DATAWITHDANNY.COM

START YOUR SQL ENGINES

PIZZA RUNNER

CASE STUDY #2

8 WEEK SQL CHALLENGE

8WEEKSQLCHALLENGE.COM

INTRODUCTION

Did you know that over 115 million kilograms of pizza is 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.

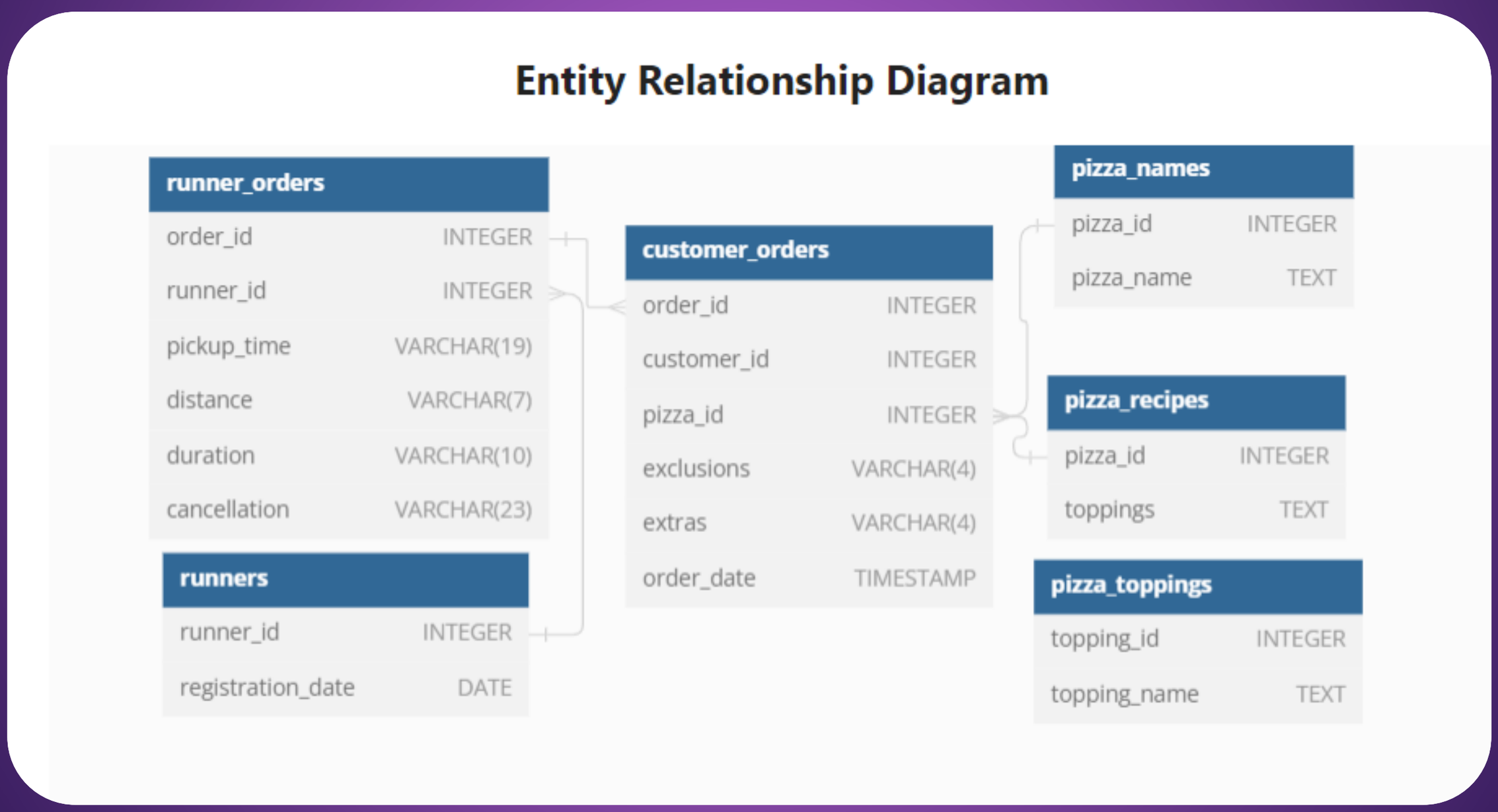
PROBLEM STATEMENT

Enhancing Customer Insights for Pizza Runner's Strategic Growth Danny, the mastermind behind Pizza Runner, envisions more than just a pizza empire.

