Name: Sumit Chauhan

Department: Data Engineering

SQL 8 Week Challenge

Case Study #2

(pizza runner)

```
------------------ Case Study #2 ------------------

------------------ SETUP ------------------

CREATE SCHEMA pizza_runner;
SET search_path = pizza_runner;

DROP TABLE IF EXISTS runners;
CREATE TABLE runners (
  "runner_id" INTEGER,
  "registration_date" DATE
);
INSERT INTO runners
  ("runner_id", "registration_date")
VALUES
  (1, '2021-01-01'),
  (2, '2021-01-03'),
  (3, '2021-01-08'),
  (4, '2021-01-15');


DROP TABLE IF EXISTS customer_orders;
CREATE TABLE customer_orders (
  "order_id" INTEGER,
  "customer_id" INTEGER,
  "pizza_id" INTEGER,
  "exclusions" VARCHAR(4),
  "extras" VARCHAR(4),
  "order_time" TIMESTAMP
);

INSERT INTO customer_orders
  ("order_id", "customer_id", "pizza_id", "exclusions", "extras", "order_time")
```

```
VALUES
  ('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');



DROP TABLE IF EXISTS runner_orders;
CREATE TABLE runner_orders (
  "order_id" INTEGER,
  "runner_id" INTEGER,
  "pickup_time" VARCHAR(19),
  "distance" VARCHAR(7),
  "duration" VARCHAR(10),
  "cancellation" VARCHAR(23)
);

INSERT INTO runner_orders
  ("order_id", "runner_id", "pickup_time", "distance", "duration",
"cancellation")
VALUES
  ('1', '1', '2020-01-01 18:15:34', '20km', '32 minutes', ''),
  ('2', '1', '2020-01-01 19:10:54', '20km', '27 minutes', ''),
  ('3', '1', '2020-01-03 00:12:37', '13.4km', '20 mins', 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', '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');
```

```sql
DROP TABLE IF EXISTS pizza_names;
CREATE TABLE pizza_names (
  "pizza_id" INTEGER,
  "pizza_name" TEXT
);
INSERT INTO pizza_names
  ("pizza_id", "pizza_name")
VALUES
  (1, 'Meatlovers'),
  (2, 'Vegetarian');


DROP TABLE IF EXISTS pizza_recipes;
CREATE TABLE pizza_recipes (
  "pizza_id" INTEGER,
  "toppings" TEXT
);
INSERT INTO pizza_recipes
  ("pizza_id", "toppings")
VALUES
  (1, '1, 2, 3, 4, 5, 6, 8, 10'),
  (2, '4, 6, 7, 9, 11, 12');


DROP TABLE IF EXISTS pizza_toppings;
CREATE TABLE pizza_toppings (
  "topping_id" INTEGER,
  "topping_name" TEXT
);
INSERT INTO pizza_toppings
  ("topping_id", "topping_name")
VALUES
  (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');



----------------- SELECT -----------------
SELECT * FROM runners;

SELECT * FROM customer_orders;

SELECT * FROM runner_orders;

SELECT * FROM pizza_names;

SELECT * FROM pizza_recipes;

SELECT * FROM pizza_toppings;

----------------- Handling Corrupted data -----------------

-- cleaning up fake null values in exclusions from customer_orders table
UPDATE customer_orders
SET exclusions=NULL
WHERE exclusions IN ('','null');
```

```sql
SELECT order_id, exclusions FROM customer_orders ORDER BY order_id;
```

	order_id integer	exclusions character varying (4)
1	1	[null]
2	2	[null]
3	3	[null]
4	3	[null]
5	4	4
6	4	4
7	4	4
8	5	[null]
9	6	[null]
10	7	[null]
11	8	[null]
12	9	4
13	10	2,6
14	10	[null]

```sql
-- cleaning up fake null values in extras from customer_orders table
UPDATE customer_orders
SET extras=NULL
WHERE extras IN ('','null');
```

```sql
SELECT order_id, extras FROM customer_orders ORDER BY order_id;
```

	order_id integer	extras character varying (4)
1	1	[null]
2	2	[null]
3	3	[null]
4	3	[null]
5	4	[null]
6	4	[null]
7	4	[null]
8	5	1
9	6	[null]
10	7	1
11	8	[null]
12	9	1,5
13	10	1,4
14	10	[null]

```sql
-- in customer_orders table
-- removing extra spaces from extras and exclusion columns
UPDATE pizza_runner.customer_orders
SET exclusions = REPLACE(exclusions, ' ', '')
WHERE exclusions IS NOT NULL;

UPDATE pizza_runner.customer_orders
SET extras = REPLACE(extras, ' ', '')
WHERE extras IS NOT NULL;

SELECT exclusions, extras FROM customer_orders;
```

