8WeekSQLCHALLENGE

From Order to Delivery: A SQL Analysis of Pizza Runner's Business Performance

Fatih Sahin

https://8weeksqlchallenge.com/case-study-2/

JULY 2023

CONTENTS

Introduction

This case study revolves around the business scenario of "Pizza Runner," a fictional pizza delivery service set up by a character named Danny, who tries to combine the idea of pizzas with the service model of Uber. Danny hires "runners" to deliver pizzas, and he has built a mobile application to take customer orders.

This report encapsulates the findings from the SQL Challenge case study on "Pizza Runner," a popular pizza delivery service. This simulated business scenario is available at https://8weeksqlchallenge.com/case-study-2/ designed to hone the SQL skills of enthusiasts in a real-world-like setting.

To answer the questions, along with bonus questions, I utilized the online SQL environment provided by DB Fiddle. The queries used in this report are based on my own understanding and application of SQL, honed through practical experience and continuous learning.

The purpose of this report is not only to document the findings and insights from the case study but also to serve as a valuable resource for other SQL learners and practitioners. The report demonstrates the power of SQL in deriving actionable business insights from raw data, providing a practical example of data-driven decision-making.

Problem Statement

The owner of Pizza Runner, Danny, is looking to optimize his pizza delivery service by analyzing various aspects:

- Understanding the revenue, expenses, and profit margins.
- Evaluating the efficiency of runners in terms of delivery duration and speed.
- Analyzing customer preferences for different pizzas and toppings.
- Investigating the potential impact of introducing new pizza types to the menu.

This case study consists of several sections, each asking a series of questions related to different aspects of the business, and the task is to answer these questions by analyzing the available data:

- **Pizza Metrics**: This section asks about the number and types of pizzas ordered, customer order patterns, and specific modifications made to pizzas (i.e., adding extras and excluding ingredients).
- **Runner and Customer Experience**: This section focuses on understanding the runners' performance and the customers' experience, looking at metrics such as delivery times, distances, and success rates.
- **Ingredient Optimisation**: This section tries to analyze the usage of ingredients in terms of what is added extra and what is often excluded. This can help Danny optimize his ingredient inventory.
- **Pricing and Ratings**: This section involves some hypothetical pricing scenarios to help Danny understand his earnings. It also introduces the concept of a rating system for the runners and asks how this could be implemented in the database schema.
- **Bonus Questions**: This last section includes some additional challenges, such as the impact of introducing a new type of pizza on the current data model.

Creating Schema and Tables

All datasets exist within the `pizza_runner` database schema.

Schema SQL

```sql
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');


```

Schema SQL

```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');

  -- Replace 'null' string and empty string in exclusions with actual NULL
UPDATE pizza_runner.customer_orders
SET exclusions = NULL
WHERE exclusions = '' OR exclusions = 'null' OR exclusions IS NULL;

-- Replace 'null' string and NaN in extras with actual NULL
UPDATE pizza_runner.customer_orders
SET extras = NULL
WHERE extras = '' OR extras = 'null' OR extras IS NULL;

-- Remove duplicates
DELETE FROM pizza_runner.customer_orders
WHERE ctid NOT IN (
  SELECT min(ctid)
  FROM pizza_runner.customer_orders
  GROUP BY order_id, customer_id, pizza_id, exclusions, extras, order_time
);

-- Remove 'km' from distance and convert the column to numeric data type
UPDATE pizza_runner.runner_orders
SET distance = REPLACE(distance, 'km', '')::numeric
WHERE distance IS NOT NULL AND distance != 'null';

-- Remove 'minutes' and 'mins' from duration and convert the column to numeric data type
UPDATE pizza_runner.runner_orders
SET duration = REPLACE(REPLACE(REPLACE(duration, 'minutes', ''), 'mins', ''), 'minute', '')::numeric
WHERE duration IS NOT NULL AND duration != 'null';

