🍜 Case Study #1: Danny's Diner

📚 Table of Contents

📌 Introduction

In early 2021, Danny opened a small restaurant called **Danny's Diner**, featuring his favorite Japanese dishes: sushi, curry, and ramen. To help the restaurant grow, Danny collected customer data but needed help to analyze it effectively. This case study aims to help Danny derive insights from the data, enhancing customer experience and supporting business decisions.

Note: All the information regarding the case study has been sourced from the following link: here. For executing the queries, I used PostgreSQL on DB Fiddle.

🔍 Problem Statement

Danny wants to answer several questions about his customers, including:

- How often do customers visit the restaurant?
- How much have they spent?
- Which menu items are their favorites?

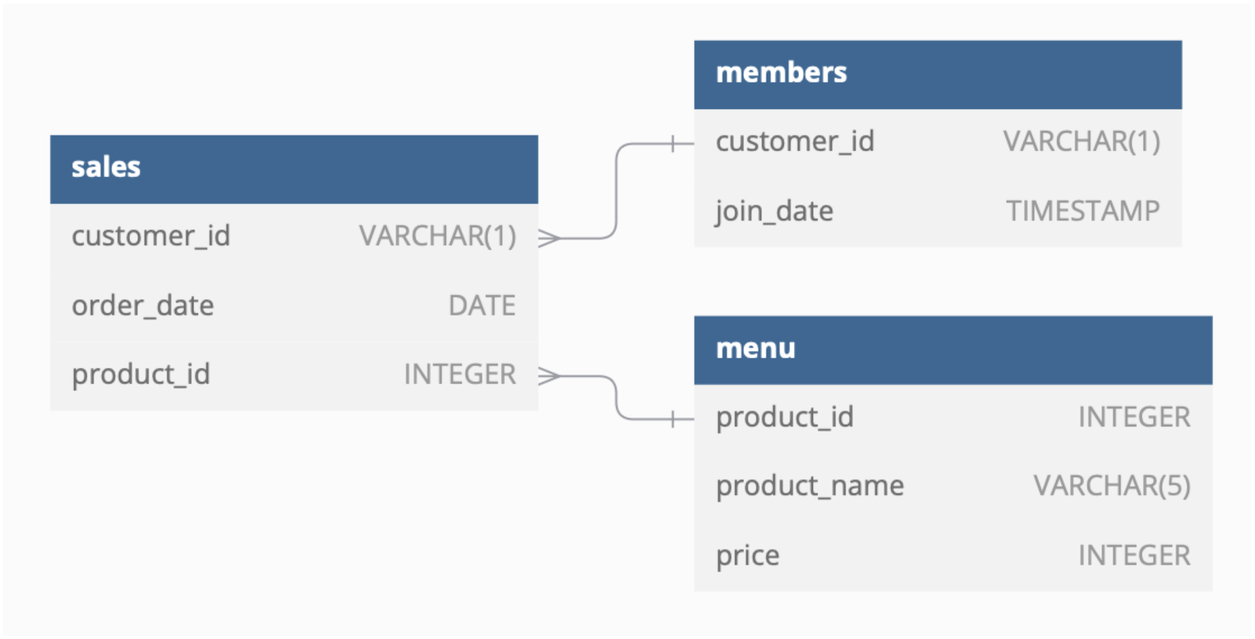
By understanding customer behavior, Danny aims to decide on expanding the customer loyalty program. Danny provided three datasets for this purpose: **sales**, **menu**, and **members**.

📊 Data Overview & Entity Relationship Diagram

Data Sources:

- **Sales**: Captures customer purchases with `customer_id`, `order_date`, and `product_id`.
- **Menu**: Maps `product_id` to the menu items, including `product_name` and `price`.
- **Members**: Shows when a `customer_id` joined the loyalty program (`join_date`).

Entity Relationship Diagram: Illustrates the relationships between sales, menu, and membership data. The diagram shows the linkages between `customer_id` and `product_id` among these tables, forming the backbone for my analysis.



❓ Case Study Questions & Analysis

1. Total Sales Per Customer

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

Steps:

- Combine the `dannys_diner.sales` and `dannys_diner.menu` tables using an `INNER JOIN`, linking them through the `product_id` column to associate customer purchases with their corresponding prices.
- Use the `SUM` function to compute the total amount spent by each customer based on the `price` column from the `menu` table.
- Group the aggregated totals by `customer_id` to display the total sales per customer.

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

Answer: Customer A spent $76, Customer B spent $74, and Customer C spent $36.

Customer ID	Total Spent
A	$76
B	$74
C	$36

2. Number of Visits Per Customer

How many days has each customer visited the restaurant?

