

Pizza Runner & Sales Analysis

By:

Rahul Kumar

21CHB0B41

NIT Warangal



Overview

This project report demonstrate the analytic skills using SQL queries to uncover insights from the dataset.

Platform used: MySQL Workbench 2024

Acknowledgement

- The dataset is taken from 8weeksqlchallenge(Case Study 2).
- It is publicly accessible from 8weeksqlchallenge.com, which reaffirms its authenticity.



About the website: 8weeksqlchallenge.com

The website **8weeksqlchallenge.com** was created by Danny Ma, an experienced data analyst from Australia. It offers eight case study problems that cover a range of real-life scenarios with their databases. The site is designed to help people start their journey with SQL by providing questions that guide users in uncovering insights from the provided databases or ER diagrams.

In addition to the case studies, the website includes resources and explanations to help users understand and practice SQL. It's a valuable tool for anyone looking to improve their SQL skills through practical, hands-on exercises.

The case study includes:

- Case Study #1 - Danny's Diner
- Case Study #2 - Pizza Runner
- Case Study #3 - Foodie-Fi
- Case Study #4 - Data Bank
- Case Study #5 - Data Mart
- Case Study #6 - Clique Bait
- Case Study #7 - Balanced Tree Clothing Co.
- Case Study #8 - Fresh Segments

About the CASE STUDY 2: Pizza Runner

This case study focuses on a fictional pizza delivery app business. It features data from various pizza-selling stores and includes information on customers, orders, cancellations, pizzas, and runners.

(In this case, a "runner" refers to a pizza store rather than an individual delivery person.)

The case study provides a comprehensive set of data to help analyze different aspects of the pizza delivery business, from order management to customer interactions.

Fun Fact:

Did you know that over **115 million kilograms** of pizza is consumed daily worldwide??? (Well according to Wikipedia anyway...)

Business Problem

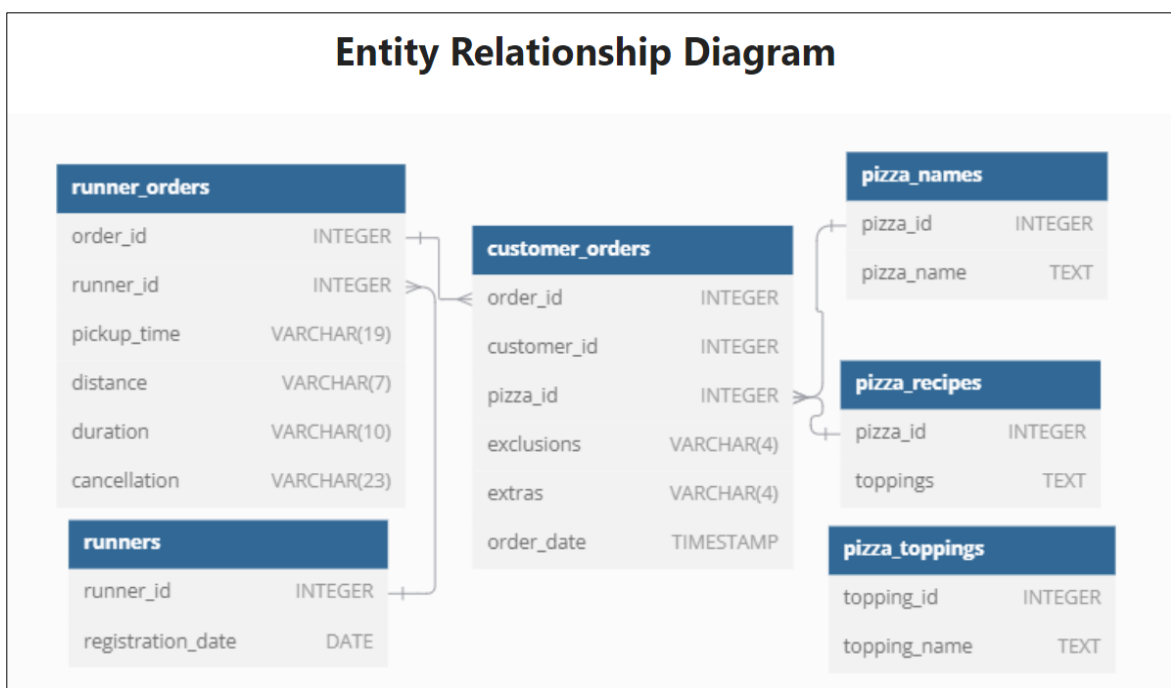
The goal of this project is to enhance data analysis skills by working with a dataset. This involves preparing the database, cleaning and transforming data, and writing queries to extract valuable insights. The main objective is to improve the efficiency of Pizza Runner's operations.

