

DATAWITHDANNY.COM
START YOUR SQL ENGINES



PIZZA RUNNER

CASE STUDY #2

8 WEEK SQL
CHALLENGE

8WEEKSQLCHALLENGE.COM

<https://8weeksqlchallenge.com/case-study-2/>

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

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

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

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

Entity Relationship Diagram

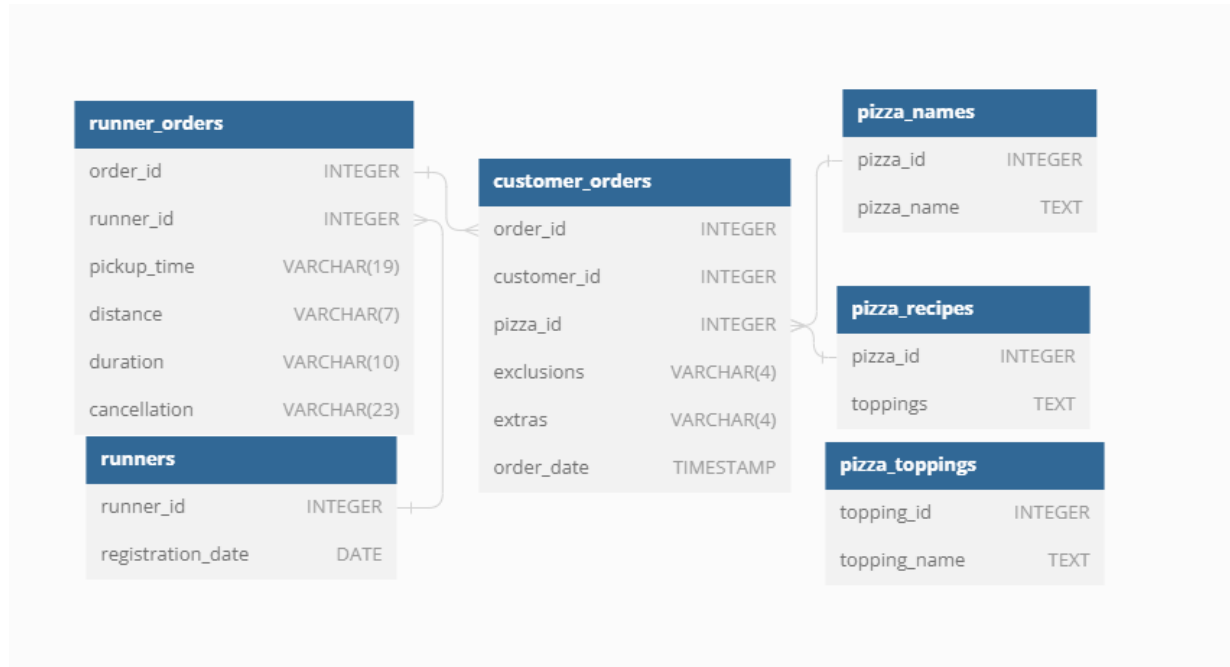


Table 1: runners

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

Table 2: 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 which should be removed from the pizza and the extras are the ingredient_id values which need to be added to the pizza.

Note that customers can order multiple pizzas in a single order with varying exclusions and extras values even if the pizza is the same type!

The exclusions and extras columns will need to be cleaned up before using them in your queries.

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
10	104	1	null	null	2021-01-11 18:34:49
10	104	1	2, 6	1, 4	2021-01-11 18:34:49

Table 3: runner_orders

After each orders are received through the system - they are assigned to a runner - however not all orders are fully completed and can be cancelled by the restaurant or the customer.

The pickup_time is the timestamp at which the runner arrives at the Pizza Runner headquarters to pick up the freshly cooked pizzas. The distance and duration fields are related to how far and long the runner had to travel to deliver the order to the respective customer.

There are some known data issues with this table so be careful when using this in your queries - make sure to check the data types for each column in the schema SQL!

