Customer Behavior Analysis by Shromana Majumder (12.04.24)

Objective

It focuses on examining patterns, trends, and factors influencing customer spending in order to gain insights into their preferences, purchasing habits, and potential areas for improvement in menu offerings or marketing strategies in a dining establishment.

Background

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 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.

Questions Explored

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?

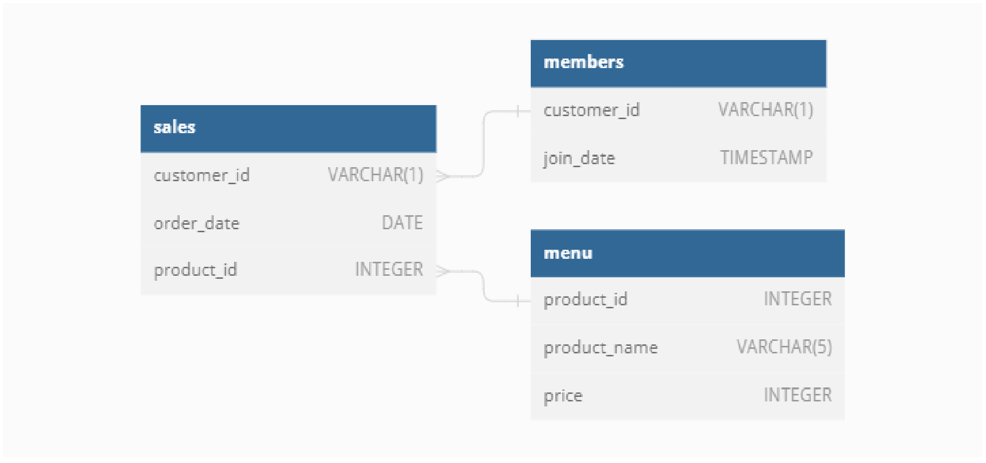
7.Which item was purchased just before the customer became a member?

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?

Entity Relationship Diagram



-- Database

```
CREATE DATABASE dannys_diner;

USE dannys_diner;
```

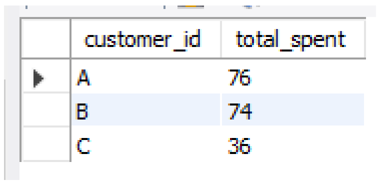
-- Tables

```
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');
```

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

In order to understand the total amount spent by each customer at the restaurant, we need to analyze the sales data. This information is crucial for evaluating customer spending patterns and identifying the most valuable customers. By combining the **sales** and **menu** tables through a join, we can link the sales records with the corresponding menu items. This allows us to calculate the total amount spent by each customer. Once the data is grouped by customer ID, we can apply the **SUM** function to add up the prices associated with their purchases. This will provide us with a comprehensive overview of how much each customer has spent at the restaurant.

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



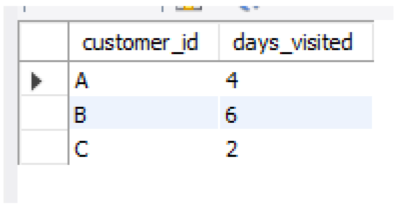
customer_id	total_spent
A	76
B	74
C	36

- **A Spent more at the restaurant.**

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

In order to determine how many days each customer has visited the restaurant, we need to analyze the sales data and count the unique order dates for each customer. By using the **COUNT(DISTINCT order_date)** function, we can count the number of distinct or unique order dates associated with each customer. The **GROUP BY** clause is used to group the sales records by customer ID, ensuring that the count is calculated for each individual customer. This allows us to obtain the number of days each customer has visited the restaurant, providing insights into their frequency of visits.

```
SELECT s.customer_id, COUNT(DISTINCT s.order_date) AS days_visited
FROM sales s
GROUP BY s.customer_id;
```



