8 Week SQL Challenge
Case Study #1 - Danny’s Diner

https://8weeksqlchallenge.com/case-study-1/

February 2023

Gino Freud D. Hobayan

https://8weeksqlchallenge.com/getting-started/

What is the 8 Week SQL Challenge?

My idea was to create an online community which supports all data professionals who were specifically starting on their SQL learning journey.

I also wanted to help everyone start crafting their own personal branding, online presence and a personal portfolio of data projects - and so the 8 Week SQL Challenge was born!

For the next 8 weeks - I challenge you to:

- Dedicate yourself to learning SQL
- Share regular updates on social media about what you are learning
- Get started on your own GitHub Pages personal website and project portfolio

- Danny Ma

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 favorite 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

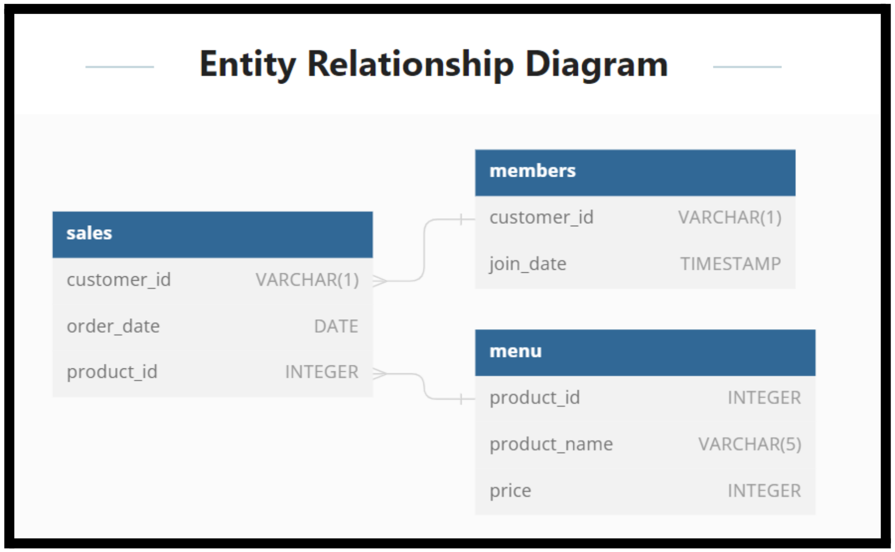
- visiting patterns,
- how much money they've spent, and
- which menu items are their favorite.

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!

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



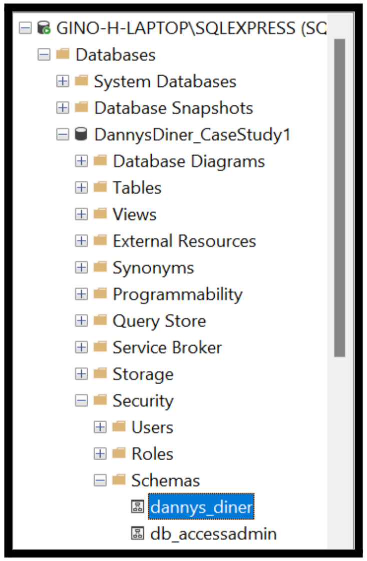
Entity Relationship Diagram

sales	
customer_id	VARCHAR(1)
order_date	DATE
product_id	INTEGER

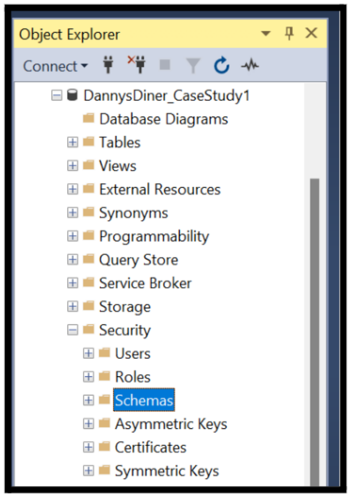
members	
customer_id	VARCHAR(1)
join_date	TIMESTAMP

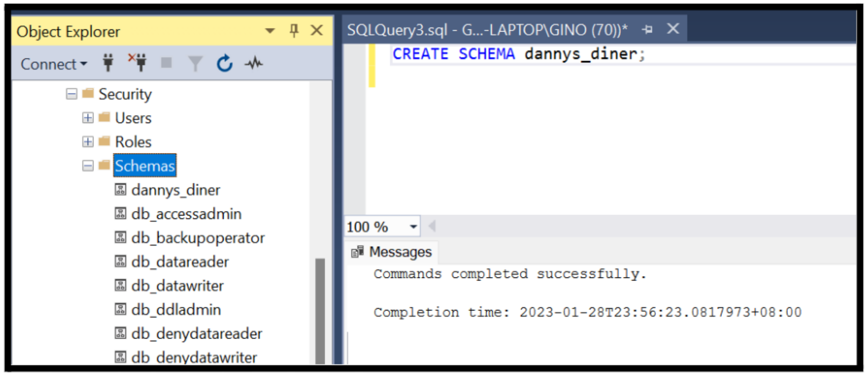
menu	
product_id	INTEGER
product_name	VARCHAR(5)
price	INTEGER

1.) Create the Database, Schema and Tables first



```
CREATE SCHEMA dannys_diner;
```

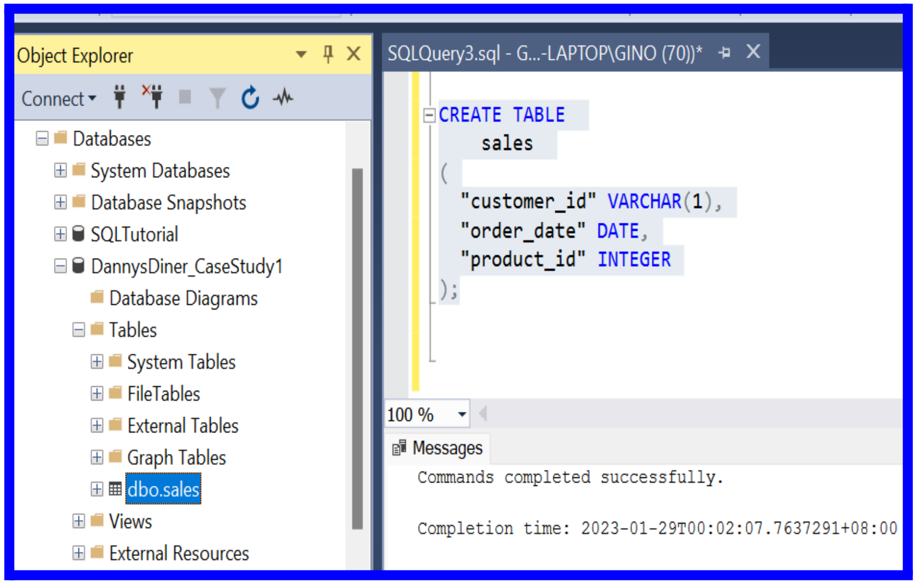




```
CREATE SCHEMA dannys_diner;
```

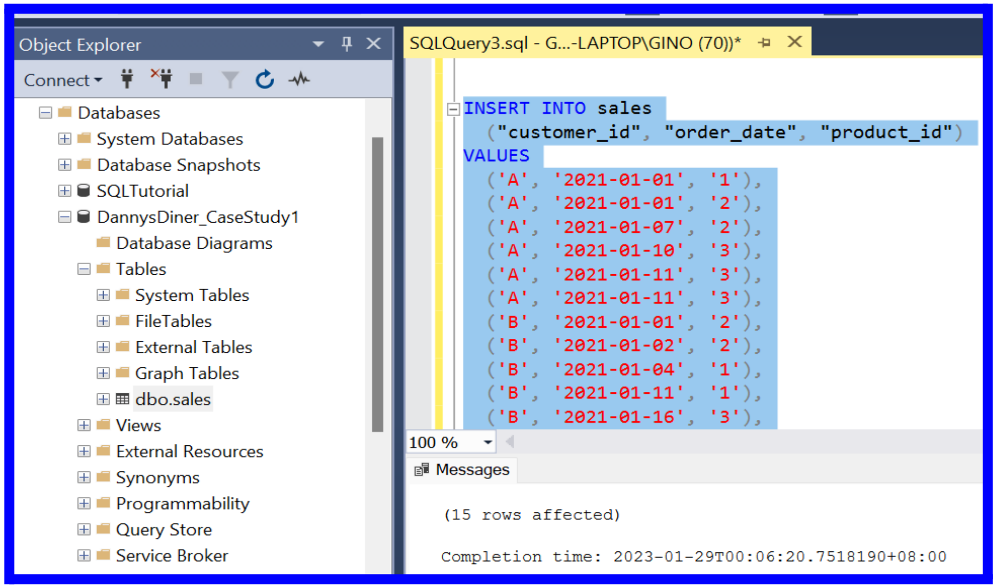
```
Commands completed successfully.

Completion time: 2023-01-28T23:56:23.0817973+08:00
```

sales table



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

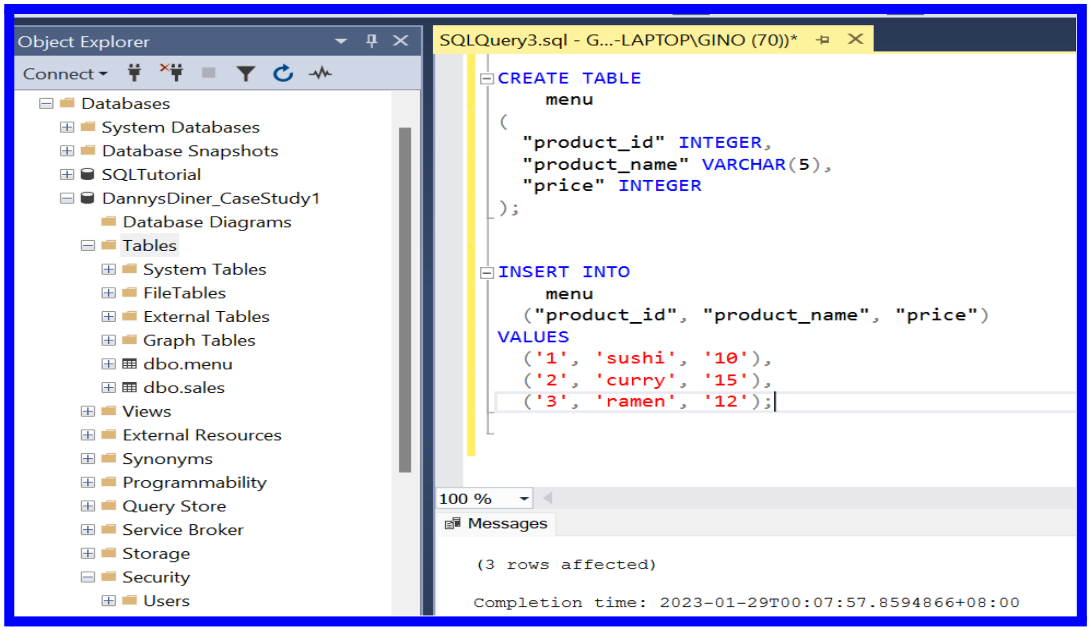
```
Commands completed successfully.

Completion time: 2023-01-29T00:02:07.7637291+08:00
```



```
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'),
```

```
(15 rows affected)

Completion time: 2023-01-29T00:06:20.7518190+08:00
```

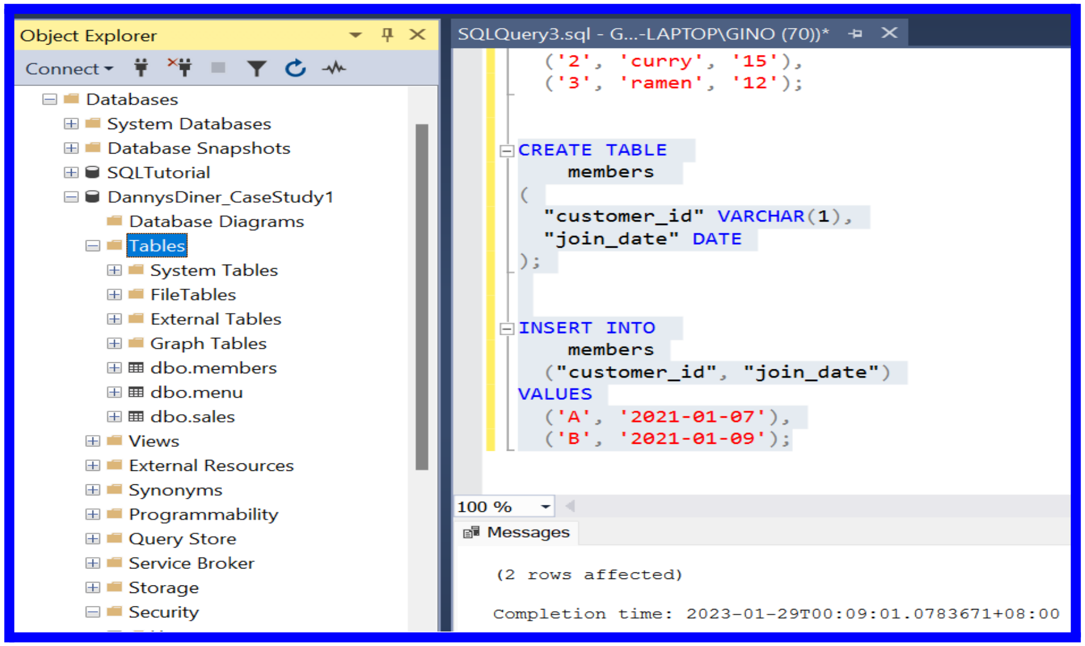
menu table



```sql
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');
```

```
(3 rows affected)

Completion time: 2023-01-29T00:07:57.8594866+08:00
```

