Danny Ma’s 8 Week SQL Challenge

By Tamara Gray

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 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 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.

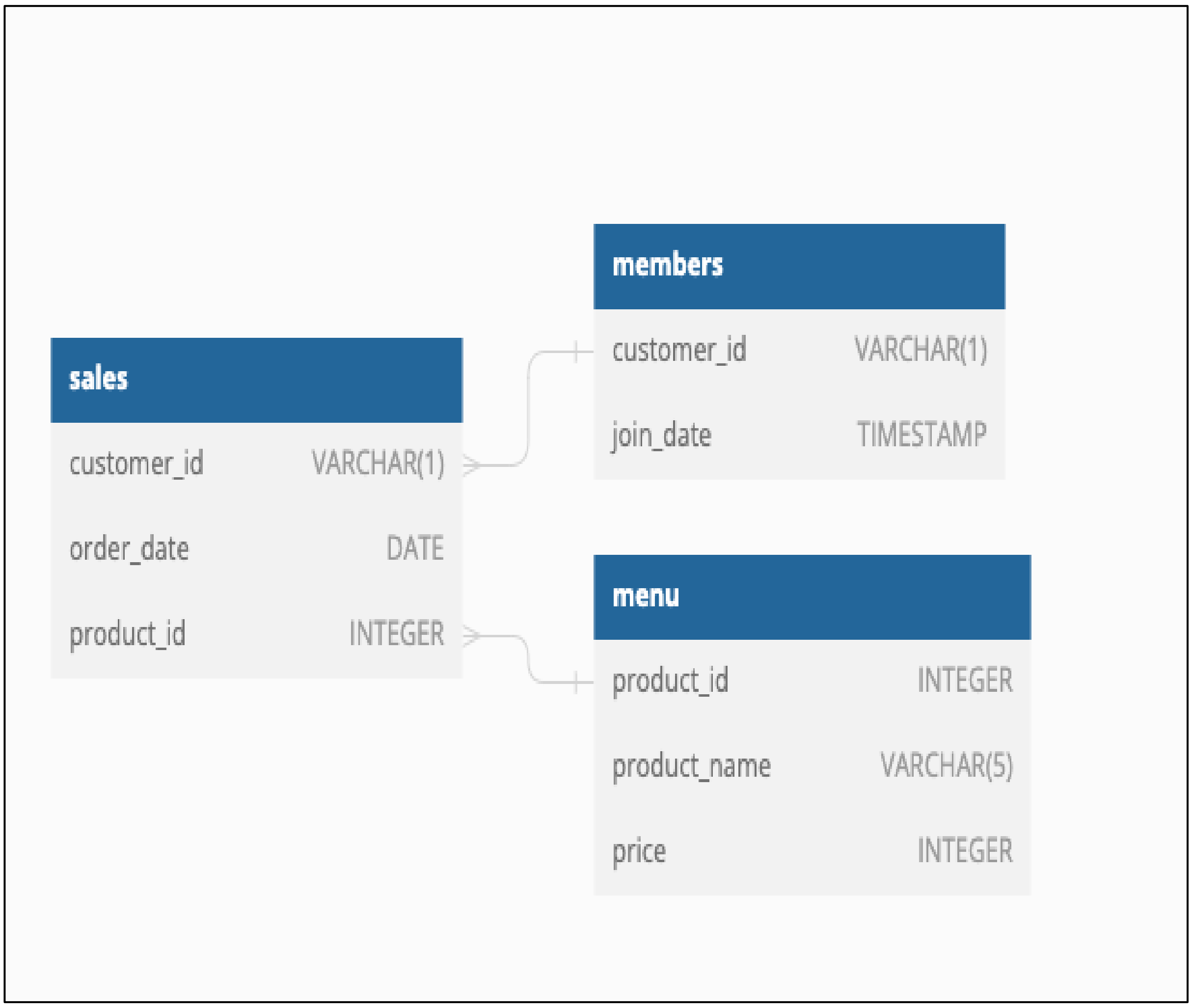
He plans on using these insights to help him decide whether he should expand the existing customer loyalty program. Additionally he needs help to generate some basic datasets so his team can easily inspect the data without needing to use SQL.

Danny has 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!

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

- sales
- menu
- members

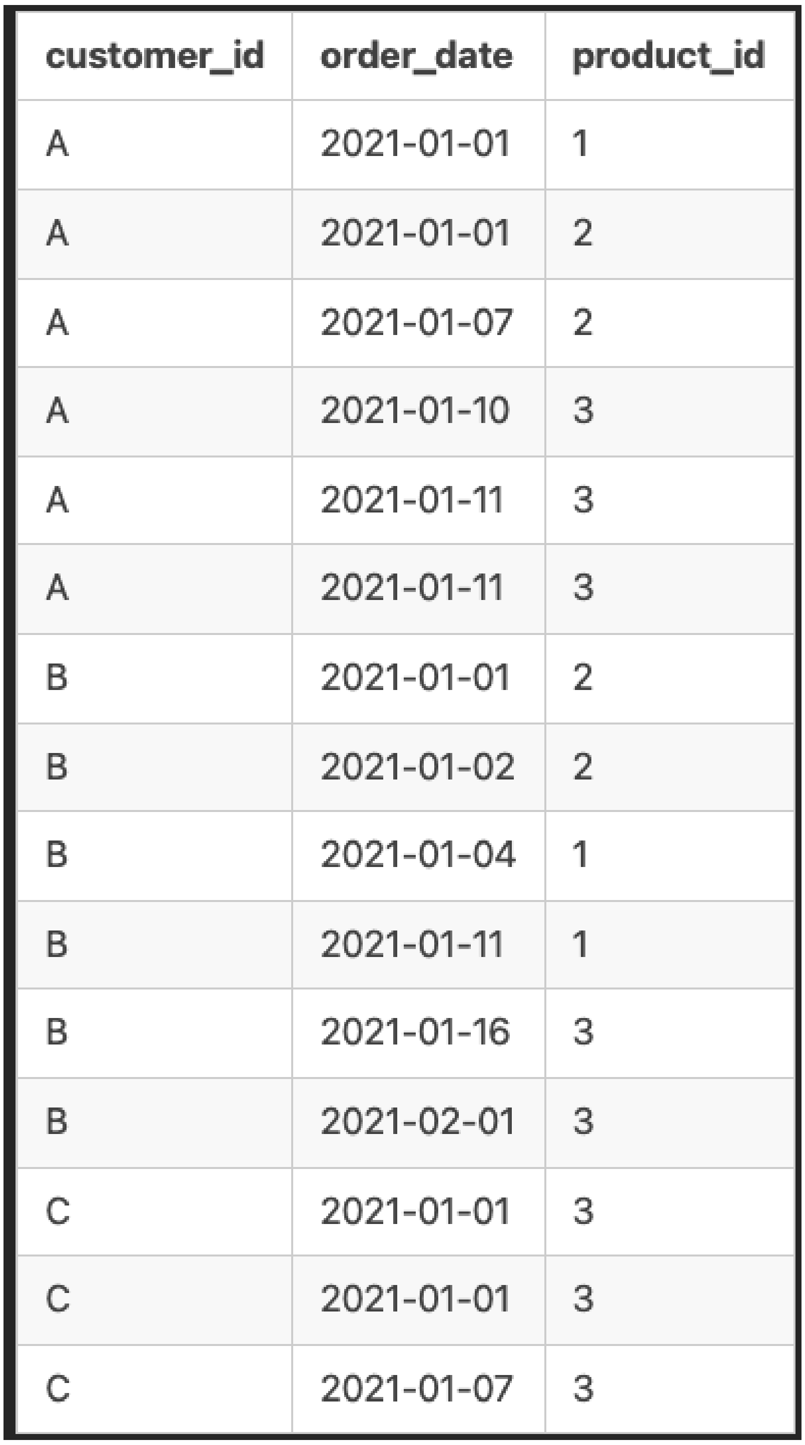
Entity Relationship Diagram



Datasets

Table 1: sales

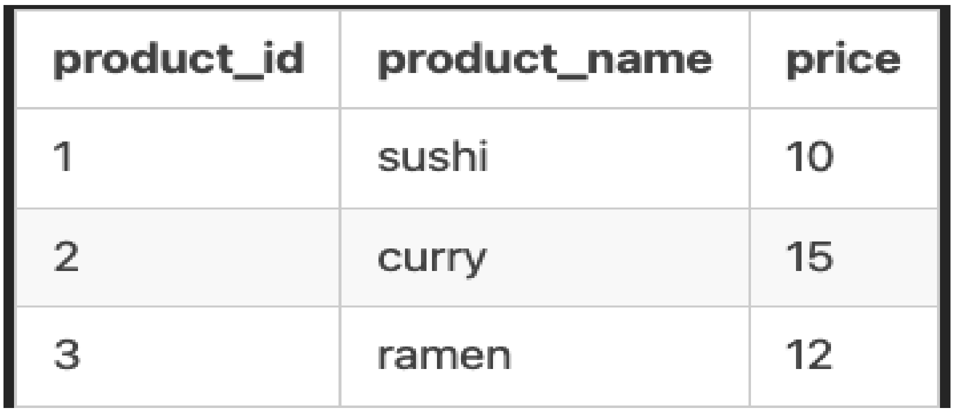
The sales table captures all ***customer_id*** level purchases with an corresponding ***order_date*** and ***product_id*** information for when and what menu items were ordered.



