8WEEKSQLCHALLENGE.COM

CASE STUDY #1

THE TASTE OF SUCCESS

DATAWITHDANNY.COM

Solution to week #1 of the #8WeekSQLChallenge

Presented By Paul Alberque

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.

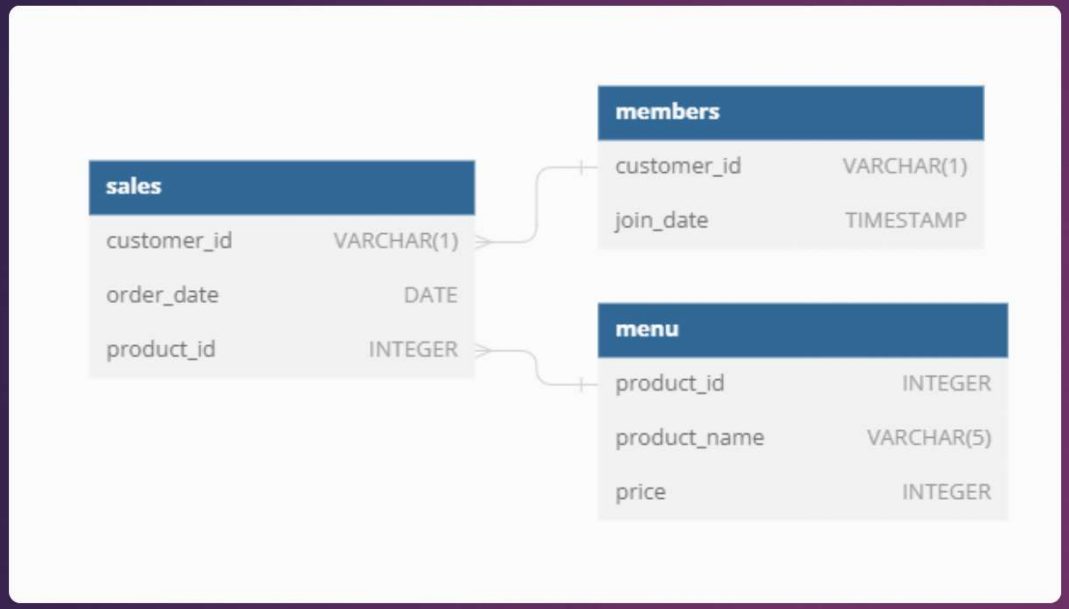
Danny's Diner is in need of assistance to help the restaurant stay afloat - the restaurant has captured some very basic data from their few months of operation but have no idea how to use their data to help them run the business.

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 which menu items are their favorite. Having this deeper connection with his customers will help him deliver a better and more personalized 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!



Entity Relationship Diagram

Datasets

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

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

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

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

Sales

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
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

Menu

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

Members

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

Let's set up the database in MySQL

```
---- 1) Start by creating the database

CREATE SCHEMA dannys_diner;
use dannys_diner;
```

```
---- 2) We are starting fresh by dropping the tables if they exist
----    prior to creation of the new table

Drop Table if exists sales;

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

Drop Table if exists members;

CREATE TABLE members (
  `customer_id` VARCHAR(1),
  `join_date` DATE
);

Drop Table if exists menu;

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

```
---- 3) Inserting the data into the previously created tables.

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);

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

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

Case Study Questions

- What is the total amount each customer spent at the restaurant?
- How many days has each customer visited the restaurant?
- What was the first item from the menu purchased by each customer?
- What is the most purchased item on the menu and how many times was it purchased by all customers?
- Which item was the most popular for each customer?
- Which item was purchased first by the customer after they became a member?
- Which item was purchased just before the customer became a member?
- What is the total items and amount spent for each member before they became a member?
- If each $1 spent equates to 10 points and sushi has a 2x points multiplier - how many points would each customer have?
- 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?

What is the **total** amount **each customer spent** at the restaurant?

```sql
SELECT
    s.customer_id,
    SUM(m.price) AS total_spent
FROM
    sales s
LEFT JOIN
    menu m ON s.product_id = m.product_id
GROUP BY
    s.customer_id;
```

customer_id	total_spent
A	76
B	74
C	36

Customers A & B came in close as the top spenders at Danny's Diner

How many **days** has **each customer visited** the restaurant?

```sql
SELECT
        customer_id,
        COUNT(DISTINCT order_date) AS order_count
FROM
        sales
GROUP BY customer_id
;
```

	customer_id	order_count
▶	A	4
	B	6
	C	2

Customers A & B had the highest number of orders. This result was expected as they are the biggest spenders.

