SQL CASE STUDY #1

Pizza Runner

Data with Danny

ARCHIS RUDRA

www.8weeksqlchallenge.com

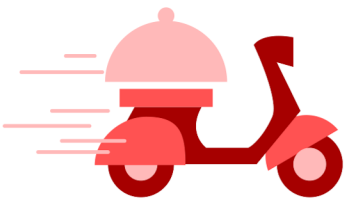
Introduction

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

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.

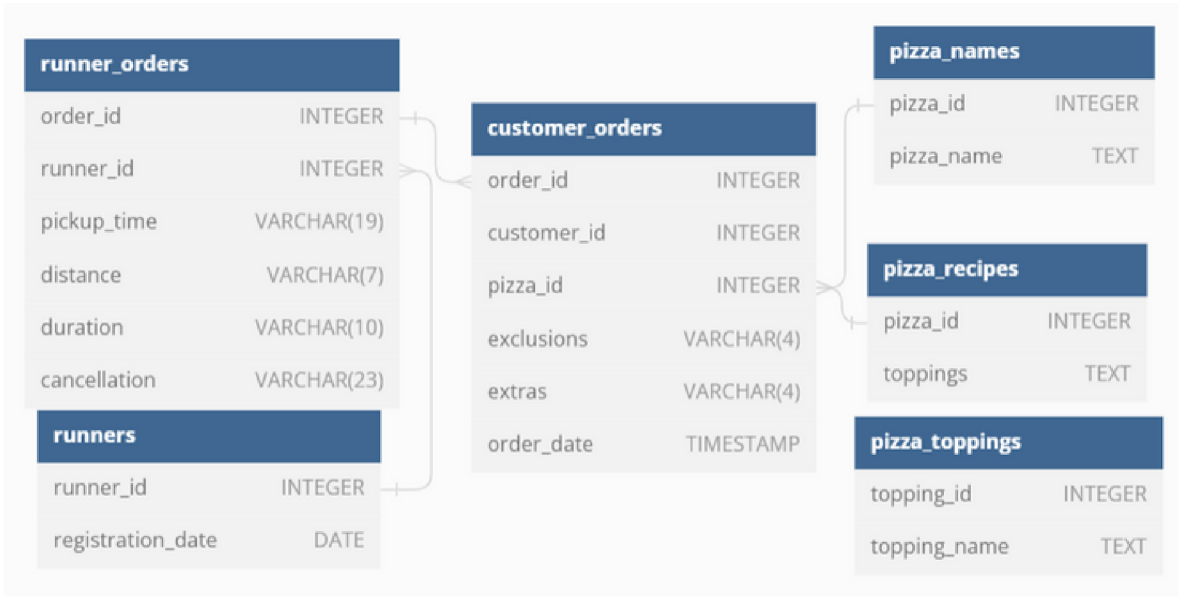


Problem Statement

Danny, a data scientist with a few years of experience, knew that data collection was essential for his business's growth. He created an **entity relationship diagram** (ERD) for his database design, but he needed help cleaning the data and applying basic calculations. This would allow him to better direct his runners and optimize Pizza Runner's operations.

I am **Archis Rudra**, a data analyst. I will use my SQL skills to help Danny make data-driven decisions. I will answer his questions and help him clean and analyze his data. This will allow Danny to make better decisions about how to run his business.

Entity Relationship Diagram



Datasets

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

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 that should be removed from the pizza and the **extras** are the **ingredient_id** values that 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!

order_id	customer_id	pizza_id	exclusions	extras	order_time
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

runners_orders:

After each order are received through the system - they are assigned to a runner - however not all orders are fully completed and can be canceled 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	25m	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

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

pizza_recipes:

Each **pizza_id** has a standard set of **toppings** that 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

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

Exploratory Data Analysis

Data Cleaning was a bit tricky for this project task. I have handled the missing and null values of tables (i.e., **runner_orders** and **customer_orders**) in order to clean the data.

