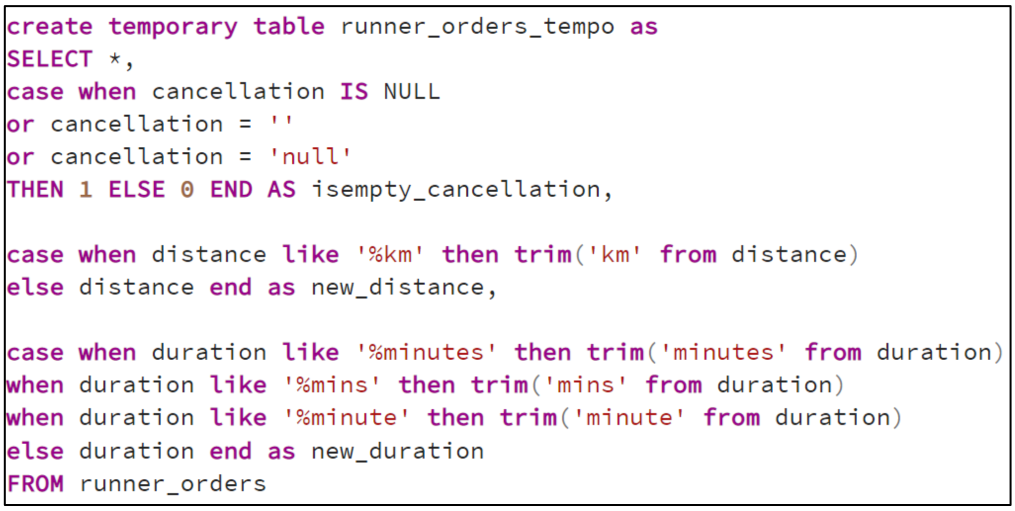
DANNY PIZZA RUNNER

Preprocessing

- Handling of missing and unsupported values like blank values
- Converting data types of string to integer/float (exp. Distance of 23.5km to 23.5)

For above problems, I created temporary tables with new columns using case statements. For example, exclusions have numeric, null, string written as ‘null’ and blank space values. Hence, I used null, string written as ‘null’ and blank space values as 1 and for any exclusion as 0. So that if I join this table with any other table I can just use ‘where exclusion = 0/1’ to filter out my data.

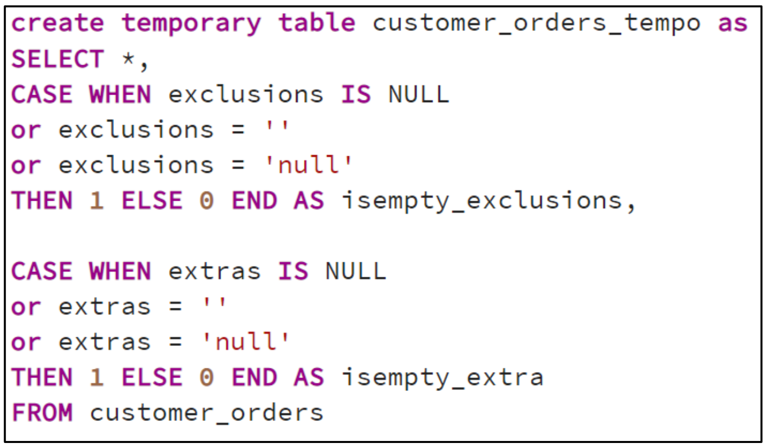
I made 2 temporary tables - customer_orders_tempo and runner_orders_tempo.



```
create temporary table runner_orders_tempo as
SELECT *,
case when cancellation IS NULL
or cancellation = ''
or cancellation = 'null'
THEN 1 ELSE 0 END AS isempty_cancellation,

case when distance like '%km' then trim('km' from distance)
else distance end as new_distance,

case when duration like '%minutes' then trim('minutes' from duration)
when duration like '%mins' then trim('mins' from duration)
when duration like '%minute' then trim('minute' from duration)
else duration end as new_duration
FROM runner_orders
```



```
create temporary table customer_orders_tempo as
SELECT *,
CASE WHEN exclusions IS NULL
or exclusions = ''
or exclusions = 'null'
THEN 1 ELSE 0 END AS isempty_exclusions,

CASE WHEN extras IS NULL
or extras = ''
or extras = 'null'
THEN 1 ELSE 0 END AS isempty_extra
FROM customer_orders
```

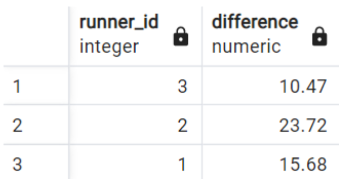
From above two tables, I made 5 new columns for cancellation, distance, duration, exclusions and extras in two different tables.

