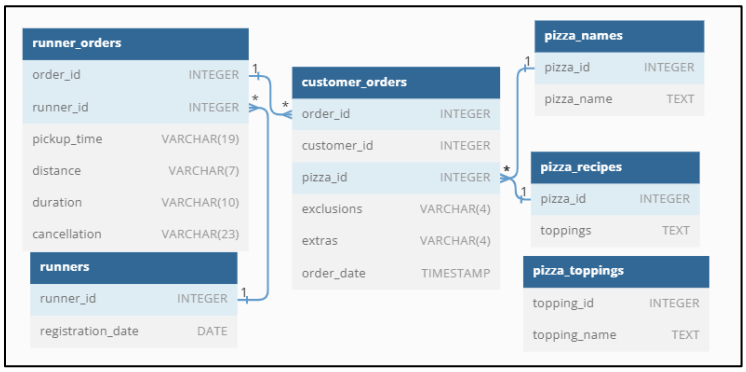
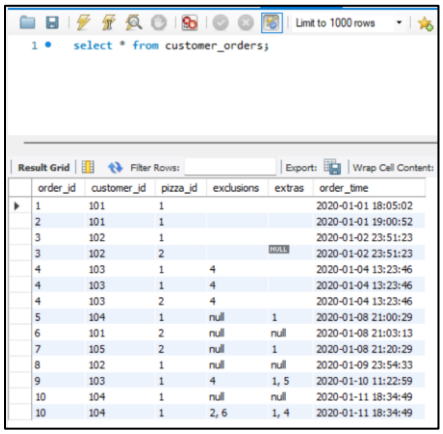
8 Week SQL Challenge
https://8weeksqlchallenge.com/case-study-2/
Case Study #2 – Pizza's Runner

Entity Relationship Diagram -



Tables -

1. customer_orders -



```
select * from 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

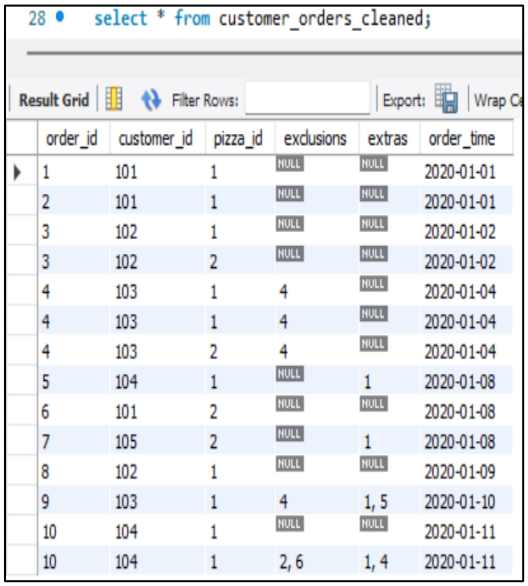
We can not proceed ahead with this table for analysis as it is. There are many **blank columns** and **null values** in 'execlusions' and 'extras' columns. The first task is to clean the table.

Data Cleaning:

- Blank values columns are cleaned
- 'null' values columns are cleaned

Following query has been used to clean the data and create new table as "customer_orders_cleaned"

```
create table customer_orders_cleaned
(order_id int
,customer_id Int
, pizza_id int
, exclusions varchar(256)
,extras varchar(256)
, order_time date
);
insert into customer_orders_cleaned
(select 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_cleaned;
```

order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1	NULL	NULL	2020-01-01
2	101	1	NULL	NULL	2020-01-01
3	102	1	NULL	NULL	2020-01-02
3	102	2	NULL	NULL	2020-01-02
4	103	1	4	NULL	2020-01-04
4	103	1	4	NULL	2020-01-04
4	103	2	4	NULL	2020-01-04
5	104	1	NULL	1	2020-01-08
6	101	2	NULL	NULL	2020-01-08
7	105	2	NULL	1	2020-01-08
8	102	1	NULL	NULL	2020-01-09
9	103	1	4	1, 5	2020-01-10
10	104	1	NULL	NULL	2020-01-11
10	104	1	2, 6	1, 4	2020-01-11

2. Pizza_names

```
select * from pizza_names;
```

pizza_id	pizza_name
1	Meatlovers
2	Vegetarian

3. Pizza_recipes

Pizza recipes – since values are stored as comma separated we will be splitting comma separated values into different rows and creating new table as row_split_pizza_receipes.

