**SQL Case Study #2 (Week 2) - Pizza Runner**

Note: All source material and respected credit is from: <https://8weeksqlchallenge.com/>

Online SQL instance used to test queries: <https://www.db-fiddle.com/f/7VcQKQwsS3CTkGRFG7vu98/65>

Additional Note: This case study ramps up in difficulty significantly compared to case study 1.

This is especially true once reaching section C.

**Table of Contents:**

1. [Dataset Structure](https://github.com/TheCaptainFalcon/8wksql-cs2#data)
2. [Cleaned Dataset](https://github.com/TheCaptainFalcon/8wksql-cs2#clean_data)
3. [Optional: Using Python to clean](https://github.com/TheCaptainFalcon/8wksql-cs2#clean_data_python)
4. [Entity Relationship Diagram](https://github.com/TheCaptainFalcon/8wksql-cs2#diagram)
5. [Case Study Questions + Answers](https://github.com/TheCaptainFalcon/8wksql-cs2#questions)
6. [Bonus Questions + Answers](https://github.com/TheCaptainFalcon/8wksql-cs2#bonus)

**Dataset Structure:**

Note: The original data was built around PostgreSQL, but was swapped to fit MySQL syntax.

CREATE SCHEMA 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');

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');

**Cleaned Dataset:**

**Cleaned version of customer\_orders:**

In this table, we remove all the blank spaces and replace all stringed (null/NaN) values as a NULL value for constistency.

CREATE TEMPORARY TABLE cust\_orders

SELECT

order\_id,

customer\_id,

pizza\_id,

CASE

WHEN exclusions = '' THEN NULL

WHEN exclusions = 'null' THEN NULL

ELSE exclusions

END AS exclusions\_cleaned,

CASE

WHEN extras = '' THEN null

WHEN extras = 'NaN' THEN null

ELSE extras

END AS extras\_cleaned,

order\_time

FROM customer\_orders;

**Cleaned version of runner\_orders:**

This table is used to prime the existing data and remove the text from the stringed numerical values.

To provide context, we change the names to include the the measurements of the values.

CREATE TEMPORARY TABLE runner\_orders\_pre

SELECT

order\_id,

runner\_id,

CASE

WHEN pickup\_time = 'null' THEN null

ELSE pickup\_time

END AS pick\_up\_time,

CASE

WHEN distance = 'null' THEN null

ELSE regexp\_replace(distance, '[a-z]+', '')

END AS distance\_km,

CASE

WHEN duration = 'null' THEN null

ELSE regexp\_replace(duration, '[a-z]+', '')

END AS duration\_mins,

CASE

WHEN cancellation = '' THEN null

WHEN cancellation = 'null' THEN null

ELSE cancellation

END AS cancellation

FROM runner\_orders;

From here, we change the data type of the stringed numbers to decimals or integers to enable numerical functions and aggregation capabilities.

CREATE TEMPORARY TABLE runner\_orders\_post

SELECT

order\_id,

runner\_id,

pick\_up\_time,

CAST(distance\_km AS DECIMAL(3,1)) AS distance\_km,

CAST(duration\_mins AS SIGNED INT) AS duration\_mins,

cancellation

FROM runner\_orders\_pre;

**(Optional) Cleaned version of pizza\_recipes**

Note: The Pizza\_Recipes dataset originally contained comma-separated-values within the toppings column by pizza\_id.

**The Thought Process:**

Having any form of a list within a cell is not a great way to proceed from a data or analytical standpoint.

For one, that data would probably be set as a string or varchar of some sorts and even if we were to cast to a signed/unsigned INT, it would be difficult to utilize numerical aggregation functions or sorting capabilities.

While it is true, we could probably use FIND\_IN\_SET, but if we look at it from a large/scalable dataset standpoint, it cant be optimized as indexes cannot be used.

**Possible Solutions:**

The initial approach was to fix this within SQL; I found that there were available solutions, but with caveats.

1. The first one was simple. You could use a CROSS APPLY and a STRING SPLIT to set individual pizza\_ids with every appropriate topping as a separate value, however this wasn't an option available in MySQL.
2. The second approach was to look at how other people solved this issue. While I did find a solution, it was extremely complicated espeically for a task that is considered simple. I really felt that this could easily be solved in a programming language (Work smarter, not harder mentality)
3. The third approach was to create a separate table and insert new values to see fit, but there was the possiblilty that in a real life scenario, being granted access to creating/inserting data may or may not be possible (not sure about a temp table tho).

Although the third approach could work, I figured it would be good practice and interesting to try out Pandas andd to modify from a dataframe view (it also seemed easier to do, as well)

**The Solution I Chose:**

Found that Pandas had a function called EXPLODE() that did exactly what cross apply + string split would have done in SQL Server. Called this table clean\_pizza\_recipes and this was the table used for the queries in section C.

