8WeekSQLCHALLENGE

Data-Driven Insights for Business Strategy: A Case Study On Danny’s Diner

Fatih Sahin

https://8weeksqlchallenge.com/case-study-1/

JULY 2023

CONTENTS

Introduction

This report presents the findings from the first case study of the '8WeekSQLChallenge' designed by Danny Ma, the Chief Data Mentor at Data With Danny and the Founder & CEO of Sydney Data Science. The challenge, available at https://8weeksqlchallenge.com/case-study-1/ , provides an excellent opportunity for SQL enthusiasts to sharpen their skills and apply them to a real-world business scenario.

The case study revolves around '**Danny's Diner**', a newly established restaurant specializing in Japanese cuisine. The primary aim of this analysis is to uncover valuable insights about customers' visiting patterns, their spending habits, their favourite menu items, and the potential benefits of expanding the existing customer loyalty program.

Danny provided three datasets for this case study: sales, menu, and members, all housed within the dannys_diner database schema. These datasets serve as the foundation for our exploratory queries and subsequent analysis.

To answer the ten main questions, along with two bonus questions, we utilized the online SQL environment provided by DB Fiddle. The queries used in this report are based on my own understanding and application of SQL, honed through practical experience and continuous learning.

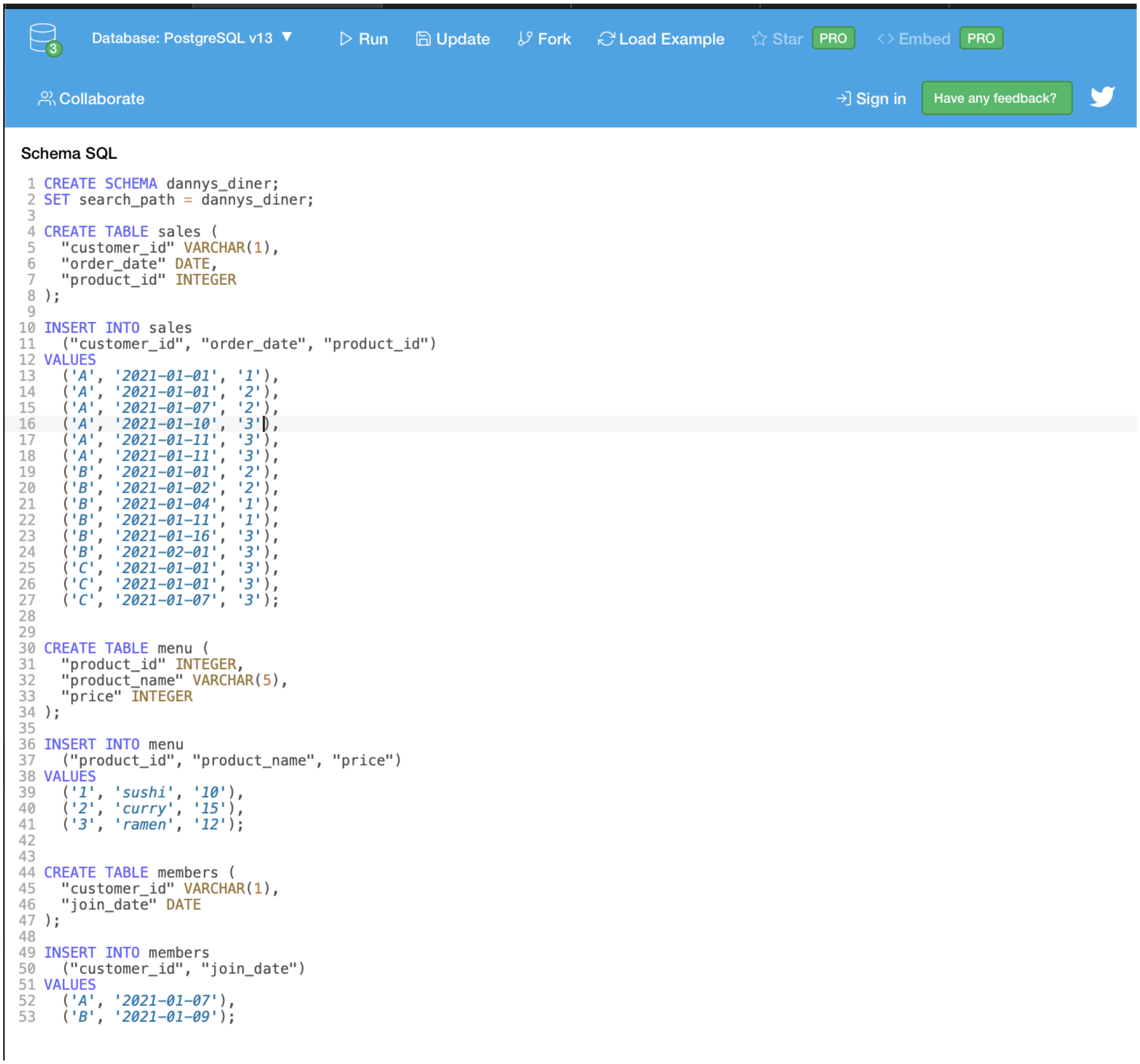
The purpose of this report is not only to document the findings and insights from the case study but also to serve as a valuable resource for other SQL learners and practitioners. The report demonstrates the power of SQL in deriving actionable business insights from raw data, providing a practical example of data-driven decision-making.