	exclusions character varying (4)	extras character varying (4)
1	[null]	[null]
2	[null]	[null]
3	[null]	[null]
4	[null]	[null]
5	[null]	[null]
6	[null]	[null]
7	[null]	[null]
8	4	[null]
9	4	[null]
10	4	[null]
11	[null]	1
12	[null]	1
13	4	1,5
14	2,6	1,4

```sql
-- Altering pickup_time column in runner_orders table
-- old type: varchar
-- new type: TIMESTAMP
ALTER TABLE runner_orders
ALTER COLUMN pickup_time TYPE TIMESTAMP
USING NULLIF(pickup_time, 'null')::TIMESTAMP;

-- SELECT pickup_time FROM runner_orders;
```

	pickup_time timestamp without time zone
1	2020-01-03 00:12:37
2	2020-01-04 13:53:03
3	2020-01-08 21:10:57
4	[null]
5	[null]
6	2020-01-01 18:15:34
7	2020-01-01 19:10:54
8	2020-01-08 21:30:45
9	2020-01-10 00:15:02
10	2020-01-11 18:50:20

```
-- distance values are inconsitent in runner_orders
-- old type: varchar
-- new type: NUMERIC
ALTER TABLE runner_orders
ALTER COLUMN distance TYPE NUMERIC
USING NULLIF(REGEXP_REPLACE(distance,'[^0-9.]','','g') ,'')::NUMERIC;
-- now the distance column contains distance in kms

-- SELECT distance FROM runner_orders;
```

	distance numeric
1	13.4
2	23.4
3	10
4	[null]
5	[null]
6	20
7	20
8	25
9	23.4
10	10

```
-- duration values are incosistent in runner_orders
-- old type: varchar
```

```sql
-- new type: NUMERIC
ALTER TABLE runner_orders
ALTER COLUMN duration TYPE INTEGER
USING NULLIF(REGEXP_REPLACE(duration,'[^0-9]','','g'),'')::INTEGER;
-- now the duration column contains duration in minutes



-- SELECT duration FROM runner_orders;
```

	duration integer
1	20
2	40
3	15
4	[null]
5	[null]
6	32
7	27
8	25
9	15
10	10

```sql

-- cancelation columns have multiple types of null values
-- like, 'null', '', NULL
-- Unify all the nulls
UPDATE pizza_runner.runner_orders
SET cancellation = NULL
WHERE cancellation IN ('', 'null');

SELECT order_id,cancellation FROM runner_orders
```

	order_id integer	cancellation character varying (23)
1	3	[null]
2	4	[null]
3	5	[null]
4	6	Restaurant Cancellati...
5	9	Customer Cancellation
6	1	[null]
7	2	[null]
8	7	[null]
9	8	[null]
10	10	[null]

```
-- in customer_orders, 1 row= 1 pizza
-- in runner_orders, 1 row = 1 order

-- giving proper name to pizza
-- this query hasn't performed
-- UPDATE pizza_runner.pizza_names
-- SET pizza_name = 'Meatlovers'
-- WHERE pizza_name = 'Meat Lovers';

SELECT * FROM pizza_names ORDER BY pizza_id;
```

	pizza_id integer	pizza_name text
1	1	Meatlovers
2	2	Vegetarian

```
-- Mixed timeline issues
SELECT r.order_id, (r.pickup_time > c.order_time) foo
FROM customer_orders c
JOIN runner_orders r ON c.order_id=r.order_id;
```

	order_id integer	foo boolean
1	3	true
2	1	true
3	2	true
4	3	true
5	6	[null]
6	8	true
7	10	true
8	4	true
9	4	true
10	4	true
11	5	true
12	7	true
13	9	[null]
14	10	true

```
-- the above query validates that all pickup times are > than order time

-- as per the questions asked in challege
-- we don't need to apply normalization
-- as the columns extras and exclusion in customer_orders table
-- are independent of each other




----------------- SELECT -----------------
SELECT * FROM runners;
```