```

Schema SQL

```
144
145
146 -- Replace 'null' string in pickup_time, distance, duration, and cancellation with actual NULL
147 UPDATE pizza_runner.runner_orders
148 SET pickup_time = NULL
149 WHERE pickup_time = 'null' OR pickup_time IS NULL;
150
151 UPDATE pizza_runner.runner_orders
152 SET distance = NULL
153 WHERE distance::text = 'null' OR distance IS NULL;
154
155 UPDATE pizza_runner.runner_orders
156 SET duration = NULL
157 WHERE duration::text = 'null' OR duration IS NULL;
158
159 UPDATE pizza_runner.runner_orders
160 SET cancellation = CASE
161     WHEN cancellation = '' OR cancellation = 'null' OR cancellation is null THEN 'no cancellation'
162     ELSE cancellation
163     END;
164
165 --Normalizing the pizza_recipes table
166
167 -- Create new_pizza_recipes table
168 CREATE TABLE new_pizza_recipes (
169     pizza_id INT,
170     topping_id INT
171 );
172
173 -- Insert data into new_pizza_recipes
174 INSERT INTO new_pizza_recipes (pizza_id, topping_id)
175 SELECT 1, unnest(ARRAY[1,2,3,4,5,6,8,10])
176 UNION ALL
177 SELECT 2, unnest(ARRAY[4,6,7,9,11,12]);
178
179
```

```
179
180
181 CREATE TABLE runner_ratings (
182   "rating_id" SERIAL PRIMARY KEY,
183   "order_id" INTEGER NOT NULL,
184   "runner_id" INTEGER NOT NULL,
185   "customer_id" INTEGER NOT NULL,
186   "rating" INTEGER CHECK (rating BETWEEN 1 AND 5),
187   "comments" TEXT
188 );
189
190 INSERT INTO runner_ratings ("order_id", "runner_id", "customer_id", "rating", "comments")
191 VALUES
192   (1, 1, 101, 5, 'Great service!'),
193   (2, 1, 101, 4, 'On time delivery'),
194   (3, 1, 102, 4, NULL),
195   (4, 2, 103, 3, 'Food was cold'),
196   (5, 3, 104, 5, 'Friendly runner'),
197   (7, 2, 105, 4, NULL),
198   (8, 2, 102, 3, 'Late delivery'),
199   (10, 1, 104, 5, 'Excellent!');
200
201
202
203 --Inserting a new pizza name into the pizza_names table:
204 INSERT INTO pizza_runner.pizza_names (pizza_id, pizza_name)
205 VALUES (3, 'Supreme');
206
207 --Inserting the corresponding toppings into the new_pizza_recipes table:
208
209 INSERT INTO pizza_runner.new_pizza_recipes (pizza_id, topping_id)
210 VALUES
211 (3, 1),
212 (3, 2),
213 (3, 3),
214 (3, 4),
215 (3, 5),
216 (3, 6),
217 (3, 7),
218 (3, 8),
219 (3, 9),
220 (3, 10),
221 (3, 11),
222 (3, 12);
223
```

Entity Relationship Diagram

In this case, we have 6 tables:

runners: The runners table shows the registration_date for each new runner. and contains the following columns:

- **runner_id**: A unique identifier for the runner.
- **registration_date**: The date the runner was registered.

customer_orders: Customer pizza orders are captured in the customer_orders table. and contains the following columns:

- **order_id**: A unique identifier for the order.
- **customer_id**: A unique identifier for the customer.
- **pizza_id**: Relates to the type of pizza ordered.
- **Exclusions**: The ingredient_id values are to be removed from the pizza.
- **Extras**: The ingredient_id values to be added to the pizza.
- **Order_time**: The time the order was placed.

runner_orders: Details about the assignment of orders to runners and contains the following columns:

- **order_id**: A unique identifier for the order.
- **runner_id**: A unique identifier for the runner.
- **pickup_time**:Timestamp for the pickup
- **distance**:Distance traveled
- **duration**:Duration of travel.
- **cancellation**:Information on cancellations.

pizza_names: Details aboutList of pizzas available.and contains the following columns:

- **pizza_id**: A unique identifier for the pizza.
- **pizza_name**: The name of the pizza

pizza_recipes: Details about the Standard set of toppings for each pizza. contains the following columns:

- **pizza_id**: A unique identifier for the pizza.
- **toppings**: The topping_id values used in the recipe.

