Case Study #2 - Pizza Runner

Danny Ma · May 4, 2021

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

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

Interactive SQL Instance

You can use the embedded DB Fiddle below to easily access these example datasets - this interactive session has everything you need to start solving these questions using SQL.

You can click on the `Edit on DB Fiddle` link on the top right hand corner of the embedded session below and it will take you to a fully functional SQL editor where you can write your own queries to analyse the data.

You can feel free to choose any SQL dialect you'd like to use, the existing Fiddle is using PostgreSQL 13 as default.

Serious SQL students can copy and paste the Schema SQL code below directly into their SQLPad GUI to create permanent tables or they can add a `TEMP` within the table creation steps like we've done throughout the entire course to keep our schemas clean and tidy!

Case Study Questions

This case study has **LOTS** of questions - they are broken up by area of focus including:

- Pizza Metrics
- Runner and Customer Experience
- Ingredient Optimisation
- Pricing and Ratings
- Bonus DML Challenges (DML = Data Manipulation Language)

Each of the following case study questions can be answered using a single SQL statement.

Again, there are many questions in this case study - please feel free to pick and choose which ones you'd like to try!

Before you start writing your SQL queries however - you might want to investigate the data, you may want to do something with some of those `null` values and data types in the `customer_orders` and `runner_orders` tables!

TABLE CREATION SCRIPT

```
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" DATETIME
);

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');
```

```
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');
```

DATA CLEANING AND TRANSFORMATION

1. Cleaning Customer_Orders Table

Replacing explicit null and implicit NULL with blank

```
SELECT * FROM customer_orders;
UPDATE customer_orders
SET exclusions = ''
WHERE exclusions IS NULL
OR exclusions LIKE 'null';

UPDATE customer_orders
SET extras = ''
WHERE extras IS NULL
```

OR extras LIKE 'null';

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

2. Cleaning Runner_Orders Table

Replacing explicit null and implicit NULL with blank

```
UPDATE runner_orders
SET pickup_time= ' '
WHERE pickup_time IS NULL
OR pickup_time LIKE 'null';

UPDATE runner_orders
SET distance= ' '
WHERE distance IS NULL
OR distance LIKE 'null';

UPDATE runner_orders
SET duration= ' '
WHERE duration IS NULL
OR duration LIKE 'null';

UPDATE runner_orders
SET cancellation= ' '
WHERE cancellation IS NULL
OR cancellation LIKE 'null';
```

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

Cleaning distance and duration column i.e removing text like 'km' ,'minutes','minute','mins' with blank space and TRIM the white spaces

Replacing 'km' from distance column

```
UPDATE runner_orders
SET distance = REPLACE(distance,'km','');
```

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

Replacing 'minutes','minute','mins' from duration column

```
UPDATE runner_orders
SET duration = REPlACE(duration,'minutes','');
UPDATE runner_orders
SET duration = REPlACE(duration,'minute','');
UPDATE runner_orders
SET duration = REPlACE(duration,'mins','');
```

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

Changing DataType of columns

```
ALTER TABLE runner_orders
ALTER COLUMN pickup_time DATETIME;

ALTER TABLE runner_orders
ALTER COLUMN distance FLOAT;

ALTER TABLE runner_orders
ALTER COLUMN duration INT;
```

A. Pizza Metrics

1. How many pizzas were ordered?

```
SELECT COUNT(*) pizzas_ordered
FROM customer_orders;
```

pizzas_ordered
14

2. How many unique customer orders were made?

```
SELECT COUNT(DISTINCT customer_id) unique_customer_orders
FROM customer_orders;
```

unique_customer_orders
5

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

```
SELECT  r.runner_id,
         COUNT(c.order_id) successful_orders
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
WHERE r.distance!=0
GROUP BY r.runner_id;
```