```
select * from pizza_recipes;
```

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

```
SELECT
    *
FROM
row_split_pizza_recipes;
```

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

4. Pizza_toppings

```
select * from pizza_toppings;
```

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

5. Runner_orders

```
select * from 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	25mins	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

We can-not proceed ahead with this table for analysis as it is. There are many **blank columns** and **null values** in different columns. The first task is to clean the table.

Data Cleaning:

- Blank values columns are cleaned
- 'null' values columns are cleaned

Following query has been used to clean the data and create new table as "runner_orders_cleaned"

```
create table runner_orders_cleaned
(order_id int
,runner_id Int
, pickup_time datetime
, distance_in_km float
,duration_in_minutes int
, cancellation varchar(256)
);
insert into runner_orders_cleaned
(select order_id,
runner_id,
case
      when pickup_time = 'null' then null
      else pickup_time
      end as pickup_time,
case
      when distance = 'null' then null
      else
      regexp_replace(distance, '[a-z]','')
      end as distance_in_Km,
case
      when duration = 'null' then null
      else
      regexp_replace(duration, '[a-z]','')
      end as duration_in_minutes,
case
      when cancellation = '' then null
      when cancellation = 'null' then null
      else
      cancellation
      end as cancellation
from
runner_orders);
```

```
select * from runner_orders_cleaned;
```

order_id	runner_id	pickup_time	distance_in_km	duration_in_minutes	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

6. Runners

```
select * from runners;
```

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

7. Another table named exc_ext is also created by splitting exclusions and extras comma separated values

```
SELECT
      *
FROM
exc_ext;
```

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	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

A. Pizza Metrics

Answer-1

```
-- How many pizzas were ordered?
select count(order_id) as "pizza's ordered"
from customer_orders_cleaned;
```

pizza's ordered
14

Answer -2

```
-- How many unique customer orders were made?
select count(distinct(order_id)) as "Unique customers' orders"
from customer_orders_cleaned;
```

Unique customers' orders
10

Answer -3

```
-- 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;
```

runner_id	Successful Orders
1	4
2	3
3	1

Answer - 4

```
-- How many of each type of pizza was delivered?
select cdc.pizza_id,count(cdc.pizza_id) as 'successful deliveries'
from customer_orders_cleaned cdc
join runner_orders_cleaned rdc
on cdc.order_id = rdc.order_id
where rdc.cancellation is null
group by cdc.pizza_id;
```

pizza_id	successful deliveries
1	9
2	3

Answer -5

```
-- How many Vegetarian and Meatlovers were ordered by each customer?
select cdc.customer_id,pn.pizza_name, count(cdc.order_id) as 'type of pizza ordered'
from customer_orders_cleaned cdc
join pizza_names pn on cdc.pizza_id = pn.pizza_id
group by cdc.customer_id,pn.pizza_name
order by cdc.customer_id;
```

customer_id	pizza_name	type of 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

Answer -6

```
-- What was the maximum number of pizzas delivered in a single order?
select customer_id, order_id,
count(order_id) as 'no. of orders placed'
from customer_orders_cleaned
group by customer_id, order_id
order by count(order_id) desc limit 1;
```

customer_id	order_id	no. of orders placed
103	4	3

Answer -7

```
-- For each customer, how many delivered pizzas had at least 1 change and how many had no changes?
with chnage_staus as (
Select cdc.customer_id,
case
    when rdc.cancellation is Null then 'delivered'
    when rdc.cancellation is not null then 'cancelled'
    end as 'status',
case
    when cdc.exclusions is Null and cdc.extras is null then 'no change'
    else 'changes made' end as 'changes_made'
from customer_orders_cleaned cdc
join runner_orders_cleaned rdc
on cdc.order_id = rdc.order_id
)
select cs.customer_id,cs.status,cs.changes_made,count(cs.changes_made) as 'count of pizzas'
from chnage_staus cs
where cs.status = 'delivered'
group by cs.status,cs.changes_made,cs.customer_id
order by cs.customer_id
```

