DANNY'S DINER

INTRODUCTION

Danny seriously loves Japanese food so, at 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 its few months of operation but has no idea how to use its data to help 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.

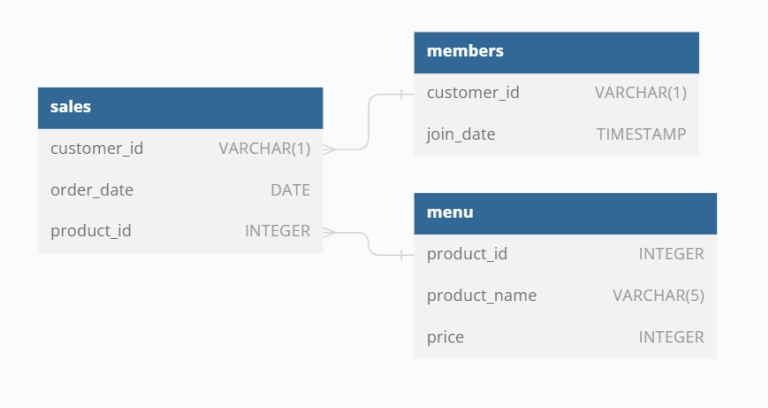
Danny has provided you with a sample of his overall customer data due to privacy issues - but he hopes that these examples are enough for you to write fully functioning SQL queries to help him answer his questions!

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

- Sales
- Menu

- Members

The entity relationship diagram is shown below:



I'll be using SQL to answer the following questions about the diner.

1. **What is the total amount each customer spent at the restaurant?**
   I joined the sales and menu tables and used the SUM aggregation to get the total amount spent. I grouped by customer id to get the sum for each customer and ordered the results in descending order of total amount.

```
SELECT sales.customer_id customer, SUM(menu.price) total_amount
FROM sales
INNER JOIN menu
ON sales.product_id = menu.product_id
GROUP BY 1
ORDER BY 2 DESC
```

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

From the results above, Customer A spent the most with a total of $76, followed closely by Customer B who had $74 and then Customer C with $36.

2. **How many days has each customer visited the restaurant?**
   I used the DISTINCT command together with the COUNT aggregation to get the distinct count of the order dates in the sales table. I grouped by the customer id to get the counts for each customer.

```
SELECT customer_id customer, COUNT(DISTINCT order_date) num_of_days
FROM sales
GROUP BY customer_id
```

	customer character varying (1)	num_of_days bigint
1	A	4
2	B	6
3	C	2

From the results above, Customer B visited the most (6 days) and Customer C visited the least (2 days).

3. **What was the first item from the menu purchased by each customer?**
   I used the DISTINCT ON command on the customer id and ordered by the customer id and ascending date. This would return the first item/entry in the sales tables corresponding to each customer when the table is ordered in ascending order of dates. Using a subquery wasn't giving me the results I wanted as Customer A ordered two items on the first day and I needed just the first item ordered on that day.

```
SELECT DISTINCT ON (sa.customer_id) sa.customer_id customer,
me.product_name
FROM sales sa
JOIN menu me
ON sa.product_id = me.product_id
GROUP BY sa.customer_id, sa.order_date, me.product_id, me.product_name
ORDER BY sa.customer_id, sa.order_date
```

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

From the results above, the first items ordered by Customer A, B, and C were sushi, curry and ramen respectively.

4. **What is the most purchased item on the menu and how many times was it purchased by all customers?**
   I joined the menu and sales table and used the COUNT aggregation on the product id. I grouped by product name and ordered in descending order of total quantity sold. I limited the results to show just the first entry (item with the highest count).

```
SELECT me.product_name, COUNT(sa.product_id) qty_sold
FROM menu me
JOIN sales sa
ON me.product_id = sa.product_id
GROUP BY 1
ORDER BY 2 DESC
LIMIT 1
```

	product_name character varying (5)	qty_sold bigint
1	ramen	8

From the result shown above, ramen was sold the most (8 times) and therefore is the most purchased item.

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

   I used the DISTINCT ON command on the customer id and ordered by descending count of product id. This allowed me to select the first entry (in this case the entry with the highest count) for each customer. I limited the results to get the first 3 results which would correspond to the highest counts for Customer A, B and C.

```
SELECT DISTINCT ON (sa.customer_id) sa.customer_id customer,
me.product_name, COUNT(sa.product_id) qty_sold
FROM menu me
JOIN sales sa
ON me.product_id = sa.product_id
GROUP BY 1, 2
ORDER BY 1, 3 DESC
LIMIT 3
```

	customer character varying (1)	product_name character varying (5)	qty_sold bigint
1	A	ramen	3
2	B	sushi	2
3	C	ramen	3

	customer character varying (1)	product_name character varying (5)	qty_sold bigint
1	B	ramen	2
2	B	curry	2
3	B	sushi	2

   From the first image above, it can be seen that Customer A and C both had ramen as their popular dish while Customer Bs was sushi but on further investigation (second picture), it was shown that Customer B ordered equal amounts of each item and had no particular preference.