Problem Statement

- Danny, the owner of Danny's Diner, seeks a deeper understanding of his customers to provide a personalized dining experience.
- He wants to analyze data on customer visiting patterns, total expenditures, and favourite menu items.
- Insights from this data will inform his decision on expanding the existing customer loyalty program.
- In addition to SQL-based analysis, Danny wants to create basic datasets for his team to inspect without SQL.
- Danny has provided three key datasets (sales, menu, and members) to enable this analysis while respecting customer privacy.

Creating Schema and Tables

All datasets exist within the `dannys_diner` database schema.



Schema SQL

```sql
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 (
  "customer_id" VARCHAR(1),
  "join_date" DATE
);

INSERT INTO members
  ("customer_id", "join_date")
VALUES
  ('A', '2021-01-07'),
  ('B', '2021-01-09');
```

Entity Relationship Diagram

In this case, we have three tables:

sales: This table records each sale made at the restaurant. Each row represents a unique transaction and contains the following columns:

- **customer_id**: A unique identifier for the customer who made the purchase.
- **order_date**: The date the purchase was made.
- **product_id**: A unique identifier for the product that was sold.

menu: This table lists all the items available for purchase at the restaurant. Each row represents a unique menu item and contains the following columns:

- **product_id**: A unique identifier for the product. This is the same ID used in the sales table.
- **product_name**: The name of the product.
- **price**: The price of the product.

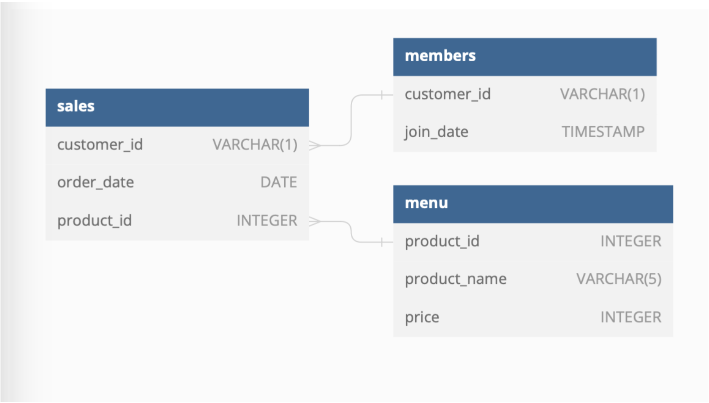
members: This table records which customers are members of the loyalty program and when they joined. Each row represents a unique customer and contains the following columns:

- **customer_id**: A unique identifier for the customer. This is the same ID used in the sales table.
- **join_date**: The date the customer joined the loyalty program.

The relationships between the tables are as follows:

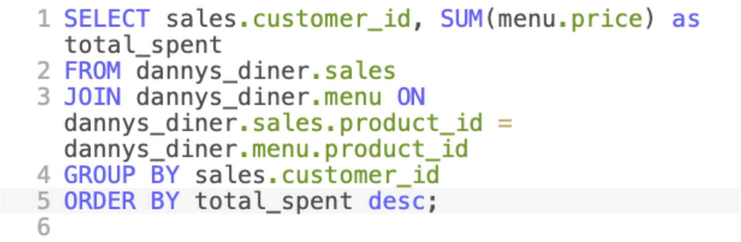
- **sales** and **menu** are related through **product_id**. Each sale in the sales table corresponds to a product in the **menu** table.
- **sales** and **members** are related through **customer_id**. Each sale in the sales table is made by a customer, and that customer might be a member as recorded in the members table.

Entity Relationship Diagram

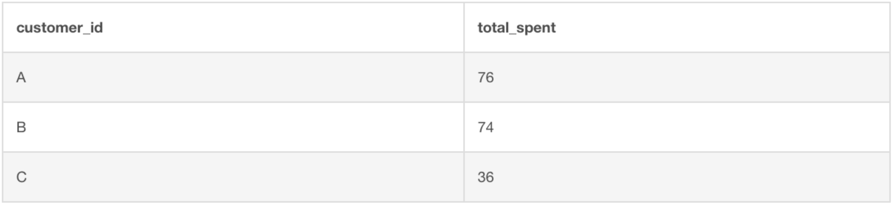


Case Study Questions

1.What is the total amount each customer spent at the restaurant?

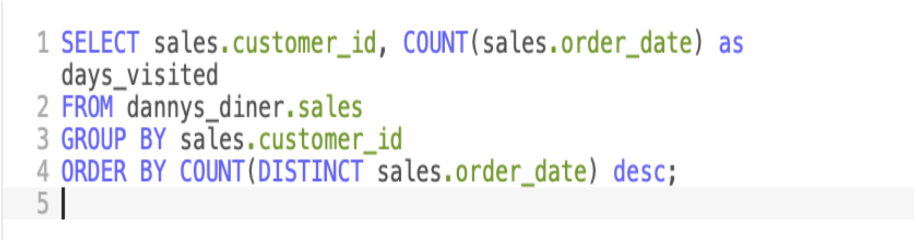


```
SELECT sales.customer_id, SUM(menu.price) as
total_spent
FROM dannys_diner.sales
JOIN dannys_diner.menu ON
dannys_diner.sales.product_id =
dannys_diner.menu.product_id
GROUP BY sales.customer_id
ORDER BY total_spent desc;
```

