SQL CaseStudy Week#2

PIZZA

Runner

www.8weeksqlchallenge.com

Satya Ranjan Ray

Introduction

According to a source, around **5 billion** pizzas are sold worldwide every year.

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!

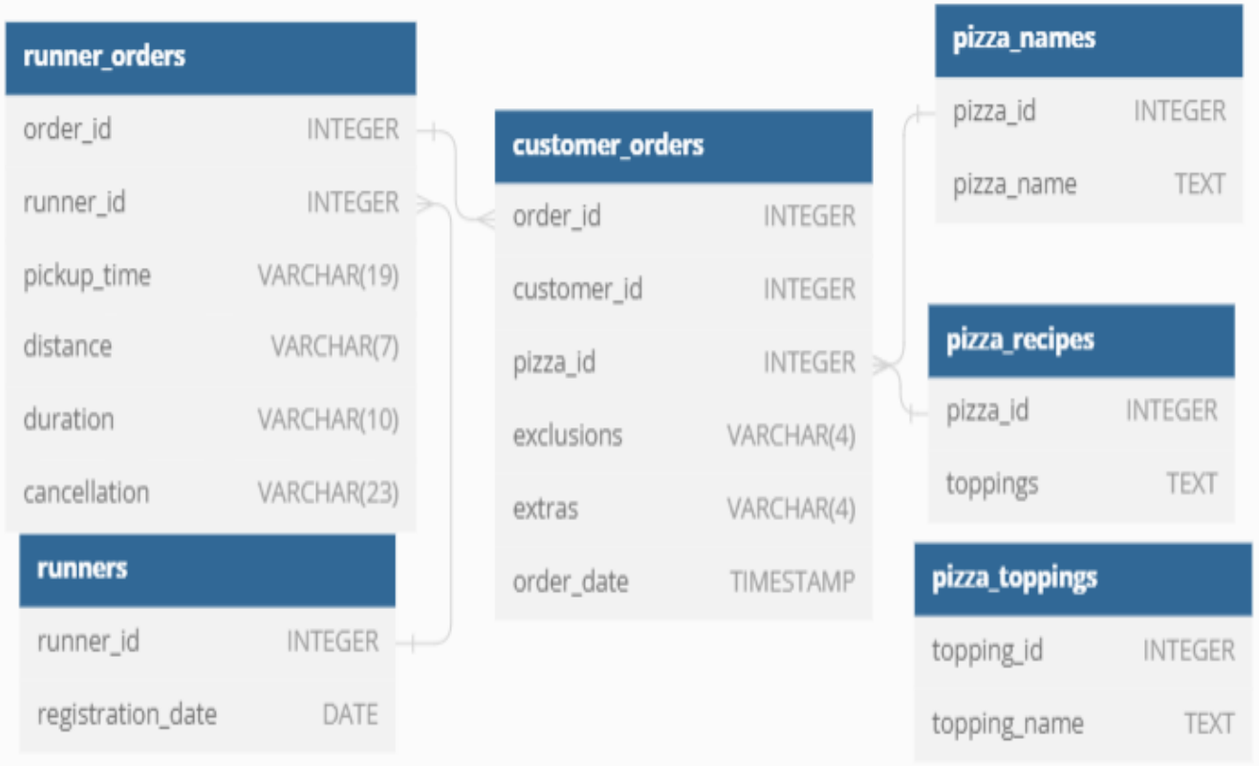
Danny started by recruiting "runners" to deliver fresh pizza from Pizza Runner Headquarters (otherwise known as **Danny's house**) and also maxed out his credit card to pay freelance developers to build a mobile app to accept orders from customers.

Problem Statement

Danny, who had worked as a data scientist for several years, was well aware of the importance of data collection for his business' s growth.

He has prepared 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 optimize Pizza Runner's operations.