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

```

SELECT * FROM customer_orders;
```

	order_id integer	customer_id integer	pizza_id integer	exclusions character varying (4)	extras character varying (4)	order_time timestamp without time zone
1	3	102	2	[null]	[null]	2020-01-02 23:51:23
2	1	101	1	[null]	[null]	2020-01-01 18:05:02
3	2	101	1	[null]	[null]	2020-01-01 19:00:52
4	3	102	1	[null]	[null]	2020-01-02 23:51:23
5	6	101	2	[null]	[null]	2020-01-08 21:03:13
6	8	102	1	[null]	[null]	2020-01-09 23:54:33
7	10	104	1	[null]	[null]	2020-01-11 18:34:49
8	4	103	1	4	[null]	2020-01-04 13:23:46
9	4	103	1	4	[null]	2020-01-04 13:23:46
10	4	103	2	4	[null]	2020-01-04 13:23:46
11	5	104	1	[null]	1	2020-01-08 21:00:29
12	7	105	2	[null]	1	2020-01-08 21:20:29
13	9	103	1	4	1,5	2020-01-10 11:22:59
14	10	104	1	2,6	1,4	2020-01-11 18:34:49

```
SELECT * FROM runner_orders;
```

	order_id integer	runner_id integer	pickup_time timestamp without time zone	distance numeric	duration integer	cancellation character varying (23)
1	3	1	2020-01-03 00:12:37	13.4	20	[null]
2	4	2	2020-01-04 13:53:03	23.4	40	[null]
3	5	3	2020-01-08 21:10:57	10	15	[null]
4	6	3	[null]	[null]	[null]	Restaurant Cancellati...
5	9	2	[null]	[null]	[null]	Customer Cancellation
6	1	1	2020-01-01 18:15:34	20	32	[null]
7	2	1	2020-01-01 19:10:54	20	27	[null]
8	7	2	2020-01-08 21:30:45	25	25	[null]
9	8	2	2020-01-10 00:15:02	23.4	15	[null]
10	10	1	2020-01-11 18:50:20	10	10	[null]

```
SELECT * FROM pizza_names;
```

	pizza_id integer	pizza_name text
1	2	Vegetarian
2	1	Meatlovers

```
SELECT * FROM pizza_recipes;
```

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

```
SELECT * FROM pizza_toppings;
```

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

```
----------------- Case Study Questions -----------------

----------------- A. Pizza Metrics -----------------
-- NOTE: customer_orders is for orders made by customer
-- and runner_orders is for orders delivery

-- One common assumption is that if an order is cancelled
-- its pickup_time will be null
-- otherwise it definitely contains valid value

-- 1. How many pizzas were ordered?
-- customer_orders tells in which order which pizza was added
-- so total records = total pizza ordered
SELECT COUNT(*) as pizzas_ordered
FROM customer_orders;
```

	pizzas_ordered bigint
1	14

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

	count bigint
1	10

```
-- 3. How many successful orders were delivered by each runner?
SELECT runner_id, COUNT(distinct order_id) successful_orders
FROM runner_orders
WHERE cancellation IS NULL AND pickup_time IS NOT 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 c.pizza_id, COUNT(*)
FROM customer_orders c
JOIN runner_orders r ON c.order_id = r.order_id
WHERE r.cancellation IS NULL
GROUP BY c.pizza_id;
```

	pizza_id integer	count bigint
1	1	9
2	2	3

```
-- remaining: join pizza table to display 0 ordered pizzas

-- 5. How many Vegetarian and Meatlovers were ordered by each customer?
SELECT customer_id, p.pizza_name , COUNT(*)
FROM customer_orders c
JOIN pizza_names p USING(pizza_id)
WHERE p.pizza_name IN ('Vegetarian', 'Meatlovers')
GROUP BY c.customer_id, p.pizza_name;
```

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

```sql
-- 6. What was the maximum number of pizzas delivered in a single order?
WITH cte AS (
  SELECT r.order_id, COUNT(*) no_of_pizzas
  FROM customer_orders c
  JOIN runner_orders r USING(order_id)
  WHERE r.cancellation IS NULL AND r.pickup_time IS NOT NULL
  GROUP BY r.order_id
)
SELECT MAX(no_of_pizzas) max_no_of_pizzas_delivered
FROM cte;
--  or we can use
-- ORDER BY no_of_pizzas DESC LIMIT 1
```

	max_no_of_pizzas_delivered bigint
1	3

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

-- here we are interpreting change as
-- addition of some ingredient or removal of some ingredient
-- so if extras or exclusions are not null then its a change
SELECT c.customer_id,
  SUM((c.exclusions IS NOT NULL OR c.extras IS NOT NULL)::int) change_needed,
  SUM((c.exclusions IS NULL AND c.extras IS NULL)::int) no_change
FROM customer_orders c
JOIN runner_orders r USING (order_id)
WHERE r.cancellation IS NULL
```