In following pages, I have only included questions that I found interesting and challenging with explanation.

TABLE OF CONTENTS

Q1- 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 runner_id, round(avg(extract(epoch from
        (cast(ro.pickup_time as timestamptz)) - (cast(co.order_time as timestamptz))) /60),2) as difference
from customer_orders co
natural join runner_orders_tempo ro
where ro.pickup_time != 'null'
group by runner_id;
```

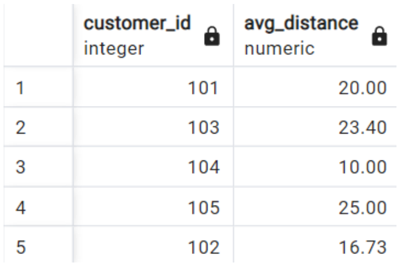


	runner_id integer	difference numeric
1	3	10.47
2	2	23.72
3	1	15.68

Explanation – To find average time for a runner to pick an order, I subtract order time from pickup time and groupby with each runner. In this query, I needed to convert data type of pickup and order time to timestamp and extract difference in seconds using epoch and convert difference in minutes by dividing it with 60. I also needed to join customers and runners table and 'where' clause to filter out cancelled orders.

Q2- What was the average distance travelled for each customer?

```
select customer_id, round(cast(avg(cast(new_distance as float)) as decimal),2) as avg_distance
from customer_orders co
natural join runner_orders_tempo ro
where new_duration != 'null'
group by customer_id;
```



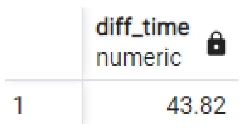
	customer_id integer	avg_distance numeric
1	101	20.00
2	103	23.40
3	104	10.00
4	105	25.00
5	102	16.73

Explanation – To find average distance for each customer, I used average distance travelled and groupby with each customer. In this query, I needed to convert data type of distance to float and average it for each customer. I also needed to join customers and runners table and 'where' clause to filter out cancelled orders.

Q3- What was the difference between the longest and shortest delivery times for all orders?

```
with cte as
(select *, extract(epoch from
        (cast(ro.pickup_time as timestamptz)) - (cast(co.order_time as timestamptz))) /60 as diff
from runner_orders_tempo ro
natural join customer_orders co
where new_duration != 'null')

select round(cast(max(total_time_diff) - min(total_time_diff) as decimal),2) as diff_time
from (select cast(new_duration as float) + diff as total_time_diff from cte) t
```



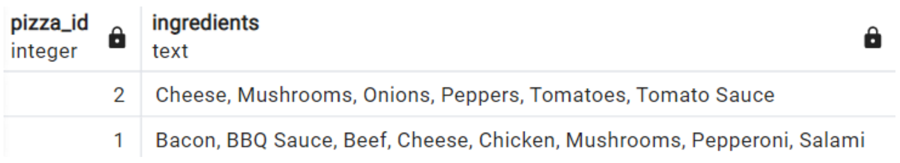
	diff_time numeric
1	43.82

Explanation – To find out the difference between longest and shortest delivery time from all orders, we need to first find out longest delivery time and shortest delivery time. To do that, I used CTE to first find out difference between order and pickup time for each order in minutes and filter out to exclude cancelled orders. Then I used this subquery to add delivery time to each of the time which I got from CTE to get total delivery time taken from ordering and till order delivery. Then I calculated maximum and minimum time from the resulted subquery.

Q4- What are the standard ingredients for each pizza?

```
with cte as (SELECT pizza_id, toppings_separated
FROM pizza_recipes
CROSS JOIN LATERAL unnest(string_to_array(toppings, ',')) as p(toppings_separated))

select pizza_id, string_agg(topping_name, ', ') as ingredients from cte
inner join pizza_toppings pt
on cast(cte.toppings_separated as int) = pt.topping_id
group by pizza_id;
```



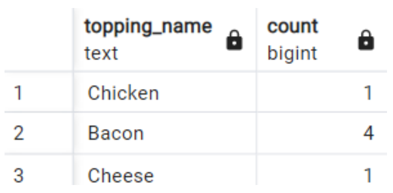
pizza_id integer	ingredients text
2	Cheese, Mushrooms, Onions, Peppers, Tomatoes, Tomato Sauce
1	Bacon, BBQ Sauce, Beef, Cheese, Chicken, Mushrooms, Pepperoni, Salami

Explanation – Originally in pizza recipes table, I had ingredients IDs written in place of topping names. So, to replace them with their names I first needed to split comma separated ingredients for each pizza id to converted into rows. I did that in the CTE using string_to_array and unnest and named the column as toppings separated. Next, I joined the CTE table with pizza toppings table to find out the names of toppings for each pizza id. Then I used string_agg to add all pizza toppings separated with comma grouping by pizza id.

Q5- What was the most commonly added extra?