Entity Relationship Diagram



Datasets

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			2020-01-01 18:05:02.000
2	101	1			2020-01-01 19:00:52.000
3	102	1			2020-01-02 23:51:23.000
3	102	2		[NULL]	2020-01-02 23:51:23.000
4	103	1	4		2020-01-04 13:23:46.000
4	103	1	4		2020-01-04 13:23:46.000
4	103	2	4		2020-01-04 13:23:46.000
5	104	1	null	1	2020-01-08 21:00:29.000
6	101	2	null	null	2020-01-08 21:03:13.000
7	105	2	null	1	2020-01-08 21:20:29.000
8	102	1	null	null	2020-01-09 23:54:33.000
9	103	1	4	1, 5	2020-01-10 11:22:59.000
10	104	1	null	null	2020-01-11 18:34:49.000
10	104	1	2, 6	1, 4	2020-01-11 18:34:49.000

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.

order_id	runner_id	pickup_time	distance	duration	cancellation
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

Table 4: pizza_names

At the moment - Pizza Runner only has 2 pizzas available the Meat Lovers or Vegetarian!

pizza_id	pizza_name
1	Meatlovers
2	Vegetarian

Table 5: pizza_recipes

Each pizza_id has a standard set of toppings which are used as part of the pizza recipe.

123 pizza_id	ABC 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.

123 topping_id	ABC 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

Data Cleaning

customer_orders

order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1			2020-01-01 18:05:02.000
2	101	1			2020-01-01 19:00:52.000
3	102	1			2020-01-02 23:51:23.000
3	102	2		[NULL]	2020-01-02 23:51:23.000
4	103	1	4		2020-01-04 13:23:46.000
4	103	1	4		2020-01-04 13:23:46.000
4	103	2	4		2020-01-04 13:23:46.000
5	104	1	null	1	2020-01-08 21:00:29.000
6	101	2	null	null	2020-01-08 21:03:13.000
7	105	2	null	1	2020-01-08 21:20:29.000
8	102	1	null	null	2020-01-09 23:54:33.000
9	103	1	4	1, 5	2020-01-10 11:22:59.000
10	104	1	null	null	2020-01-11 18:34:49.000
10	104	1	2, 6	1, 4	2020-01-11 18:34:49.000

```sql
UPDATE customer_orders
SET exclusions = CASE WHEN exclusions = 'null' THEN '' ELSE exclusions END,
    extras = CASE WHEN extras = 'null' OR extras IS NULL THEN '' ELSE extras END;
```

order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1			2020-01-01 18:05:02.000
2	101	1			2020-01-01 19:00:52.000
3	102	1			2020-01-02 23:51:23.000
3	102	2			2020-01-02 23:51:23.000
4	103	1	4		2020-01-04 13:23:46.000
4	103	1	4		2020-01-04 13:23:46.000
4	103	2	4		2020-01-04 13:23:46.000
5	104	1		1	2020-01-08 21:00:29.000
6	101	2			2020-01-08 21:03:13.000
7	105	2		1	2020-01-08 21:20:29.000
8	102	1			2020-01-09 23:54:33.000
9	103	1	4	1, 5	2020-01-10 11:22:59.000
10	104	1			2020-01-11 18:34:49.000
10	104	1	2, 6	1, 4	2020-01-11 18:34:49.000

runner_orders

order_id	runner_id	pickup_time	distance	duration	cancellation
1	1	2020-01-01 18:15:34	20km	32 minutes	
2	1	2020-01-01 19:10:54	20km	27 minutes	
3	1	2020-01-03 00:12:37	13.4km	20 mins	[NULL]
4	2	2020-01-04 13:53:03	23.4	40	[NULL]
5	3	2020-01-08 21:10:57	10	15	[NULL]
6	3	null	null	null	Restaurant Cancellation
7	2	2020-01-08 21:30:45	25km	25mins	null
8	2	2020-01-10 00:15:02	23.4 km	15 minute	null
9	2	null	null	null	Customer Cancellation
10	1	2020-01-11 18:50:20	10km	10minutes	null

```sql
/* Instead of updating the runner_orders , I cleaned and transformed the existing table and create a new table
named runner_orders_new with the existing data */

CREATE TABLE runner_orders_new AS
SELECT order_id,
       runner_id,
       CASE WHEN pickup_time = 'null' THEN NULL ELSE CAST(pickup_time AS TIMESTAMP) END AS pickup_time_adjusted,
       CAST(REGEXP_SUBSTR(distance, '[0-9]+(\.[0-9]+)?') AS FLOAT) AS dist_adjusted,
       CAST(REGEXP_SUBSTR(duration, '^[0-9]+') AS INT) AS dur_adjusted,
       CASE WHEN cancellation IN ('null', '') THEN NULL ELSE cancellation END AS cancellation_adjusted
FROM runner_orders;
```

order_id	runner_id	pickup_time_adjusted	dist_adjusted	dur_adjusted	cancellation_adjusted
1	1	2020-01-01 18:15:34.000	20	32	[NULL]
2	1	2020-01-01 19:10:54.000	20	27	[NULL]
3	1	2020-01-03 00:12:37.000	13.4	20	[NULL]
4	2	2020-01-04 13:53:03.000	23.4	40	[NULL]
5	3	2020-01-08 21:10:57.000	10	15	[NULL]
6	3	[NULL]	[NULL]	[NULL]	Restaurant Cancellation
7	2	2020-01-08 21:30:45.000	25	25	[NULL]
8	2	2020-01-10 00:15:02.000	23.4	15	[NULL]
9	2	[NULL]	[NULL]	[NULL]	Customer Cancellation
10	1	2020-01-11 18:50:20.000	10	10	[NULL]