He's delving into customer behaviour, spending patterns, and menu preferences to elevate Pizza Runner's strategic growth.

The data framework, including runner_orders, runners, customer_orders, pizza_names, pizza_recipes, and pizza_toppings, holds the key to unlocking these insights. Danny has shared with you 6 key datasets for this case study: runner_orders, runners, customer_orders, pizza_names, pizza_recipes, pizza_toppings

ENITITY RELATIONSHIP DIAGRAM



DATA CLEANING

```sql
-- Data cleaning of customer_orders table

create temporary table customer_orders_temp_tbl as
select  trim(order_id) as order_id,
        customer_id,
        pizza_id,
        case
            when exclusions = '' then null
            when exclusions = 'null' then null
            else exclusions
        end as exclusions,
        case
            when extras = '' then null
            when extras = 'null' then null
            else extras
        end as extras,
        order_time
    from customer_orders;

select * from customer_orders_temp_tbl;
```

```sql
-- Creating seperate rows for each value in ' exclusion' and 'extras'

create table customer_orders_cleaned as
 select co.order_id,
        co.customer_id,
        co.pizza_id,
        trim(jc1.exclusions) as exclusions,
        trim(jc2.extras) as extras,
        co.order_time
from customer_orders_temp_tbl co
inner join json_table(trim(replace(json_array(co.exclusions), ',','","')), '$[*]' columns(exclusions varchar(50) path '$')) jc1
inner join json_table(trim(replace(json_array(co.extras), ',','","')), '$[*]' columns(extras varchar(50) path '$')) jc2;
```

```sql
select * from customer_orders_cleaned;
```

Result Grid | Filter Rows: | Edit: | Export/I

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
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
NULL	NULL	NULL	NULL	NULL	NULL

