**Introduction**

All of the SQL queries which solve the questions written below are included in the file Queries\_Part\_2.sql, along with comments indicating which is which. They are reproduced in this document in plain text.

**A note on SQL implementations and syntax**

All of the SQL queries were written in a PostgreSQL database; there may be some differences in syntax between PostgreSQL and other SQL server tools. For example, the query

SELECT TOP 10 \* FROM TABLE;

in SQL Server would be implemented as

SELECT \* FROM TABLE LIMIT 10;

in PostgreSQL. All of the queries written function correctly in a PostgreSQL implementation where the products, transaction, and user files are imported as PRODUCTS, TRANSACTIONS, and USERS respectively.

One key difference in syntax between PostgreSQL and some other implementations of SQL involves performing calculations on columns computed by a query. Some implementations of SQL allow a user to calculate a new column within a query and also perform a mathematical operation on that computed column to compute a second column. PostgreSQL does not allow this directly, so I wrote the main query logic as a subquery and wrapped that subquery in another query to perform calculations on a calculated column. This method does not induce significant slowdowns in performance.

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

To answer this question, I interpreted “users 21 and over” to represent the user’s age at the time of the purchase instead of the user’s present age. I also removed null entries from consideration as a viable brand.

SELECT BRAND, N, RANK() OVER (ORDER BY N DESC) AS BRAND\_RANK

FROM (SELECT BRAND,COUNT(DISTINCT T.RECEIPT\_ID) AS N

FROM (TRANSACTIONS AS T INNER JOIN USERS AS U ON T.USER\_ID = U.ID)

LEFT JOIN PRODUCTS AS P

ON T.BARCODE = P.BARCODE

WHERE CAST(U.BIRTH\_DATE AS DATE) <= CAST(T.PURCHASE\_DATE AS DATE) - INTERVAL '21' YEAR

AND P.BRAND IS NOT NULL

GROUP BY P.BRAND

ORDER BY N DESC

FETCH FIRST 5 ROWS WITH TIES);

By checking that FINAL\_SALE is not null and FINAL\_QUANTITY is nonzero, the duplicate rows which had non-numeric values in the original transaction file are ignored. Additionally, I used the main query logic as a sub-query and wrapped it in a function to compute each brand’s rank based on the number of receipts which contain that brand. Finally, I counted only distinct Receipt IDs to account for the scenario where a single receipt contains multiple different products of the same brand; this condition also corrects for the duplicate rows in the Transaction file, so I did not need to separately check for those.

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

I used the following definition of generations, taken from [this guide](https://www.parents.com/parenting/better-parenting/style/generation-names-and-years-a-cheat-sheet-for-parents/):

* Greatest Generation: born 1901-1927
* Silent Generation: born 1928-1945
* Baby Boomers: born 1946-1964
* Generation X: born 1965-1980
* Millennials: born 1981-1996
* Generation Z: born 1997-2010
* Generation Alpha: born 2011-2024

The following query retrieves the percentage of sales from the Health & Wellness category to users born in each generation, as well as the total value of the sales. It computes the percentage of sales only out of sales which can be traced to a specific user.

SELECT GENERATION, N, SALES\_TOTAL, SALES\_TOTAL/SUM(SALES\_TOTAL) OVER() AS SALES\_PCT

FROM(

SELECT CASE

WHEN EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) >= 1901 AND EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) <= 1927 THEN '0\_GREATEST'

WHEN EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) >= 1928 AND EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) <= 1945 THEN '1\_SILENT'

WHEN EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) >= 1946 AND EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) <= 1964 THEN '2\_BABY\_BOOMER'

WHEN EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) >= 1965 AND EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) <= 1980 THEN '3\_GEN\_X'

WHEN EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) >= 1981 AND EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) <= 1996 THEN '4\_MILLENNIAL'

WHEN EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) >= 1997 AND EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) <= 2010 THEN '5\_GEN\_Z'

WHEN EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) >= 2011 AND EXTRACT(YEAR FROM CAST(U.BIRTH\_DATE AS DATE)) <= 2024 THEN '6\_GEN\_ALPHA'

ELSE 'LIKELY\_ERROR'

END AS GENERATION,

COUNT(\*) AS N,

SUM(T.FINAL\_SALE) AS SALES\_TOTAL

FROM ((TRANSACTIONS AS T INNER JOIN USERS AS U ON T.USER\_ID = U.ID)

INNER JOIN PRODUCTS AS P

ON T.BARCODE = P.BARCODE)

WHERE P.CATEGORY\_1 = 'Health & Wellness'

AND T.FINAL\_SALE IS NOT NULL

AND T.FINAL\_QUANTITY > 0.0

GROUP BY GENERATION

ORDER BY GENERATION);

To make the percentage calculation of this query run properly in PostgreSQL, I included the primary logic as a subquery and referenced SALES\_TOTAL from the subquery within the percentage calculation. This step is not necessary in all SQL implementations, but it should be noted that using a subquery in this manner does not create performance issues. Within the query, each generation is assigned a numerical index as part of its name to ensure that the generations are sorted in chronological order.

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

I used the following query; initially, I used a modified version which checked all four category fields in the products file, and then modified it when I discovered that Dips & Salsa is in Category\_2.

SELECT P.BRAND, SUM(T.FINAL\_SALE) AS FINAL\_SALE\_SUM, SUM(T.FINAL\_QUANTITY), P.CATEGORY\_2

FROM TRANSACTIONS AS T

LEFT JOIN PRODUCTS AS P

ON T.BARCODE = P.BARCODE

WHERE P.CATEGORY\_2 = 'Dips & Salsa'

AND T.FINAL\_SALE IS NOT NULL

AND T.FINAL\_QUANTITY > 0.0

AND P.BRAND IS NOT NULL

GROUP BY P.BRAND,P.CATEGORY\_2

ORDER BY FINAL\_SALE\_SUM DESC;

From this query, I verified that Tostitos leads in both total number and total cost of items sold based on the existing transactions data. Since user information is not necessary to answer this question, I was able to use the full transactions data set as presented instead of limiting my query to the transactions associated with user IDs in the users data set.