Diner Insights

Danny, a fan of Japanese cuisine, took a leap of faith in early 2021 and opened a restaurant called Danny's Diner.

His menu features his three favorite dishes:

- Sushi
- Curry
- Ramen

However, Danny's Diner is facing challenges and seeks your help to navigate the complexities of running a restaurant.

Despite collecting some basic data during their short operational period, they lack the expertise to interpret and utilize this information effectively to improve their business performance.

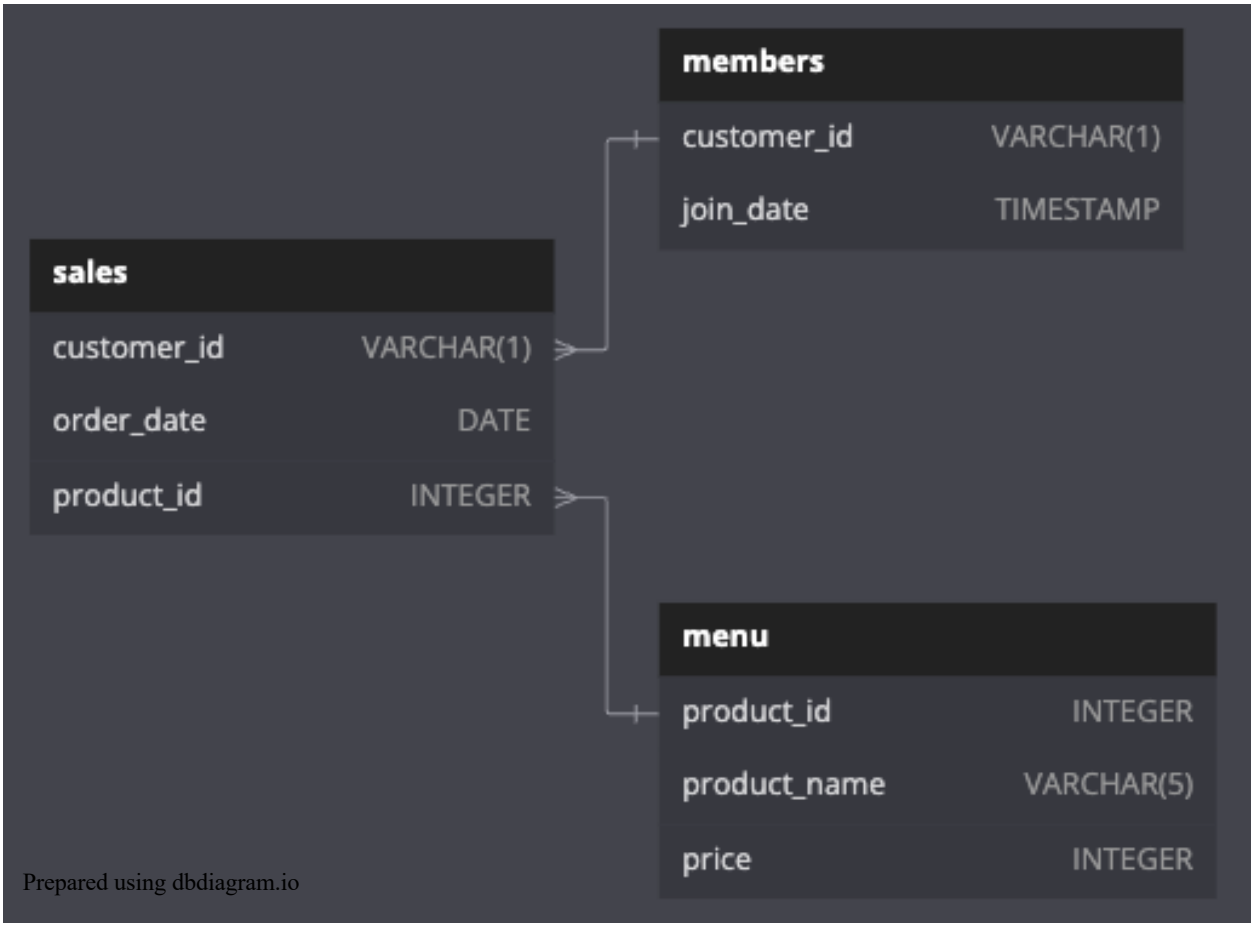
To help him deliver a better and more personalized experience for his loyal customers. We can answer a few simple questions about his customers, listed below, to help him with his business. Also, he intends to use these insights to help him decide whether to expand the existing customer loyalty program.

