wants to add an additional ratings system that allows customers to rate their runner, how would you design an additional table for this new dataset generate a schema for this new table and insert your own data for ratings for each successful customer order between 1 to 5.

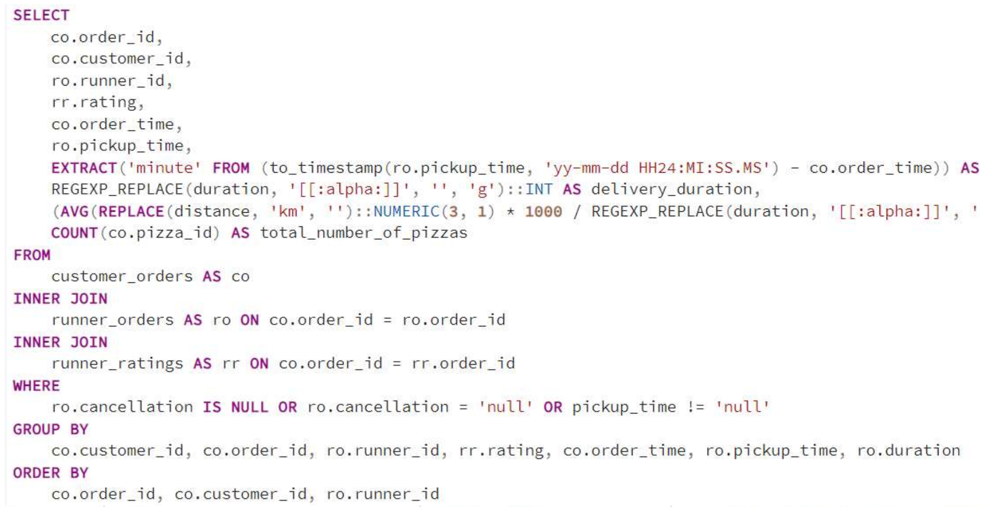


```
INSERT INTO runner_ratings (order_id, runner_id, rating, customer_id)
VALUES
    (1, 1, 4,101),
    (2, 1, 5,101),
    (3, 1, 5,102),
    (4, 2, 4,103),
    (5, 3, 4,104),
    (6, 3, null,101),
    (7, 2, 4,105),
    (8, 2, 5,102),
    (9, 2, null,103),
    (10, 1, 5,104);
SELECT * FROM runner_ratings;
```

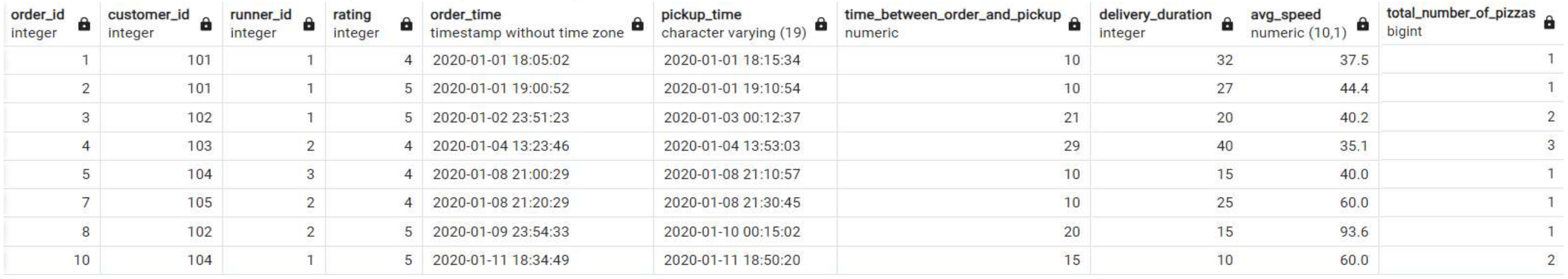


order_id integer	runner_id integer	rating integer	customer_id integer
1	1	4	101
2	1	5	101
3	1	5	102
4	2	4	103
5	3	4	104
6	3	[null]	101
7	2	4	105
8	2	5	102
9	2	[null]	103
10	1	5	104

4. Using your newly generated table - can you join all of the information together to form a table -- which has the following information for successful deliveries?

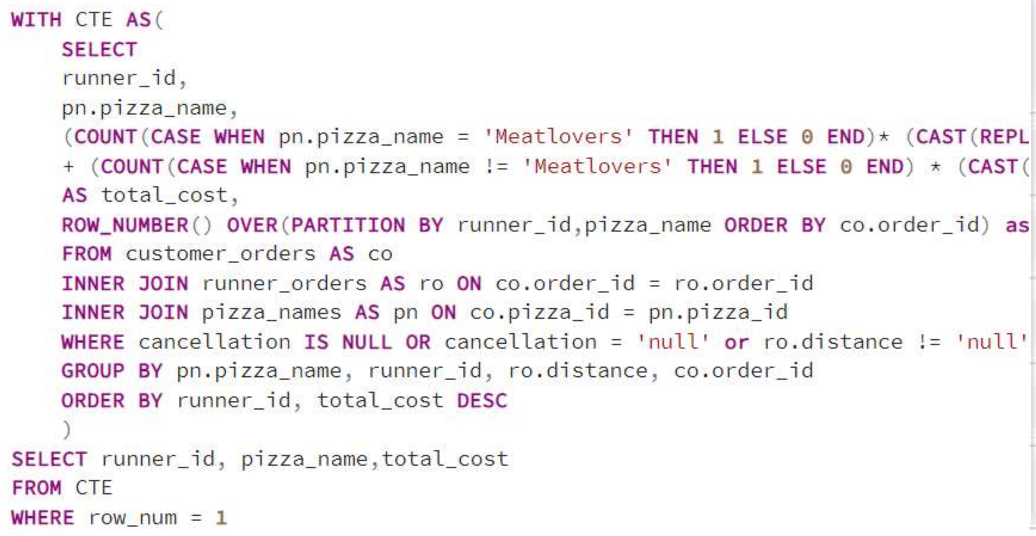


```
SELECT
    co.order_id,
    co.customer_id,
    ro.runner_id,
    rr.rating,
    co.order_time,
    ro.pickup_time,
    EXTRACT('minute' FROM (to_timestamp(ro.pickup_time, 'yy-mm-dd HH24:MI:SS.MS') - co.order_time)) AS
    REGEXP_REPLACE(duration, '[[:alpha:]]', '', 'g')::INT AS delivery_duration,
    (AVG(REPLACE(distance, 'km', '')::NUMERIC(3, 1) * 1000 / REGEXP_REPLACE(duration, '[[:alpha:]]', '
    COUNT(co.pizza_id) AS total_number_of_pizzas
FROM
    customer_orders AS co
INNER JOIN
    runner_orders AS ro ON co.order_id = ro.order_id
INNER JOIN
    runner_ratings AS rr ON co.order_id = rr.order_id
WHERE
    ro.cancellation IS NULL OR ro.cancellation = 'null' OR pickup_time != 'null'
GROUP BY
    co.customer_id, co.order_id, ro.runner_id, rr.rating, co.order_time, ro.pickup_time, ro.duration
ORDER BY
    co.order_id, co.customer_id, ro.runner_id
```