```
with cte as (SELECT *
FROM customer_orders_tempo
CROSS JOIN LATERAL unnest(string_to_array(isempty_extra, ',')) as p(isempty_extra_separated))

select topping_name, count(topping_name) from cte
left join pizza_toppings pt
on cast(cte.isempty_extra_separated as int) = pt.topping_id
where isempty_extra_separated != 'no_extra'
group by topping_name;
```



	topping_name text	count bigint
1	Chicken	1
2	Bacon	4
3	Cheese	1

Explanation – To find out most commonly I used customers_orders temporary table and pizza toppings table. In customer orders table I have isempty_extra column with topping ids separated with comma. I used string_to_array and unnest to return each topping id in each row and named the column as isempty_extra_seperated. Then I joined pizza toppings table to get names of the topping names and counted topping names grouped by topping names to get count of each. Hence, I founded Bacon is most commonly added extra.

Q6- 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

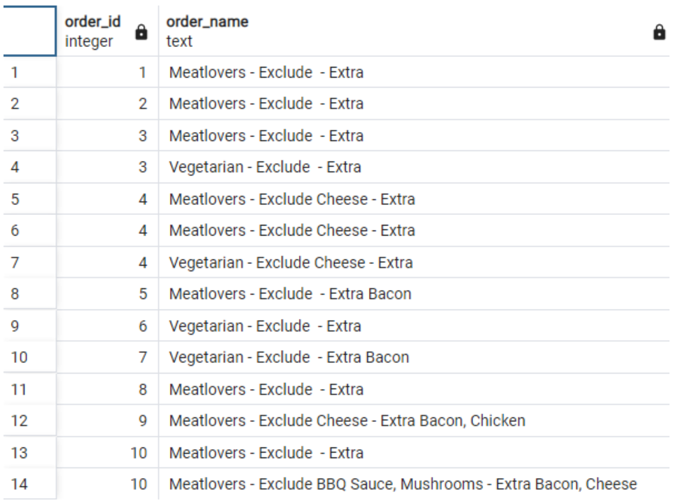
Meat Lovers - Extra Bacon

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

```
create temporary table customer_orders_tempo_2 as
SELECT *,
CASE WHEN exclusions IS NULL
or exclusions = ''
or exclusions = 'null'
THEN '0' ELSE exclusions END AS isempty_exclusions,
CASE WHEN extras IS NULL
or extras = ''
or extras = 'null'
THEN '0' ELSE extras END AS isempty_extra
FROM customer_orders
```

```sql
with cte as (select *,
CASE WHEN exclusion_2 = ''
THEN '0' ELSE exclusion_2 END AS exclusion_2_new,
CASE WHEN extra_2 = ''
THEN '0' ELSE extra_2 END AS extra_2_new
from
(select co.order_id, co.customer_id, pn.pizza_id, pn.pizza_name,
row_number() over (partition by order_id order by order_id) as row_id,
split_part(isempty_exclusions, ',', 1) as exclusion_1,
split_part(isempty_exclusions, ',', 2) as exclusion_2,
split_part(isempty_extra, ',', 1) as extra_1,
split_part(isempty_extra, ',', 2) as extra_2
from customer_orders_tempo_2 co
inner join pizza_names pn
on co.pizza_id = pn.pizza_id) t)