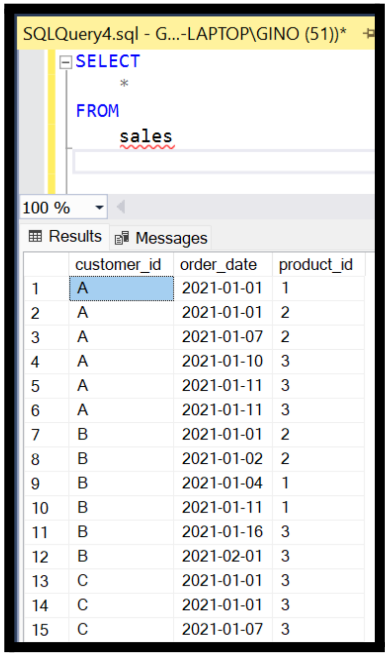
members table



```sql
  ('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');
```

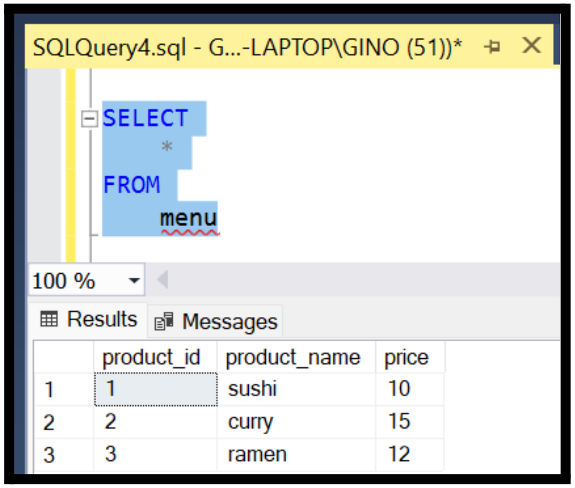
```
(2 rows affected)

Completion time: 2023-01-29T00:09:01.0783671+08:00
```

2.) SELECT Everything from the tables to check if the data is correct.



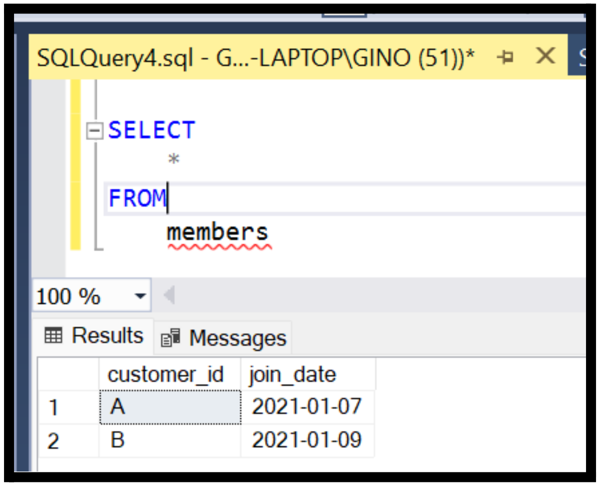
```
SELECT
    *
FROM
    sales
```

	customer_id	order_date	product_id
1	A	2021-01-01	1
2	A	2021-01-01	2
3	A	2021-01-07	2
4	A	2021-01-10	3
5	A	2021-01-11	3
6	A	2021-01-11	3
7	B	2021-01-01	2
8	B	2021-01-02	2
9	B	2021-01-04	1
10	B	2021-01-11	1
11	B	2021-01-16	3
12	B	2021-02-01	3
13	C	2021-01-01	3
14	C	2021-01-01	3
15	C	2021-01-07	3



```
SELECT
    *
FROM
    menu
```

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

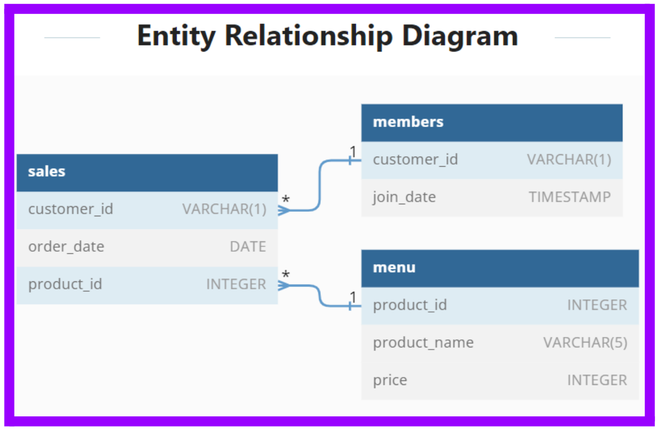


```
SELECT
    *
FROM
    members
```

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

Steps:

1. Break down the question
2. Inspect the ERD and pick only the tables that contain the info we need to answer that specific question

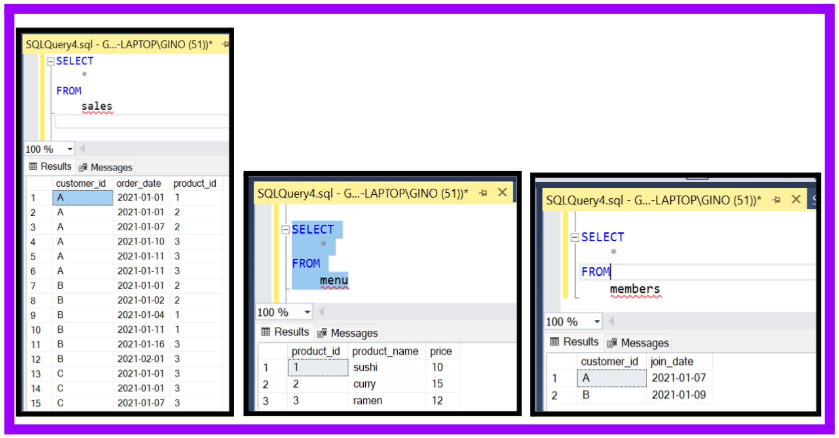


Entity Relationship Diagram

sales	
customer_id	VARCHAR(1)
order_date	DATE
product_id	INTEGER

members	
customer_id	VARCHAR(1)
join_date	TIMESTAMP

menu	
product_id	INTEGER
product_name	VARCHAR(5)
price	INTEGER


SQLQuery4.sql - G...-LAPTOP\GINO (51))*

```
SELECT
    *
FROM
    sales
```

	customer_id	order_date	product_id
1	A	2021-01-01	1
2	A	2021-01-01	2
3	A	2021-01-07	2
4	A	2021-01-10	3
5	A	2021-01-11	3
6	A	2021-01-11	3
7	B	2021-01-01	2
8	B	2021-01-02	2
9	B	2021-01-04	1
10	B	2021-01-11	1
11	B	2021-01-16	3
12	B	2021-02-01	3
13	C	2021-01-01	3
14	C	2021-01-01	3
15	C	2021-01-07	3

```
SELECT
    *
FROM
    menu
```

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

```
SELECT
    *
FROM
    members
```

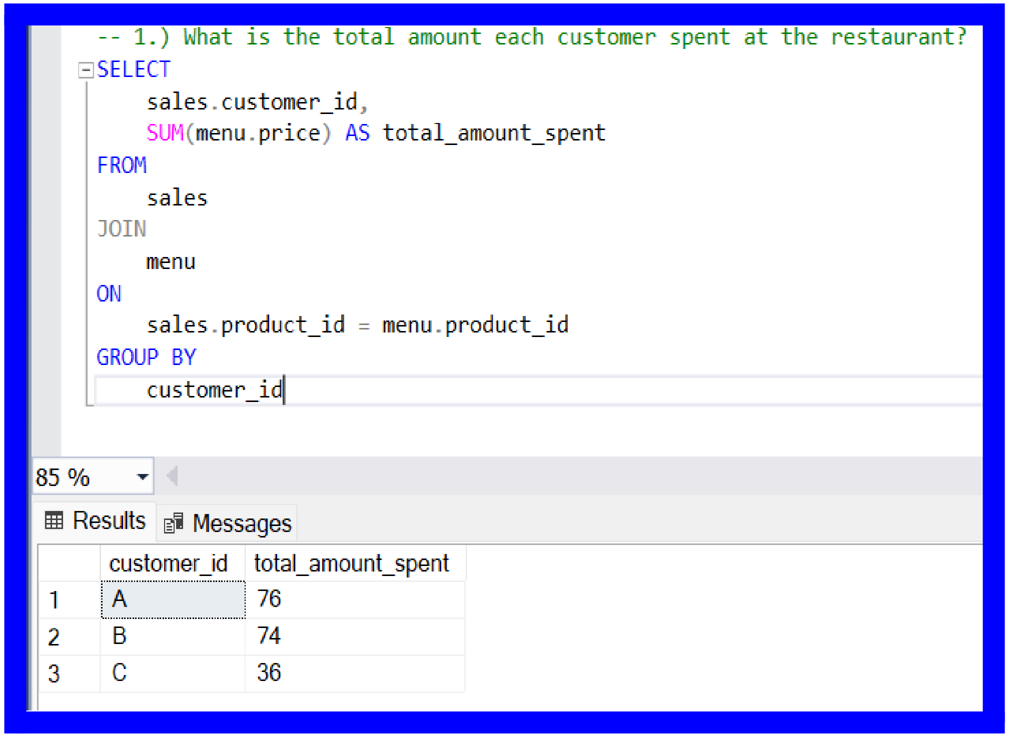
	customer_id	join_date
1	A	2021-01-07
2	B	2021-01-09

3. Which of the following tables contain the data/info we need to solve this specific question?
4. QUERY!

Case Study Questions:

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

What (?) is **the total amount** **SUM(menu.price)**
Each customer **(sales.customer_id)** spent at the restaurant?



```
-- 1.) What is the total amount each customer spent at the restaurant?
SELECT
    sales.customer_id,
    SUM(menu.price) AS total_amount_spent
FROM
    sales
JOIN
    menu
ON
    sales.product_id = menu.product_id
GROUP BY
    customer_id
```

```
   customer_id  total_amount_spent
1  A            76
2  B            74
3  C            36
```