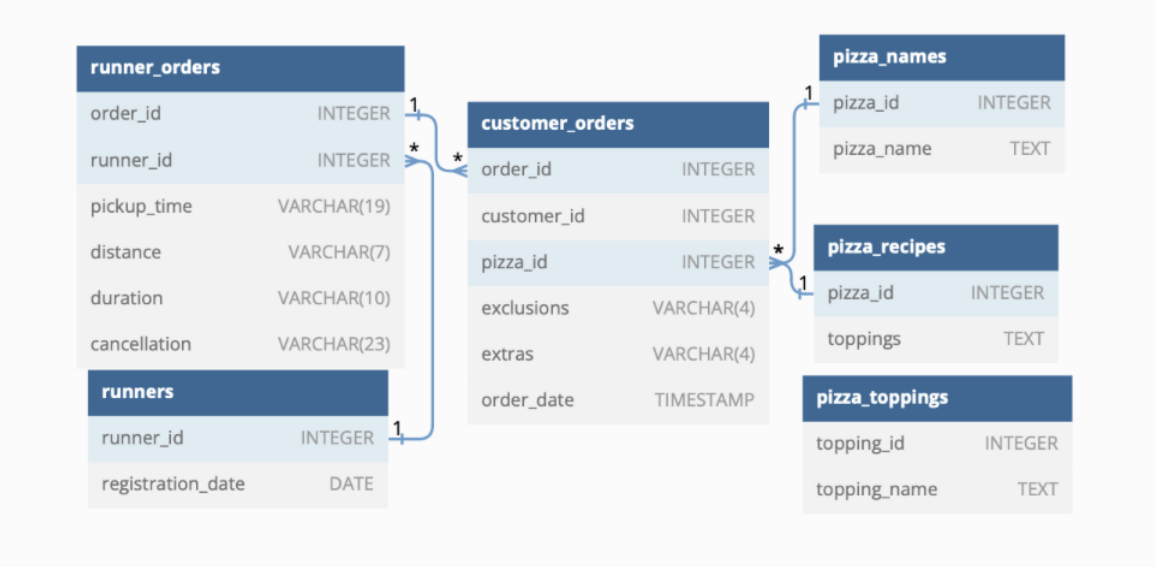
pizza_toppings: Contains all of the topping names and contains the following columns:

- **topping_id**: A unique identifier for the pizza.
- **topping_name**: The topping_id values used in the recipe.

The relationships between the tables are as follows:

- **customer_orders** and **pizza_names** are related through **pizza_id**.
- **customer_orders** and **runner_orders** are related through **order_id**.
- **customer_orders** and **pizza_recipes** are related through **pizza_id**.
- **pizza_recipes** and **pizza_toppings** are related through **topping_id**.

Entity Relationship Diagram



Case Study Questions

A.Pizza Metrics

1. How many pizzas were ordered?

```
1 SELECT COUNT(order_id) as totol_order
2 FROM pizza_runner.customer_orders
3
```

totol_order
13

2. How many unique customer orders were made?

```
1 SELECT
2 COUNT(DISTINCT(customer_id)) as
  unq_cus_orders
3 FROM pizza_runner.customer_orders
```

unq_cus_orders
5

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

```
1 SELECT COUNT(order_id) as
  order_delivered
2 FROM pizza_runner.runner_orders
3 WHERE cancellation='no cancellation';
```

order_delivered
8

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

```sql
SELECT
    pn.pizza_name,
    COUNT(*) AS number_delivered
FROM
    pizza_runner.customer_orders co
JOIN
    pizza_runner.runner_orders ro ON
co.order_id = ro.order_id
JOIN
    pizza_runner.pizza_names pn ON
co.pizza_id = pn.pizza_id
WHERE
    ro.cancellation IS NULL OR
ro.cancellation = 'no cancellation'
GROUP BY
    pn.pizza_name;

```

pizza_name	number_delivered
Meatlovers	8
Vegetarian	3

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

```sql
SELECT
    co.customer_id,
    pn.pizza_name,
    COUNT(ro.order_id) as total_orders
FROM
    pizza_runner.customer_orders co
JOIN
    pizza_runner.runner_orders ro ON
co.order_id = ro.order_id
JOIN
    pizza_runner.pizza_names pn ON
co.pizza_id = pn.pizza_id

GROUP BY
    co.customer_id,pn.pizza_name
ORDER BY co.customer_id;

