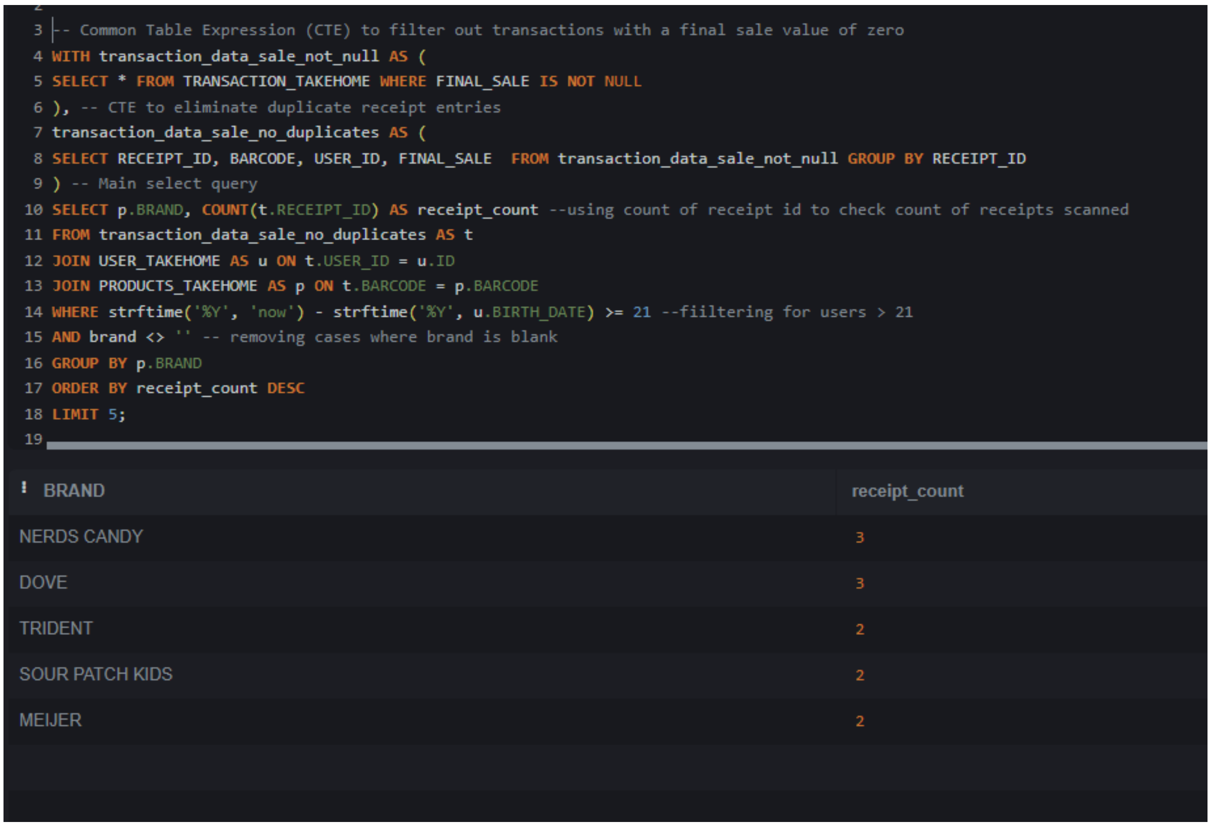
Q1 What are the top 5 brands by receipts scanned among users 21 and over?



```
-- Common Table Expression (CTE) to filter out transactions with a final sale value of zero
WITH transaction_data_sale_not_null AS (
SELECT * FROM TRANSACTION_TAKEHOME WHERE FINAL_SALE IS NOT NULL
), -- CTE to eliminate duplicate receipt entries
transaction_data_sale_no_duplicates AS (
SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE  FROM transaction_data_sale_not_null GROUP BY RECEIPT_ID
) -- Main select query
SELECT p.BRAND, COUNT(t.RECEIPT_ID) AS receipt_count --using count of receipt id to check count of receipts scanned
FROM transaction_data_sale_no_duplicates AS t
JOIN USER_TAKEHOME AS u ON t.USER_ID = u.ID
JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE
WHERE strftime('%Y', 'now') - strftime('%Y', u.BIRTH_DATE) >= 21 --fiiltering for users > 21
AND brand <> '' -- removing cases where brand is blank
GROUP BY p.BRAND
ORDER BY receipt_count DESC
LIMIT 5;
```

BRAND	receipt_count
NERDS CANDY	3
DOVE	3
TRIDENT	2
SOUR PATCH KIDS	2
MEIJER	2

WITH transaction_data_sale_not_null AS (

SELECT * FROM TRANSACTION_TAKEHOME WHERE FINAL_SALE IS NOT NULL

), -- CTE to eliminate duplicate receipt entries

transaction_data_sale_no_duplicates AS (

SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE FROM transaction_data_sale_not_null GROUP BY RECEIPT_ID

) -- Main select query

SELECT p.BRAND, COUNT(t.RECEIPT_ID) AS receipt_count --using count of receipt id to check count of receipts scanned

FROM transaction_data_sale_no_duplicates as t

JOIN USER_TAKEHOME AS u ON t.USER_ID = u.ID

JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE

WHERE strftime('%Y', 'now') - strftime('%Y', u.BIRTH_DATE) >= 21 --fiiltering for users > 21