Future reference: <https://stackoverflow.com/questions/12680754/split-explode-pandas-dataframe-string-entry-to-separate-rows>

**Exported the table into a CSV to load into Jupyter.**

import numpy as np

import pandas as pd

pizza = pd.read\_csv('pizza\_recipe\_copy.csv')

\*\* = kwargs

toppings is the column name of the multi-value list

Here we are assigning an object key-value pair for the topping name = topping values separated by the comma delimiter

x = pizza.assign(\*\*('toppings' : pizza['toppings'].str.split(',')))

**Here is how it looks after assigning.**

x

| **pizza\_id** | **toppings** |
| --- | --- |
| 1 | [1, 2, 3, 4, 5, 6, 8, 10] |
| 2 | [4, 6, 7, 9, 11, 12] |

**We then use the explode method to separate each pair into separate rows**

x\_clean = x.explode('toppings')

**This is how it looks post explode.**

x\_clean['toppings']

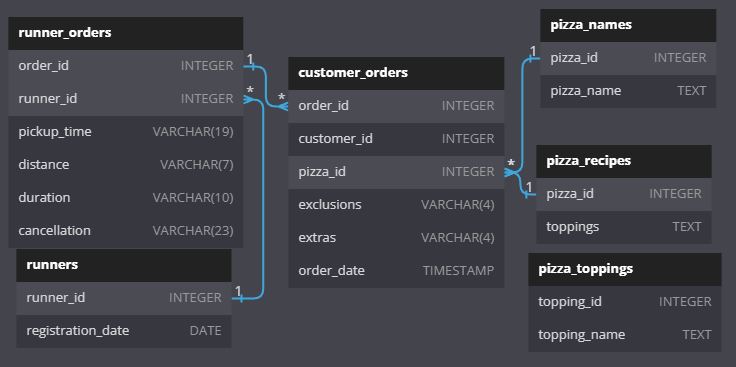
| **pizza\_id** | **toppings** |
| --- | --- |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 1 | 5 |
| 1 | 6 |
| 1 | 8 |
| 1 | 10 |
| 2 | 4 |
| 2 | 6 |
| 2 | 7 |
| 2 | 9 |
| 2 | 11 |
| 2 | 12 |

**In order to utilize this data, we create a new and updated CSV to import into the MySQL database.**

x\_clean.to\_csv('clean\_ingredients.csv')

**Entity Relationship Diagram View**

Original image source: <https://dbdiagram.io/d/5f3e085ccf48a141ff558487/?utm_source=dbdiagram_embed&utm_medium=bottom_open>

[](https://github.com/TheCaptainFalcon/8wksql-cs2/blob/master/ERD-CS2.JPG)

**Case Study Questions:**

**A. Pizza Metrics**

**1. How many pizzas were ordered?**

**Query #1**

SELECT COUNT(pizza\_id) AS pizza\_count

FROM customer\_orders;

| **pizza\_count** |
| --- |
| 14 |

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

**Query #2**

SELECT COUNT(distinct order\_id) AS unique\_pizza\_count

FROM customer\_orders;

| **unique\_pizza\_count** |
| --- |
| 10 |

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

**Query #3**

SELECT

runner\_id,

COUNT(order\_id) AS order\_count

FROM runner\_orders\_post

WHERE duration\_mins IS NOT NULL

GROUP BY runner\_id;

| **runner\_id** | **order\_count** |
| --- | --- |
| 1 | 4 |
| 2 | 3 |
| 3 | 1 |

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

**Query #4**

WITH pizza\_counter AS (

SELECT

c.order\_id,

COUNT(c.pizza\_id) AS pizza\_count

FROM cust\_orders c

LEFT JOIN runner\_orders\_post r

ON c.order\_id = r.order\_id

WHERE duration\_mins IS NOT NULL

GROUP BY c.order\_id

)

SELECT

SUM(pizza\_count) AS total\_pizza\_count

FROM pizza\_counter;

| **total\_pizza\_count** |
| --- |
| 12 |

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

**Query #5**

WITH pizza\_counter\_1 AS (

SELECT

customer\_id,

COUNT(pizza\_id) AS pizza\_count

FROM customer\_orders

WHERE pizza\_id = 1

GROUP BY customer\_id

),

pizza\_counter\_2 AS (

SELECT

customer\_id,

COUNT(pizza\_id) AS pizza\_count

FROM customer\_orders

WHERE pizza\_id = 2

GROUP BY customer\_id

)

SELECT DISTINCT

pc1.customer\_id,

pc1.pizza\_count AS total\_meatlovers,

pc2.pizza\_count AS total\_vegetarian

FROM pizza\_counter\_1 pc1, pizza\_counter\_2 pc2

ORDER BY 1;

| **customer\_id** | **total\_meatlovers** | **total\_vegetarian** |
| --- | --- | --- |
| 101 | 2 | 1 |
| 102 | 2 | 1 |
| 103 | 3 | 1 |
| 104 | 3 | 1 |

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

**Query #6**

SELECT

order\_id,

COUNT(pizza\_id) AS total\_pizzas

FROM customer\_orders

GROUP BY order\_id

ORDER BY total\_orders DESC;

| **order\_id** | **total\_orders** |
| --- | --- |
| 4 | 3 |
| 3 | 2 |
| 10 | 2 |
| 1 | 1 |
| 2 | 1 |
| 5 | 1 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 9 | 1 |

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

**Query #7**

WITH pizza\_changes\_counter AS (

SELECT

co.customer\_id,

CASE

WHEN co.exclusions\_cleaned LIKE '%' OR co.extras\_cleaned LIKE '%' THEN 1

ELSE 0

END AS pizza\_change\_count,

CASE

WHEN co.exclusions\_cleaned IS NULL AND co.extras\_cleaned IS NULL THEN 1

WHEN co.exclusions\_cleaned IS NULL AND co.extras\_cleaned = 'NaN' THEN 1

ELSE 0

END AS pizza\_no\_change\_count

FROM cust\_orders co

LEFT JOIN runner\_orders\_post ro

ON co.order\_id = ro.order\_id

WHERE ro.duration\_mins IS NOT NULL

)

