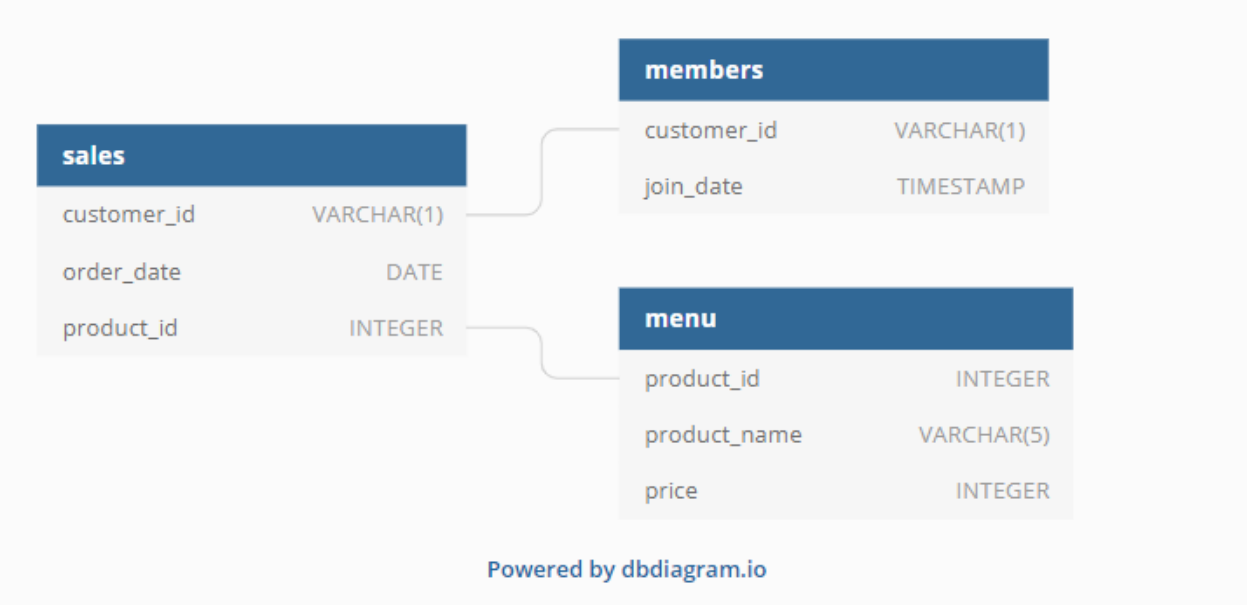
🍜 Case Study : Danny's Diner

Business Task

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.

Entity Relationship Diagram



Question and Solution

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

```
SELECT
 sales.customer_id,
 SUM(menu.price) AS total_sales
FROM dannys_diner.sales
INNER JOIN dannys_diner.menu
```

```
 ON sales.product_id = menu.product_id
GROUP BY sales.customer_id
ORDER BY sales.customer_id ASC;
```

Steps:

- Use **JOIN** to merge dannys_diner.sales and dannys_diner.menu tables as sales.customer_id and menu.price are from both tables.
- Use **SUM** to calculate the total sales contributed by each customer.
- Group the aggregated results by sales.customer_id.

Answer:

customer_id	total_sales
A	76
B	74
C	36

- Customer A spent $76.
- Customer B spent $74.
- Customer C spent $36.

---

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

```
SELECT
 customer_id,
 COUNT(DISTINCT order_date) AS visit_count
FROM dannys_diner.sales
GROUP BY customer_id;
```

Steps:

- To determine the unique number of visits for each customer, utilize **COUNT(DISTINCT order_date)**.
- It's important to apply the **DISTINCT** keyword while calculating the visit count to avoid duplicate counting of days. For instance, if Customer A visited the restaurant twice on '2021–01–07', counting without **DISTINCT** would result in 2 days instead of the accurate count of 1 day.

Answer:

customer_id	visit_count
A	4
B	6
C	2

- Customer A visited 4 times.
- Customer B visited 6 times.
- Customer C visited 2 times.

