A. Pizza Metrics

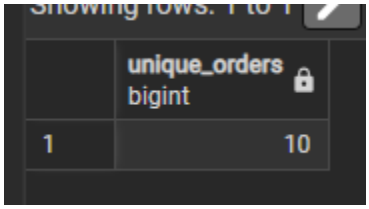
--1. How many pizzas were ordered?

```
select count(order_id) as total_pizza
from customer_orders;
```

	total_pizza bigint
1	14

-- 2. How many unique customer orders were made?

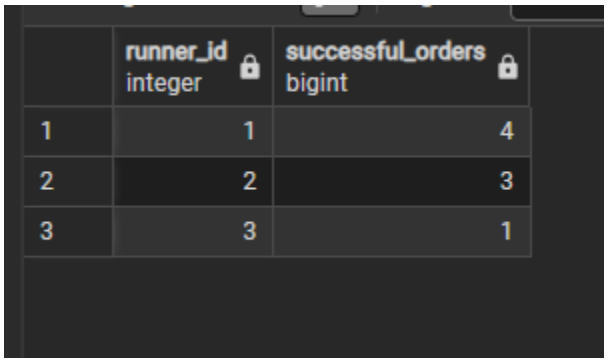
```
select count(distinct(order_id)) as unique_orders
from customer_orders;
```



	unique_orders bigint
1	10

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

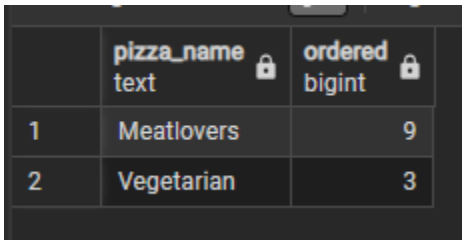
```
select
  runner_id,
  COUNT(order_id) AS successful_orders
from runner_orders
Where cancellation is null
group by runner_id;
```



	runner_id integer	successful_orders bigint
1	1	4
2	2	3
3	3	1

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

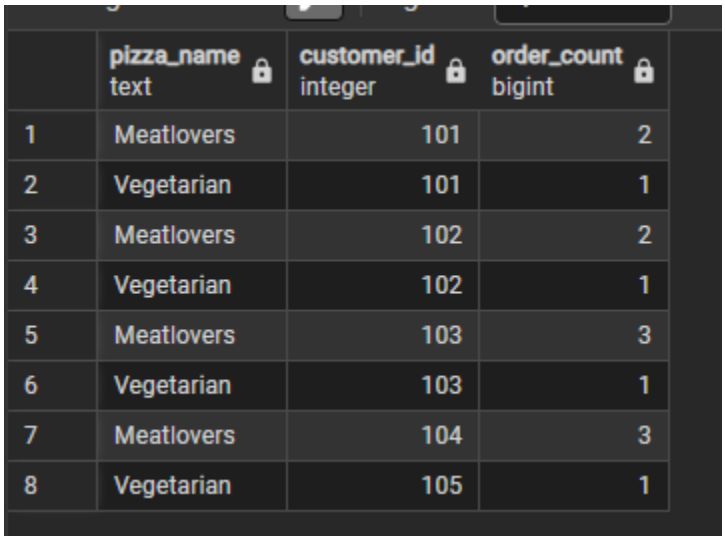
```
select p.pizza_name, count(c.pizza_id) as ordered
from customer_orders c
join pizza_names p using (pizza_id)
where
group by pizza_name;
```



	pizza_name text	ordered bigint
1	Meatlovers	9
2	Vegetarian	3

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

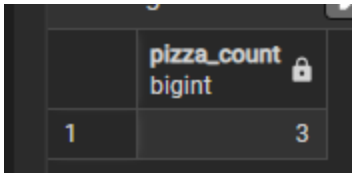
```
select pizza_name, customer_id, count(pizza_name) as order_count
from customer_orders
join pizza_names using (pizza_id)
group by customer_id, pizza_name
order by customer_id
```