Database:

The database is set up using the SQL code provided in the data dump available on the website. It was imported into MySQL Workbench 2024 for analysis.

About the dataset:

The Entity-Relation diagram of the dataset is:



Description and Instances of the Tables

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.

order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1			2021-01-01 18:05:02
2	101	1			2021-01-01 19:00:52
3	102	1			2021-01-02 23:51:23
3	102	2		NaN	2021-01-02 23:51:23
4	103	1	4		2021-01-04 13:23:46
4	103	1	4		2021-01-04 13:23:46
4	103	2	4		2021-01-04 13:23:46
5	104	1	null	1	2021-01-08 21:00:29
6	101	2	null	null	2021-01-08 21:03:13
7	105	2	null	1	2021-01-08 21:20:29
8	102	1	null	null	2021-01-09 23:54:33
9	103	1	4	1, 5	2021-01-10 11:22:59
10	104	1	null	null	2021-01-11 18:34:49
10	104	1	2, 6	1, 4	2021-01-11 18:34:49



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.

Table 3: runner_orders

After each orders are received through the system - they are assigned to a runner - however not all orders are fully completed and can be cancelled by the restaurant or the customer.

The pickup_time is the timestamp at which the runner arrives at the Pizza Runner headquarters to pick up the freshly cooked pizzas. The distance and duration fields are related to how far and long the runner had to travel to deliver the order to the respective customer.

There are some known data issues with this table so be careful when using this in your queries - make sure to check the data types for each column in the schema SQL!

order_id	runner_id	pickup_time	distance	duration	cancellation
1	1	2021-01-01 18:15:34	20km	32 minutes	
2	1	2021-01-01 19:10:54	20km	27 minutes	
3	1	2021-01-03 00:12:37	13.4km	20 mins	NaN
4	2	2021-01-04 13:53:03	23.4	40	NaN
5	3	2021-01-08 21:10:57	10	15	NaN
6	3	null	null	null	Restaurant C
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 Ca
10	1	2020-01-11 18:50:20	10km	10minutes	null

Table 4: pizza_names

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

pizza_id	pizza_name
1	Meat Lovers
2	Vegetarian

Table 5: pizza_recipes

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

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

Table 6: pizza_toppings

This table contains all of the topping_name values with their corresponding topping_id value

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



Questions to be answered:

A. Pizza Metrics

1. How many pizzas were ordered?
2. How many unique customer orders were made?
3. How many successful orders were delivered by each runner?
4. How many of each type of pizza was delivered?
5. How many Vegetarian and Meatlovers were ordered by each customer?
6. What was the maximum number of pizzas delivered in a single order?
7. For each customer, how many delivered pizzas had at least 1 change and how many had no changes?
8. How many pizzas were delivered that had both exclusions and extras?
9. What was the total volume of pizzas ordered for each hour of the day?
10. What was the volume of orders for each day of the week?

B. Runner and Customer Experience

1. How many runners signed up during each week, starting from January 1, 2021?
2. What was the average time, in minutes, for each runner to arrive at Pizza Runner HQ to pick up an order?
3. Is there a correlation between the number of pizzas ordered and the time it takes to prepare the order?
4. What is the average distance traveled by each customer?
5. What is the difference between the longest and shortest delivery times for all orders?
6. What is the average speed of each runner for their deliveries, and do you observe any trends in these speeds?
7. What percentage of deliveries was successful for each runner?

C. Ingredient Optimisation

1. What are the standard ingredients for each pizza?
2. What was the most commonly added extra?
3. What was the most common exclusion?