before

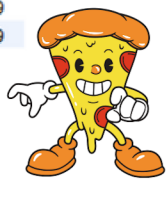
customer_orders:

order_id	customer_id	pizza_id	exclusions	extras	order_time
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

```sql
update customer_orders
set exclusions = null
where exclusions = '';

update customer_orders
set exclusions = null
where exclusions = "null";

update customer_orders
set extras = null
where extras = '';

update customer_orders
set extras = null
where extras = "null";
```

order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1	NULL	NULL	2020-01-01 18:05:02
2	101	1	NULL	NULL	2020-01-01 19:00:52
3	102	1	NULL	NULL	2020-01-02 23:51:23
3	102	2	NULL	NULL	2020-01-02 23:51:23
4	103	1	4	NULL	2020-01-04 13:23:46
4	103	1	4	NULL	2020-01-04 13:23:46
4	103	2	4	NULL	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

after



before

runner_orders:

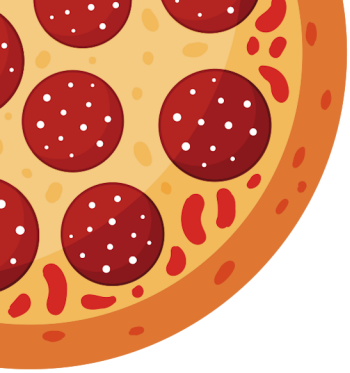
order_id	runner_id	pickup_time	distance	duration	cancellation
1	1	2020-01-01 18:15:34	20km	32 minutes	
2	1	2020-01-01 19:10:54	20km	27 minutes	
3	1	2020-01-03 00:12:37	13.4km	20 mins	NULL
4	2	2020-01-04 13:53:03	23.4	40	NULL
5	3	2020-01-08 21:10:57	10	15	NULL
6	3	null	null	null	Restaurant Cancellation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

```
update runner_orders
set pickup_time = null
where exclusions = "null";

update runner_orders
set distance = null
where distance = "null";

update runner_orders
set duration = null
where duration = "null";
```

```
update runner_orders
set cancellation = null
where cancellation = "null";

update runner_orders
set cancellation = null
where cancellation = '';
```

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

after



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 Optimisation**
- **Pricing and Ratings**

Each of the following case study questions can be answered using a single SQL statement.

Pizza Metrics

1. How many pizzas were ordered?

pizza_runner

Limit to 1000 rows

```
1 •  select count(*) as total_pizzas_ordered
2    from customer_orders;
3
4
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

total_pizzas_ordered
14

2. How many unique customer orders were made?

pizza_runner*

Limit to 1000 rows

```
1 •  select count(distinct customer_id) as total_customers
2    from customer_orders;
3
4
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

total_customers
5

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

pizza_runner*

Limit to 1000 rows

```
1 •  select runner_id, count(*) as successful_delivered
2    from runner_orders
3    where cancellation is null
4    group by 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

runner_id	successful_delivered
1	4
2	3
3	1

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

```sql
select pizza_name, count(*) as pizza_delivered
from customer_orders c
left join pizza_names p using(pizza_id)
where order_id in (
                    select order_id
                    from pizza_runner.runner_orders
                    where cancellation is null)
group by 1;
```

pizza_name	pizza_delivered
Meatlovers	9
Vegetarian	3

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

```sql
select customer_id, pizza_name, count(*) as pizza_ordered
from customer_orders c
join pizza_names p using(pizza_id)
group by 1,2 with rollup
having pizza_name is not null;
```

customer_id	pizza_name	pizza_ordered
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?