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 Optimization**
- **Pricing and Ratings**

Pizza Metrics

Q1: How many pizzas were ordered?

```sql
SELECT COUNT(*) AS Total_orders
FROM customer_orders;
```

total_orders
14

Q2: How many unique customer orders were made?

```sql
SELECT COUNT(DISTINCT order_id) AS Unique_Customerorders
FROM customer_orders;
```

unique_customerorders
10

Q3: How many successful orders were delivered by each runner?

```sql
SELECT runner_id,
       COUNT(order_id) AS Successful_orders
FROM runner_orders_new
WHERE cancellation_adjusted IS NULL
GROUP BY runner_id;
```

runner_id	successful_orders
1	4
2	3
3	1

Q4: How many of each type of pizza was delivered?

```
SELECT pn.pizza_name,
       COUNT(co.order_id) AS successful_orders
FROM customer_orders co
JOIN runner_orders_new ro
ON co.order_id = ro.order_id
JOIN pizza_names pn
ON co.pizza_id = pn.pizza_id
WHERE cancellation_adjusted IS NULL
GROUP BY pn.pizza_name;
```

pizza_name	successful_orders
Meatlovers	9
Vegetarian	3

Q5: How many Vegetarian and Meatlovers were ordered by each customer?

```
SELECT co.customer_id,
       COUNT(CASE WHEN pizza_name = 'Vegetarian' THEN 1 END) AS Vegetarian_count,
       COUNT(CASE WHEN pizza_name = 'Meatlovers' THEN 1 END) AS Meatlovers_count
FROM customer_orders co
JOIN runner_orders_new ro ON co.order_id = ro.order_id
JOIN pizza_names pn ON co.pizza_id = pn.pizza_id
GROUP BY co.customer_id;
```

customer_id	vegetarian_count	meatlovers_count
101	1	2
102	1	2
103	1	3
104	0	3
105	1	0

Q6: What was the maximum number of pizzas delivered in a single order?

```
WITH Delivered_pizzas AS (
    SELECT co.order_id,
           COUNT(co.pizza_id) AS pizzas_delivered
    FROM customer_orders co
    JOIN runner_orders_new ro
    ON co.order_id = ro.order_id
    WHERE cancellation_adjusted IS NULL
    GROUP BY co.order_id
)
SELECT
    order_id,
    pizzas_delivered
FROM
    ( SELECT order_id,
             pizzas_delivered,
             RANK() OVER (ORDER BY pizzas_delivered DESC) AS rnk
      FROM Delivered_pizzas
    ) x
WHERE x.rnk = 1;
```

order_id	pizzas_delivered
4	3

Q7: For each customer, how many delivered pizzas had at least 1 change and how many had no changes?

```
SELECT co.customer_id,
      COUNT(CASE WHEN co.exclusions <> '' OR co.extras <> '' THEN 1 END) AS changes,
      COUNT(CASE WHEN co.exclusions = '' AND co.extras = '' THEN 1 END) AS no_changes
FROM customer_orders co
JOIN runner_orders_new ro ON co.order_id = ro.order_id
WHERE cancellation_adjusted IS NULL
GROUP BY co.customer_id;
```

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

Q8: How many pizzas were delivered that had both exclusions and extras?

```
SELECT COUNT(*) AS special_pizzas
FROM customer_orders co
JOIN runner_orders_new ro ON co.order_id = ro.order_id
WHERE co.exclusions <> ''
      AND co.extras <> ''
      AND cancellation_adjusted IS NULL;
```

special_pizzas
1

Q9: What was the total volume of pizzas ordered for each hour of the day?

```
SELECT EXTRACT(HOUR FROM order_time) AS hour_of_the_day,
       COUNT(pizza_id) AS pizzas_count
FROM customer_orders
GROUP BY hour_of_the_day
ORDER BY hour_of_the_day ASC;
```