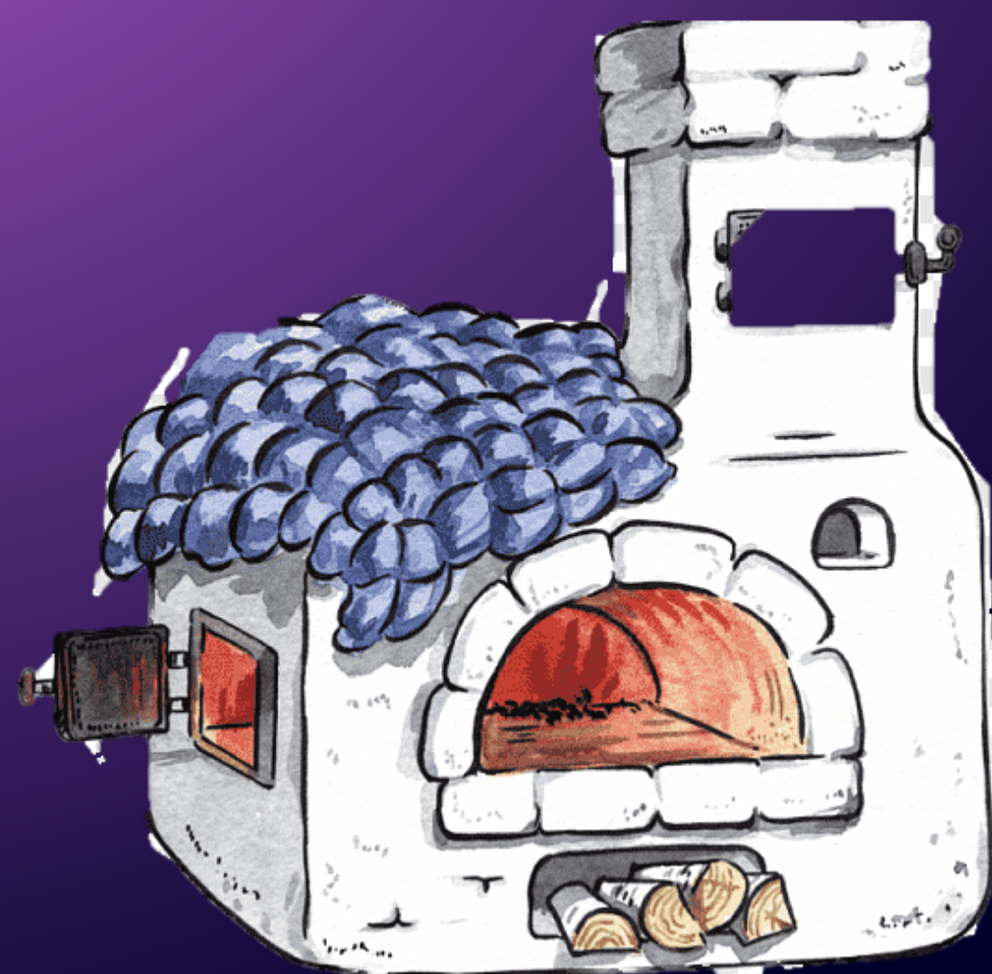
tomer_orders_cleaned 2

```sql
-- cleaning runner_orders

select * from runner_orders;

create table runner_orders_cleaned as
    select order_id,
           runner_id,
           case
                when distance = 'null' then null
                else cast(regexp_replace(distance,'[a-z]','') as float)
           end as distance,
           case
                when duration = 'null' then null
                else cast(regexp_replace(duration,'[a-z]','') as float)
           end as duration,
           case
                when cancellation = 'null' then null
                when cancellation = '' then null
                else cancellation
            end as cancellation
    from runner_orders;

    select * from runner_orders_cleaned;
```

order_id	runner_id	pickup_time	distance	duration	cancellation
1	1	2020-01-01 18:15:34	20	32	NULL
2	1	2020-01-01 19:10:54	20	27	NULL
3	1	2020-01-03 00:12:37	13.4	20	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	25	25	NULL
8	2	2020-01-10 00:15:02	23.4	15	NULL
9	2	NULL	NULL	NULL	Customer Cancellation
10	1	2020-01-11 18:50:20	10	10	NULL

```sql
--  Data cleaning of pizza_receipe

select * from pizza_recipes;

-- Concept of using json array

select * ,
        json_array(toppings),
        replace(json_array(toppings),',','","'),
        trim(replace(json_array(toppings),',','","'))
from pizza_recipes;

-- Actual code for string to rows transformation of 'toppings'

create table pizza_recipes_cleaned as
    select pr.pizza_id, jpr.toppings
    from pizza_recipes pr
    join json_table(trim(replace(json_array(toppings),',','","')),'$[*]' columns (toppings varchar(50) path '$')) jpr;

select * from pizza_recipes_cleaned;
```

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



A. PIZZA METRICS

```sql
# A. Pizza Metrics
-- How many pizzas were ordered?

SELECT
    COUNT(pizza_id) AS Total_Pizza_Orderd
FROM
    customer_orders_cleaned;
```

Result Grid | Filter Rows: | Export: | W

Total_Pizza_Orderd
15

```sql
-- How many unique customer orders were made?

SELECT
    COUNT(DISTINCT order_id) AS unique_customer_orders
FROM
    customer_orders_cleaned;
```

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

unique_customer_orders
9

```sql
-- How many successful orders were delivered by each runner?

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

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

runner_id	successful_orders
1	4
2	3
3	1

```sql
-- How many of each type of pizza was delivered?