```sql
with delivered as (
    select order_id, count(*) as pizza_delivered
    from customer_orders
    where order_id in (
                        select order_id
                        from pizza_runner.runner_orders
                        where cancellation is null)
    group by 1)
select max(pizza_delivered) as max_pizza_deliver
from delivered;
```

max_pizza_deliver
3

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

```
select customer_id,
    sum(case when exclusions is null and extras is null then 1 else 0 end) as no_changes,
    sum(case when (exclusions is null and extras is not null) or
                  (exclusions is not null and extras is null) or
                  (exclusions is not null and extras is not null) then 1 else 0 end) as atleast_one_change
from customer_orders
where order_id in (
                    select order_id
                    from pizza_runner.runner_orders
                    where cancellation is null)
```

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

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

```
select sum(case when exclusions is not null and extras is not null then 1 end) as no_of_pizzas
from customer_orders
where order_id in (
                    select order_id
                    from pizza_runner.runner_orders
                    where cancellation is null);
```

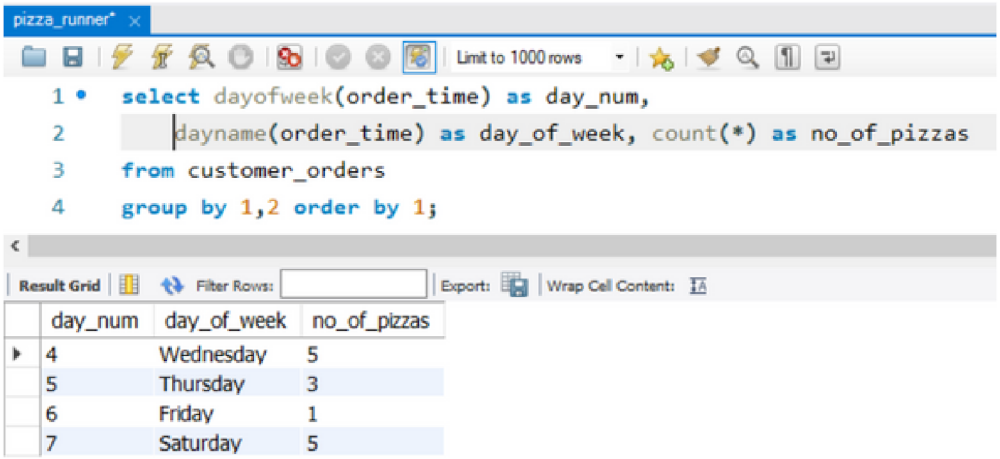
no_of_pizzas
1

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

```
select hour(order_time) as hour_of_day, count(*) as no_of_pizzas
from customer_orders
group by 1 order by 1;
```

hour_of_day	no_of_pizzas
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?



day_num	day_of_week	no_of_pizzas
4	Wednesday	5
5	Thursday	3
6	Friday	1
7	Saturday	5

INSIGHTS GATHERED

- A total of **14 pizzas** were ordered.
- A total of **5 unique customers** ordered pizzas.
- The number of successful orders was **eight** with which ***Runner 1*** has the *highest* number of successful orders delivered.
- Meatlovers type of pizza was sold **nine units** whereas Vegetarian was sold **three units**.
- In a single order, the *maximum* number of pizzas delivered was **three**.
- **One** single pizza was delivered that had both exclusions and extras.
- **Wednesday** has delivered the highest number of pizzas.

Runner and Customer Experience

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

```
select weekofyear(registration_date + interval 1 week) as week_num, count(*) as runners_signed
from runners
group by 1;
```

week_num	runners_signed
1	2
2	1
3	1

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

```
with avgtime as (
    select distinct r.order_id, r.runner_id,
        timestampdiff(minute, order_time, pickup_time) as minutes
    from runner_orders r
    inner join customer_orders c using(order_id)
    where cancellation is null)
select runner_id, round(avg(minutes)) as average_time_minutes
from avgtime
```

runner_id	average_time_minutes
1	14
2	20
3	10

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

```
select pizza_count, round(avg(time_required)) as avg_time_required
from (select distinct r.order_id,
        timestampdiff(minute, order_time, pickup_time) as time_required,
        count(*) over(partition by order_id) as pizza_count
    from runner_orders r
    inner join customer_orders c using(order_id)
    where cancellation is null) as relation
group by 1;
```

pizza_count	avg_time_required
1	12
2	18
3	29

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