Data Preparation and Data Cleaning

Things to be fixed in the Tables:-

1. Replace '' and NaN values with null
2. Handle the string consistency in runner_orders table (mins, min, minutes)
3. Check for the datatypes
4. Handle multivalued column in the pizza_recipes table

Replace '' and NaN values with null

I used 'CASE Statement' clause for replacing empty cells and NaN with null. Below is the query I wrote for this:

```
/* Cleaning Customer_Orders Table */
```

```
create table cleaned_cust_ord (  
  select  
    order_id,  
    customer_id,  
    pizza_id,  
    order_time,  
    case  
      when extras = '' then null  
      when extras = 'NaN' then null  
      else extras  
    end as cleaned_extras,  
    case  
      when exclusions = '' then null  
      when exclusions = 'NaN' then null  
      else exclusions  
    end as cleaned_exclusions  
  from customer_orders);
```

Handling string consistency

I used regexp function in MySQL to check for string consistency. Here is the brief explanation of regexp function:

Breaking Regexp function:

1. `^`: Asserts the start of the string.
2. `[A-Za-z]`: Matches exactly one letter (either uppercase or lowercase).
3. `[A-Za-z0-9_.-]*`: Matches zero or more of the following characters: letters, digits, underscores (`_`), periods (`.`), or hyphens (`-`).
4. `$`: Asserts the end of the string.

Note:

- `[]` square brackets are used to denote a character class, which matches any single character contained within the brackets.
- The `*` quantifier here means that this group of characters can repeat as many times as needed, including zero times.
- The dot `.` is escaped with a double backslash to indicate that it should be treated as a literal dot rather than a wildcard character.

```
/* Cleaning Revenue_Orders Table */
CREATE TABLE cleaned_runner_orders(
  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);
```

Altering data types

```
-- Altering data types
ALTER TABLE pizza_names
  modify COLUMN pizza_name VARCHAR(100);

ALTER TABLE pizza_recipes
  modify COLUMN toppings VARCHAR(100);

ALTER TABLE pizza_toppings
  modify COLUMN topping_name VARCHAR(100);
```

Handle multivalued column

```
CREATE TABLE pizza_ingredients AS
WITH RECURSIVE numbers AS (
  SELECT 1 AS n
  UNION ALL
  SELECT n + 1
  FROM numbers
  WHERE n < 10
)
, split_toppings AS (
  SELECT
    pr.pizza_id,
    pn.pizza_name,
    CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(pr.toppings, ',', numbers.n), ',', -1) AS UNSIGNED) AS topping_id
  FROM
    pizza_recipes pr
  JOIN
    numbers ON CHAR_LENGTH(pr.toppings) - CHAR_LENGTH(REPLACE(pr.toppings, ',', '')) >= numbers.n - 1
  JOIN
    pizza_names pn ON pr.pizza_id = pn.pizza_id
)
SELECT
  st.pizza_name,
  GROUP_CONCAT(pt.topping_name ORDER BY pt.topping_id ASC) AS topping_names,
  GROUP_CONCAT(st.topping_id ORDER BY st.topping_id ASC) AS topping_ids
FROM
  split_toppings st
JOIN
  pizza_toppings pt ON st.topping_id = pt.topping_id
GROUP BY
  st.pizza_name
ORDER BY
  st.pizza_name;
```

Answering the query:

```
/*-----  
Pizza Runner  
-----*/
```

-- 1. How many pizzas were ordered?

```
SELECT  
    COUNT(order_id) AS total_orders  
FROM  
    customer_orders;
```

Output:

	total_orders
▶	14

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

```
SELECT  
    COUNT(DISTINCT order_id) AS unique_orders  
FROM  
    cleaned_cust_ord;
```

Output:

Result Grid	
	unique_orders
▶	10

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

```
SELECT  
    runner_id, COUNT(runner_id) AS successful_orders  
FROM  
    cleaned_runner_orders  
WHERE  
    cancellation IS NULL  
GROUP BY runner_id;
```