SELECT
    pizza_id, COUNT(order_id) no_of_pizzas
FROM
    customer_orders_cleaned
GROUP BY pizza_id;
```

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

pizza_id	no_of_pizzas
1	12
2	3

```sql
-- How many Vegetarian and Meatlovers were ordered by each customer

SELECT
    customer_id,
    SUM(CASE
        WHEN pizza_id = 1 THEN 1
        ELSE 0
        END) AS Meatlovers_pizza,
    SUM(CASE
        WHEN pizza_id = 2 THEN 1
        ELSE 0
        END) AS Vegetarian_pizza
FROM
    customer_orders_cleaned
GROUP BY customer_id;
```

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

customer_id	Meatlovers_pizza	Vegetarian_pizza
101	2	1
102	2	1
104	6	0
105	0	1
103	2	0

```sql
-- What was the maximum number of pizzas delivered in a single order?

SELECT
    customer_id, order_id, COUNT(order_id) AS Pizza_count
FROM
    customer_orders_cleaned
GROUP BY customer_id , order_id
ORDER BY Pizza_count DESC
LIMIT 1;
```

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

customer_id	order_id	Pizza_count
104	10	5

```sql
-- 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 not null
                        or extras is not null)
               then 1
               else 0
           end) as Change_in_pizza,
        sum(case
                when (exclusions is null
                      and extras is null)
                then 1
                else 0
            end) as No_change_in_pizza
from customer_orders_cleaned co inner join runner_orders_cleaned ro on co.order_id = ro.order_id
where cancellation is null
group by customer_id
order by customer_id;
```

customer_id	Change_in_pizza	No_change_in_pizza
101	0	2
102	0	3
104	5	1
105	1	0

```sql
-- How many pizzas were delivered that had both exclusions and extras?

select * from customer_orders_cleaned;
select * from runner_orders_cleaned;

SELECT
    pizza_id, COUNT(pizza_id) AS Delivered_pizzas
FROM
    customer_orders_cleaned co
        JOIN
    runner_orders_cleaned ro ON co.order_id = ro.order_id
WHERE
    exclusions IS NOT NULL
        AND extras IS NOT NULL
        AND cancellation IS NULL
GROUP BY pizza_id;
```

pizza_id	Delivered_pizzas
1	4

```sql
-- What was the total volume of pizzas ordered for each hour of the day?

SELECT
    HOUR(order_time) AS Hour_of_day,
    COUNT(order_id) AS volume_of_pizza
FROM
    customer_orders_cleaned
GROUP BY Hour_of_day;
```

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

Hour_of_day	volume_of_pizza
18	6
19	1
23	3
21	3
11	2

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

SELECT
    DAYNAME(order_time) AS Day_of_week,
    COUNT(order_id) AS volume_of_pizza
FROM
    customer_orders_cleaned
GROUP BY Day_of_week;
```