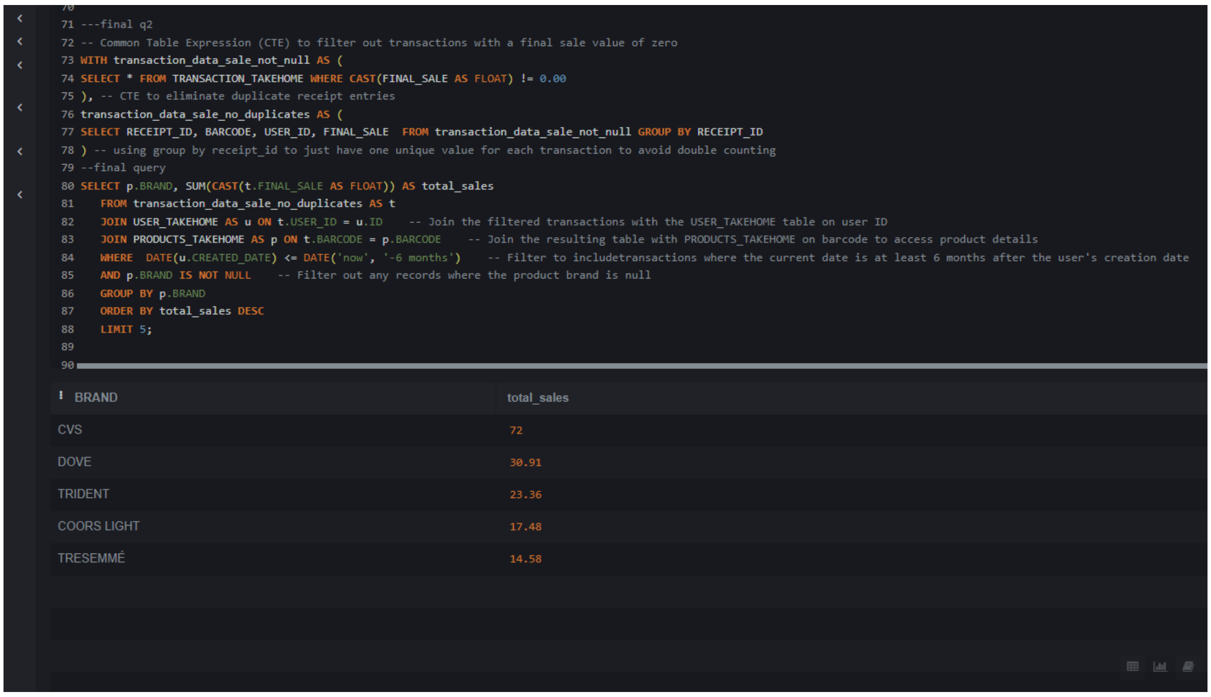
and brand <> '' -- removing cases where brand is blank

GROUP BY p.BRAND

ORDER BY receipt_count DESC

LIMIT 5;

Q2 What are the top 5 brands by sales among users that have had their account for at least six months?



```sql
---final q2
-- Common Table Expression (CTE) to filter out transactions with a final sale value of zero
WITH transaction_data_sale_not_null AS (
SELECT * FROM TRANSACTION_TAKEHOME WHERE CAST(FINAL_SALE AS FLOAT) != 0.00
), -- CTE to eliminate duplicate receipt entries
transaction_data_sale_no_duplicates AS (
SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE  FROM transaction_data_sale_not_null GROUP BY RECEIPT_ID
) -- using group by receipt_id to just have one unique value for each transaction to avoid double counting
--final query
SELECT p.BRAND, SUM(CAST(t.FINAL_SALE AS FLOAT)) AS total_sales
   FROM transaction_data_sale_no_duplicates AS t
   JOIN USER_TAKEHOME AS u ON t.USER_ID = u.ID    -- Join the filtered transactions with the USER_TAKEHOME table on user ID
   JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE    -- Join the resulting table with PRODUCTS_TAKEHOME on barcode to access product details
   WHERE  DATE(u.CREATED_DATE) <= DATE('now', '-6 months')    -- Filter to includetransactions where the current date is at least 6 months after the user's creation date
   AND p.BRAND IS NOT NULL    -- Filter out any records where the product brand is null
   GROUP BY p.BRAND
   ORDER BY total_sales DESC
   LIMIT 5;
```

BRAND	total_sales
CVS	72
DOVE	30.91
TRIDENT	23.36
COORS LIGHT	17.48
TRESEMMÉ	14.58

-- Common Table Expression (CTE) to filter out transactions with a final sale value of zero

WITH transaction_data_sale_not_null AS (

SELECT * FROM TRANSACTION_TAKEHOME WHERE CAST(FINAL_SALE AS FLOAT) != 0.00

), -- CTE to eliminate duplicate receipt entries

transaction_data_sale_no_duplicates AS (

SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE FROM transaction_data_sale_not_null GROUP BY RECEIPT_ID

) -- using group by receipt_id to just have one unique value for each transaction to avoid double counting