1. What is the total amount each customer spent at the restaurant?
2. How many days has each customer visited the restaurant?
3. What was the first item from the menu purchased by each customer?
4. What is the most purchased item on the menu and how many times was it purchased by all customers?
5. Which item was the most popular for each customer?
6. Which item was purchased first by the customer after they became a member?
7. Which item was purchased just before the customer became a member?
8. What are the total items and amount spent for each member 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?
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?
11. Join the tables above to create some basic datasets that his team can easily examine without resorting to SQL.
12. Danny needs product rankings for loyalty program members.

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

- Sales
- Menu
- Members

Entity Relationship diagram:



Dataset schema: dannys_diner

Menu Table

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

Members Table

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

Sales Table

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

Solution Link: https://www.db-fiddle.com/f/2rM8RAnq7h5LLDTzZiRWcd/8791

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

customer_id	total_spent
A	76
B	74
C	36

Solution:

- Use an aggregate query to sum the prices of each product ordered by customers.
- Join the `sales` and `menu` tables on `product_id` to calculate the total for each customer.
- Group the results by `customer_id` to get the total amount spent by each customer.

Key Points:

- **Aggregation:** `SUM()` is used to total the spent amount.
- **Join:** The `sales` and `menu` tables are joined based on `product_id`.
- **Grouping:** Group by `customer_id` to calculate totals per customer.

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

customer_id	visit_count
A	4
B	6
C	2

Solution:

- Use `COUNT(DISTINCT)` to count the number of unique days each customer visited.
- Join the `sales` table with `members` to filter by customer visits.
- Group by `customer_id` to get the visit count for each customer.

Key Points:

- **Distinct Dates:** `COUNT(DISTINCT)` ensures counting only unique visit days.
- **Join:** The `sales` table is used to track the visit dates.
- **Grouping:** Group by `customer_id` for visits per customer.

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

customer_id	product_name	first_purchase_date
A	sushi	2021-01-01
B	curry	2021-01-01
C	ramen	2021-01-01

Solution:

- Use `DISTINCT ON` to select the first item ordered based on the earliest order date.
- Filter by `customer_id` and order by `order_date` to get the first purchase.

Key Points:

- **DISTINCT ON:** Ensures only the first purchase is selected for each customer.
- **Order Date:** Orders are sorted by `order_date` to get the earliest.
- **Selection:** Use `product_name` from the `menu` to identify the first item.

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

product_name	purchase_count
ramen	8

Solution:

- Aggregate the data with `COUNT()` to find the most purchased item.
- Group by `product_name` and order by the purchase count to get the most popular item.

Key Points:

- **COUNT:** Used to count the occurrences of each item purchased.
- **Grouping and Ordering:** Group by `product_name` and order by purchase count.
- **Most Popular:** The item with the highest count is selected.

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

customer_id	most_popular_item	most_popular_count
A	ramen	3
B	ramen	2
B	curry	2
B	sushi	2
C	ramen	3

Solution:

- Use `COUNT()` and `RANK()` to rank the products based on how often they were purchased by each customer.
- Partition by `customer_id` and order by purchase count to find the most popular item.

Key Points:

- **COUNT:** Calculates how many times each item was purchased.
- **RANK:** Used to rank the most purchased item per customer.
- **Partition:** Ensures ranking happens per customer, not globally.

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

customer_id	product_name
A	ramen
B	sushi

Solution:

- Filter the sales data where the `order_date` is after the `join_date` for each customer.
- Use `ROW_NUMBER()` to rank purchases by date and select the first product purchased.

Key Points:

- **Filter:** Only consider purchases made after the customer joined the program.
- **ROW_NUMBER:** Ranks the purchases by date to get the first item.
- **Joining:** Join with the `members` table to filter based on `join_date`.

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

customer_id	product_name
A	sushi
B	sushi

Solution:

- Filter purchases made before the `join_date` of each customer.
- Use `ROW_NUMBER()` with `ORDER BY` to get the last product purchased before joining.

Key Points:

- **Filter:** Only consider purchases before the membership date.
- **ROW_NUMBER:** Used to rank purchases by date to find the most recent one before joining.
- **Join:** The `sales` table is joined with the `members` table for filtering.

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

customer_id	total_items	total_spent
A	2	25
B	3	40

Solution:

- Aggregate data for purchases made before the customer joined, summing item counts and amounts.
- Filter by `order_date` before the `join_date`.