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

Table 2: menu

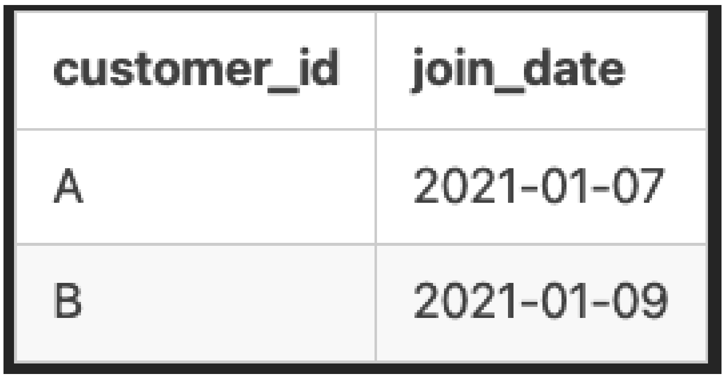
The menu table maps the ***product_id*** to the actual ***product_name*** and price of each menu item.



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

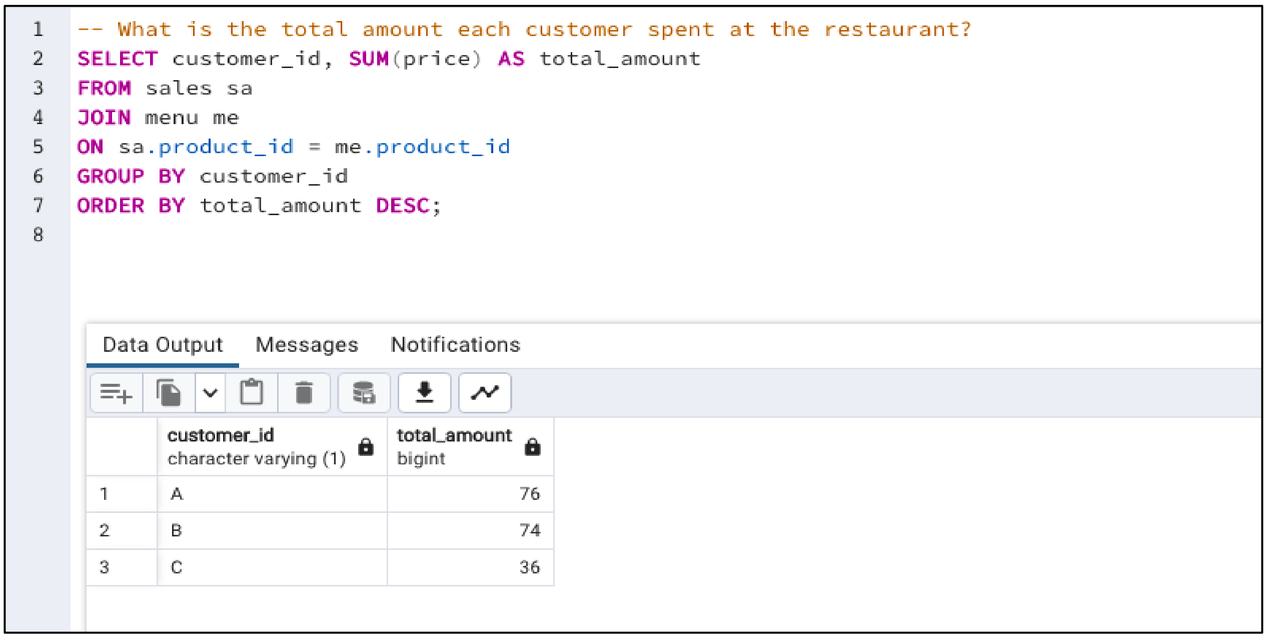
Table 3: members

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



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

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



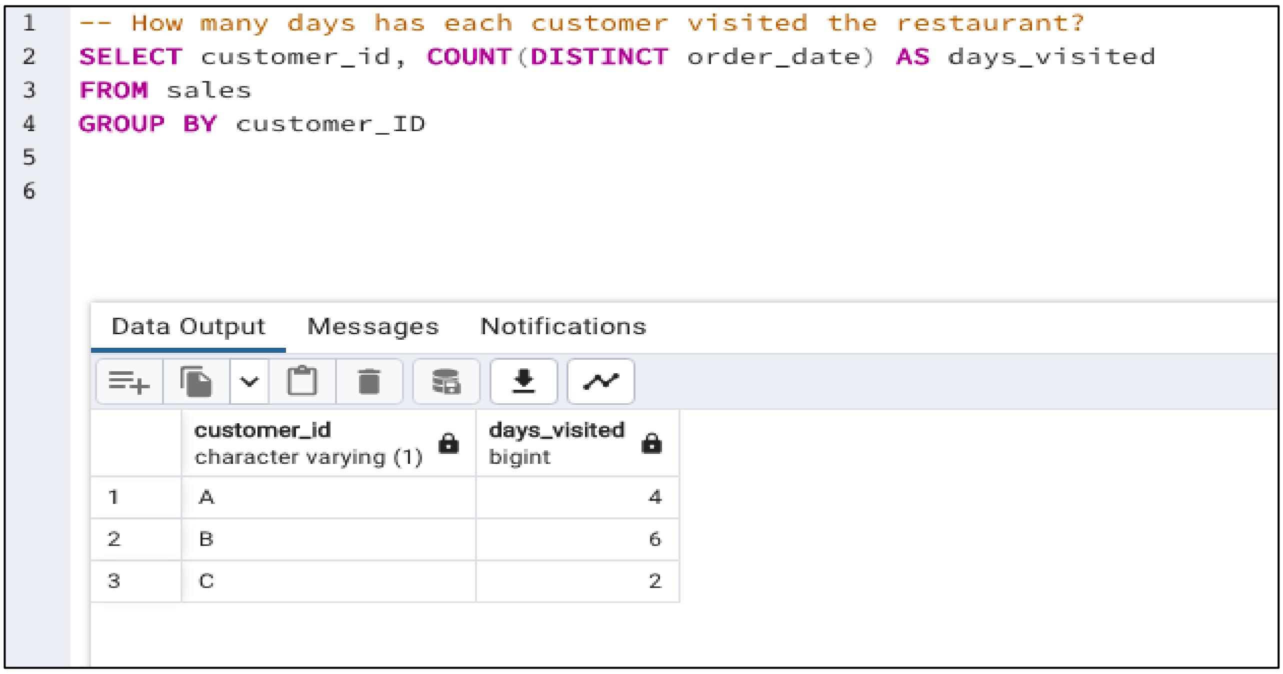
```sql
-- What is the total amount each customer spent at the restaurant?
SELECT customer_id, SUM(price) AS total_amount
FROM sales sa
JOIN menu me
ON sa.product_id = me.product_id
GROUP BY customer_id
ORDER BY total_amount DESC;
```