select order_id,
concat(pizza_name , ' - Exclude ',
       string_agg(distinct(exclusions), ', ') , ' - Extra ',
       string_agg(distinct(extras), ', ')) as order_name
from
(select *, pt.topping_name as exclusions, pt2.topping_name as extras from cte
left join pizza_toppings pt
on cast(exclusion_1 as int) = pt.topping_id
or
cast(exclusion_2_new as int) = pt.topping_id
left join pizza_toppings pt2
on cast(extra_1 as int) = pt2.topping_id
or
cast(extra_2_new as int) = pt2.topping_id) t
group by order_id, pizza_name, row_id
order by order_id, pizza_name, row_id
```

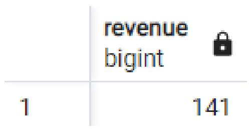


	order_id integer	order_name text
1	1	Meatlovers - Exclude - Extra
2	2	Meatlovers - Exclude - Extra
3	3	Meatlovers - Exclude - Extra
4	3	Vegetarian - Exclude - Extra
5	4	Meatlovers - Exclude Cheese - Extra
6	4	Meatlovers - Exclude Cheese - Extra
7	4	Vegetarian - Exclude Cheese - Extra
8	5	Meatlovers - Exclude - Extra Bacon
9	6	Vegetarian - Exclude - Extra
10	7	Vegetarian - Exclude - Extra Bacon
11	8	Meatlovers - Exclude - Extra
12	9	Meatlovers - Exclude Cheese - Extra Bacon, Chicken
13	10	Meatlovers - Exclude - Extra
14	10	Meatlovers - Exclude BBQ Sauce, Mushrooms - Extra Bacon, Cheese

Explanation – To generate an output like above, I have to concat pizza names, extras names and exclusions names. To concat extra and exclusions names, first I had to separate comma separated ids into rows for each order id. Then join pizza toppings tables to get their names and again add them in comma separated format with some strings in it. To do this I first made temporary table, then cte with a subquery in it. In the subquery I separated extras and exclusions in rows to get each of them for each order id. I also assigned row_id using row number partitioned by order_id as in each id their could be two orders with same pizzas. Outside this subquery I made two more columns using case statement such that blank values will be 0. In second subquery I joined pizza topping to get names against each topping. Outside this subquery, I concated in above pattern grouping by order_id, pizza_name and row id.

Q7- What if Meat Lovers pizza costs $12, Vegetarian costs $10 and an additional $1 charge for any pizza extras, how much money has Pizza Runner made so far if there are no delivery fees?

```
select sum(price) + sum(xtra_price) as revenue from (select *,
case when pizza_name = 'Meatlovers' then 12
else 10 end as price,
case when isempty_extra = '0' then 0
else 1 end as xtra_price
from customer_orders_tempo_2 co
left join pizza_names pn
on co.pizza_id = pn.pizza_id) t
left join runner_orders_tempo ro
on t.order_id = ro.order_id
where isempty_cancellation = 1;
```

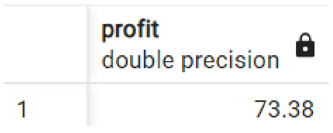


	revenue bigint
1	141

Explanation – For pricing purpose I made two new columns in subquery that are price and xtra_price using case statements and join to get pizza names. Then I used this table and join it with runners orders to exclude cancelled orders. At last, I added total price by adding price and xtra_price columns and got $141 as total revenue.

Q8- 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?

```
select sum(price) - 0.3*(sum(cast(new_distance as float))) as profit
from (select *,
case when pizza_name = 'Meatlovers' then 12
else 10 end as price
from customer_orders_tempo_2 co
left join pizza_names pn
on co.pizza_id = pn.pizza_id) t
left join runner_orders_tempo ro
on t.order_id = ro.order_id
where new_distance != 'null'
```

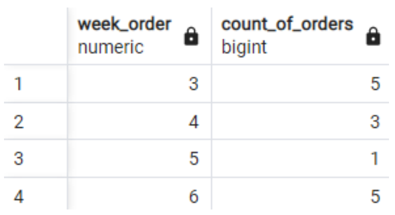


	profit double precision
1	73.38

Explanation – In this question again I used case statement to set up price for pizzas like in previous question. Only difference is that there no cost for extras for customers but pizza runner gives $0.3 per kilometer traveled to their runners. Hence, I needed to find total distance covered by all runners. I first convert the data type of new_distance to float and added them to find out 30% of it. Then I subtract this number from total revenue and got $73.38 as profit earned.

Q9- What was the volume of orders for each day of the week?