Key Points:

- **Aggregation:** Use `COUNT()` for item totals and `SUM()` for the amount spent.
- **Filtering:** Ensure purchases are made before `join_date`.
- **Group:** Group by `customer_id` to calculate the totals per member.

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

customer_id	total_points
A	860
B	940
C	360

Solution:

- Calculate points based on the `price` of each item.
- Apply the multiplier for sushi and calculate points accordingly.

Key Points:

- **Points Calculation:** Points are based on the price, with a 2x multiplier for sushi.
- **Multiplier:** Special rule for sushi to award double points.
- **Aggregation:** Sum points for each customer to get total 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?**

customer_id	total_points
A	1270
B	840

Solution:

- Calculate points, applying a 2x multiplier for the first week after joining.
- Apply standard points for other purchases made outside the first week.

Key Points:

- **Points Multiplier:** Apply the 2x multiplier for the first week after the join date.
- **First Week Calculation:** Filter purchases within the first 7 days of membership.
- **Standard Points:** Use regular points calculation for purchases after the first week.

11. **Join the tables above to create some basic datasets that his team can easily examine without resorting to SQL.**

CODE Output, Recreated the table. Link provided above.

Solution:

- Use `JOIN` to combine `sales`, `menu`, and `members` tables into a comprehensive dataset.
- Ensure necessary fields are included (e.g., `customer_id`, `order_date`, `product_name`).

Key Points:

- **JOINs:** Combine the necessary tables based on relevant keys.
- **Comprehensive Data:** Include all relevant data for analysis (order, customer, and product details).
- **Readable Dataset:** Create a simple dataset that can be easily analyzed by the team.

12. Danny needs product rankings for loyalty program members.

CODE Output, Recreated the table. Link provided above.

Solution:

- Rank products based on how often they are purchased by loyalty program members.
- Use `RANK()` or `DENSE_RANK()` to rank products for each customer.

Key Points:

- **Ranking:** Use window functions like `RANK()` to rank products.
- **Loyalty Members:** Only consider purchases from members who are part of the loyalty program.
- **Aggregation:** Group by product to calculate its ranking based on purchase frequency.

Schema	Query

CREATE SCHEMA dannys_diner;
SET search_path = dannys_diner;
 	
CREATE TABLE sales (
  "customer_id" VARCHAR(1),
  "order_date" DATE,
  "product_id" INTEGER
);
 	
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');
	/* --------------------
	  Case Study Questions
	  --------------------*/
	
	-- 1. What is the total amount each customer spent at the restaurant?
	
	-- Tables utilised here: “Menu” and “Sales”
	
	-- SELECT
	-- sales.customer_id,
	-- SUM(menu.price) AS total_spent
	-- FROM sales
	-- JOIN menu
	-- ON sales.product_id = menu.product_id
	-- GROUP BY sales.customer_id
	-- ORDER BY sales.customer_id;
	
	-- 2. How many days has each customer visited the restaurant?
	
	-- Tables utilised here: “Sales”
	
	-- SELECT
	-- customer_id,
	-- COUNT(DISTINCT order_date) AS visit_count
	-- FROM sales

```
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');


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');
```

```
-- GROUP BY customer_id;

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

-- Tables utilised here: “Menu” and “Sales”

-- SELECT DISTINCT ON (sales.customer_id)
--         sales.customer_id,
--     menu.product_name,
--     sales.order_date AS first_purchase_date
-- FROM sales
-- JOIN menu
--   ON sales.product_id = menu.product_id
-- ORDER BY sales.customer_id, sales.order_date, sales.product_id;

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

-- Tables utilised here: “Menu” and “Sales”

-- SELECT
--  menu.product_name,
--  COUNT(*) AS purchase_count
-- FROM sales
-- JOIN menu
--  ON  sales.product_id = menu.product_id
-- GROUP BY menu.product_name
-- ORDER BY purchase_count DESC
-- LIMIT 1;

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

-- Tables utilised here: “Menu” and “Sales”

-- WITH product_counts AS (
--     SELECT
--         customer_id,
--         product_name,
--         COUNT(*) AS purchase_count
--     FROM
--         sales
```