Data Output   Messages   Notifications

	customer_id character varying (1)	total_amount bigint
1	A	76
2	B	74
3	C	36

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

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



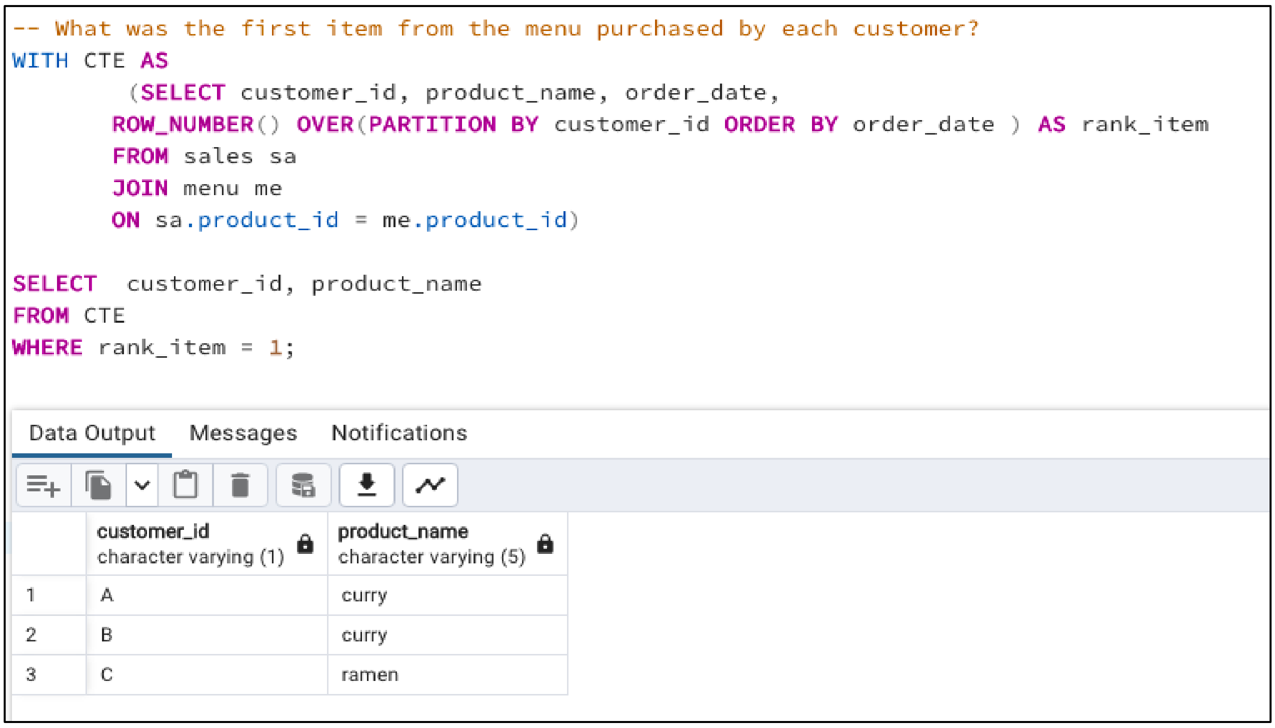
```sql
-- How many days has each customer visited the restaurant?
SELECT customer_id, COUNT(DISTINCT order_date) AS days_visited
FROM sales
GROUP BY customer_ID
```

Data Output   Messages   Notifications

	customer_id character varying (1)	days_visited bigint
1	A	4
2	B	6
3	C	2

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

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



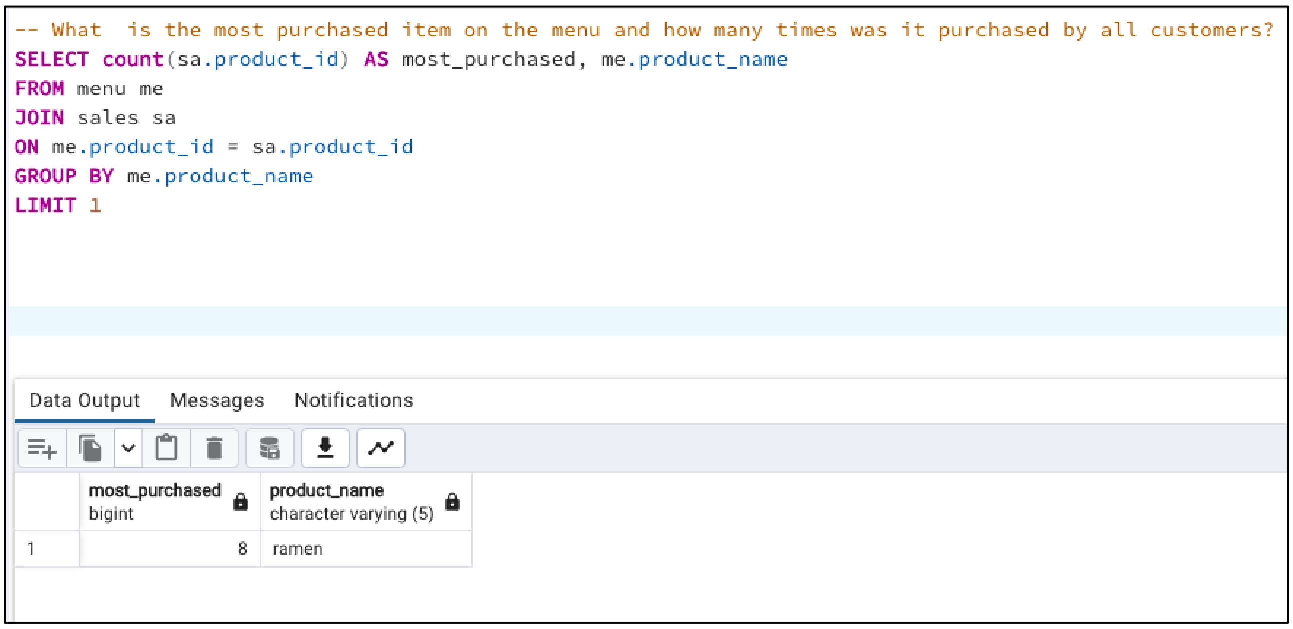
```sql
-- What was the first item from the menu purchased by each customer?
WITH CTE AS
        (SELECT customer_id, product_name, order_date,
       ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY order_date ) AS rank_item
       FROM sales sa
       JOIN menu me
       ON sa.product_id = me.product_id)

SELECT  customer_id, product_name
FROM CTE
WHERE rank_item = 1;
```

Data Output Messages Notifications

	customer_id character varying (1)	product_name character varying (5)
1	A	curry
2	B	curry
3	C	ramen

Result: Customer A purchased curry first, Customer B purchased curry and Customer C purchased ramen.

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



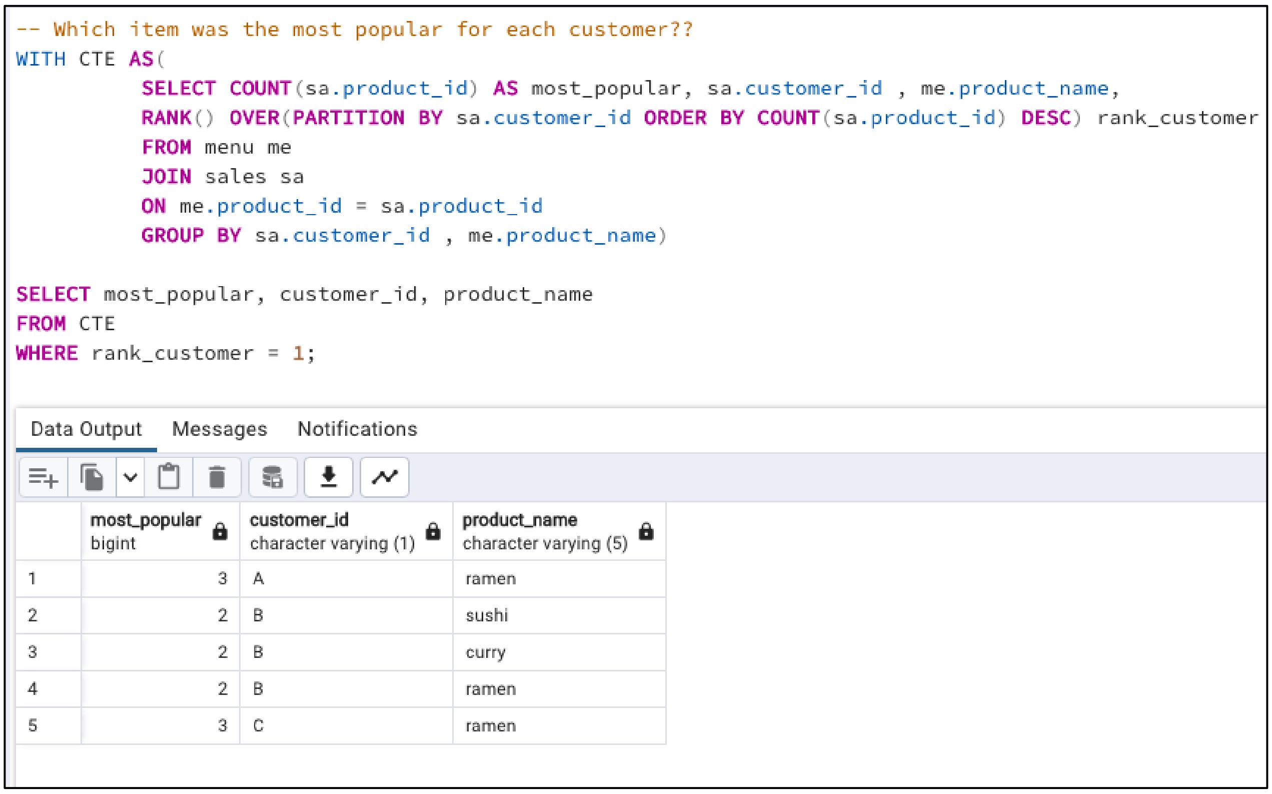
```sql
-- What  is the most purchased item on the menu and how many times was it purchased by all customers?
SELECT count(sa.product_id) AS most_purchased, me.product_name
FROM menu me
JOIN sales sa
ON me.product_id = sa.product_id
GROUP BY me.product_name
LIMIT 1
```

Data Output Messages Notifications

	most_purchased bigint	product_name character varying (5)
1	8	ramen

Result: The most purchased item was ramen and it was purchased 8 times.