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

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

```
with pizza_count_cte as
(
  select
    order_id, COUNT(pizza_id) AS pizza_order
  FROM customer_orders
  JOIN runner_orders using (order_id)
  WHERE cancellation is null
  GROUP BY order_id
)

SELECT
  MAX(pizza_order) AS pizza_count
FROM pizza_count_cte;
```



	pizza_count bigint
1	3

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

```
SELECT customer_id,
      COUNT(CASE WHEN exclusions IS NOT NULL OR extras IS NOT NULL THEN 1 END)
AS modified_pizzas,
      COUNT(CASE WHEN exclusions IS NULL AND extras IS NULL THEN 1 END) AS
unmodified_pizzas
FROM customer_orders join runner_orders using (order_id)
where cancellation is null
GROUP BY customer_id
ORDER BY customer_id;
```

	customer_id integer	modified_pizzas bigint	unmodified_pizzas bigint
1	101	0	2
2	102	1	2
3	103	3	0
4	104	3	0
5	105	1	0

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

```
SELECT COUNT(*) AS pizzas_with_both_exclusions_and_extras
FROM customer_orders
JOIN runner_orders using (order_id)
WHERE cancellation IS NULL
  AND exclusions IS NOT NULL
  AND extras IS NOT NULL;
```

	pizzas_with_both_exclusions_and_extras bigint
1	1

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

```
select date_part('hour',order_time),
count(*)
from customer_orders
group by order_time
order by order_time
```

	date_part double precision	count bigint
1	18	1
2	19	1
3	23	2
4	13	3
5	21	1
6	21	1
7	21	1
8	23	1
9	11	1
10	18	2

-- 10

```
select extract(DOW FROM order_time) AS day_num,
    TO_CHAR(order_time, 'Day') AS day_name,
    count(distinct order_id) AS total_orders
FROM customer_orders
GROUP BY day_num, day_name
ORDER BY day_num;
```

	day_num numeric	day_name text	total_orders bigint
1	3	Wednesday	5
2	4	Thursday	2
3	5	Friday	1
4	6	Saturday	2

B. Runner and Customer Experience

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

```
SELECT (registration_date - DATE '2021-01-01') / 7 + 1 AS week_no, count(*) signed_up
FROM runners
GROUP BY week_no
ORDER BY week_no;
```

	week_no integer	signed_up bigint
1	1	2
2	2	1
3	3	1

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

```
SELECT runner_id, ROUND(AVG(EXTRACT(EPOCH FROM (pickup_time - order_time)) / 60)
,2) AS avg_arrival_minutes
FROM runner_orders
join customer_orders using (order_id)
GROUP BY  runner_id
ORDER BY runner_id;
```

	runner_id integer	avg_arrival_minutes numeric
1	1	15.68
2	2	23.72
3	3	10.47

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

```
WITH prep_time_cte AS (
SELECT order_id, COUNT(order_id) AS pizza_order, order_time,pickup_time,
       EXTRACT(EPOCH FROM (pickup_time - order_time)) / 60 AS prep_time
   FROM customer_orders
   JOIN runner_orders USING (order_id)
   WHERE cancellation IS NULL
   GROUP BY order_id, order_time, pickup_time
)

SELECT
   pizza_order, ROUND(AVG(prep_time), 2) AS avg_prep_time
FROM prep_time_cte
WHERE prep_time > 1
GROUP BY pizza_order
ORDER BY pizza_order;
```

	pizza_order bigint	avg_prep_time numeric
1	1	12.36
2	2	18.38
3	3	29.28

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

```
select customer_id,ROUND(AVG(distance), 2) from
customer_orders join runner_orders using (order_id)
group by customer_id
order by customer_id
```

	customer_id integer	round numeric
1	101	20.00
2	102	16.73
3	103	23.40
4	104	10.00
5	105	25.00