SELECT

customer\_id,

SUM(pizza\_change\_count) AS total\_pizzas\_with\_changes,

SUM(pizza\_no\_change\_count) AS total\_pizzas\_without\_changes

FROM pizza\_changes\_counter

GROUP BY customer\_id;

| **customer\_id** | **total\_pizzas\_with\_changes** | **total\_pizzas\_without\_changes** |
| --- | --- | --- |
| 101 | 0 | 2 |
| 102 | 1 | 2 |
| 103 | 3 | 0 |
| 104 | 3 | 0 |
| 105 | 1 | 0 |

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

**Query #8**

WITH pizzas\_changes\_counter AS (

SELECT

co.customer\_id,

CASE

WHEN co.exclusions\_cleaned LIKE '%' AND co.extras\_cleaned LIKE '%' THEN 1

ELSE 0

END AS pizza\_change\_count

FROM cust\_orders co

LEFT JOIN runner\_orders\_post ro

ON co.order\_id = ro.order\_id

WHERE ro.duration\_mins IS NOT NULL

)

SELECT

customer\_id,

SUM(pizza\_change\_count) AS pizzas\_with\_changes

FROM pizzas\_changes\_counter

GROUP BY customer\_id

ORDER BY pizzas\_with\_changes DESC;

| **customer\_id** | **pizzas\_with\_changes** |
| --- | --- |
| 104 | 1 |
| 101 | 0 |
| 102 | 0 |
| 103 | 0 |
| 105 | 0 |

Normally, the answer would be 2, but because the question specified 'delivered', we would exclude the orders that were cancelled.

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

**Query #9**

SELECT

COUNT(order\_id) AS order\_count,

HOUR(order\_time) AS hour

FROM cust\_orders

GROUP BY hour;

| **order\_count** | **hour** |
| --- | --- |
| 3 | 18 |
| 1 | 19 |
| 3 | 23 |
| 3 | 13 |
| 3 | 21 |
| 1 | 11 |

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

**Query #10**

WITH orders\_by\_day AS (

SELECT

COUNT(order\_id) AS order\_count,

WEEKDAY(order\_time) AS day

FROM cust\_orders

GROUP BY day

ORDER BY day

)

SELECT

order\_count,

CASE

WHEN day = 0 THEN 'Monday'

WHEN day = 1 THEN 'Tuesday'

WHEN day = 2 THEN 'Wednesday'

WHEN day = 3 THEN 'Thursday'

WHEN day = 4 THEN 'Friday'

WHEN day = 5 THEN 'Saturday'

WHEN day = 6 THEN 'Sunday'

END AS day

FROM orders\_by\_day;

| **order\_count** | **day** |
| --- | --- |
| 5 | Wednesday |
| 3 | Thursday |
| 1 | Friday |
| 5 | Saturday |

**B. Runner and Customer Experience**

**1. How many runner signed up for each 1 week period? (ie.week starts 2021-01-01)**

**Query #1**

SELECT

COUNT(runner\_id) AS runner\_count,

WEEK(registration\_date) AS week

FROM runners

GROUP BY week;

| **runner\_count** | **week** |
| --- | --- |
| 1 | 0 |
| 2 | 1 |
| 1 | 2 |

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

**Query #2**

SELECT

r.runner\_id,

AVG(MINUTE(TIMEDIFF(r.pick\_up\_time, c.order\_time))) AS time\_mins

FROM cust\_orders c

LEFT JOIN runner\_orders\_post r

ON c.order\_id = r.order\_id

GROUP BY r.runner\_id;

| **runner\_id** | **time\_mins** |
| --- | --- |
| 1 | 15.3333 |
| 2 | 23.4000 |
| 3 | 10.0000 |

Note: TIMEDIFF(later time, earlier time)

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

Based on the limited data, there is a possible relationship.

As the number of pizzas increases per order, the time to prepare an order increases. This is shown with order 4 with customer id 103 ranking at the highest in time prep of 29 mins. There seems to be a slight variance with orders consisting of 2 pizzas taking anywhere from 15 - 21 mins, while an order of 1 pizza takes as short as 10 minutes.

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

**Query #4**

SELECT

c.customer\_id,

AVG(r.distance\_km) AS avg\_dist\_km

FROM cust\_orders c

LEFT JOIN runner\_orders\_post r

ON c.order\_id = r.order\_id

GROUP BY c.customer\_id;

| **customer\_id** | **avg\_dist\_km** |
| --- | --- |
| 101 | 20.00000 |
| 102 | 16.73333 |
| 103 | 23.40000 |
| 104 | 10.00000 |
| 105 | 25.00000 |

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

**Query #5**

SELECT

MAX(duration\_mins) - MIN(duration\_mins) AS delivery\_time\_diff

FROM runner\_orders\_post;

| **delivery\_time\_diff** |
| --- |
| 30 |

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

**Query #6**

SELECT

runner\_id,

AVG(distance\_km),

AVG(duration\_mins)

FROM runner\_orders\_post

GROUP BY runner\_id;

| **runner\_id** | **avg(distance\_km)** | **avg(duration\_mins)** |
| --- | --- | --- |
| 1 | 15.85000 | 22.2500 |
| 2 | 23.93333 | 26.6667 |
| 3 | 10.00000 | 15.0000 |

**PART 2 ANSWER:**

Yes, as expected.

As the distance increases, the time it takes to deliver an order, increases as well.