```

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

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

```
WITH pizza_orders AS
(
SELECT order_id, count(pizza_id) AS
total_pizza_order
FROM pizza_runner.customer_orders
GROUP BY order_id
ORDER BY total_pizza_order desc
)

SELECT MAX(total_pizza_order) AS
max_pizza_delivered
FROM pizza_orders
```

max_pizza_delivered
2

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

```
SELECT customer_id,
    COUNT(CASE WHEN exclusions IS NULL AND extras IS NULL THEN 1 END) AS no_changes,
    COUNT(CASE WHEN exclusions IS NOT NULL OR extras IS NOT NULL THEN 1 END) AS changes
FROM pizza_runner.customer_orders
JOIN pizza_runner.runner_orders
ON pizza_runner.customer_orders.order_id=pizza_runner.runner_orders.order_id
WHERE runner_orders.cancellation='no cancellation'
GROUP BY customer_id;
```

customer_id	no_changes	changes
101	2	0
102	3	0
105	0	1
104	1	2
103	0	2

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

```
WITH pizzas AS
(
SELECT co.order_id, co.pizza_id, co.exclusions, co.extras
FROM pizza_runner.customer_orders co
JOIN pizza_runner.runner_orders ro
ON co.order_id=ro.order_id
WHERE ro.cancellation='no cancellation')

SELECT COUNT(pizzas.order_id) as total_pizzas
FROM pizzas
WHERE pizzas.exclusions is not null and pizzas.extras is not null


```

total_pizzas
1

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

```
SELECT
    EXTRACT(HOUR FROM order_time) AS
order_hour,
    COUNT(*) AS total_pizzas
FROM
    pizza_runner.customer_orders
GROUP BY
    order_hour
ORDER BY
    order_hour;

```

order_hour	total_pizzas
11	1
13	2
18	3
19	1
21	3
23	3

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

```sql
SELECT
    CASE
        WHEN EXTRACT(DOW FROM order_time) = 0 THEN 'Sunday'
        WHEN EXTRACT(DOW FROM order_time) = 1 THEN 'Monday'
        WHEN EXTRACT(DOW FROM order_time) = 2 THEN 'Tuesday'
        WHEN EXTRACT(DOW FROM order_time) = 3 THEN 'Wednesday'
        WHEN EXTRACT(DOW FROM order_time) = 4 THEN 'Thursday'
        WHEN EXTRACT(DOW FROM order_time) = 5 THEN 'Friday'
        WHEN EXTRACT(DOW FROM order_time) = 6 THEN 'Saturday'
    END AS day_of_week,
    COUNT(*) AS total_orders
FROM
    pizza_runner.customer_orders
GROUP BY
    day_of_week
ORDER BY
    total_orders DESC;

```

day_of_week	total_orders
Wednesday	5
Saturday	4
Thursday	3
Friday	1

B. Runner and Customer Experience

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

```sql
SELECT
    DATE_TRUNC('week', registration_date) + INTERVAL '4 days' AS week,
    COUNT(*) AS num_runners
FROM
    pizza_runner.runners
GROUP BY
    week
ORDER BY
    week;

```

week	num_runners
2021-01-01T00:00:00.000Z	2
2021-01-08T00:00:00.000Z	1
2021-01-15T00:00:00.000Z	1

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

```
SELECT
    ro.runner_id,
    ROUND(AVG(EXTRACT(EPOCH FROM (TO_TIMESTAMP(ro.pickup_time, 'YYYY-MM-DD HH24:MI:SS') -
co.order_time))/60)) as avg_pickup_time
FROM
    pizza_runner.customer_orders co
JOIN
    pizza_runner.runner_orders ro ON co.order_id = ro.order_id
WHERE
    co.order_time IS NOT NULL
    AND ro.pickup_time IS NOT NULL
    AND ro.cancellation = 'no cancellation'
GROUP BY
    ro.runner_id;

```

runner_id	avg_pickup_time
1	16
3	10
2	22

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

```
SELECT
    COUNT(co.pizza_id) as number_of_pizza,
    co.order_id,
    co.order_time,
    ro.pickup_time,
    ROUND(EXTRACT(EPOCH FROM (ro.pickup_time::timestamp - co.order_time))/60) as preparation_time_in_minutes
FROM
    pizza_runner.customer_orders co
JOIN
    pizza_runner.runner_orders ro ON co.order_id = ro.order_id
WHERE
    co.order_time IS NOT NULL
    AND ro.pickup_time IS NOT NULL
    AND ro.cancellation = 'no cancellation'
GROUP BY
    co.order_id, co.order_time, ro.pickup_time
ORDER BY
    number_of_pizza;