```sql
select extract(dow from order_time) as week_order, count(order_id) as count_of_orders
from customer_orders_tempo
group by extract(dow from order_time)
order by extract(dow from order_time) asc;
--(0-6; Sunday is 0)
```

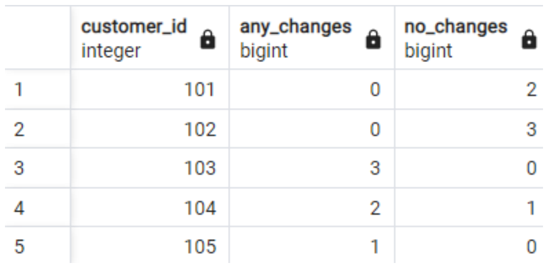


	week_order numeric	count_of_orders bigint
1	3	5
2	4	3
3	5	1
4	6	5

Explanation – Firstly to extract week number I extracted ‘dow’ from the order_time. Then I grouped by weeks and counted the number of orders as count_of_orders. From here, I got to know that in week 3 and 6 i.e. (Wednesday and Saturdat) there were maximum number of orders.

Q10- For each customer, how many delivered pizzas had at least 1 change, and how many had no changes?

```sql
select customer_id, sum(Any_Changes) as Any_Changes, sum(No_Changes) as No_Changes from
(select * ,case when isempty_exclusions = 0 or isempty_extra = 0
then 1 else 0 end as Any_Changes,
case when isempty_exclusions = 1 and isempty_extra = 1
then 1 else 0 end as No_Changes
from customer_orders_tempo
natural join runner_orders_tempo rot
where isempty_cancellation = 1) t
group by customer_id
```

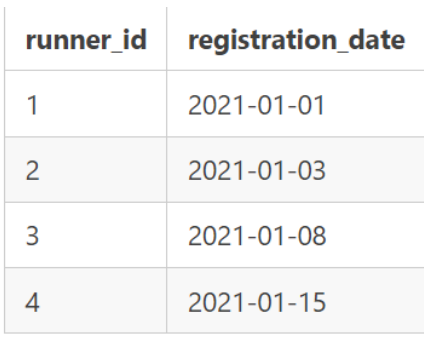


	customer_id integer	any_changes bigint	no_changes bigint
1	101	0	2
2	102	0	3
3	103	3	0
4	104	2	1
5	105	1	0

Explanation – At least 1 change means either there is any change in extras or in exclusions or in both. No change is quite simple. To do that I used case statement so that it will create two columns for each condition and in each column for a desired condition it will pass 1, so that when I add that column values it will give me total changes or no changes. To find sum I used subquery grouping by each order.