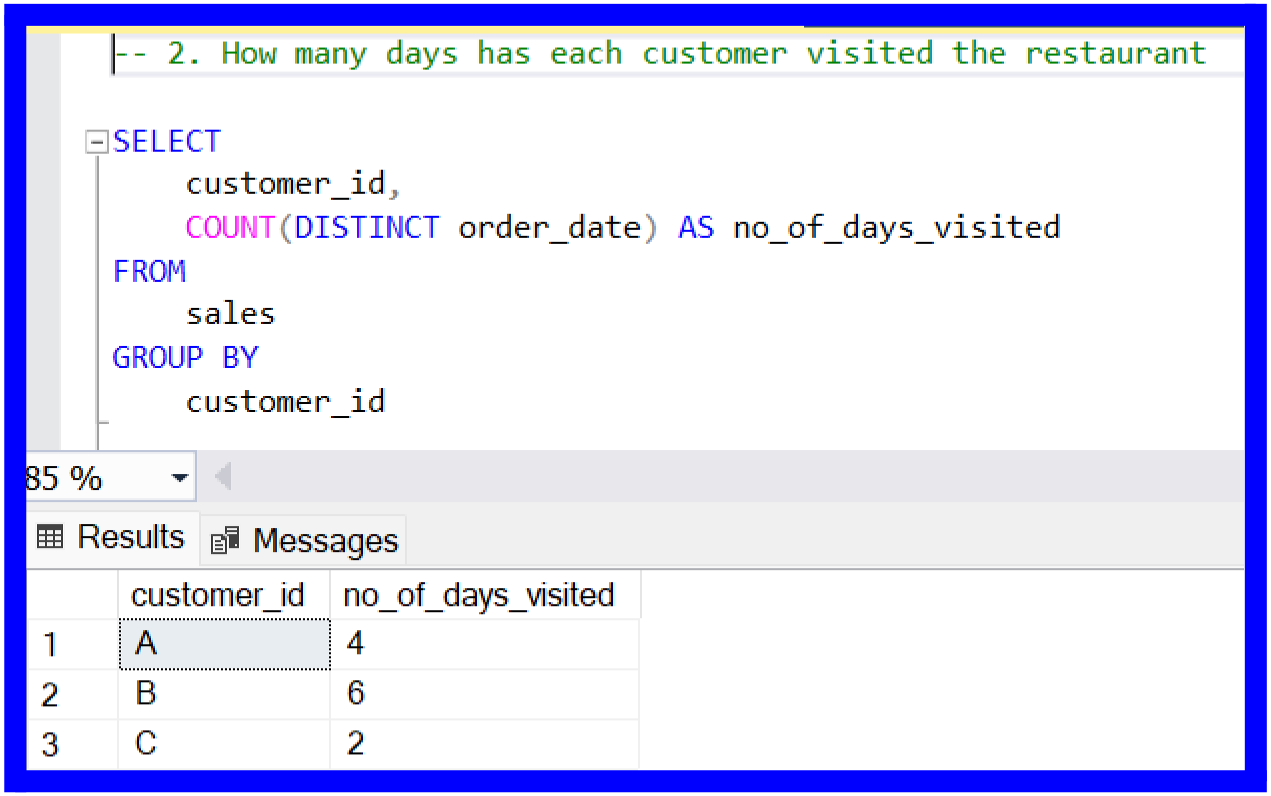
EXPLANATION:

- **I JOINED the 2 tables that contain the information we need.**
    - **sales and menu tables.**
- **SELECTED columns**
    - **customer_id (sales table) (each customer spent?)**
    - **price (menu table) (What is the total amount) SUM(menu.price) AS total_amount_spent**
- **Then GROUPED BY customer_id**

  (Grouped by the distinct values on the customer_id column)

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

How many (COUNT) days
has **each customer (customer_id)**
visited the restaurant (order_date)?



```
-- 2. How many days has each customer visited the restaurant

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

	customer_id	no_of_days_visited
1	A	4
2	B	6
3	C	2

EXPLANATION:

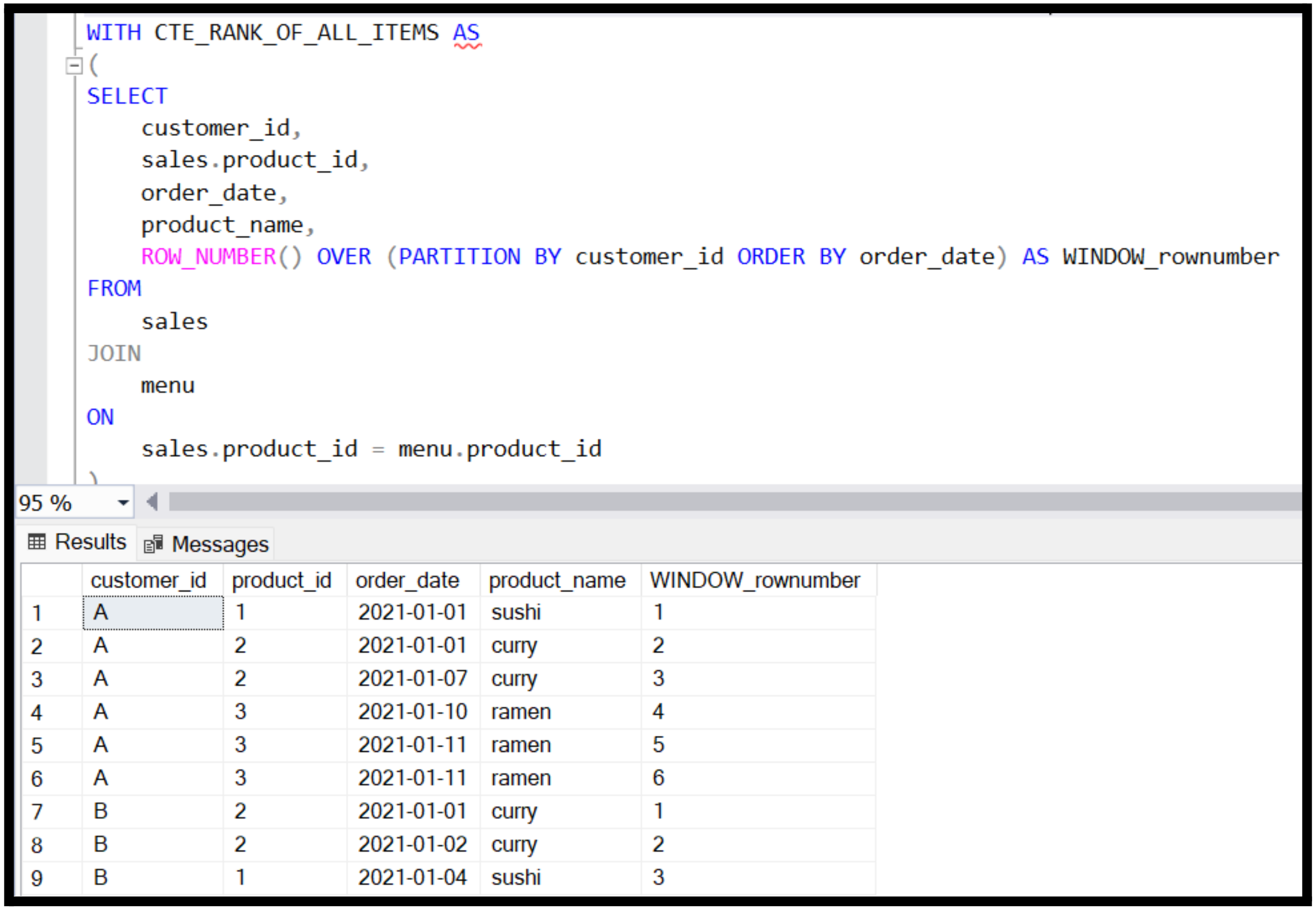
- We used **COUNT** = to answer **How many**
  We used **COUNT DISTINCT** = to get the **distinct/unique values only**.

- **If we do not use COUNT DISTINCT on order_date, we would get a different result. Since a customer can order/visit multiple times on the same day.**

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

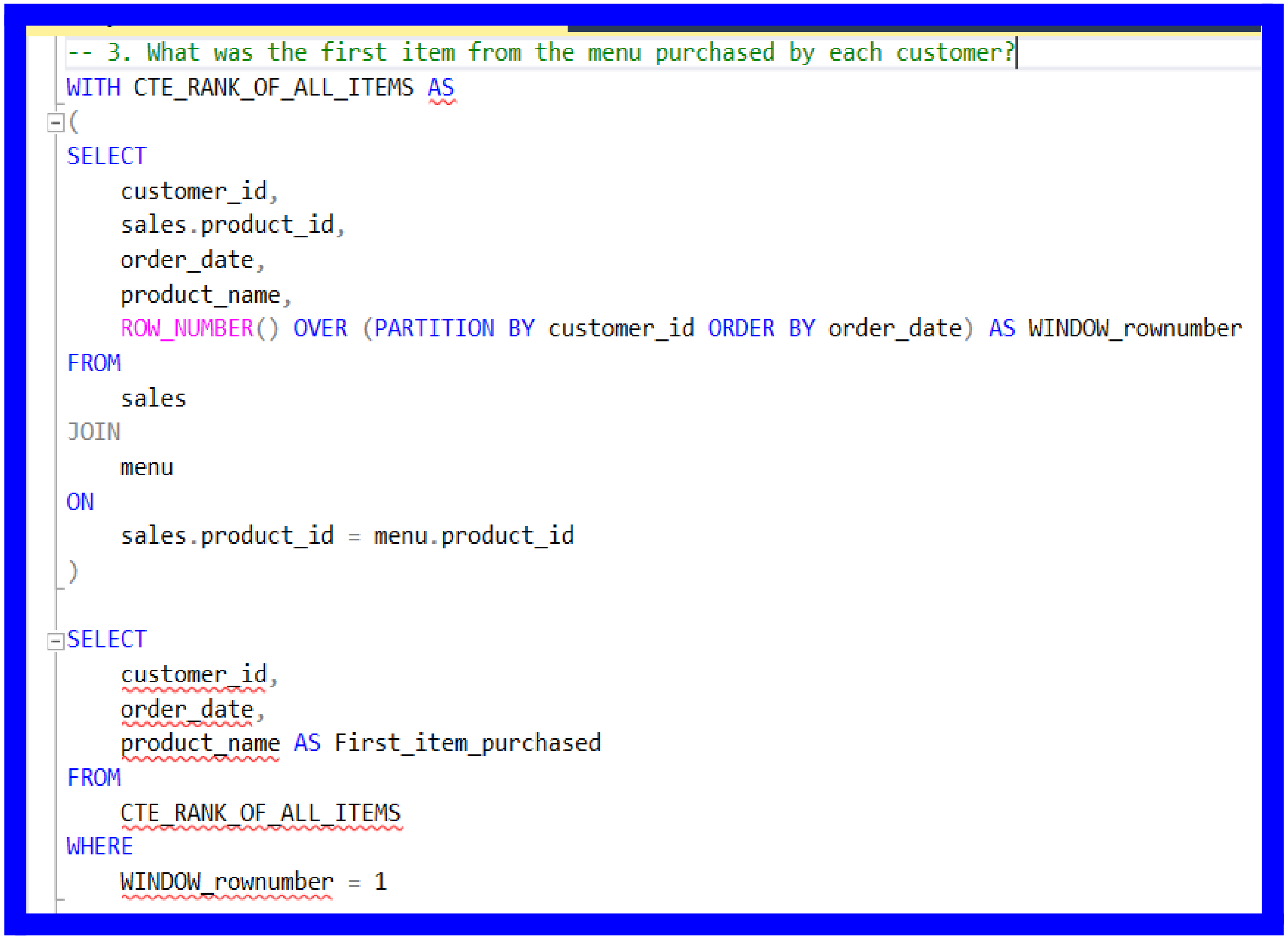
What was the **first item** from the **menu (ROW_NUMBER)**
purchased **by each customer? (customer_id)**

WITH CTE_RANK_OF_ALL_ITEMS AS

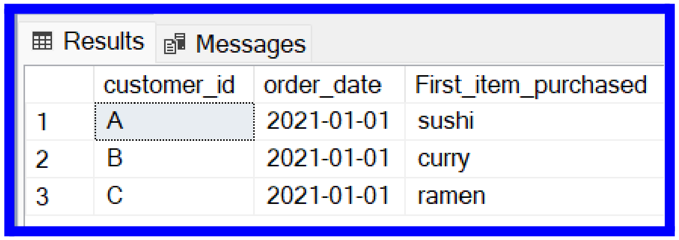


```
WITH CTE_RANK_OF_ALL_ITEMS AS
(
SELECT
    customer_id,
    sales.product_id,
    order_date,
    product_name,
    ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date) AS WINDOW_rownumber
FROM
    sales
JOIN
    menu
ON
    sales.product_id = menu.product_id
)
```

	customer_id	product_id	order_date	product_name	WINDOW_rownumber
1	A	1	2021-01-01	sushi	1
2	A	2	2021-01-01	curry	2
3	A	2	2021-01-07	curry	3
4	A	3	2021-01-10	ramen	4
5	A	3	2021-01-11	ramen	5
6	A	3	2021-01-11	ramen	6
7	B	2	2021-01-01	curry	1
8	B	2	2021-01-02	curry	2
9	B	1	2021-01-04	sushi	3



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

SELECT
    customer_id,
    order_date,
    product_name AS First_item_purchased
FROM
    CTE_RANK_OF_ALL_ITEMS
WHERE
    WINDOW_rownumber = 1
```



	customer_id	order_date	First_item_purchased
1	A	2021-01-01	sushi
2	B	2021-01-01	curry
3	C	2021-01-01	ramen

EXPLANATION:

- I created a temp table/CTE with the Window function ROW_NUMBER to get the RANKINGS of ALL of the items based on their order_date (purchase date)
  ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date) AS WINDOW_rownumber