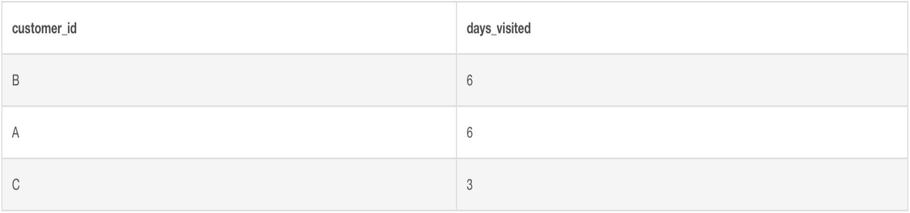


customer_id	total_spent
A	76
B	74
C	36

2. How many days has each customer visited the restaurant?

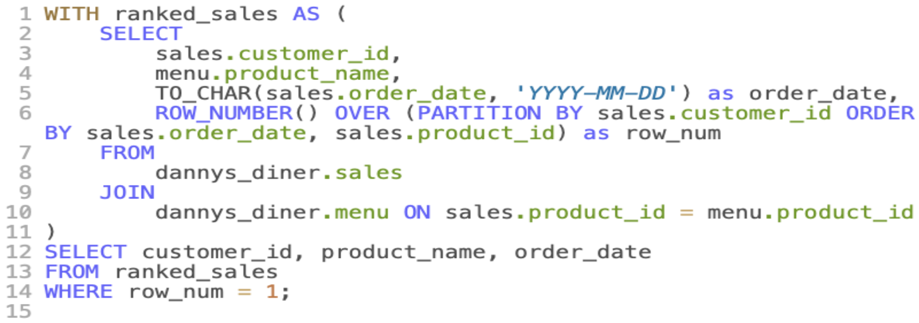


```
SELECT sales.customer_id, COUNT(sales.order_date) as
days_visited
FROM dannys_diner.sales
GROUP BY sales.customer_id
ORDER BY COUNT(DISTINCT sales.order_date) desc;
```

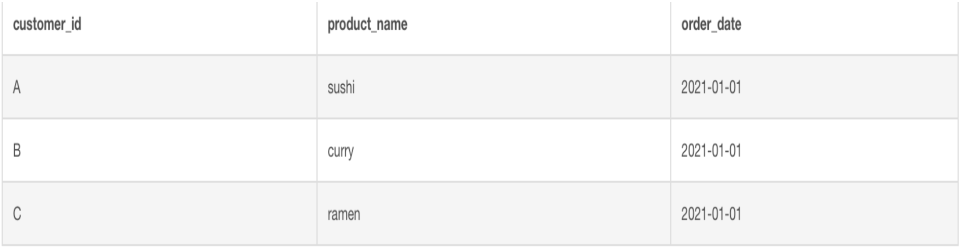


customer_id	days_visited
B	6
A	6
C	3

3. What was the first item from the menu purchased by each customer?

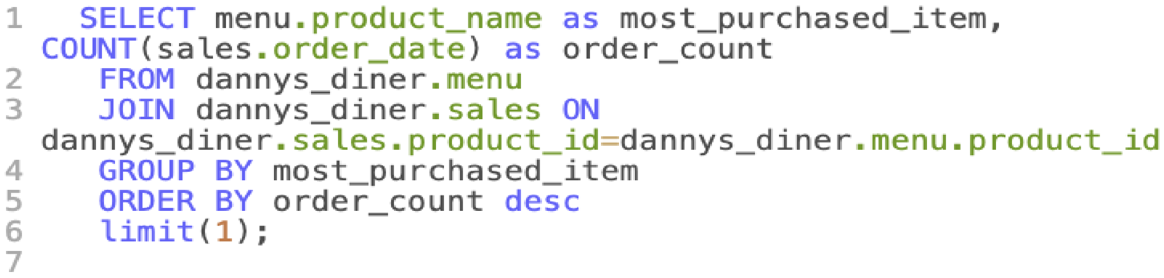


```sql
WITH ranked_sales AS (
    SELECT
        sales.customer_id,
        menu.product_name,
        TO_CHAR(sales.order_date, 'YYYY-MM-DD') as order_date,
        ROW_NUMBER() OVER (PARTITION BY sales.customer_id ORDER BY sales.order_date, sales.product_id) as row_num
    FROM
        dannys_diner.sales
    JOIN
        dannys_diner.menu ON sales.product_id = menu.product_id
)
SELECT customer_id, product_name, order_date
FROM ranked_sales
WHERE row_num = 1;
```

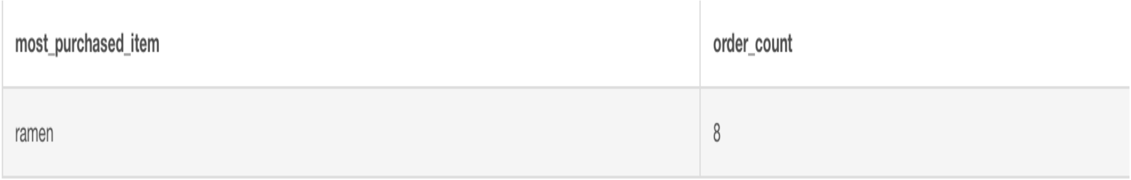


customer_id	product_name	order_date
A	sushi	2021-01-01
B	curry	2021-01-01
C	ramen	2021-01-01