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



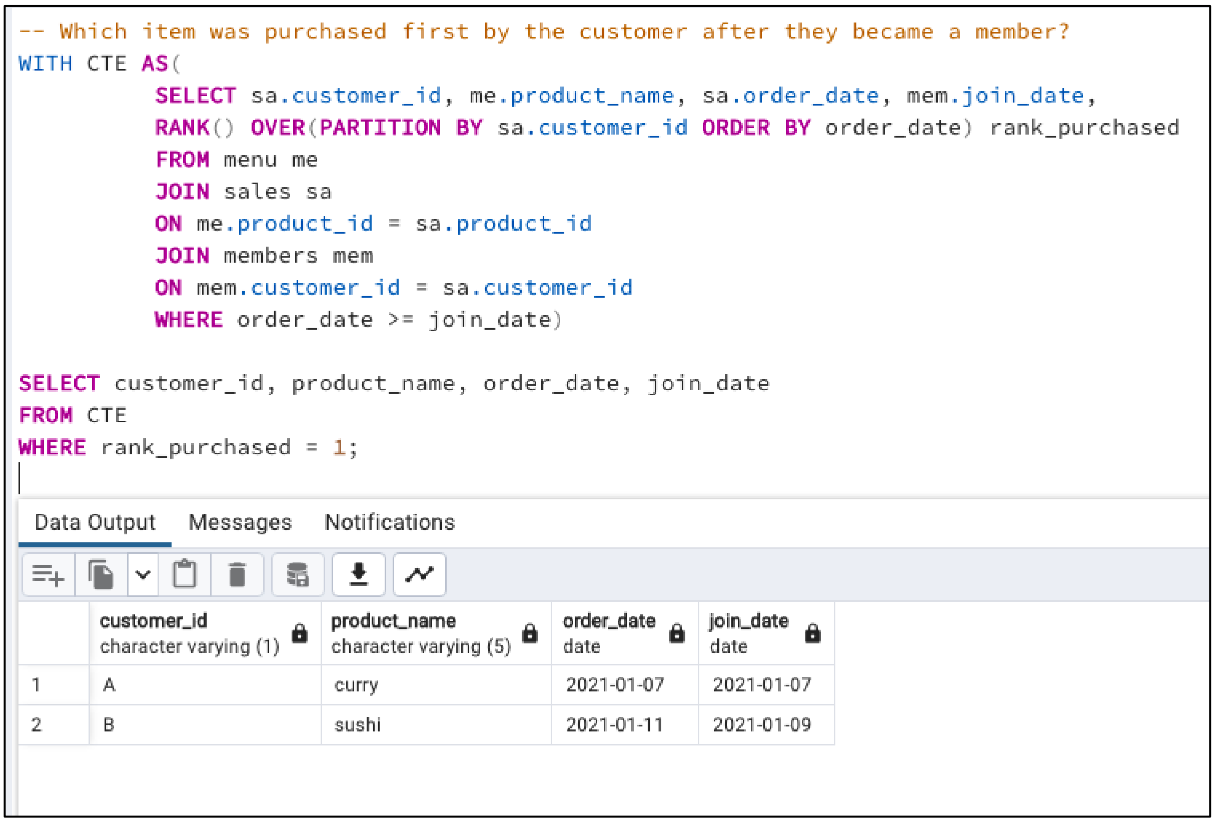
```
-- Which item was the most popular for each customer??
WITH CTE AS(
          SELECT COUNT(sa.product_id) AS most_popular, sa.customer_id , me.product_name,
          RANK() OVER(PARTITION BY sa.customer_id ORDER BY COUNT(sa.product_id) DESC) rank_customer
          FROM menu me
          JOIN sales sa
          ON me.product_id = sa.product_id
          GROUP BY sa.customer_id , me.product_name)

SELECT most_popular, customer_id, product_name
FROM CTE
WHERE rank_customer = 1;
```

Data Output Messages Notifications

	most_popular bigint	customer_id character varying (1)	product_name character varying (5)
1	3	A	ramen
2	2	B	sushi
3	2	B	curry
4	2	B	ramen
5	3	C	ramen

Result: The most popular item for Customer A was ramen, Customer B was sushi, curry and ramen while Customer C was ramen.

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

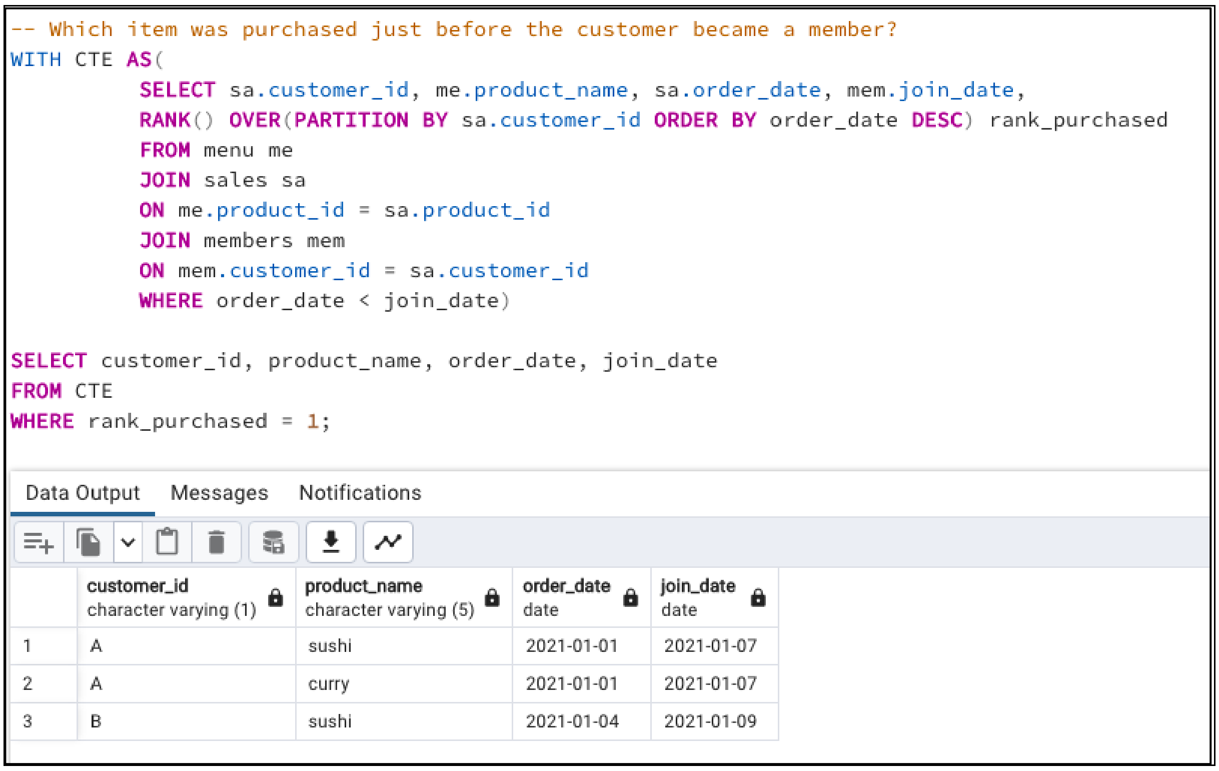


```
-- Which item was purchased first by the customer after they became a member?
WITH CTE AS(
          SELECT sa.customer_id, me.product_name, sa.order_date, mem.join_date,
          RANK() OVER(PARTITION BY sa.customer_id ORDER BY order_date) rank_purchased
          FROM menu me
          JOIN sales sa
          ON me.product_id = sa.product_id
          JOIN members mem
          ON mem.customer_id = sa.customer_id
          WHERE order_date >= join_date)

SELECT customer_id, product_name, order_date, join_date
FROM CTE
WHERE rank_purchased = 1;
```

	customer_id character varying (1)	product_name character varying (5)	order_date date	join_date date
1	A	curry	2021-01-07	2021-01-07
2	B	sushi	2021-01-11	2021-01-09

Result: The item purchased by Customer A after they became a member was curry and the item purchased by Customer B after they became a member was sushi.

