**Pizza Runner Case Study**

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…

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.

Because Danny had a few years of experience as a data scientist - he was very aware that data collection was going to be critical for his business’ growth.

He has prepared for us an entity relationship diagram of his database design but requires further assistance to clean his data and apply some basic calculations so he can better direct his runners and optimise Pizza Runner’s operations.

All datasets exist within the**pizza\_runner**database schema - be sure to include this reference within your SQL scripts as you start exploring the data and answering the case study questions.

The **runners**information shows the **registration\_date** for each new runner. And 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.

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!

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

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

This table contains all of the **topping\_name** values with their corresponding **topping\_id** value

**QUERIES**

1. How many pizzas were ordered by customer?
2. How many successful orders were delivered by each runner?
3. Find greater duration of all runner\_order whose distance is 20km
4. Find the detail of runner\_order who is exsist at least one customer\_order
5. To find all customer\_order detail where exclusion is not null or extras = 1
6. To find the count of pizza\_id in customers\_order table according to group by order\_id where order\_id groups according to exclusion = 4
7. How many of each type of pizza was delivered?
8. How many Vegetarian and Meatlovers were ordered by each customer?
9. create a view that shows all customer\_order and runner\_order information in the one view.
10. Create stored procedure in runner\_order using stored procedure to insert the one row in runner\_order

**COMPANY DATA STORAGE REQUIREMENT CASE STUDY**

The company is organized into branches. Each branch has a unique number, a name, and a particular employee who manages it.

The company makes it’s money by selling to clients. Each client has a name and a unique number to identify it.

The foundation of the company is it’s employees. Each employee has a name, birthday, gender, salary and a unique number.

An employee can work for one branch at a time, and each branch will be managed by one of the employees that work there. We’ll also want to keep track of when the current manager started as manager.

An employee can act as a supervisor for other employees at the branch, an employee may also act as the supervisor for employees at other branches. An employee can have at most one supervisor.

A branch may handle a number of clients, with each client having a name and a unique number to identify it. A single client may only be handled by one branch at a time.

Employees can work with clients controlled by their branch to sell them stuff. If nescessary multiple employees can work with the same client. We’ll want to keep track of how many dollars worth of stuff each employee sells to each client they work with.

Many branches will need to work with suppliers to buy inventory. For each supplier we’ll keep track of their name and the type of product they’re selling the branch. A single supplier may supply products to multiple branches.

**QUERIES**

1. Find all employee's id's and names who were born after 1969
2. Find all employees who are female & born after 1969 or who make over 80000
3. Find all employees named Jim, Michael, Johnny or David
4. Find all clients who are handles by the branch that Michael Scott manages

Assume you know Michael's ID

1. Find a list of employee and branch names
2. To find the sum of total\_sales in work\_with table according to group by client\_id
3. Find any employee born on the 10th day of the month
4. create a view that shows all brunch\_supplier and client information in the one view.