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

**Query #7**

WITH cancellation\_counter AS (

SELECT

runner\_id,

CASE

WHEN cancellation IS NULL OR cancellation = 'NaN' THEN 1

ELSE 0

END AS no\_cancellation\_count,

CASE

WHEN cancellation IS NOT NULL OR cancellation != 'NaN' THEN 1

ELSE 0

END AS cancellation\_count

FROM runner\_orders\_post

)

SELECT

runner\_id,

SUM(no\_cancellation\_count) / (SUM(no\_cancellation\_count) + SUM(cancellation\_count))\*100 AS delivery\_success\_percentage

FROM cancellation\_counter

GROUP BY runner\_id;

| **runner\_id** | **delivery\_success\_percentage** |
| --- | --- |
| 1 | 100.0000 |
| 2 | 75.0000 |
| 3 | 50.0000 |

**C. Ingredient Optimisation**

Note: At this point, the pizza\_recipes data that was cleaned using Python (Pandas) come into light here.

Questions 5 and 6 were obnoxiously difficult, as the trend of utilizing a max of roughly 3 multi-step queries rose to about 7-8.

Furthermore, complex nested functions that could normally be used in a programming language could not be used in the same manner in SQL (typical nuances of working in any new language/tool & because SQL is a query language) and each major aggregation/operation had to be split into multiple separate subqueries.

Granted, it is possible that the queries on my end were not optimized to further reduce the number of steps required, however this was also due to working in "chunks" prior to reaching the final solution.

By working in this manner, data can be checked to ensure accuracy within the process, rather than after.

In addition, taking into consideration a real life scenario and the concept of future proofing, as data comes in, each chunk allows for direct debugging and modifying to fit new data.

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

**Query #1**

SELECT

pizza\_id,

topping\_name

FROM clean\_pizza\_recipes cpr

LEFT JOIN pizza\_toppings pt

ON cpr.toppings = pt.topping\_id;

| **pizza\_id** | **topping\_name** |
| --- | --- |
| 1 | Bacon |
| 1 | BBQ Sauce |
| 1 | Beef |
| 1 | Cheese |
| 1 | Chicken |
| 1 | Mushrooms |
| 1 | Pepperoni |
| 1 | Salami |
| 2 | Cheese |
| 2 | Mushrooms |
| 2 | Onions |
| 2 | Peppers |
| 2 | Tomatoes |
| 2 | Tomato Sauce |

**ANSWER:**

Pizza 1 (Meatlovers) = [Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami]

Pizza 2 (Vegetarian) = [Cheese, Mushrooms, Onions, Peppers, Tomatoes, Tomato Sauce]

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

**Query #2**

SELECT

extras\_cleaned as extras,

COUNT(extras\_cleaned) AS extras\_counted

FROM cust\_orders

WHERE extras\_cleaned LIKE '%'

GROUP BY extras\_cleaned;

| **extras** | **extras\_counted** |
| --- | --- |
| 1 | 2 |
| 1, 5 | 1 |
| 1, 4 | 1 |

**3. What was the most common exclusion?**

**Query #3**

SELECT

exclusions\_cleaned,

COUNT(exclusions\_cleaned) AS exclusions\_count

FROM cust\_orders

WHERE exclusions\_cleaned LIKE '%'

GROUP BY exclusions\_cleaned;

| **exclusions\_cleaned** | **exclusions\_count** |
| --- | --- |
| 4 | 4 |
| 2, 6 | 1 |

**4. Generate an order item for each record in the customers\_orders table in the format of one of the following:**

**Meat Lovers**

SELECT order\_id

FROM cust\_orders

WHERE pizza\_id = 1

GROUP BY order\_id;

| **order\_id** |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 8 |
| 9 |
| 10 |

**Meat Lovers - Exclude Beef**

SELECT order\_id

FROM cust\_orders

WHERE pizza\_id = 1 AND exclusions\_cleaned = 3 OR exclusions\_cleaned LIKE '%3%'

GROUP BY order\_id

Note: There is no query result, as the dataset does not have any existing exclusions with Beef (3), oddly enough.

**Meat Lovers - Extra Bacon**

SELECT order\_id

FROM cust\_orders

WHERE pizza\_id = 1 AND extras\_cleaned = 1 OR extras\_cleaned LIKE '%1%'

GROUP BY order\_id

| **order\_id** |
| --- |
| 5 |
| 7 |
| 9 |
| 10 |

**Meat Lovers - Exclude Cheese, Bacon - Extra Mushroom, Peppers**

WITH exc\_ext\_counter AS (

SELECT

order\_id,

CASE

WHEN exclusions\_cleaned IN (1,4) OR exclusions\_cleaned LIKE '%1%' OR exclusions\_cleaned LIKE '%4%' THEN 1

WHEN extras\_cleaned IN (6,9) AND extras\_cleaned LIKE '%6%' OR extras\_cleaned LIKE '%9%' THEN 1

END AS exc\_ext\_count

FROM cust\_orders

WHERE pizza\_id = 1

)

SELECT order\_id

FROM exc\_ext\_counter

WHERE exc\_ext\_count = 1

GROUP BY order\_id;

| **order\_id** |
| --- |
| 4 |
| 9 |

**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"**

Starting off, what I am trying to do is to use an IF statement to create an indicator for whether a row within the exclusions or extras columns contains a comma delimiter. This will be used later to extract the values and to determine whether those values 'stack' on top of the standard ingredients of the pizza, thus resulting in a 2x.

It is also worth noting, that while this would not work if there were more than 2 values (within the extras/exclusions column), the proof of concept could stil be utilized to fit this edge case.