Steps:

- Use the `COUNT` function with the `DISTINCT` keyword on the `order_date` column to calculate the unique number of visit days for each customer.
- Group the results by `customer_id` to display the unique visit count for each customer.

```
SELECT
    customer_id,
    COUNT(DISTINCT(order_date)) AS total_days
FROM sales
GROUP BY customer_id
```

Answer: Customer A visited 4 times, Customer B visited 6 times, and Customer C visited 2 times.

Customer ID	Total Days
A	4
B	6
C	2

3. First Menu Item Purchased

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

Steps:

- Define a Common Table Expression (CTE) named `ordered_sales_cte` to organize the query. Within the CTE, create a new column `rank` using the `DENSE_RANK()` window function. Use the `PARTITION BY` clause to divide the data by `customer_id` and the `ORDER BY` clause to sort rows within each partition by `order_date`.
- In the main query, select the required columns and filter the results using a `WHERE` clause to include only rows where the `rank` equals 1, representing the first order for each `customer_id`.
- Group the final output by `customer_id` and `product_name` to display each customer's first ordered products.

```
WITH rank_date AS(
    SELECT
    sales.customer_id,
    sales.order_date,
    menu.product_name,
    DENSE_RANK() OVER (
      PARTITION BY sales.customer_id
      ORDER BY sales.order_date) AS rank
    FROM dannys_diner.sales
    INNER JOIN dannys_diner.menu
    ON sales.product_id = menu.product_id
)

SELECT
    customer_id,
    product_name
FROM rank_date
WHERE rank = 1
GROUP BY customer_id,
    product_name
```

Answer: Customer A's first order included both sushi and curry, while B's and C's first orders were curry and ramen, respectively.

Customer ID	First Product Purchased
A	Curry
A	Sushi
B	Curry
C	Ramen

---

4. Most Purchased Items on the Menu

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

Steps:

- Join the `sales` table with the `menu` table on `product_id` to associate each sale with its product details.
- Count how many times each `product_name` appears in the sales data using `COUNT` and group the results by `product_name`.
- Sort the aggregated results by `most_ordered` in descending order to rank products by popularity.
- Use `LIMIT 1` to retrieve the most frequently ordered product.

```
SELECT
    m.product_name,
    COUNT(m.product_name) most_ordered
FROM sales AS s
INNER JOIN menu AS m
ON s.product_id = m.product_id
GROUP BY m.product_name
ORDER BY most_ordered DESC
LIMIT 1;
```

Answer: Ramen was the most popular item, purchased 8 times.

Product Name	Orders Count
Ramen	8

5. Most Popular Item Per Customer

What is the most popular item ordered by each customer?

Steps:

- Create a CTE named `fav_item_cte` to join the `menu` and `sales` tables on `product_id`.
- Group results by `customer_id` and `product_name`, counting the total orders for each combination.

- Use `DENSE_RANK()` to rank items within each customer's group based on order count in descending order.
- Filter for rows where the rank equals 1 to identify the most popular items for each customer.

```
WITH fav_item_cte AS (
  SELECT
    s.customer_id,
    m.product_name,
    COUNT(s.product_id) AS order_count,
    DENSE_RANK() OVER (
      PARTITION BY s.customer_id
      ORDER BY COUNT(s.product_id) DESC
    ) AS rank
  FROM sales AS s
  INNER JOIN menu AS m
    ON s.product_id = m.product_id
  GROUP BY s.customer_id, m.product_name
)
SELECT
  customer_id,
  product_name,
  order_count
FROM fav_item_cte
WHERE rank = 1;
```

Answer: Ramen is the favorite item of Customers A and C, while Customer B enjoys all menu items equally.

customer_id	product_name	order_count
A	ramen	3
B	sushi	2
C	ramen	3

6. Items Purchased After Becoming a Member

What is the first item ordered by each customer after becoming a member?

Steps:

- Create a CTE `joined_as_member` to join `members` and `sales` on `customer_id`, filtering for orders placed after the membership start date.
- Use `ROW_NUMBER()` to rank orders by date within each customer group.
- Join the CTE with the `menu` table to get product details and filter for the first-ranked order.

```
WITH joined_as_member AS (
  SELECT
    m.customer_id,
    s.product_id,
    ROW_NUMBER() OVER (
      PARTITION BY m.customer_id
      ORDER BY s.order_date
    ) AS row_num
  FROM members AS m
  INNER JOIN sales AS s
    ON m.customer_id = s.customer_id
    AND s.order_date > m.join_date
)
SELECT
  jm.customer_id,
  menu.product_name
FROM joined_as_member AS jm
INNER JOIN menu
  ON jm.product_id = menu.product_id
WHERE row_num = 1
ORDER BY jm.customer_id;
```