--final query

SELECT p.BRAND, SUM(CAST(t.FINAL_SALE AS FLOAT)) AS total_sales

  FROM transaction_data_sale_no_duplicates AS t

  JOIN USER_TAKEHOME AS u ON t.USER_ID = u.ID   -- Join the filtered transactions with the USER_TAKEHOME table on user ID

  JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE   -- Join the resulting table with PRODUCTS_TAKEHOME on barcode to access product details

  WHERE DATE(u.CREATED_DATE) <= DATE('now', '-6 months')   -- Filter to includetransactions where the current date is at least 6 months after the user's creation date

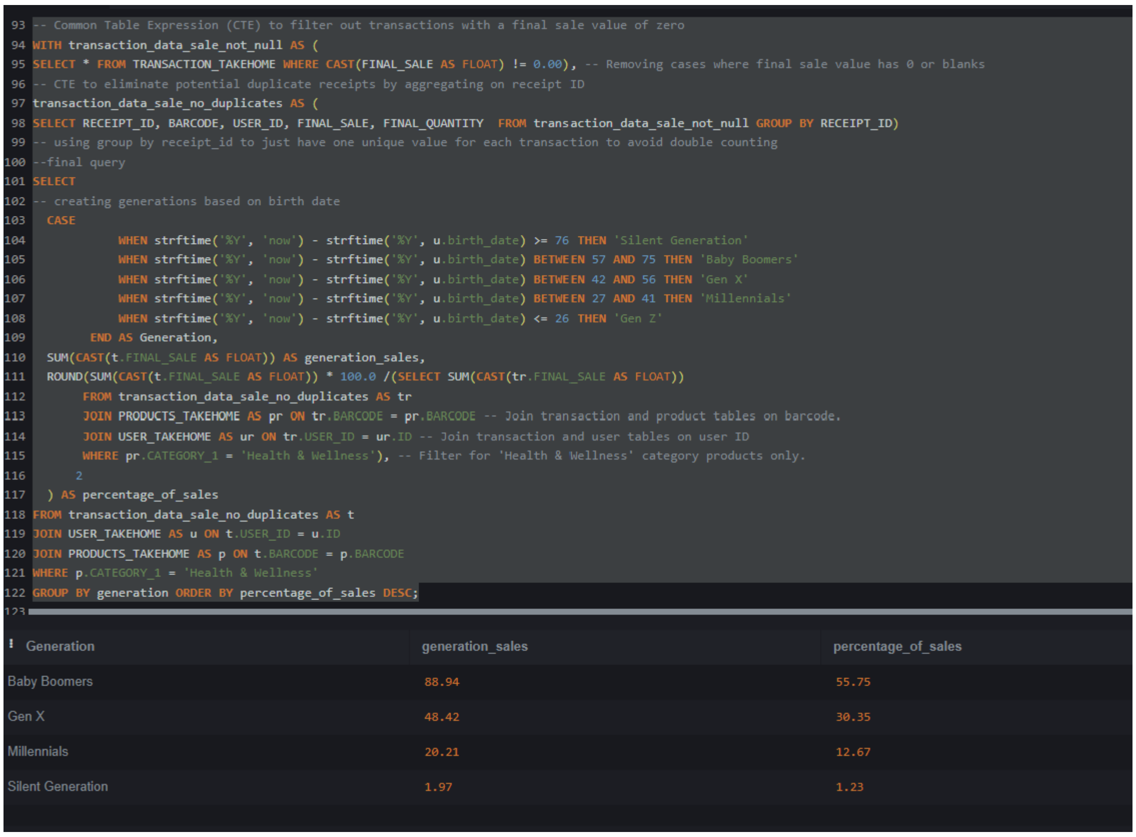
  AND p.BRAND IS NOT NULL   -- Filter out any records where the product brand is null

  GROUP BY p.BRAND

  ORDER BY total_sales DESC

  LIMIT 5;

Q3 What is the percentage of sales in the Health & Wellness category by generation?



```
-- Common Table Expression (CTE) to filter out transactions with a final sale value of zero
WITH transaction_data_sale_not_null AS (
SELECT * FROM TRANSACTION_TAKEHOME WHERE CAST(FINAL_SALE AS FLOAT) != 0.00), -- Removing cases where final sale value has 0 or blanks
-- CTE to eliminate potential duplicate receipts by aggregating on receipt ID
transaction_data_sale_no_duplicates AS (
SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE, FINAL_QUANTITY  FROM transaction_data_sale_not_null GROUP BY RECEIPT_ID)
-- using group by receipt_id to just have one unique value for each transaction to avoid double counting
--final query
SELECT
-- creating generations based on birth date
  CASE
            WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) >= 76 THEN 'Silent Generation'
            WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) BETWEEN 57 AND 75 THEN 'Baby Boomers'
            WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) BETWEEN 42 AND 56 THEN 'Gen X'
            WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) BETWEEN 27 AND 41 THEN 'Millennials'
            WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) <= 26 THEN 'Gen Z'
        END AS Generation,
  SUM(CAST(t.FINAL_SALE AS FLOAT)) AS generation_sales,
  ROUND(SUM(CAST(t.FINAL_SALE AS FLOAT)) * 100.0 /(SELECT SUM(CAST(tr.FINAL_SALE AS FLOAT))
      FROM transaction_data_sale_no_duplicates AS tr
      JOIN PRODUCTS_TAKEHOME AS pr ON tr.BARCODE = pr.BARCODE -- Join transaction and product tables on barcode.
      JOIN USER_TAKEHOME AS ur ON tr.USER_ID = ur.ID -- Join transaction and user tables on user ID
      WHERE pr.CATEGORY_1 = 'Health & Wellness'), -- Filter for 'Health & Wellness' category products only.
    2
  ) AS percentage_of_sales
FROM transaction_data_sale_no_duplicates AS t
JOIN USER_TAKEHOME AS u ON t.USER_ID = u.ID
JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE
WHERE p.CATEGORY_1 = 'Health & Wellness'
GROUP BY generation ORDER BY percentage_of_sales DESC;
```