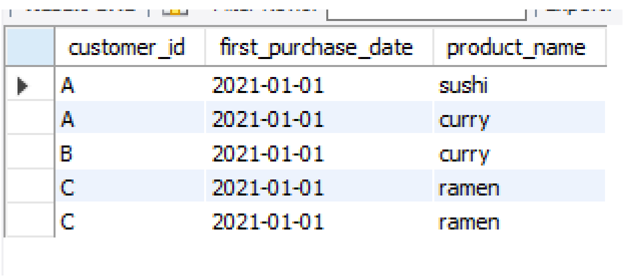
customer_id	days_visited
A	4
B	6
C	2

- **B has visited the restaurant more than A,C**

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

To determine the first item purchased from the menu by each customer, we analyze the sales data and match it with the corresponding menu items. By joining the **sales** and **menu** tables on the **product_id** column, we can retrieve the product name associated with each purchase. The **GROUP BY** clause is used to group the sales records by customer ID, ensuring that the analysis is done for each individual customer. By using the **MIN(s.order_date)** function, we can also identify the date of the first purchase made by each customer. The result is a list that displays the customer ID, the first purchased item from the menu, and the date of the first purchase. The records are ordered by customer ID for easier reference. This information helps us understand the initial preferences and choices made by each customer at the restaurant.

```
WITH customer_first_purchase AS(
        SELECT s.customer_id, MIN(s.order_date) AS first_purchase_date
        FROM sales s
        GROUP BY s.customer_id
)
SELECT cfp.customer_id, cfp.first_purchase_date, m.product_name
FROM customer_first_purchase cfp
INNER JOIN sales s ON s.customer_id = cfp.customer_id
AND cfp.first_purchase_date = s.order_date
INNER JOIN menu m on m.product_id = s.product_id;
```



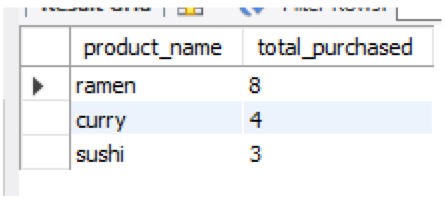
customer_id	first_purchase_date	product_name
A	2021-01-01	sushi
A	2021-01-01	curry
B	2021-01-01	curry
C	2021-01-01	ramen
C	2021-01-01	ramen

- **First item purchased by each customer**

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

To determine the most purchased item on the menu and the number of times it was purchased by all customers, we analyze the sales data and count the occurrences of each product. By joining the **sales** and **menu** tables on the **product_id** column, we can retrieve the product names associated with each purchase. The **ORDER BY** clause sorts the results in descending order based on the count, ensuring that the most purchased item appears at the top. The **LIMIT 3** clause ensures that only the three record is returned, giving us the most purchased item and the number of times it was purchased. This information helps us understand the popularity of different menu items among customers.

```
SELECT  m.product_name, COUNT(s.product_id) AS total_purchased
FROM sales s
INNER JOIN menu m on s.product_id = m.product_id
GROUP BY m.product_name
ORDER BY total_purchased DESC
LIMIT 3;
```

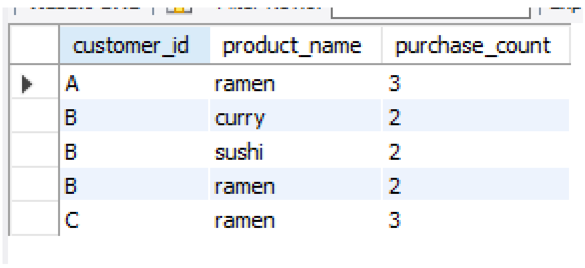


product_name	total_purchased
ramen	8
curry	4
sushi	3

- **Most Purchased item on the menu is Ramen.**

5. **Which item was the most popular for each customer?**
To determine the most popular item for each customer, we analyze the sales data and identify the item with the highest purchase count for each customer. By joining the sales and menu tables on the product_id column, we can retrieve the product names associated with each purchase. This information helps us understand the preferences of each customer and identify the most popular items among them.

```
WITH customer_popularity AS (
   SELECT s.customer_id, m.product_name, COUNT(*) AS purchase_count,
      DENSE_RANK() OVER (PARTITION BY s.customer_id ORDER BY COUNT(*) DESC) AS
popularity_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, purchase_count
FROM customer_popularity
WHERE popularity_rank = 1;
```



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

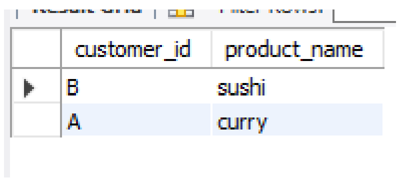
- **Ramen is mostly popular among customers.**

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

To identify the item purchased first by customers after they became a member, we analyze the sales data, membership information, and order dates. By joining the sales, menu, and members tables based on their respective IDs, we can link the customer, product, and membership information together. The WHERE clause filters the sales records to include only orders made after the customer's join date. This information helps us understand the initial preferences of customers after joining the membership program.

```
WITH first_purchase_after_membership AS (
 SELECT s.customer_id, MIN(s.order_date) as first_purchase_date
 FROM sales s
 JOIN members mb ON s.customer_id = mb.customer_id
 WHERE s.order_date >= mb.join_date
 GROUP BY s.customer_id
)
SELECT fpam.customer_id, m.product_name
FROM first_purchase_after_membership fpam
JOIN sales s ON fpam.customer_id = s.customer_id
```

```
AND fpam.first_purchase_date = s.order_date
JOIN menu m ON s.product_id = m.product_id;
```