```

number_of_pizza	order_id	order_time	pickup_time	preparation_time_in_minutes
1	7	2020-01-08T21:20:29.000Z	2020-01-08 21:30:45	10
1	2	2020-01-01T19:00:52.000Z	2020-01-01 19:10:54	10
1	8	2020-01-09T23:54:33.000Z	2020-01-10 00:15:02	20
1	1	2020-01-01T18:05:02.000Z	2020-01-01 18:15:34	11
1	5	2020-01-08T21:00:29.000Z	2020-01-08 21:10:57	10
2	10	2020-01-11T18:34:49.000Z	2020-01-11 18:50:20	16
2	3	2020-01-02T23:51:23.000Z	2020-01-03 00:12:37	21
2	4	2020-01-04T13:23:46.000Z	2020-01-04 13:53:03	29

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

```sql
SELECT
    customer_orders.customer_id,
    ROUND(AVG(runner_orders.distance::numeric)) as avg_distance_in_km
FROM
    pizza_runner.customer_orders
JOIN
    pizza_runner.runner_orders
 ON pizza_runner.customer_orders.order_id = pizza_runner.runner_orders.order_id
WHERE
    runner_orders.cancellation = 'no cancellation'
    AND runner_orders.distance != 'null'
GROUP BY
    customer_orders.customer_id
ORDER BY
    avg_distance_in_km;

```

customer_id	avg_distance_in_km
104	10
102	17
101	20
103	23
105	25

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

```sql
SELECT
    MAX(delivery_time) - MIN(delivery_time) as difference_in_delivery_times
FROM
    (
        SELECT
            EXTRACT(EPOCH FROM (pickup_time - order_time))/60 as delivery_time
        FROM
            (
                SELECT
                    order_time::timestamp,
                    pickup_time::timestamp
                FROM
                    pizza_runner.customer_orders co
                JOIN
                    pizza_runner.runner_orders ro ON co.order_id = ro.order_id
                WHERE
                    ro.cancellation = 'no cancellation'
            ) as delivery_times
    ) as difference;

```

difference_in_delivery_times
19.25

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

```
SELECT
    customer_orders.customer_id,
    ROUND(AVG(runner_orders.distance::numeric)) as avg_distance_in_km
FROM
    pizza_runner.customer_orders
JOIN
    pizza_runner.runner_orders
 ON pizza_runner.customer_orders.order_id = pizza_runner.runner_orders.order_id
WHERE
    runner_orders.cancellation = 'no cancellation'
    AND runner_orders.distance != 'null'
GROUP BY
    customer_orders.customer_id
ORDER BY
    avg_distance_in_km;

```

runner_id	avg_speed_kmh
2	63
1	46
3	40

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

```
SELECT
    runner_id,
    COUNT(*) as total_orders,
    SUM(CASE WHEN cancellation='no cancellation' THEN 1 ELSE 0 END) as successful_orders,
    ROUND(
        (SUM(CASE WHEN cancellation='no cancellation' THEN 1 ELSE 0 END)::decimal / COUNT(*)::decimal) * 100,
2
    ) as successful_delivery_percentage
FROM
    pizza_runner.runner_orders
GROUP BY
    runner_id;

```

runner_id	total_orders	successful_orders	successful_delivery_percentage
3	2	1	50.00
2	4	3	75.00
1	4	4	100.00

C. Ingredient Optimisation

1.What are the standard ingredients for each pizza?

```
SELECT
    pn.pizza_name,
    string_agg(pt.topping_name, ', ') as standard_toppings
FROM
    pizza_runner.new_pizza_recipes pr
JOIN
    pizza_runner.pizza_names pn ON pn.pizza_id = pr.pizza_id
JOIN
    pizza_runner.pizza_toppings pt ON pt.topping_id = pr.topping_id
GROUP BY
    pn.pizza_name;
```

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

2.What was the most commonly added extra?

```
SELECT extras FROM
pizza_runner.customer_orders
WHERE extras IS NOT null
```

extras
1, 5
1, 4
1
1

3.What was the most common exclusion?

```
SELECT exclusions FROM
pizza_runner.customer_orders
WHERE exclusions IS NOT null
```