Generation	generation_sales	percentage_of_sales
Baby Boomers	88.94	55.75
Gen X	48.42	30.35
Millennials	20.21	12.67
Silent Generation	1.97	1.23

-- Common Table Expression (CTE) to filter out transactions with a final sale value of zero

WITH transaction_data_sale_not_null AS (

SELECT * FROM TRANSACTION_TAKEHOME WHERE CAST(FINAL_SALE AS FLOAT) != 0.00), -- Removing cases where final sale value has 0 or blanks

-- CTE to eliminate potential duplicate receipts by aggregating on receipt ID

transaction_data_sale_no_duplicates AS (

SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE, FINAL_QUANTITY FROM transaction_data_sale_not_null GROUP BY RECEIPT_ID)

-- using group by receipt_id to just have one unique value for each transaction to avoid double counting

--final query

SELECT

-- creating generations based on birth date

 CASE

      WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) >= 76 THEN 'Silent Generation'

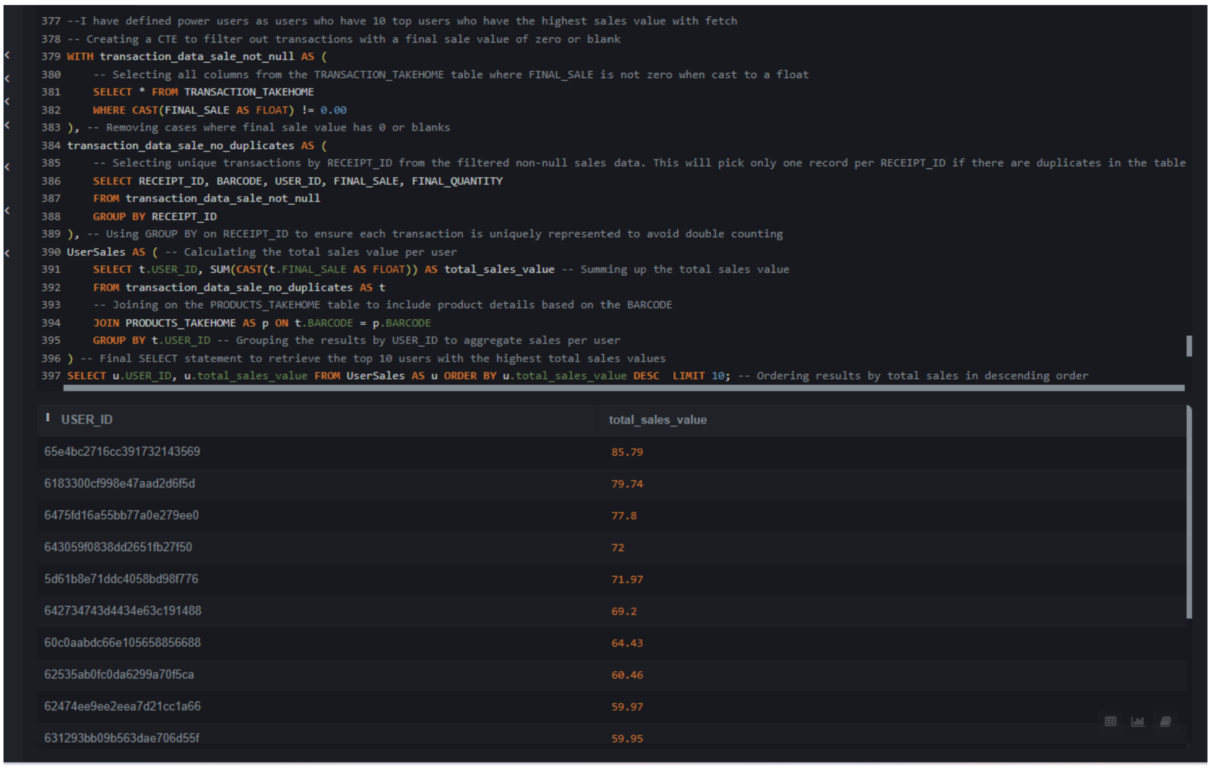
      WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) BETWEEN 57 AND 75 THEN 'Baby Boomers'

      WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) BETWEEN 42 AND 56 THEN 'Gen X'

```
        WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) BETWEEN 27 AND 41 THEN 'Millennials'

        WHEN strftime('%Y', 'now') - strftime('%Y', u.birth_date) <= 26 THEN 'Gen Z'

     END AS Generation,

 SUM(CAST(t.FINAL_SALE AS FLOAT)) AS generation_sales,

 ROUND(SUM(CAST(t.FINAL_SALE AS FLOAT)) * 100.0 /(SELECT SUM(CAST(tr.FINAL_SALE AS FLOAT))

     FROM transaction_data_sale_no_duplicates AS tr

     JOIN PRODUCTS_TAKEHOME AS pr ON tr.BARCODE = pr.BARCODE -- Join transaction and product tables on
barcode.

     JOIN USER_TAKEHOME AS ur ON tr.USER_ID = ur.ID -- Join transaction and user tables on user ID

     WHERE pr.CATEGORY_1 = 'Health & Wellness'), -- Filter for 'Health & Wellness' category products only.

    2

 ) AS percentage_of_sales

FROM transaction_data_sale_no_duplicates AS t

JOIN USER_TAKEHOME AS u ON t.USER_ID = u.ID

JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE

WHERE p.CATEGORY_1 = 'Health & Wellness'

GROUP BY generation ORDER BY percentage_of_sales DESC;
```

Q4. Who are Fetch's power users?

There are two ways to check power users and I have pasted results for both queries. In the first query I have defined power users as users who have 10 top users who have the highest sales value with fetch. In the second query I have defined power users as users who have 10 top users who have the most number of transactions with fetch



```
--I have defined power users as users who have 10 top users who have the highest sales value with fetch
-- Creating a CTE to filter out transactions with a final sale value of zero or blank
WITH transaction_data_sale_not_null AS (
    -- Selecting all columns from the TRANSACTION_TAKEHOME table where FINAL_SALE is not zero when cast to a float
    SELECT * FROM TRANSACTION_TAKEHOME
    WHERE CAST(FINAL_SALE AS FLOAT) != 0.00
), -- Removing cases where final sale value has 0 or blanks
transaction_data_sale_no_duplicates AS (
    -- Selecting unique transactions by RECEIPT_ID from the filtered non-null sales data. This will pick only one record per RECEIPT_ID if there are duplicates in the table
    SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE, FINAL_QUANTITY
    FROM transaction_data_sale_not_null
    GROUP BY RECEIPT_ID
), -- Using GROUP BY on RECEIPT_ID to ensure each transaction is uniquely represented to avoid double counting
UserSales AS ( -- Calculating the total sales value per user
    SELECT t.USER_ID, SUM(CAST(t.FINAL_SALE AS FLOAT)) AS total_sales_value -- Summing up the total sales value
    FROM transaction_data_sale_no_duplicates AS t
    -- Joining on the PRODUCTS_TAKEHOME table to include product details based on the BARCODE
    JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE
    GROUP BY t.USER_ID -- Grouping the results by USER_ID to aggregate sales per user
) -- Final SELECT statement to retrieve the top 10 users with the highest total sales values
SELECT u.USER_ID, u.total_sales_value FROM UserSales AS u ORDER BY u.total_sales_value DESC  LIMIT 10; -- Ordering results by total sales in descending order
```

USER_ID	total_sales_value
65e4bc2716cc391732143569	85.79
6183300cf998e47aad2d6f5d	79.74
6475fd16a55bb77a0e279ee0	77.8
643059f0838dd2651fb27f50	72
5d61b8e71ddc4058bd98f776	71.97
642734743d4434e63c191488	69.2
60c0aabdc66e105658856688	64.43
62535ab0fc0da6299a70f5ca	60.46
62474ee9ee2eea7d21cc1a66	59.97
631293bb09b563dae706d55f	59.95

```
--  Here I have defined power users as users who have 10 top users who have the highest sales value with fetch

-- Creating a CTE to filter out transactions with a final sale value of zero or blank

WITH transaction_data_sale_not_null AS (

   -- Selecting all columns from the TRANSACTION_TAKEHOME table where FINAL_SALE is not zero when cast to a float

   SELECT * FROM TRANSACTION_TAKEHOME

   WHERE CAST(FINAL_SALE AS FLOAT) != 0.00

), -- Removing cases where final sale value has 0 or blanks

transaction_data_sale_no_duplicates AS (

   -- Selecting unique transactions by RECEIPT_ID from the filtered non-null sales data. This will pick only one record per RECEIPT_ID if there are duplicates in the table

   SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE, FINAL_QUANTITY

   FROM transaction_data_sale_not_null

   GROUP BY RECEIPT_ID

), -- Using GROUP BY on RECEIPT_ID to ensure each transaction is uniquely represented to avoid double counting

UserSales AS ( -- Calculating the total sales value per user

   SELECT t.USER_ID, SUM(CAST(t.FINAL_SALE AS FLOAT)) AS total_sales_value -- Summing up the total sales value
```

FROM transaction_data_sale_no_duplicates AS t

-- Joining on the PRODUCTS_TAKEHOME table to include product details based on the BARCODE

