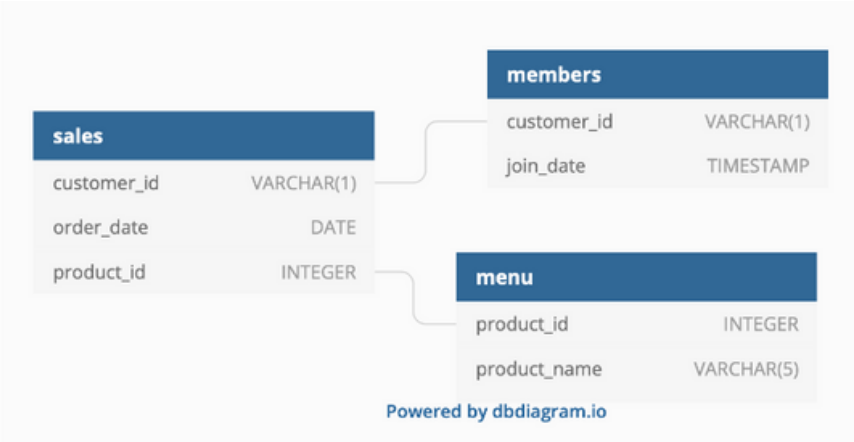
ADVANCED SQL CASE STUDIES

ZEP ANALYTICS

SQL CASE STUDY 1

Consider the following schema for this SQL case study



The sales table captures all customer_id level purchases with and corresponding order_date and product_id information for when and what menu items were ordered.

customer_id	order date	productid
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

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

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

TRY FOLLOWING QUESTIONS

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 is 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 All the Table ?
12. Rank All the Table?

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

```
SELECT s.customer_id, SUM(price) AS total_sales
FROM dbo.sales AS s
JOIN dbo.menu AS m
ON s.product_id = m.product_id
GROUP BY customer_id;
```

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

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

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

```
WITH ordered_sales_cte AS
(
SELECT customer_id, order_date, product_name,
DENSE_RANK() OVER(PARTITION BY s.customer_id
ORDER BY s.order_date) AS rank

FROM dbo.sales AS s
JOIN dbo.menu AS m
ON s.product_id = m.product_id
)
```

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

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

```
SELECT TOP 1 (COUNT(s.product_id)) AS most_purchased,
product_name
FROM dbo.sales AS s
JOIN dbo.menu AS m
ON s.product_id = m.product_id
GROUP BY s.product_id, product_name
ORDER BY most_purchased DESC;SC;
```

5. 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 rank
FROM dbo.menu AS m
JOIN dbo.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 rank = 1;
```

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

```
WITH member_sales_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) AS rank
FROM sales AS s
JOIN members AS m
ON s.customer_id = m.customer_id
WHERE s.order_date &gt;= m.join_date
)
SELECT s.customer_id, s.order_date, m2.product_name
FROM member_sales_cte AS s
JOIN menu AS m2
ON s.product_id = m2.product_id
WHERE rank = 1;
```

7. 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 rank
FROM sales AS s
JOIN members AS m
ON s.customer_id = m.customer_id
WHERE s.order_date &lt; m.join_date
)
SELECT s.customer_id, s.order_date, m2.product_name
FROM prior_member_purchased_cte AS s
JOIN menu AS m2
ON s.product_id = m2.product_id
WHERE rank = 1;
```

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

```
SELECT s.customer_id, COUNT(DISTINCT s.product_id) AS
unique_menu_item, SUM(mm.price) AS total_sales
FROM sales AS s
JOIN members AS m
ON s.customer_id = m.customer_id
JOIN menu AS mm
ON s.product_id = mm.product_id
WHERE s.order_date &amp;lt; m.join_date
GROUP BY s.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?