runner_id	successful_orders
1	6
2	5
3	1

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

```
ALTER TABlE pizza_names
ALTER COLUMN pizza_name VARCHAR(max);

SELECT p.pizza_name,
        COUNT(1) no_pizzas_delivered
FROM customer_orders c
INNER JOIN pizza_names p
ON c.pizza_id=p.pizza_id
INNER JOIN runner_orders r
ON c.order_id=r.order_id
WHERE r.distance!=0
GROUP BY p.pizza_name;
```

pizza_name	no_pizzas_delivered
Meatlovers	9
Vegetarian	3

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

```
SELECT c.customer_id,
        p.pizza_name,
        COUNT(1) no_pizzas_delivered
FROM customer_orders c
INNER JOIN pizza_names p
ON c.pizza_id=p.pizza_id
INNER JOIN runner_orders r
ON c.order_id=r.order_id
GROUP BY c.customer_id,p.pizza_name;
```

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

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

```
SELECT TOP 1 c.order_id,
       COUNT(*) max_pizzas_delivered
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
INNER JOIN pizza_names p
ON c.pizza_id=p.pizza_id
WHERE r.distance!=0
GROUP BY c.order_id
ORDER BY max_pizzas_delivered DESC;
```

order_id	max_pizzas_delivered
4	3

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

```
SELECT c.customer_id,
       SUM(CASE WHEN c.exclusions IS NOT NULL
                 OR c.extras IS NOT NULL
                 THEN 1
                ELSE 0
            END) at_least_1_change,
        SUM(CASE WHEN c.exclusions IS NULL
                 AND c.extras IS NULL
                 THEN 1
                ELSE 0
            END) no_change
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
WHERE r.distance<>0
GROUP BY c.customer_id;
```

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

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

```
SELECT COUNT(1) both_exclusions_extras
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
WHERE c.exclusions IS NOT NULL
AND c.extras IS NOT NULL
AND r.distance<>0;
```

both_exclusions_extras
1

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

```
SELECT DATEPART(HOUR,order_time) hour_of_the_day,
       COUNT(1) pizzas_ordered
FROM customer_orders
GROUP BY  DATEPART(HOUR,order_time);
```

hour_of_the_day	pizzas_ordered
11	1
13	3
18	3
19	1
21	3
23	3

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

```
SELECT DATENAME(WEEKDAY,order_time) day_of_the_week,
       COUNT(1) pizzas_ordered
FROM customer_orders
GROUP BY DATENAME(WEEKDAY,order_time);
```

day_of_the_week	pizzas_ordered
Friday	1
Saturday	5
Thursday	3
Wednesday	5

B. Runner and Customer Experience

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

```
SELECT DATEPART(WEEK,DATEADD(DAY,-
DATEDIFF(DAY,'2021-01-01',registration_date)%7,registration_date)) as week_no
       ,COUNT(1) no_of_runners_signedup
FROM runners
GROUP BY DATEPART(WEEK,DATEADD(DAY,-
DATEDIFF(DAY,'2021-01-01',registration_date)%7,registration_date));
```

week_no	no_of_runners_signedup
1	2
2	1
3	1

12. 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
       ,AVG(DATEDIFF(MINUTE,c.order_time,r.pickup_time)) avg_pickup_time
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
WHERE r.pickup_time IS NOT NULL
GROUP BY r.runner_id;
```

runner_id	avg_pickup_time
1	15
2	24
3	10

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

```
WITH prep_cte AS (
SELECT  c.order_id
       ,COUNT(1) number_of_pizzas
       ,DATEDIFF(MINUTE,c.order_time,r.pickup_time) prep_time
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
WHERE r.distance<>0
GROUP BY c.order_id,DATEDIFF(MINUTE,c.order_time,r.pickup_time)
)
SELECT  number_of_pizzas
      ,AVG(prep_time) avg_prep_time
FROM prep_cte
GROUP BY number_of_pizzas;
```

number_of_pizzas	avg_prep_time
1	12
2	18
3	30

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