```
GROUP BY c.customer_id
ORDER BY c.customer_id;
```

	customer_id integer	change_needed bigint	no_change 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(*) exc_and_extras
FROM customer_orders c
JOIN runner_orders r USING (order_id)
WHERE r.cancellation IS NULL
  AND c.exclusions IS NOT NULL AND c.extras IS NOT NULL
;
-- we can use the same thing we used in 7th query
```

	exc_and_extras bigint
1	1

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

-- here there is no mention of delivery so ignoring runner_orders table
-- showing hours in which pizzas were ordered
-- ignoring hours in which there were no orders
-- also grouping is done based on hours
-- and not on the basis of date + hours
SELECT EXTRACT(HOUR FROM order_time) hour_of_day,
    COUNT(*) pizzas_ordered_per_hour
FROM customer_orders
GROUP BY hour_of_day
ORDER BY hour_of_day;
```

	hour_of_day numeric	pizzas_ordered_per_hour 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?
-- here also we are not showing the day of week
-- in which not a single pizza ordered
-- here also the grouping is done on the basis of day of week and
-- not on specific day (date)

SELECT TRIM(TO_CHAR(order_time, 'Day')) day_of_week,
    COUNT(DISTINCT order_id) orders_of_day
FROM customer_orders
GROUP BY day_of_week
ORDER BY day_of_week;
```

	day_of_week text	orders_of_day bigint
1	Friday	1
2	Saturday	2
3	Thursday	2
4	Wednesday	5

```
---------------- B. Runner and Customer Experience ----------------

-- 1. How many runners signed up for each 1 week period?
-- (i.e. week starts 2021-01-01)
-- here 2021-01-01 to 2021-01-07 is 1st week
SELECT (registration_date- DATE '2021-01-01')/7 + 1 week, COUNT(*)
runners_signed_up
FROM runners
GROUP BY week
ORDER BY week
;
-- temp = cur_date - 2021-01-01 % 7
```

	week integer	runners_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 r.runner_id, ROUND(EXTRACT(EPOCH FROM AVG(r.pickup_time - c.order_time)) /
60,2) avg_minutes
FROM runner_orders r
JOIN (
  SELECT DISTINCT order_id, order_time FROM customer_orders
) c USING(order_id)
WHERE r.cancellation IS NULL AND r.pickup_time IS NOT NULL
GROUP BY r.runner_id
ORDER BY r.runner_id;
```

	runner_id integer	avg_minutes numeric
1	1	14.33
2	2	20.01
3	3	10.47

```
-- 3. Is there any relationship between the number of pizzas
-- and how long the order takes to prepare?
-- assumption: we are assuming that if one order is placed then immidiately pizza
HQ will start preparing pizzas
-- and also we are assuming that if all pizzas are prepared then immidiately
runner will pick them
-- so start time = order_time and end time = pickup time

WITH cnt_time_rel AS (
SELECT order_id,
    COUNT(*) pizza_cnt,
    ROUND(EXTRACT(EPOCH FROM (r.pickup_time - c.order_time)) / 60,2) prep_time
FROM customer_orders c
JOIN runner_orders r USING (order_id)
WHERE r.pickup_time IS NOT NULL
GROUP BY order_id, r.pickup_time, c.order_time
ORDER BY pizza_cnt
)
```

```sql
SELECT pizza_cnt,ROUND(AVG(prep_time),2)
FROM cnt_time_rel
GROUP BY pizza_cnt
ORDER BY pizza_cnt;
```

	pizza_cnt bigint	round numeric
1	1	12.36
2	2	18.38
3	3	29.28

```sql
-- we have first compute the total time and number of pizzas per order
-- we haven't found linear relationship between pizza count and preparation time
-- then we tried average
-- it seem like   less the no. of pizzas, less the average preparation time
--      and   more the no. of pizzas, more the average preparation time