```
select customer_id, round(avg(distance),1) as avg_distance
from runner_orders r
join customer_orders c using(order_id)
where distance is not null
group by 1 order by 1;
```

customer_id	avg_distance
101	20
102	16.7
103	23.4
104	10
105	25

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

```
select max(duration) as longest_delivery_times,
       min(duration) as shortest_delivery_time,
       (max(duration) - min(duration)) as difference
from runner_orders;
```

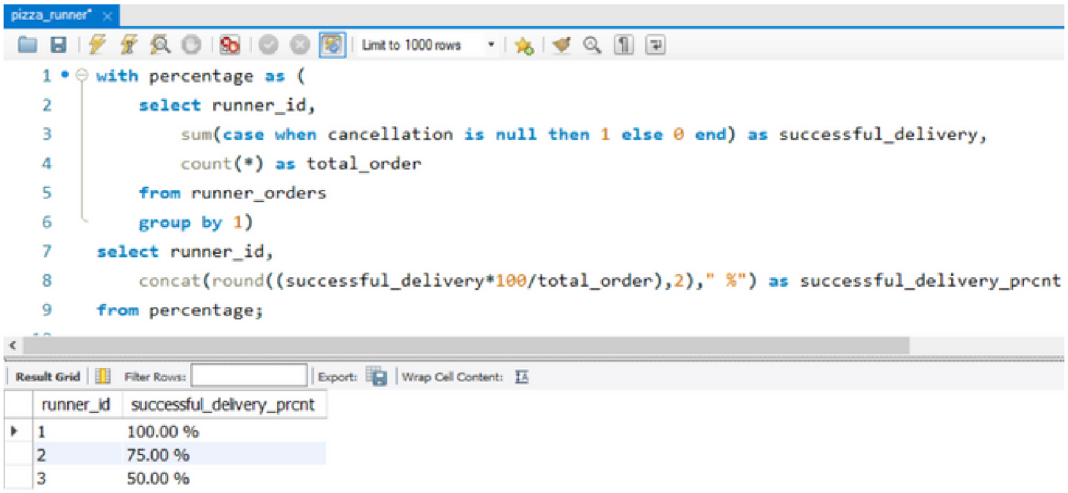
longest_delivery_times	shortest_delivery_time	difference
40 min	10 min	30

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

```
with average as (
    select order_id, runner_id,
        round((distance * 60)/duration, 2) as speed_kmph
    from runner_orders
    where cancellation is null
    )
select *, round(avg(speed_kmph) over(partition by runner_id),2) as avg_speed
from average;
```

order_id	runner_id	speed_kmph	avg_speed
1	1	37.5	45.54
2	1	44.44	45.54
3	1	40.2	45.54
10	1	60	45.54
4	2	35.1	62.9
7	2	60	62.9
8	2	93.6	62.9
5	3	40	40

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



```
with percentage as (
    select runner_id,
        sum(case when cancellation is null then 1 else 0 end) as successful_delivery,
        count(*) as total_order
    from runner_orders
    group by 1)
select runner_id,
    concat(round((successful_delivery*100/total_order),2)," %") as successful_delivery_prcnt
from percentage;
```

runner_id	successful_delivery_prcnt
1	100.00 %
2	75.00 %
3	50.00 %

INSIGHTS GATHERED

- In the **first week**, there is the *highest* number of runners were signed.
- **Pizza Runner 2** takes a *longer* time whereas **Pizza Runner 3** takes the *shortest* time to arrive at Pizza HQ to pick up the order.
- The relationship between the number of pizzas and the time required to prepare is **positively correlated**.
- The difference between the *longest* and *shortest* delivery times for all orders was **30 minutes**.
- **Pizza Runner 1** has the *highest* successful delivery percentage.

Ingredient Optimisation

So, to gather information about the ingredients we need to transform the tables (i.e., **pizza_recipes** and **customer_orders**) and make them as a virtual table (**VIEW**) based on the result-set of an SQL statement.

pizza_recipes:

before

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