```
SELECT c.customer_id
      ,ROUND(AVG(r.distance),2) avg_distance_travelled
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
WHERE r.distance<>0
GROUP BY c.customer_id;
```

customer_id	avg_distance_travelled
101	20
102	16.73
103	23.4
104	10
105	25

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

```
SELECT MAX(duration)-MIN(duration) [difference]
FROM runner_orders
WHERE distance<>0;
```

difference
30

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

```
SELECT  order_id
       ,runner_id
       ,ROUND(distance*60/duration,2) average_speed
FROM runner_orders
WHERE distance<>0
ORDER BY order_id;
```

order_id	runner_id	average_speed
1	1	37.5
2	1	44.44
3	1	40.2
4	2	35.1
5	3	40
7	2	60
8	2	93.6
10	1	60

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

```
SELECT runner_id
       ,ROUND(COUNT(pickup_time)*100/COUNT(*),2) delivery_percentage
FROM runner_orders
GROUP BY runner_id
ORDER BY runner_id;
```

runner_id	delivery_percentage
1	100
2	75
3	50

C. Ingredient Optimisation

```
Creating temporary tables for splitted values in pizza_recipes

SELECT pizza_id
       ,TRIM(VALUE) toppings
INTO #pizza_recipes_splitted
FROM pizza_recipes
CROSS APPLY STRING_SPLIT(CAST(toppings AS VARCHAR),',');

SELECT * FROM #pizza_recipes_splitted
```

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

```
Creating temporary tables for splitted values in customer_orders
WITH exclusions_cte AS (
SELECT  order_id
       ,customer_id
       ,pizza_id
       ,TRIM(VALUE) exclusions
       ,extras
       ,order_time
FROM customer_orders
OUTER APPLY STRING_SPLIT(exclusions,',')
),
extras_cte AS (
SELECT  order_id
       ,customer_id
       ,pizza_id
       ,exclusions
       ,TRIM(VALUE) extras
       ,order_time
FROM exclusions_cte
OUTER APPLY STRING_SPLIT(extras,',')
)
SELECT *
INTO #customer_orders_splitted
FROM extras_cte;

SELECT * FROM #customer_orders_splitted
```

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

18. What are the standard ingredients for each pizza?

```
WITH recipe_cte AS (
SELECT  n.pizza_name
       ,t.topping_name
FROM #pizza_recipes_splitted r
INNER JOIN pizza_toppings t
ON r.toppings=t.topping_id
INNER JOIN pizza_names n
ON r.pizza_id=n.pizza_id
)
SELECT pizza_name
      ,STRING_AGG(CAST(topping_name 'AS VARCHAR),' , ') standard_ingredients
FROM recipe_cte
GROUP BY pizza_name;
```

pizza_name	standard_ingredients
Meatlovers	Bacon , BBQ Sauce , Beef , Cheese , Chicken , Mushrooms , Pepperoni , Salami
Vegetarian	Cheese , Mushrooms , Onions , Peppers , Tomatoes , Tomato Sauce

19. What was the most commonly added extra?

```
SELECT TOP 1 extras most_common_extra FROM (
SELECT CAST(t.topping_name AS VARCHAR) extras
      ,COUNT(1) no_of_times_added
FROM #customer_orders_splitted c
INNER JOIN pizza_toppings t
ON c.extras=t.topping_id
GROUP BY CAST(t.topping_name AS VARCHAR)) subquery
ORDER BY no_of_times_added DESC;
```

most_common_extra
Bacon

20. What was the most common exclusion?

```
SELECT TOP 1 exclusions most_common_exclusion FROM (
SELECT CAST(t.topping_name AS VARCHAR) exclusions
      ,COUNT(1) no_of_times_added
FROM #customer_orders_splitted c
INNER JOIN pizza_toppings t
ON c.exclusions=t.topping_id
GROUP BY CAST(t.topping_name AS VARCHAR)) subquery
ORDER BY no_of_times_added DESC;
```

most_common_exclusion
Cheese

21. 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

22. 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"`

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

D. Pricing and Ratings

24. 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?