-- 4. What was the average distance travelled for each customer?
WITH cte AS (
  SELECT DISTINCT customer_id, order_id FROM customer_orders
)
SELECT customer_id, ROUND(AVG(r.distance),2)
FROM cte c
JOIN runner_orders r USING(order_id)
WHERE r.cancellation IS NULL AND r.pickup_time IS NOT NULL
GROUP BY customer_id
ORDER BY customer_id;
```

	customer_id integer	round numeric
1	101	20.00
2	102	18.40
3	103	23.40
4	104	10.00
5	105	25.00

```sql
-- 5. What was the difference between the longest and shortest delivery times for
all orders?
-- here delivery time refers to duration

-- assumption: delivery time is with respect to runner,
-- meaning that we need to take the duration
```

```sql
-- if it was respect to customer,
-- we had to take duration + (preparation time = pickup_time - order_time)
SELECT
    MAX(duration) - MIN(duration) AS difference_in_delivery_time
FROM runner_orders
WHERE pickup_time IS NOT NULL;
```

	difference_in_delivery_time integer
1	30

```sql
-- 6. What was the average speed for each runner for each delivery
--    and do you notice any trend for these values?
-- for each runner for each delivery
SELECT runner_id,
  order_id,
  ROUND((distance/(duration/60.0)),2) speed_km_per_hour
FROM runner_orders
WHERE duration IS NOT NULL AND distance IS NOT NULL
ORDER BY runner_id, order_id;
```

	runner_id integer	order_id integer	speed_km_per_hour numeric
1	1	1	37.50
2	1	2	44.44
3	1	3	40.20
4	1	10	60.00
5	2	4	35.10
6	2	7	60.00
7	2	8	93.60
8	3	5	40.00

```sql
-- for each runner (grouped data)
SELECT runner_id,
  ROUND(AVG(distance/(duration/60.0)),2) speed_km_per_hour
FROM runner_orders
WHERE duration IS NOT NULL AND distance IS NOT NULL
GROUP BY runner_id
ORDER BY runner_id;
```

	runner_id integer	speed_km_per_hour numeric
1	1	45.54
2	2	62.90
3	3	40.00

```sql
-- one thing we can notice
-- is that as the runners have taken more orders, more they increases their speed

-- 7. What is the successful delivery percentage for each runner?
SELECT runner_id,
  ROUND(
    (SUM(
      (cancellation IS NULL AND pickup_time IS NOT NULL)::integer
    )/COUNT(*)::numeric) * 100.0
  ,2) percentage
FROM runner_orders
GROUP BY runner_id
ORDER BY runner_id;
```

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

```sql
----------------- C. Ingredient Optimisation -----------------


-- 1. What are the standard ingredients for each pizza?
WITH cte AS (
  SELECT pizza_id,
    unnest(string_to_array(toppings,', '))::integer topping_id
  FROM pizza_recipes
)
SELECT pn.pizza_name, ARRAY_AGG(topping_name)
FROM cte pr
JOIN pizza_toppings pt USING (topping_id)
JOIN pizza_names pn USING (pizza_id)
GROUP BY pn.pizza_name;
```

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

```sql
-- 2. What was the most commonly added extra?
WITH cte AS (
  SELECT unnest(string_to_array(extras,','))::numeric extra
  FROM customer_orders
  WHERE extras IS NOT NULL
),
freq AS (
  SELECT extra, COUNT(*) cnt
  FROM cte
  GROUP BY extra
)
SELECT extra, cnt
FROM freq
WHERE cnt = (select max(cnt) from freq)
```

	extra numeric	cnt bigint
1	1	4