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

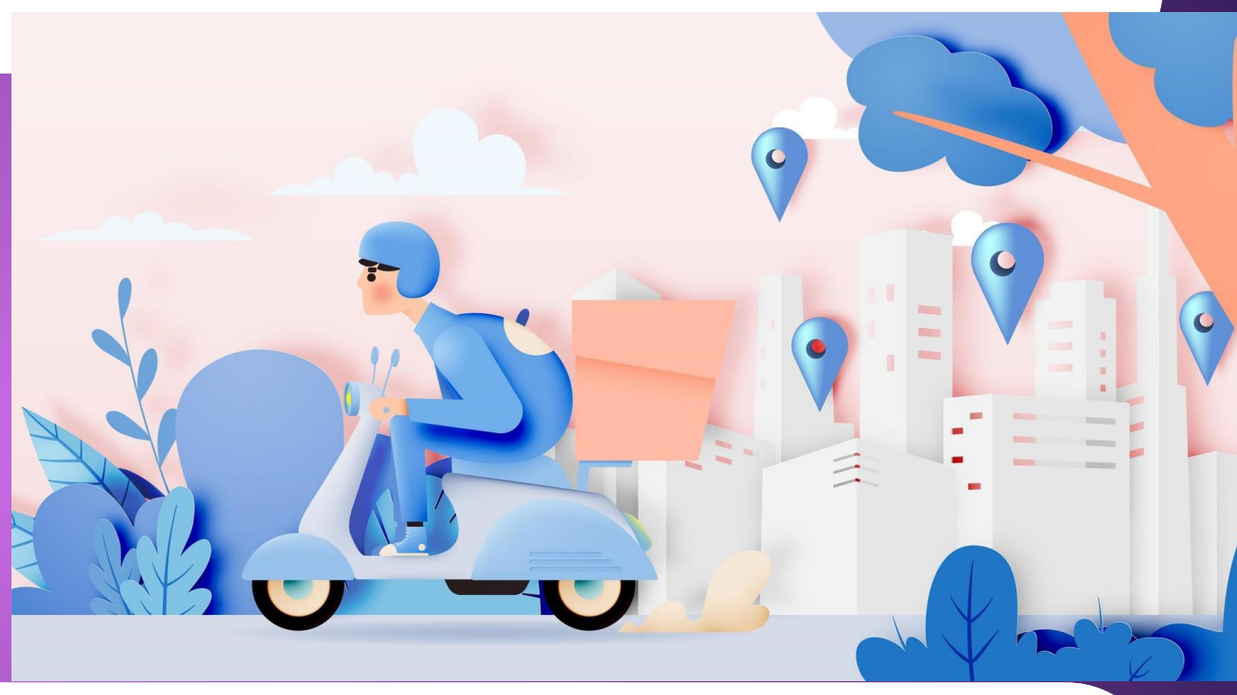
Day_of_week	volume_of_pizza
Wednesday	5
Thursday	3
Friday	2
Saturday	5

B. RUNNER & CUSTOMER EXPERIENCE

```sql
#  Runner and Customer Experience
-- How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)

SELECT
    WEEK(registration_date) AS week_of_registration,
    COUNT(runner_id) AS number_of_runner
FROM
    runners
GROUP BY week_of_registration;
```

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

week_of_registration	number_of_runner
0	1
1	2
2	1



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

SELECT
    ro.runner_id,
    ROUND(AVG(TIMESTAMPDIFF(MINUTE, co.order_time, ro.pickup_time)),2) AS Avg_arrive_time_to_HQ
FROM
    customer_orders_cleaned co
        JOIN
    runner_orders_cleaned ro ON co.order_id = ro.order_id
GROUP BY ro.runner_id;
```

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

runner_id	Avg_arrive_time_to_HQ
1	15.22
3	10.00
2	15.00

```sql
-- Is there any relationship between the number of pizzas and how long the order takes to prepare?

SET sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY',''));

WITH cte AS (
        SELECT count(co.pizza_id) AS pizza_count,
               co.order_id,
               round(AVG(TIMESTAMPDIFF(MINUTE, co.order_time, ro.pickup_time)),2) AS order_prep_time
        FROM customer_orders_cleaned co
                JOIN
             runner_orders_cleaned ro
                ON co.order_id = ro.order_id
        WHERE ro.pickup_time IS NOT NULL
        GROUP BY co.order_id)
SELECT pizza_count,order_prep_time
FROM cte
GROUP BY pizza_count;
```

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

pizza_count	order_prep_time
1	10.00
2	21.00
5	15.00

```sql
#PEARSON CORRELATION FOR FINDING REALTION BETWEEN 'PIZZA_COUNT' AND 'PREPARATION TIME'

create temporary table pearson
select count(co.pizza_id) as pizza_count,
co.order_id, avg(timestampdiff(minute,co.order_time,ro.pickup_time)) as order_prep_time
from customer_orders_cleaned co join runner_orders_cleaned ro on co.order_id = ro.order_id
where ro.pickup_time is not null
group by co.order_id;
```