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

```
select (max(duration) - min(duration)) as max_diff
from runner_orders
```

	max_diff interval
1	00:30:00

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

```
SELECT  runner_id,  customer_id, order_id, COUNT(order_id) AS pizza_count, distance,
    ROUND(
        distance / (EXTRACT(EPOCH FROM duration) / 3600), 2 ) AS avg_speed
FROM runner_orders JOIN customer_orders USING (order_id) WHERE cancellation IS NULL
GROUP BY runner_id, customer_id, order_id,  distance, duration
ORDER BY order_id;
```

	runner_id integer	customer_id integer	order_id integer	pizza_count bigint	distance numeric (5,2)	avg_speed numeric
1	1	101	1	1	20.00	37.50
2	1	101	2	1	20.00	44.44
3	1	102	3	2	13.40	40.20
4	2	103	4	3	23.40	35.10
5	3	104	5	1	10.00	40.00
6	2	105	7	1	25.00	60.00
7	2	102	8	1	23.40	93.60
8	1	104	10	2	10.00	60.00

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

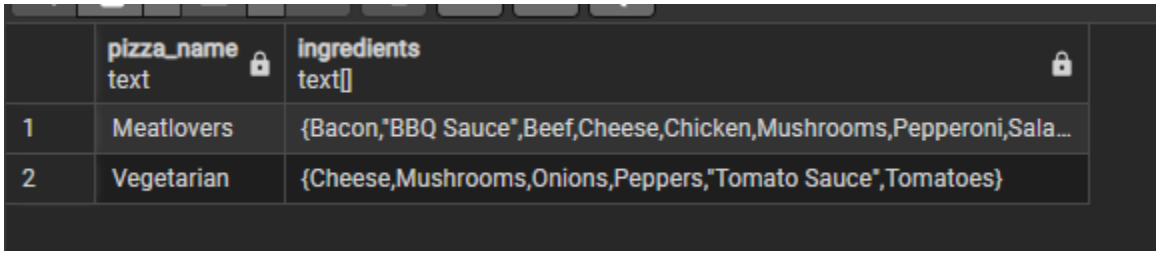
```
SELECT runner_id,
ROUND( 100.0 * COUNT(CASE WHEN cancellation IS NULL THEN 1 END) / COUNT(*),2) AS
success_percentage
FROM runner_orders
GROUP BY runner_id
ORDER BY runner_id;
```

	runner_id integer	success_percentage numeric
1	1	100.00
2	2	75.00
3	3	50.00

C. Ingredient Optimisation

--1. What are the standard ingredients for each pizza?

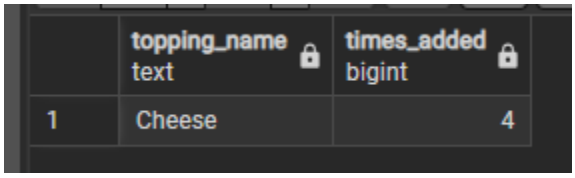
```
SELECT pizza_name,
   ARRAY_AGG(topping_name ORDER BY topping_name) AS ingredients
FROM pizza_recipes
JOIN pizza_names using (pizza_id)
JOIN pizza_toppings using (topping_id)
GROUP BY pizza_name
ORDER BY pizza_name;
```



	pizza_name text	ingredients text[]
1	Meatlovers	{Bacon,"BBQ Sauce",Beef,Cheese,Chicken,Mushrooms,Pepperoni,Sala...
2	Vegetarian	{Cheese,Mushrooms,Onions,Peppers,"Tomato Sauce",Tomatoes}

--2. What was the most commonly added extra?

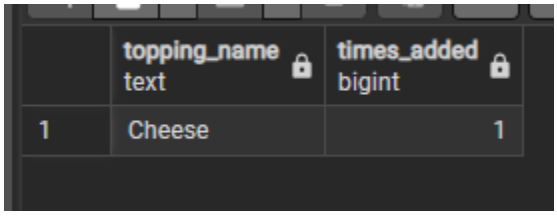
```
WITH extras_split AS (
   SELECT regexp_split_to_table(exclusions, ',')::int AS topping_id
   FROM customer_orders
   WHERE exclusions IS NOT NULL
)
SELECT pt.topping_name,COUNT(*) AS times_added
FROM extras_split
JOIN pizza_toppings pt USING(topping_id)
GROUP BY pt.topping_name
ORDER BY times_added DESC
limit 1
```