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

Q10: What was the volume of orders for each day of the week?

```
SELECT TO_CHAR(order_time, 'Day') AS day_of_the_week,
       COUNT(pizza_id) AS pizzas_count
FROM customer_orders
GROUP BY day_of_the_week
ORDER BY day_of_the_week ASC;
```

day_of_the_week	pizzas_count
Friday	1
Saturday	5
Thursday	3
Wednesday	5

Pizza Metrics - Insights

- Out of the **14 pizzas** that were ordered, **10** were unique customer orders.
- Out of the **8 successful orders**, **Runner_1** had the highest number of successful orders.
- **9** Meatlover pizzas and **3** Vegetarian pizzas were sold.
- The maximum number of pizzas delivered in a single order is **3** pizzas.
- There was **1** pizza that was delivered with **both exclusions and extras**.

Runner and Customer Experience

Q1: How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)

```sql
SELECT EXTRACT(WEEK FROM (registration_date + INTERVAL '1 week')) AS week_num,
       COUNT(*) AS signups_per_week
FROM runners
GROUP BY week_num
ORDER BY week_num;
```

week_num	signups_per_week
1	2
2	1
3	1

Q2: What was the average time in minutes it took for each runner to arrive at the Pizza Runner HQ to pick-up the order?

```sql
WITH avg_time_cte AS (
    SELECT ro.runner_id,
           AVG(ro.pickup_time_adjusted - co.order_time) AS avg_time
    FROM runner_orders_new ro
    JOIN customer_orders co ON ro.order_id = co.order_id
    WHERE ro.cancellation_adjusted IS NULL
    GROUP BY ro.runner_id
)
SELECT runner_id,
       CONCAT(EXTRACT(MINUTE FROM avg_time), ' minutes') AS avg_arrival_time
FROM avg_time_cte;
```

runner_id	avg_arrival_time
1	15 minutes
2	23 minutes
3	10 minutes

Q3: Is there any relationship between the number of pizzas and how long the order takes to prepare?

```
WITH relation_cte AS (
    SELECT DISTINCT ro.order_id,
           ro.pickup_time_adjusted - co.order_time AS time_required,
           COUNT(*) OVER (PARTITION BY ro.order_id) AS pizza_count
    FROM runner_orders_new ro
    JOIN customer_orders co ON ro.order_id = co.order_id
    WHERE ro.cancellation_adjusted IS NULL
)
SELECT pizza_count,
       CONCAT(ROUND(AVG(EXTRACT(MINUTE FROM time_required))), ' minutes') AS avg_time_required
FROM relation_cte
GROUP BY pizza_count
ORDER BY pizza_count;
```

123 pizza_count	ABC avg_time_required
1	12 minutes
2	18 minutes
3	29 minutes

Q4: What was the average distance travelled for each customer?

```
SELECT co.customer_id,
       CONCAT(ROUND(AVG(ro.dist_adjusted)::NUMERIC,1),' km') AS avg_distance
FROM runner_orders_new ro
JOIN customer_orders co ON ro.order_id = co.order_id
WHERE ro.cancellation_adjusted IS NULL
GROUP BY co.customer_id
ORDER BY co.customer_id;
```

123 customer_id	ABC avg_distance
101	20.0 km
102	16.7 km
103	23.4 km
104	10.0 km
105	25.0 km

Q5: What was the difference between the longest and shortest delivery times for all orders?

```
SELECT CONCAT(MAX(dur_adjusted),' minutes') AS max_deliverytime,
       CONCAT(MIN(dur_adjusted),' minutes') AS min_deliverytime,
       CONCAT(MAX(dur_adjusted)-MIN(dur_adjusted), ' minutes') AS delivery_timediff
FROM runner_orders_new
WHERE cancellation_adjusted IS NULL;
```

max_deliverytime	min_deliverytime	delivery_timediff
40 minutes	10 minutes	30 minutes

Q6: 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,
       CONCAT(ROUND(AVG((dist_adjusted*60)/dur_adjusted)::NUMERIC,1),' km') AS avg_speed
FROM runner_orders_new
WHERE cancellation_adjusted is null
GROUP BY order_id,runner_id
```

order_id	runner_id	avg_speed
1	1	37.5 km
2	1	44.4 km
3	1	40.2 km
4	2	35.1 km
5	3	40.0 km
7	2	60.0 km
8	2	93.6 km
10	1	60.0 km

Q7: What is the successful delivery percentage for each runner?

