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 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:

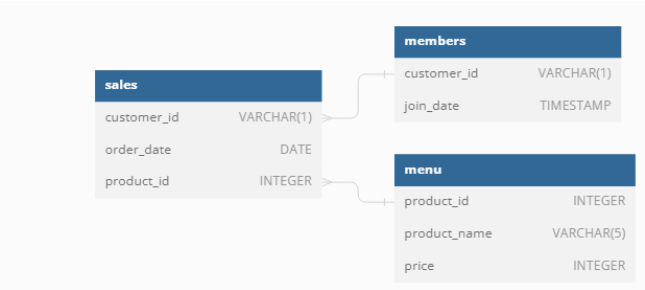
sales

menu

members

You can inspect the entity relationship diagram and example data below.

Entity Relationship Diagram



SOLUTION:

The dataset from Danny's Diner was managed using MySQL to address the technical questions. With the entity relationship diagram available as a data model, the initial step involved creating a database, necessary tables, and inputting the data into these tables using the infile function.

CREATE DATABASE

The database named dannys_diner was created and manage thus;

```
-- create database for dannys diner dataset
DROP DATABASE IF EXISTS dannys_diner;
CREATE DATABASE dannys_diner;

-- select the right database dannys_diner
use dannys_diner;
```

Create Tables and insert data into the tables

```
-- Create table name members in the dannys_diner database
CREATE TABLE members (
                      customer_id VARCHAR(50),
                      join_date DATE
                     );

-- insert data into the members table in dannys_diner database
INSERT INTO members(
                        customer_id,
                        join_date
                    )
VALUES
                    ('A', '2021-01-07'),
                    ('B', '2021-01-09');


-- Create table name menu in the dannys_diner database
CREATE TABLE menu (
                    product_id INTEGER,
                    product_name VARCHAR(50),
                    price INTEGER
                    );
```

```sql
-- insert data into the menu table in dannys_diner database
INSERT INTO menu(
                    product_id,
                    product_name,
                    price
                )
VALUES
              ('1', 'sushi', '10'),
              ('2', 'curry', '15'),
              ('3', 'ramen', '12');
```

```sql
-- create table name sales in the dannys_diner database
CREATE TABLE sales (
                    customer_id VARCHAR(50),
                    order_date DATE,
                    product_id INTEGER
                    );
```

```sql
-- insert data into the sales table in dannys_diner database
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'),
       ('B', '2021-02-01', '3'),
       ('C', '2021-01-01', '3'),
       ('C', '2021-01-01', '3'),
       ('C', '2021-01-07', '3');
```

Q1. What is the total amount each customer spent at the restaurant?

```
SELECT
        S.CUSTOMER_ID,
       SUM(MN.PRICE) AS 'AMOUNT_SPENT' -- Calculate the total amount spent for each customer
FROM
    MENU MN
RIGHT JOIN
            SALES S
ON
    MN.PRODUCT_ID = S.PRODUCT_ID -- Join the MENU and SALES tables on the PRODUCT_ID
GROUP BY
        CUSTOMER_ID; -- Group the results by CUSTOMER_ID to get the total amount spent for each customer
```

RESULT 1

	CUSTOMER_ID	AMOUNT_SPENT
▶	A	76
	B	74
	C	36

Q2. How many days has each customer visited the restaurant?

```
SELECT
        CUSTOMER_ID,
        COUNT(DISTINCT(ORDER_DATE)) AS TOTAL_DAYS  -- Count the number of unique order dates for each customer
FROM
    SALES  -- From the sales table
GROUP BY
        CUSTOMER_ID;  -- Group the results by customer ID
```