---

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

```
WITH ordered_sales AS (
 SELECT
  sales.customer_id,
  sales.order_date,
  menu.product_name,
  DENSE_RANK() OVER (
   PARTITION BY sales.customer_id
   ORDER BY sales.order_date) AS rank
 FROM dannys_diner.sales
 INNER JOIN dannys_diner.menu
```

```
  ON sales.product_id = menu.product_id
)

SELECT
 customer_id,
 product_name
FROM ordered_sales
WHERE rank = 1
GROUP BY customer_id, product_name;
```

Steps:

- Create a Common Table Expression (CTE) named ordered_sales_cte. Within the CTE, create a new column rank and calculate the row number using **DENSE_RANK()** window function. The **PARTITION BY** clause divides the data by customer_id, and the **ORDER BY** clause orders the rows within each partition by order_date.
- In the outer query, select the appropriate columns and apply a filter in the **WHERE** clause to retrieve only the rows where the rank column equals 1, which represents the first row within each customer_id partition.
- Use the GROUP BY clause to group the result by customer_id and product_name.

Answer:

customer_id	product_name
A	curry
A	sushi
B	curry
C	ramen

- Customer A placed an order for both curry and sushi simultaneously, making them the first items in the order.
- Customer B's first order is curry.
- Customer C's first order is ramen.

I have received feedback suggesting the use of ROW_NUMBER() instead of DENSE_RANK() for determining the "first order" in this question.

However, since the order_date does not have a timestamp, it is impossible to determine the exact sequence of items ordered by the customer.

Therefore, it would be inaccurate to conclude that curry is the customer's first order purely based on the alphabetical order of the product names. For this reason, I maintain my solution of using DENSE_RANK() and consider both curry and sushi as Customer A's first order.

---

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

```
SELECT
  menu.product_name,
  COUNT(sales.product_id) AS most_purchased_item
FROM dannys_diner.sales
INNER JOIN dannys_diner.menu
  ON sales.product_id = menu.product_id
GROUP BY menu.product_name
ORDER BY most_purchased_item DESC
LIMIT 1;
```

Steps:

- Perform a **COUNT** aggregation on the product_id column and **ORDER BY** the result in descending order using most_purchased field.
- Apply the **LIMIT** 1 clause to filter and retrieve the highest number of purchased items.

Answer:

most_purchased	product_name
8	ramen

- Most purchased item on the menu is ramen which is 8 times. Yummy!

---

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

```
WITH most_popular AS (
 SELECT
  sales.customer_id,
  menu.product_name,
  COUNT(menu.product_id) AS order_count,
  DENSE_RANK() OVER (
   PARTITION BY sales.customer_id
   ORDER BY COUNT(sales.customer_id) DESC) AS rank
 FROM dannys_diner.menu
 INNER JOIN dannys_diner.sales
  ON menu.product_id = sales.product_id
 GROUP BY sales.customer_id, menu.product_name
)

SELECT
 customer_id,
 product_name,
 order_count
FROM most_popular
```

WHERE rank = 1;

Each user may have more than 1 favourite item.

Steps:

- Create a CTE named fav_item_cte and within the CTE, join the menu table and sales table using the product_id column.
- Group results by sales.customer_id and menu.product_name and calculate the count of menu.product_id occurrences for each group.
- Utilize the **DENSE_RANK()** window function to calculate the ranking of each sales.customer_id partition based on the count of orders **COUNT(sales.customer_id)** in descending order.
- In the outer query, select the appropriate columns and apply a filter in the **WHERE** clause to retrieve only the rows where the rank column equals 1, representing the rows with the highest order count for each customer.

Answer:

customer_id	product_name	order_count
A	ramen	3
B	sushi	2
B	curry	2
B	ramen	2
C	ramen	3

- Customer A and C's favourite item is ramen.
- Customer B enjoys all items on the menu. He/she is a true foodie, sounds like me.