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

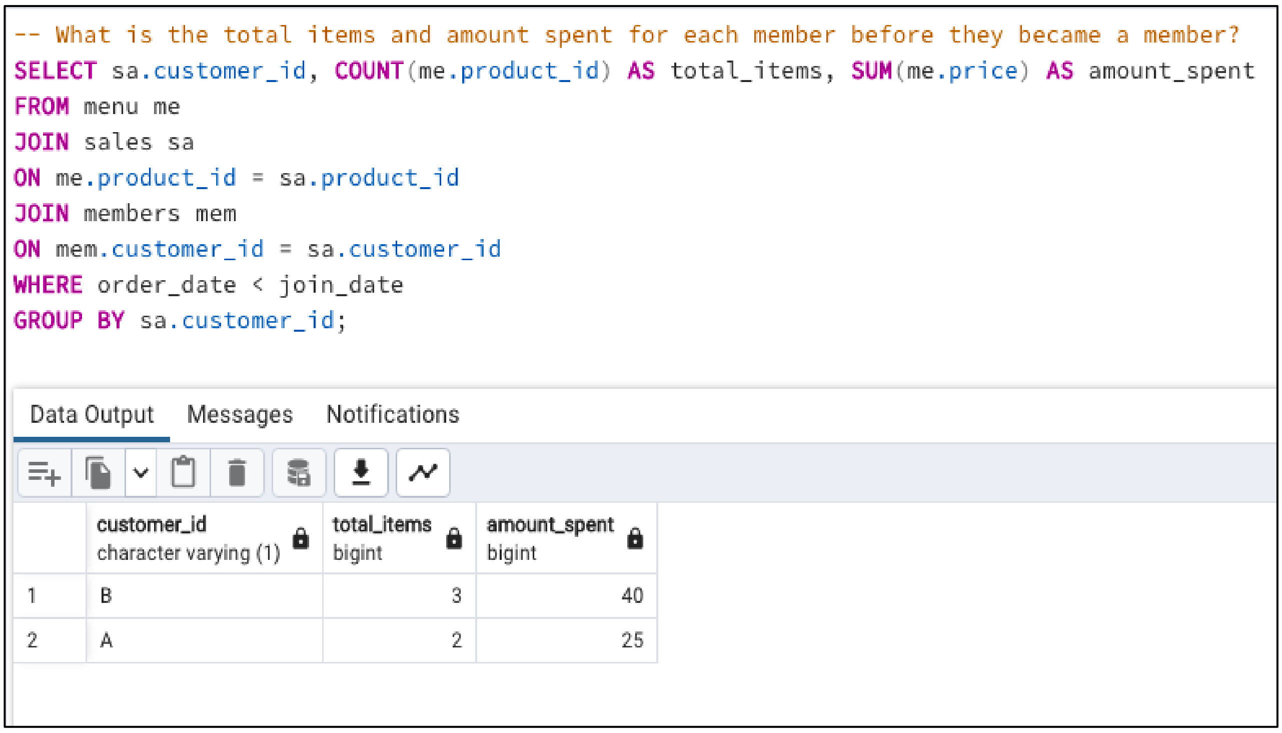


```
-- Which item was purchased just before the customer became a member?
WITH CTE AS(
          SELECT sa.customer_id, me.product_name, sa.order_date, mem.join_date,
          RANK() OVER(PARTITION BY sa.customer_id ORDER BY order_date DESC) rank_purchased
          FROM menu me
          JOIN sales sa
          ON me.product_id = sa.product_id
          JOIN members mem
          ON mem.customer_id = sa.customer_id
          WHERE order_date < join_date)

SELECT customer_id, product_name, order_date, join_date
FROM CTE
WHERE rank_purchased = 1;
```

	customer_id character varying (1)	product_name character varying (5)	order_date date	join_date date
1	A	sushi	2021-01-01	2021-01-07
2	A	curry	2021-01-01	2021-01-07
3	B	sushi	2021-01-04	2021-01-09

Result: The item purchased by Customer A just before they became a member was curry and sushi. The item purchased by Customer B just before they became a member was sushi.

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



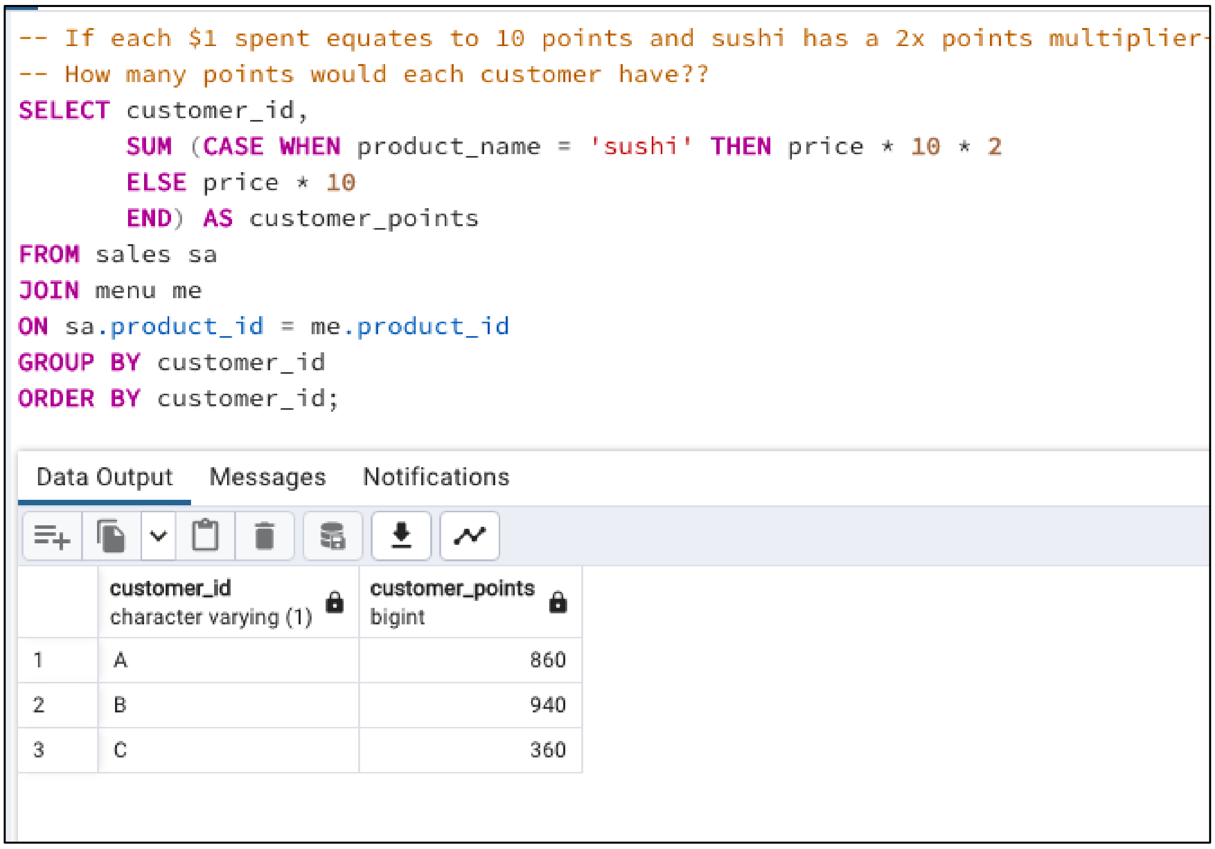
```
-- What is the total items and amount spent for each member before they became a member?
SELECT sa.customer_id, COUNT(me.product_id) AS total_items, SUM(me.price) AS amount_spent
FROM menu me
JOIN sales sa
ON me.product_id = sa.product_id
JOIN members mem
ON mem.customer_id = sa.customer_id
WHERE order_date < join_date
GROUP BY sa.customer_id;
```

Data Output Messages Notifications

	customer_id character varying (1)	total_items bigint	amount_spent bigint
1	B	3	40
2	A	2	25

Result: Customer A purchased 2 items and spent $25 just before they became a member. Customer B purchased 3 items and spent $40 just 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?**