```
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_cte AS p
JOIN sales AS s
ON p.product_id = s.product_id
GROUP BY s.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?

```
WITH dates_cte AS
(
SELECT *,
DATEADD(DAY, 6, join_date) AS valid_date,
EOMONTH(&#39;2021-01-31&#39;) 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 = &#39;sushi&#39; 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 &amp;lt; 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
```

11)Join All The Things Recreate the table with: customer_id, order_date, product_name, price, member (Y/N)

```
SELECT s.customer_id, s.order_date, m.product_name,
m.price,
CASE
WHEN mm.join_date &amp;gt; s.order_date THEN &amp;#39;N&amp;#39;
WHEN mm.join_date &amp;lt;= s.order_date THEN &amp;#39;Y&amp;#39;
ELSE &amp;#39;N&amp;#39;
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;
```

12) Rank All The Things

```
WITH summary_cte AS
(
SELECT s.customer_id, s.order_date, m.product_name,
m.price,
CASE
WHEN mm.join_date &gt; s.order_date THEN &#39;N&#39;
WHEN mm.join_date &lt;= s.order_date THEN &#39;Y&#39;
ELSE &#39;N&#39; 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 = &#39;N&#39; then NULL
ELSE
RANK () OVER(PARTITION BY customer_id, member
ORDER BY order_date) END AS ranking
FROM summary_cte;
```

SQL CASE STUDY 2

The runners table shows the registration_date for each new runner.

runner_id	registration_date
1	2021-01-01
2	2021-01-03
3	2021-01-08
4	2021-01-15

Customer pizza orders are captured in the customer_orders table with 1 row for each individual pizza that is part of the order. The pizza_id relates to the type of pizza which was ordered whilst the exclusions are the ingredient_id values which should be removed from the pizza and the extras are the ingredient_id values which need to be added to the pizza.

Note that customers can order multiple pizzas in a single order with varying exclusionsand extras values even if the pizza is the same type! The exclusions and extras columns will need to be cleaned up before using them in your queries.

order_id	customer_id	pizza_id	exclusions	extras	order_time
1	101	1			2021-01-01 18:05:02
2	101	1			2021-01-01 19:00:52
3	102	1			2021-01-02 23:51:23
3	102	2		NaN	2021-01-02 23:51:23
4	103	1	4		2021-01-04 13:23:46
4	103	1	4		2021-01-04 13:23:46
4	103	2	4		2021-01-04 13:23:46
5	104	1	null	1	2021-01-08 21:00:29
6	101	2	null	null	2021-01-08 21:03:13
7	105	2	null	1	2021-01-08 21:20:29
8	102	1	null	null	2021-01-09 23:54:33
9	103	1	4	1, 5	2021-01-10 11:22:59
10	104	1	null	null	2021-01-11 18:34:49
10	104	1	2, 6	1, 4	2021-01-11 18:34:49

Table : runner_orders

After each orders are received through the system - they are assigned to a runner - however not all orders are fully completed and can be cancelled by the restaurant or the customer.

The pickup_time is the timestamp at which the runner arrives at the Pizza Runner headquarters to pick up the freshly cooked pizzas. The distance and duration fields are related to how far and long the runner had to travel to deliver the order to the respective customer.

There are some known data issues with this table so be careful when using this in your queries - make sure to check the data types for each column in the schema SQL!

order_id	runner_id	pickup_time	distance	duration	cancellation
1	1	2021-01-01 18:15:34	20km	32 minutes	
2	1	2021-01-01 19:10:54	20km	27 minutes	
3	1	2021-01-03 00:12:37	13.4km	20 mins	NaN
4	2	2021-01-04 13:53:03	23.4	40	NaN
5	3	2021-01-08 21:10:57	10	15	NaN
6	3	null	null	null	Restaurant Cancellation
7	2	2020-01-08 21:30:45	25km	25mins	null
8	2	2020-01-10 00:15:02	23.4 km	15 minute	null
9	2	null	null	null	Customer Cancellation
10	1	2020-01-11 18:50:20	10km	10minutes	null

Table : pizza_names

At the moment - Pizza Runner only has 2 pizzas available the Meat Lovers or Vegetarian!

pizza_id	pizza_name
1	Meat Lovers
2	Vegetarian

Table: pizza_recipes

Each pizza_id has a standard set of toppings which are used as part of the pizza recipe.

pizza_id	toppings
1	1, 2, 3, 4, 5, 6, 8, 10
2	4, 6, 7, 9, 11, 12

Table: pizza_toppings

This table contains all of the topping_name values with their corresponding topping_id value

topping_id	topping_name
1	Bacon
2	BBQ Sauce
3	Beef
4	Cheese
5	Chicken
6	Mushrooms
7	Onions
8	Pepperoni
9	Peppers
10	Salami
11	Tomatoes
12	Tomato Sauce

TRY FOLLOWING QUESTIONS

1. Perform Data Cleaning and Transformation ?
2. How many pizzas were ordered?
3. How many unique customer orders were made?
4. How many successful orders were delivered by each runner?
5. How many of each type of pizza was delivered?
6. How many Vegetarian and Meatlovers were ordered by each customer?
7. What was the maximum number of pizzas delivered in a single order?
8. For each customer, how many delivered pizzas had at least 1 change and how many had no changes?
9. How many pizzas were delivered that had both exclusions and extras?
10. What was the total volume of pizzas ordered for each hour of the day?
11. What was the volume of orders for each day of the week?
12. How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)
13. What was the average time in minutes it took for each runner to arrive at the Pizza Runner HQ to pickup the order?