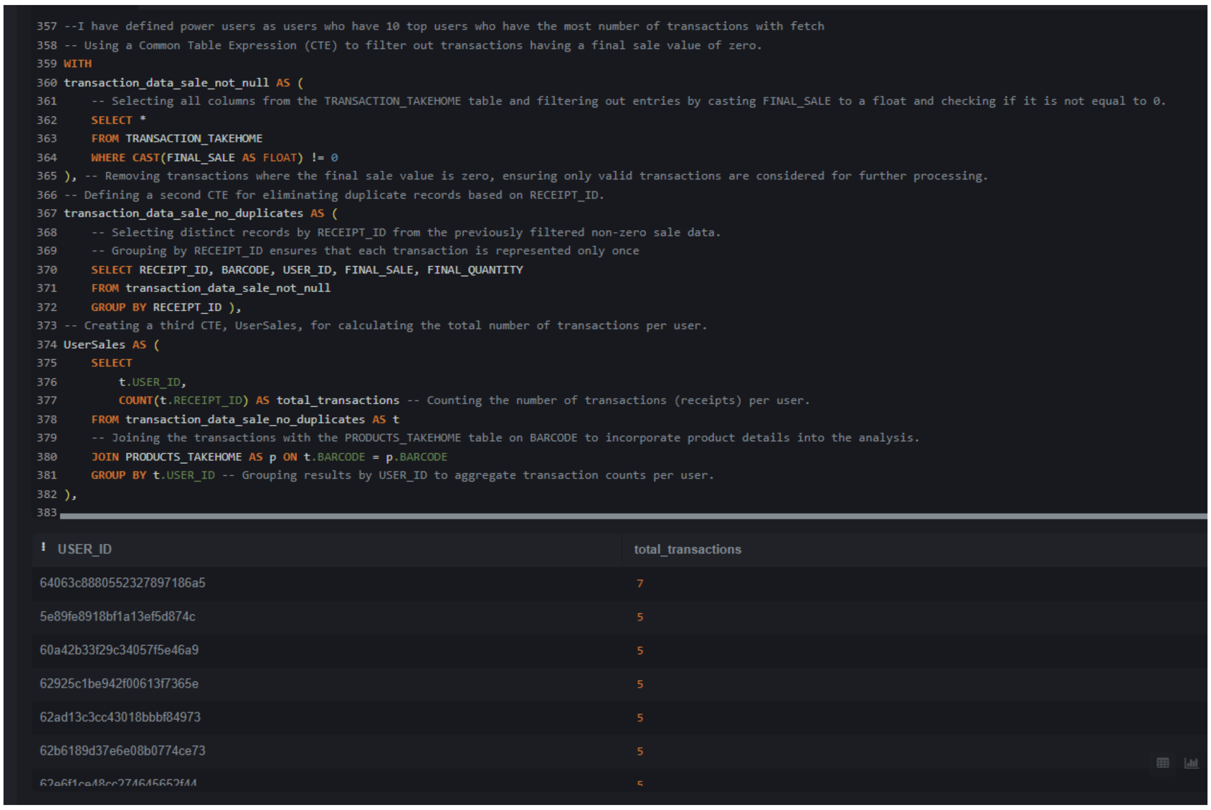
JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE

GROUP BY t.USER_ID -- Grouping the results by USER_ID to aggregate sales per user

) -- Final SELECT statement to retrieve the top 10 users with the highest total sales values

SELECT u.USER_ID, u.total_sales_value FROM UserSales AS u ORDER BY u.total_sales_value DESC LIMIT 10; -- Ordering results by total sales in descending order

Here I have defined power users as users who have 10 top users who have the most number of transactions with fetch.



```
--I have defined power users as users who have 10 top users who have the most number of transactions with fetch
-- Using a Common Table Expression (CTE) to filter out transactions having a final sale value of zero.
WITH
transaction_data_sale_not_null AS (
    -- Selecting all columns from the TRANSACTION_TAKEHOME table and filtering out entries by casting FINAL_SALE to a float and checking if it is not equal to 0.
    SELECT *
    FROM TRANSACTION_TAKEHOME
    WHERE CAST(FINAL_SALE AS FLOAT) != 0
), -- Removing transactions where the final sale value is zero, ensuring only valid transactions are considered for further processing.
-- Defining a second CTE for eliminating duplicate records based on RECEIPT_ID.
transaction_data_sale_no_duplicates AS (
    -- Selecting distinct records by RECEIPT_ID from the previously filtered non-zero sale data.
    -- Grouping by RECEIPT_ID ensures that each transaction is represented only once
    SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE, FINAL_QUANTITY
    FROM transaction_data_sale_not_null
    GROUP BY RECEIPT_ID ),
-- Creating a third CTE, UserSales, for calculating the total number of transactions per user.
UserSales AS (
    SELECT
        t.USER_ID,
        COUNT(t.RECEIPT_ID) AS total_transactions -- Counting the number of transactions (receipts) per user.
    FROM transaction_data_sale_no_duplicates AS t
    -- Joining the transactions with the PRODUCTS_TAKEHOME table on BARCODE to incorporate product details into the analysis.
    JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE
    GROUP BY t.USER_ID -- Grouping results by USER_ID to aggregate transaction counts per user.
),

```

USER_ID	total_transactions
64063c8880552327897186a5	7
5e89fe8918bf1a13ef5d874c	5
60a42b33f29c34057f5e46a9	5
62925c1be942f00613f7365e	5
62ad13c3cc43018bbbf84973	5
62b6189d37e6e08b0774ce73	5
62e6f1ce48cc274645652f44	5

-- Removing cases where final sale value has 0 or blanks. This CTE helps to ensure that only transactions with non-zero sales are considered in further analysis