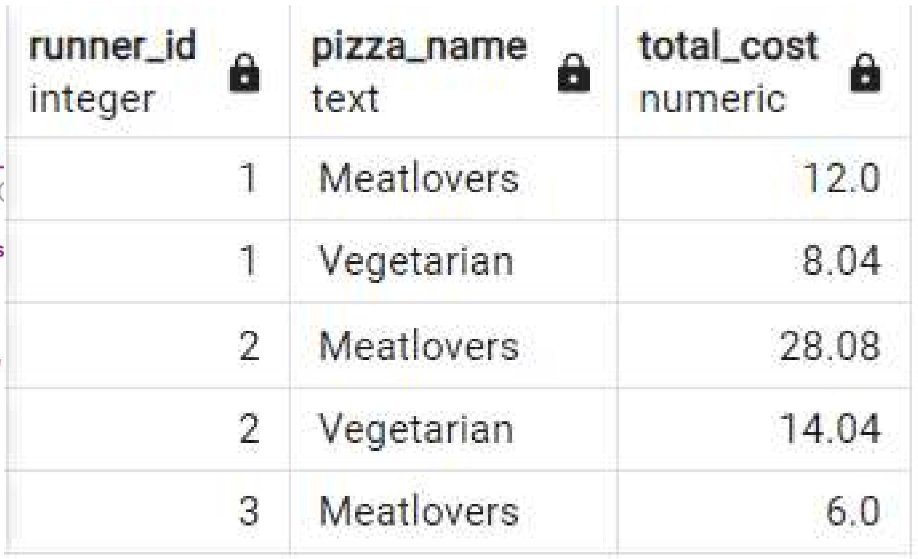


order_id integer	customer_id integer	runner_id integer	rating integer	order_time timestamp without time zone	pickup_time character varying (19)	time_between_order_and_pickup numeric	delivery_duration integer	avg_speed numeric (10,1)	total_number_of_pizzas bigint
1	101	1	4	2020-01-01 18:05:02	2020-01-01 18:15:34	10	32	37.5	1
2	101	1	5	2020-01-01 19:00:52	2020-01-01 19:10:54	10	27	44.4	1
3	102	1	5	2020-01-02 23:51:23	2020-01-03 00:12:37	21	20	40.2	2
4	103	2	4	2020-01-04 13:23:46	2020-01-04 13:53:03	29	40	35.1	3
5	104	3	4	2020-01-08 21:00:29	2020-01-08 21:10:57	10	15	40.0	1
7	105	2	4	2020-01-08 21:20:29	2020-01-08 21:30:45	10	25	60.0	1
8	102	2	5	2020-01-09 23:54:33	2020-01-10 00:15:02	20	15	93.6	1
10	104	1	5	2020-01-11 18:34:49	2020-01-11 18:50:20	15	10	60.0	2

5. If a Meat Lovers pizza was $12 and Vegetarian $10 fixed prices with no cost for extras and each runner is paid $0.30 per kilometer traveled how much money does Pizza Runner have left over after these deliveries?



```
WITH CTE AS(
    SELECT
    runner_id,
    pn.pizza_name,
    (COUNT(CASE WHEN pn.pizza_name = 'Meatlovers' THEN 1 ELSE 0 END)* (CAST(REPL
    + (COUNT(CASE WHEN pn.pizza_name != 'Meatlovers' THEN 1 ELSE 0 END) * (CAST(
    AS total_cost,
    ROW_NUMBER() OVER(PARTITION BY runner_id,pizza_name ORDER BY co.order_id) as
    FROM customer_orders AS co
    INNER JOIN runner_orders AS ro ON co.order_id = ro.order_id
    INNER JOIN pizza_names AS pn ON co.pizza_id = pn.pizza_id
    WHERE cancellation IS NULL OR cancellation = 'null' or ro.distance != 'null'
    GROUP BY pn.pizza_name, runner_id, ro.distance, co.order_id
    ORDER BY runner_id, total_cost DESC
    )
SELECT runner_id, pizza_name,total_cost
FROM CTE
WHERE row_num = 1
```



runner_id integer	pizza_name text	total_cost numeric
1	Meatlovers	12.0
1	Vegetarian	8.04
2	Meatlovers	28.08
2	Vegetarian	14.04
3	Meatlovers	6.0