4. What is the most purchased item on the menu and how many times was it purchased by all customers?

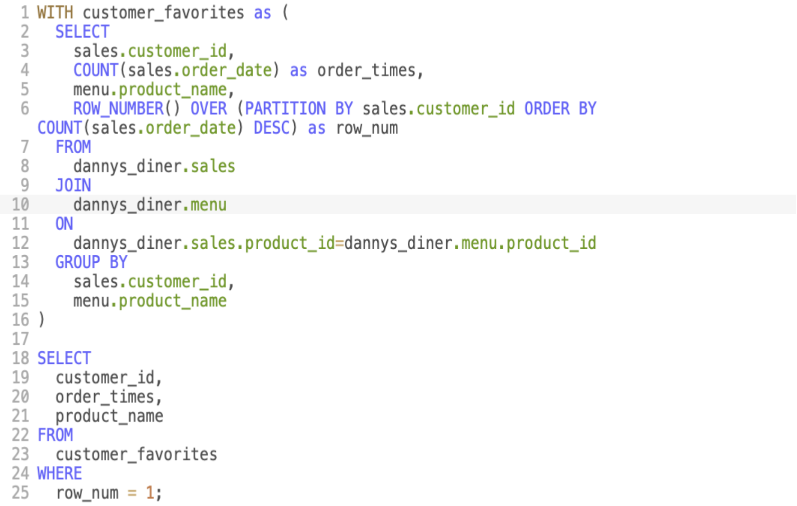


```sql
  SELECT menu.product_name as most_purchased_item, COUNT(sales.order_date) as order_count
  FROM dannys_diner.menu
  JOIN dannys_diner.sales ON dannys_diner.sales.product_id=dannys_diner.menu.product_id
  GROUP BY most_purchased_item
  ORDER BY order_count desc
  limit(1);
```

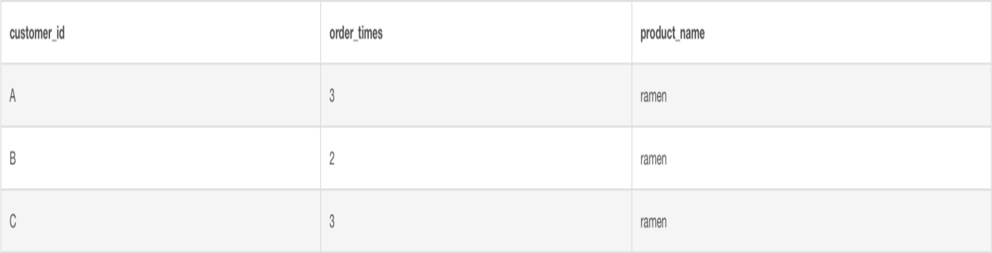


most_purchased_item	order_count
ramen	8

5. Which item was the most popular for each customer?

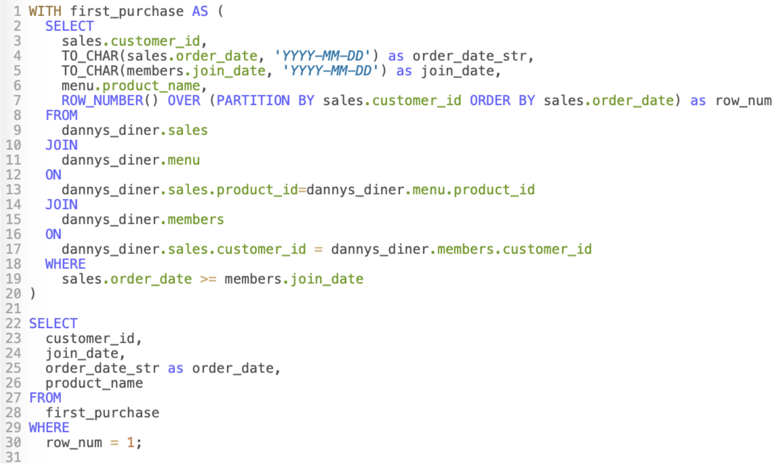


```
WITH customer_favorites as (
  SELECT
    sales.customer_id,
    COUNT(sales.order_date) as order_times,
    menu.product_name,
    ROW_NUMBER() OVER (PARTITION BY sales.customer_id ORDER BY COUNT(sales.order_date) DESC) as row_num
  FROM
    dannys_diner.sales
  JOIN
    dannys_diner.menu
  ON
    dannys_diner.sales.product_id=dannys_diner.menu.product_id
  GROUP BY
    sales.customer_id,
    menu.product_name
)

SELECT
  customer_id,
  order_times,
  product_name
FROM
  customer_favorites
WHERE
  row_num = 1;
```

