**Case Study #1 - Danny's Diner**

Danny Ma · May 1, 2021



**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 also 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

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

**Entity Relationship Diagram**

**Example Datasets**

All datasets exist within the dannys\_diner database schema - be sure to include this reference within your SQL scripts as you start exploring the data and answering the case study questions.

**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 |

**Interactive SQL Session**

You can use the embedded DB Fiddle below to easily access these example datasets - this interactive session has everything you need to start solving these questions using SQL.

You can click on the Edit on DB Fiddle link on the top right hand corner of the embedded session below and it will take you to a fully functional SQL editor where you can write your own queries to analyse the data.

You can feel free to choose any SQL dialect you’d like to use, the existing Fiddle is using PostgreSQL 13 as default.

Serious SQL students have access to a dedicated SQL script in the 8 Week SQL Challenge section of the course which they can use to generate relevant temporary tables like we’ve done throughout the entire course!

**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');

**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'),

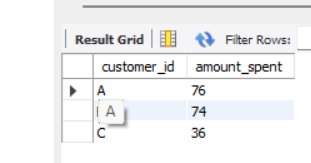
('B', '2021-01-09');

**Case Study Questions**

Each of the following case study questions can be answered using a single SQL statement:

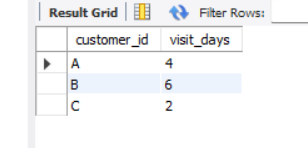
1. What is the total amount each customer spent at the restaurant?

select customer\_id, sum(price) as amount\_spent from sales as s join menu as m on s.product\_id = m.product\_id group by customer\_id;



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

select customer\_id, count(Distinct order\_date) as visit\_days from sales group by customer\_id;



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

select customer\_id, product\_name from

(

select s.customer\_id, s.order\_date, m.product\_name,

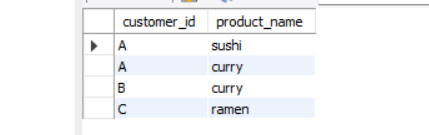
dense\_rank() over (partition by customer\_id order by order\_date) as order\_ranks from

sales as s join menu as m on s.product\_id= m.product\_id

)abc

where order\_ranks = 1

group by customer\_id, product\_name;



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

with most\_popular\_cte as (

SELECT (COUNT(s.product\_id)) AS most\_purchased, product\_name, s.product\_id

FROM sales AS s

JOIN menu AS m

ON s.product\_id = m.product\_id

GROUP BY s.product\_id, product\_name

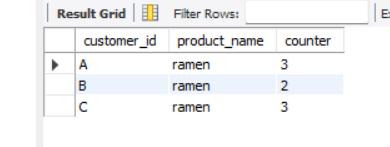
ORDER BY most\_purchased DESC

limit 1)

select customer\_id, product\_name, count(s.product\_id) as counter from sales as s join most\_popular\_cte as m

on s.product\_id = m.product\_id

group by customer\_id;



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

WITH fav\_item\_cte AS

(

SELECT s.customer\_id, m.product\_name,

COUNT(m.product\_id) AS order\_count,

DENSE\_RANK() OVER(PARTITION BY s.customer\_id

ORDER BY COUNT(s.customer\_id) DESC) AS ranker

FROM menu AS m

JOIN sales AS s

ON m.product\_id = s.product\_id

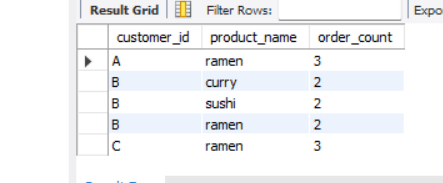
GROUP BY s.customer\_id, m.product\_name

)

SELECT customer\_id, product\_name, order\_count

FROM fav\_item\_cte

WHERE ranker = 1;



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

select s.customer\_id, first\_value(s.product\_id) over (partition by customer\_id)

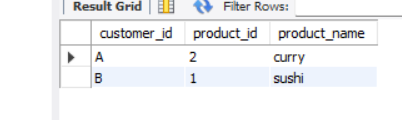
as product\_id, me.product\_name

from sales as s join members as m on s.customer\_id = m.customer\_id

join menu as me on s.product\_id = me.product\_id

where m.join\_date <= s.order\_date

group by customer\_id;