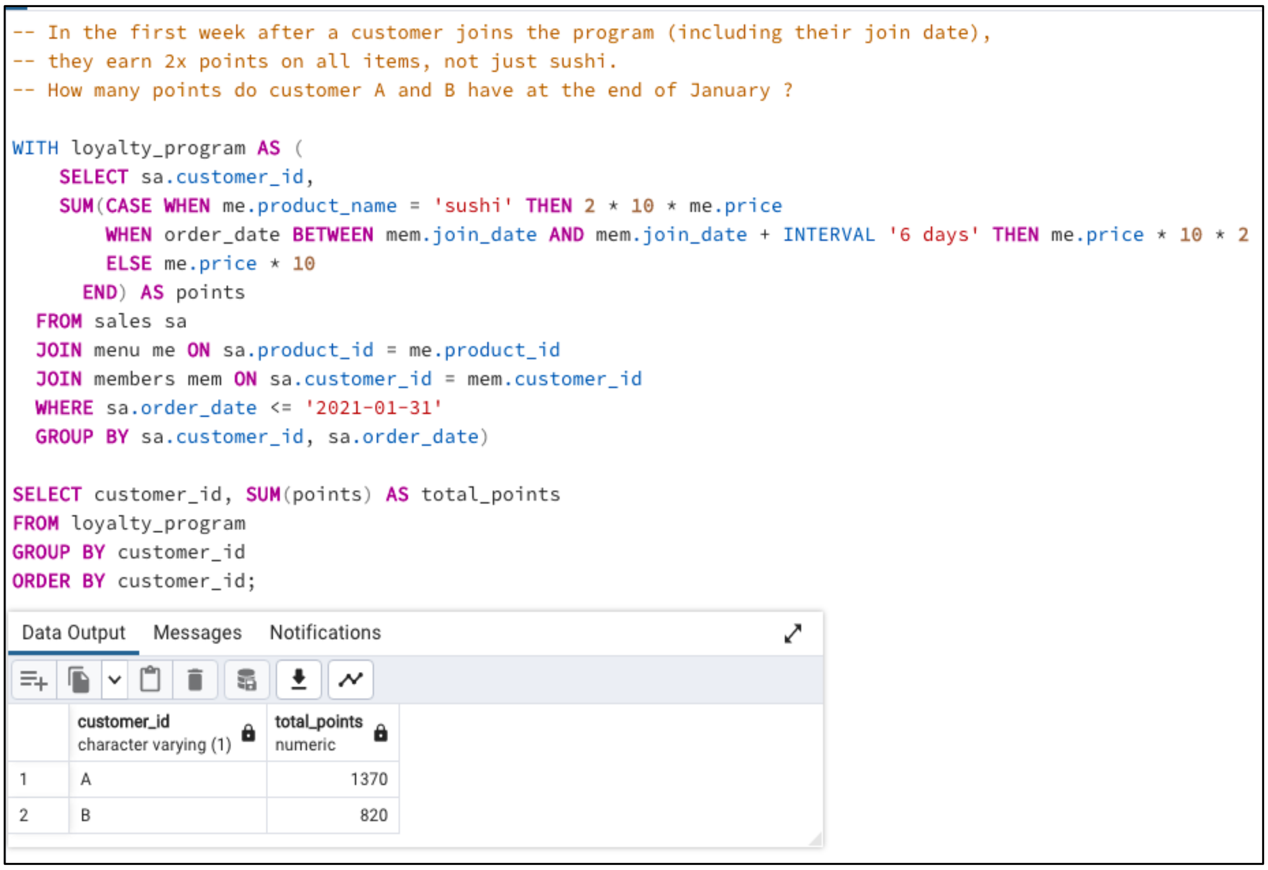
```
-- If each $1 spent equates to 10 points and sushi has a 2x points multiplier
-- How many points would each customer have??
SELECT customer_id,
       SUM (CASE WHEN product_name = 'sushi' THEN price * 10 * 2
       ELSE price * 10
       END) AS customer_points
FROM sales sa
JOIN menu me
ON sa.product_id = me.product_id
GROUP BY customer_id
ORDER BY customer_id;
```

Data Output Messages Notifications

	customer_id character varying (1)	customer_points bigint
1	A	860
2	B	940
3	C	360

Result: Customer A had 860 points, Customer B had 940 points and Customer C had 360 points.

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 ?



```
-- 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 loyalty_program AS (
    SELECT sa.customer_id,
    SUM(CASE WHEN me.product_name = 'sushi' THEN 2 * 10 * me.price
        WHEN order_date BETWEEN mem.join_date AND mem.join_date + INTERVAL '6 days' THEN me.price * 10 * 2
        ELSE me.price * 10
      END) AS points
  FROM sales sa
  JOIN menu me ON sa.product_id = me.product_id
  JOIN members mem ON sa.customer_id = mem.customer_id
  WHERE sa.order_date <= '2021-01-31'
  GROUP BY sa.customer_id, sa.order_date)

SELECT customer_id, SUM(points) AS total_points
FROM loyalty_program
GROUP BY customer_id
ORDER BY customer_id;
```

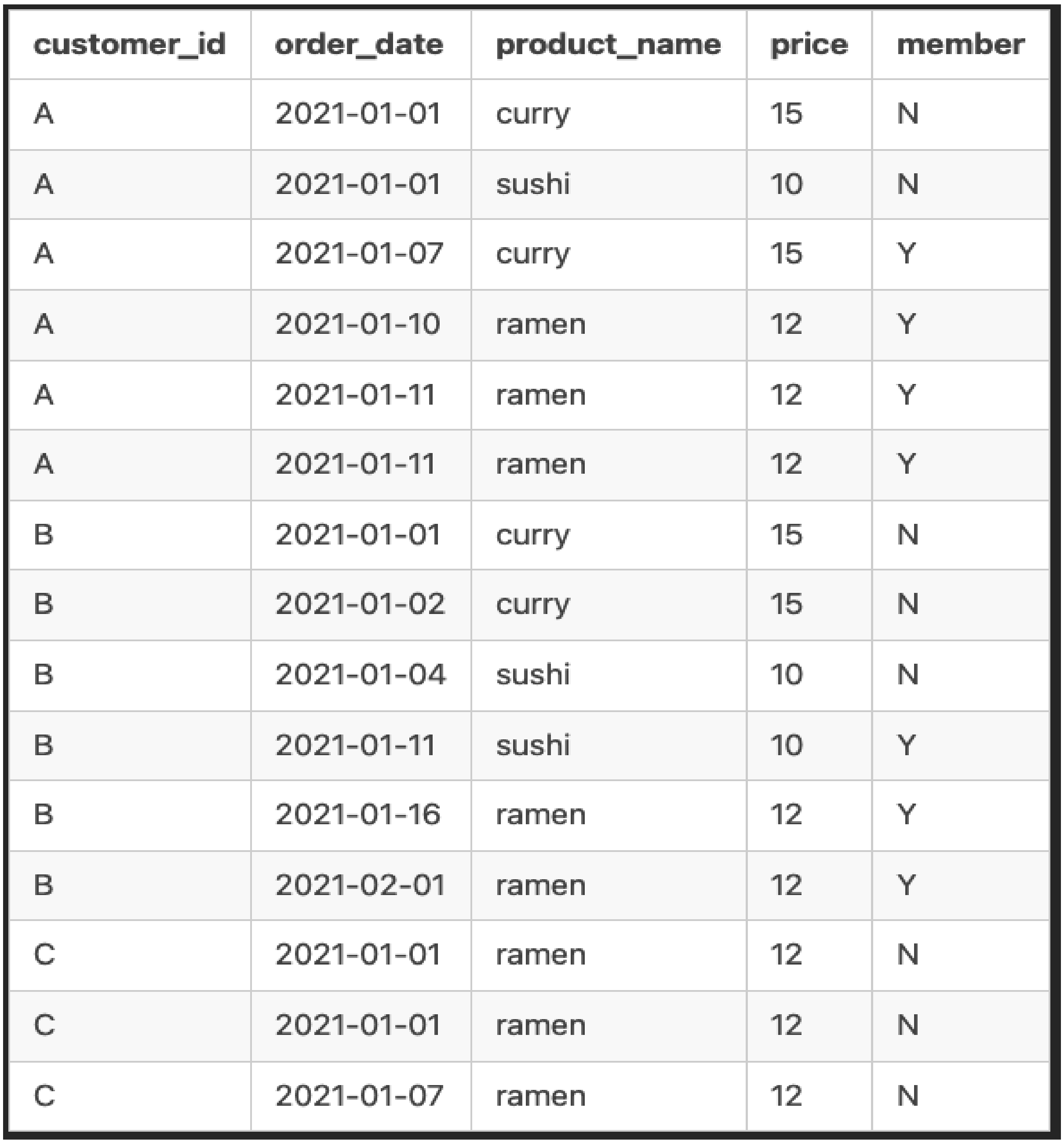
	customer_id character varying (1)	total_points numeric
1	A	1370
2	B	820

Result: Customer A had 1370 points, Customer B had 820 points after the first week of joining the loyalty program.