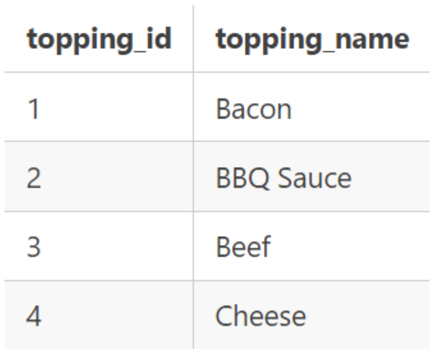
Table snippets

runners



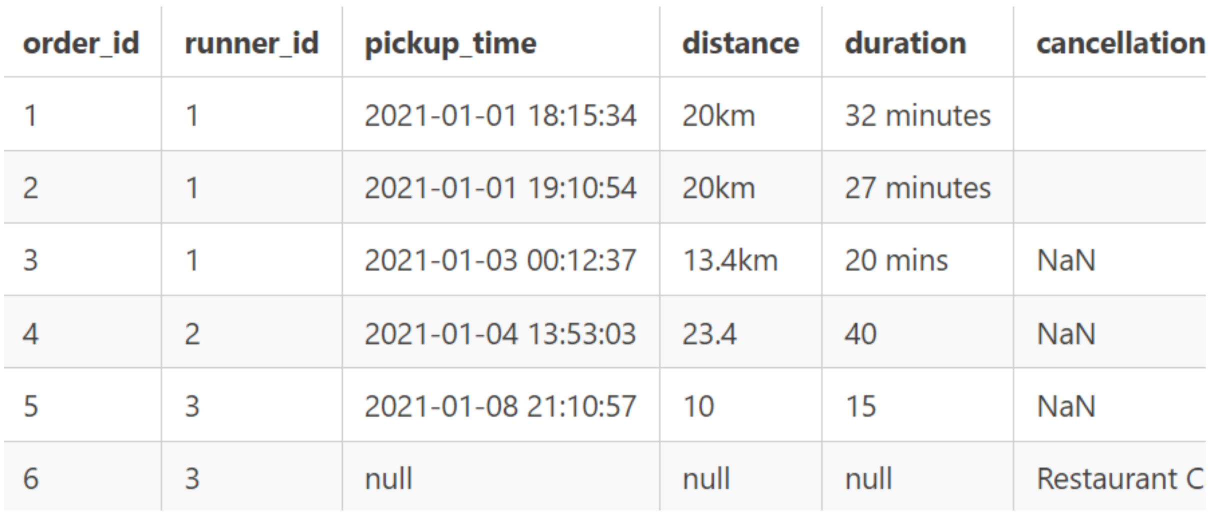
runner_id	registration_date
1	2021-01-01
2	2021-01-03
3	2021-01-08
4	2021-01-15

pizza_toppings



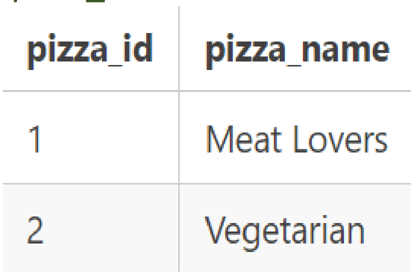
topping_id	topping_name
1	Bacon
2	BBQ Sauce
3	Beef
4	Cheese

runner_orders



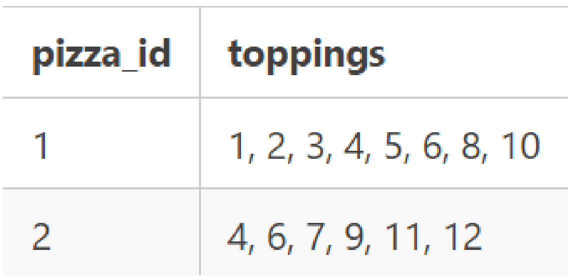
order_id	runner_id	pickup_time	distance	duration	cancellation
1	1	2021-01-01 18:15:34	20km	32 minutes	
2	1	2021-01-01 19:10:54	20km	27 minutes	
3	1	2021-01-03 00:12:37	13.4km	20 mins	NaN
4	2	2021-01-04 13:53:03	23.4	40	NaN
5	3	2021-01-08 21:10:57	10	15	NaN
6	3	null	null	null	Restaurant C

pizza_names



pizza_id	pizza_name
1	Meat Lovers
2	Vegetarian

pizza_recipes

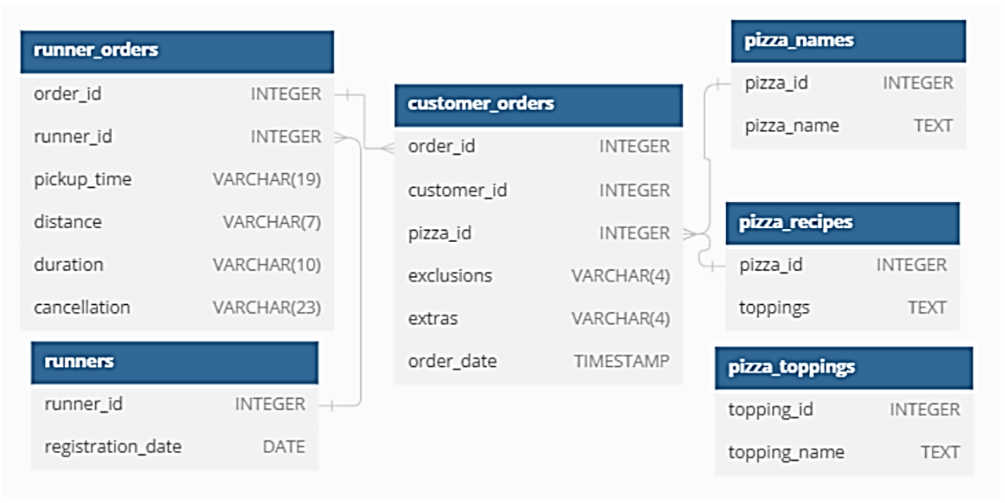


pizza_id	toppings
1	1, 2, 3, 4, 5, 6, 8, 10
2	4, 6, 7, 9, 11, 12

customer_orders

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

Entity Relationship Diagram



runner_orders
order_id INTEGER
runner_id INTEGER
pickup_time VARCHAR(19)
distance VARCHAR(7)
duration VARCHAR(10)
cancellation VARCHAR(23)
runners
runner_id INTEGER
registration_date DATE
customer_orders
order_id INTEGER
customer_id INTEGER
pizza_id INTEGER
exclusions VARCHAR(4)
extras VARCHAR(4)
order_date TIMESTAMP
pizza_names
pizza_id INTEGER
pizza_name TEXT
pizza_recipes
pizza_id INTEGER
toppings TEXT
pizza_toppings
topping_id INTEGER
topping_name TEXT