```sql
#create average and standard rows to calculate pearson r value

SELECT
    @ax:=AVG(order_prep_time),
    @ay:=AVG(pizza_count),
    @div:=(STDDEV_SAMP(order_prep_time) * STDDEV_SAMP(pizza_count))
FROM
    pearson;

# calculate pearson r value

SELECT
    SUM((order_prep_time - @ax) * (pizza_count - @ay)) / ((COUNT(order_prep_time) - 1) * @div) AS pearson_r
FROM
    pearson;

-- pearson r value is '0.277'. This proves that there is a relation between pizza count and preparation time
```

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

pearson_r
0.27745303737757626

```sql
-- What was the average distance travelled for each customer?

SELECT
    co.customer_id, round(AVG(ro.distance),2) AS avg_distance
FROM
    customer_orders_cleaned co
        JOIN
    runner_orders_cleaned ro ON co.order_id = ro.order_id
GROUP BY co.customer id;
```

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

customer_id	avg_distance
101	20
102	16.73
104	10
105	25
103	NULL

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

SELECT
    MAX(duration) - MIN(duration) AS difference
FROM
    runner_orders_cleaned
WHERE
    duration IS NOT NULL;
```

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

difference
30

```sql
-- What was the average speed for each runner for each delivery and do you notice any trend for these values
SELECT
    co.customer_id,
    ro.runner_id,
    ro.order_id,
    ROUND(AVG((ro.distance*1000) / (ro.duration *60)),2) AS avg_speed_M_per_S
FROM
    runner_orders_cleaned ro
        JOIN
    customer_orders_cleaned co
WHERE
    distance IS NOT NULL
GROUP BY customer_id,ro.runner_id , ro.order_id
order by customer_id asc,ro.runner_id asc, ro.order_id asc;

-- There is no trend for these values
```

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

customer_id	runner_id	order_id	avg_speed_M_per_S
101	1	1	10.42
101	1	2	12.35
101	1	3	11.17
101	1	10	16.67
101	2	4	9.75
101	2	7	16.67

```sql
-- What is the successful delivery percentage for each runner?

SELECT
    runner_id,
    (SUM(CASE
        WHEN cancellation IS NULL THEN 1
        ELSE 0
    END) * 100 / COUNT(runner_id)) AS Percentage
FROM
    runner_orders_cleaned
GROUP BY runner_id;
```

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

runner_id	Percentage
1	100.0000
2	75.0000
3	50.0000

C. INGREDIENT OPTIMISATION

```sql
#C. Ingredient Optimisation

-- What are the standard ingredients for each pizza?

SELECT pr.pizza_id, pn.pizza_name,
       GROUP_CONCAT(pt.topping_name, ', ') AS ingredients
FROM pizza_recipes_cleaned pr
JOIN pizza_names pn ON pr.pizza_id = pn.pizza_id
JOIN pizza_toppings pt on pr.toppings = pt.topping_id
GROUP BY pr.pizza_id,pn.pizza_name;
```

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

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

```sql
-- What was the most commonly added extra?

SELECT pt.topping_name,
       COUNT(co.extras) AS times_added
FROM customer_orders_cleaned co
LEFT JOIN pizza_toppings pt ON co.extras = pt.topping_id
WHERE topping_name IS NOT NULL
GROUP BY topping_name
LIMIT 1;
```

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

topping_name	times_added
Bacon	5

```sql
-- What was the most common exclusion?

SELECT pt.topping_name,
       COUNT(exclusions) AS exclusions_count
FROM customer_orders_cleaned co
JOIN pizza_toppings pt ON co.exclusions = pt.topping_id
GROUP BY pt.topping_name;
```