Answer: After becoming members customer A and Customer B first purchases were ramen and sushi respectively.

customer_id	product_name
A	ramen
B	sushi

---

7. Last Item Ordered Before Membership

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

Steps:

- Create a CTE `purchased_prior_member` to join `members` and `sales` on `customer_id`, filtering for orders placed before the membership start date.
- Use `ROW_NUMBER()` to rank orders by date in descending order within each customer group.
- Join the CTE with the `menu` table to get product details and filter for the first-ranked order.

```
WITH purchased_prior_member AS (
  SELECT
    m.customer_id,
    s.product_id,
    ROW_NUMBER() OVER (
      PARTITION BY m.customer_id
      ORDER BY s.order_date DESC
    ) AS rank
  FROM members AS m
  INNER JOIN sales AS s
    ON m.customer_id = s.customer_id
    AND s.order_date < m.join_date
)
SELECT
  ppm.customer_id,
  menu.product_name
FROM purchased_prior_member AS ppm
INNER JOIN menu
  ON ppm.product_id = menu.product_id
```

```
WHERE rank = 1
ORDER BY ppm.customer_id;
```

Answer: Before becoming members, both Customer A and Customer B's last order was sushi.

customer_id	product_name
A	sushi
B	sushi

---

8. Total Items and Amount Spent Before Membership

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

Steps:

- Join the `sales` table with the `members` table on `customer_id`, ensuring the `order_date` is earlier than the membership start date.
- Join the resulting table with the `menu` table to retrieve product prices.
- Calculate the total number of items purchased (`COUNT`) and the total amount spent (`SUM`) for each customer.
- Group results by `customer_id` to display totals.

```
SELECT
  s.customer_id,
  COUNT(s.product_id) AS total_items,
  SUM(m.price) AS total_spent
FROM sales AS s
INNER JOIN members AS mem
  ON s.customer_id = mem.customer_id
  AND s.order_date < mem.join_date
INNER JOIN menu AS m
  ON s.product_id = m.product_id
```

```
GROUP BY s.customer_id
ORDER BY s.customer_id;
```

Answer: Before becoming a member, Customer A spent $25 on 2 items, while Customer B spent $40 on 3 items.

customer_id	total_items	total_spent
A	2	25
B	3	40

---

9. Points Earned by Each Customer

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

Steps:

- Create a CTE `points_cte` to calculate points for each product using a `CASE` statement:
  - Multiply the price by 20 points for sushi.
  - Multiply the price by 10 points for all other items.
- Join the `sales` table with `points_cte` on `product_id` to calculate total points.
- Group results by `customer_id`.

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

Answer: Customer A earned 860 points, Customer B earned 940 points, and Customer C earned 360 points.

customer_id	total_points
A	860
B	940
C	360

10. Points Earned in January Post-Membership

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?

Steps:

- Create a CTE `dates_cte` to calculate:
  - The valid period for double points (7 days from `join_date`).
  - The last day of January.
- Join `sales` with `dates_cte` and `menu` to calculate points using a `CASE` statement:
  - Multiply the price by 20 points for sushi.
  - Multiply the price by 20 points for all items during the double points period.
  - Multiply the price by 10 points for all other items.
- Group results by `customer_id`.

```
WITH dates_cte AS (
  SELECT
    customer_id,
    join_date,
    join_date + INTERVAL '6 days' AS valid_date,
    '2021-01-31'::DATE AS last_date
  FROM members
)
SELECT
  s.customer_id,
  SUM(CASE
    WHEN s.order_date BETWEEN d.join_date AND d.valid_date TH
EN m.price * 20
    WHEN m.product_name = 'sushi' THEN m.price * 20
    ELSE m.price * 10
  END) AS total_points
FROM sales AS s
INNER JOIN dates_cte AS d
  ON s.customer_id = d.customer_id
  AND s.order_date <= d.last_date
INNER JOIN menu AS m
  ON s.product_id = m.product_id
GROUP BY s.customer_id;
```

Answer: At the end of January, Customer A earned 1,020 points, and Customer B earned 320 points.

customer_id	total_points
A	1020
B	320

🛠️ Tools Used

- **Languages**: SQL (PostgreSQL dialect)

- **Interactive Environment**: DB Fiddle for running SQL queries
- **Visualization Tools**: Matplotlib/Tableau for creating insights charts (optional)
- **Database Management**: PostgreSQL for structured queries and analysis

---

🌍 Additional Links

- GitHub Repository: Access the complete codebase and SQL queries.
- LinkedIn: Connect with me for further discussion or opportunities.