BONUS QUESTION 1: JOIN ALL THE THINGS

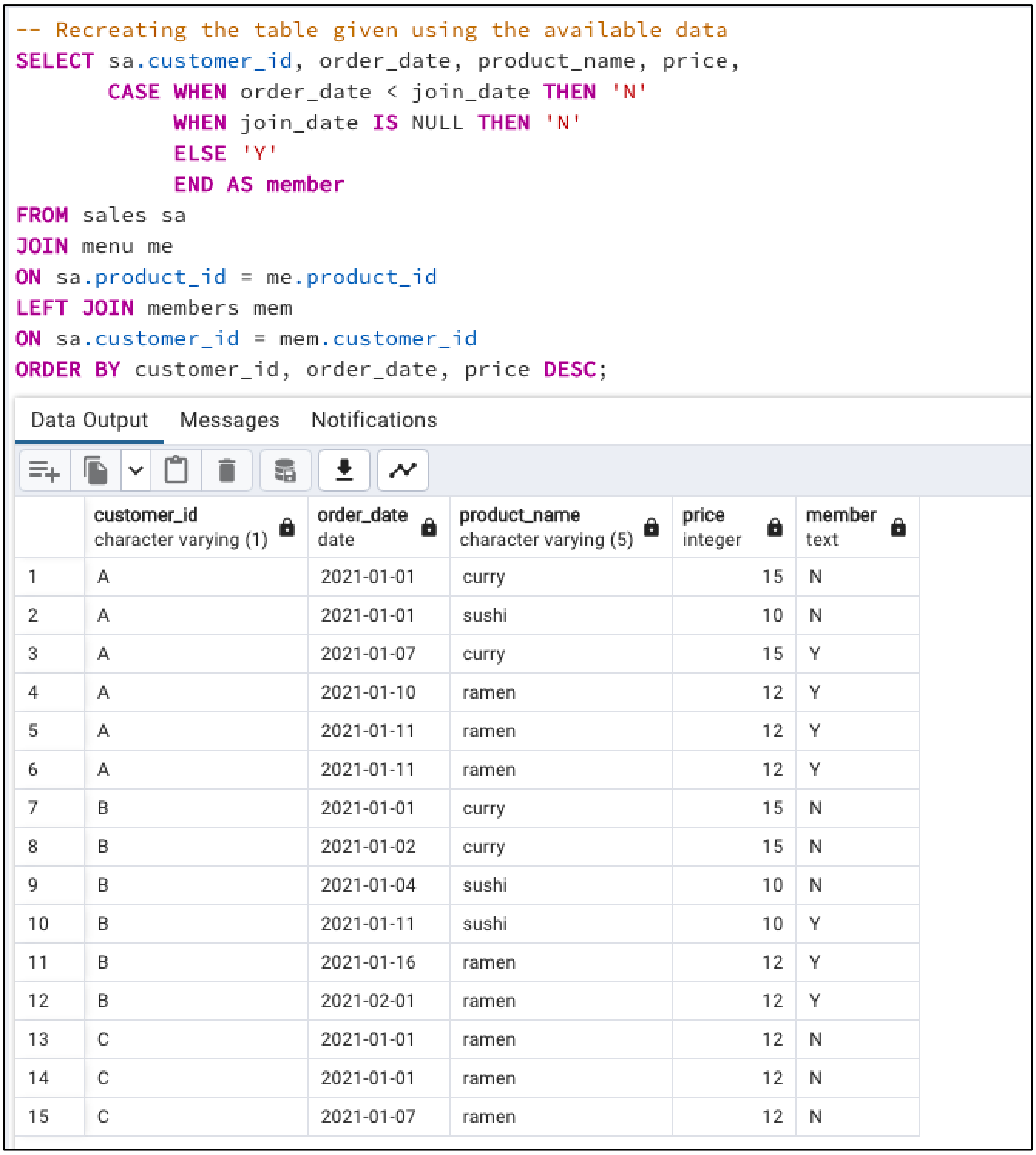
The following questions are related creating basic data tables that Danny and his team can use to quickly derive insights without needing to join the underlying tables using SQL.

Recreate the following table output using the available data:



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

RESULT

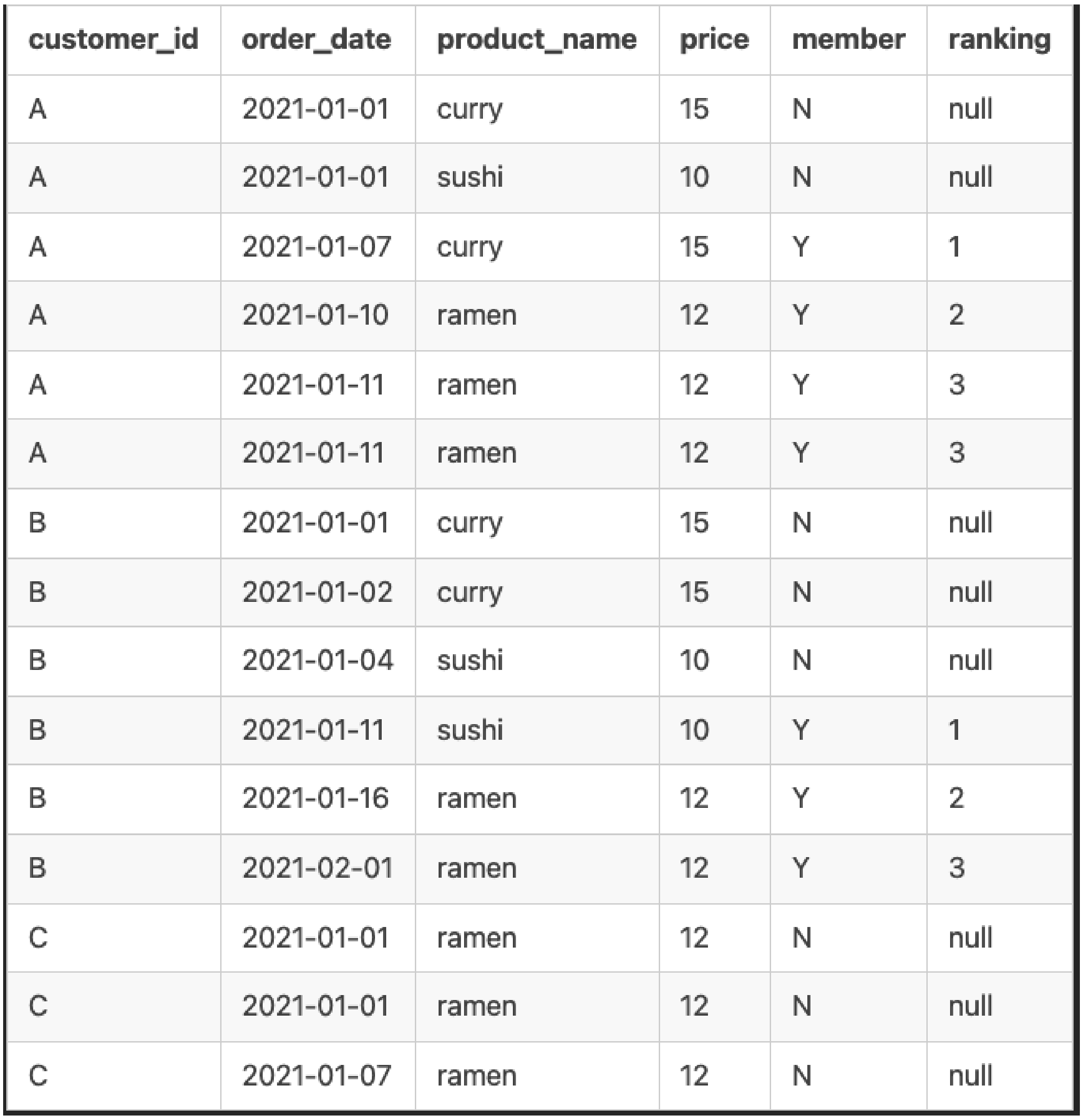


```
-- Recreating the table given using the available data
SELECT sa.customer_id, order_date, product_name, price,
       CASE WHEN order_date < join_date THEN 'N'
            WHEN join_date IS NULL THEN 'N'
            ELSE 'Y'
            END AS member
FROM sales sa
JOIN menu me
ON sa.product_id = me.product_id
LEFT JOIN members mem
ON sa.customer_id = mem.customer_id
ORDER BY customer_id, order_date, price DESC;
```

	customer_id character varying (1)	order_date date	product_name character varying (5)	price integer	member text
1	A	2021-01-01	curry	15	N
2	A	2021-01-01	sushi	10	N
3	A	2021-01-07	curry	15	Y
4	A	2021-01-10	ramen	12	Y
5	A	2021-01-11	ramen	12	Y
6	A	2021-01-11	ramen	12	Y
7	B	2021-01-01	curry	15	N
8	B	2021-01-02	curry	15	N
9	B	2021-01-04	sushi	10	N
10	B	2021-01-11	sushi	10	Y
11	B	2021-01-16	ramen	12	Y
12	B	2021-02-01	ramen	12	Y
13	C	2021-01-01	ramen	12	N
14	C	2021-01-01	ramen	12	N
15	C	2021-01-07	ramen	12	N