	-- JOIN -- menu ON sales.product_id = menu.product_id -- GROUP BY -- customer_id, product_name --), -- ranked_products AS (-- SELECT -- customer_id, -- product_name, -- purchase_count, -- DENSE_RANK() OVER (PARTITION BY customer_id ORDER BY purchase_count DESC) AS rank -- FROM -- product_counts --) -- SELECT -- customer_id, -- product_name AS most_popular_item, -- purchase_count AS most_popular_count -- FROM -- ranked_products -- WHERE -- rank = 1; -- 6. Which item was purchased first by the customer after they became a member? -- Tables utilised here: “Menu”, "Member" and “Sales” -- SELECT -- m.customer_id, -- mn.product_name -- FROM dannys_diner.members AS m -- JOIN dannys_diner.sales AS s -- ON m.customer_id = s.customer_id -- AND s.order_date > m.join_date -- JOIN dannys_diner.menu AS mn -- ON s.product_id = mn.product_id -- WHERE s.order_date = (-- SELECT MIN(s2.order_date) -- FROM dannys_diner.sales AS s2 -- WHERE s2.customer_id = m.customer_id -- AND s2.order_date > m.join_date

	--) -- ORDER BY m.customer_id; -- 7. Which item was purchased just before the customer became a member? -- Tables utilised here: “Menu”, "Member" and “Sales” -- WITH last_purchase_before_membership 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 -- prior_members.customer_id, -- menu.product_name -- FROM last_purchase_before_membership AS prior_members -- INNER JOIN dannys_diner.menu -- ON prior_members.product_id = menu.product_id -- WHERE rank = 1 -- ORDER BY prior_members.customer_id; -- 8. What are the total items and amount spent for each member before they became a member? -- Tables utilised here: “Menu”, "Member" and “Sales” -- SELECT -- sales.customer_id,

```
--     COUNT(sales.product_id) as
total_items,
--     SUM(menu.price) as total_spent
-- FROM sales
-- JOIN menu
--   ON sales.product_id = menu.product_id
-- JOIN members
--   ON sales.customer_id =
members.customer_id
-- WHERE sales.order_date <
members.join_date
-- GROUP BY sales.customer_id
-- ORDER BY sales.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?

-- Tables utilised here: “Menu” and “Sales”

-- Each $1 spent = 10 points, sushi = 2x
points multiplier

-- SELECT
--         sales.customer_id,
--     SUM(
--         CASE
--           WHEN menu.product_name ='sushi'
THEN menu.price*20
--           ELSE menu.price*10
--         END) AS total_points
--  FROM sales
--  JOIN menu
--    ON sales.product_id = menu.product_id
--  GROUP BY sales.customer_id
--  ORDER BY sales.customer_id;

-- 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?

-- Tables utilised here: “Menu”, "Member"
and “Sales”
```

	-- SELECT -- sales.customer_id, -- SUM(-- CASE -- WHEN sales.order_date BETWEEN members.join_date AND members.join_date + INTERVAL '7 days' THEN menu.price * 20 -- ELSE menu.price*10 -- END) AS total_points -- FROM sales -- JOIN menu -- ON sales.product_id = menu.product_id -- JOIN members -- ON sales.customer_id = members.customer_id -- WHERE sales.order_date <= '2021-01-31' -- GROUP BY sales.customer_id -- ORDER BY sales.customer_id; -- 11. Join the tables above to create some basic datasets that the team can easily examine without resorting to SQL. -- Tables utilised here: “Menu”, "Member" and “Sales” -- SELECT -- sales.customer_id, -- sales.order_date, -- menu.product_name, -- menu.price, -- CASE -- WHEN members.join_date > sales.order_date THEN 'N' -- WHEN members.join_date <= sales.order_date THEN 'Y' -- ELSE 'N' END as member -- FROM sales -- LEFT JOIN menu -- ON sales.product_id = menu.product_id -- LEFT JOIN members -- ON sales.customer_id = members.customer_id -- ORDER BY sales.customer_id, sales.order_date;

	-- 12. Danny needs product rankings for loyalty program members. -- Tables utilised here: “Menu”, "Member" and “Sales” -- WITH customer AS (-- SELECT -- sales.customer_id, -- sales.order_date, -- menu.product_name, -- menu.price, -- CASE -- WHEN members.join_date > sales.order_date THEN 'N' -- WHEN members.join_date <= sales.order_date THEN 'Y' -- ELSE 'N' END AS member -- FROM dannys_diner.sales -- LEFT JOIN dannys_diner.members -- ON sales.customer_id = members.customer_id -- LEFT JOIN dannys_diner.menu -- ON sales.product_id = menu.product_id --) -- SELECT -- *, -- CASE -- WHEN member = 'N' then NULL -- ELSE RANK () OVER (-- PARTITION BY customer_id, member -- ORDER BY order_date --) END AS ranking -- FROM customer;