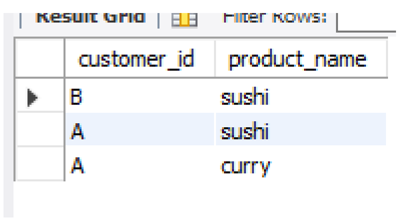
customer_id	product_name
B	sushi
A	curry

- **B First bought sushi and A first bought Curry after beoming members.**

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

   To determine the item purchased just before the customer became a member, we analyze the sales data, membership information, and order dates. By joining the **sales**, **menu**, and **members** tables based on their respective IDs, we can link the customer, product, and membership information together. The **WHERE** clause filters the sales records to include only orders made before the customer's join date. This information helps us understand the preferences and purchasing behavior of customers right before joining the membership program.

```
 WITH last_purchase_before_membership AS (
 SELECT s.customer_id, MAX(s.order_date) AS last_purchase_date
 FROM sales s
 JOIN members mb ON s.customer_id = mb.customer_id
 WHERE s.order_date < mb.join_date
 GROUP BY s.customer_id
)
SELECT lpbm.customer_id, m.product_name
FROM last_purchase_before_membership lpbm
JOIN sales s ON lpbm.customer_id = s.customer_id
AND lpbm.last_purchase_date = s.order_date
JOIN menu m ON s.product_id = m.product_id;
```



Result Grid | Filter Rows:

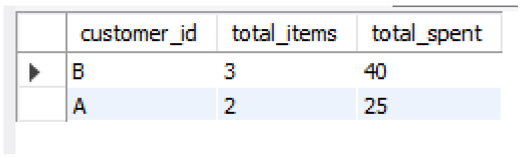
customer_id	product_name
B	sushi
A	sushi
A	curry

- **Item Purchased before membership.**

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

To calculate the total number of items and the total amount spent by each member before they became a member, we analyze the sales data, membership information, and order dates. The query joins the **sales**, **members**, and **menu** tables based on their respective IDs. Additionally, **SUM(m.price)** is used to calculate the total amount spent for each member by summing up the prices from the joined **menu** table. This information provides insights into the purchasing behavior and spending patterns of members before they joined the loyalty program.

```
SELECT s.customer_id, COUNT(*) as total_items, SUM(m.price) AS total_spent
FROM sales s
JOIN menu m ON s.product_id = m.product_id
JOIN members mb ON s.customer_id = mb.customer_id
WHERE s.order_date < mb.join_date
GROUP BY s.customer_id;
```



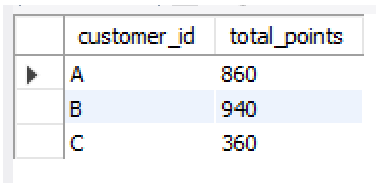
	customer_id	total_items	total_spent
▶	B	3	40
	A	2	25

- **B bought more items and spent more before membership.**

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

To calculate the number of points each customer would have based on their spending and the points multiplier, we analyze the sales data and join the **sales** and **menu** tables based on their respective IDs. The query groups the data by customer ID using **GROUP BY**, ensuring that we calculate the points for each individual customer. This information helps us understand the number of points accumulated by each customer based on their spending, taking into account the 2x points multiplier for sushi.

```
SELECT s.customer_id, SUM(
        CASE
                WHEN m.product_name = 'sushi' THEN m.price*20
                ELSE m.price*10 END) AS total_points
FROM sales s
JOIN menu m ON s.product_id = m.product_id
GROUP BY s.customer_id;
```



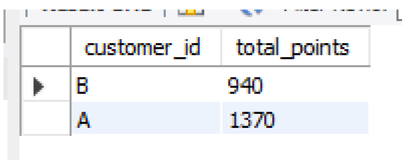
	customer_id	total_points
▶	A	860
	B	940
	C	360

- **B has highest point.**

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?

To calculate the number of points for customer A and customer B at the end of January, taking into account the 2x points multiplier for the first week after joining, we analyze the sales data, membership information, and order dates. The query joins the **sales**, **menu**, and **members** tables based on their respective IDs. The **WHERE** clause filters the sales records to include only orders made in January 2021, specifically on the first day of January using **DATE_FORMAT**. This information helps us understand the number of points earned by each customer at the end of January, accounting for the 2x points multiplier during the first week after joining the loyalty program.