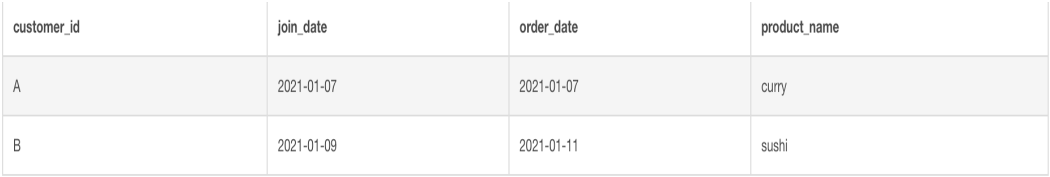


customer_id	order_times	product_name
A	3	ramen
B	2	ramen
C	3	ramen

6. Which item was purchased first by the customer after they became a member?

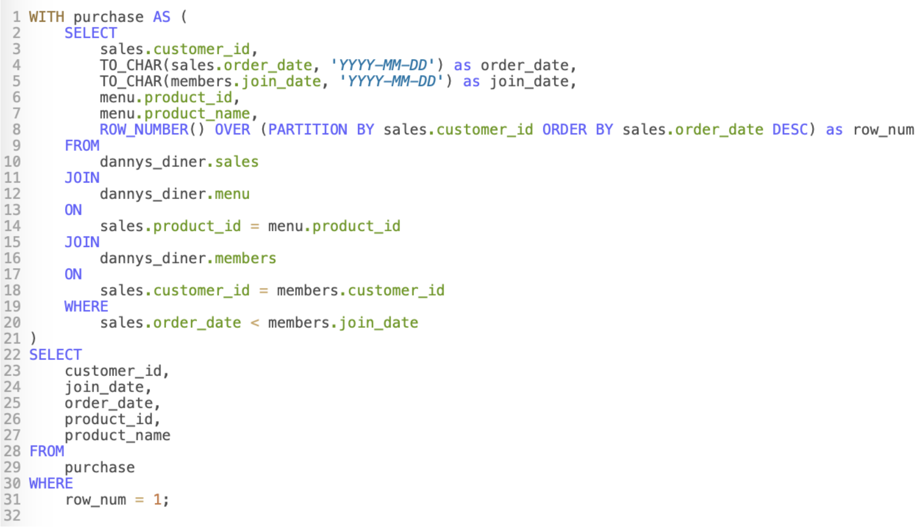


```
WITH first_purchase AS (
  SELECT
    sales.customer_id,
    TO_CHAR(sales.order_date, 'YYYY-MM-DD') as order_date_str,
    TO_CHAR(members.join_date, 'YYYY-MM-DD') as join_date,
    menu.product_name,
    ROW_NUMBER() OVER (PARTITION BY sales.customer_id ORDER BY sales.order_date) as row_num
  FROM
    dannys_diner.sales
  JOIN
    dannys_diner.menu
  ON
    dannys_diner.sales.product_id=dannys_diner.menu.product_id
  JOIN
    dannys_diner.members
  ON
    dannys_diner.sales.customer_id = dannys_diner.members.customer_id
  WHERE
    sales.order_date >= members.join_date
)

SELECT
  customer_id,
  join_date,
  order_date_str as order_date,
  product_name
FROM
  first_purchase
WHERE
  row_num = 1;

```

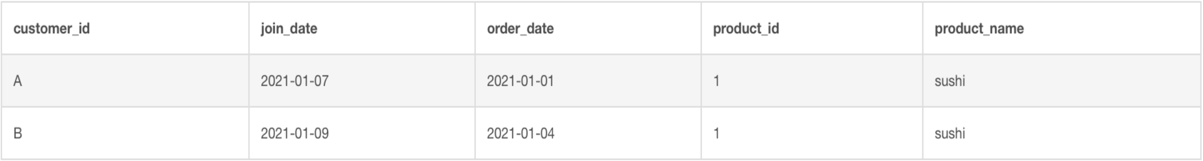


customer_id	join_date	order_date	product_name
A	2021-01-07	2021-01-07	curry
B	2021-01-09	2021-01-11	sushi

7. Which item was purchased just before the customer became a member?

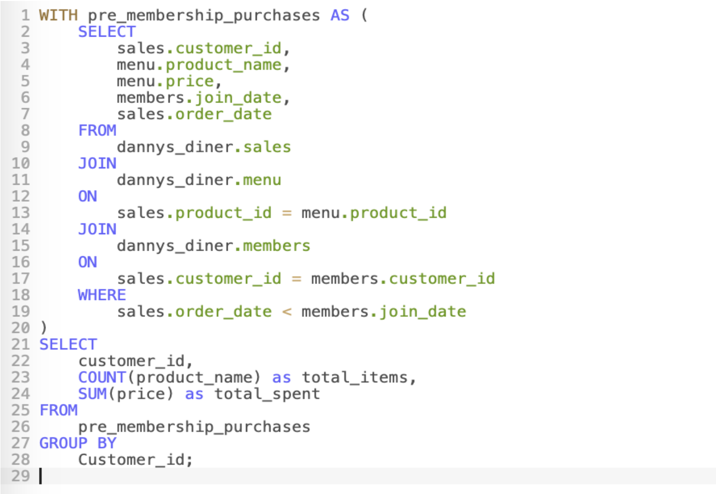


```
WITH purchase AS (
    SELECT
        sales.customer_id,
        TO_CHAR(sales.order_date, 'YYYY-MM-DD') as order_date,
        TO_CHAR(members.join_date, 'YYYY-MM-DD') as join_date,
        menu.product_id,
        menu.product_name,
        ROW_NUMBER() OVER (PARTITION BY sales.customer_id ORDER BY sales.order_date DESC) as row_num
    FROM
        dannys_diner.sales
    JOIN
        dannys_diner.menu
    ON
        sales.product_id = menu.product_id
    JOIN
        dannys_diner.members
    ON
        sales.customer_id = members.customer_id
    WHERE
        sales.order_date < members.join_date
)
SELECT
    customer_id,
    join_date,
    order_date,
    product_id,
    product_name
FROM
    purchase
WHERE
    row_num = 1;