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

```sql
SELECT customer_id, product_name
FROM
        /* A subquery is needed due to order of execution.
           In SQL although the SELECT statement is written first,
           FROM, WHERE, GROUP BY and HAVING statements execute first.
           In order to filter based on the new column order_ranking,
           it needs to be defined in a subquery or Common Tabe Expression (CTE).
           This allows a select statement to be executed during the WHERE statement
           Going forward, I will be using CTE's as I find they are easier to read.
        */
(SELECT
        s.customer_id,
        s.order_date,
        m.product_name,
                -- windows function to rank orders. The number restarts when  the column(s) we partition by change.
        ROW_NUMBER() OVER (PARTITION BY s.customer_id ORDER BY s.order_date) order_ranking
FROM
        sales s
JOIN
        menu m ON s.product_id = m.product_id
) AS ranked_orders

WHERE order_ranking = 1
;
```

	customer_id	product_name
▶	A	sushi
	B	curry
	C	ramen

There is no trend here as each customer ordered something different.

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

```
SELECT
    m.product_name, COUNT(product_name) AS times_ordered
FROM
    sales s
        LEFT JOIN
    menu m ON s.product_id = m.product_id
GROUP BY product_name
ORDER BY COUNT(product_name) DESC
;
```

	product_name	times_ordered
▶	ramen	8
	curry	4
	sushi	3

Ramen had the highest number of orders and Sushi had the least.

Which **item** was the **most popular** for **each customer**?

```sql
WITH cte_customer_order AS (
    SELECT
        s.customer_id,
        m.product_name,
        COUNT(product_name) AS times_ordered,
        RANK() OVER (PARTITION BY s.customer_id ORDER BY COUNT(product_name) DESC) AS order_rank
    FROM
        sales s
    INNER JOIN
        menu m ON s.product_id = m.product_id
    GROUP BY
        s.customer_id, m.product_name
)

SELECT
    customer_id, product_name, times_ordered
FROM
    cte_customer_order
WHERE
    order_rank = 1;
```

customer_id	product_name	times_ordered
A	ramen	3
B	curry	2
B	sushi	2
B	ramen	2
C	ramen	3

All 3 customers ordered the ramen on more than one occasion. Customer B is our only customer that has tried everything on the menu. Look into offering a promotion to customers if they try more of the products.

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

```sql
WITH cte_customer_order AS (
    SELECT
        mbr.customer_id,
        mbr.join_date,
        s.order_date,
        s.product_id,
        m.product_name,
        RANK() OVER (PARTITION BY s.customer_id ORDER BY order_date) AS order_rank
    FROM
        members mbr
    INNER JOIN
        sales s ON mbr.customer_id = s.customer_id -- inner join to exclude customers that are not a member
    LEFT JOIN
        menu m ON s.product_id = m.product_id
    WHERE
        mbr.customer_id IS NOT NULL
        AND s.order_date > mbr.join_date
    ORDER BY
        customer_id, s.order_date ASC
)

SELECT
    customer_id,
    product_name
FROM
    cte_customer_order
WHERE
    order_rank = 1;
```

	customer_id	product_name
▶	A	ramen
	B	sushi

We would need to see more data to make a conclusion.

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

```sql
WITH cte_customer_order AS (
    SELECT
        mbr.customer_id,
        mbr.join_date,
        s.order_date,
        s.product_id,
        m.product_name,
        RANK() OVER (PARTITION BY s.customer_id ORDER BY order_date DESC) AS order_rank
    FROM
        members mbr
    INNER JOIN
        sales s ON mbr.customer_id = s.customer_id -- inner join to exclude customers that are not a member
    LEFT JOIN
        menu m ON s.product_id = m.product_id
    WHERE
        mbr.customer_id IS NOT NULL
        AND s.order_date < mbr.join_date
    ORDER BY
        customer_id, s.order_date DESC
)

SELECT
    customer_id,
    product_name
FROM
    cte_customer_order
WHERE
    order_rank = 1;
```

	customer_id	product_name
▶	A	sushi
	A	curry
	B	sushi

Sushi was the most common item a customer ordered before signing up for a membership.