```sql
-- 3. What was the most common exclusion?
WITH cte AS (
  SELECT unnest(string_to_array(exclusions,','))::numeric exclusion
  FROM customer_orders
  WHERE exclusions IS NOT NULL
),
freq AS (
  SELECT exclusion, COUNT(*) cnt
  FROM cte
  GROUP BY exclusion
)
SELECT exclusion, cnt
FROM freq
WHERE cnt = (select max(cnt) from freq)
```

	exclusion numeric	cnt bigint
1	4	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 (3)
  -- Meat Lovers - Extra Bacon (1)
  -- Meat Lovers - Exclude Cheese(4) , Bacon(1) - Extra Mushroom(6), Peppers (9)

-- SELECT order_id, customer_id, c.pizza_id, c.order_time,
--  CONCAT(
--    pizza_name,
--    CASE WHEN exclusions IS NOT NULL THEN CONCAT(' - Exclude
',string_to_array(exclusions,','),' ')
--    ELSE ''
--    END,
--    CASE WHEN extras IS NOT NULL THEN CONCAT(' - Extra
',string_to_array(extras,','),' ')
--    ELSE ''
--    END
--  ) order_item
-- FROM customer_orders c
-- JOIN pizza_names pn USING(pizza_id)


WITH cte AS (
  SELECT
  ROW_NUMBER() OVER (ORDER BY customer_id) row_id,
  order_id, customer_id, pizza_id,order_time, exclusions, extras
  FROM customer_orders
),
cteExclusion AS (
  SELECT
  ROW_NUMBER() OVER (ORDER BY customer_id) row_id,
  unnest(string_to_array(exclusions,',')) exclusion_id
  FROM customer_orders
),
cteExtra AS (
  SELECT
  ROW_NUMBER() OVER (ORDER BY customer_id) row_id,
  unnest(string_to_array(extras,',')) extra_id
  FROM customer_orders
),
```

```sql
cteExcString AS (
  SELECT row_id, ARRAY_TO_STRING(ARRAY_AGG(pt.topping_name),', ') exclusions
  FROM cteExclusion c
  JOIN pizza_toppings pt
  ON c.exclusion_id::integer = pt.topping_id
  GROUP BY row_id
),
cteExtString AS (
  SELECT row_id, ARRAY_TO_STRING(ARRAY_AGG(pt.topping_name),', ') extras
  FROM cteExtra c
  JOIN pizza_toppings pt
  ON c.extra_id::integer = pt.topping_id
  GROUP BY row_id
),
f_final AS (
  SELECT
    order_id, customer_id, pizza_id,
    CONCAT(
      pizza_name,
      CASE
        WHEN c.exclusions IS NOT NULL
        THEN CONCAT(' - Exclude ',exc.exclusions)
        ELSE ''
      END,
      CASE
        WHEN c.extras IS NOT NULL
        THEN CONCAT(' - Extra ',ext.extras)
        ELSE ''
      END
    ) order_item
  FROM cte c
  LEFT JOIN cteExtString ext USING(row_id)
  LEFT JOIN cteExcString exc USING(row_id)
  JOIN pizza_names pn USING(pizza_id)
)
SELECT * FROM f_final
ORDER BY customer_id, order_id;
```

	order_id integer	customer_id integer	pizza_id integer	order_item text
1	1	101	1	Meatlovers
2	2	101	1	Meatlovers
3	6	101	2	Vegetarian
4	3	102	1	Meatlovers
5	3	102	2	Vegetarian
6	8	102	1	Meatlovers
7	4	103	1	Meatlovers - Exclude Cheese
8	4	103	2	Vegetarian - Exclude Cheese
9	4	103	1	Meatlovers - Exclude Cheese
10	9	103	1	Meatlovers - Exclude Cheese - Extra Bac...
11	5	104	1	Meatlovers - Extra Bacon
12	10	104	1	Meatlovers
13	10	104	1	Meatlovers - Exclude BBQ Sauce, Mushr...
14	7	105	2	Vegetarian - Extra Bacon

```sql
-- 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_orders AS (
    SELECT
        ROW_NUMBER() OVER (ORDER BY order_id,pizza_id,order_time) row_id,
        *
    FROM customer_orders
),
base AS (
    SELECT b.row_id, unnest(string_to_array(r.toppings,', ')) topping_id, 1 cnt
    FROM base_orders b
    JOIN pizza_recipes r USING(pizza_id)
),
extra AS (
    SELECT row_id, unnest(string_to_array(extras,',')) topping_id, 1 cnt
    FROM base_orders
),
exclusion AS (
```

```
    SELECT row_id, unnest(string_to_array(exclusions,',')) topping_id, -1 cnt
    FROM base_orders
),
combined AS (
    SELECT * FROM base
    UNION ALL
    SELECT * FROM extra
    UNION ALL
    SELECT * FROM exclusion
),
grouped AS (
    SELECT row_id, topping_id, SUM(cnt) cnt
    FROM combined
    GROUP BY row_id, topping_id
    HAVING SUM(cnt) > 0
),
named AS (
    SELECT
        g.row_id,
        pt.topping_name,
        g.cnt
    FROM grouped g
    JOIN pizza_toppings pt
    ON g.topping_id::int = pt.topping_id
),
formatted AS (
    SELECT
        row_id,
        CASE
            WHEN cnt > 1
            THEN cnt || 'x' || topping_name
            ELSE topping_name
        END AS ingredient
    FROM named
),
final_string AS (
    SELECT
        row_id,
        STRING_AGG(ingredient, ', ' ORDER BY ingredient) ingredients
    FROM formatted
    GROUP BY row_id
)
SELECT
    row_id, b.order_id, b.customer_id, b.pizza_id, order_time,
```

```
    pn.pizza_name || ': ' || fs.ingredients AS ingredients
FROM final_string fs
JOIN base_orders b USING(row_id)
JOIN pizza_names pn USING(pizza_id)
ORDER BY b.order_id, b.row_id;
```