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

customer_id	status	changes_made	count of pizzas
101	delivered	no change	2
102	delivered	no change	3
103	delivered	changes made	3
104	delivered	changes made	2
104	delivered	no change	1
105	delivered	changes made	1

Answer -8

```
-- How many pizzas were delivered that had both exclusions and extras?
with chnage_staus as (
Select cdc.customer_id,
case
    when rdc.cancellation is Null then 'delivered'
    when rdc.cancellation is not null then 'cancelled'
    end as 'status',
case
    when cdc.exclusions is Null and cdc.extras is null then 'no change'
    when cdc.exclusions is Not Null and cdc.extras is not null then 'both change'
    else ' single changes' end as 'changes_made'
from customer_orders_cleaned cdc
join runner_orders_cleaned rdc
on cdc.order_id = rdc.order_id
)
select count(cs.changes_made) as 'count of delivered pizzas with both changes'
from chnage_staus cs
where cs.changes_made = 'both change' and cs.status = 'delivered';
```

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

count of delivered pizzas with both changes
1

Answer -9

Limit to 1000 rows

```
-- What was the total volume of pizzas ordered for each hour of the day?
SELECT HOUR(order_time) AS 'Hour',
       count(order_id) AS "Volume of pizzas' ordered"
FROM customer_orders_cleaned
GROUP BY 1
ORDER BY 1;
```

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

Hour	Volume of pizzas' ordered
11	1
13	3
18	3
19	1
21	3
23	3

Answer -10

```
-- What was the volume of orders for each day of the week?
SELECT dayname(order_time) AS 'weekday',
       count(order_id) AS "Volume of pizzas' ordered"
FROM customer_orders_cleaned
GROUP BY dayname(order_time)
ORDER BY count(order_id);
```

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

weekday	Volume of pizzas' ordered
Friday	1
Thursday	3
Wednesday	5
Saturday	5

B. Runner and Customer Experience

Answer -1

```
-- How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)
SELECT
    WEEK(registration_date) AS week_period,
    COUNT(runner_id) AS runners_signed_up_no
FROM
    runners
GROUP BY WEEK(registration_date);
```

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

week_period	runners_signed_up_no
0	1
1	2
2	1

Explanation:

Returned week number is between 0 and 52 or 0 and 53.
Default mode of the week =0 -> First day of the week is Sunday
Extract week -> WEEK(registration_date) or EXTRACT(week from registration_date)

Answer – 2

```
-- What was the average time in minutes it took for each
-- runner to arrive at the Pizza Runner HQ to pickup the order?
SELECT  roc.runner_id,
        AVG(TIMESTAMPDIFF(MINUTE,cdc.order_time,roc.pickup_time)) as average_time_in_minutes
FROM
    customer_orders_cleaned cdc
JOIN
    runner_orders_cleaned roc ON cdc.order_id = roc.order_id
```

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

runner_id	average_time_in_minutes
1	15.3333
2	23.4000
3	10.0000

Answer – 3

```sql
-- Is there any relationship between the number of
-- pizzas and how long the order takes to prepare?
with prep_time  as (
SELECT
    coc.order_id,
    COUNT(coc.order_id) AS 'no_of_pizzas_ordered',
    TIMESTAMPDIFF(MINUTE,coc.order_time,roc.pickup_time) AS 'prep_time'
FROM customer_orders_cleaned coc
JOIN runner_orders_cleaned roc ON coc.order_id = roc.order_id
WHERE cancellation IS NULL
GROUP BY order_id , TIMESTAMPDIFF(MINUTE,coc.order_time,roc.pickup_time)
)
SELECT no_of_pizzas_ordered, AVG(prep_time)
FROM prep_time
GROUP BY no_of_pizzas_ordered;
```

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

no_of_pizzas_ordered	avg(prep_time)
1	12.0000
2	18.0000
3	29.0000

- On average, a single pizza order takes 12 minutes to prepare.
- An order with 3 pizzas takes 29 minutes at an average of 9.99 minutes per pizza.
- It takes 18 minutes to prepare an order with 2 pizzas which is 9 minutes per pizza — making 2 pizzas in a single order the ultimate efficiency rate.