RESULT 2

	CUSTOMER_ID	TOTAL_DAYS
	A	4
	B	6
	C	2

Q3. What was the first item from the menu purchased by each customer?

```
WITH ITEM_FIRST_ORDER_CTE AS (
    -- Select customer ID, order date, product name, product ID, and calculate purchase rank
    SELECT
           S.CUSTOMER_ID,
           S.ORDER_DATE,
           MN.PRODUCT_NAME,
           S.PRODUCT_ID,
           -- Calculate the rank of each purchase per customer ordered by order date
           DENSE_RANK() OVER (PARTITION BY S.CUSTOMER_ID
                              ORDER BY S.ORDER_DATE) AS PURCHASE_RANK
    -- From the SALES table alias S
    FROM
        SALES S
    -- Inner join with the MENU table alias MN on the product ID
    INNER JOIN
                MENU MN
    ON
        S.PRODUCT_ID = MN.PRODUCT_ID
)
-- Select the customer ID, product name, and order date for the first purchase of each product by each customer
SELECT
        CUSTOMER_ID,
        PRODUCT_NAME,
        ORDER_DATE
-- From the CTE where the purchase rank is 1 (i.e., the first purchase)
FROM
        ITEM_FIRST_ORDER_CTE
WHERE
        PURCHASE_RANK = 1
-- Group by customer ID and product name to ensure unique combinations
GROUP BY
        CUSTOMER_ID,
        PRODUCT_NAME;
```

RESULT 3

	CUSTOMER_ID	PRODUCT_NAME	ORDER_DATE
▶	A	sushi	2021-01-01
	A	curry	2021-01-01
	B	curry	2021-01-01
	C	ramen	2021-01-01

Q4. What is the most purchased item on the menu and how many times was it purchased by all customers?

```
SELECT
        S.PRODUCT_ID,
        MN.PRODUCT_NAME,
        COUNT(S.PRODUCT_ID) AS SALES_COUNT
-- From the SALES table alias S
FROM
    SALES S
-- Inner join with the MENU table alias MN on the product ID
INNER JOIN
            MENU MN
ON
    MN.PRODUCT_ID = S.PRODUCT_ID
-- Group by product ID and product name to aggregate the sales count for each product
GROUP BY
        S.PRODUCT_ID, MN.PRODUCT_NAME
-- Order the results by sales count in descending order to get the most sold product first
ORDER BY
        SALES_COUNT DESC
-- Limit the results to the top 1 to get only the product with the highest sales count
LIMIT 1;
```

RESULT 4

	PRODUCT_ID	PRODUCT_NAME	SALES_COUNT
▶	3	ramen	8

Q5. Which item was the most popular for each customer?

```
WITH POPULAR_ITEM_CTE AS (
    -- Select customer ID, product name, count of product ID, and calculate order rank
    SELECT
           S.CUSTOMER_ID,
           MN.PRODUCT_NAME,
           COUNT(MN.PRODUCT_ID) AS ITEM_COUNT,
           -- Calculate the rank of each product per customer ordered by the count of product ID in descending order
           DENSE_RANK() OVER(PARTITION BY S.CUSTOMER_ID
                             ORDER BY COUNT(MN.PRODUCT_ID) DESC) AS ORDER_RANK
    -- From the SALES table alias S
    FROM
        SALES S
    -- Join with the MENU table alias MN on the product ID
    JOIN
        MENU MN
    ON
        S.PRODUCT_ID = MN.PRODUCT_ID
    -- Group by customer ID and product name to aggregate the item counts for each product per customer
    GROUP BY
            S.CUSTOMER_ID,
            MN.PRODUCT_NAME
)

-- Select the product name, customer ID, and item count for the most popular item of each customer
SELECT
        PRODUCT_NAME,
        CUSTOMER_ID,
        ITEM_COUNT
-- From the CTE where the order rank is 1 (i.e., the most frequently purchased item per customer)
FROM
    POPULAR_ITEM_CTE
WHERE
        ORDER_RANK = 1;
```

RESULT 5

	PRODUCT_NAME	CUSTOMER_ID	ITEM_COUNT
▶	ramen	A	3
	curry	B	2
	sushi	B	2
	ramen	B	2
	ramen	C	3