	topping_name text	times_added bigint
1	Cheese	4

--3. What was the most common exclusion?

```
WITH extras_split AS (
   SELECT regexp_split_to_table(exclusions, ',')::int AS topping_id
   FROM customer_orders
   WHERE extras IS NOT NULL
)
SELECT pt.topping_name,COUNT(*) AS times_added
FROM extras_split
JOIN pizza_toppings pt USING(topping_id)
GROUP BY pt.topping_name
ORDER BY times_added DESC
limit 1
```

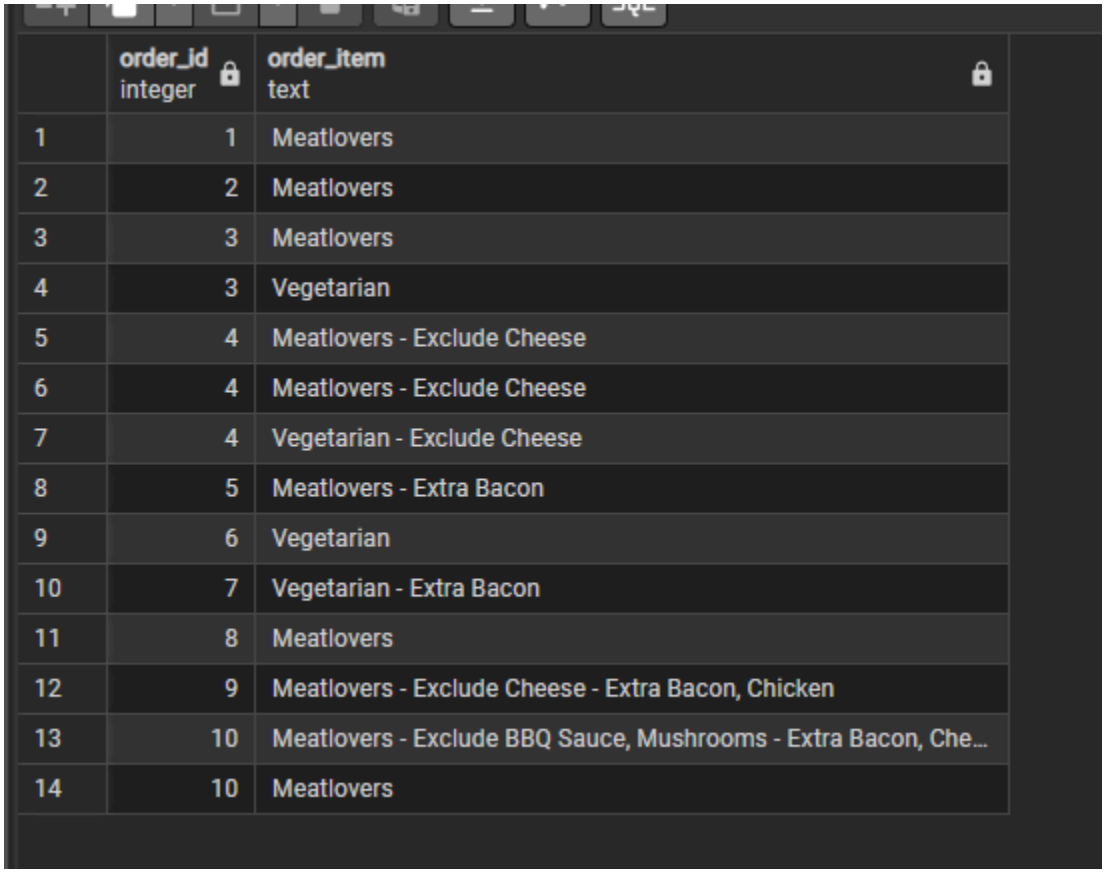


	topping_name text	times_added bigint
1	Cheese	1

-- 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 formatted_orders AS (
   SELECT order_id, pizza_name,
-- exclusions
  (
          SELECT STRING_AGG(topping_name, ', ' ORDER BY topping_name)
          FROM unnest(string_to_array(exclusions, ',')) AS ex_id
          JOIN pizza_toppings
            ON topping_id = ex_id::int
       ) AS exclusion_names,
-- extras

