



8 Week SQL Challenge

Start your SQL learning journey today!



click here ↗

[Data With Danny](#)

[Case Studies](#)

[Getting Started](#)

[Resources](#)

[About](#)

Case Study #1 - Danny's Diner

Danny Ma · May 1, 2021

8WEEKSQLCHALLENGE.COM CASE STUDY #1



THE TASTE OF SUCCESS

DATAWITHDANNY.COM

Introduction

Danny seriously loves Japanese food so in the beginning of 2021, he decides to embark upon a risky venture and opens up a cute little restaurant that sells his 3 favourite foods: sushi, curry and ramen.

Danny's Diner is in need of your assistance to help the restaurant stay afloat - the restaurant has captured some very basic data from their few months of operation but have no idea how to use their data to help them run the business.

Problem Statement

Danny wants to use the data to answer a few simple questions about his customers, especially about their visiting patterns, how much money they've spent and also which menu items are their favourite. Having this deeper connection with his customers will help him deliver a better and more personalised experience for his loyal customers.

He plans on using these insights to help him decide whether he should expand the existing customer loyalty program - additionally he needs help to generate some basic datasets so his team can easily inspect the data without needing to use SQL.

Danny has provided you with a sample of his overall customer data due to privacy issues - but he hopes that these examples are enough for you to write fully functioning SQL queries to help him answer his questions!

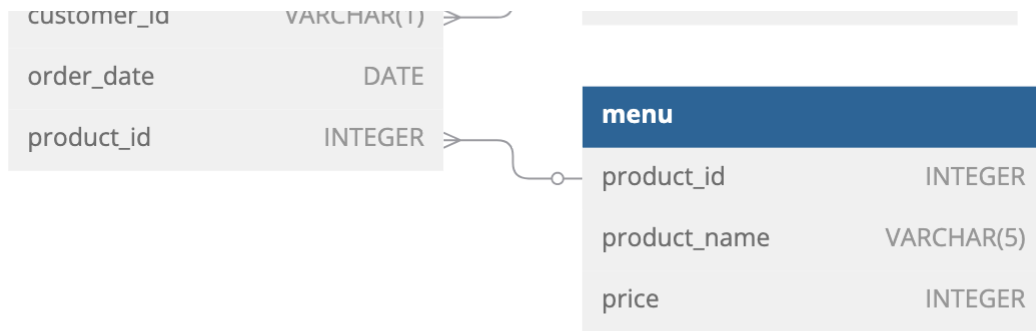
Danny has shared with you 3 key datasets for this case study:

- **sales**
- **menu**
- **members**

You can inspect the entity relationship diagram and example data below.

Entity Relationship Diagram





Example Datasets

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

Table 1: sales

The `sales` table captures all `customer_id` level purchases with an corresponding `order_date` and `product_id` information for when and what menu items were ordered.

customer_id	order_date	product_id
A	2021-01-01	1
A	2021-01-01	2
A	2021-01-07	2
A	2021-01-10	3
A	2021-01-11	3
A	2021-01-11	3
B	2021-01-01	2
B	2021-01-02	2
B	2021-01-04	1
B	2021-01-11	1

customer_id	order_date	product_id
B	2021-01-16	3
B	2021-02-01	3
C	2021-01-01	3
C	2021-01-01	3
C	2021-01-07	3

Table 2: menu

The **menu** table maps the **product_id** to the actual **product_name** and **price** of each menu item.

product_id	product_name	price
1	sushi	10
2	curry	15
3	ramen	12

Table 3: members

The final **members** table captures the **join_date** when a **customer_id** joined the beta version of the Danny's Diner loyalty program.

customer_id	join_date
A	2021-01-07
B	2021-01-09

Interactive SQL Session

You can use the embedded DB Fiddle below to easily access these example datasets - this interactive session has everything you need to start solving these questions using SQL.

You can click on the [Edit on DB Fiddle](#) link on the top right hand corner of the embedded session below and it will take you to a fully functional SQL editor where you can write your own queries to analyse the data.

You can feel free to choose any SQL dialect you'd like to use, the existing Fiddle is using PostgreSQL 13 as default.

Serious SQL students have access to a dedicated SQL script in the 8 Week SQL Challenge section of the course which they can use to generate relevant temporary tables like we've done throughout the entire course!

[Schema SQL](#) [Query SQL](#) [Results](#)[Edit on DB Fiddle](#)

```
CREATE SCHEMA dannys_diner;
SET search_path = dannys_diner;

CREATE TABLE sales (
  "customer_id" VARCHAR(1),
  "order_date" DATE,
  "product_id" INTEGER
);

INSERT INTO sales
  ("customer_id", "order_date", "product_id")
VALUES
  ('A', '2021-01-01', '1'),
  ('A', '2021-01-01', '2'),
  ('A', '2021-01-07', '2'),
  ('A', '2021-01-10', '3'),
  ('A', '2021-01-11', '3'),
  ('A', '2021-01-11', '3'),
  ('B', '2021-01-01', '2'),
  ('B', '2021-01-02', '2'),
  ('B', '2021-01-04', '1'),
  ('B', '2021-01-11', '1'),
  ('B', '2021-01-16', '3'),
  ('B', '2021-02-01', '3'),
  ('C', '2021-01-01', '3'),
  ('C', '2021-01-01', '3'),
  ('C', '2021-01-07', '3');

CREATE TABLE menu (
  "product_id" INTEGER,
  "product_name" VARCHAR(5),
  "price" INTEGER
);

INSERT INTO menu
  ("product_id", "product_name", "price")
VALUES
  ('1', 'sushi', '10'),
  ('2', 'curry', '15'),
  ('3', 'ramen', '12');

CREATE TABLE members (
```