TRY FOLLOWING QUESTIONS

14. Is there any relationship between the number of pizzas and how long the order takes to prepare?
15. What was the average distance travelled for each customer?
16. What was the difference between the longest and shortest delivery times for all orders?
17. What was the average speed for each runner for each delivery and do you notice any trend for these values?
18. What is the successful delivery percentage for each runner?

1)Data Cleaning and Transformation

Firstly, to clean up exclusions and extras in the customer_orders — we create TEMP TABLE #customer_orders and use CASE WHEN.

```
SELECT order_id, customer_id, pizza_id,
CASE
WHEN exclusions IS null OR exclusions LIKE 'null' THEN ' '
ELSE exclusions
END AS exclusions,
CASE
WHEN extras IS NULL or extras LIKE 'null' THEN ' '
ELSE extras
END AS extras,
order_time
INTO #customer_orders -- create TEMP TABLE
FROM customer_orders;
```

Then, we clean the runner_orders table with CASE WHEN and TRIM and create TEMP TABLE #runner_orders.

In summary,
·pickup_time — Remove nulls and replace with ' '
·distance — Remove 'km' and nulls
·duration — Remove 'minutes' and nulls
·cancellation — Remove NULL and null and replace with ' '

·SELECT order_id, runner_id,
CASE
WHEN pickup_time LIKE 'null' THEN ' '
ELSE pickup_time
END AS pickup_time,
CASE
WHEN distance LIKE 'null' THEN ' '
WHEN distance LIKE '%km' THEN TRIM('km' from distance)
ELSE distance END AS distance,
CASE
WHEN duration LIKE 'null' THEN ' '
WHEN duration LIKE '%mins' THEN TRIM('mins' from duration)
WHEN duration LIKE '%minute' THEN TRIM('minute' from duration)
WHEN duration LIKE '%minutes' THEN TRIM('minutes' from duration)
ELSE duration END AS duration,
CASE
WHEN cancellation IS NULL or cancellation LIKE 'null' THEN ''
ELSE cancellation END AS cancellation
INTO #runner_orders
FROM runner_orders;

Then, we alter the date according to its correct data type.
·pickup_time to DATETIME type
·distance to FLOAT type
·duration to INT type

```
ALTER TABLE #runner_orders
ALTER COLUMN pickup_time DATETIME,
ALTER COLUMN distance FLOAT,
ALTER COLUMN duration INT;
```

2)How many pizzas were ordered?

```
SELECT COUNT(*) AS pizza_order_count
 FROM #customer_orders;
```

3) How many unique customer orders were made?

```
SELECT COUNT(DISTINCT order_id) AS unique_order_count
 FROM #customer_orders;
```

4) How many successful orders were delivered by each runner?

```
SELECT runner_id, COUNT(order_id) AS successful_orders
 FROM #runner_orders
 WHERE distance != 0
 GROUP BY runner_id;
```

5) How many of each type of pizza was delivered?

```
SELECT p.pizza_name, COUNT(c.pizza_id) AS
delivered_pizza_count
 FROM #customer_orders AS c
 JOIN #runner_orders AS r
 ON c.order_id = r.order_id
 JOIN pizza_names AS p
```

```
ON c.pizza_id = p.pizza_id
 WHERE r.distance != 0
 GROUP BY p.pizza_name;
```

6). How many Vegetarian and Meatlovers were ordered by each customer?

```
SELECT c.customer_id, p.pizza_name, COUNT(p.pizza_name) AS order_count
 FROM #customer_orders AS c
 JOIN pizza_names AS p
 ON c.pizza_id= p.pizza_id
 GROUP BY c.customer_id, p.pizza_name
 ORDER BY c.customer_id;
```