---

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

```
WITH joined_as_member AS (
 SELECT
```

```
  members.customer_id,
  sales.product_id,
  ROW_NUMBER() OVER (
    PARTITION BY members.customer_id
    ORDER BY sales.order_date) AS row_num
 FROM dannys_diner.members
 INNER JOIN dannys_diner.sales
  ON members.customer_id = sales.customer_id
  AND sales.order_date > members.join_date
)

SELECT
 customer_id,
 product_name
FROM joined_as_member
INNER JOIN dannys_diner.menu
 ON joined_as_member.product_id = menu.product_id
WHERE row_num = 1
ORDER BY customer_id ASC;
```

Steps:

- Create a CTE named joined_as_member and within the CTE, select the appropriate columns and calculate the row number using the **ROW_NUMBER()** window function. The **PARTITION BY** clause divides the data by members.customer_id and the **ORDER BY** clause orders the rows within each members.customer_id partition by sales.order_date.
- Join tables dannys_diner.members and dannys_diner.sales on customer_id column. Additionally, apply a condition to only include sales that occurred *after* the member's join_date (sales.order_date > members.join_date).

- In the outer query, join the joined_as_member CTE with the dannys_diner.menu on the product_id column.
- In the **WHERE** clause, filter to retrieve only the rows where the row_num column equals 1, representing the first row within each customer_id partition.
- Order result by customer_id in ascending order.

Answer:

customer_id	product_name
A	ramen
B	sushi

- Customer A's first order as a member is ramen.
- Customer B's first order as a member is sushi.

---

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

```
WITH purchased_prior_member 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
  p_member.customer_id,
  menu.product_name
FROM purchased_prior_member AS p_member
INNER JOIN dannys_diner.menu
  ON p_member.product_id = menu.product_id
WHERE rank = 1
ORDER BY p_member.customer_id ASC;
```

Steps:

- Create a CTE called purchased_prior_member.
- In the CTE, select the appropriate columns and calculate the rank using the **ROW_NUMBER()** window function. The rank is determined based on the order dates of the sales in descending order within each customer's group.
- Join dannys_diner.members table with dannys_diner.sales table based on the customer_id column, only including sales that occurred *before* the customer joined as a member (sales.order_date < members.join_date).
- Join purchased_prior_member CTE with dannys_diner.menu table based on product_id column.
- Filter the result set to include only the rows where the rank is 1, representing the earliest purchase made by each customer before they became a member.
- Sort the result by customer_id in ascending order.

Answer:

customer_id	product_name
A	sushi
B	sushi

- Both customers' last order before becoming members are sushi.

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

```
SELECT
  sales.customer_id,
  COUNT(sales.product_id) AS total_items,
  SUM(menu.price) AS total_sales
FROM dannys_diner.sales
INNER JOIN dannys_diner.members
  ON sales.customer_id = members.customer_id
  AND sales.order_date < members.join_date
INNER JOIN dannys_diner.menu
  ON sales.product_id = menu.product_id
GROUP BY sales.customer_id
ORDER BY sales.customer_id;
```

Steps:

- Select the columns sales.customer_id and calculate the count of sales.product_id as total_items for each customer and the sum of menu.price as total_sales.
- From dannys_diner.sales table, join dannys_diner.members table on customer_id column, ensuring that sales.order_date is earlier than members.join_date (sales.order_date < members.join_date).
- Then, join dannys_diner.menu table to dannys_diner.sales table on product_id column.
- Group the results by sales.customer_id.
- Order the result by sales.customer_id in ascending order.

Answer:

customer_id	total_items	total_sales
A	2	25
B	3	40

Before becoming members,

- Customer A spent $25 on 2 items.
- Customer B spent $40 on 3 items.

---

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

```
WITH points_cte AS (
  SELECT
    menu.product_id,
    CASE
      WHEN product_id = 1 THEN price * 20
      ELSE price * 10 END AS points
  FROM dannys_diner.menu
)

SELECT
  sales.customer_id,
  SUM(points_cte.points) AS total_points
FROM dannys_diner.sales
INNER JOIN points_cte
  ON sales.product_id = points_cte.product_id
GROUP BY sales.customer_id
```

```
ORDER BY sales.customer_id;
```

Steps:

Let's break down the question to understand the point calculation for each customer's purchases.