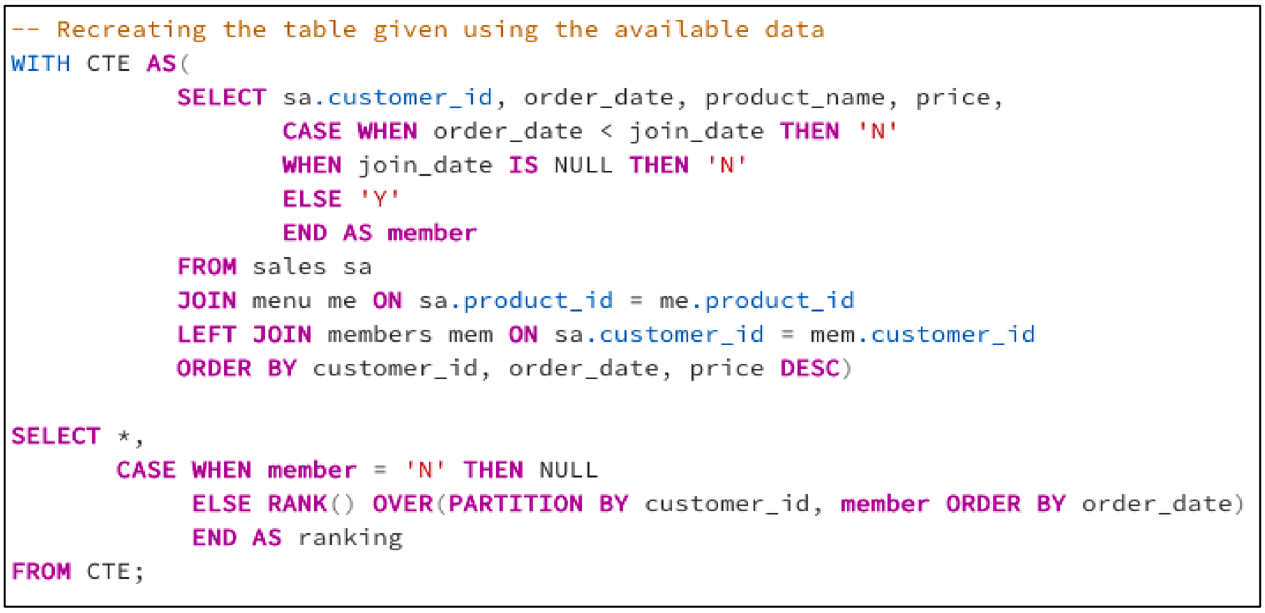
BONUS QUESTION 2: RANK ALL THE THINGS

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.

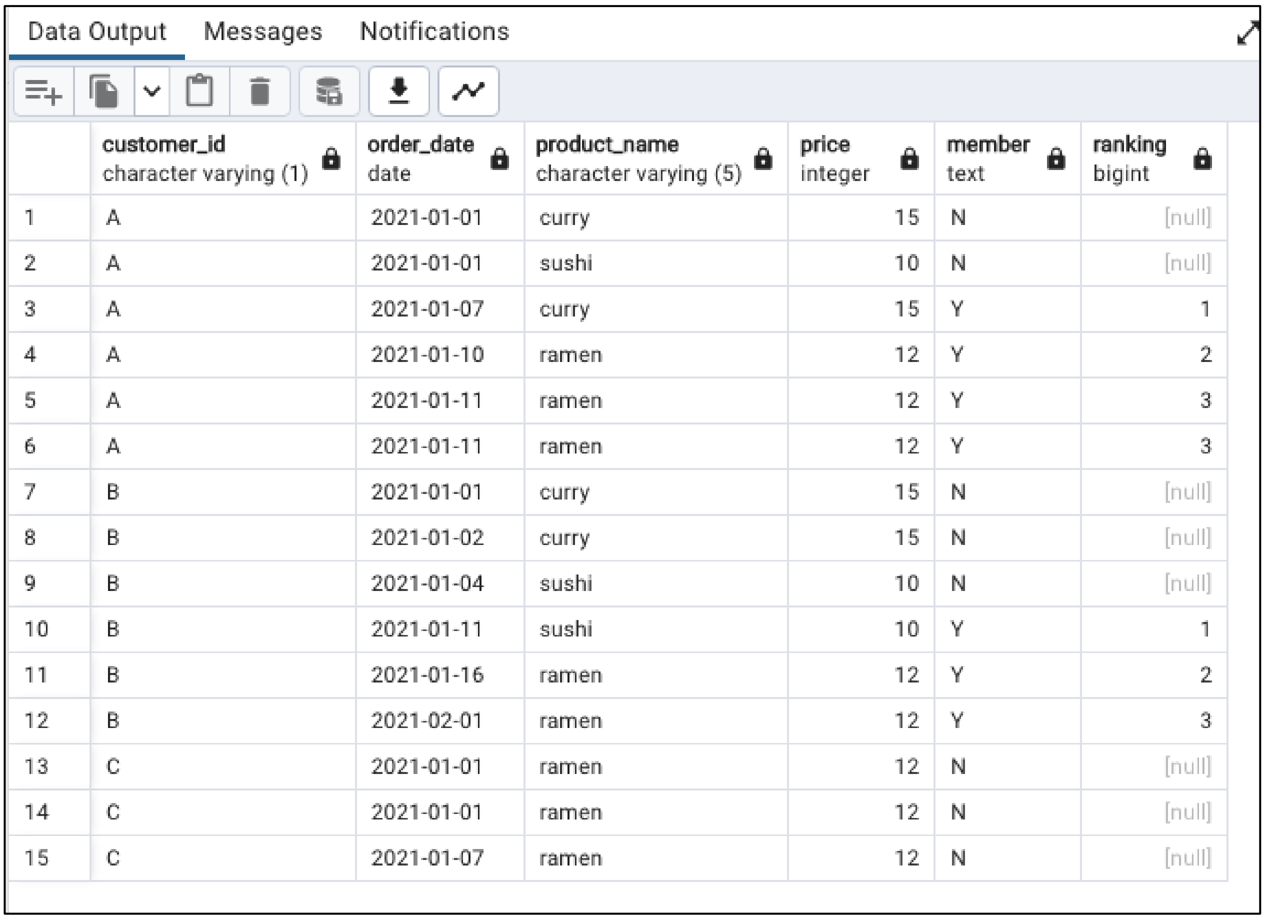


customer_id	order_date	product_name	price	member	ranking
A	2021-01-01	curry	15	N	null
A	2021-01-01	sushi	10	N	null
A	2021-01-07	curry	15	Y	1
A	2021-01-10	ramen	12	Y	2
A	2021-01-11	ramen	12	Y	3
A	2021-01-11	ramen	12	Y	3
B	2021-01-01	curry	15	N	null
B	2021-01-02	curry	15	N	null
B	2021-01-04	sushi	10	N	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

RESULT



```sql
-- Recreating the table given using the available data
WITH CTE AS(
            SELECT sa.customer_id, order_date, product_name, price,
                   CASE WHEN order_date < join_date THEN 'N'
                   WHEN join_date IS NULL THEN 'N'
                   ELSE 'Y'
                   END AS member
            FROM sales sa
            JOIN menu me ON sa.product_id = me.product_id
            LEFT JOIN members mem ON sa.customer_id = mem.customer_id
            ORDER BY customer_id, order_date, price DESC)

SELECT *,
       CASE WHEN member = 'N' THEN NULL
            ELSE RANK() OVER(PARTITION BY customer_id, member ORDER BY order_date)
            END AS ranking
FROM CTE;
```



Data Output Messages Notifications

	customer_id character varying (1)	order_date date	product_name character varying (5)	price integer	member text	ranking bigint
1	A	2021-01-01	curry	15	N	[null]
2	A	2021-01-01	sushi	10	N	[null]
3	A	2021-01-07	curry	15	Y	1
4	A	2021-01-10	ramen	12	Y	2
5	A	2021-01-11	ramen	12	Y	3
6	A	2021-01-11	ramen	12	Y	3
7	B	2021-01-01	curry	15	N	[null]
8	B	2021-01-02	curry	15	N	[null]
9	B	2021-01-04	sushi	10	N	[null]
10	B	2021-01-11	sushi	10	Y	1
11	B	2021-01-16	ramen	12	Y	2
12	B	2021-02-01	ramen	12	Y	3
13	C	2021-01-01	ramen	12	N	[null]
14	C	2021-01-01	ramen	12	N	[null]
15	C	2021-01-07	ramen	12	N	[null]