7)What was the maximum number of pizzas delivered in a single order?

```
WITH pizza_count_cte AS
 (
 SELECT c.order_id, COUNT(c.pizza_id) AS pizza_per_order
 FROM #customer_orders AS c
 JOIN #runner_orders AS r
 ON c.order_id = r.order_id
 WHERE r.distance != 0
 GROUP BY c.order_id
 )
SELECT MAX(pizza_per_order) AS pizza_count
 FROM pizza_count_cte;
```

8) For each customer, how many delivered pizzas had at least 1 change and how many had no changes?

```
SELECT c.customer_id,
 SUM(CASE
 WHEN c.exclusions <> ' ' OR c.extras <> ' ' THEN 1
 ELSE 0
 END) AS at_least_1_change,
 SUM(CASE
 WHEN c.exclusions = ' ' AND c.extras = ' ' THEN 1
 ELSE 0
 END) AS no_change
 FROM #customer_orders AS c
 JOIN #runner_orders AS r
 ON c.order_id = r.order_id
 WHERE r.distance != 0
 GROUP BY c.customer_id
 ORDER BY c.customer_id;
```

9) How many pizzas were delivered that had both exclusions and extras?

```
SELECT
 SUM(CASE
 WHEN exclusions IS NOT NULL AND extras IS NOT NULL THEN 1
 ELSE 0
 END) AS pizza_count_w_exclusions_extras
 FROM #customer_orders AS c
```

```
JOIN #runner_orders AS r
 ON c.order_id = r.order_id
 WHERE r.distance >= 1
 AND exclusions <> ' '
 AND extras <> ' ';
```

10) What was the total volume of pizzas ordered for each hour of the day?

```
SELECT DATEPART(HOUR, [order_time]) AS hour_of_day,
 COUNT(order_id) AS pizza_count
 FROM #customer_orders
 GROUP BY DATEPART(HOUR, [order_time]);
```

11) What was the volume of orders for each day of the week?

```
SELECT FORMAT(DATEADD(DAY, 2, order_time),'dddd') AS day_of_week,
 -- add 2 to adjust 1st day of the week as Monday
 COUNT(order_id) AS total_pizzas_ordered
 FROM #customer_orders
 GROUP BY FORMAT(DATEADD(DAY, 2, order_time),'dddd');
```

12)How many runners signed up for each 1 week period? (i.e. week starts 2021-01-01)

```
SELECT DATEPART(WEEK, registration_date) AS registration_week,
 COUNT(runner_id) AS runner_signup
FROM runners
GROUP BY DATEPART(WEEK, registration_date);
```

13) What was the average time in minutes it took for each runner to arrive at the Pizza Runner HQ to pickup the order?

```
WITH time_taken_cte AS
 (
 SELECT c.order_id, c.order_time, r.pickup_time,
      DATEDIFF(MINUTE, c.order_time, r.pickup_time) AS pickup_minutes
 FROM #customer_orders AS c
 JOIN #runner_orders AS r
 ON c.order_id = r.order_id
 WHERE r.distance != 0
 GROUP BY c.order_id, c.order_time, r.pickup_time
 )
SELECT AVG(pickup_minutes) AS avg_pickup_minutes
FROM time_taken_cte
WHERE pickup_minutes > 1;
```

14) Is there any relationship between the number of pizzas and how long the order takes to prepare?

```
WITH prep_time_cte AS
 (
 SELECT c.order_id, COUNT(c.order_id) AS pizza_order,
 c.order_time, r.pickup_time,
      DATEDIFF(MINUTE, c.order_time, r.pickup_time) AS prep_time_minutes
 FROM #customer_orders AS c
```

```
JOIN #runner_orders AS r
ON c.order_id = r.order_id
WHERE r.distance != 0
GROUP BY c.order_id, c.order_time, r.pickup_time
)

SELECT pizza_order, AVG(prep_time_minutes) AS avg_prep_time_minutes
FROM prep_time_cte
WHERE prep_time_minutes > 1
GROUP BY pizza_order;
```

15) What was the average distance travelled for each customer?

```
SELECT c.customer_id, AVG(r.distance) AS avg_distance
 FROM #customer_orders AS c
 JOIN #runner_orders AS r
 ON c.order_id = r.order_id
 WHERE r.duration != 0
 GROUP BY c.customer_id;
```

16) What was the difference between the longest and shortest delivery times for all orders?