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

topping_name	times_added
Bacon	5

```sql
/*
-- 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
*/

with order_details as (
        SELECT
            co.order_id,
            co.customer_id,
            pn.pizza_id,
            pn.pizza_name,
            co.exclusions,
            co.extras
        FROM customer_orders_cleaned co
        JOIN pizza_names pn
        ON co.pizza_id = pn.pizza_id
        ),
```

```sql
toppings as (
        SELECT topping_id,
               topping_name
        FROM pizza_toppings
        ),
formatted_exclusions as (
        SELECT od.order_id,
               GROUP_CONCAT(DISTINCT t.topping_name ORDER BY t.topping_name SEPARATOR ', ') AS exclusions
        FROM order_details od
        LEFT JOIN toppings t ON FIND_IN_SET(t.topping_id, od.exclusions)
        GROUP BY od.order_id
        ),
formatted_extras as (
        SELECT od.order_id,
               GROUP_CONCAT(DISTINCT t.topping_name ORDER BY t.topping_name SEPARATOR ', ') AS extras
        FROM order_details od
        LEFT JOIN toppings t ON FIND_IN_SET(t.topping_id, od.extras)
        GROUP BY od.order_id
        )
```

```sql
SELECT od.order_id,od.customer_id,od.pizza_id,
       CONCAT_WS(' ', od.pizza_name,
                                    CASE
                                        WHEN fe.exclusions is not null and fe.exclusions != ''
                                        THEN CONCAT('- Exclude ', fe.exclusions)
                                        ELSE ''
                                    END,
                                    CASE
                                        WHEN fx.extras is not null and fx.extras != ''
                                        THEN CONCAT('- Extra ', fx.extras)
                                        ELSE ''
                                    END) AS order_description
FROM order_details od
left JOIN formatted_exclusions fe ON od.order_id = fe.order_id
left JOIN formatted_extras fx ON od.order_id = fx.order_id;
```

order_id	customer_id	pizza_id	order_description
1	101	1	Meatlovers
2	101	1	Meatlovers
3	102	1	Meatlovers
3	102	2	Vegetarian
5	104	1	Meatlovers - Extra Bacon
6	101	2	Vegetarian
7	105	2	Vegetarian - Extra Bacon
8	102	1	Meatlovers
9	103	1	Meatlovers - Exclude Cheese - Extra Bacon, Chicken
9	103	1	Meatlovers - Exclude Cheese - Extra Bacon, Chicken
10	104	1	Meatlovers - Exclude BBQ Sauce, Mushrooms - Extra Bacon, Cheese
10	104	1	Meatlovers - Exclude BBQ Sauce, Mushrooms - Extra Bacon, Cheese
10	104	1	Meatlovers - Exclude BBQ Sauce, Mushrooms - Extra Bacon, Cheese
10	104	1	Meatlovers - Exclude BBQ Sauce, Mushrooms - Extra Bacon, Cheese
10	104	1	Meatlovers - Exclude BBQ Sauce, Mushrooms - Extra Bacon, Cheese

```sql
-- What is the total quantity of each ingredient used in all delivered pizzas sorted by most frequent first?

WITH frequent_ingredients AS (
                                SELECT co.order_id,
                                       pt.topping_name as topping_name,
                                       CASE
                                            WHEN pt.topping_id IN ( SELECT extras
                                                                    FROM customer_orders_cleaned) THEN 2
                                            WHEN pt.topping_id IN ( SELECT exclusions
                                                                    FROM customer_orders_cleaned ) THEN 0
                                            ELSE 1
                                        END AS times_used
                                FROM customer_orders_cleaned co
                                LEFT JOIN pizza_recipes_cleaned pr USING(pizza_id)
                                JOIN pizza_toppings pt on pt.topping_id = pr.toppings)
SELECT topping_name,
       SUM(times_used) as times_used
FROM frequent_ingredients
GROUP BY topping_name
ORDER BY times_used DESC;
```

topping_name	times_used
Cheese	30
Chicken	24
Bacon	24
Salami	12
Pepperoni	12
Beef	12
Tomato Sauce	3
Tomatoes	3
Peppers	3
Onions	3
Mushrooms	0
BBQ Sauce	0