       (
          SELECT STRING_AGG(topping_name, ', ' ORDER BY topping_name)
          FROM unnest(string_to_array(extras, ',')) AS ex_id
          JOIN pizza_toppings
            ON topping_id = ex_id::int
       ) AS extra_names

   FROM customer_orders
   JOIN pizza_names using (pizza_id)
)

SELECT
   order_id,
   CASE
      WHEN exclusion_names IS NULL AND extra_names IS NULL
         THEN pizza_name

      WHEN exclusion_names IS NOT NULL AND extra_names IS NULL
         THEN pizza_name || ' - Exclude ' || exclusion_names

      WHEN exclusion_names IS NULL AND extra_names IS NOT NULL
         THEN pizza_name || ' - Extra ' || extra_names

      ELSE pizza_name ||
          ' - Exclude ' || exclusion_names ||
          ' - Extra ' || extra_names
   END AS order_item
FROM formatted_orders
ORDER BY order_id;
```



	order_id integer	order_item text
1	1	Meatlovers
2	2	Meatlovers
3	3	Meatlovers
4	3	Vegetarian
5	4	Meatlovers - Exclude Cheese
6	4	Meatlovers - Exclude Cheese
7	4	Vegetarian - Exclude Cheese
8	5	Meatlovers - Extra Bacon
9	6	Vegetarian
10	7	Vegetarian - Extra Bacon
11	8	Meatlovers
12	9	Meatlovers - Exclude Cheese - Extra Bacon, Chicken
13	10	Meatlovers - Exclude BBQ Sauce, Mushrooms - Extra Bacon, Che...
14	10	Meatlovers

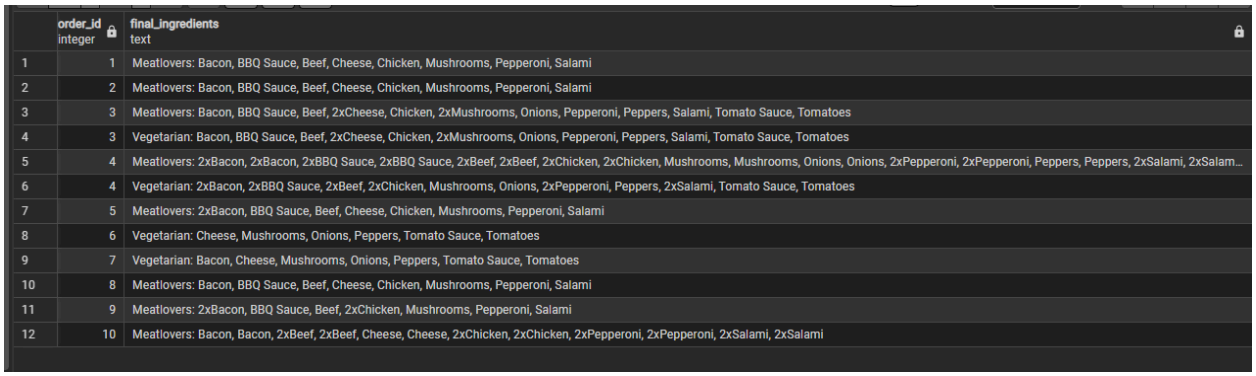
— 5. Generate an alphabetically ordered comma separated ingredient list for each pizza order from the `customer_orders` table and add a `2x` in front of any relevant ingredients

For example: `"Meat Lovers: 2xBacon, Beef, ... , Salami"`

```
WITH

base_cte AS (
   SELECT
      co.order_id,
      pt.topping_name
   FROM customer_orders co
   JOIN pizza_recipes pr USING (pizza_id)
   JOIN pizza_toppings pt USING (topping_id)
),
```