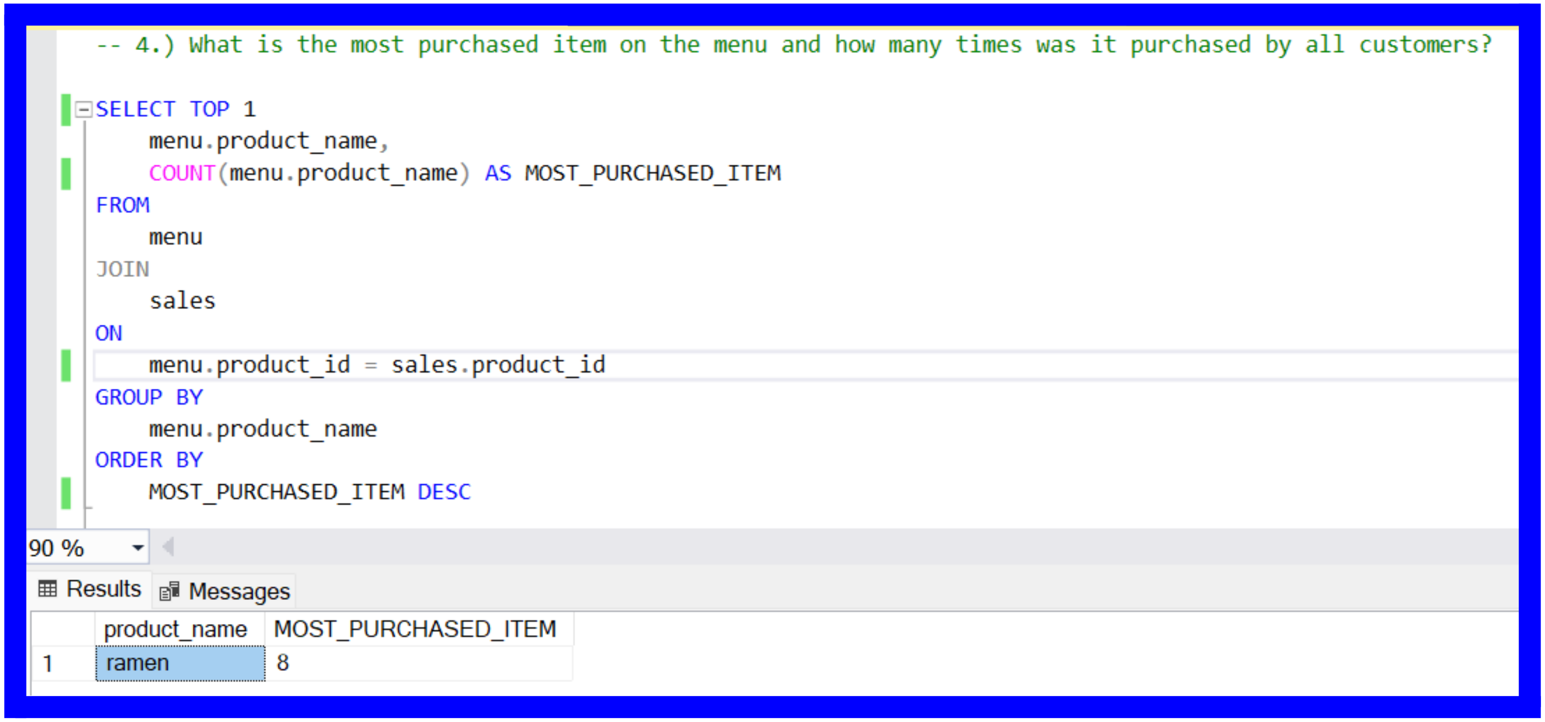
- We used ROW_NUMBER since it gives a ranking per row and is divided per Customer A,B,C because of the **(PARTITION BY customer_id)**

- Ordered by their order_date and in default ASC order since we want the earliest date. **(ORDER BY order_date)**

- So that we can have a filtered set of data to query **WITH CTE_RANK_OF_ALL_ITEMS AS**
  In order to find the 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?

What is the most purchased item **(product_name)**
on the menu **(menu table)**
and **how many times was it purchased** **COUNT (product_name)**
by all customers? **GROUP BY menu.product_name**



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

SELECT TOP 1
    menu.product_name,
    COUNT(menu.product_name) AS MOST_PURCHASED_ITEM
FROM
    menu
JOIN
    sales
ON
    menu.product_id = sales.product_id
GROUP BY
    menu.product_name
ORDER BY
    MOST_PURCHASED_ITEM DESC
```

	product_name	MOST_PURCHASED_ITEM
1	ramen	8

EXPLANATION:

- Used **JOIN** on the 2 tables sales and menu, then selected **product_name** from menu table

- Used **SELECT TOP 1** to get the top 1 result (LIMIT doesn’t work for Microsoft SQL)

- Performed 2 operations with the column: **product_name**
  1st operation = Selected the column normally
  2nd operation = used a function (COUNT) to get the quantity.

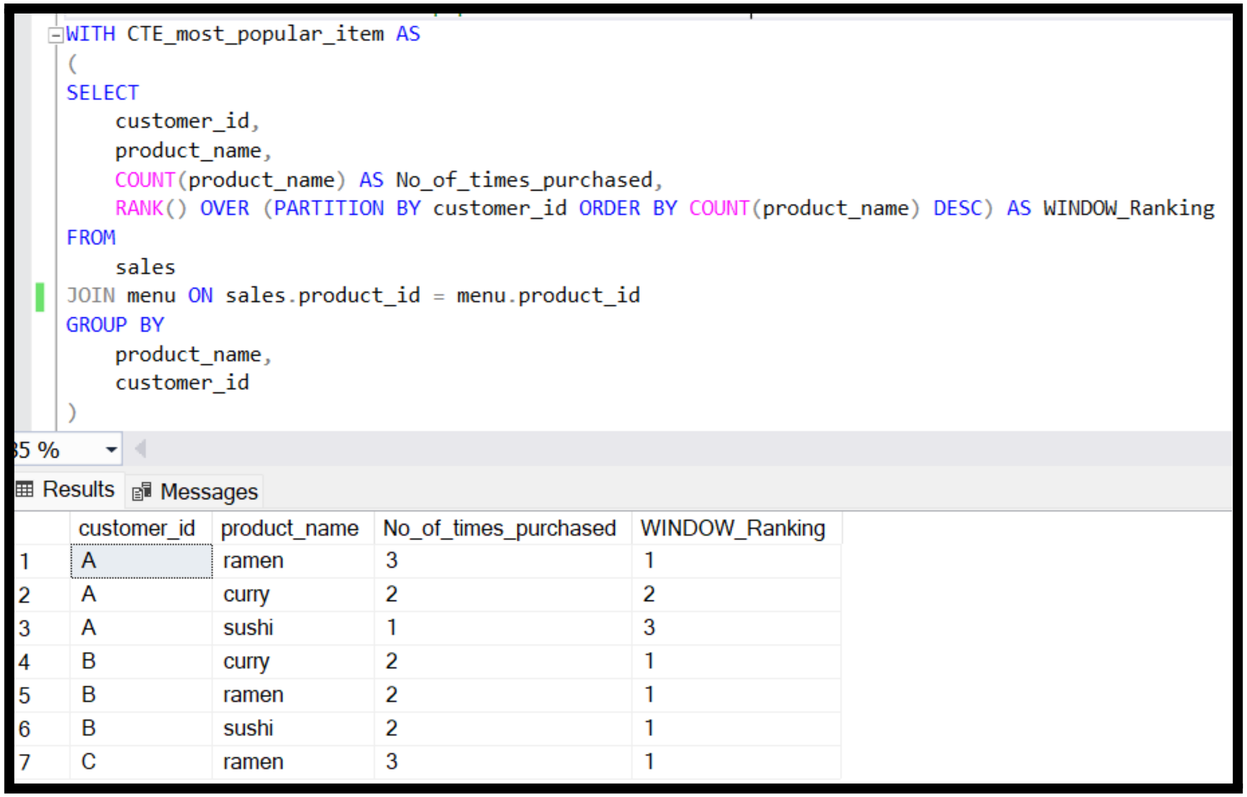
The most purchased item on the menu was ramen.
It was purchased 8 times.

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

Which item was the most popular (product_name)
for each customer? (PARTITION BY customer_id)

Most popular = most purchased item? **COUNT (product_name)**

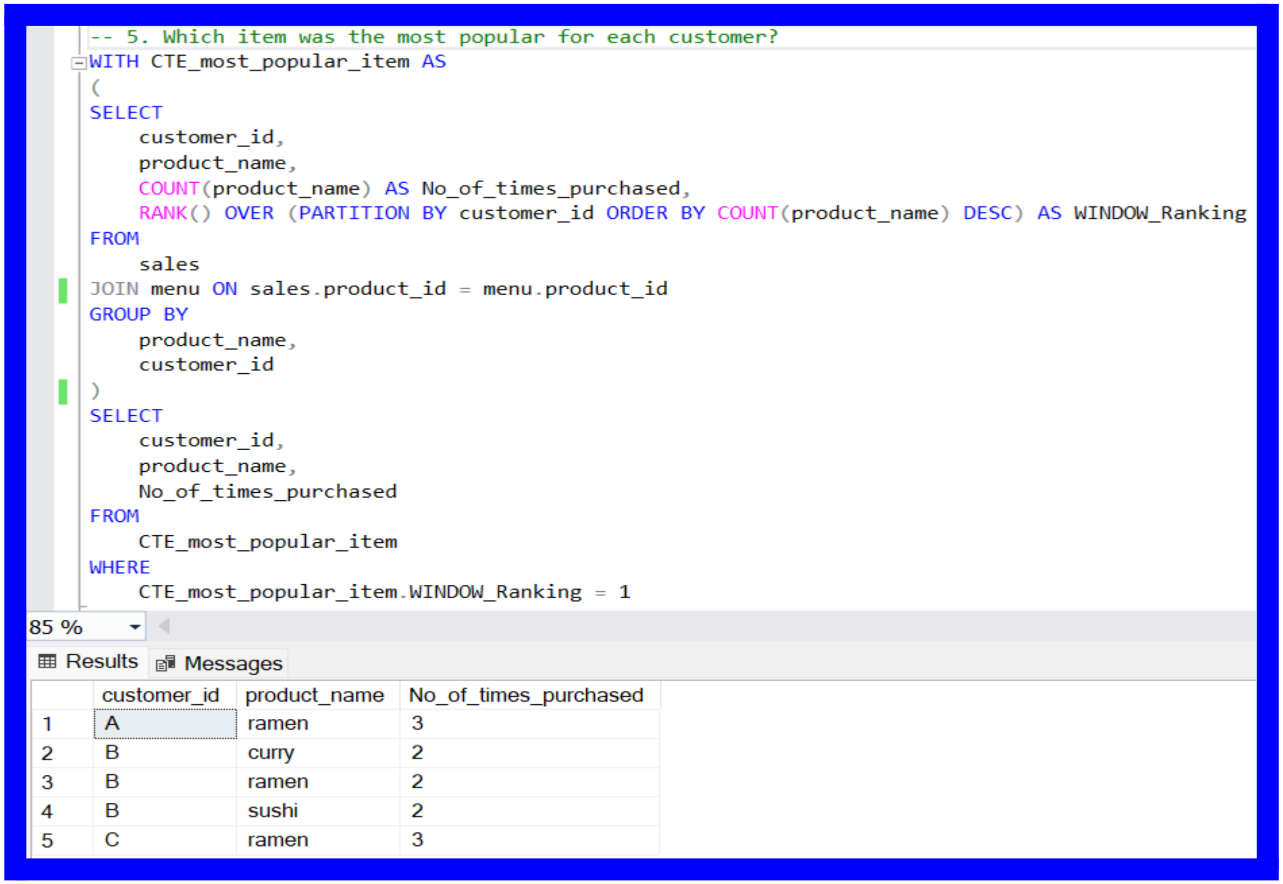
WITH CTE_most_popular_item AS



```
WITH CTE_most_popular_item AS
(
SELECT
    customer_id,
    product_name,
    COUNT(product_name) AS No_of_times_purchased,
    RANK() OVER (PARTITION BY customer_id ORDER BY COUNT(product_name) DESC) AS WINDOW_Ranking
FROM
    sales
JOIN menu ON sales.product_id = menu.product_id
GROUP BY
    product_name,
    customer_id
)
```

	customer_id	product_name	No_of_times_purchased	WINDOW_Ranking
1	A	ramen	3	1
2	A	curry	2	2
3	A	sushi	1	3
4	B	curry	2	1
5	B	ramen	2	1
6	B	sushi	2	1
7	C	ramen	3	1

ANSWER:



```
-- 5. Which item was the most popular for each customer?
WITH CTE_most_popular_item AS
(
SELECT
    customer_id,
    product_name,
    COUNT(product_name) AS No_of_times_purchased,
    RANK() OVER (PARTITION BY customer_id ORDER BY COUNT(product_name) DESC) AS WINDOW_Ranking
FROM
    sales
JOIN menu ON sales.product_id = menu.product_id
GROUP BY
    product_name,
    customer_id
)
SELECT
    customer_id,
    product_name,
    No_of_times_purchased
FROM
    CTE_most_popular_item
WHERE
    CTE_most_popular_item.WINDOW_Ranking = 1
```

```
   customer_id  product_name  No_of_times_purchased
1  A            ramen         3
2  B            curry         2
3  B            ramen         2
4  B            sushi         2
5  C            ramen         3
```

EXPLANATION:

- Made a CTE ranking all of the items based on the No. of times it was purchased (how popular it is)
- Used RANK not ROW_NUMBER since it would be more useful in this situation
  (Especially when it comes to Customer B, who has purchased 3 different items the same no. of times = Same rank)

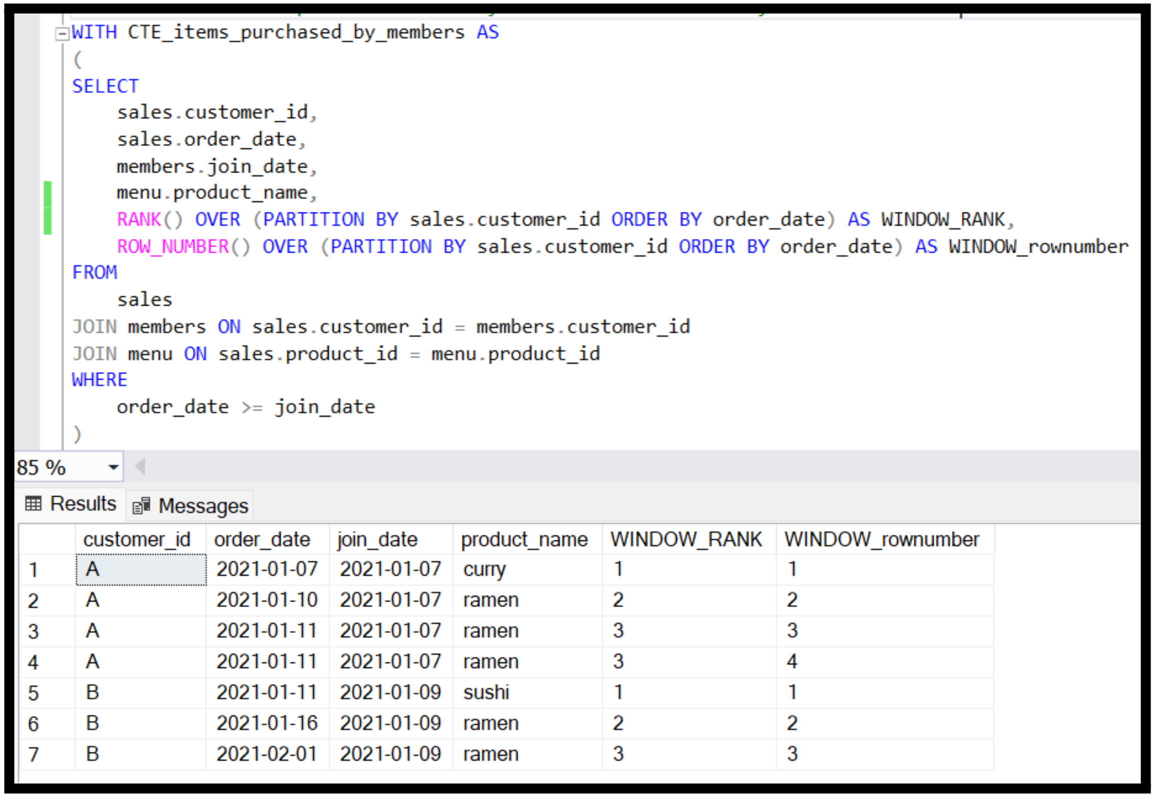
The most popular item for each customer:

- **Customer A: ramen**
- **Customer B: curry, ramen, and sushi**
- **Customer C: ramen**

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

- JOINED the 3 tables, since we need info from all 3 of them
- Then, compared the difference between RANK and ROW_NUMBER

WITH CTE_items_purchased_by_members AS



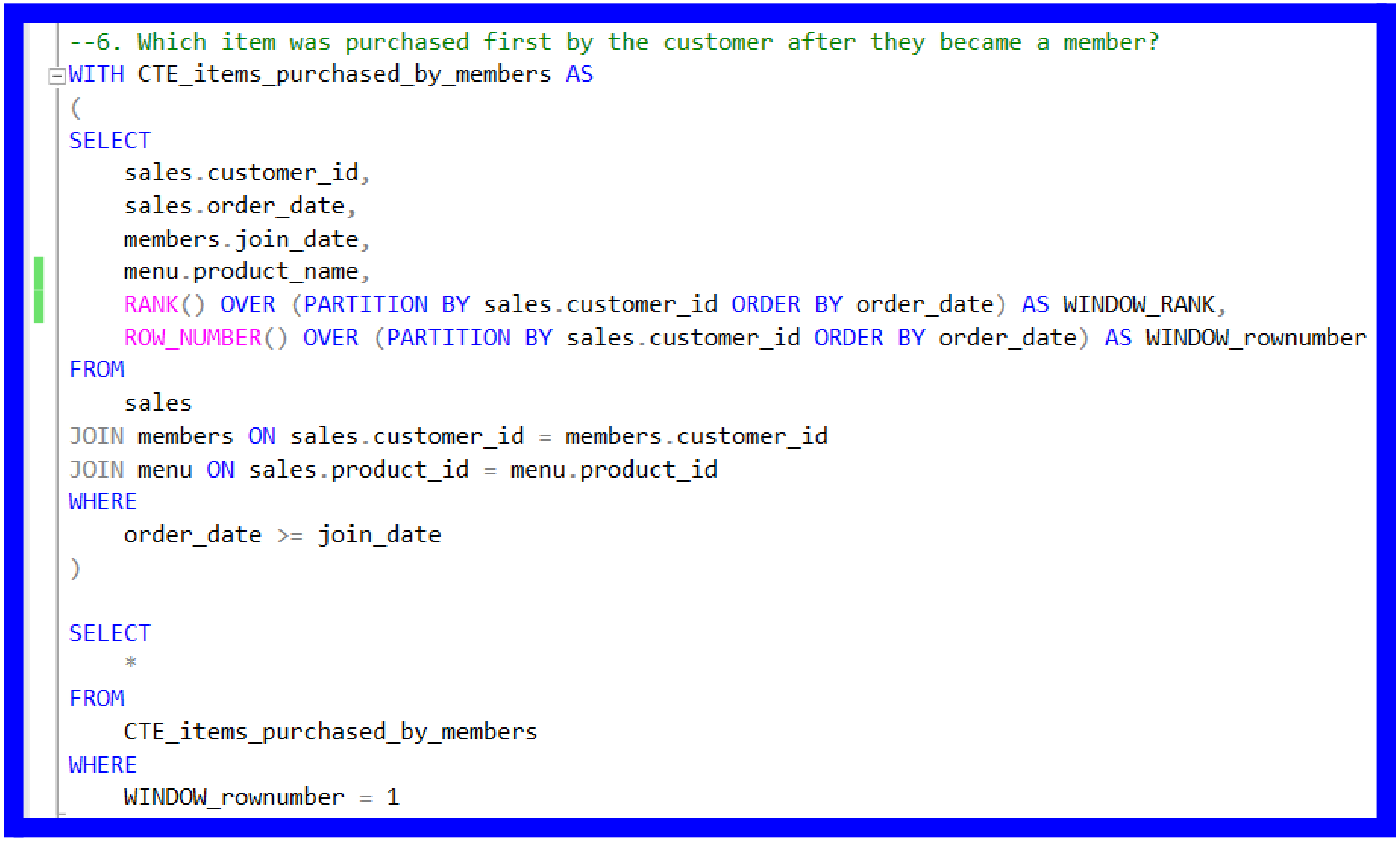
```
WITH CTE_items_purchased_by_members AS
(
SELECT
    sales.customer_id,
    sales.order_date,
    members.join_date,
    menu.product_name,
    RANK() OVER (PARTITION BY sales.customer_id ORDER BY order_date) AS WINDOW_RANK,
    ROW_NUMBER() OVER (PARTITION BY sales.customer_id ORDER BY order_date) AS WINDOW_rownumber
FROM
    sales
JOIN members ON sales.customer_id = members.customer_id
JOIN menu ON sales.product_id = menu.product_id
WHERE
    order_date >= join_date
)
```

	customer_id	order_date	join_date	product_name	WINDOW_RANK	WINDOW_rownumber
1	A	2021-01-07	2021-01-07	curry	1	1
2	A	2021-01-10	2021-01-07	ramen	2	2
3	A	2021-01-11	2021-01-07	ramen	3	3
4	A	2021-01-11	2021-01-07	ramen	3	4
5	B	2021-01-11	2021-01-09	sushi	1	1
6	B	2021-01-16	2021-01-09	ramen	2	2
7	B	2021-02-01	2021-01-09	ramen	3	3

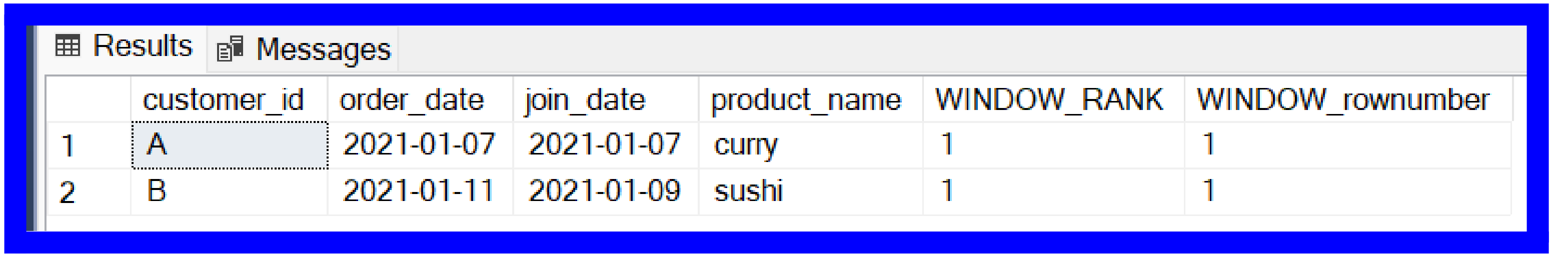
Applied **WHERE** clause:
order_date is **GREATER THAN** or **EQUAL TO** join_date
= **to find out which item was purchased first by the customer after becoming a member**

- SELECTED everything FROM **CTE_items_purchased_by_members**
- WHERE WINDOW_rownumber = 1

(Since they are divided by PARTITION BY sales.customer_id, it should only show the first result per PARTITION,
In this case, the first item purchased per customer)



```
--6. Which item was purchased first by the customer after they became a member?
WITH CTE_items_purchased_by_members AS
(
SELECT
    sales.customer_id,
    sales.order_date,
    members.join_date,
    menu.product_name,
    RANK() OVER (PARTITION BY sales.customer_id ORDER BY order_date) AS WINDOW_RANK,
    ROW_NUMBER() OVER (PARTITION BY sales.customer_id ORDER BY order_date) AS WINDOW_rownumber
FROM
    sales
JOIN members ON sales.customer_id = members.customer_id
JOIN menu ON sales.product_id = menu.product_id
WHERE
    order_date >= join_date
)