Case Study Questions

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

1. What is the total amount each customer spent at the restaurant?
2. How many days has each customer visited the restaurant?
3. What was the first item from the menu purchased by each customer?
4. What is the most purchased item on the menu and how many times was it purchased by all customers?
5. Which item was the most popular for each customer?
6. Which item was purchased first by the customer after they became a member?
7. Which item was purchased just before the customer became a member?
8. What is the total items and amount spent for each member before they became a member?
9. If each \$1 spent equates to 10 points and sushi has a 2x points multiplier - how many points would each customer have?
10. In the first week after a customer joins the program (including their join date) they earn 2x points on all items, not just sushi - how many points do customer A and B have at the end of January?

Bonus Questions

Join All The Things

The following questions are related creating basic data tables that Danny and his team can use to quickly derive insights without needing to join the underlying tables using SQL.

Recreate the following table output using the available data:

customer_id	order_date	product_name	price	member
A	2021-01-01	curry	15	N
A	2021-01-01	sushi	10	N

customer_id	order_date	product_name	price	member
A	2021-01-07	curry	15	Y
A	2021-01-10	ramen	12	Y
A	2021-01-11	ramen	12	Y
A	2021-01-11	ramen	12	Y
B	2021-01-01	curry	15	N
B	2021-01-02	curry	15	N
B	2021-01-04	sushi	10	N
B	2021-01-11	sushi	10	Y
B	2021-01-16	ramen	12	Y
B	2021-02-01	ramen	12	Y
C	2021-01-01	ramen	12	N
C	2021-01-01	ramen	12	N
C	2021-01-07	ramen	12	N

Rank All The Things

Danny also requires further information about the **ranking** of customer products, but he purposely does not need the ranking for non-member purchases so he expects null **ranking** values for the records when customers are not yet part of the loyalty program.

customer_id	order_date	product_name	price	member	ranking
A	2021-01-01	curry	15	N	null
A	2021-01-01	sushi	10	N	null
A	2021-01-07	curry	15	Y	1
A	2021-01-10	ramen	12	Y	2

customer_id	order_date	product_name	price	member	ranking
A	2021-01-11	ramen	12	Y	3
A	2021-01-11	ramen	12	Y	3
B	2021-01-01	curry	15	N	null
B	2021-01-02	curry	15	N	null
B	2021-01-04	sushi	10	N	null
B	2021-01-11	sushi	10	Y	1
B	2021-01-16	ramen	12	Y	2
B	2021-02-01	ramen	12	Y	3
C	2021-01-01	ramen	12	N	null
C	2021-01-01	ramen	12	N	null
C	2021-01-07	ramen	12	N	null

Next Steps

It's highly recommended to save all of your code in a separate IDE or text editor as you are trying to solve the problems in the provided SQL Fiddle instance above!

If you'd like to use this case study for one of your portfolio projects or in a personal blog post - please remember to link back to this URL and also don't forget to share some LinkedIn updates using the **#8WeekSQLChallenge** hashtag and remember to tag me!

Ready for the next 8 Week SQL challenge case study? Click on the banner below to get started with case study #2!



Conclusion

I really hope you enjoyed this fun little case study - it definitely was fun for me to create!

Official Solutions

If you'd like to see the official code solutions and explanations for this case study and a whole lot more, please consider joining me for the [Serious SQL course](#) - you'll get access to all course materials and I'm on hand to answer all of your additional SQL questions directly!

Serious SQL is priced at \$49USD and \$29 for students and includes access to all written course content, community events as well as live and recorded SQL training videos!

Please send an email to support@datawithdanny.com from your educational email or include your enrolment details or student identification for a speedy response!

The following topics relevant to the Danny's Diner case study are covered lots of depth in the Serious SQL course:

- Common Table Expressions
- Group By Aggregates
- Window Functions for ranking
- Table Joins

Don't forget to review the comprehensive list of SQL resources I've put together for the 8 Week SQL Challenge on the [Resources](#) page!

Community Solutions

This section will be updated in the future with any community member solutions with a link to their respective GitHub repos!

Final Thoughts

The 8 Week SQL Challenge is proudly brought to you by me - Danny Ma and the [Data With Danny](#) virtual data apprenticeship program.

Students or anyone undertaking further studies are eligible for a \$20USD student discount off the price of Serious SQL please send an email to support@datawithdanny.com from your education email or include information about your enrolment for a fast response!

We have a large student community active on the official DWD Discord server with regular live events, trainings and workshops available to all Data With Danny students, plus early discounted access to all future paid courses.

There are also opportunities for 1:1 mentoring, resume reviews, interview training and more from myself or others in the DWD Mentor Team.

From your friendly data mentor, Danny :)

All 8 Week SQL Challenge Case Studies

All of the 8 Week SQL Challenge case studies can be found below:

- [Case Study #1 - Danny's Diner](#)
- [Case Study #2 - Pizza Runner](#)
- [Case Study #3 - Foodie-Fi](#)
- [Case Study #4 - Data Bank](#)
- [Case Study #5 - Data Mart](#)
- [Case Study #6 - Clique Bait](#)
- [Case Study #7 - Balanced Tree Clothing Co.](#)
- [Case Study #8 - Fresh Segments](#)

Share: [Twitter](#), [Facebook](#)