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

WITH prior\_member\_purchased\_cte AS

(

SELECT s.customer\_id, m.join\_date, s.order\_date, s.product\_id,

DENSE\_RANK() OVER(PARTITION BY s.customer\_id

ORDER BY s.order\_date DESC) AS ranker

FROM sales AS s

JOIN members AS m

ON s.customer\_id = m.customer\_id

WHERE s.order\_date < m.join\_date

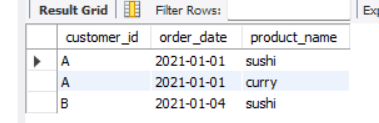
)

select p.customer\_id, p.order\_date, m.product\_name from prior\_member\_purchased\_cte as p

join menu as m on m.product\_id = p.product\_id

where ranker = 1

order by customer\_id;



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

select s.customer\_id, count(m.product\_id) as Total\_items, sum(price) as amount\_spent

from sales as s join menu as m on s.product\_id = m.product\_id

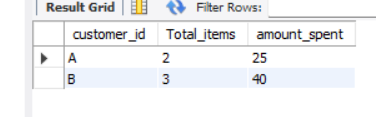
join members as me

on s.customer\_id = me.customer\_id

where s.order\_date < me.join\_date

group by s.customer\_id

order by s.customer\_id;



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

WITH price\_points AS

(

SELECT \*,

CASE

WHEN product\_id = 1 THEN price \* 20

ELSE price \* 10

END AS points

FROM menu

)

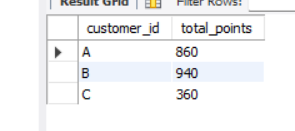
SELECT s.customer\_id, SUM(p.points) AS total\_points

FROM price\_points AS p

JOIN sales AS s

ON p.product\_id = s.product\_id

GROUP BY s.customer\_id;



1. 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 dateadd(day, 6, "2021-01-02");

WITH dates\_cte AS

(

SELECT \*, DATEADD(DAY, 6, join\_date) AS valid\_date,

EOMONTH('2021-01-31') AS last\_date

FROM members AS m

)

SELECT d.customer\_id, s.order\_date, d.join\_date,

d.valid\_date, d.last\_date, m.product\_name, m.price,

SUM(CASE

WHEN m.product\_name = 'sushi' THEN 2 \* 10 \* m.price

WHEN s.order\_date BETWEEN d.join\_date AND d.valid\_date THEN 2 \* 10 \* m.price

ELSE 10 \* m.price

END) AS points

FROM dates\_cte AS d

JOIN sales AS s

ON d.customer\_id = s.customer\_id

JOIN menu AS m

ON s.product\_id = m.product\_id

WHERE s.order\_date < d.last\_date

GROUP BY d.customer\_id, s.order\_date, d.join\_date, d.valid\_date, d.last\_date,

m.product\_name, m.price;

## Bonus Questions

### 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 |

select s.customer\_id, s.order\_date, m.product\_name, m.price,

case when s.order\_date >= me.join\_date then "Y"

else "N"

end as members

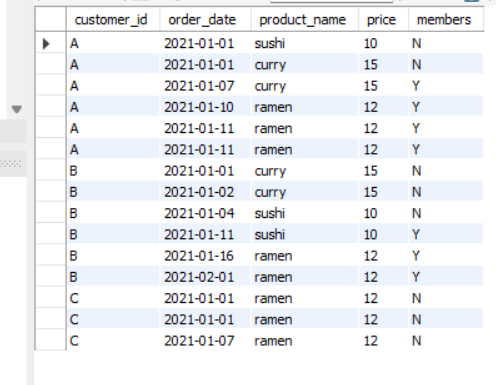
from sales as s left join menu as m

on s.product\_id = m.product\_id

left join members as me

on me.customer\_id = s.customer\_id

order by s.customer\_id;



### 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 |

WITH summary\_cte AS

(

SELECT s.customer\_id, s.order\_date, m.product\_name, m.price,

CASE

WHEN mm.join\_date > s.order\_date THEN 'N'

WHEN mm.join\_date <= s.order\_date THEN 'Y'

ELSE 'N' END AS member

FROM sales AS s

LEFT JOIN menu AS m

ON s.product\_id = m.product\_id

LEFT JOIN members AS mm

ON s.customer\_id = mm.customer\_id

)

SELECT \*, CASE

WHEN member = 'N' then NULL

ELSE

RANK () OVER(PARTITION BY customer\_id, member

ORDER BY order\_date) END AS ranking

FROM summary\_cte;