Firstly, let's see all the durations for the orders.

```
SELECT order_id, duration
 FROM #runner_orders
 WHERE duration not like ' ';
```

Then, we find the difference by deducting the shortest (MIN) from the longest (MAX) delivery times.

```
SELECT MAX(duration::NUMERIC) - MIN(duration::NUMERIC) AS delivery_time_difference
 FROM #runner_orders
 WHERE duration not like '% %'
```

17) What was the average speed for each runner for each delivery and do you notice any trend for these values?

```
SELECT r.runner_id, c.customer_id, c.order_id,
 COUNT(c.order_id) AS pizza_count,
 r.distance, (r.duration / 60) AS duration_hr ,
 ROUND((r.distance/r.duration * 60), 2) AS avg_speed
 FROM #runner_orders AS r
 JOIN #customer_orders AS c
 ON r.order_id = c.order_id
 WHERE distance != 0
  GROUP BY r.runner_id, c.customer_id, c.order_id, r.distance, r.duration
 ORDER BY c.order_id;
```

18) What is the successful delivery percentage for each runner?

```
SELECT runner_id,
 ROUND(100 * SUM
 (CASE WHEN distance = 0 THEN 0
 ELSE 1
 END) / COUNT(*), 0) AS success_perc
 FROM #runner_orders
 GROUP BY runner_id;
```

SQL CASE STUDY 3

Users

Customers who visit the Clique Bait website are tagged via their cookie_id.

user_id	cookie_id	start_date
397	3759ff	2020-03-30 00:00:00
215	863329	2020-01-26 00:00:00
191	eefca9	2020-03-15 00:00:00
89	764796	2020-01-07 00:00:00
127	17ccc5	2020-01-22 00:00:00
81	b0b666	2020-03-01 00:00:00
260	a4f236	2020-01-08 00:00:00
203	d1182f	2020-04-18 00:00:00
23	12dbc8	2020-01-18 00:00:00
375	f61d69	2020-01-03 00:00:00

.

Events

Customer visits are logged in this events table at a cookie_id level and the event_typeand page_id values can be used to join onto relevant satellite tables to obtain further information about each event. The sequence_number is used to order the events within each visit.

visit_id	cookie_id	page_id	event_type	sequence_number	event_time
719fd3	3d83d3	5	1	4	2020-03-02 00:29:09.975502
fb1eb1	c5ff25	5	2	8	2020-01-22 07:59:16.761931
23fe81	1e8c2d	10	1	9	2020-03-21 13:14:11.745667
ad91aa	648115	6	1	3	2020-04-27 16:28:09.824606
5576d7	ac418c	6	1	4	2020-01-18 04:55:10.149236
48308b	c686c1	8	1	5	2020-01-29 06:10:38.702163
46b17d	78f9b3	7	1	12	2020-02-16 09:45:31.926407
9fd196	ccf057	4	1	5	2020-02-14 08:29:12.922164
edf853	f85454	1	1	1	2020-02-22 12:59:07.652207
3c6716	02e74f	3	2	5	2020-01-31 17:56:20.777383

Event Identifier

The event_identifier table shows the types of events which are captured by Clique Bait's digital data systems.

event_type	event_name
1	Page View
2	Add to Cart
3	Purchase
4	Ad Impression
5	Ad Click

Campaign Identifier

This table shows information for the 3 campaigns that Clique Bait has ran on their website so far in 2020.

campaign_id	products	campaign_name	start_date	end_date
1	1-3	BOGOF - Fishing For Compliments	2020-01-01 00:00:00	2020-01-14 00:00:00
2	4-5	25% Off - Living The Lux Life	2020-01-15 00:00:00	2020-01-28 00:00:00
3	6-8	Half Off - Treat Your Shellf(ish)	2020-02-01 00:00:00	2020-03-31 00:00:00

Page Hierarchy

This table lists all of the pages on the Clique Bait website which are tagged and have data passing through from user interaction events.

page_id	page_name	product_category	product_id
1	Home Page	null	null
2	All Products	null	null
3	Salmon	Fish	1
4	Kingfish	Fish	2
5	Tuna	Fish	3
6	Russian Caviar	Luxury	4
7	Black Truffle	Luxury	5
8	Abalone	Shellfish	6
9	Lobster	Shellfish	7
10	Crab	Shellfish	8
11	Oyster	Shellfish	9
12	Checkout	null	null
13	Confirmation	null	null

TRY FOLLOWING QUESTIONS

1. How many users are there?
2. How many cookies does each user have on average?
3. What is the unique number of visits by all users per month?
4. What is the number of events for each event type?
5. What is the percentage of visits which have a purchase event?
6. What is the percentage of visits which view the checkout page but do not have a purchase event?
7. What are the top 3 pages by number of views?
8. What is the number of views and cart adds for each product category?
9. Using a single SQL query - create a new output table which has the following details:

- How many times was each product viewed?
- How many times was each product added to cart?
- How many times was each product added to a cart but not purchased (abandoned)?
- How many times was each product purchased?

