## **Problem Statement**

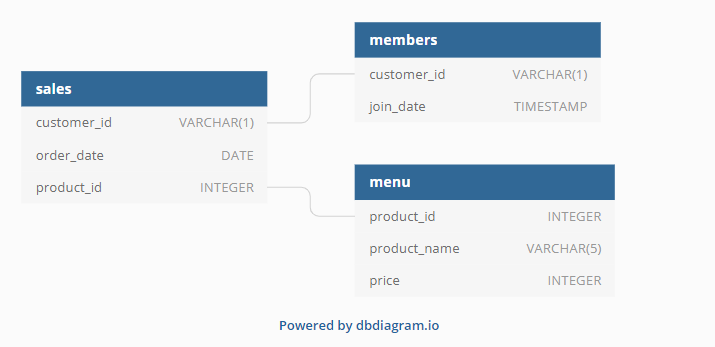
Danny wants to use the data to answer a few simple questions about his customers, especially about their visiting patterns, how much money they’ve spent and also which menu items are their favourite. Having this deeper connection with his customers will help him deliver a better and more personalised 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 shared with you 3 key datasets for this case study:

* sales
* menu
* Members

## **Entity Relationship Diagram**



## **Questions and Solutions**

Executing the queries using PostgreSQL on [DB Fiddle](https://www.db-fiddle.com/f/2rM8RAnq7h5LLDTzZiRWcd/138). It would be great to work together on the questions!

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

SELECT

sales.customer\_id,

SUM(menu.price) AS total\_salesFROM dannys\_diner.salesINNER JOIN dannys\_diner.menu

ON sales.product\_id = menu.product\_idGROUP BY sales.customer\_idORDER BY sales.customer\_id ASC;

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps)

* Use **JOIN** to merge dannys\_diner.sales and dannys\_diner.menu tables as sales.customer\_id and menu.price are from both tables.
* Use **SUM** to calculate the total sales contributed by each customer.
* Group the aggregated results by sales.customer\_id.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer)

| **customer\_id** | **total\_sales** |
| --- | --- |
| A | 76 |
| B | 74 |
| C | 36 |

* Customer A spent $76.
* Customer B spent $74.
* Customer C spent $36.

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

SELECT

customer\_id,

COUNT(DISTINCT order\_date) AS visit\_countFROM dannys\_diner.salesGROUP BY customer\_id;

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps-1)

* To determine the unique number of visits for each customer, utilize **COUNT(DISTINCT**order\_date**)**.
* It's important to apply the **DISTINCT** keyword while calculating the visit count to avoid duplicate counting of days. For instance, if Customer A visited the restaurant twice on '2021–01–07', counting without **DISTINCT** would result in 2 days instead of the accurate count of 1 day.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer-1)

| **customer\_id** | **visit\_count** |
| --- | --- |
| A | 4 |
| B | 6 |
| C | 2 |

* Customer A visited 4 times.
* Customer B visited 6 times.
* Customer C visited 2 times.

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

WITH ordered\_sales 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\_nameFROM ordered\_salesWHERE rank = 1GROUP BY customer\_id, product\_name;

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps-2)

* Create a Common Table Expression (CTE) named ordered\_sales\_cte. Within the CTE, create a new column rank and calculate the row number using **DENSE\_RANK()** window function. The **PARTITION BY** clause divides the data by customer\_id, and the **ORDER BY** clause orders the rows within each partition by order\_date.
* In the outer query, select the appropriate columns and apply a filter in the **WHERE** clause to retrieve only the rows where the rank column equals 1, which represents the first row within each customer\_id partition.
* Use the GROUP BY clause to group the result by customer\_id and product\_name.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer-2)

| **customer\_id** | **product\_name** |
| --- | --- |
| A | curry |
| A | sushi |
| B | curry |
| C | ramen |

* Customer A placed an order for both curry and sushi simultaneously, making them the first items in the order.
* Customer B's first order is curry.
* Customer C's first order is ramen.

I have received feedback suggesting the use of ROW\_NUMBER() instead of DENSE\_RANK() for determining the "first order" in this question.

However, since the order\_date does not have a timestamp, it is impossible to determine the exact sequence of items ordered by the customer.

Therefore, it would be inaccurate to conclude that curry is the customer's first order purely based on the alphabetical order of the product names. For this reason, I maintain my solution of using DENSE\_RANK() and consider both curry and sushi as Customer A's first order.

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

SELECT

menu.product\_name,

COUNT(sales.product\_id) AS most\_purchased\_itemFROM dannys\_diner.salesINNER JOIN dannys\_diner.menu

ON sales.product\_id = menu.product\_idGROUP BY menu.product\_nameORDER BY most\_purchased\_item DESCLIMIT 1;

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps-3)

* Perform a **COUNT** aggregation on the product\_id column and **ORDER BY** the result in descending order using most\_purchased field.
* Apply the **LIMIT** 1 clause to filter and retrieve the highest number of purchased items.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer-3)

| **most\_purchased** | **product\_name** |
| --- | --- |
| 8 | ramen |

* Most purchased item on the menu is ramen which is 8 times. Yummy!