SELECT
    *
FROM
    CTE_items_purchased_by_members
WHERE
    WINDOW_rownumber = 1
```



	customer_id	order_date	join_date	product_name	WINDOW_RANK	WINDOW_rownumber
1	A	2021-01-07	2021-01-07	curry	1	1
2	B	2021-01-11	2021-01-09	sushi	1	1

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

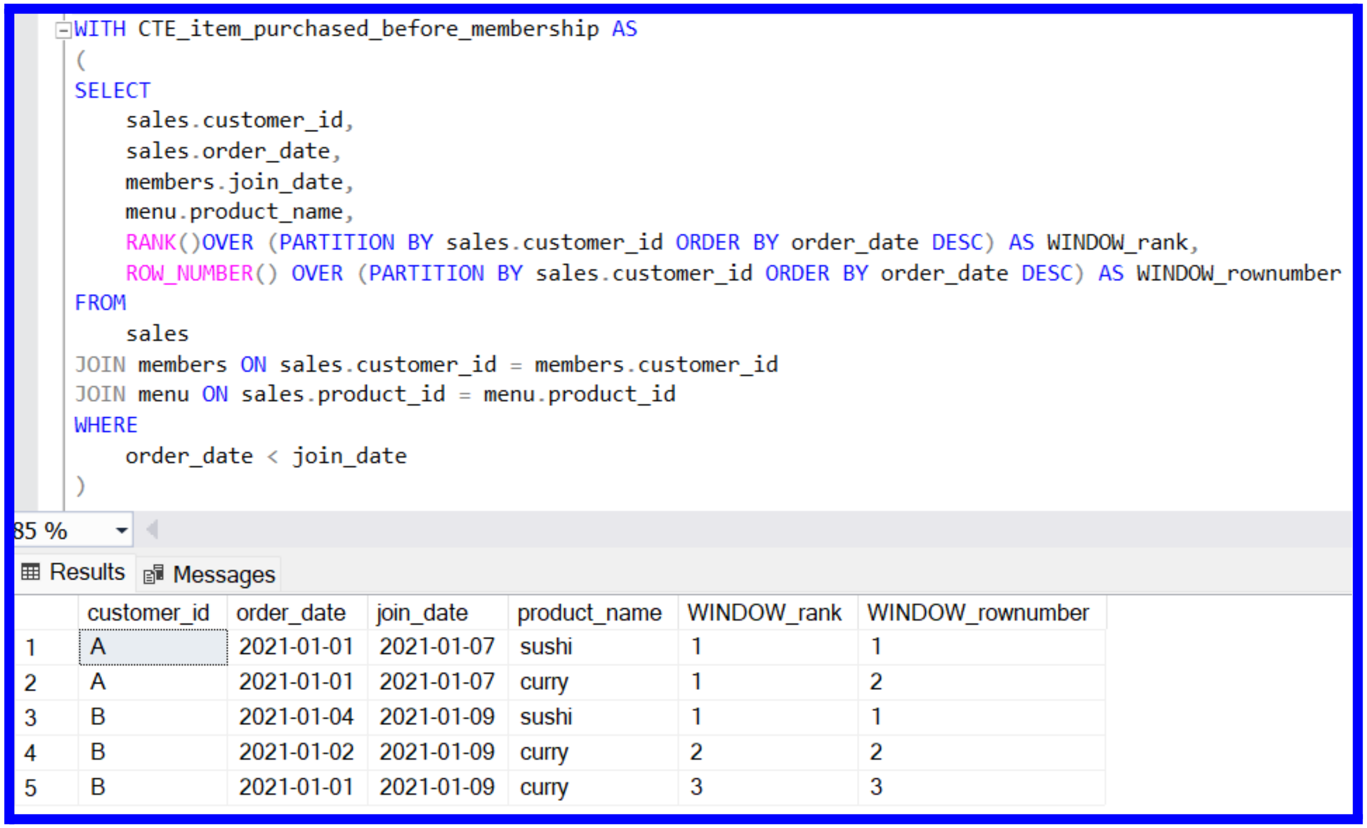
- **Customer A: curry**
- **Customer B: sushi**

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

Quite similar to the earlier question, so we'll just copy and edit a few details.

Changed the filter WHERE clause to:
WHERE order_date is less than join_date
(to only find the orders the customer made before becoming a member)

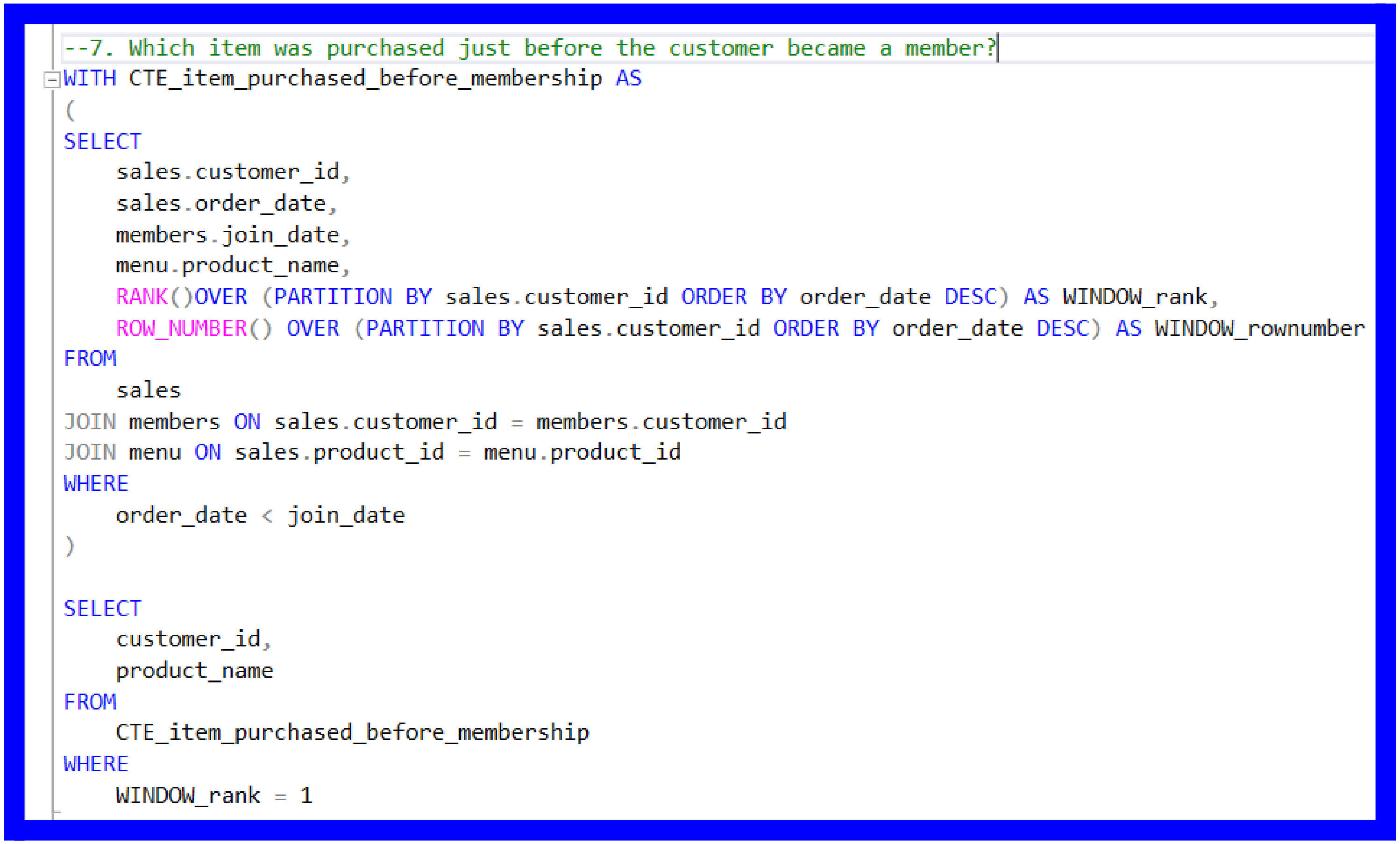
WITH CTE_item_purchased_before_membership AS



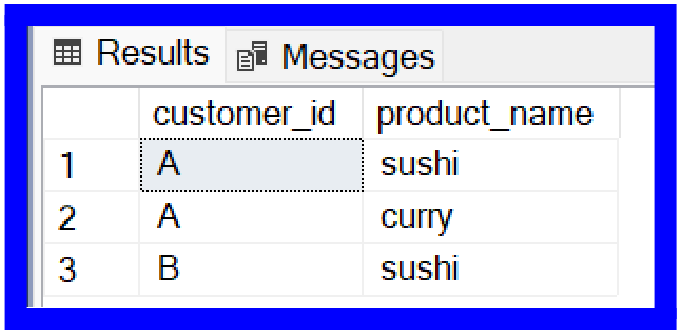
```
WITH CTE_item_purchased_before_membership AS
(
SELECT
    sales.customer_id,
    sales.order_date,
    members.join_date,
    menu.product_name,
    RANK()OVER (PARTITION BY sales.customer_id ORDER BY order_date DESC) AS WINDOW_rank,
    ROW_NUMBER() OVER (PARTITION BY sales.customer_id ORDER BY order_date DESC) AS WINDOW_rownumber
FROM
    sales
JOIN members ON sales.customer_id = members.customer_id
JOIN menu ON sales.product_id = menu.product_id
WHERE
    order_date < join_date
)
```

	customer_id	order_date	join_date	product_name	WINDOW_rank	WINDOW_rownumber
1	A	2021-01-01	2021-01-07	sushi	1	1
2	A	2021-01-01	2021-01-07	curry	1	2
3	B	2021-01-04	2021-01-09	sushi	1	1
4	B	2021-01-02	2021-01-09	curry	2	2
5	B	2021-01-01	2021-01-09	curry	3	3

This time we included the RANK Window function for comparison



```sql
--7. Which item was purchased just before the customer became a member?
WITH CTE_item_purchased_before_membership AS
(
SELECT
    sales.customer_id,
    sales.order_date,
    members.join_date,
    menu.product_name,
    RANK()OVER (PARTITION BY sales.customer_id ORDER BY order_date DESC) AS WINDOW_rank,
    ROW_NUMBER() OVER (PARTITION BY sales.customer_id ORDER BY order_date DESC) AS WINDOW_rownumber
FROM
    sales
JOIN members ON sales.customer_id = members.customer_id
JOIN menu ON sales.product_id = menu.product_id
WHERE
    order_date < join_date
)

SELECT
    customer_id,
    product_name
FROM
    CTE_item_purchased_before_membership
WHERE
    WINDOW_rank = 1
```



	customer_id	product_name
1	A	sushi
2	A	curry
3	B	sushi

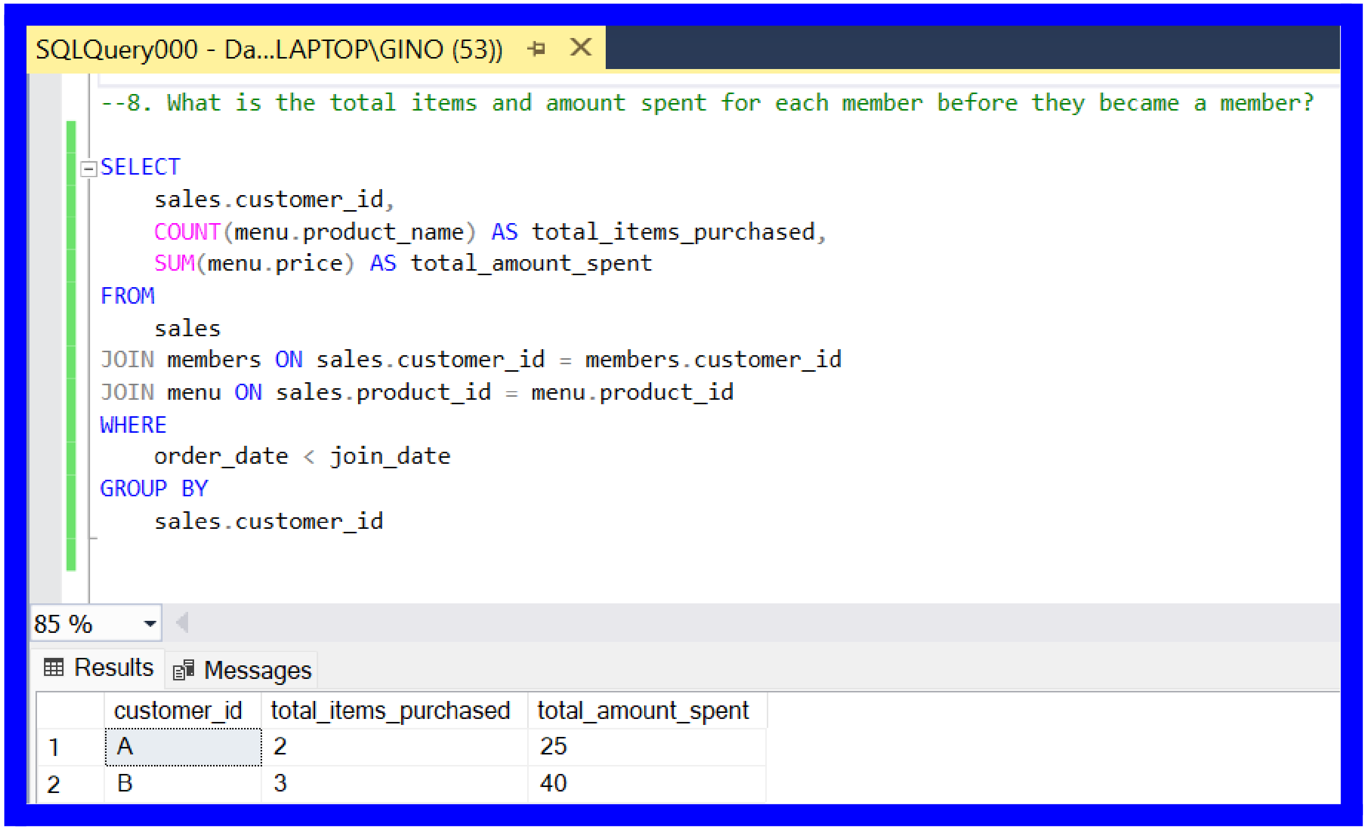
Which item was purchased just before the customer became a member?

- **Customer A: sushi & curry**
- **Customer B: sushi**

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

What is the total items **COUNT(menu.product_name)**
And total amount spent **SUM(menu.price)**
for each member **customer_id**
before they became a member? **order_date < join_date**

COUNT = Always answers the questions: How many? What is the total?



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

SELECT
    sales.customer_id,
    COUNT(menu.product_name) AS total_items_purchased,
    SUM(menu.price) AS total_amount_spent
FROM
    sales
JOIN members ON sales.customer_id = members.customer_id
JOIN menu ON sales.product_id = menu.product_id
WHERE
    order_date < join_date
GROUP BY
    sales.customer_id
```

	customer_id	total_items_purchased	total_amount_spent
1	A	2	25
2	B	3	40

EXPLANATION:

- We just copy pasted the query from question no. 7 since they are very similar,
- Removed the window functions and some unnecessary columns,
- Used COUNT and SUM Aggregate functions and GROUPED BY customer_id.