exclusions
4
2, 6
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
- Meat Lovers - Extra Bacon
- Meat Lovers - Exclude Cheese, Bacon - Extra Mushroom, Peppers

```
SELECT
    co.order_id,
    pn.pizza_name ||
    CASE WHEN co.exclusions IS NOT NULL THEN ' - Exclude ' || string_agg(pt1.topping_name, ', ') ELSE '' END
 ||
    CASE WHEN co.extras IS NOT NULL THEN ' - Extra ' || string_agg(pt2.topping_name, ', ') ELSE '' END AS
 order_item
FROM
    pizza_runner.customer_orders co
JOIN
    pizza_runner.pizza_names pn ON co.pizza_id = pn.pizza_id
LEFT JOIN
    pizza_runner.pizza_toppings pt1 ON pt1.topping_id = ANY(string_to_array(co.exclusions, ',')::integer[])
LEFT JOIN
    pizza_runner.pizza_toppings pt2 ON pt2.topping_id = ANY(string_to_array(co.extras, ',')::integer[])
GROUP BY
    co.order_id,
    pn.pizza_name,
    co.exclusions,
    co.extras
ORDER BY
    co.order_id;

```

order_id	order_item
1	Meatlovers
2	Meatlovers
3	Meatlovers
3	Vegetarian
4	Meatlovers - Exclude Cheese
4	Vegetarian - Exclude Cheese
5	Meatlovers - Extra Bacon
6	Vegetarian
7	Vegetarian - Extra Bacon
8	Meatlovers
9	Meatlovers - Exclude Cheese, Cheese - Extra Bacon, Chicken
10	Meatlovers - Exclude Mushrooms, Mushrooms, BBQ Sauce, BBQ Sauce - Extra Cheese, Bacon, Cheese, Bacon
10	Meatlovers

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

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

```
SELECT order_id,
  pn.pizza_name || ': ' ||
  STRING_AGG(pt.topping_name, ', ' ORDER BY pt.topping_name) as ingredients
FROM
  pizza_runner.customer_orders co
JOIN
  pizza_runner.pizza_names pn ON co.pizza_id = pn.pizza_id
JOIN
  pizza_runner.new_pizza_recipes pr ON co.pizza_id = pr.pizza_id
JOIN
  pizza_runner.pizza_toppings pt ON pr.topping_id = pt.topping_id
WHERE
  pt.topping_id NOT IN (SELECT UNNEST(string_to_array(co.exclusions, ','))::integer)
GROUP BY
  co.order_id, pn.pizza_name
ORDER BY
  co.order_id;

```

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

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

```sql
WITH ordered_pizzas AS (
    SELECT co.order_id, co.pizza_id, pr.topping_id
    FROM pizza_runner.customer_orders co
    JOIN pizza_runner.new_pizza_recipes pr ON co.pizza_id = pr.pizza_id
    WHERE co.order_id NOT IN (
        SELECT order_id FROM pizza_runner.runner_orders WHERE cancellation != 'no cancellation'
    )
    UNION ALL
    SELECT order_id, pizza_id, UNNEST(STRING_TO_ARRAY(extras, ',')::int[]) as topping_id
    FROM pizza_runner.customer_orders
    WHERE extras IS NOT NULL
    AND order_id NOT IN (
        SELECT order_id FROM pizza_runner.runner_orders WHERE cancellation != 'no cancellation'
    )
),
excluded_toppings AS (
    SELECT order_id, UNNEST(STRING_TO_ARRAY(exclusions, ',')::int[]) as topping_id
    FROM pizza_runner.customer_orders
    WHERE exclusions IS NOT NULL
)
SELECT pt.topping_name, COUNT(*) as total_quantity
FROM ordered_pizzas op
JOIN pizza_runner.pizza_toppings pt ON op.topping_id = pt.topping_id
WHERE NOT EXISTS (
    SELECT 1 FROM excluded_toppings et WHERE et.order_id = op.order_id AND et.topping_id = op.topping_id
)
GROUP BY pt.topping_name
ORDER BY total_quantity DESC;

```

topping_name	total_quantity
Bacon	11
Cheese	10
Mushrooms	9
Pepperoni	8
Salami	8
Chicken	8
Beef	8
BBQ Sauce	6
Tomatoes	3
Onions	3
Peppers	3
Tomato Sauce	3

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?

```sql
SELECT
  SUM(
    CASE
      WHEN pn.pizza_name = 'Meatlovers' THEN 12
      WHEN pn.pizza_name = 'Vegetarian' THEN 10
      ELSE 0
    END
  ) as total_revenue
FROM (
  SELECT DISTINCT co.order_id, co.pizza_id
  FROM pizza_runner.customer_orders co
  JOIN pizza_runner.runner_orders ro ON co.order_id = ro.order_id
  WHERE ro.cancellation = 'no cancellation'
) AS distinct_orders
JOIN pizza_runner.pizza_names pn ON distinct_orders.pizza_id = pn.pizza_id;