```

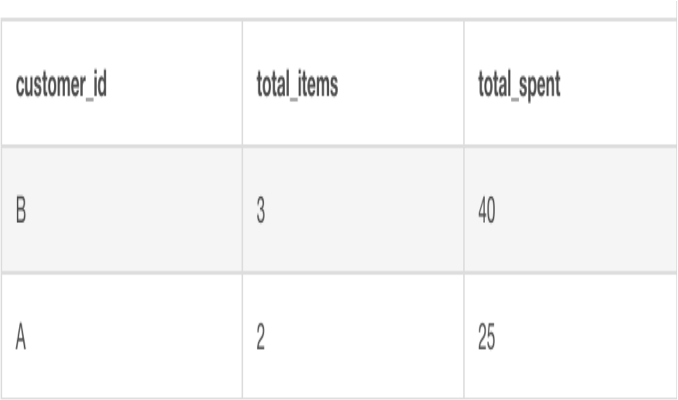


customer_id	join_date	order_date	product_id	product_name
A	2021-01-07	2021-01-01	1	sushi
B	2021-01-09	2021-01-04	1	sushi

8. What are the total items and amount spent for each member before they became a member?

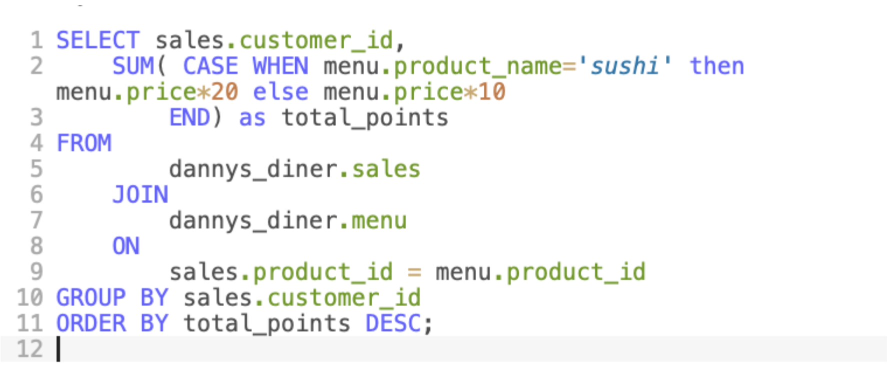


```sql
WITH pre_membership_purchases AS (
    SELECT
        sales.customer_id,
        menu.product_name,
        menu.price,
        members.join_date,
        sales.order_date
    FROM
        dannys_diner.sales
    JOIN
        dannys_diner.menu
    ON
        sales.product_id = menu.product_id
    JOIN
        dannys_diner.members
    ON
        sales.customer_id = members.customer_id
    WHERE
        sales.order_date < members.join_date
)
SELECT
    customer_id,
    COUNT(product_name) as total_items,
    SUM(price) as total_spent
FROM
    pre_membership_purchases
GROUP BY
    Customer_id;
```

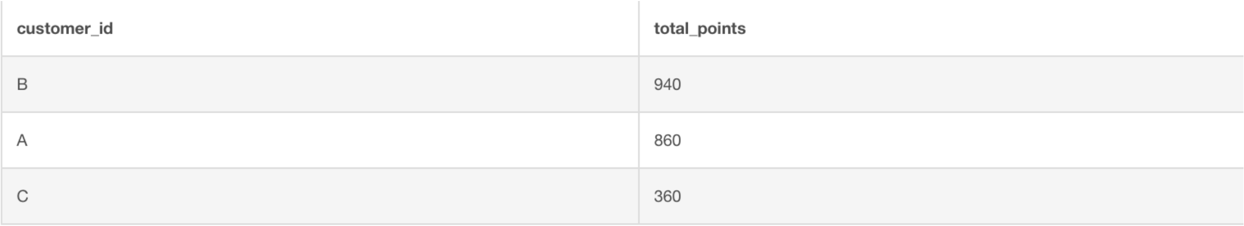


customer_id	total_items	total_spent
B	3	40
A	2	25

9. If each $1 spent equates to 10 points and sushi has a 2x points multiplier - how many points would each customer have?