```sql
SELECT r.runner_id,
       COUNT(ro.order_id) AS total_orders,
       COUNT(ro.dist_adjusted) AS successful_orders,
       CASE WHEN COUNT(ro.order_id) = 0 THEN NULL
            ELSE CONCAT(ROUND((COUNT(ro.dist_adjusted)::numeric / COUNT(ro.order_id)::numeric) * 100), '%')
            END AS delivery_percentage
FROM runners r
LEFT JOIN runner_orders_new ro ON r.runner_id = ro.runner_id
GROUP BY r.runner_id
ORDER BY r.runner_id;
```

runner_id	total_orders	successful_orders	delivery_percentage
1	4	4	100%
2	4	3	75%
3	2	1	50%
4	0	0	[NULL]

Runner & Customer Experience Insights

- The **1st week** saw the highest number of runners signing up.
- **Runner 2** had the longest time to arrive at the pizza HQ to pick up the order, while **Runner 3** had the shortest time.
- The difference between the longest and shortest delivery times for all orders was **30 minutes**.
- **Runner 1** has the highest delivery percentage.
- The study also revealed the **average distance travelled** and **average speed trends.**

Ingredient Optimization

For the below questions I created virtual table for **pizza_recipes** by splitting the comma-separated records of toppings to distinct rows.

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

```
CREATE VIEW pizza_recipes_new AS
SELECT pizza_id,
       UNNEST(STRING_TO_ARRAY(toppings, ','))::INTEGER AS toppings
FROM pizza_recipes;
```

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

Q1: What are the standard ingredients for each pizza?

```
SELECT pn.pizza_id,
       pn.pizza_name,
       STRING_AGG(pt.topping_name, ',') AS standard_ingredients
FROM pizza_recipes_new pr
INNER JOIN pizza_names pn ON pr.pizza_id = pn.pizza_id
INNER JOIN pizza_toppings pt ON pr.toppings=pt.topping_id
GROUP BY pn.pizza_id,pn.pizza_name;
```

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

Q2: What was the most commonly added extra?

```
WITH extras_cte AS (
  SELECT pizza_id,
         UNNEST(STRING_TO_ARRAY(extras, ','))::INTEGER AS extras
  FROM customer_orders
)
SELECT topping_name AS most_commonly_added_extra,
       COUNT(*) AS times_added
FROM extras_cte ec
INNER JOIN pizza_toppings pt ON pt.topping_id = ec.extras
GROUP BY topping_name
ORDER BY COUNT(*) DESC
LIMIT 1;
```

most_commonly_added_extra	times_added
Bacon	4

Q3: What was the most common exclusion?

```sql
WITH exclusions_cte AS (
  SELECT pizza_id,
         UNNEST(STRING_TO_ARRAY(exclusions, ','))::INTEGER AS exclusions
  FROM customer_orders
)
SELECT topping_name AS most_commonly_added_exclusion,
       COUNT(*) AS times_added
FROM exclusions_cte ec
INNER JOIN pizza_toppings pt ON pt.topping_id = ec.exclusions
GROUP BY topping_name
ORDER BY COUNT(*) DESC
LIMIT 1;
```

most_commonly_added_exclusion	times_added
Cheese	4

Ingredient Optimization Insights

- The study identified the standard ingredients for each type of pizza i.e. Meatlovers and Vegetarian.
- Most Commonly added extra was **Bacon**.
- Most Common exclusion was **Cheese**.

Pricing and Ratings

Q1: If a Meat Lovers pizza costs $12 and Vegetarian costs $10 and there were no charges for changes -how much money has Pizza Runner made so far if there are no delivery fees?

```
WITH Pizza_runner_sales AS (
  SELECT co.order_id,
         pn.pizza_id,
         pn.pizza_name,
         CASE WHEN pn.pizza_name = 'Meatlovers' THEN 12 ELSE 10 END AS price
  FROM customer_orders co
  INNER JOIN runner_orders_new ro ON co.order_id = ro.order_id
  INNER JOIN pizza_names pn ON co.pizza_id = pn.pizza_id
  WHERE ro.cancellation_adjusted IS NULL
)

SELECT CONCAT('$ ', SUM(price)) AS Total_sales
FROM Pizza_runner_sales;
```

total_sales
$ 138

Q2: What if there was an additional $1 charge for any pizza extras?