Q6. Which item was purchased first by the customer after they became a member?

```
WITH MEMBER_FIRST_PUR_CTE AS (
    -- Select customer ID, join date, product ID, product name, order date, and calculate the rank of each purchase
    SELECT
           S.CUSTOMER_ID,
           MM.JOIN_DATE,
           MN.PRODUCT_ID,
           MN.PRODUCT_NAME,
           S.ORDER_DATE,
           -- Calculate the rank of each purchase per customer ordered by order date
           DENSE_RANK() OVER (PARTITION BY S.CUSTOMER_ID
                              ORDER BY S.ORDER_DATE) AS MEMBER_FIRST_PUR_RANK
    -- From the MENU table alias MN
    FROM
        MENU MN
    -- Right join with the SALES table alias S on the product ID
    RIGHT JOIN
               SALES S
    ON
        MN.PRODUCT_ID = S.PRODUCT_ID
    -- Left join with the MEMBERS table alias MM on the customer ID
    LEFT JOIN
               MEMBERS MM
    ON
        S.CUSTOMER_ID = MM.CUSTOMER_ID
    -- Only include orders where the order date is after the member's join date
    WHERE
        S.ORDER_DATE > MM.JOIN_DATE
)

-- Select the product ID, product name, customer ID, and order date for the first purchase of each member after their join date
SELECT
       PRODUCT_ID,
       PRODUCT_NAME,
       JOIN_DATE,
       CUSTOMER_ID,
       ORDER_DATE
-- From the CTE where the purchase rank is 1 (i.e., the first purchase after the join date for each member)
FROM
    MEMBER_FIRST_PUR_CTE
WHERE
    MEMBER_FIRST_PUR_RANK = 1;
```

RESULT 6

	PRODUCT_ID	PRODUCT_NAME	JOIN_DATE	CUSTOMER_ID	ORDER_DATE
▶	3	ramen	2021-01-07	A	2021-01-10
	1	sushi	2021-01-09	B	2021-01-11

Q7. which menu item(s) was purchased just before the customer became a member and when?

```
WITH BEFORE_MEM_CTE AS (
    -- Select customer ID, order date, join date, product ID, and calculate the rank of each purchase
    SELECT
          S.CUSTOMER_ID,
          S.ORDER_DATE,
          MM.JOIN_DATE,
          S.PRODUCT_ID,
          -- Calculate the rank of each purchase per customer ordered by order date in descending order
          DENSE_RANK() OVER(PARTITION BY S.CUSTOMER_ID
                            ORDER BY S.ORDER_DATE DESC) AS ORDER_RANK
    -- From the SALES table alias S
    FROM
        SALES S
    -- Join with the MEMBERS table alias MM on the customer ID
    JOIN
        MEMBERS MM
    ON
        S.CUSTOMER_ID = MM.CUSTOMER_ID
    -- Only include orders where the order date is before the member's join date
    WHERE
        MM.JOIN_DATE > S.ORDER_DATE
)

-- Select the product name, order date, join date, order rank, and customer ID for the last purchase of each customer before they became a member
SELECT
       MN.PRODUCT_NAME,
       BM.ORDER_DATE,
       BM.JOIN_DATE,
       BM.ORDER_RANK,
       BM.CUSTOMER_ID
-- From the MENU table alias MN
FROM
    MENU MN
-- Join with the BEFORE_MEM_CTE alias BM on the product ID
JOIN
    BEFORE_MEM_CTE BM
ON
    MN.PRODUCT_ID = BM.PRODUCT_ID
-- Where the order rank is 1 (i.e., the most recent purchase before the join date for each customer)
WHERE
    BM.ORDER_RANK = 1;
```

RESULT 7

	PRODUCT_NAME	ORDER_DATE	JOIN_DATE	ORDER_RANK	CUSTOMER_ID
▶	sushi	2021-01-01	2021-01-07	1	A
	sushi	2021-01-04	2021-01-09	1	B
	curry	2021-01-01	2021-01-07	1	A

Q8. What is the total items and amount spent for each member before they became a member?

```
SELECT
        COUNT(S.PRODUCT_ID) AS PRODUCT_COUNT,   -- Count of products purchased
        SUM(MN.PRICE) AS TOTAL_SPENT,           -- Sum of the prices of the products purchased
        MM.JOIN_DATE,                           -- Join date of the member
        S.ORDER_DATE,                           -- Order date of the purchase
        S.CUSTOMER_ID                           -- Customer ID
-- From the MENU table alias MN
FROM
    MENU MN
-- Join with the SALES table alias S on the product ID
JOIN
    SALES S
ON
    MN.PRODUCT_ID = S.PRODUCT_ID
-- Join with the MEMBERS table alias MM on the customer ID
JOIN
    MEMBERS MM
ON
    MM.CUSTOMER_ID = S.CUSTOMER_ID
-- Only include orders where the order date is before the member's join date
WHERE
    S.ORDER_DATE < MM.JOIN_DATE
-- Group by customer ID to aggregate the results per customer
GROUP BY
        S.CUSTOMER_ID;
```