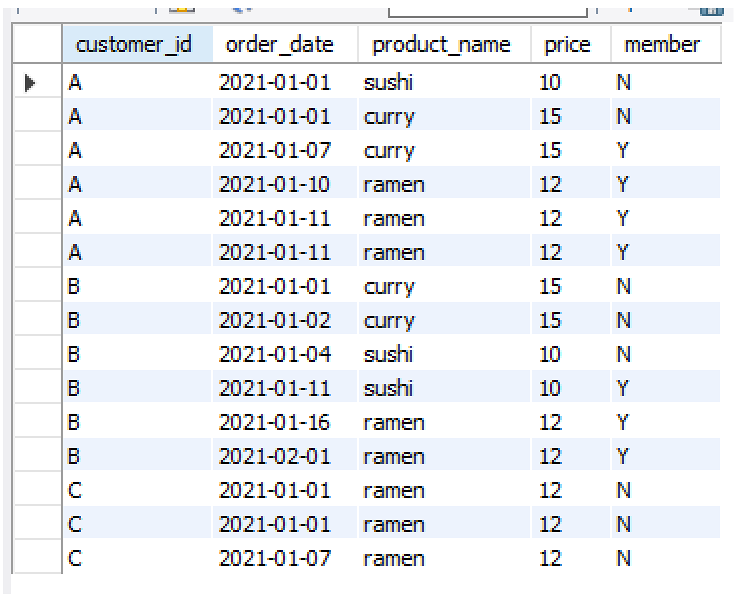
```
SELECT s.customer_id, SUM(
   CASE
      WHEN s.order_date BETWEEN mb.join_date AND DATE_ADD(mb.join_date, INTERVAL 7
DAY) THEN m.price*20
      WHEN m.product_name = 'sushi' THEN m.price*20
      ELSE m.price*10
   END) AS total_points
FROM sales s
JOIN menu m ON s.product_id = m.product_id
LEFT JOIN members mb ON s.customer_id = mb.customer_id
WHERE s.customer_id IN ('A', 'B') AND s.order_date <= '2021-01-31'
GROUP BY s.customer_id;
```



customer_id	total_points
B	940
A	1370

11. Recreate the table output using the available data

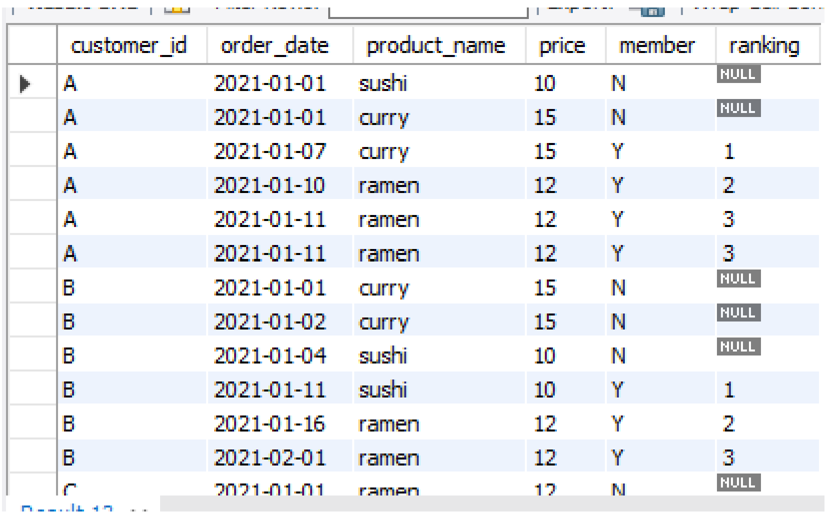
```
SELECT s.customer_id, s.order_date, m.product_name, m.price,
CASE WHEN s.order_date >= mb.join_date THEN 'Y'
ELSE 'N' END AS member
FROM sales s
JOIN menu m ON s.product_id = m.product_id
LEFT JOIN members mb ON s.customer_id = mb.customer_id
ORDER BY s.customer_id, s.order_date;
```



customer_id	order_date	product_name	price	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

12. Rank all the things

```
WITH customers_data AS (
        SELECT s.customer_id, s.order_date, m.product_name, m.price,
        CASE
                WHEN s.order_date < mb.join_date THEN 'N'
                WHEN s.order_date >= mb.join_date THEN 'Y'
                ELSE 'N' END AS member
        FROM sales s
        LEFT JOIN members mb ON s.customer_id = mb.customer_id
        JOIN menu m ON s.product_id = m.product_id
)
SELECT *,
CASE WHEN member = 'N' THEN NULL
ELSE RANK() OVER(PARTITION BY customer_id, member ORDER BY order_date)
END AS ranking
FROM customers_data
ORDER BY customer_id, order_date;
```



customer_id	order_date	product_name	price	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

Insights:

- Customer B is the most frequent visitor .
  •Danny's Diner's most popular item is ramen, followed by curry and sushi.
  •Customer A loves ramen, Customer C loves only ramen whereas Customer B seems to enjoy sushi, curry and ramen equally.
  •The last item ordered by Customers A and B before they became members are sushi and curry.