```
excluded_cte AS (
    SELECT
        co.order_id,
        pt.topping_name AS topping_name
    FROM customer_orders co
    JOIN regexp_split_to_table(co.exclusions, ',') AS e(tid) ON TRUE
    JOIN pizza_toppings pt ON pt.topping_id = e.tid::int
    WHERE co.exclusions IS NOT NULL
),


extras_cte AS (
    SELECT
        co.order_id,
        pt.topping_name AS topping_name
    FROM customer_orders co
    JOIN regexp_split_to_table(co.extras, ',') AS e(tid) ON TRUE
    JOIN pizza_toppings pt ON pt.topping_id = e.tid::int
    WHERE co.extras IS NOT NULL
),


final_rows AS (
    SELECT order_id, topping_name
    FROM base_cte
    WHERE (order_id, topping_name) NOT IN (
        SELECT order_id, topping_name FROM excluded_cte
    )
    UNION ALL
    SELECT * FROM extras_cte
),


counted AS (
    SELECT
        order_id,
        topping_name,
        COUNT(*) AS cnt
    FROM final_rows
    GROUP BY order_id, topping_name
)
```

```
SELECT
    co.order_id,
    pn.pizza_name || ': ' ||
    STRING_AGG(
        CASE WHEN c.cnt = 2 THEN '2x' || c.topping_name
            ELSE c.topping_name
        END,
        ', ' ORDER BY c.topping_name
    ) AS final_ingredients
FROM counted c
JOIN customer_orders co USING (order_id)
JOIN pizza_names pn USING (pizza_id)
GROUP BY co.order_id, pn.pizza_name
ORDER BY co.order_id;
```

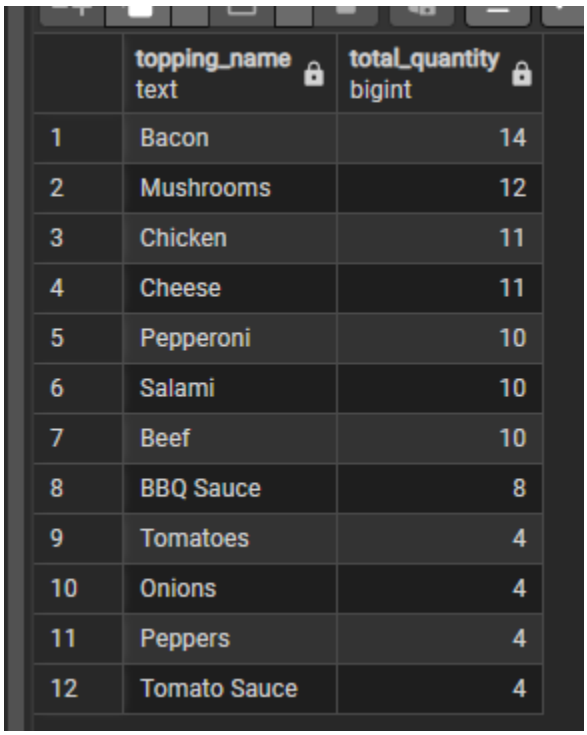


	order_id integer	final_ingredients text
1	1	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
2	2	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
3	3	Meatlovers: Bacon, BBQ Sauce, Beef, 2xCheese, Chicken, 2xMushrooms, Onions, Pepperoni, Peppers, Salami, Tomato Sauce, Tomatoes
4	3	Vegetarian: Bacon, BBQ Sauce, Beef, 2xCheese, Chicken, 2xMushrooms, Onions, Pepperoni, Peppers, Salami, Tomato Sauce, Tomatoes
5	4	Meatlovers: 2xBacon, 2xBacon, 2xBBQ Sauce, 2xBBQ Sauce, 2xBeef, 2xBeef, 2xChicken, 2xChicken, Mushrooms, Mushrooms, Onions, Onions, 2xPepperoni, 2xPepperoni, Peppers, Peppers, 2xSalami, 2xSalam…
6	4	Vegetarian: 2xBacon, 2xBBQ Sauce, 2xBeef, 2xChicken, Mushrooms, Onions, 2xPepperoni, Peppers, 2xSalami, Tomato Sauce, Tomatoes
7	5	Meatlovers: 2xBacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
8	6	Vegetarian: Cheese, Mushrooms, Onions, Peppers, Tomato Sauce, Tomatoes
9	7	Vegetarian: Bacon, Cheese, Mushrooms, Onions, Peppers, Tomato Sauce, Tomatoes
10	8	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
11	9	Meatlovers: 2xBacon, BBQ Sauce, Beef, 2xChicken, Mushrooms, Pepperoni, Salami
12	10	Meatlovers: Bacon, Bacon, 2xBeef, 2xBeef, Cheese, Cheese, 2xChicken, 2xChicken, 2xPepperoni, 2xPepperoni, 2xSalami, 2xSalami

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