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 Ca
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 Ca
10	1	2020-01-11 18:50:20	10km	10minutes	null

Table 4: pizza_names

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

pizza_id	pizza_name
1	Meat Lovers
2	Vegetarian

Table 5: pizza_recipes

Each pizza_id has a standard set of toppings which 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

Table 6: 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

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
- Bonus DML Challenges (DML = Data Manipulation Language)

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

Again, there are many questions in this case study - please feel free to pick and choose which ones you'd like to try!

Before you start writing your SQL queries however - you might want to investigate the data, you may want to do something with some of those null values and data types in the `customer_orders` and `runner_orders` tables!

B. Runner and Customer Experience

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

```
1 • select weekofyear(registration_date+interval 1 week), count(runner_id)
2   from runners
3  group by 1;
4
```

<		
Result Grid		
Filter Rows: <input type="text"/>		
Export:		
Wrap Cell Content:		
	weekofyear(registration_date+interval 1 week)	count(runner_id)
▶	1	2
	2	1
	3	1

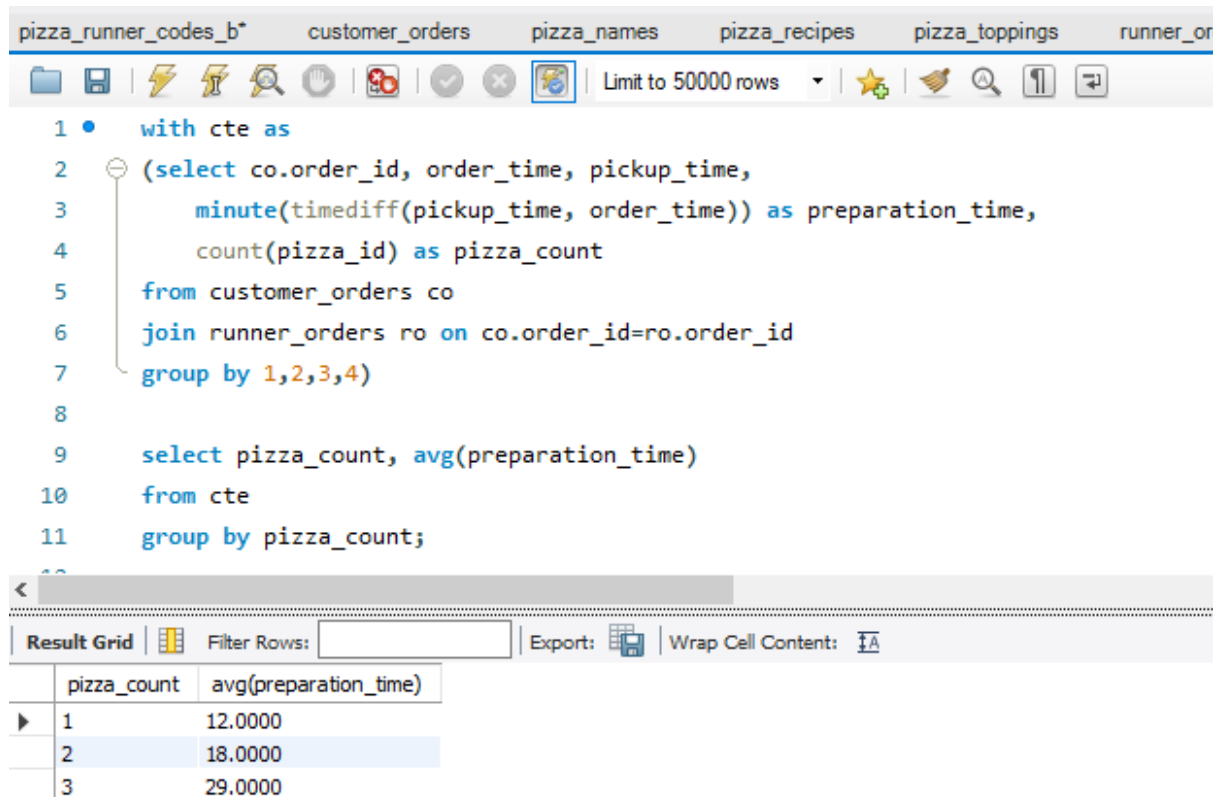
2. What was the average time in minutes it took for each runner to arrive at the Pizza Runner HQ to pickup the order?