WITH exc\_ext\_bool AS (

SELECT

order\_id,

pizza\_id,

exclusions\_cleaned,

extras\_cleaned,

IF(LOCATE(',', exclusions\_cleaned), TRUE, FALSE) AS exclusions\_bool,

IF(LOCATE(',', extras\_cleaned), TRUE, FALSE) AS extras\_bool

FROM cust\_orders

),

| **order\_id** | **pizza\_id** | **exclusions\_cleaned** | **extras\_cleaned** | **exclusions\_bool** | **extras\_bool** |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | null | null | 0 | 0 |
| 2 | 1 | null | null | 0 | 0 |
| 3 | 1 | null | null | 0 | 0 |
| 3 | 2 | null | null | 0 | 0 |
| 4 | 1 | 4 | null | 0 | 0 |
| 4 | 1 | 4 | null | 0 | 0 |
| 4 | 2 | 4 | null | 0 | 0 |
| 5 | 1 | null | 1 | 0 | 0 |
| 6 | 2 | null | null | 0 | 0 |
| 7 | 2 | null | 1 | 0 | 0 |
| 8 | 1 | null | null | 0 | 0 |
| 9 | 1 | 4 | 1, 5 | 0 | 1 |
| 10 | 1 | null | null | 0 | 0 |
| 10 | 1 | 2, 6 | 1, 4 | 1 | 1 |

**We now see that the exclusions\_bool and the extras\_bool act as our indicators.**

1 = TRUE

2 = FALSE

**From those indicators, we use the substring\_index to extract the values from the left and right of the comma delimiter for those extras/exclusions that have a 1 value from the columns we created.**

In the instances where this applies, another set of columns are used to house the extracted values.

**Note: The following are part of the existing query, not as separate queries.**

The formatting is displayed in this manner to better showcase everything that is going on in 'chunks.'

base\_exc\_ext AS (

SELECT

eeb.order\_id,

eeb.pizza\_id,

pn.pizza\_name,

IF(LOCATE(0, extras\_bool), extras\_cleaned, NULL) AS base\_extras,

IF(LOCATE(0, exclusions\_bool), exclusions\_cleaned, NULL) AS base\_exclusions,

IF(LOCATE(1, exclusions\_bool), SUBSTRING\_INDEX(exclusions\_cleaned, ',', 1), NULL) AS exclusions\_1,

IF(LOCATE(1, exclusions\_bool), SUBSTRING\_INDEX(exclusions\_cleaned, ',', -1), NULL) AS exclusions\_2,

IF(LOCATE(1, extras\_bool), SUBSTRING\_INDEX(extras\_cleaned, ',', 1), NULL) AS extras\_1,

IF(LOCATE(1, extras\_bool), SUBSTRING\_INDEX(extras\_cleaned, ',', -1), NULL) AS extras\_2

FROM exc\_ext\_bool eeb

INNER JOIN pizza\_names pn

ON eeb.pizza\_id = pn.pizza\_id

),

| **order\_id** | **pizza\_id** | **pizza\_name** | **base\_exclusions** | **base\_extras** | **exclusions\_1** | **exclusions\_2** | **extras\_1** | **extras\_2** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | Meatlovers |  |  |  |  |  |  |
| 2 | 1 | Meatlovers |  |  |  |  |  |  |
| 3 | 1 | Meatlovers |  |  |  |  |  |  |
| 3 | 2 | Vegetarian |  |  |  |  |  |  |
| 4 | 1 | Meatlovers | 4 |  |  |  |  |  |
| 4 | 1 | Meatlovers | 4 |  |  |  |  |  |
| 4 | 2 | Vegetarian | 4 |  |  |  |  |  |
| 5 | 1 | Meatlovers |  | 1 |  |  |  |  |
| 6 | 2 | Vegetarian |  | null |  |  |  |  |
| 7 | 2 | Vegetarian |  | 1 |  |  |  |  |
| 8 | 1 | Meatlovers |  | null |  |  |  |  |
| 9 | 1 | Meatlovers | 4 |  |  |  | 1 | 5 |
| 10 | 1 | Meatlovers |  | null |  |  |  |  |
| 10 | 1 | Meatlovers |  |  | 2 | 6 | 1 | 4 |

**For whatever reason, db-fiddle does a terrible job with displaying the correct differentiation between 'null' and NULL, which explains the blank spaces, which I am not going back to fix everytime.**

The blank spaces should display null in all of them.

**What follows next in the series of nearly identical queries, is where we join the topping\_id with the values that we extracted from the previous step. This will help us in determining the topping names associated with the topping\_id.**