```
with base_cte AS (
    SELECT order_id, topping_name
    FROM customer_orders
    JOIN pizza_recipes pr USING (pizza_id)
    JOIN pizza_toppings pt USING (topping_id)
),
excluded_cte AS (
    SELECT order_id,topping_name
    FROM customer_orders
    JOIN regexp_split_to_table(exclusions, ',') AS e(tid) ON TRUE
    JOIN pizza_toppings pt ON pt.topping_id = e.tid::int
```

```
    WHERE exclusions IS NOT NULL
),
extras_cte AS (
    SELECT order_id, pt.topping_name AS topping_name
    FROM customer_orders co
    JOIN regexp_split_to_table(co.extras, ',') AS e(tid) ON TRUE
    JOIN pizza_toppings pt ON pt.topping_id = e.tid::int
    WHERE extras IS NOT NULL
),
final_rows AS (

    SELECT order_id, topping_name FROM base_cte
    WHERE (order_id, topping_name) NOT IN (
        SELECT order_id, topping_name FROM excluded_cte
    )
    UNION ALL
    SELECT * FROM extras_cte
)

SELECT topping_name,
    COUNT(*) AS total_quantity
FROM final_rows
GROUP BY topping_name
ORDER BY total_quantity DESC;
```

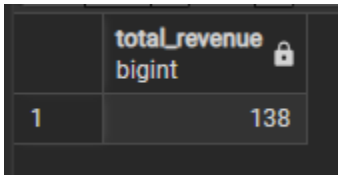


	topping_name text	total_quantity bigint
1	Bacon	14
2	Mushrooms	12
3	Chicken	11
4	Cheese	11
5	Pepperoni	10
6	Salami	10
7	Beef	10
8	BBQ Sauce	8
9	Tomatoes	4
10	Onions	4
11	Peppers	4
12	Tomato Sauce	4

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

```
WITH delivered AS (
  SELECT * FROM customer_orders
  JOIN runner_orders USING(order_id)
  WHERE cancellation IS NULL OR cancellation IN ('', 'null')
)

SELECT
  SUM(CASE
        WHEN pizza_id = 1 THEN 12
        WHEN pizza_id = 2 THEN 10
      END) AS total_revenue
FROM delivered;
```

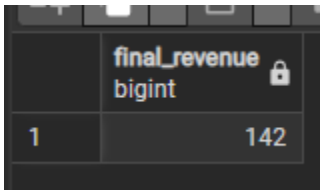


	total_revenue bigint
1	138

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

```
WITH delivered AS (
  SELECT * FROM customer_orders
  JOIN runner_orders USING(order_id)
  WHERE cancellation IS NULL
     OR cancellation IN ('', 'null')
),

base_revenue AS (
  SELECT
    SUM(CASE
          WHEN pizza_id = 1 THEN 12
          WHEN pizza_id = 2 THEN 10
        END) AS base_total
  FROM delivered
),
```

```
extra_count AS (
  SELECT COUNT(*) AS total_extras
  FROM delivered, UNNEST(STRING_TO_ARRAY(extras, ',')) AS e
  WHERE extras IS NOT NULL
    AND extras NOT IN ('', 'null')
)

SELECT
  base_total + total_extras AS final_revenue
FROM base_revenue, extra_count;
```



	final_revenue bigint
1	142

-- 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.