```
create view pizza_recipe_new as (
select r.pizza_id, trim(j.topping) as topping
from pizza_recipes r
join json_table(trim(replace(json_array(r.toppings), ',', '","')),
                '$[*]' columns (topping varchar(50) path '$')) as j);
```

pizza_id	topping
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

create view as **pizza_receipe_new**

before

customer_orders:

order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1	NULL	NULL	2020-01-01 18:05:02
2	101	1	NULL	NULL	2020-01-01 19:00:52
3	102	1	NULL	NULL	2020-01-02 23:51:23
3	102	2	NULL	NULL	2020-01-02 23:51:23
4	103	1	4	NULL	2020-01-04 13:23:46
4	103	1	4	NULL	2020-01-04 13:23:46
4	103	2	4	NULL	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

```sql
create view customer_order_new as (
select c.order_id, c.customer_id, c.pizza_id, trim(j.exclusions) as exclusion,
       trim(k.extras) as extras, order_time
from customer_orders c
join json_table(trim(replace(json_array(c.exclusions), ',', '","')),
                '$[*]' columns (exclusions varchar(50) path '$')) as j
join json_table(trim(replace(json_array(c.extras), ',', '","')),
                '$[*]' columns (extras varchar(50) path '$')) as k);
```

order_id	customer_id	pizza_id	exclusion	extras	order_time
1	101	1	NULL	NULL	2020-01-01 18:05:02
2	101	1	NULL	NULL	2020-01-01 19:00:52
3	102	1	NULL	NULL	2020-01-02 23:51:23
3	102	2	NULL	NULL	2020-01-02 23:51:23
4	103	1	4	NULL	2020-01-04 13:23:46
4	103	1	4	NULL	2020-01-04 13:23:46
4	103	2	4	NULL	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	2020-01-10 11:22:59
9	103	1	4	5	2020-01-10 11:22:59
10	104	1	NULL	NULL	2020-01-11 18:34:49
10	104	1	2	1	2020-01-11 18:34:49
10	104	1	2	4	2020-01-11 18:34:49
10	104	1	6	1	2020-01-11 18:34:49
10	104	1	6	4	2020-01-11 18:34:49

create view as
customer_order_new

1. What are the standard ingredients for each pizza?

```
select p2.pizza_name,
    group_concat(p3.topping_name separator ', ') as standard_ingredients
from pizza_recipe_new p1
inner join pizza_names p2 using(pizza_id)
inner join pizza_toppings p3 on p3.topping_id = p1.topping
group by 1;
```

pizza_name	standard_ingredients
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 topping_name as most_commonly_added_extra
from pizza_toppings
where topping_id = (
                    select extras
                    from (
                        select extras, count(*) as times_used
                        from customer_order_new
                        where extras is not null
                        group by 1 order by 2 desc limit 1)
                    as t);
```

most_commonly_added_extra
Bacon

3. What was the most common exclusion?

```
select topping_name as most_common_exclusion
from pizza_toppings
where topping_id = (
                    select exclusion
                    from (
                        select exclusion, count(*) as times_used
                        from customer_order_new
                        where exclusion is not null
                        group by 1 order by 2 desc limit 1)
                    as t);
```

most_common_exclusion
Cheese

4. Generate an order item for each record in the customers_orders table in the format of one of the following:

o Meat Lovers

o Meat Lovers - Exclude Beef

o Meat Lovers - Extra Bacon

o Meat Lovers - Exclude Cheese, Bacon - Extra Mushroom, Peppers.