pizza_runner_codes_b* customer_orders x pizza_names pizza_recipes pizza_toppings runner_orders r

```
1 • select runner_id, avg(minute(timediff(pickup_time, order_time))) as avg_time
2   from customer_orders co
3  join runner_orders ro on co.order_id=ro.order_id
4  group by runner_id;
5
6
```

<		
Result Grid		
Filter Rows: <input type="text"/>		
Export:		
Wrap Cell Content:		
	runner_id	avg_time
▶	1	15.3333
	2	23.4000
	3	10.0000

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



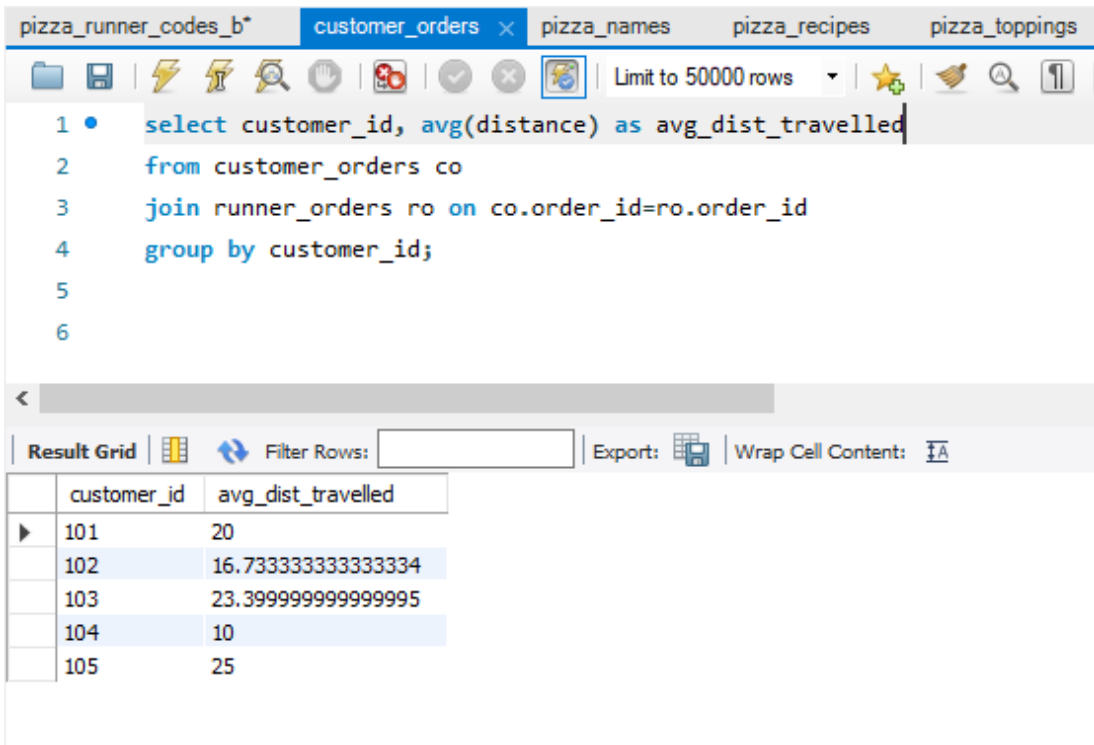
The screenshot shows a SQL IDE with a query editor and a result grid. The query is as follows:

```
1 • with cte as
2   (select co.order_id, order_time, pickup_time,
3         minute(timediff(pickup_time, order_time)) as preparation_time,
4         count(pizza_id) as pizza_count
5   from customer_orders co
6   join runner_orders ro on co.order_id=ro.order_id
7   group by 1,2,3,4)
8
9   select pizza_count, avg(preparation_time)
10  from cte
11  group by pizza_count;
```