```
CREATE TABLE runner_ratings (
    rating_id SERIAL PRIMARY KEY,
    order_id INT NOT NULL,
    runner_id INT NOT NULL,
    rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5),
    comment TEXT
)

INSERT INTO runner_ratings (order_id, runner_id, rating, comment)
VALUES
(1, 1, 5, 'Super fast delivery'),
(2, 1, 4, 'Quick delivery'),
(3, 1, 5, 'Excellent service'),
(4, 2, 3, 'took a bit longer'),
(5, 3, 4, 'Good service'),
(7, 2, 2, 'Late arrived'),
(8, 2, 5, 'professional and fast delivery'),
(10, 1, 4, 'friendly runner');

select * from runner_ratings;
```

	rating_id [PK] integer	order_id integer	runner_id integer	rating integer	comment text
1	1	1	1	5	Super fast delivery
2	2	2	1	4	Quick delivery
3	3	3	1	5	Excellent service
4	4	4	2	3	took a bit longer
5	5	5	3	4	Good service
6	6	7	2	2	Late arrived
7	7	8	2	5	professional and fast deliv...
8	8	10	1	4	friendly runner

-- 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**

```
select customer_id,order_id,
rating, order_time as ot,pickup_time as pt,
(pickup_time  - order_time) as time_diff, duration,
ROUND( distance / (EXTRACT(EPOCH FROM duration) / 3600), 2 ) AS avg_speed,
Count(pizza_id) as total_pizza
from customer_orders join runner_orders using (order_id)
join runner_ratings using (order_id)
group by distance,rating_id, customer_id, order_id,pizza_id, order_time,pickup_time,duration
```

	customer_id integer	order_id integer	rating integer	ot timestamp without time zone	pt timestamp without time zone	time_diff interval	duration interval	avg_speed numeric	total_pizza bigint
1	102	8	5	2020-01-09 23:54:33	2020-01-10 00:15:02	00:20:29	00:15:00	93.60	1
2	101	1	5	2020-01-01 18:05:02	2020-01-01 18:15:34	00:10:32	00:32:00	37.50	1
3	102	3	5	2020-01-02 23:51:23	2020-01-03 00:12:37	00:21:14	00:20:00	40.20	1
4	101	2	4	2020-01-01 19:00:52	2020-01-01 19:10:54	00:10:02	00:27:00	44.44	1
5	104	10	4	2020-01-11 18:34:49	2020-01-11 18:50:20	00:15:31	00:10:00	60.00	2
6	104	5	4	2020-01-08 21:00:29	2020-01-08 21:10:57	00:10:28	00:15:00	40.00	1
7	102	3	5	2020-01-02 23:51:23	2020-01-03 00:12:37	00:21:14	00:20:00	40.20	1
8	103	4	3	2020-01-04 13:23:46	2020-01-04 13:53:03	00:29:17	00:40:00	35.10	2
9	105	7	2	2020-01-08 21:20:29	2020-01-08 21:30:45	00:10:16	00:25:00	60.00	1
10	103	4	3	2020-01-04 13:23:46	2020-01-04 13:53:03	00:29:17	00:40:00	35.10	1

-- 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?

```
WITH delivered AS (
  SELECT * FROM customer_orders
  JOIN runner_orders USING(order_id)
  WHERE cancellation IS NULL OR cancellation IN ('', 'null')
),

pizza_cte as (
select SUM(CASE
        WHEN pizza_id = 1 THEN 12
        WHEN pizza_id = 2 THEN 10
      END) as pizza_cost FROM delivered
),

deliver_cte as (
select sum(distance)*0.30 as deliver_cost
from runner_orders
)

select pizza_cost - deliver_cost as
profit from pizza_cte, deliver_cte
```

	profit numeric
1	94.4400