RESULT 8

	PRODUCT_COUNT	TOTAL_SPENT	JOIN_DATE	ORDER_DATE	CUSTOMER_ID
▶	3	40	2021-01-09	2021-01-04	B
	2	25	2021-01-07	2021-01-01	A

Q9. If each $1 spent equates to 10 points and sushi has a 2x points multiplier - how many points would each customer have?

```sql
WITH PRODUCT_POINT_CTE AS (
    -- Select customer ID and calculate price points and points acquired
    SELECT
        S.CUSTOMER_ID,
        SUM(MN.PRICE) AS PRICE_POINT,  -- Sum of the prices of the products purchased
        SUM(
            CASE
                WHEN MN.PRODUCT_ID = 1 THEN MN.PRICE * 20  -- If product ID is 1, multiply price by 20
                ELSE MN.PRICE * 10  -- For all other product IDs, multiply price by 10
            END
        ) AS POINTS_ACQUIRED  -- Calculate total points acquired
    -- From the MENU table alias MN
    FROM
        MENU MN
    -- Join with the SALES table alias S on the product ID
    JOIN
        SALES S
    ON
        MN.PRODUCT_ID = S.PRODUCT_ID
    -- Group by customer ID to aggregate the results per customer
    GROUP BY
            S.CUSTOMER_ID
)

-- Select the customer ID and points acquired for each customer
SELECT
        S.CUSTOMER_ID,
        PP.POINTS_ACQUIRED
-- From the PRODUCT_POINT_CTE alias PP
FROM
    PRODUCT_POINT_CTE PP
-- Join with the SALES table alias S on the customer ID
JOIN
    SALES S
ON
    S.CUSTOMER_ID = PP.CUSTOMER_ID
-- Group by customer ID to ensure unique results
GROUP BY
        S.CUSTOMER_ID,
        PP.POINTS_ACQUIRED;
```

RESULT 9

	CUSTOMER_ID	POINTS_ACQUIRED
▶	A	860
	B	940
	C	360

Q10. 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?

```
SELECT
    S.CUSTOMER_ID,
    -- Sum the points acquired based on the conditions specified
    SUM(
        CASE
            -- If the order date is within 6 days after the join date, multiply the price by 20
            WHEN S.ORDER_DATE BETWEEN MM.JOIN_DATE AND DATE_ADD(MM.JOIN_DATE, INTERVAL 6 DAY) THEN
                MN.PRICE * 20
            -- If the product ID is 1, multiply the price by 20
            WHEN MN.PRODUCT_ID = 1 THEN
                MN.PRICE * 20
            -- For all other cases, multiply the price by 10
            ELSE
                MN.PRICE * 10
        END
    ) AS TOTAL_POINTS
-- From the SALES table alias S
FROM
    SALES S
-- Join with the MENU table alias MN on the product ID
JOIN
    MENU MN
ON
    S.PRODUCT_ID = MN.PRODUCT_ID
-- Join with the MEMBERS table alias MM on the customer ID
JOIN
    MEMBERS MM
ON
    S.CUSTOMER_ID = MM.CUSTOMER_ID
-- Filter for specific customers and order dates up to a specific date
WHERE
    S.CUSTOMER_ID IN ('A', 'B')  -- Only include customers 'A' and 'B'
    AND S.ORDER_DATE <= '2021-01-31'  -- Only include orders up to '2021-01-31'
-- Group by customer ID to aggregate the results per customer
GROUP BY
    S.CUSTOMER_ID;
```

RESULT 10

	CUSTOMER_ID	TOTAL_POINTS
▶	B	820
	A	1370

Recommendations

By leveraging on the insight generated, Danny's diner can:

(1) Enhance menu offerings by incorporating popular items and pinpointing menu items that drive customer decisions.
(2) Boost loyalty with customized offers and incentives. Align marketing strategies with pre-membership purchase behavior.
(3) Adjust promotions based on customers' point accumulation and track the influence of point multipliers on customer behavior.
(4) Streamline operations by leveraging customer preferences insights and improve onboarding with customized promotions.
(5) Evaluate points earned during specific periods to assess program effectiveness and make data-driven decisions for future promotions.