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

   I joined the sales and members table and used the DISTINCT ON the customer id and ordered by ascending date. I also filtered the results to select the first item on the sales table where the order date was equal to or greater than the date the customer joined as a member.

```
SELECT DISTINCT ON (sa.customer_id) sa.customer_id, me.product_name
FROM menu me
JOIN sales sa
ON me.product_id = sa.product_id
JOIN members mem
ON sa.customer_id = mem.customer_id
WHERE sa.order_date >= mem.join_date
ORDER BY sa.customer_id, sa.order_date
```

	customer_id character varying (1)	product_name character varying (5)
1	A	curry
2	B	sushi

From the results above, Customer A and B purchased curry and sushi respectively. Customer C had no entry in the results as he/she is not a member.

7. **Which item was purchased just before the customer became a member?**
   In the FROM statement, I used a subquery that returned the customer id, product name and order date for order dates that were less than the join date of the corresponding customer. The outer query returned the customer id, product name and the order date. The DISTINCT ON command was also used to get the first entry in the table when arranged in descending order of order date. It was arranged in descending order because we needed the order that was placed just before gaining membership. Customer A and B's join dates were 2021-01-07 and 2021-01-09 respectively.

```
SELECT DISTINCT ON (customer_id) customer_id, product_name, order_date
FROM(
    SELECT sa.customer_id customer_id, me.product_name product_name,
sa.order_date order_date
    FROM menu me
    JOIN sales sa
    ON me.product_id = sa.product_id
    JOIN members mem
    ON sa.customer_id = mem.customer_id
    WHERE sa.order_date < mem.join_date
    GROUP BY sa.customer_id, me.product_name, sa.order_date
    ORDER BY sa.customer_id, sa.order_date DESC
) prior_membership_orders
ORDER BY customer_id
```

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

8. **What are the total items and amount spent for each member before they became a member?**
   I joined the 3 tables (sales, menu and members) and used the COUNT aggregation for the total items and the SUM aggregation for the total amount. I filtered for records that had order dates less than the customer's join date and grouped by the customer id.

```
SELECT sa.customer_id customer_id, COUNT(sa.product_id) total_items,
        SUM(me.price) total_amount
FROM menu me
JOIN sales sa
ON me.product_id = sa.product_id
JOIN members mem
```

```
ON sa.customer_id = mem.customer_id
WHERE sa.order_date < mem.join_date
GROUP BY 1
```

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

From the results above, it can be seen that Customer B purchased more items and spent more money than Customer A before becoming 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?**
   I joined the sales and menu tables and used the CASE statement to set conditions for points earned based on the given criteria and wrapped it in a SUM aggregation to get the total points earned by each customer. I grouped and ordered by the customer id.

```
SELECT s.customer_id customer,
SUM(
    CASE
        WHEN me.product_name = 'sushi' THEN me.price * 20
        ELSE me.price * 10
    END
) points
FROM sales s
JOIN menu me
ON s.product_id = me.product_id
GROUP BY s.customer_id
ORDER BY s.customer_id
```

	customer character varying (1)	points bigint
1	A	860
2	B	940
3	C	360

From the results, Customer A, B and C had 860, 940 and 360 points respectively.

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?**
    For this question, I used the same query from the last question but modified the CASE statement and added a condition before the other conditions that multiplies the price by 20 if the order date is within a week from the customer's join date (inclusive of the join date). I also filtered the results to include only orders made in January.

```
SELECT s.customer_id customer,
SUM(
    CASE
        WHEN s.order_date BETWEEN mem.join_date AND (mem.join_date + 6)
```

```
            THEN (me.price * 20)
            WHEN me.product_name = 'sushi' THEN (me.price * 20)
            ELSE me.price * 10
        END
    ) points
    FROM sales s
    JOIN members mem
    ON s.customer_id = mem.customer_id
    JOIN menu me
    ON s.product_id = me.product_id
    WHERE s.order_date >= '2021-01-01' AND s.order_date < '2021-02-01'
    GROUP BY s.customer_id
    ORDER BY s.customer_id
```

	customer character varying (1)	points bigint
1	A	1370
2	B	820

From the results above, Customer A (1370 points) gained more points than Customer B (820 points) in January.

Bonus Questions

For the bonus questions, I was required to recreate two tables. You can view the tables here.

1. **Join All the Things**

   I joined the three tables together and also selected the necessary columns. I utilized a LEFT JOIN so I could get both matched and unmatched records from the left table. I also used the CASE statement to create a column called member.

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

	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

2. Rank All the Things

I used the query from the previous question as a Common Table Expression (CTE) called joined and then selected the necessary columns from it. I also used a CASE statement along with the RANK window function to rank the data.

```
WITH joined AS (
    SELECT s.customer_id, s.order_date, me.product_name, me.price,
        CASE
            WHEN s.order_date >= mem.join_date THEN 'Y'
            ELSE 'N'
        END AS member
    FROM sales s
    LEFT JOIN menu me
    ON s.product_id = me.product_id
    LEFT JOIN members mem
    ON s.customer_id = mem.customer_id
    ORDER BY s.customer_id, s.order_date, me.product_name
)

SELECT customer_id, order_date, product_name, price, member,
    CASE
        WHEN member = 'Y' THEN RANK() OVER(
                                                PARTITION BY customer_id,
member
                                               ORDER BY order_date)
    END AS ranking
FROM joined
ORDER BY customer_id
```

	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]