```
SELECT SUM(CASE WHEN pizza_id=1 THEN 12
                WHEN pizza_id=2 THEN 10
           END) total_revenue
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
WHERE r.distance<>0;
```

total_revenue
138

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

26. 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 (
order_id INT,
rating INT CONSTRAINT check_rating CHECK (rating BETWEEN 1 AND 5),
review TEXT
)
INSERT INTO runner_ratings VALUES
(1,3,'Good Food'),
(2,4,'Good food and fast delivery'),
(3,2,'Bad service'),
(4,4,''),
(5,5,'Fantastic service and food'),
(6,NULL,''),
(7,3,''),
(8,4,'Nice service and on time delivery'),
(9,NULL,''),
(10,5,'');

SELECT * FROM runner_ratings;
```

order_id	rating	review
1	3	Good Food
2	4	Good food and fast delivery
3	2	Bad service
4	4	
5	5	Fantastic service and food
6	NULL	
7	3	
8	4	Nice service and on time delivery
9	NULL	
10	5	

27. 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 c.customer_id
       ,c.order_id
       ,r.runner_id
       ,rr.rating
       ,c.order_time
       ,r.pickup_time
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
INNER JOIN runner_ratings rr
ON r.order_id=rr.order_id
WHERE r.distance<>0;
```

customer_id	order_id	runner_id	rating	order_time	pickup_time
101	1	1	3	05:02.0	01-01-2020 18:15:34
101	2	1	4	00:52.0	01-01-2020 19:10:54
102	3	1	2	51:23.0	03-01-2020 00:12:37
102	3	1	2	51:23.0	03-01-2020 00:12:37
103	4	2	4	23:46.0	04-01-2020 13:53:03
103	4	2	4	23:46.0	04-01-2020 13:53:03
103	4	2	4	23:46.0	04-01-2020 13:53:03
104	5	3	5	00:29.0	08-01-2020 21:10:57
105	7	2	3	20:29.0	08-01-2020 21:30:45
102	8	2	4	54:33.0	10-01-2020 00:15:02
104	10	1	5	34:49.0	11-01-2020 18:50:20
104	10	1	5	34:49.0	11-01-2020 18:50:20

28. 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?

```
SELECT SUM(revenue)-SUM(distance)*0.3 actual_revnue FROM (
SELECT      c.order_id
         ,r.distance
         ,SUM(CASE WHEN c.pizza_id=1 THEN 12
                   WHEN c.pizza_id=2 THEN 10
               END) revenue
FROM customer_orders c
INNER JOIN runner_orders r
ON c.order_id=r.order_id
WHERE r.distance<>0
GROUP BY c.order_id, r.distance) subquery
```

actual_revnue
94.44

E. Bonus Questions

If Danny wants to expand his range of pizzas - how would this impact the existing data design? Write an INSERT statement to demonstrate what would happen if a new Supreme pizza with all the toppings was added to the Pizza Runner menu?

```
INSERT INTO pizza_names VALUES
(3,'Supreme');

INSERT INTO pizza_recipes VALUES
(3,(SELECT STRING_AGG(topping_id,',') FROM pizza_toppings));

SELECT n.pizza_id
      ,n.pizza_name
      ,r.toppings
FROM pizza_names n
INNER JOIN pizza_recipes r
ON n.pizza_id=r.pizza_id;
```

pizza_id	pizza_name	toppings
1	Meatlovers	1, 2, 3, 4, 5, 6, 8, 10
2	Vegetarian	4, 6, 7, 9, 11, 12
3	Supreme	1,2,3,4,5,6,7,8,9,10,11,12
3	Supreme	1,2,3,4,5,6,7,8,9,10,11,12
3	Supreme	1,2,3,4,5,6,7,8,9,10,11,12
3	Supreme	1,2,3,4,5,6,7,8,9,10,11,12