D. PRICING & RATINGS

```sql
# 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?
SELECT SUM(CASE
                WHEN co.pizza_id = 1 THEN 12
                ELSE 10
            END) AS Earned_money
FROM customer_orders_cleaned co LEFT JOIN
     runner_orders_cleaned ro USING (order_id)
WHERE ro.cancellation IS NULL;
```

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

Earned_money
140

```sql
# 2. What if there was an additional $1 charge for any pizza extras
        -- Add cheese is $1 extra

  SELECT SUM(CASE
                WHEN co.pizza_id = 1 THEN 12
                ELSE 10
            END) +
         SUM(CASE
                WHEN co.extras = 4 THEN 2
                WHEN co.extras IS NULL THEN 0
                ELSE 1
            END) AS updated_money
FROM customer_orders_cleaned co LEFT JOIN
     runner_orders_cleaned ro USING (order_id)
WHERE ro.cancellation IS NULL;
```

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

updated_money
148

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

CREATE TABLE ratings (
    order_id INT,
    rating INT
);

INSERT INTO ratings (order_id, rating)
VALUES (1,3),
       (2,5),
       (3,3),
       (4,1),
       (5,5),
       (7,3),
       (8,4),
       (9,2),
       (10,4);

SELECT * FROM ratings;
```

order_id	rating
1	3
2	5
3	3
4	1
5	5
7	3
8	4
9	2
10	4

```sql
# 4. Using your 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

SELECT co.customer_id,
       co.order_id,
       ro.runner_id,
       rt.rating,
       co.order_time,
       ro.pickup_time,
       TIMESTAMPDIFF(MINUTE,co.order_time, ro.pickup_time) AS Time_betn_Order_and_Pickup,
       ro.duration,
       ROUND(AVG(ro.distance/ro.duration * 60),1) AS Average_speed,
       COUNT(co.order_id) AS Pizza_count
FROM customer_orders_cleaned co
LEFT JOIN runner_orders_cleaned ro USING (order_id)
JOIN ratings rt USING (order_id)
GROUP BY co.customer_id,
         co.order_id,
         ro.runner_id,
         co.order_time,
         ro.pickup_time,
         ro.duration;
```

customer_id	order_id	runner_id	rating	order_time	pickup_time	Time_betn_Order_and_Pickup	duration	Average_speed	Pizza_count
101	1	1	3	2020-01-01 18:05:02	2020-01-01 18:15:34	10	32	37.5	1
101	2	1	5	2020-01-01 19:00:52	2020-01-01 19:10:54	10	27	44.4	1
102	3	1	3	2020-01-02 23:51:23	2020-01-03 00:12:37	21	20	40.2	2
104	5	3	5	2020-01-08 21:00:29	2020-01-08 21:10:57	10	15	40	1
105	7	2	3	2020-01-08 21:20:29	2020-01-08 21:30:45	10	25	60	1
102	8	2	4	2020-01-09 23:54:33	2020-01-10 00:15:02	20	15	93.6	1
103	9	2	2	2020-01-10 11:22:59	NULL	NULL	NULL	NULL	2
104	10	1	4	2020-01-11 18:34:49	2020-01-11 18:50:20	15	10	60	5

```sql
# 5. 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 kilometre traveled -
-- how much money does Pizza Runner have left over after these deliveries?

SELECT SUM(CASE
                WHEN co.pizza_id = 1 THEN 12
                ELSE 10
            END) AS Revenue,
        ROUND(SUM(ro.distance) * 0.3, 2) AS runner_paid,
        ROUND(SUM(CASE
                    WHEN co.pizza_id = 1 THEN 12
                    ELSE 10
                END) - (SUM(ro.distance) * 0.3),2) AS Money_left
FROM customer_orders_cleaned co JOIN
     runner_orders_cleaned ro USING (order_id)
WHERE ro.cancellation IS NULL;
```

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

Revenue	runner_paid	Money_left
140	52.56	87.44

THANK YOU !