Answer – 4

```sql
-- What was the average distance travelled for each customer?
SELECT
    coc.customer_id,
    ROUND(AVG(roc.distance_in_km), 2) AS 'average distance travelled'
FROM
    customer_orders_cleaned coc
        JOIN
    runner_orders_cleaned roc ON coc.order_id = roc.order_id
WHERE
    roc.cancellation IS NULL
GROUP BY coc.customer_id;
```

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

customer_id	average distance travelled
101	20
102	16.73
103	23.4
104	10
105	25

Answer – 5

```sql
-- What was the difference between the longest
-- and shortest delivery times for all orders?
SELECT
    MAX(duration_in_minutes) AS 'longest_time',
    MIN(duration_in_minutes) 'shortest_time',
    (MAX(duration_in_minutes) - MIN(duration_in_minutes)) AS 'difference'
FROM
    runner_orders_cleaned;
```

Result Grid

longest_time	shortest_time	difference
40	10	30

Answer – 6

```sql
-- What was the average speed for each runner for
-- each delivery and do you notice any trend for these values?
SELECT
    runner_id,
    order_id,
    ROUND(AVG(distance_in_km / (duration_in_minutes / 60)),
            2) AS 'avg_speed'
FROM
    runner_orders_cleaned
WHERE
    cancellation IS NULL
GROUP BY runner_id , order_id
ORDER BY runner_id;
```

Result Grid

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

Answer – 7

```sql
-- What is the successful delivery percentage for each runner?
with cte_1 as  (
SELECT runner_id,COUNT(order_id) AS 'orders',
    CASE
        WHEN cancellation IS NULL THEN COUNT(order_id) ELSE 0 END AS 'delivered',
    CASE
        WHEN cancellation IS NOT NULL THEN COUNT(order_id) ELSE 0 END AS 'notDelivered'
FROM runner_orders_cleaned
GROUP BY runner_id , cancellation
ORDER BY runner_id
)
SELECT
    runner_id,
    ROUND(SUM(delivered) / SUM(orders) * 100, 2) AS 'delivery_percentage'
FROM cte_1
GROUP BY runner_id;
```

Result Grid

runner_id	delivery_percentage
1	100.00
2	75.00
3	50.00

C. Ingredient Optimization

Answer – 1

```sql
-- What are the standard ingredients for each pizza?
SELECT  row_split_pizza_recipes.pizza_id,
        pizza_names.pizza_name,
        group_concat(pizza_toppings.topping_name) AS 'standard_ingredients'
FROM    row_split_pizza_recipes
JOIN    pizza_toppings ON row_split_pizza_recipes.topping_id = pizza_toppings.topping_id
JOIN    pizza_names ON pizza_names.pizza_id = row_split_pizza_recipes.pizza_id
GROUP BY row_split_pizza_recipes.pizza_id,
        pizza_names.pizza_name
ORDER BY row_split_pizza_recipes.pizza_id;
```

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

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

Answer – 2

```sql
-- What was the most commonly added extra?
with split_extras as (
SELECT j.extras
FROM customer_orders_cleaned coc
-- creating json table to split comma seperated values in extras columns into diff rows
JOIN JSON_TABLE(TRIM(REPLACE(JSON_ARRAY(coc.extras),',','",""')),
                '$[*]' COLUMNS(extras INT PATH '$')) j
WHERE j.extras IS NOT NULL
)
SELECT  c.extras,p.topping_name,COUNT(c.extras) AS 'most commonly added extra'
FROM split_extras c
JOIN pizza_toppings p ON c.extras = p.topping_id
GROUP BY c.extras , p.topping_name
ORDER BY COUNT(c.extras) DESC LIMIT 1;
```

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

extras	topping_name	most commonly added extra
1	Bacon	4

Answer – 3

```sql
-- What was the most common exclusion?
with split_exclusions as (SELECT j.exclusions_id
FROM customer_orders_cleaned coc
-- creating json table to split comma seperated values in extras columns into diff rows
JOIN JSON_TABLE(TRIM(REPLACE(JSON_ARRAY(coc.exclusions),',','",""')),
                '$[*]' COLUMNS(exclusions_id INT PATH '$')) j
WHERE j.exclusions_id IS NOT NULL
),
top_exclusions as ( SELECT  c.exclusions_id,p.topping_name,COUNT(c.exclusions_id) AS 'most common exclusion'
FROM split_exclusions c
JOIN pizza_toppings p ON c.exclusions_id = p.topping_id
GROUP BY c.exclusions_id , p.topping_name
ORDER BY COUNT(c.exclusions_id) DESC LIMIT 1
)
SELECT topping_name AS 'Most common Exclusion'
FROM top_exclusions;
```