- Each $1 spent = 10 points. However, product_id 1 sushi gets 2x points, so each $1 spent = 20 points.
- Here's how the calculation is performed using a conditional CASE statement:
    - If product_id = 1, multiply every $1 by 20 points.
    - Otherwise, multiply $1 by 10 points.
- Then, calculate the total points for each customer.

Answer:

customer_id	total_points
A	860
B	940
C	360

- Total points for Customer A is $860.
- Total points for Customer B is $940.
- Total points for Customer C is $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?

```
WITH dates_cte AS (
 SELECT
  customer_id,
   join_date,
```

```
    join_date + 6 AS valid_date,
    DATE_TRUNC(
     'month', '2021-01-31'::DATE)
     + interval '1 month'
     - interval '1 day' AS last_date
  FROM dannys_diner.members
)

SELECT
  sales.customer_id,
  SUM(CASE
   WHEN menu.product_name = 'sushi' THEN 2 * 10 * menu.price
   WHEN sales.order_date BETWEEN dates.join_date AND dates.valid_date THEN 2 * 10 *
menu.price
   ELSE 10 * menu.price END) AS points
FROM dannys_diner.sales
INNER JOIN dates_cte AS dates
  ON sales.customer_id = dates.customer_id
  AND dates.join_date <= sales.order_date
  AND sales.order_date <= dates.last_date
INNER JOIN dannys_diner.menu
  ON sales.product_id = menu.product_id
GROUP BY sales.customer_id;
```

Assumptions:

- On Day -X to Day 1 (the day a customer becomes a member), each $1 spent earns 10 points. However, for sushi, each $1 spent earns 20 points.

- From Day 1 to Day 7 (the first week of membership), each $1 spent for any items earns 20 points.
- From Day 8 to the last day of January 2021, each $1 spent earns 10 points. However, sushi continues to earn double the points at 20 points per $1 spent.

Steps:

- Create a CTE called dates_cte.
- In dates_cte, calculate the valid_date by adding 6 days to the join_date and determine the last_date of the month by subtracting 1 day from the last day of January 2021.
- From dannys_diner.sales table, join dates_cte on customer_id column, ensuring that the order_date of the sale is after the join_date (dates.join_date <= sales.order_date) and not later than the last_date (sales.order_date <= dates.last_date).
- Then, join dannys_diner.menu table based on the product_id column.
- In the outer query, calculate the points by using a CASE statement to determine the points based on our assumptions above.
    - If the product_name is 'sushi', multiply the price by 2 and then by 10. For orders placed between join_date and valid_date, also multiply the price by 2 and then by 10.
    - For all other products, multiply the price by 10.
- Calculate the sum of points for each customer.

Answer:

customer_id	total_points
A	1020
B	320

- Total points for Customer A is 1,020.
- Total points for Customer B is 320.

BONUS QUESTIONS

Join All The Things

Recreate the table with: customer_id, order_date, product_name, price, member (Y/N)

```
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_status
FROM dannys_diner.sales
LEFT JOIN dannys_diner.members
  ON sales.customer_id = members.customer_id
INNER JOIN dannys_diner.menu
  ON sales.product_id = menu.product_id
ORDER BY members.customer_id, sales.order_date
```

Answer:

customer_id	order_date	product_name	price	member
A	2021-01-01	sushi	10	N
A	2021-01-01	curry	15	N
A	2021-01-07	curry	15	Y
A	2021-01-10	ramen	12	Y

customer_id	order_date	product_name	price	member
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

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.

```
WITH customers_data 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_status
 FROM dannys_diner.sales
 LEFT JOIN dannys_diner.members
  ON sales.customer_id = members.customer_id
 INNER JOIN dannys_diner.menu
  ON sales.product_id = menu.product_id
)

SELECT
 *,
 CASE
  WHEN member_status = 'N' then NULL
  ELSE RANK () OVER (
   PARTITION BY customer_id, member_status
   ORDER BY order_date
 ) END AS ranking
FROM customers_data;
```

Answer:

customer_id	order_date	product_name	price	member	ranking
A	2021-01-01	sushi	10	N	NULL

customer_id	order_date	product_name	price	member	ranking
A	2021-01-01	curry	15	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