pizza_runner*

Limit to 1000 rows

```
with generate_cust_order as (
    select distinct con.order_id, con.customer_id, con.pizza_id,
           concat("Exclude ",pt.topping_name) as exclusion,
           con.extras, con.order_time
    from customer_order_new con
    left join pizza_toppings pt
    on con.exclusion = pt.topping_id
)
select distinct gc.order_id, gc.customer_id, gc.pizza_id,
       gc.exclusion as exclusion,
       concat("Extra ", pt.topping_name) as extras, gc.order_time
from generate_cust_order gc
left join pizza_toppings pt
on gc.extras = pt.topping_id;
```

order_id	customer_id	pizza_id	exclusion	extras	order_time
1	101	1	NULL	NULL	2020-01-01 18:05:02
2	101	1	NULL	NULL	2020-01-01 19:00:52
3	102	1	NULL	NULL	2020-01-02 23:51:23
3	102	2	NULL	NULL	2020-01-02 23:51:23
4	103	1	Exclude Cheese	NULL	2020-01-04 13:23:46
4	103	2	Exclude Cheese	NULL	2020-01-04 13:23:46
5	104	1	NULL	Extra Bacon	2020-01-08 21:00:29
6	101	2	NULL	NULL	2020-01-08 21:03:13
7	105	2	NULL	Extra Bacon	2020-01-08 21:20:29
8	102	1	NULL	NULL	2020-01-09 23:54:33
9	103	1	Exclude Cheese	Extra Bacon	2020-01-10 11:22:59
9	103	1	Exclude Cheese	Extra Chicken	2020-01-10 11:22:59
10	104	1	NULL	NULL	2020-01-11 18:34:49
10	104	1	Exclude BBQ Sauce	Extra Bacon	2020-01-11 18:34:49
10	104	1	Exclude BBQ Sauce	Extra Cheese	2020-01-11 18:34:49
10	104	1	Exclude Mushrooms	Extra Bacon	2020-01-11 18:34:49
10	104	1	Exclude Mushrooms	Extra Cheese	2020-01-11 18:34:49

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

```sql
with updation as (
    select distinct r.order_id, duration, c.pizza_id,
    length(c.exclusions) - length(replace(c.exclusions, ',', '')) + 1 as total_exclusions,
    length(c.extras) - length(replace(c.extras, ',', '')) + 1 as total_extras,
    length(p.toppings) - length(replace(p.toppings, ',', '')) + 1 as total_toppings
    from runner_orders r
    join customer_orders c using(order_id)
    join pizza_recipes p on p.pizza_id = c.pizza_id
    where r.cancellation is null
    order by duration asc)
select order_id, pizza_id, total_exclusions, total_extras,
case when total_exclusions = total_extras then total_toppings
     when total_exclusions is null and total_extras is null then total_toppings
     when total_exclusions is null and total_extras >= 1 then (total_toppings + total_extras)
     else (total_toppings - total_exclusions)
     end as toppings_required
from updation;
```

order_id	pizza_id	total_exclusions	total_extras	toppings_required
10	1	2	2	8
10	1	NULL	NULL	8
8	1	NULL	NULL	8
5	1	NULL	1	9
3	1	NULL	NULL	8
3	2	NULL	NULL	6
7	2	NULL	1	7
2	1	NULL	NULL	8
1	1	NULL	NULL	8
4	1	1	NULL	7
4	2	1	NULL	5

INSIGHTS GATHERED

- The most commonly added extra was "**Bacon**".
- The most common exclusion was "**Cheese**".
- The maximum number of toppings added was **nine**.

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?

```
pizza_runner*
Limit to 1000 rows
1  with total as (
2      with pizza_details as (
3          select *
4          from runner_orders
5          where cancellation is null)
6      select distinct *, if(pizza_id = 1, 12, 10) as pizza_cost
7      from pizza_details pd
8      join customer_orders co using(order_id))
9  select runner_id, sum(pizza_cost) as total_earning
10 from total
11 group by 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

runner_id	total_earning
1	70
2	44
3	12

What if there was an additional $1 charge for any pizza extras?
o Add cheese is $1 extra