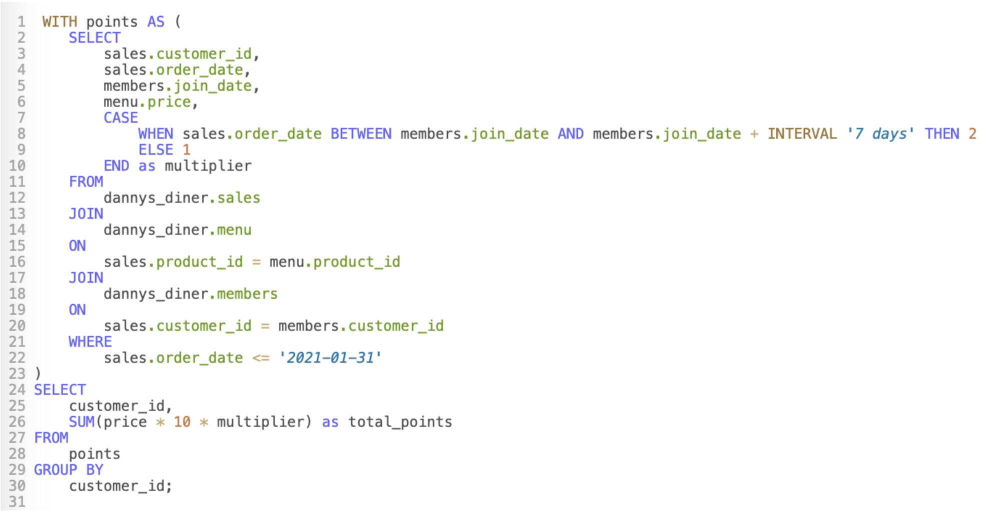


```sql
SELECT sales.customer_id,
    SUM( CASE WHEN menu.product_name='sushi' then menu.price*20 else menu.price*10
        END) as total_points
FROM
        dannys_diner.sales
    JOIN
        dannys_diner.menu
    ON
        sales.product_id = menu.product_id
GROUP BY sales.customer_id
ORDER BY total_points DESC;
```

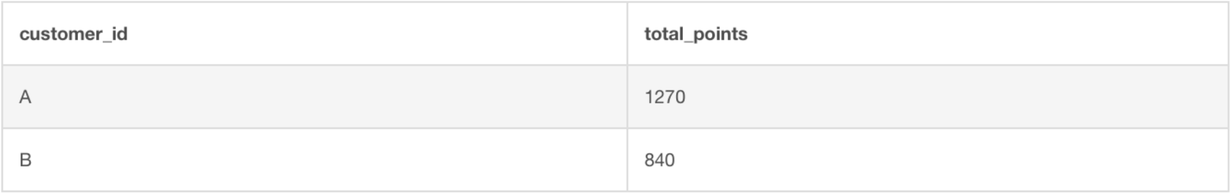


customer_id	total_points
B	940
A	860
C	360

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 customers A and B have at the end of January?



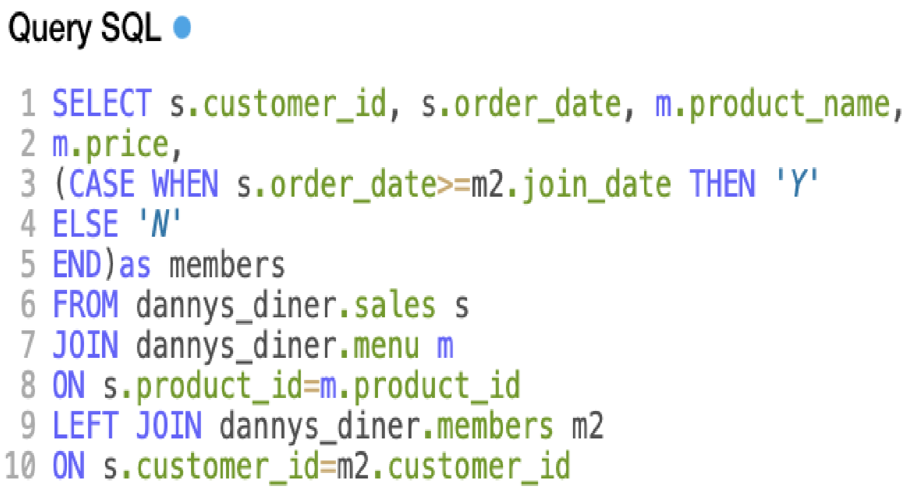
```
WITH points AS (
   SELECT
       sales.customer_id,
       sales.order_date,
       members.join_date,
       menu.price,
       CASE
           WHEN sales.order_date BETWEEN members.join_date AND members.join_date + INTERVAL '7 days' THEN 2
           ELSE 1
       END as multiplier
   FROM
       dannys_diner.sales
   JOIN
       dannys_diner.menu
   ON
       sales.product_id = menu.product_id
   JOIN
       dannys_diner.members
   ON
       sales.customer_id = members.customer_id
   WHERE
       sales.order_date <= '2021-01-31'
)
SELECT
    customer_id,
    SUM(price * 10 * multiplier) as total_points
FROM
    points
GROUP BY
    customer_id;
```