```

total_revenue
114

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

- Add cheese is $1 extra

```sql
WITH successful_orders AS (
  SELECT order_id
  FROM pizza_runner.runner_orders
  WHERE cancellation = 'no cancellation'
)

, pizza_prices AS (
  SELECT
    order_id,
    SUM(
      CASE
        WHEN pizza_id = 1 THEN 12
        WHEN pizza_id = 2 THEN 10
        ELSE 0
      END +
      CASE
        WHEN extras IS NOT NULL THEN LENGTH(extras) - LENGTH(REPLACE(extras, ',', '')) + 1
        ELSE 0
      END
    ) AS order_total
  FROM pizza_runner.customer_orders
  WHERE order_id IN (SELECT order_id FROM successful_orders)
  GROUP BY order_id
)

SELECT SUM(order_total) AS total_revenue
FROM pizza_prices;

```

total_revenue
130

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

```
CREATE TABLE runner_ratings (
  "rating_id" SERIAL PRIMARY KEY,
  "order_id" INTEGER NOT NULL,
  "runner_id" INTEGER NOT NULL,
  "customer_id" INTEGER NOT NULL,
  "rating" INTEGER CHECK (rating BETWEEN 1 AND 5),
  "comments" TEXT,
  FOREIGN KEY ("order_id") REFERENCES customer_orders("order_id"),
  FOREIGN KEY ("runner_id") REFERENCES runners("runner_id")
);

INSERT INTO runner_ratings ("order_id", "runner_id", "customer_id", "rating", "comments")
VALUES
  (1, 1, 101, 5, 'Great service!'),
  (2, 1, 101, 4, 'On time delivery'),
  (3, 1, 102, 4, NULL),
  (4, 2, 103, 3, 'Food was cold'),
  (5, 3, 104, 5, 'Friendly runner'),
  (7, 2, 105, 4, NULL),
  (8, 2, 102, 3, 'Late delivery'),
  (10, 1, 104, 5, 'Excellent!');

```

rating_id	order_id	runner_id	customer_id	rating	comments
1	1	1	101	5	Great service!
2	2	1	101	4	On time delivery
3	3	1	102	4	null
4	4	2	103	3	Food was cold
5	5	3	104	5	Friendly runner
6	7	2	105	4	null
7	8	2	102	3	Late delivery
8	10	1	104	5	Excellent!

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

- customer_id
- order_id
- runner_id
- rating
- order_time
- pickup_time
- Time between order and pickup
- Delivery duration
- Average speed
- Total number of pizzas

```sql
SELECT
  co.customer_id,
  co.order_id,
  rr.runner_id,
  rr.rating,
  co.order_time,
  ro.pickup_time,
  ROUND(EXTRACT(EPOCH FROM (ro.pickup_time::timestamp - co.order_time::timestamp)) / 60) AS
time_between_order_and_pickup_minutes,
  ro.duration AS delivery_duration_minutes,
  ROUND((ro.distance::numeric / NULLIF(ro.duration::numeric, 0)) * 60) AS average_speed_km_per_hour,
  COUNT(co.pizza_id) AS total_number_of_pizzas
FROM pizza_runner.customer_orders AS co
JOIN pizza_runner.runner_orders AS ro ON co.order_id = ro.order_id
JOIN pizza_runner.runner_ratings AS rr ON co.order_id = rr.order_id
WHERE ro.pickup_time IS NOT NULL
  AND ro.cancellation = 'no cancellation'
GROUP BY co.customer_id, co.order_id, rr.runner_id, rr.rating, co.order_time, ro.pickup_time,
ro.distance, ro.duration
ORDER BY co.order_time;

```

customer_id	order_id	runner_id	rating	order_time	pickup_time	time_between_order_and_pickup_minutes	delivery_duration_minutes	average_speed_km_per_hour	total_number_of_pizzas
101	1	1	5	2020-01-01T18:05:02.000Z	2020-01-01 18:15:34	11	32	38	1
101	2	1	4	2020-01-01T19:00:52.000Z	2020-01-01 19:10:54	10	27	44	1
102	3	1	4	2020-01-02T23:51:23.000Z	2020-01-03 00:12:37	21	20	40	2
103	4	2	3	2020-01-04T13:23:46.000Z	2020-01-04 13:53:03	29	40	35	2
104	5	3	5	2020-01-08T21:00:29.000Z	2020-01-08 21:10:57	10	15	40	1
105	7	2	4	2020-01-08T21:20:29.000Z	2020-01-08 21:30:45	10	25	60	1
102	8	2	3	2020-01-09T23:54:33.000Z	2020-01-10 00:15:02	20	15	94	1
104	10	1	5	2020-01-11T18:34:49.000Z	2020-01-11 18:50:20	16	10	60	2