Output:

	runner_id	successful_orders
▶	1	4
	2	3
	3	1

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

```
SELECT
    pizza_name, COUNT(*)
FROM
    cleaned_runner_orders cro
    JOIN
    cleaned_cust_ord cco ON cro.order_id = cco.order_id
    JOIN
    pizza_names pn ON cco.pizza_id = pn.pizza_id
WHERE
    cancellation IS NULL
GROUP BY pizza_name;
```

	pizza_name	COUNT(*)
▶	Meatlovers	9
	Vegetarian	3

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

```
WITH cte1 as (SELECT
    customer_id, pizza_name, COUNT(pizza_name) AS no_of_orders
FROM
    cleaned_cust_ord cco
    INNER JOIN
    cleaned_runner_orders cro ON cco.order_id = cro.order_id
    INNER JOIN
    pizza_names pn ON cco.pizza_id = pn.pizza_id
GROUP BY customer_id , pizza_name)
```

```
SELECT
    customer_id, pizza_name, no_of_orders
FROM
    cte1
GROUP BY customer_id , pizza_name
ORDER BY customer_id , pizza_name;
```

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

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

```
• SELECT
    cro.order_id, COUNT(pizza_id) AS num_pizza_ordered
FROM
    cleaned_runner_orders cro
    JOIN
    cleaned_cust_ord cco ON cro.order_id = cco.order_id
GROUP BY order_id
ORDER BY num_pizza_ordered DESC
LIMIT 1;
```

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

```
• With cte2 as
    (select
        customer_id,
        case
            when cleaned_exclusions is null and cleaned_extras is null then 'no change'
            else 'atleast one change'
        end as pizza_topping_changes
    from cleaned_cust_ord c
    inner join cleaned_runner_orders cro on c.order_id = cro.order_id
    where cancellation is null)
```

```
SELECT
    customer_id,
    pizza_topping_changes,
    COUNT(pizza_topping_changes) AS num
FROM
    cte2
GROUP BY customer_id , pizza_topping_changes
ORDER BY customer_id , pizza_topping_changes;
```

	customer_id	pizza_topping_changes	num
▶	101	no change	2
	102	atleast one change	1
	102	no change	2
	103	atleast one change	3
	104	atleast one change	3
	105	atleast one change	1

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

```
SELECT
    COUNT(*) AS pizza_with_exclusions_and_extras
FROM
    cleaned_cust_ord cco
    INNER JOIN
    cleaned_runner_orders cro ON cco.order_id = cro.order_id
WHERE
    cleaned_exclusions IS NOT NULL
    AND cleaned_extras IS NOT NULL
    AND cancellation IS NULL;
```

	pizza_with_exclusions_and_extras
▶	5

-- 9. 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(*)
FROM
    customer_orders
GROUP BY hour_of_the_day
ORDER BY hour_of_the_day;
```

	hour_of_the_day	COUNT(*)
▶	11	1
	13	3
	18	3
	19	1
	21	3
	23	3

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

```
SELECT
    DATE_FORMAT(order_time, '%W') AS day_of_week, COUNT(*)
FROM
    cleaned_cust_ord
GROUP BY day_of_week
ORDER BY day_of_week;
```

	day_of_week	COUNT(*)
▶	Friday	1
	Saturday	5
	Thursday	3
	Wednesday	5

```
/* -----  
    RUNNER AND CUSTOMER EXPERIENCE  
-----*/  
  
-- How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)  
SELECT  
    FLOOR((DATEDIFF(registration_date, '2021-01-01') / 7) + 1) AS week_num,  
    COUNT(runner_id) AS runner_count  
FROM runners  
GROUP BY week_num  
ORDER BY week_num;
```

	week_num	runner_count
▶	1	2
	2	1
	3	1



Conclusion:

In my "Pizza Store Sales Analysis" project for January 2024, I focused on transforming and analyzing data to uncover key insights that could drive business decisions. I began by cleaning the dataset, handling null values, and splitted the multivalued column data using recursive functions and the GROUP_CONCAT function. This allowed me to normalize the data, making it easier to analyze critical metrics such as order volumes, runner performance, and ingredient usage. Through this process, I identified patterns in customer preferences, delivery times, and ingredient demand, which laid the foundation for my analysis.

Based on the insights generated, I observed that certain pizzas were consistently ordered during peak hours, while delivery times varied significantly depending on distance and runner availability. To optimize operations, I recommended adjusting staffing levels during peak times, implementing route optimization for runners, and refining ingredient inventory management to prevent shortages of popular toppings. These suggestions aimed to improve overall efficiency, enhance customer satisfaction, and reduce operational costs for the pizza store.