The result grid shows the following data:

	pizza_count	avg(preparation_time)
▶	1	12.0000
	2	18.0000
	3	29.0000

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



The screenshot shows a SQL IDE with a query editor and a result grid. The query is as follows:

```
1 • select customer_id, avg(distance) as avg_dist_travelled
2   from customer_orders co
3   join runner_orders ro on co.order_id=ro.order_id
4   group by customer_id;
5
6
```

The result grid shows the following data:

	customer_id	avg_dist_travelled
▶	101	20
	102	16.733333333333334
	103	23.399999999999995
	104	10
	105	25

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

Database interface showing a SQL query and its results.

Query:

```
1 • select max(duration) as max_duration, min(duration) as min_duration,
2       max(duration)-min(duration) as difference
3 from runner_orders;
4
5
```

Result Grid:

	max_duration	min_duration	difference
▶	40	10	30

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

Database interface showing a SQL query and its results.

Query:

```
1 • select runner_id, round((distance*60/duration),1) as speed,
2       round(avg(distance*60/duration) over(partition by runner_id),2) as avg_speed
3 from runner_orders;
4
5
```

Result Grid:

	runner_id	speed	avg_speed
▶	1	37.5	45.54
	1	44.4	45.54
	1	40.2	45.54
	1	60	45.54
	2	35.1	62.9
	2	60	62.9
	2	93.6	62.9
	2	NULL	62.9
	3	40	40
	3	NULL	40

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

unner_codes_b* customer_orders pizza_names pizza_recipes pizza_toppings runner_orders x runner_orders

Limit to 50000 rows

```
1 with cte as
2   (select runner_id,
3      sum(case when cancellation is not NULL then 1 else 0 end) as cancelled_order,
4      sum(case when cancellation is NULL then 1 else 0 end) as delivered_order
5   from runner_orders
6   group by runner_id)
7
8   select runner_id, cancelled_order, delivered_order,
9      round((delivered_order/(delivered_order+cancelled_order))*100,1) as succesful_delivery_prnt
10  from cte;
11
```

Result Grid Filter Rows: Export: Wrap Cell Content:

	runner_id	cancelled_order	delivered_order	succesful_delivery_prnt
▶	1	0	4	100.0
	2	1	3	75.0
	3	1	1	50.0

C. Ingredient Optimisation

1. What are the standard ingredients for each pizza?

The screenshot shows a SQL IDE with tabs for 'pizza_runner_codes_c', 'pizza_toppings', 'pizza_recipes', 'pizza_names', 'customer_orders', and 'pizza_rec'. The 'pizza_recipes' tab is active. The SQL query is as follows:

```
1 with recursive cte as
2 (select *
3  from pizza_recipes
4  union all
5  select pizza_id, regexp_replace(toppings, '^[^,]*,') toppings
6  from cte
7  where toppings like '%,%'),
8
9  cte2 as
10 (select pizza_id, trim(regexp_replace(toppings, ',.*', '')) toppings
11  from cte
12  order by pizza_id),
13
14  cte3 as
15 (select *
16  from cte2
17  join pizza_toppings pt on cte2.toppings=pt.topping_id)
18
19 select pizza_id, group_concat(topping_name) as standard_ingredients
20 from cte3
21 group by pizza_id
```

Below the query editor, the 'Result Grid' shows the following data:

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

2. What was the most commonly added extra?

The screenshot shows a SQL IDE with tabs for 'pizza_runner_codes_c*', 'pizza_toppings', 'pizza_recipes', 'pizza_names', and 'customer_orders'. The 'pizza_recipes' tab is active. The SQL query is as follows:

```
1 with recursive cte as
2 (select extras
3  from customer_orders
4  union all
5  select regexp_replace(extras, '^[^,]*,') extras
6  from cte
7  where extras like '%,%'),
8
9  cte2 as
10 (select trim(regexp_replace(extras, ',.*', '')) extras, count(extras) as count
11  from cte
12  where extras is not NULL
13  group by 1
14  order by extras)
15
16 select cte2.extras, count, topping_name
17 from cte2
18 join pizza_toppings pt on cte2.extras=pt.topping_id;
19
```

Below the query editor, the 'Result Grid' shows the following data:

	extras	count	topping_name
▶	1	4	Bacon
	4	1	Cheese
	5	1	Chicken

3. What was the most common exclusion?

SQL query editor showing a query to find the most common exclusion from the customer_orders table.

```
1 with recursive cte as
2   (select exclusions
3    from customer_orders
4   union all
5    select regexp_replace(exclusions,'^[,]*','') exclusions
6    from cte
7   where exclusions like '%,%'),
8
9   cte2 as
10  (select trim(regexp_replace(exclusions','.*','')) exclusions, count(exclusions) as count
11   from cte
12   where exclusions is not NULL
13   group by 1
14   order by count desc)
15
16  select cte2.exclusions, count, topping_name
17  from cte2
18  join pizza_toppings pt on cte2.exclusions=pt.topping_id;
```

Result Grid:

exclusions	count	topping_name
4	4	Cheese
2	1	BBQ Sauce
6	1	Mushrooms

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

SQL query editor showing a query to generate order items from the customer_orders table.

```
1 with recursive cte as
2   (select order_id, customer_id, pizza_id, exclusions, extras
3    from customer_orders
4   union all
5    select order_id, customer_id, pizza_id, regexp_replace(exclusions,'^[,]*','') exclusions,
6          regexp_replace(extras,'^[,]*','') extras
7    from cte
8   where exclusions like '%,%' and extras like '%,%'),
9
10  cte2 as
11  (select order_id, customer_id, pizza_id,
12   trim(regexp_replace(exclusions','.*','')) exclusions, trim(regexp_replace(extras','.*','')) extras
13   from cte)
```

Result Grid:

order_id	customer_id	pizza_name	exclusions	extras
1	101	Meatlovers	NULL	NULL
2	101	Meatlovers	NULL	NULL
3	102	Meatlovers	NULL	NULL
3	102	Vegetarian	NULL	NULL
4	103	Meatlovers	Cheese,Cheese	NULL
4	103	Vegetarian	Cheese	NULL
5	104	Meatlovers	NULL	Bacon
6	101	Vegetarian	NULL	NULL
7	105	Vegetarian	NULL	Bacon
8	102	Meatlovers	NULL	NULL
9	103	Meatlovers	Cheese,Bacon	Cheese,Bacon
10	104	Meatlovers	BBQ Sauce,Bacon,Mushrooms,Cheese	BBQ Sauce,Bacon,Mushrooms,Cheese

Insights

The following topics are completely covered in this case study:

- Common Table Expressions
- Group By and Aggregates
- Table Joins
- Case When clause
- Subqueries
- Group_concat
- Regexp_replace
- Window functions
- Date and Time functions like minute, timediff, weekofyear, etc.

The following insights can be gathered for this case study:

- As the number of pizzas increases the average time for cooking also increases.
- While the average speed of Runner 2 is maximum i.e., 62.9 kmph, but the average time to reach for pickup is minimum for Runner 3.
- Runner 1 has maximum successful delivery while Runner 3 has 50% of successful deliveries only.
- Bacon is the most added extra in Pizzas and Cheese is the most common exclusion from the Pizzas.
- Cheese and Mushrooms are put in both the pizzas.
- Customer 104 is nearest to restaurant while customer 105 lives farthest.