m1 AS (

SELECT

order\_id,

pizza\_id,

pizza\_name,

base\_exclusions,

base\_extras,

exclusions\_1,

exclusions\_2,

extras\_1,

extras\_2,

topping\_name AS exclusions\_1\_txt

FROM base\_exc\_ext

LEFT JOIN pizza\_toppings pt

ON pt.topping\_id = exclusions\_1

),

m2 AS (

SELECT

order\_id,

pizza\_id,

pizza\_name,

base\_exclusions,

base\_extras,

exclusions\_1,

exclusions\_2,

extras\_1,

extras\_2,

exclusions\_1\_txt,

topping\_name AS exclusions\_2\_txt

FROM m1

LEFT JOIN pizza\_toppings pt

ON pt.topping\_id = exclusions\_2

),

m3 AS (

SELECT

order\_id,

pizza\_id,

pizza\_name,

base\_exclusions,

base\_extras,

exclusions\_1,

exclusions\_2,

extras\_1,

extras\_2,

exclusions\_1\_txt,

exclusions\_2\_txt,

topping\_name AS extras\_1\_txt

FROM m2

LEFT JOIN pizza\_toppings pt

ON pt.topping\_id = extras\_1

),

m4 AS (

SELECT

order\_id,

pizza\_id,

pizza\_name,

base\_exclusions,

base\_extras,

exclusions\_1,

exclusions\_2,

extras\_1,

extras\_2,

exclusions\_1\_txt,

exclusions\_2\_txt,

extras\_1\_txt,

topping\_name AS extras\_2\_txt

FROM m3

LEFT JOIN pizza\_toppings pt

ON pt.topping\_id = extras\_2

),

m5 AS (

SELECT

order\_id,

pizza\_id,

pizza\_name,

base\_exclusions,

base\_extras,

exclusions\_1,

exclusions\_2,

extras\_1,

extras\_2,

exclusions\_1\_txt,

exclusions\_2\_txt,

extras\_1\_txt,

extras\_2\_txt,

topping\_name AS base\_exclusions\_1

FROM m4

LEFT JOIN pizza\_toppings pt

ON pt.topping\_id = base\_exclusions

),

m6 AS (

SELECT

order\_id,

pizza\_id,

pizza\_name,

base\_exclusions,

base\_extras,

exclusions\_1,

exclusions\_2,

extras\_1,

extras\_2,

exclusions\_1\_txt,

exclusions\_2\_txt,

extras\_1\_txt,

extras\_2\_txt,

base\_exclusions\_1,

topping\_name AS base\_extras\_1

FROM m5

LEFT JOIN pizza\_toppings pt

ON pt.topping\_id = base\_extras

),

**Here we are seeing whether any ingredients stack and assigning them a 2x through concatination.**

abc\_exc\_ext AS (

SELECT

order\_id,

pizza\_id,

pizza\_name,

base\_exclusions,

base\_extras,

base\_extras\_1,

exclusions\_1,

exclusions\_2,

extras\_1,

extras\_2,

CASE

WHEN base\_exclusions\_1 IS NULL AND COALESCE(exclusions\_1\_txt, exclusions\_2\_txt) IS NOT NULL THEN CONCAT(exclusions\_1\_txt, ', ', exclusions\_2\_txt)

WHEN base\_exclusions\_1 IS NOT NULL THEN CONCAT(base\_exclusions\_1)

WHEN COALESCE(base\_exclusions\_1, exclusions\_1\_txt, exclusions\_2\_txt) IS NOT NULL THEN CONCAT(base\_exclusions\_1, ', ', exclusions\_1\_txt, ', ', exclusions\_2\_txt)

END AS exclusions\_list,

CASE

WHEN base\_extras\_1 IS NULL AND COALESCE(extras\_1\_txt, extras\_2\_txt) IS NOT NULL and pizza\_id = 1 AND extras\_1 in (1,2,3,4,5,6,8,10) AND extras\_2 IN (1,2,3,4,5,6,8,10) THEN CONCAT('2x ', extras\_1\_txt, ', ', '2x ',extras\_2\_txt)

WHEN base\_extras\_1 IS NOT NULL AND pizza\_id = 1 AND base\_extras IN (1,2,3,4,5,6,7,10) THEN CONCAT('2x ', base\_extras\_1)

WHEN base\_extras\_1 IS NOT NULL THEN base\_extras\_1

END AS extras\_list

FROM m6

)

| **order\_id** | **pizza\_id** | **pizza\_name** | **base\_exclusions** | **base\_extras** | **base\_extras\_1** | **exclusions\_1** | **exclusions\_2** | **extras\_1** | **extras\_2** | **exclusions\_list** | **extras\_list** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | Meatlovers |  |  |  |  |  |  |  |  |  |
| 2 | 1 | Meatlovers |  |  |  |  |  |  |  |  |  |
| 3 | 1 | Meatlovers |  |  |  |  |  |  |  |  |  |
| 3 | 2 | Vegetarian |  |  |  |  |  |  |  |  |  |
| 4 | 1 | Meatlovers | 4 |  |  |  |  |  |  | Cheese |  |
| 4 | 1 | Meatlovers | 4 |  |  |  |  |  |  | Cheese |  |
| 4 | 2 | Vegetarian | 4 |  |  |  |  |  |  | Cheese |  |
| 5 | 1 | Meatlovers |  | 1 | Bacon |  |  |  |  |  | 2x Bacon |
| 6 | 2 | Vegetarian |  | null |  |  |  |  |  |  |  |
| 7 | 2 | Vegetarian |  | 1 | Bacon |  |  |  |  |  | Bacon |
| 8 | 1 | Meatlovers |  | null |  |  |  |  |  |  |  |
| 9 | 1 | Meatlovers | 4 |  |  |  |  | 1 | 5 | Cheese | 2x Bacon, 2x Chicken |
| 10 | 1 | Meatlovers |  | null |  |  |  |  |  |  |  |
| 10 | 1 | Meatlovers |  |  |  | 2 | 6 | 1 | 4 | BBQ Sauce, Mushrooms | 2x Bacon, 2x Cheese |

**This is the final product, where we remove all the excess columns into a summarized query.**

SELECT

order\_id,

CASE

WHEN exclusions\_list IS NOT NULL AND extras\_list IS NULL THEN CONCAT(pizza\_name, ' - ', ' |Exclude| ', exclusions\_list)

WHEN extras\_list IS NOT NULL AND exclusions\_list IS NULL THEN CONCAT(pizza\_name, ' - ', ' |Extras| ' , extras\_list)