Additionally, create another table which further aggregates the data for the above points but this time for each product category instead of individual products. Use your 2 new output tables - answer the following questions:

10. Which product had the highest view to purchase percentage?
11. What is the average conversion rate from cart add to purchase?

1. How many users are there?

```
SELECT COUNT(DISTINCT user_id) AS user_count
FROM clique_bait.users;
```

2. How many cookies does each user have on average?

This was one of those tricky questions that seems easy, but the solution is not as clear as it seems.

- Question is asking the number of cookies each user have on average. That's calling us to either use a DISTINCT or GROUP BY in order to ensure the count of cookies belonging to each user is unique.
- Next, round up the average cookie count with 0 decimal point as it will not make sense for the cookie to be in fractional form.

```
WITH cookie AS (
 SELECT
   user_id,
COUNT(cookie_id) AS cookie_id_count
 FROM clique_bait.users
 GROUP BY user_id)

SELECT
ROUND(AVG(cookie_id_count),0) AS avg_cookie_id
FROM cookie;
```

3. What is the unique number of visits by all users per month?

- First, extract numerical month from event_time so that we can group the data by month.
- Unique is a keyword to use DISTINCT.

```
SELECT
EXTRACT(MONTH FROM event_time) as month,
COUNT(DISTINCT visit_id) AS unique_visit_count
FROM clique_bait.events
GROUP BY EXTRACT(MONTH FROM event_time);
```

4. What is the number of events for each event type?

```
SELECT
 event_type,
 COUNT(*) AS event_count
FROM clique_bait.events
GROUP BY event_type
ORDER BY event_type;
```

5. What is the percentage of visits which have a purchase event?

- Join the events and events_identifier table and filter by Purchase event only.
- As the data is now filtered to having Purchase events only, counting the distinct visit IDs would give you the number of purchase events.
- Then, divide the number of purchase events with a subquery of total number of distinct visits from the events table.

```
SELECT
 100 * COUNT(DISTINCT e.visit_id)/
  (SELECT COUNT(DISTINCT visit_id) FROM clique_bait.events) AS
percentage_purchase
FROM clique_bait.events AS e
JOIN clique_bait.event_identifier AS ei
 ON e.event_type = ei.event_type
WHERE ei.event_name = 'Purchase';
```

6. What is the percentage of visits which view the checkout page but do not have a purchase event? The strategy to answer this question is to breakdown the question into 2 parts.

Part 1: Create a CTE and using CASE statements, find the MAX() of:

- event_type = 1 (Page View) and page_id = 12 (Checkout), and assign "1" to these events. These events are when user viewed the checkout page.
- event_type = 3 (Purchase) and assign "1" to these events. These events signifies users who made a purchase.

We're using MAX() because we do not want to group the results by event_type and page_id. Since the max score is "1", it would mean "Give me the max score for each event".
Part 2: Using the table we have created, find the percentage of visits which view checkout page.

```
WITH checkout_purchase AS (
SELECT
 visit_id,
 MAX(CASE WHEN event_type = 1 AND page_id = 12 THEN 1 ELSE 0 END) AS checkout,
 MAX(CASE WHEN event_type = 3 THEN 1 ELSE 0 END) AS purchase
FROM clique_bait.events
GROUP BY visit_id)
SELECT
 ROUND(100 * (1-(SUM(purchase)::numeric/SUM(checkout))),2) AS percentage_checkout_view_with_no_purchase
FROM checkout_purchase
```

7. What are the top 3 pages by number of views?

```
SELECT
ph.page_name,
 COUNT(*) AS page_views
FROM clique_bait.events AS e
JOIN clique_bait.page_hierarchy AS ph
 ON e.page_id = ph.page_id
WHERE e.event_type = 1 -- "Page View"
```

```
GROUP BY ph.page_name
ORDER BY page_views DESC -- Order by descending to retrieve highest to lowest number of views
LIMIT 3; -- Limit results to 3 to find the top 3
```