```
WITH Pizza_runner_sales AS (
    SELECT co.order_id,
           pn.pizza_id,
           pn.pizza_name,
           CASE WHEN pn.pizza_name = 'Meatlovers' THEN 12 ELSE 10 END AS price,
           CASE WHEN LENGTH(extras) = 1 THEN 1
                WHEN LENGTH(extras) > 1 THEN 2
                ELSE 0 END AS extras_price
    FROM customer_orders co
    INNER JOIN runner_orders_new ro ON co.order_id = ro.order_id
    INNER JOIN pizza_names pn ON co.pizza_id = pn.pizza_id
    WHERE ro.cancellation_adjusted IS NULL
)

SELECT CONCAT('$ ', SUM(price) + SUM(extras_price)) AS Total_sales
FROM Pizza_runner_sales;
```

total_sales
$ 142

Q3: The Pizza Runner team now wants to add an additional ratings system that allows customers to rate their runner. Generate a schema for this new table and insert your own data for ratings for each successful customer order between 1 to 5.

```sql
CREATE TABLE rating_system (
    order_id INT DEFAULT NULL,
    customer_id INT DEFAULT NULL,
    runner_id INT DEFAULT NULL,
    rating INT DEFAULT NULL
);

INSERT INTO rating_system
(order_id, customer_id, runner_id, rating)
VALUES
(1, 101, 1, 4),
(2, 101, 1, 5),
(3, 102, 1, 4),
(4, 103, 2, 2),
(5, 104, 3, 3),
(7, 105, 2, 1),
(8, 102, 2, 4),
(10, 104, 1, 5);
```

order_id	customer_id	runner_id	rating
1	101	1	4
2	101	1	5
3	102	1	4
4	103	2	2
5	104	3	3
7	105	2	1
8	102	2	4
10	104	1	5

Q4: Using 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
WITH order_details AS (
  SELECT co.customer_id,
         co.order_id,
         ro.runner_id,
         co.order_time,
         ro.pickup_time_adjusted AS pickup_time,
         ro.pickup_time_adjusted - co.order_time AS order_pickup_duration,
         ro.dur_adjusted AS delivery_duration,
         ROUND((ro.dist_adjusted / (ro.dur_adjusted / 60.0))::NUMERIC, 1) AS avg_speed,
         COUNT(*) OVER (PARTITION BY ro.order_id) AS total_orders,
         ROW_NUMBER() OVER (PARTITION BY ro.order_id) AS rn
  FROM customer_orders co
  RIGHT JOIN runner_orders_new ro ON co.order_id = ro.order_id
  WHERE ro.cancellation_adjusted IS NULL
)

SELECT o.customer_id,
       o.order_id,
       o.runner_id,
       rs.rating,
       o.order_time,
       o.pickup_time,
       o.order_pickup_duration,
       o.delivery_duration,
       o.avg_speed,
       o.total_orders
FROM order_details o
INNER JOIN rating_system rs ON o.order_id = rs.order_id
WHERE o.rn = 1;
```

Q5: 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 kilometer traveled , how much money does Pizza Runner have left over after these deliveries?

```sql
WITH pizza_income_cte AS (
  SELECT co.order_id,
         ro.runner_id,
         ro.dist_adjusted,
         pn.pizza_name,
         CASE WHEN pn.pizza_name = 'Meatlovers' THEN 12 ELSE 10 END AS pizza_price
  FROM customer_orders co
  INNER JOIN runner_orders_new ro ON co.order_id = ro.order_id
  INNER JOIN pizza_names pn ON co.pizza_id = pn.pizza_id
  WHERE ro.cancellation_adjusted IS NULL
),

pizza_income_per_order_cte AS
(
  SELECT SUM(pizza_price) AS sales_per_order,
         AVG(dist_adjusted)*0.30 AS runner_payment
  FROM pizza_income_cte
  GROUP BY order_id
)

SELECT CONCAT('$ ',SUM(sales_per_order) - SUM(runner_payment)) AS Profit
FROM pizza_income_per_order_cte;
```

ABC profit
$ 94.44

Pricing and Ratings Insights

- According to the Pizza Runner case study, if a Meatlovers Pizza costs **$12** and a Vegetarian Pizza costs **$10**, and there are no charges for changes, then the Pizza Runner will make **$138**. However, if there is an additional charge of **$1** for any extras added, then the Pizza Runner will make **$142**.
- The study also incorporated a **Rating system** that enables customers to rate their runner.

THANK YOU!