WHEN COALESCE(exclusions\_list, extras\_list) IS NULL THEN pizza\_name

WHEN COALESCE(exclusions\_list, extras\_list) IS NOT NULL THEN CONCAT(pizza\_name, ' - ', ' |Exclude| ', exclusions\_list, ' |Extras| ', extras\_list)

END AS pizza\_type

FROM abc\_exc\_ext;

| **order\_id** | **pizza\_type** |
| --- | --- |
| 1 | Meatlovers |
| 2 | Meatlovers |
| 3 | Meatlovers |
| 3 | Vegetarian |
| 4 | Meatlovers - |
| 4 | Meatlovers - |
| 4 | Vegetarian - |
| 5 | Meatlovers - |
| 6 | Vegetarian |
| 7 | Vegetarian - |
| 8 | Meatlovers |
| 9 | Meatlovers - |
| 10 | Meatlovers |
| 10 | Meatlovers - |

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

WITH delivered\_bool AS (

SELECT

c.order\_id,

c.pizza\_id,

c.exclusions\_cleaned,

c.extras\_cleaned,

IF(LOCATE(',', exclusions\_cleaned), TRUE, FALSE) AS exc\_bool,

IF(LOCATE(',', extras\_cleaned), TRUE, FALSE) AS ext\_bool

FROM cust\_orders c

LEFT JOIN runner\_orders\_post r

ON c.order\_id = r.order\_id

WHERE duration\_mins IS NOT NULL

),

exc\_ext\_list AS (

SELECT

order\_id,

pizza\_id,

CASE

WHEN exc\_bool = 0 THEN exclusions\_cleaned

END AS base\_exc,

CASE

WHEN ext\_bool = 0 THEN extras\_cleaned

END AS base\_ext,

CASE

WHEN exc\_bool = 1 THEN SUBSTRING\_INDEX(exclusions\_cleaned, ',', 1)

END AS exc\_1,

CASE

WHEN exc\_bool = 1 THEN SUBSTRING\_INDEX(exclusions\_cleaned, ',', -1)

END AS exc\_2,

CASE

WHEN ext\_bool = 1 THEN SUBSTRING\_INDEX(extras\_cleaned, ',', 1)

END AS ext\_1,

CASE

WHEN ext\_bool = 1 THEN SUBSTRING\_INDEX(extras\_cleaned, ',', -1)

END AS ext\_2

FROM delivered\_bool

)

SELECT

order\_id,

pizza\_id,

base\_exc,

base\_ext,

exc\_1,

exc\_2,

ext\_1,

ext\_2

FROM exc\_ext\_list;

WITH topping\_list AS (

SELECT

cr.pizza\_id,

cr.toppings,

pt.topping\_name

FROM clean\_pizza\_recipes cr

LEFT JOIN pizza\_toppings pt

ON cr.toppings = pt.topping\_id

),

pizza\_counter AS (

SELECT

c.order\_id,

c.pizza\_id,

COUNT(c.pizza\_id) AS pizza\_count

FROM cust\_orders c

GROUP BY c.pizza\_id

)

SELECT

topping\_name,

COUNT(topping\_name) X pizza\_count AS total\_topping\_count

FROM topping\_list

INNER JOIN pizza\_counter

ON topping\_list.pizza\_id = pizza\_counter.pizza\_id

GROUP BY topping\_name

ORDER BY total\_topping\_count DESC

**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?**

**Query #1**

SELECT

SUM(CASE

WHEN pizza\_id = 1 THEN 12

WHEN pizza\_id = 2 THEN 10

END) AS pizza\_cost

FROM cust\_orders;

| **pizza\_cost** |
| --- |
| 160 |

**2. What if there was an additional $1 charge for any pizza extras?**

Here, we are basing this off the same approach used in questions 6/7 (last section), where we use a locate the comma delimiter as part of a boolean based column. We also use a case statement that reflects the standard costs of a meat lover and vegetarian pizza from the previous question.

WITH ex\_bool\_list AS (

SELECT

order\_id,

pizza\_id,

extras\_cleaned,

IF(LOCATE(',', extras\_cleaned), TRUE, FALSE) AS ex\_bool,

CASE

WHEN pizza\_id = 1 THEN 12

WHEN pizza\_id = 2 THEN 10

END AS base\_pizza\_cost

FROM cust\_orders

),

| **order\_id** | **pizza\_id** | **extras\_cleaned** | **ex\_bool** | **base\_pizza\_cost** |
| --- | --- | --- | --- | --- |
| 1 | 1 |  | 0 | 12 |
| 2 | 1 |  | 0 | 12 |
| 3 | 1 |  | 0 | 12 |
| 3 | 2 |  | 0 | 10 |
| 4 | 1 |  | 0 | 12 |
| 4 | 1 |  | 0 | 12 |
| 4 | 2 |  | 0 | 10 |
| 5 | 1 | 1 | 0 | 12 |
| 6 | 2 | null | 0 | 10 |
| 7 | 2 | 1 | 0 | 10 |
| 8 | 1 | null | 0 | 12 |
| 9 | 1 | 1, 5 | 1 | 12 |
| 10 | 1 | null | 0 | 12 |
| 10 | 1 | 1, 4 | 1 | 12 |

**Again, we use the 1 values generated from the previous query to extract the values to the left and right of the comma delimiter to a separate column.**

ext\_list AS (

SELECT

order\_id,

base\_pizza\_cost,

CASE

WHEN ex\_bool = 0 AND extras\_cleaned IS NOT NULL THEN extras\_cleaned

END AS base\_extras\_cleaned,

CASE

WHEN ex\_bool = 1 THEN SUBSTRING\_INDEX(extras\_cleaned, ',', 1)