	row_id bigint	order_id integer	customer_id integer	pizza_id integer	order_time timestamp without time zone	ingredients text
1	1	1	101	1	2020-01-01 18:05:02	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
2	2	2	101	1	2020-01-01 19:00:52	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
3	3	3	102	1	2020-01-02 23:51:23	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
4	4	3	102	2	2020-01-02 23:51:23	Vegetarian: Cheese, Mushrooms, Onions, Peppers, Tomato Sauce, Tomatoes
5	5	4	103	1	2020-01-04 13:23:46	Meatlovers: Bacon, BBQ Sauce, Beef, Chicken, Mushrooms, Pepperoni, Salami
6	6	4	103	1	2020-01-04 13:23:46	Meatlovers: Bacon, BBQ Sauce, Beef, Chicken, Mushrooms, Pepperoni, Salami
7	7	4	103	2	2020-01-04 13:23:46	Vegetarian: Mushrooms, Onions, Peppers, Tomato Sauce, Tomatoes
8	8	5	104	1	2020-01-08 21:00:29	Meatlovers: 2xBacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Sa…
9	9	6	101	2	2020-01-08 21:03:13	Vegetarian: Cheese, Mushrooms, Onions, Peppers, Tomato Sauce, Tomatoes
10	10	7	105	2	2020-01-08 21:20:29	Vegetarian: Bacon, Cheese, Mushrooms, Onions, Peppers, Tomato Sauce, Tomatoes
11	11	8	102	1	2020-01-09 23:54:33	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami
12	12	9	103	1	2020-01-10 11:22:59	Meatlovers: 2xBacon, 2xChicken, BBQ Sauce, Beef, Mushrooms, Pepperoni, Salami
13	13	10	104	1	2020-01-11 18:34:49	Meatlovers: 2xBacon, 2xCheese, Beef, Chicken, Pepperoni, Salami
14	14	10	104	1	2020-01-11 18:34:49	Meatlovers: Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami

```
-- 6. What is the total quantity of each ingredient used in
-- all delivered pizzas sorted by most frequent first?
WITH base_orders AS (
    SELECT
        ROW_NUMBER() OVER (ORDER BY order_id,pizza_id,order_time) row_id,
        *
    FROM customer_orders c
    JOIN runner_orders r USING(order_id)
    WHERE r.pickup_time IS NOT NULL AND r.cancellation IS NULL
),
base AS (
    SELECT b.row_id, unnest(string_to_array(r.toppings,', ')) topping_id, 1 cnt
    FROM base_orders b
    JOIN pizza_recipes r USING(pizza_id)
),
extra AS (
    SELECT row_id, unnest(string_to_array(extras,',')) topping_id, 1 cnt
    FROM base_orders
),
exclusion AS (
    SELECT row_id, unnest(string_to_array(exclusions,',')) topping_id, -1 cnt
    FROM base_orders
),
combined AS (
    SELECT * FROM base
```

```
    UNION ALL
    SELECT * FROM extra
    UNION ALL
    SELECT * FROM exclusion
),
grouped AS (
    SELECT topping_id::int, SUM(cnt) cnt
    FROM combined
    GROUP BY topping_id
    HAVING SUM(cnt) > 0
)
SELECT
    pt.topping_name,
    g.cnt
FROM grouped g
JOIN pizza_toppings pt
    ON g.topping_id = pt.topping_id
ORDER BY g.cnt DESC;
```