WITH transaction_data_sale_not_null AS (

SELECT * FROM TRANSACTION_TAKEHOME WHERE CAST(FINAL_SALE AS FLOAT) != 0

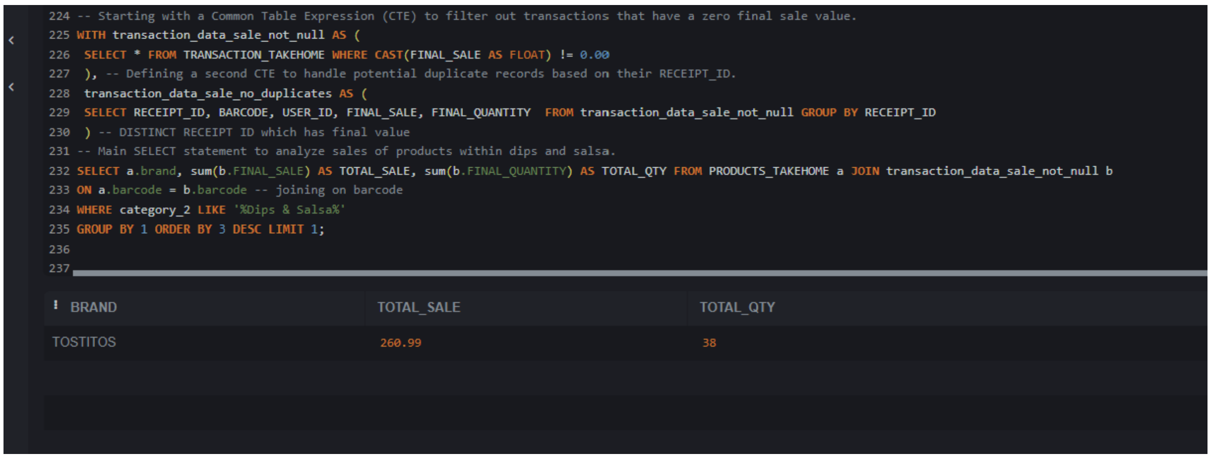
), -- Defining another CTE to handle duplicate records by focusing on unique receipt IDs.

transaction_data_sale_no_duplicates AS (

```
 SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE, FINAL_QUANTITY  FROM transaction_data_sale_not_null
GROUP BY RECEIPT_ID

 ), -- using group by receipt_id to just have one unique value for each transaction to avoid double counting

-- Creating a third CTE, UserSales, to calculate the total number of transactions per user.

UserSales AS (

  SELECT

     t.USER_ID,

     COUNT(t.RECEIPT_ID) AS total_transactions -- count receipt id to check the count of transactions

  FROM transaction_data_sale_no_duplicates AS t

  JOIN PRODUCTS_TAKEHOME AS p ON t.BARCODE = p.BARCODE

  GROUP BY t.USER_ID)

SELECT u.USER_ID, u.total_transactions

FROM UserSales AS u

ORDER BY u.total_transactions DESC

LIMIT 10;
```

Q5 Which is the leading brand in the Dips & Salsa category?

Here I have only considered sales which are > 0 and are unique, removing any double counting and have joined with the products table to get the brand. Category 2 had dips and salsa so I have used category 2 in the where clause.



```
-- Starting with a Common Table Expression (CTE) to filter out transactions that have a zero final sale value.
WITH transaction_data_sale_not_null AS (
 SELECT * FROM TRANSACTION_TAKEHOME WHERE CAST(FINAL_SALE AS FLOAT) != 0.00
 ), -- Defining a second CTE to handle potential duplicate records based on their RECEIPT_ID.
 transaction_data_sale_no_duplicates AS (
 SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE, FINAL_QUANTITY  FROM transaction_data_sale_not_null GROUP BY RECEIPT_ID
 ) -- DISTINCT RECEIPT ID which has final value
-- Main SELECT statement to analyze sales of products within dips and salsa.
SELECT a.brand, sum(b.FINAL_SALE) AS TOTAL_SALE, sum(b.FINAL_QUANTITY) AS TOTAL_QTY FROM PRODUCTS_TAKEHOME a JOIN transaction_data_sale_not_null b
ON a.barcode = b.barcode -- joining on barcode
WHERE category_2 LIKE '%Dips & Salsa%'
GROUP BY 1 ORDER BY 3 DESC LIMIT 1;
```

BRAND	TOTAL_SALE	TOTAL_QTY
TOSTITOS	260.99	38

-- Starting with a Common Table Expression (CTE) to filter out transactions that have a zero final sale value.