What is the **total items** and **amount spent** for each customer **before** they became a member?

```sql
SELECT
    mbr.customer_id,
    count(s.product_id) as count_of_products,
    sum(m.price) as total_spent
FROM
    members mbr
INNER JOIN
    sales s ON mbr.customer_id = s.customer_id -- Inner join to exclude customers that are not a member
LEFT JOIN
    menu m ON s.product_id = m.product_id
WHERE
    s.order_date < mbr.join_date
GROUP BY
            mbr.customer_id
ORDER BY
    customer_id
    ;
```

	customer_id	count_of_products	total_spent
▶	A	2	25
	B	3	40

Customer B ordered one more item than A, however spent ~50% more.

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

```sql
SELECT
    mbr.customer_id,
    SUM(
        CASE
            WHEN m.product_name = 'sushi' THEN (price * 10) * 2 -- double points for sushi
            ELSE m.price * 10 -- regular points for all other products
        END
    ) AS points_earned
FROM
    members mbr
INNER JOIN
    sales s ON mbr.customer_id = s.customer_id
LEFT JOIN
    menu m ON s.product_id = m.product_id
WHERE
   s.order_date >= mbr.join_date -- We only count purchases on or after join date
GROUP BY
    customer_id
ORDER BY
    customer_id;
```

	customer_id	points_earned
▶	A	510
	B	440

Both members earned a similar number of points. How would this vary based on offering a promo

In the first week after a customer joins the program (including the join date) they earn 2x points on all items – how many points do customer A & B have at the end of January?

```sql
SELECT
    mbr.customer_id,
    SUM(
        CASE
            WHEN DATEDIFF(s.order_date, mbr.join_date) <= 7 THEN  m.price * 20 -- Orders within 7 days of join date get double
            WHEN m.product_name = 'sushi' THEN (price * 10) * 2 -- Sushi is always double points
            ELSE m.price * 10 -- All other purchases get 10 points per dollar
        END
    ) AS points_earned
FROM
    members mbr
INNER JOIN
    sales s ON mbr.customer_id = s.customer_id
LEFT JOIN
    menu m ON s.product_id = m.product_id
WHERE
   s.order_date >= mbr.join_date -- We only count purchases on or after join date
   AND s.order_date < '2021-02-01'-- Promo ends in January
GROUP BY
    customer_id
ORDER BY
    customer_id;
```

	customer_id	points_earned
▶	A	1020
	B	440

Customer A had significantly more points than customer B.

Bonus Question – Join All The Things

Danny has asked for a recreation of a report which his team can use to quickly derive insights without needing to join the underlying tables using SQL.

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

This code was used to generate the requested table. The query could easily be used to create a dynamic sales dashboard in PowerBI or Tableau

```
SELECT
  s.customer_id,
  s.order_date,
  m.product_name,
  m.price,
  CASE
    WHEN s.order_date >= mbr.join_date THEN 'Y'
    ELSE 'N'
  END AS is_member
FROM
  sales s
  LEFT JOIN menu m ON s.product_id = m.product_id
  LEFT JOIN members mbr ON s.customer_id = mbr.customer_id
ORDER BY
  s.customer_id,
  s.order_date;
```

Bonus Question 2

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	is_member	ranking
A	2021-01-01	sushi	10	N	NULL
A	2021-01-01	curry	15	N	NULL
A	2021-01-07	curry	15	Y	1
A	2021-01-10	ramen	12	Y	2
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

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

SELECT
    *,
    CASE
        WHEN is_member = 'Y'
        THEN RANK() OVER (
            PARTITION BY customer_id, is_member
            ORDER BY order_date
        )
    END AS ranking
FROM cte_all_orders;
```

This is not the end, only the beginning

SPECIAL THANKS TO DANNY FROM DATA WITH DANNY FOR PROVIDING THE CASE STUDY QUESTIONS AND DATASET.

THANK YOU TO ALEX THE ANALYST AND HIS SQL EDUCATION PLATFORM ANALYST BUILDER FOR REFRESHING MY KNOWLEDGE AND TEACHING THE ADVANCE FUNCTIONS THROUGH EASY TO DIGEST MODULES.

SPECIAL MENTION TO OPEN AI. CHAT GPT HELPED ME TO DEBUG THE CODE AND GAVE INSIGHTS ON HOW I CAN STRUCTURE MY QUERIES BETTER. ADDITIONALLY, YOU HELPED WITH THE BEAUTIFICATION OF MY CODE FOR READABILITY. THE YUMMY FOOD GRAPHICS IN THE PRESENTATION WERE GENERATED USING THE DALL-E MODULE INCLUDED IN GPT-4.