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

WITH most\_popular AS (

SELECT

sales.customer\_id,

menu.product\_name,

COUNT(menu.product\_id) AS order\_count,

DENSE\_RANK() OVER (

PARTITION BY sales.customer\_id

ORDER BY COUNT(sales.customer\_id) DESC) AS rank

FROM dannys\_diner.menu

INNER JOIN dannys\_diner.sales

ON menu.product\_id = sales.product\_id

GROUP BY sales.customer\_id, menu.product\_name

)

SELECT

customer\_id,

product\_name,

order\_countFROM most\_popular WHERE rank = 1;

Each user may have more than 1 favourite item.

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps-4)

* Create a CTE named fav\_item\_cte and within the CTE, join the menu table and sales table using the product\_id column.
* Group results by sales.customer\_id and menu.product\_name and calculate the count of menu.product\_id occurrences for each group.
* Utilize the **DENSE\_RANK()** window function to calculate the ranking of each sales.customer\_id partition based on the count of orders **COUNT(**sales.customer\_id**)** in descending order.
* In the outer query, select the appropriate columns and apply a filter in the **WHERE** clause to retrieve only the rows where the rank column equals 1, representing the rows with the highest order count for each customer.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer-4)

| **customer\_id** | **product\_name** | **order\_count** |
| --- | --- | --- |
| A | ramen | 3 |
| B | sushi | 2 |
| B | curry | 2 |
| B | ramen | 2 |
| C | ramen | 3 |

* Customer A and C's favourite item is ramen.
* Customer B enjoys all items on the menu. He/she is a true foodie, sounds like me.

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

WITH joined\_as\_member AS (

SELECT

members.customer\_id,

sales.product\_id,

ROW\_NUMBER() OVER (

PARTITION BY members.customer\_id

ORDER BY sales.order\_date) AS row\_num

FROM dannys\_diner.members

INNER JOIN dannys\_diner.sales

ON members.customer\_id = sales.customer\_id

AND sales.order\_date > members.join\_date

)

SELECT

customer\_id,

product\_name FROM joined\_as\_memberINNER JOIN dannys\_diner.menu

ON joined\_as\_member.product\_id = menu.product\_idWHERE row\_num = 1ORDER BY customer\_id ASC;

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps-5)

* Create a CTE named joined\_as\_member and within the CTE, select the appropriate columns and calculate the row number using the **ROW\_NUMBER()** window function. The **PARTITION BY** clause divides the data by members.customer\_id and the **ORDER BY** clause orders the rows within each members.customer\_id partition by sales.order\_date.
* Join tables dannys\_diner.members and dannys\_diner.sales on customer\_id column. Additionally, apply a condition to only include sales that occurred after the member's join\_date (sales.order\_date > members.join\_date).
* In the outer query, join the joined\_as\_member CTE with the dannys\_diner.menu on the product\_id column.
* In the **WHERE** clause, filter to retrieve only the rows where the row\_num column equals 1, representing the first row within each customer\_id partition.
* Order result by customer\_id in ascending order.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer-5)

| **customer\_id** | **product\_name** |
| --- | --- |
| A | ramen |
| B | sushi |

* Customer A's first order as a member is ramen.
* Customer B's first order as a member is sushi.

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

WITH purchased\_prior\_member AS (

SELECT

members.customer\_id,

sales.product\_id,

ROW\_NUMBER() OVER (

PARTITION BY members.customer\_id

ORDER BY sales.order\_date DESC) AS rank

FROM dannys\_diner.members

INNER JOIN dannys\_diner.sales

ON members.customer\_id = sales.customer\_id

AND sales.order\_date < members.join\_date

)

SELECT

p\_member.customer\_id,

menu.product\_name FROM purchased\_prior\_member AS p\_memberINNER JOIN dannys\_diner.menu

ON p\_member.product\_id = menu.product\_idWHERE rank = 1ORDER BY p\_member.customer\_id ASC;

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps-6)

* Create a CTE called purchased\_prior\_member.
* In the CTE, select the appropriate columns and calculate the rank using the **ROW\_NUMBER()** window function. The rank is determined based on the order dates of the sales in descending order within each customer's group.
* Join dannys\_diner.members table with dannys\_diner.sales table based on the customer\_id column, only including sales that occurred before the customer joined as a member (sales.order\_date < members.join\_date).
* Join purchased\_prior\_member CTE with dannys\_diner.menu table based on product\_id column.
* Filter the result set to include only the rows where the rank is 1, representing the earliest purchase made by each customer before they became a member.
* Sort the result by customer\_id in ascending order.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer-6)

| **customer\_id** | **product\_name** |
| --- | --- |
| A | sushi |
| B | sushi |

* Both customers' last order before becoming members are sushi.