WITH transaction_data_sale_not_null AS (

 SELECT * FROM TRANSACTION_TAKEHOME WHERE CAST(FINAL_SALE AS FLOAT) != 0.00

 ), -- Defining a second CTE to handle potential duplicate records based on their RECEIPT_ID.

 transaction_data_sale_no_duplicates AS (

 SELECT RECEIPT_ID, BARCODE, USER_ID, FINAL_SALE, FINAL_QUANTITY FROM transaction_data_sale_not_null GROUP BY RECEIPT_ID

) -- DISTINCT RECEIPT ID which has final value

-- Main SELECT statement to analyze sales of products within dips and salsa.

select a.brand, sum(b.FINAL_SALE) as TOTAL_SALE, sum(b.FINAL_QUANTITY) as TOTAL_QTY from PRODUCTS_TAKEHOME a join transaction_data_sale_not_null b

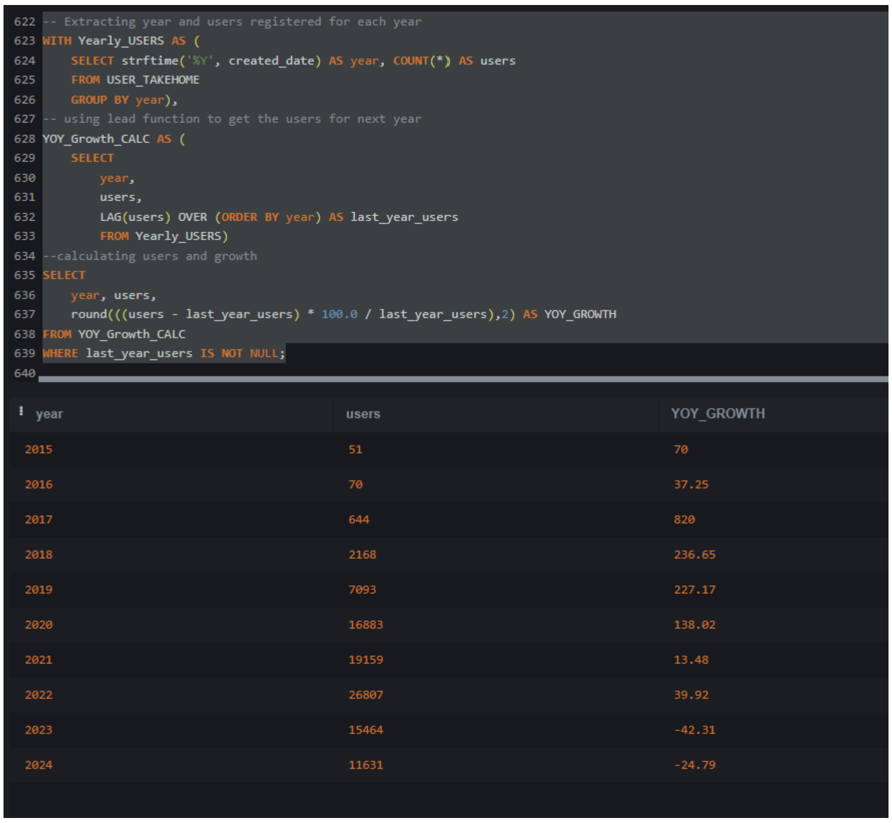
on a.barcode = b.barcode -- joining on barcode

where category_2 like '%Dips & Salsa%'

group by 1 order by 3 desc limit 1;

Q6. At what percent has Fetch grown year over year? – **Answering based on YOY User Signup Growth**

Assumption: I have taken the users table to answer this question since the transactions table only had data for only 2024 year. Since we do not have complete data for 2024 yet (only until August), the number for 2024 can be misleading



```sql
-- Extracting year and users registered for each year
WITH Yearly_USERS AS (
    SELECT strftime('%Y', created_date) AS year, COUNT(*) AS users
    FROM USER_TAKEHOME
    GROUP BY year),
-- using lead function to get the users for next year
YOY_Growth_CALC AS (
    SELECT
        year,
        users,
        LAG(users) OVER (ORDER BY year) AS last_year_users
        FROM Yearly_USERS)
--calculating users and growth
SELECT
    year, users,
    round(((users - last_year_users) * 100.0 / last_year_users),2) AS YOY_GROWTH
FROM YOY_Growth_CALC
WHERE last_year_users IS NOT NULL;
```

year	users	YOY_GROWTH
2015	51	70
2016	70	37.25
2017	644	820
2018	2168	236.65
2019	7093	227.17
2020	16883	138.02
2021	19159	13.48
2022	26807	39.92
2023	15464	-42.31
2024	11631	-24.79

-- Extracting year and users registered for each year

WITH Yearly_USERS AS (

SELECT strftime('%Y', created_date) AS year, COUNT(*) AS users

FROM USER_TAKEHOME

GROUP BY year),

-- using lead function to get the users for next year

YOY_Growth_CALC AS (

SELECT

year,

users,

LAG(users) OVER (ORDER BY year) AS last_year_users

FROM Yearly_USERS)

--calculating users and growth

SELECT

year, users,

round(((users - last_year_users) * 100.0 / last_year_users),2) AS YOY_GROWTH

FROM YOY_Growth_CALC

```
WHERE last_year_users IS NOT NULL;




-- Extracting year and users registered for each year
WITH Yearly_USERS AS (
  SELECT EXTRACT(YEAR FROM created_date) AS year, COUNT(*) AS users
  FROM USER_TAKEHOME
  GROUP BY year
),
-- Using the LAG function to get the users from the previous year
YOY_Growth_CALC AS (
  SELECT
    year,
    users,
    LAG(users) OVER (ORDER BY year) AS last_year_users
  FROM Yearly_USERS
)
-- Calculating YoY growth comparing to the previous year
SELECT
  year,
  users,
  ROUND(((users - last_year_users) * 100.0 / last_year_users), 2) AS YOY_GROWTH
FROM YOY_Growth_CALC
WHERE last_year_users IS NOT NULL;
```