customer_id	total_points
A	1270
B	840

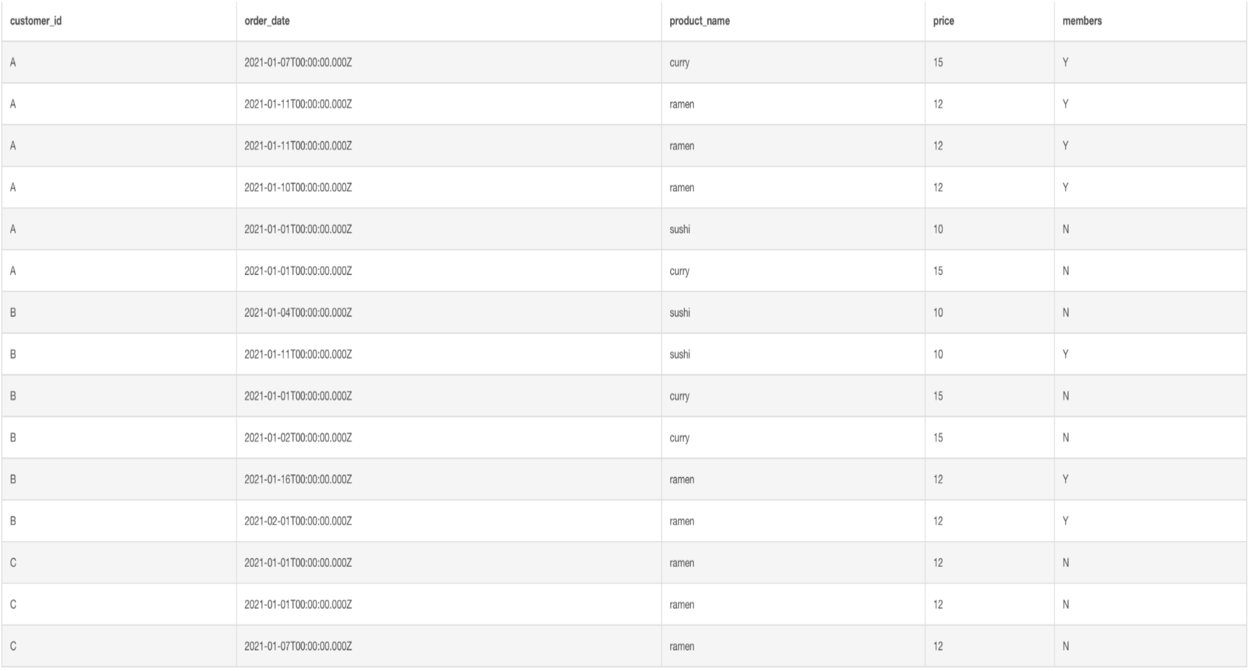
Bonus Question

Linking all the tables together to enable Danny's team to rapidly gain insights.



Query SQL

```sql
SELECT s.customer_id, s.order_date, m.product_name,
m.price,
(CASE WHEN s.order_date>=m2.join_date THEN 'Y'
ELSE 'N'
END)as members
FROM dannys_diner.sales s
JOIN dannys_diner.menu m
ON s.product_id=m.product_id
LEFT JOIN dannys_diner.members m2
ON s.customer_id=m2.customer_id
```



customer_id	order_date	product_name	price	members
A	2021-01-07T00:00:00.000Z	curry	15	Y
A	2021-01-11T00:00:00.000Z	ramen	12	Y
A	2021-01-11T00:00:00.000Z	ramen	12	Y
A	2021-01-10T00:00:00.000Z	ramen	12	Y
A	2021-01-01T00:00:00.000Z	sushi	10	N
A	2021-01-01T00:00:00.000Z	curry	15	N
B	2021-01-04T00:00:00.000Z	sushi	10	N
B	2021-01-11T00:00:00.000Z	sushi	10	Y
B	2021-01-01T00:00:00.000Z	curry	15	N
B	2021-01-02T00:00:00.000Z	curry	15	N
B	2021-01-16T00:00:00.000Z	ramen	12	Y
B	2021-02-01T00:00:00.000Z	ramen	12	Y
C	2021-01-01T00:00:00.000Z	ramen	12	N
C	2021-01-01T00:00:00.000Z	ramen	12	N
C	2021-01-07T00:00:00.000Z	ramen	12	N

Key Findings and Observations

- **Among all customers, Customer A** has the highest total spending, followed by Customer B.
- Both Customer A and B visited Danny's Diner six times, **double the number of visits by Customer C**, who only visited three times.
- **'Ramen' is the most preferred item** on the menu, accounting for **53% of total purchases**, with a total of eight orders.
- While 'Ramen' is the top choice for Customers A and C, **Customer B shows a balanced preference**, enjoying all three items equally.
- **Customer A leads in loyalty**, having joined the loyalty program before Customer B.
- Interestingly, despite the popularity of 'Ramen', Customers A and B ordered 'Sushi' and 'Curry' just before joining the 'Customer Loyalty program.
- **Customer C, who is not a member of the 'loyalty program**', has the least total purchases. To understand his preferences and improve his engagement, Danny's team could consider collecting feedback through a customer survey.

CONCLUSION

This case study served as an intriguing opportunity to further bolster my SQL skills and confidence in problem-solving. Throughout this exercise, I employed a range of SQL functions to answer the posed questions. This included **Aggregate functions** like **`COUNT`** and **`SUM`**, **Window functions** such as **`ROW_NUMBER`, `RANK`, and `DENSE_RANK`**, as well as **filtering and sorting functions**(**`WHERE`, `ORDER BY`, `GROUP BY`**). I also made use of constructed **complex subqueries with CTEs.**

The online SQL environment provided by DB Fiddle was the platform of choice for this case study, providing a robust environment for complex data analysis.

After finishing this case study, I feel more comfortable with using ranking tools and writing complicated queries. I'm also really excited to start the next case study from Danny Ma.