```
pizza_runner*
Limit to 1000 rows
1  with additional_details as (with pizza_details as (
2          select *
3          from runner_orders
4          where cancellation is null)
5      select distinct *, if(pizza_id = 1, 12, 10) as pizza_cost,
6          case when co.extras is null then 0
7               else (length(co.extras) - length(replace(co.extras, ',', '')) + 1)
8               end as additional_cost
9      from pizza_details pd
10     join customer_orders co using(order_id))
11 select runner_id, sum(pizza_cost+additional_cost) as total_cost
12 from additional_details
13 group by 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

runner_id	total_cost
1	72
2	45
3	13

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

```
create view runner_orders_new as (
    select *, if(cancellation is null, floor(1 + rand() * 5), null) as ratings
from runner_orders);
```

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

4. If a Meat Lovers pizza was $12 and a 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?

pizza_runner*

Limit to 1000 rows

```
with leftover as (
    with total as(
        select distinct c.order_id, r.runner_id, c.pizza_id, r.distance,
        if(c.pizza_id = 1, 12, 10) as pizza_price, (r.distance*0.30) as runner_paid
        from customer_orders c
        join runner_orders r using(order_id)
        where cancellation is null)
    select runner_id, sum(pizza_price) as total_pizza_price,
        sum(runner_paid) as total_runner_paid
    from total
    group by 1)
select runner_id,
    concat("$ ",round(total_pizza_price - total_runner_paid, 2)) as left_over_money
from leftover;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

runner_id	left_over_money
1	$ 34.96
2	$ 15.44
3	$ 9

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

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

pizza_runner*

Limit to 1000 rows

```
1  with cte as (select distinct con.customer_id, ron.order_id,
2          ron.runner_id, ron.ratings, con.order_time,
3          ron.pickup_time,
4          concat(timestampdiff(minute, con.order_time, ron.pickup_time)," min") as time_diff_order_and_pickup,
5          ron.duration as delivery_duration, round(ron.distance*60/ron.duration,1) as speed_kmph,
6          count(*) over(partition by ron.order_id) as total_no_of_pizzas
7  from customer_order_new con
8  right join runner_orders_new ron using(order_id)
9  where cancellation is null)
10 select customer_id, order_id, runner_id, round(avg(ratings)) as ratings,
11         order_time, pickup_time, time_diff_order_and_pickup,
12         delivery_duration, speed_kmph, total_no_of_pizzas
13 from cte
14 group by 1,2,3,5,6,7,8,9,10;
```

customer_id	order_id	runner_id	ratings	order_time	pickup_time	time_diff_order_and_pickup	delivery_duration	speed_kmph	total_no_of_pizzas
101	1	1	3	2020-01-01 18:05:02	2020-01-01 18:15:34	10 min	32 min	37.5	1
101	2	1	1	2020-01-01 19:00:52	2020-01-01 19:10:54	10 min	27 min	44.4	1
102	3	1	2	2020-01-02 23:51:23	2020-01-03 00:12:37	21 min	20 min	40.2	2
103	4	2	3	2020-01-04 13:23:46	2020-01-04 13:53:03	29 min	40 min	35.1	3
104	5	3	3	2020-01-08 21:00:29	2020-01-08 21:10:57	10 min	15 min	40	1
105	7	2	4	2020-01-08 21:20:29	2020-01-08 21:30:45	10 min	25 min	60	1
102	8	2	4	2020-01-09 23:54:33	2020-01-10 00:15:02	20 min	15 min	93.6	1
104	10	1	2	2020-01-11 18:34:49	2020-01-11 18:50:20	15 min	10 min	60	5

INSIGHTS GATHERED

- Pizza **Runner 1** has earned **$ 70** among the other two runners if there is ***no extra charge*** for the extras.
- Pizza **Runner 1** has earned **$ 72** among the other two runners if there is ***an extra charge*** for the extras.
- Pizza **Runner 3** has **$ 9** left after all the deliveries.

Thank You!!

Keep Learning and Happy SQL'ing