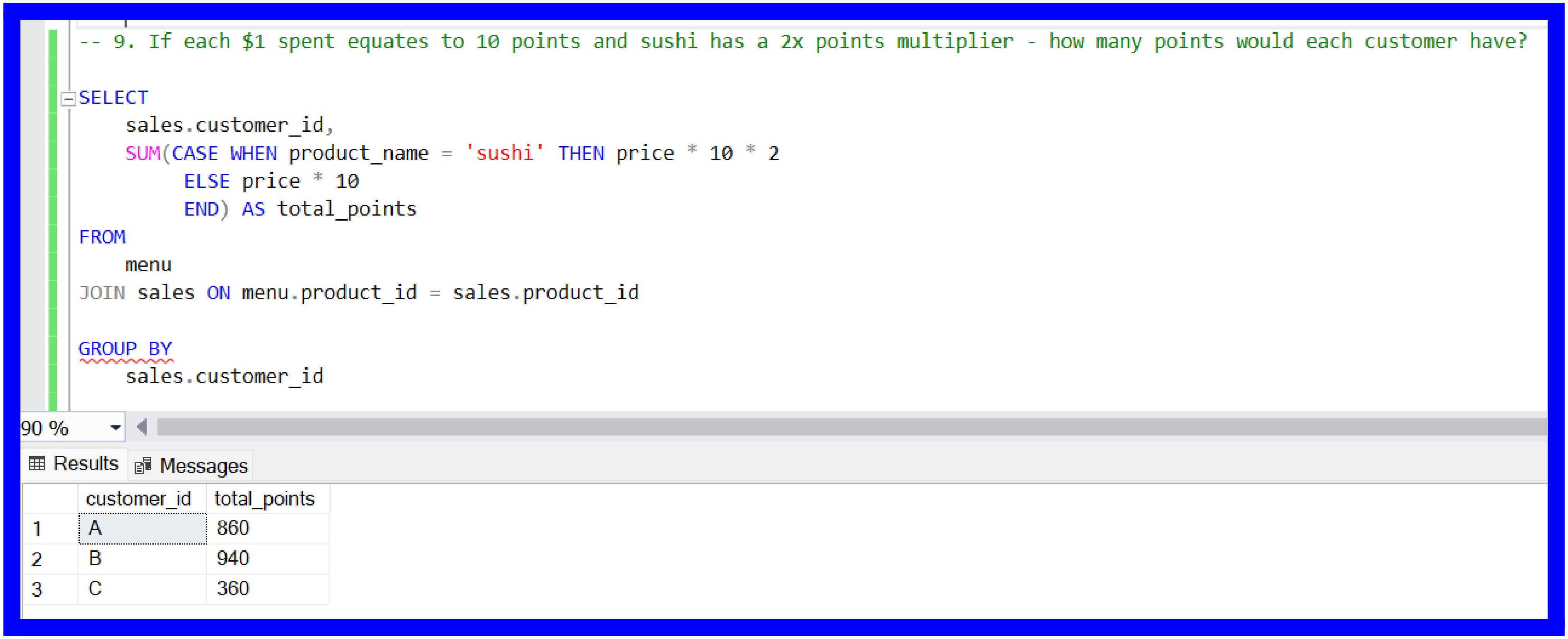
9. If each $1 spent equates to 10 points and sushi has a 2x points multiplier - how many points would each customer have?

We'll start with a SELECT * FROM menu
JOIN menu table with sales table

Then use a CASE statement:

- **WHEN** the product name is sushi = **price** multiply by **10** ($1 = 10 points), **THEN** multiply by **2** (2x pts multiplier)
- Everything **ELSE** = **price** multiply by **10** ($1 = 10 points)

You can use Aggregate functions (SUM, COUNT, etc) on CASE statements.



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

SELECT
    sales.customer_id,
    SUM(CASE WHEN product_name = 'sushi' THEN price * 10 * 2
         ELSE price * 10
         END) AS total_points
FROM
    menu
JOIN sales ON menu.product_id = sales.product_id

GROUP BY
    sales.customer_id
```

	customer_id	total_points
1	A	860
2	B	940
3	C	360

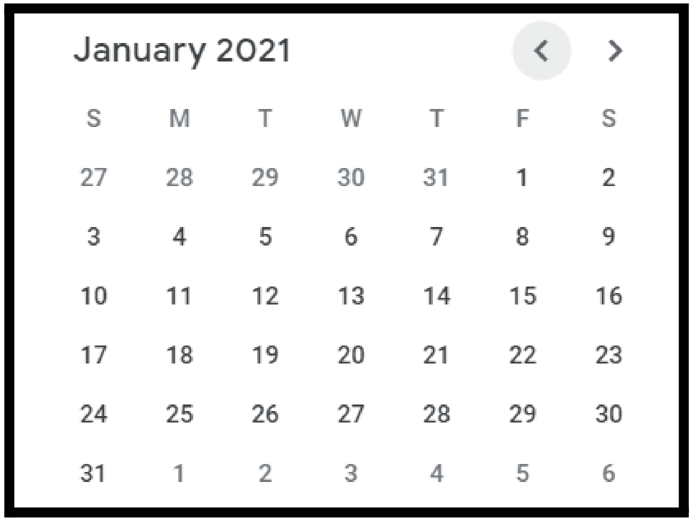
10. **In the first week after a customer joins the program (including their join date) they earn 2x points on all items, not just sushi - how many points do customer A and B have at the end of January?**

1st week = 2x points
Join date = Jan 07 and Jan 09
End = January 31

DATEADD function

- Customer A = Jan. 07-14 (2x)
  Jan. 15 - 31 (normal)

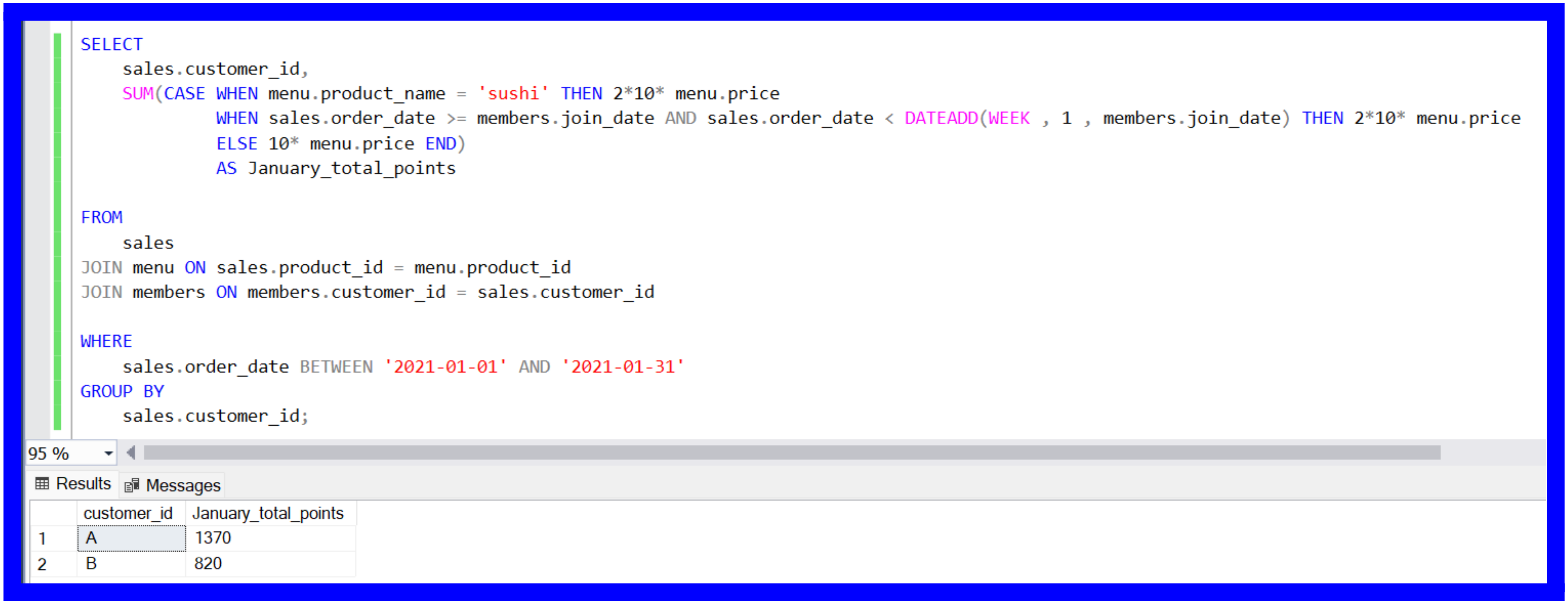
- Customer B = Jan. 09-16 (2x)
  Jan. 17 - 31 (normal)



January 2021

S	M	T	W	T	F	S
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

CASE statements + DATEADD



```
SELECT
    sales.customer_id,
    SUM(CASE WHEN menu.product_name = 'sushi' THEN 2*10* menu.price
             WHEN sales.order_date >= members.join_date AND sales.order_date < DATEADD(WEEK , 1 , members.join_date) THEN 2*10* menu.price
             ELSE 10* menu.price END)
             AS January_total_points

FROM
    sales
JOIN menu ON sales.product_id = menu.product_id
JOIN members ON members.customer_id = sales.customer_id