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

```
-- for total quantity, we need to add up extras and exclude exclusions
-- assumption: we are assuming that given exclusion list will always contain the
ingredients
-- that are present in that pizza (meaning that pizza count should never go -ve)

---------------- D. Pricing and Ratings ----------------
```

```sql
-- 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?

-- assumption: we are taking only successfull deliveries
SELECT
    SUM(
        CASE
            WHEN p.pizza_name = 'Meatlovers'
            THEN 12
            WHEN p.pizza_name = 'Vegetarian'
            THEN 10
            ELSE 0
        END
    ) as total_cost_earned
FROM customer_orders c
JOIN pizza_names p USING(pizza_id)
JOIN runner_orders r USING(order_id)
WHERE r.cancellation IS NULL;
```

	total_cost_earned bigint
1	138

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

SELECT
    SUM(
        CASE
            WHEN p.pizza_name = 'Meatlovers'
            THEN 12
            WHEN p.pizza_name = 'Vegetarian'
            THEN 10
            ELSE 0
        END
        +
        CASE
            WHEN extras IS NOT NULL
            THEN array_length(string_to_array(extras,','),1)
            ELSE 0
        END
    ) as total_cost_earned
```

```
FROM customer_orders c
JOIN pizza_names p USING(pizza_id)
JOIN runner_orders r USING(order_id)
WHERE r.cancellation IS NULL;
```

	total_cost_earned 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 ratings(
    rating_id SERIAL PRIMARY KEY,
    customer_id INTEGER,
    order_id INTEGER UNIQUE,
    runner_id INTEGER,
    rating INTEGER CHECK (rating BETWEEN 1 AND 5),
    feedback VARCHAR(255)
);
-- remaining: adding foreign key constraints

INSERT INTO ratings (order_id, customer_id, runner_id, rating, feedback)
VALUES
(1, 101, 1, 5, 'Very fast delivery'),
(2, 101, 1, 4, 'Good service'),
(3, 102, 1, 5, 'Excellent runner'),
(4, 103, 2, 3, 'Average delivery'),
(5, 104, 3, 4, 'Polite and quick'),
(7, 105, 2, 5, 'Perfect service'),
(8, 102, 2, 4, 'Good experience'),
(10, 104, 1, 5, 'Outstanding delivery');

SELECT * FROM ratings;
```

	rating_id [PK] integer	customer_id integer	order_id integer	runner_id integer	rating integer	feedback character varying (255)
1	1	101	1	1	5	Very fast delivery
2	2	101	2	1	4	Good service
3	3	102	3	1	5	Excellent runner
4	4	103	4	2	3	Average delivery
5	5	104	5	3	4	Polite and quick
6	6	105	7	2	5	Perfect service
7	7	102	8	2	4	Good experience
8	8	104	10	1	5	Outstanding delivery

```sql
-- 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

SELECT
    c.customer_id,
    c.order_id,
    r.runner_id,
    rt.rating,
    c.order_time,
    r.pickup_time,
    (r.pickup_time - c.order_time)
        AS time_between_order_and_pickup,
    r.duration
        AS delivery_duration_minutes,
    ROUND(r.distance/(r.duration/60.0),2)
        AS avg_speed_kmph,
    COUNT(c.pizza_id)
        AS total_number_of_pizzas
FROM customer_orders c
JOIN runner_orders r USING(order_id)
```

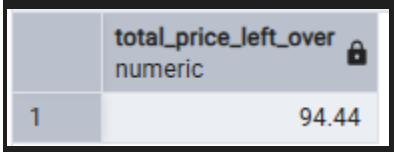
```
JOIN ratings rt USING(order_id)
WHERE r.cancellation IS NULL
GROUP BY
    c.customer_id,
    c.order_id,
    r.runner_id,
    rt.rating,
    c.order_time,
    r.pickup_time,
    r.duration,
    r.distance;
```

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

```
-- 5. Total number of pizzas
-- 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?

-- total price left over = total price got by making pizzas (revenue)
--                         - cost need to be given to runners for their rides
WITH total_revenue_cte AS (
    SELECT
        SUM(
            CASE
                WHEN p.pizza_name = 'Meatlovers'
                THEN 12
                WHEN p.pizza_name = 'Vegetarian'
                THEN 10
                ELSE 0
            END
        ) AS tot_revenue
    FROM customer_orders c
    JOIN pizza_names p USING(pizza_id)
    JOIN runner_orders r USING (order_id)
    WHERE r.cancellation IS NULL
),
runner_cost_cte AS (
```

```
    SELECT SUM(distance * 0.3) runner_cost FROM runner_orders
)
SELECT
    tot_revenue-runner_cost
        AS total_price_left_over
FROM runner_cost_cte, total_revenue_cte;
```



	total_price_left_over numeric
1	94.44