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

Most common Exclusion
Cheese

Answer – 4

```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
cte_1 AS (
SELECT c.order_id, c.customer_id,c.pizza_id,pn.pizza_name,j1.exclusions, j2.extras
FROM customer_orders_cleaned c
JOIN pizza_names pn on c.pizza_id = pn.pizza_id
JOIN JSON_TABLE(trim(replace(json_array(c.exclusions),',','","')),'$[*]' columns (exclusions int path '$')) j1
JOIN JSON_TABLE(trim(replace(json_array(c.extras),',','","')),'$[*]' columns (extras int path '$')) j2
),
cte_2 as (
SELECT c.order_id, c.customer_id,c.pizza_id,c.pizza_name, pt.topping_name as 'exclusion',p.topping_name as 'extras'
FROM cte_1 c
LEFT JOIN pizza_toppings pt ON c.exclusions = pt.topping_id
LEFT JOIN pizza_toppings p ON c.extras = p.topping_id)
SELECT c.order_id, c.customer_id,
       CASE WHEN exclusion IS NULL AND extras IS NULL THEN pizza_name
            WHEN extras IS NULL AND exclusion IS NOT NULL THEN concat(pizza_name, ' - Exclude ', GROUP_CONCAT(DISTINCT exclusion))
            WHEN exclusion IS NULL AND extras IS NOT NULL THEN concat(pizza_name, ' - Include ', GROUP_CONCAT(DISTINCT extras))
            ELSE concat(pizza_name, ' - Include ', GROUP_CONCAT(DISTINCT extras), ' - Exclude ', GROUP_CONCAT(DISTINCT exclusion))
       END AS pizza_type
from cte_2 c
GROUP BY c.order_id, c.customer_id,c.pizza_name,c.exclusion,c.extras;
```

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

order_id	customer_id	pizza_type
1	101	Meatlovers
2	101	Meatlovers
3	102	Meatlovers
3	102	Vegetarian
4	103	Meatlovers - Exclude Cheese
4	103	Vegetarian - Exclude Cheese
5	104	Meatlovers - Include Bacon
6	101	Vegetarian
7	105	Vegetarian - Include Bacon
8	102	Meatlovers
9	103	Meatlovers - Include Bacon - Exclude Cheese
9	103	Meatlovers - Include Chicken - Exclude Cheese
10	104	Meatlovers
10	104	Meatlovers - Include Bacon - Exclude BBQ Sauce
10	104	Meatlovers - Include Cheese - Exclude BBQ Sauce
10	104	Meatlovers - Include Bacon - Exclude Mushrooms
10	104	Meatlovers - Include Cheese - Exclude Mushrooms

Answer – 5

```sql
-- Generate an alphabetically ordered comma separated ingredient list for each
-- pizza order from the customer_orders table
with
pizza_ingredients as (
select pr.pizza_id, pt.topping_name
from row_split_pizza_recipes pr
join pizza_toppings pt on pr.topping_id = pt.topping_id
order by pt.topping_name
),
cte_2 as (
select pizza_id, group_concat(topping_name) as 'ingredients'
from pizza_ingredients
group by pizza_id
)
select c.order_id, c.pizza_id,c2.ingredients
from customer_orders_cleaned c
join cte_2 c2 on c.pizza_id = c2.pizza_id
group by c.order_id, c.pizza_id,c2.ingredients;
```