WHERE
    sales.order_date BETWEEN '2021-01-01' AND '2021-01-31'
GROUP BY
    sales.customer_id;
```

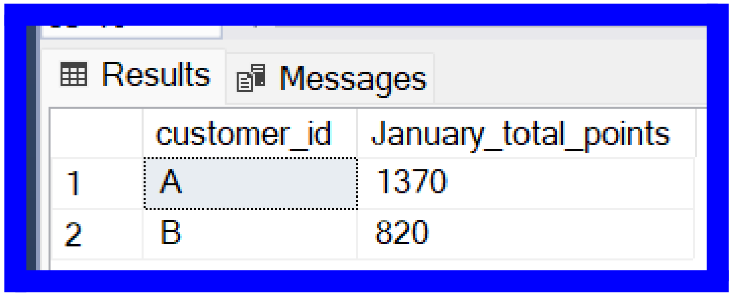
	customer_id	January_total_points
1	A	1370
2	B	820

CASE statement explanation:

- **WHEN** menu.product_name is sushi, **THEN 2** multiply by **10** multiply by **menu.price**

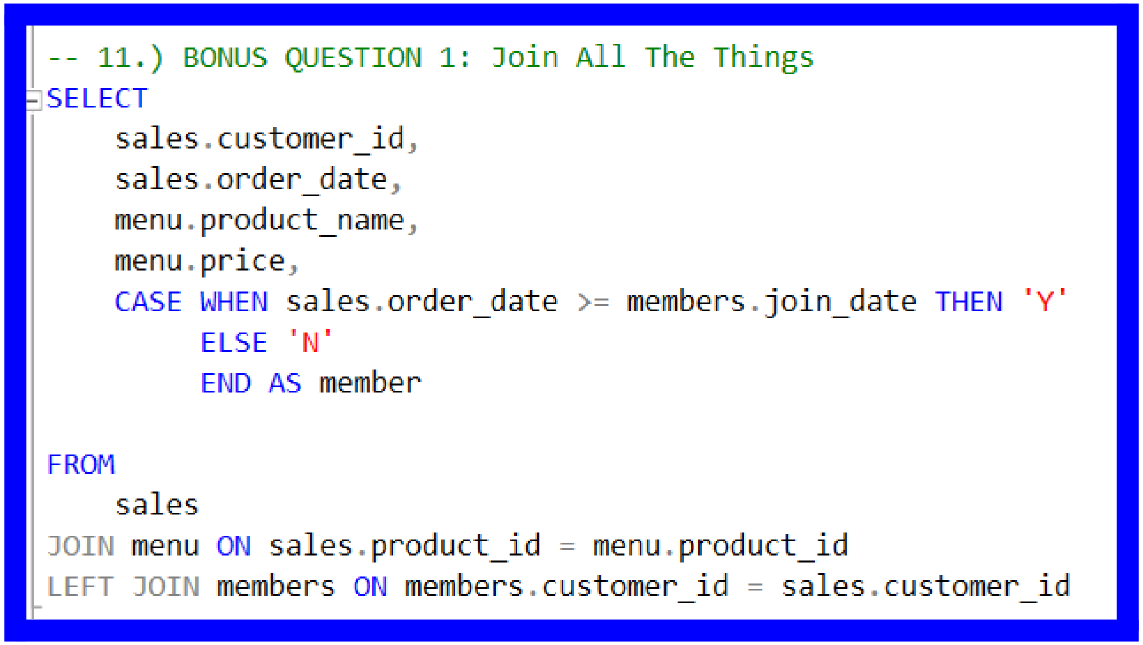
- **WHEN** sales.order_date **>= {is greater than or equal to}** members.join_date
  AND
  sales.order_date is **< DATEADD (WEEK, 1, members.join_date) {Less than 1 week of members join date}**
  THEN 2 multiply by **10** multiply by **menu.price**

- Everything **ELSE** = **10** multiply by **menu.price**
- Used the Aggregate function SUM on the CASE statement result to get the January_total_points

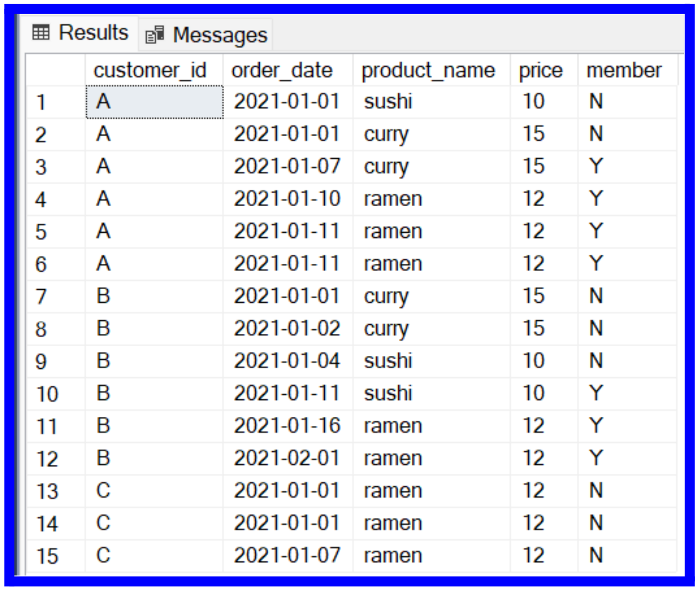


	customer_id	January_total_points
1	A	1370
2	B	820

BONUS QUESTION #1 - Join All The Things



```sql
-- 11.) BONUS QUESTION 1: Join All The Things
SELECT
    sales.customer_id,
    sales.order_date,
    menu.product_name,
    menu.price,
    CASE WHEN sales.order_date >= members.join_date THEN 'Y'
         ELSE 'N'
         END AS member

FROM
    sales
JOIN menu ON sales.product_id = menu.product_id
LEFT JOIN members ON members.customer_id = sales.customer_id
```

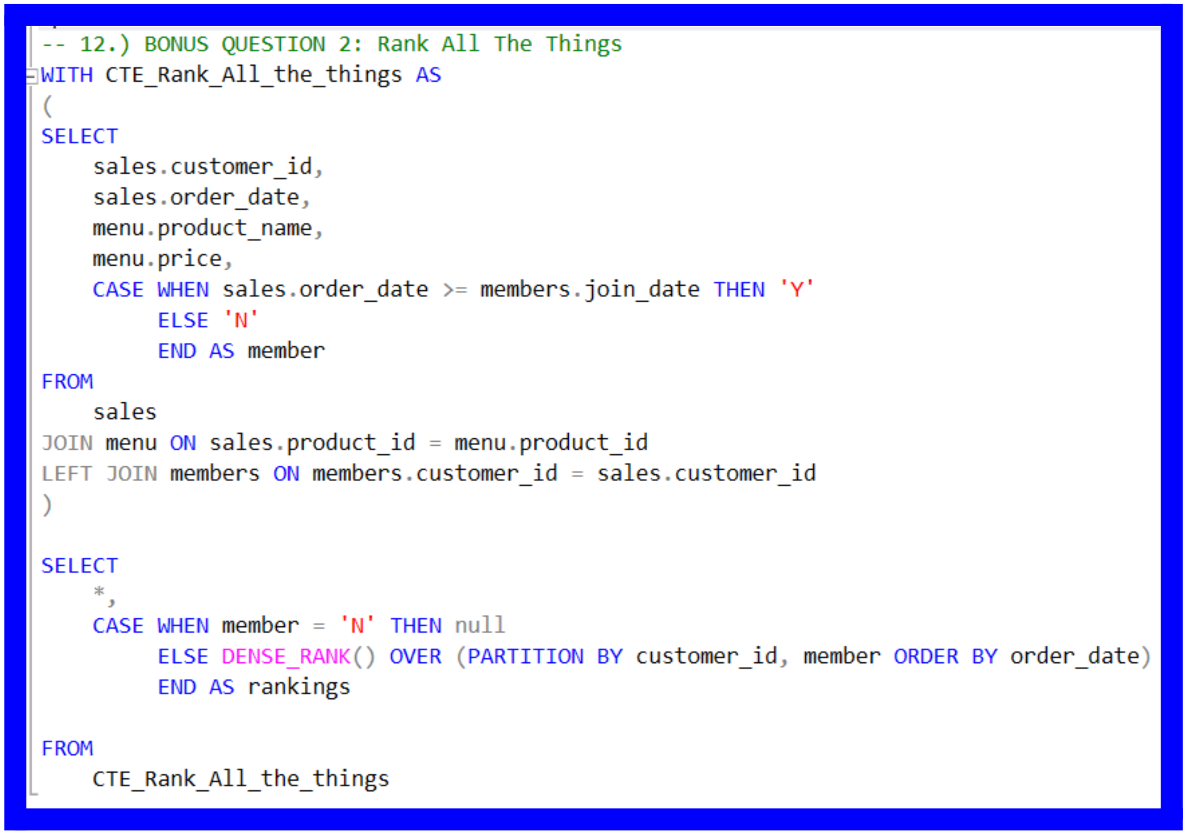


	customer_id	order_date	product_name	price	member
1	A	2021-01-01	sushi	10	N
2	A	2021-01-01	curry	15	N
3	A	2021-01-07	curry	15	Y
4	A	2021-01-10	ramen	12	Y
5	A	2021-01-11	ramen	12	Y
6	A	2021-01-11	ramen	12	Y
7	B	2021-01-01	curry	15	N
8	B	2021-01-02	curry	15	N
9	B	2021-01-04	sushi	10	N
10	B	2021-01-11	sushi	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

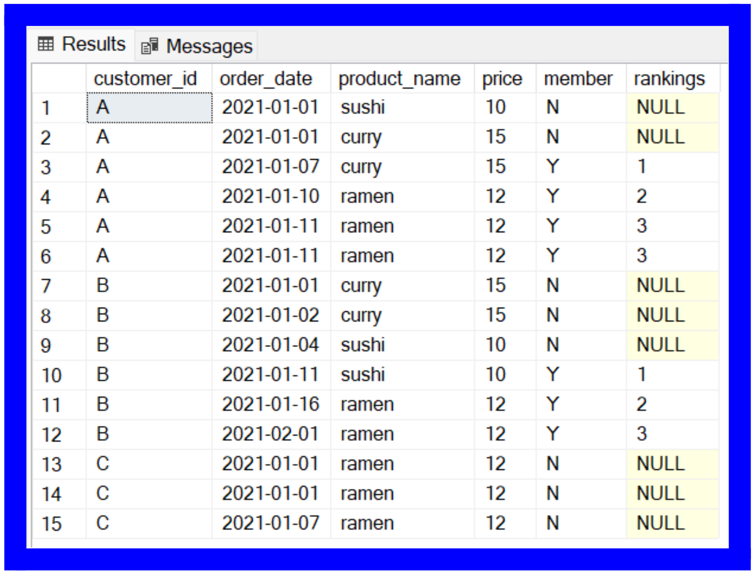
We have to use LEFT JOIN on members and sales table
In order to include customer C, because he/she doesn’t appear using INNER JOIN,
since he/she never became a member.

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.**



```
-- 12.) BONUS QUESTION 2: Rank All The Things
WITH CTE_Rank_All_the_things AS
(
SELECT
    sales.customer_id,
    sales.order_date,
    menu.product_name,
    menu.price,
    CASE WHEN sales.order_date >= members.join_date THEN 'Y'
         ELSE 'N'
         END AS member
FROM
    sales
JOIN menu ON sales.product_id = menu.product_id
LEFT JOIN members ON members.customer_id = sales.customer_id
)

SELECT
    *,
    CASE WHEN member = 'N' THEN null
         ELSE DENSE_RANK() OVER (PARTITION BY customer_id, member ORDER BY order_date)
         END AS rankings

FROM
    CTE_Rank_All_the_things
```



	customer_id	order_date	product_name	price	member	rankings
1	A	2021-01-01	sushi	10	N	NULL
2	A	2021-01-01	curry	15	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

- Copy pasted the Query from Bonus question #1 since they're almost the same then turned that query into CTE_Rank_All_the_things
- Created another CASE statement for our column named 'ranking'
- Used DENSE_RANK window function inside the CASE statement to find the rankings of the remaining rows that are not NULL.

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.**

(Danny's Diner)
Business insights after answering the Case study questions:

- **Customer A had spent the most amount of money (76)**
  followed shortly by Customer B (74)
  Customer C had spent less than half (36) of what Customer A or B spent.

- **Customer B visited Danny's Diner the most (6 days)**
  Customer A visited 4 days and Customer C visited 2 days

- **The most purchased item on the menu was ramen. (Best seller!)**
  It was purchased 8 items in January.

- **Ramen was the most popular item for customer A and C**
  while Customer B purchased sushi, curry, and ramen an equal amount of times.

- Before they became members, Customer A's last order was curry and sushi, Customer B's last order was sushi. Ramen may be the most popular/purchased item on the menu, but it seems like sushi was one of the reasons they signed up for a membership.

- Customer A had the most points for the month of January (1,370 points - if we use the 2x pts multiplier),
  he/she was also the customer that spent the most amount of money.

Lessons learned:

- Break down the question first,
  you can find out which info/data you need (and don't need) once you've analyzed the question.

- Learned a lot of things, Started naming my CTE and Window functions with CTE and WINDOW to easily recognize them. (Best practice!)

- Really helpful Case study. I struggled a lot, especially with the complicated queries, but I was able to practice a lot of the things I learned from the Google Data Analytics course when it comes to SQL, and was also able to practice the use of Window functions, CTE and CASE statements.

- CASE statements are really helpful.
  Mastering the use of CASE statements is a game changer in solving complicated queries.
  It's easier to solve complex queries with the combination of CASE statements, Window functions, and CTE.

- Begin with the end in mind - we first need to create a blueprint of the data/info we need to answer the question, so that constructing CTE's becomes a lot easier.

- There are multiple ways to solve the given problem.
  Just like with the previous questions, there are multiple solutions available,
  A data analyst can have a different approach and still arrive at the same conclusion.
  Example: In question #10, You can get the answer/their total points with the use of CTE, or without CTE.

LinkedIn: https://www.linkedin.com/in/ginofreudhobayan/
8 Week SQL Challenge: https://8weeksqlchallenge.com/

Thank you for reading this far! 🙂