8. What is the number of views and cart adds for each product category?

```
SELECT
ph.product_category,
 SUM(CASE WHEN e.event_type = 1 THEN 1 ELSE 0 END) AS page_views,
 SUM(CASE WHEN e.event_type = 2 THEN 1 ELSE 0 END) AS cart_adds
FROM clique_bait.events AS e
JOIN clique_bait.page_hierarchy AS ph
 ON e.page_id = ph.page_id
WHERE ph.product_category IS NOT NULL
GROUP BY ph.product_category
ORDER BY page_views DESC;
```

9)Using a single SQL query - create a new output table which has the following details:

1. How many times was each product viewed?
2. How many times was each product added to cart?
3. How many times was each product added to a cart but not purchased (abandoned)?
4. How many times was each product purchased?

Planning Our Strategy

Let us visualize the output table.

Column	Description
product	Name of the product
views	Number of views for each product
cart_adds	Number of cart adds for each product
abandoned	Number of times product was added to a cart, but not purchased
purchased	Number of times product was purchased

These information would come from these 2 tables.

- events table - visit_id, page_id, event_type
- page_hierarchy table - page_id, product_category

Solution

- Note 1 - In product_page_events CTE, find page views and cart adds for individual visit ids by wrapping SUM around CASE statements so that we do not have to group the results by event_type as well.
- Note 2 - In purchase_events CTE, get only visit ids that have made purchases.

- Note 3 - In combined_table CTE, merge product_page_events and purchase_events using LEFT JOIN. Take note of the table sequence. In order to filter for visit ids with purchases, we use a CASE statement and where visit id is not null, it means the visit id is a purchase.

```
WITH product_page_events AS ( -- Note 1
 SELECT
e.visit_id,
ph.product_id,
ph.page_name AS product_name,
ph.product_category,
   SUM(CASE WHEN e.event_type = 1 THEN 1 ELSE 0 END) AS
page_view, -- 1 for Page View
   SUM(CASE WHEN e.event_type = 2 THEN 1 ELSE 0 END) AS
cart_add -- 2 for Add Cart
 FROM clique_bait.events AS e
 JOIN clique_bait.page_hierarchy AS ph
   ON e.page_id = ph.page_id
 WHERE product_id IS NOT NULL
 GROUP BY e.visit_id, ph.product_id, ph.page_name,
ph.product_category
),
purchase_events AS ( -- Note 2
 SELECT
   DISTINCT visit_id
 FROM clique_bait.events
 WHERE event_type = 3 -- 3 for Purchase
```

```
),
combined_table AS ( -- Note 3
 SELECT
ppe.visit_id,
ppe.product_id,
ppe.product_name,
ppe.product_category,
ppe.page_view,
ppe.cart_add,
   CASE WHEN pe.visit_id IS NOT NULL THEN 1 ELSE 0 END AS
purchase
 FROM product_page_events AS ppe
 LEFT JOIN purchase_events AS pe
   ON ppe.visit_id = pe.visit_id
),
product_info AS (
 SELECT
product_name,
product_category,
SUM(page_view) AS views,
   SUM(cart_add) AS cart_adds,
   SUM(CASE WHEN cart_add = 1 AND purchase = 0 THEN 1 ELSE
0 END) AS abandoned,
```

```
   SUM(CASE WHEN cart_add = 1 AND purchase = 1 THEN 1 ELSE
0 END) AS purchases
 FROM combined_table
 GROUP BY product_id, product_name, product_category)
SELECT *
FROM product_info
ORDER BY product_id;
```

10. Which product had the highest view to purchase percentage?

```
SELECT
product_name,
product_category,
 ROUND(100 * purchases/views,2) AS
purchase_per_view_percentage
FROM product_info
ORDER BY purchase_per_view_percentage DESC
```

11. What is the average conversion rate from cart add to purchase?

```
SELECT  ROUND(100*AVG(cart_adds/views),2) AS
avg_view_to_cart_add_conversion,
ROUND(100*AVG(purchases/cart_adds),2) AS
avg_cart_add_to_purchases_conversion_rate
FROM product_info
```

Ready to take the next steps?

So these were some of the important DAX functions which you should remember.

Remember DAX is very vast and there are many more but these are the basic and the most used ones.