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

order_id	pizza_id	ingredients
1	1	Bacon,BBQ Sauce,Beef,Cheese,Chicken,Mushrooms,Pepperoni,Salami
2	1	Bacon,BBQ Sauce,Beef,Cheese,Chicken,Mushrooms,Pepperoni,Salami
3	1	Bacon,BBQ Sauce,Beef,Cheese,Chicken,Mushrooms,Pepperoni,Salami
3	2	Cheese,Mushrooms,Onions,Peppers,Tomatoes,Tomato Sauce
4	1	Bacon,BBQ Sauce,Beef,Cheese,Chicken,Mushrooms,Pepperoni,Salami
4	2	Cheese,Mushrooms,Onions,Peppers,Tomatoes,Tomato Sauce
5	1	Bacon,BBQ Sauce,Beef,Cheese,Chicken,Mushrooms,Pepperoni,Salami
6	2	Cheese,Mushrooms,Onions,Peppers,Tomatoes,Tomato Sauce
7	2	Cheese,Mushrooms,Onions,Peppers,Tomatoes,Tomato Sauce
8	1	Bacon,BBQ Sauce,Beef,Cheese,Chicken,Mushrooms,Pepperoni,Salami
9	1	Bacon,BBQ Sauce,Beef,Cheese,Chicken,Mushrooms,Pepperoni,Salami
10	1	Bacon,BBQ Sauce,Beef,Cheese,Chicken,Mushrooms,Pepperoni,Salami

Answer – 6

```sql
-- What is the total quantity of each ingredient used in all delivered pizzas sorted by most frequent first?
with exc as (select ifnull(exclusions,0) as excluded_toppings, count(ifnull(exclusions,0)) as times_excluded
from exc_ext
join runner_orders_cleaned on exc_ext.order_id = runner_orders_cleaned.order_id
where cancellation is null group by 1),
ext as (select ifnull(extras,0) as extra_toppings, count(ifnull(extras,0)) as times_added
from exc_ext
join runner_orders_cleaned on exc_ext.order_id = runner_orders_cleaned.order_id
where cancellation is null
group by 1),
cte_1 as (select e.order_id, r.toppings
from exc_ext e
left join runner_orders_cleaned rc on rc.order_id = e.order_id
left join row_split_pizza_recipes r on e.pizza_id = r.pizza_id
where rc.cancellation is null),
ttl as (SELECT toppings, count(toppings) as total_toppings
from cte_1
GROUP BY 1)
select  c1.toppings, pt.topping_name, c1.total_toppings,ifnull(e2.times_added,0) as extra_time_added,ifnull(e1.times_excluded,0) as no_of_times_excluded,
        (ifnull(c1.total_toppings,0)+ifnull(e2.times_added,0))-ifnull(e1.times_excluded,0) as ttl_qty_used
from ttl c1
left join exc e1 on c1.toppings = e1.excluded_toppings
left join ext e2 on c1.toppings = e2.extra_toppings
left join pizza_toppings pt on pt.topping_id = c1.toppings ORDER BY c1.toppings;
```

toppings	topping_name	total_toppings	extra_time_added	no_of_times_excluded	ttl_qty_used
1	Bacon	12	4	0	16
2	BBQ Sauce	12	0	2	10
3	Beef	12	0	0	12
4	Cheese	15	2	3	14
5	Chicken	12	0	0	12
6	Mushrooms	15	0	2	13
7	Onions	3	0	0	3
8	Pepperoni	12	0	0	12
9	Peppers	3	0	0	3
10	Salami	12	0	0	12
11	Tomatoes	3	0	0	3
12	Tomato Sauce	3	0	0	3

D. Pricing and Ratings

Answer -1

```
-- If a Meat Lovers pizza costs $12 and Vegetarian costs $10 and there were no charges for changes
-- how much money has Pizza Runner made so far if there are no delivery fees?
with pizza_earning_details as (
select c.order_id, pizza_name,
case    when pizza_name = 'Meatlovers' THEN 12
        else 10 end as pizza_cost_in_$
from customer_orders_cleaned c
join runner_orders_cleaned r on c.order_id = r.order_id
join pizza_names p on c.pizza_id = p.pizza_id
where cancellation is null
ORDER BY c.order_id)
select  pizza_name, count(pizza_name) as 'no_of_orders', pizza_cost_in_$,
        concat('$',count(pizza_name)*pizza_cost_in_$) as 'earnings'
from pizza_earning_details
GROUP BY pizza_name;
```

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

pizza_name	no_of_orders	pizza_cost_in_$	earnings
Meatlovers	9	12	$108
Vegetarian	3	10	$30