END AS ex\_1,

CASE

WHEN ex\_bool = 1 THEN SUBSTRING\_INDEX(extras\_cleaned, ',', -1)

END AS ex\_2

FROM ex\_bool\_list

),

| **order\_id** | **base\_pizza\_cost** | **base\_extras\_cleaned** | **ex\_1** | **ex\_2** |
| --- | --- | --- | --- | --- |
| 1 | 12 |  |  |  |
| 2 | 12 |  |  |  |
| 3 | 12 |  |  |  |
| 3 | 10 |  |  |  |
| 4 | 12 |  |  |  |
| 4 | 12 |  |  |  |
| 4 | 10 |  |  |  |
| 5 | 12 | 1 |  |  |
| 6 | 10 | null |  |  |
| 7 | 10 | 1 |  |  |
| 8 | 12 | null |  |  |
| 9 | 12 |  | 1 | 5 |
| 10 | 12 | null |  |  |
| 10 | 12 |  | 1 | 4 |

**At this point, we accumulate all the separated values to the associated standard pizza costs.**

total\_cost\_list AS (

SELECT

order\_id,

CASE

WHEN COALESCE(base\_extras\_cleaned, ex\_1, ex\_2) IS NULL THEN base\_pizza\_cost

WHEN base\_extras\_cleaned IS NOT NULL AND COALESCE(ex\_1, ex\_2) IS NULL THEN 1 + base\_pizza\_cost

WHEN base\_extras\_cleaned IS NULL AND COALESCE(ex\_1, ex\_2) IS NOT NULL THEN 2 + base\_pizza\_cost

END AS total\_pizza\_cost

FROM ext\_list

)

| **order\_id** | **total\_pizza\_cost** |
| --- | --- |
| 1 | 12 |
| 2 | 12 |
| 3 | 12 |
| 3 | 10 |
| 4 | 12 |
| 4 | 12 |
| 4 | 10 |
| 5 | 13 |
| 6 | 11 |
| 7 | 11 |
| 8 | 13 |
| 9 | 14 |
| 10 | 13 |
| 10 | 14 |

**Aggregate the sum of total pizza costs in each row.**

SELECT SUM(total\_pizza\_cost)

FROM total\_cost\_list;

| **sum(total\_pizza\_cost)** |
| --- |
| 169 |

**Add cheese is $1 extra.**

This is built on top of the previous answer. The biggest difference is the conditional statement added for 4 (for cheese)

total\_cost\_list AS (

SELECT

order\_id,

CASE

WHEN COALESCE(base\_extras\_cleaned, ex\_1, ex\_2) IS NULL THEN base\_pizza\_cost

WHEN base\_extras\_cleaned IS NOT NULL AND COALESCE(ex\_1, ex\_2) IS NULL THEN 1 + base\_pizza\_cost

WHEN base\_extras\_cleaned IS NULL AND ex\_1 is not null AND ex\_2 != 4 THEN 2 + base\_pizza\_cost

WHEN base\_extras\_cleaned IS NULL AND ex\_1 is not null AND ex\_2 = 4 THEN 3 + base\_pizza\_cost

END AS total\_pizza\_cost

FROM ext\_list

)

| **order\_id** | **total\_pizza\_cost** |
| --- | --- |
| 1 | 12 |
| 2 | 12 |
| 3 | 12 |
| 3 | 10 |
| 4 | 12 |
| 4 | 12 |
| 4 | 10 |
| 5 | 13 |
| 6 | 11 |
| 7 | 11 |
| 8 | 13 |
| 9 | 14 |
| 10 | 13 |
| 10 | 15 |

SELECT SUM(total\_pizza\_cost)

FROM total\_cost\_list;

| **sum(total\_pizza\_cost)** |
| --- |
| 170 |

Skipped 3-4, as its very subjective and theoretical. Only really interested in the puzzle solving questions.

**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?**

We use the first part from the answer in question 1.

WITH base\_pizza\_cost AS (

SELECT

SUM(CASE

WHEN pizza\_id = 1 THEN 12

WHEN pizza\_id = 2 THEN 10

END) AS pizza\_cost

FROM cust\_orders

),

| **pizza\_cost** |
| --- |
| 160 |

**When a distance\_km is labeled as null, it means that the delivery was cancelled, so we identify the orders that weren't cancelled and multiply it by 0.30 per the question.**

runner\_cost\_list AS (

SELECT distance\_km,

CASE

WHEN distance\_km IS NOT NULL THEN distance\_km\*0.30

END AS runner\_cost

FROM runner\_orders\_post

),

| **distance\_km** | **runner\_cost** |
| --- | --- |
| 20.0 | 6.000 |
| 20.0 | 6.000 |
| 13.4 | 4.020 |
| 23.4 | 7.020 |
| 10.0 | 3.000 |
|  |  |
| 25.0 | 7.500 |
| 23.4 | 7.020 |
|  |  |
| 10.0 | 3.000 |

**Here is the total sum of the runner costs.**

runner\_cost\_total AS (

SELECT SUM(runner\_cost) AS total\_runner\_cost

FROM runner\_cost\_list

)

| **total\_runner\_cost** |
| --- |
| 43.560 |

**We then subtract the pizza\_costs (which represent the profits) with the previous total query to account for the runner expenses, to get the true profit.**

SELECT

pizza\_cost - total\_runner\_cost

FROM base\_pizza\_cost, runner\_cost\_total

| **pizza\_cost - total\_runner\_cost** |
| --- |
| 116.440 |