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

SELECT

sales.customer\_id,

COUNT(sales.product\_id) AS total\_items,

SUM(menu.price) AS total\_salesFROM dannys\_diner.salesINNER JOIN dannys\_diner.members

ON sales.customer\_id = members.customer\_id

AND sales.order\_date < members.join\_dateINNER JOIN dannys\_diner.menu

ON sales.product\_id = menu.product\_idGROUP BY sales.customer\_idORDER BY sales.customer\_id;

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps-7)

* Select the columns sales.customer\_id and calculate the count of sales.product\_id as total\_items for each customer and the sum of menu.price as total\_sales.
* From dannys\_diner.sales table, join dannys\_diner.members table on customer\_id column, ensuring that sales.order\_date is earlier than members.join\_date (sales.order\_date < members.join\_date).
* Then, join dannys\_diner.menu table to dannys\_diner.sales table on product\_id column.
* Group the results by sales.customer\_id.
* Order the result by sales.customer\_id in ascending order.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer-7)

| **customer\_id** | **total\_items** | **total\_sales** |
| --- | --- | --- |
| A | 2 | 25 |
| B | 3 | 40 |

Before becoming members,

* Customer A spent $25 on 2 items.
* Customer B spent $40 on 3 items.

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

WITH points\_cte AS (

SELECT

menu.product\_id,

CASE

WHEN product\_id = 1 THEN price \* 20

ELSE price \* 10 END AS points

FROM dannys\_diner.menu

)

SELECT

sales.customer\_id,

SUM(points\_cte.points) AS total\_pointsFROM dannys\_diner.salesINNER JOIN points\_cte

ON sales.product\_id = points\_cte.product\_idGROUP BY sales.customer\_idORDER BY sales.customer\_id;

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps-8)

Let's break down the question to understand the point calculation for each customer's purchases.

* Each $1 spent = 10 points. However, product\_id 1 sushi gets 2x points, so each $1 spent = 20 points.
* Here's how the calculation is performed using a conditional CASE statement:
  + If product\_id = 1, multiply every $1 by 20 points.
  + Otherwise, multiply $1 by 10 points.
* Then, calculate the total points for each customer.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer-8)

| **customer\_id** | **total\_points** |
| --- | --- |
| A | 860 |
| B | 940 |
| C | 360 |

* Total points for Customer A is $860.
* Total points for Customer B is $940.
* Total points for Customer C is $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 customer A and B have at the end of January?**

WITH dates\_cte AS (

SELECT

customer\_id,

join\_date,

join\_date + 6 AS valid\_date,

DATE\_TRUNC(

'month', '2021-01-31'::DATE)

+ interval '1 month'

- interval '1 day' AS last\_date

FROM dannys\_diner.members

)

SELECT

sales.customer\_id,

SUM(CASE

WHEN menu.product\_name = 'sushi' THEN 2 \* 10 \* menu.price

WHEN sales.order\_date BETWEEN dates.join\_date AND dates.valid\_date THEN 2 \* 10 \* menu.price

ELSE 10 \* menu.price END) AS pointsFROM dannys\_diner.salesINNER JOIN dates\_cte AS dates

ON sales.customer\_id = dates.customer\_id

AND dates.join\_date <= sales.order\_date

AND sales.order\_date <= dates.last\_dateINNER JOIN dannys\_diner.menu

ON sales.product\_id = menu.product\_idGROUP BY sales.customer\_id;

#### [Assumptions:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "assumptions)

* On Day -X to Day 1 (the day a customer becomes a member), each $1 spent earns 10 points. However, for sushi, each $1 spent earns 20 points.
* From Day 1 to Day 7 (the first week of membership), each $1 spent for any items earns 20 points.
* From Day 8 to the last day of January 2021, each $1 spent earns 10 points. However, sushi continues to earn double the points at 20 points per $1 spent.

#### [Steps:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "steps-9)

* Create a CTE called dates\_cte.
* In dates\_cte, calculate the valid\_date by adding 6 days to the join\_date and determine the last\_date of the month by subtracting 1 day from the last day of January 2021.
* From dannys\_diner.sales table, join dates\_cte on customer\_id column, ensuring that the order\_date of the sale is after the join\_date (dates.join\_date <= sales.order\_date) and not later than the last\_date (sales.order\_date <= dates.last\_date).
* Then, join dannys\_diner.menu table based on the product\_id column.
* In the outer query, calculate the points by using a CASE statement to determine the points based on our assumptions above.
  + If the product\_name is 'sushi', multiply the price by 2 and then by 10. For orders placed between join\_date and valid\_date, also multiply the price by 2 and then by 10.
  + For all other products, multiply the price by 10.
* Calculate the sum of points for each customer.

#### [Answer:](https://github.com/katiehuangx/8-Week-SQL-Challenge/blob/main/Case Study %231 - Danny's Diner/README.md" \l "answer-9)

| **customer\_id** | **total\_points** |
| --- | --- |
| A | 1020 |
| B | 320 |

* Total points for Customer A is 1,020.
* Total points for Customer B is 320.