Answer – 2

```
-- What if there was an additional $1 charge for any pizza extras?
with extras as (
select order_id, customer_id ,pizza_id ,exclusions, j.extras,order_time
from customer_orders_cleaned c
join JSON_TABLE(trim(replace(JSON_ARRAY(c.extras),',','","')),'$[*]' columns(extras int path '$')) j),
extras_count as (
select pizza_name,count(extras) as extras_added
from extras
join runner_orders_cleaned on extras.order_id = runner_orders_cleaned.order_id
join pizza_names on pizza_names.pizza_id = extras.pizza_id
where cancellation is null
group by pizza_name),
pizza_price_details as (
select c.order_id,pizza_name,
case when pizza_name = 'Meatlovers' then 12 else 10 end as Pizza_price
from customer_orders_cleaned c
join runner_orders_cleaned r on c.order_id = r.order_id
join pizza_names p on c.pizza_id = p.pizza_id
where cancellation is null)
select pd.pizza_name, count(pd.pizza_name) as 'no_of_orders', Pizza_price as 'price_in_$',
    count(pd.pizza_name)*Pizza_price as 'Earnings_bfr_ext', (count(pd.pizza_name)*Pizza_price)+extras_added as 'Total_earnings'
from pizza_price_details pd
join extras_count ec on pd.pizza_name = ec.pizza_name
group by pd.pizza_name
```

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

pizza_name	no_of_orders	price_in_$	Earnings_bfr_ext	Total_earnings
Meatlovers	9	12	108	111
Vegetarian	3	10	30	31

Answer – 3

```sql
-- 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 runners_ratings (
Order_id int,
runner_id int,
rating int check (rating BETWEEN 1 and 5));

insert into runners_ratings values
(1,1,5),(2,1,4),(3,1,3),(4,2,2),(5,3,1),
(7,2,2),(8,2,3),(10,1,5);

select * from runners_ratings;
```

Result Grid

Order_id	runner_id	rating
1	1	5
2	1	4
3	1	3
4	2	2
5	3	1
7	2	2
8	2	3
10	1	5

Answer – 4

```sql
-- 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  r.order_id, customer_id, r.runner_id, rating,order_time,
concat(timestampdiff(minute, order_time,pickup_time), ' min') as 'Time between order and pickup',
concat(duration_in_minutes,' min') as 'Delivery duration',
concat(round(avg(distance_in_km/(duration_in_minutes/60)),2),' km/hr') as 'Average speed',
count(pizza_id) as 'Total number of pizzas'
From runner_orders_cleaned r
left join customer_orders_cleaned c on r.order_id = c.order_id
left join runners_ratings rr on r.order_id = rr.order_id
where cancellation is null
GROUP BY r.order_id,customer_id;
```

Result Grid

order_id	customer_id	runner_id	rating	order_time	Time between order and pickup	Delivery duration	Average speed	Total number of pizzas
1	101	1	5	2020-01-01 18:05:02	10 min	32 min	37.5 km/hr	1
2	101	1	4	2020-01-01 19:00:52	10 min	27 min	44.44 km/hr	1
3	102	1	3	2020-01-02 23:51:23	21 min	20 min	40.2 km/hr	2
4	103	2	2	2020-01-04 13:23:46	29 min	40 min	35.1 km/hr	3
5	104	3	1	2020-01-08 21:00:29	10 min	15 min	40 km/hr	1
7	105	2	2	2020-01-08 21:20:29	10 min	25 min	60 km/hr	1
8	102	2	3	2020-01-09 23:54:33	20 min	15 min	93.6 km/hr	1
10	104	1	5	2020-01-11 18:34:49	15 min	10 min	60 km/hr	2

Answer – 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?

with total_earnings as (
select sum(case when pizza_id = 1 then 12
            else 10 end) as 'total_earning'
from customer_orders_cleaned c
join runner_orders_cleaned r on c.order_id = r.order_id
where cancellation is null),
runners_earnings as (
select sum(distance_in_km)*0.3 as 'runner_earnings'
from runner_orders_cleaned)
select  concat('$ ',round(total_earning - runner_earnings,2))
        as 'Pizza_Runner_money'
from total_earnings,runners_earnings;
```

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

Pizza_Runner_money
$ 94.44