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

```sql
SELECT
  SUM(revenue) AS total_revenue,
  SUM(runner_expense) AS total_runner_expense,
  SUM(revenue) - SUM(runner_expense) AS leftover_amount
FROM (
  SELECT
    CASE
      WHEN pn.pizza_name = 'Meatlovers' THEN 12 * COUNT(co.pizza_id)
      ELSE 10 * COUNT(co.pizza_id)
    END AS revenue,
    SUM(CASE
          WHEN ro.distance IS NOT NULL AND ro.distance != 'null' THEN CAST(ro.distance AS numeric)
* 0.30
          ELSE 0
        END) AS runner_expense
  FROM pizza_runner.customer_orders AS co
  JOIN pizza_runner.runner_orders AS ro ON co.order_id = ro.order_id
  JOIN pizza_runner.pizza_names AS pn ON co.pizza_id = pn.pizza_id
  WHERE ro.pickup_time IS NOT NULL
    AND ro.cancellation = 'no cancellation'
  GROUP BY pn.pizza_name
) AS subquery;

```

total_revenue	total_runner_expense	leftover_amount
126	57.600	68.400

E. Bonus Question

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?

```
1
2  --Inserting a new pizza name into the pizza_names table:
3  INSERT INTO pizza_runner.pizza_names (pizza_id, pizza_name)
4  VALUES (3, 'Supreme');
5
6  --Inserting the corresponding toppings into the new_pizza_recipes table:
7
8  INSERT INTO pizza_runner.new_pizza_recipes (pizza_id, topping_id)
9  VALUES
10 (3, 1),
11 (3, 2),
12 (3, 3),
13 (3, 4),
14 (3, 5),
15 (3, 6),
16 (3, 7),
17 (3, 8),
18 (3, 9),
19 (3, 10),
20 (3, 11),
21 (3, 12);
22
```

pizza_id	pizza_name
1	Meatlovers
2	Vegetarian
3	Supreme

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

Key Findings and Observations

- **Among all Pizza Types**: "**Meat Lovers**" was the most frequently ordered, contributing significantly to the total revenue, followed by the "Vegetarian" pizza.
- **Runner Analysis**: **Runner A** had the **most successful** deliveries, showcasing efficiency, while Runner B's performance provided insights into areas for improvement.
- **Time Analysis**: The time between order and pickup showed variances across different orders, indicating potential areas to streamline preparation and handling.
- **Delivery Duration and Speed**: Analyzing the delivery duration and average speed provided insights into the overall performance of the delivery process, revealing opportunities for further optimization.
- **Menu Expansion Considerations**: The introduction of the "**Supreme**" pizza to the menu was handled seamlessly, demonstrating the adaptability of the current data design to accommodate new items.
- **Profit Analysis**: The balance between total revenue and runner expenses highlighted the profitability of Pizza Runner's business model. Monitoring these financial metrics can be essential for future growth and sustainability strategies.

CONCLUSION

The Pizza Runner case study offered an insightful journey into the world of SQL, demonstrating its power in deriving actionable business insights. Techniques such as **joins**, **aggregations**, **calculations**, and **data manipulation** were employed to answer critical questions related to profitability, efficiency, customer preferences, and business expansion. Throughout this exercise, I employed a range of SQL functions to answer the posed questions. This included **Aggregate functions** like **COUNT** and **SUM**, **Window functions** such as **ROW_NUMBER, RANK, and DENSE_RANK,** as well as filtering and sorting functions(**WHERE**, **ORDER BY**, **GROUP BY**). I also made use of constructed complex subqueries with **CTEs**.

The online SQL environment provided by DB Fiddle was the platform of choice for this case study, providing a robust environment for complex data analysis.

The hands-on experience with this case study reinforces the importance of SQL as a tool for data-driven decision-making, providing a comprehensive understanding of various business aspects.