Fetch Rewards Take Home Part 2

In [152…
```
import psycopg2
from psycopg2.extras import execute_values
import pandas as pd
```

In [153…
```
# File paths
products_file = '/Users/etc1999/Downloads/PRODUCTS_TAKEHOME.csv'
transactions_file = '/Users/etc1999/Downloads/TRANSACTION_TAKEHOME.csv'
users_file = '/Users/etc1999/Downloads/USER_TAKEHOME.csv'
```

In [154…
```
# Read CSV files
products_df = pd.read_csv(products_file)
transactions_df = pd.read_csv(transactions_file)
users_df = pd.read_csv(users_file)
```

In [155…
```
# Data cleaning: Convert BARCODE to string and remove '.0'
products_df['BARCODE'] = products_df['BARCODE'].fillna(-1).astype(str).str.replace(r'\.0$', '', regex=True)
transactions_df['BARCODE'] = transactions_df['BARCODE'].fillna(-1).astype(str).str.replace(r'\.0$', '', regex=T

# Remove duplicate receipt_id values
transactions_df = transactions_df.drop_duplicates(subset=['RECEIPT_ID'])

# Clean all date columns in the users_df
for col in ['CREATED_DATE', 'BIRTH_DATE']:
    users_df[col] = pd.to_datetime(users_df[col].str.replace(' Z', '', regex=False), errors='coerce')  # Conver
    users_df[col] = users_df[col].apply(lambda x: x.isoformat() if pd.notna(x) else None)  # Replace NaT with N
```

In [156…
```
# Database connection parameters
db_params = {
    'dbname': 'trial',  # Replace with your database name
    'user': 'postgres',         # Replace with your PostgreSQL username
    'password': '',             # Replace with your PostgreSQL password
    'host': 'localhost',
    'port': 5432
}

# SQL commands to create tables
CREATE_TABLES_SQL = [
    """
    DROP TABLE IF EXISTS products CASCADE;
    CREATE TABLE products (
        category_1 VARCHAR,
        category_2 VARCHAR,
        category_3 VARCHAR,
        category_4 VARCHAR,
        manufacturer VARCHAR,
        brand VARCHAR,
        barcode TEXT  -- Changed to TEXT to handle large values
    );
    """,
    """
    DROP TABLE IF EXISTS transactions CASCADE;
    CREATE TABLE transactions (
        receipt_id VARCHAR PRIMARY KEY,
        purchase_date DATE,
        scan_date TIMESTAMP,
        store_name VARCHAR,
        user_id VARCHAR,
        barcode TEXT,  -- Changed to TEXT to match products table
        final_quantity VARCHAR,
        final_sale VARCHAR
    );
    """,
    """
    DROP TABLE IF EXISTS users CASCADE;
    CREATE TABLE users (
        id VARCHAR PRIMARY KEY,
        created_date TIMESTAMP,
        birth_date TIMESTAMP,
        state VARCHAR,
        language VARCHAR,
        gender VARCHAR
    );
    """
]

# Function to create tables
def create_tables(conn):
    with conn.cursor() as cur:
        for query in CREATE_TABLES_SQL:
            cur.execute(query)
        conn.commit()
    print("Tables created successfully!")
```

```
# Function to insert data into the database
def insert_data_to_db(df, table_name, conn):
    # Convert DataFrame to a list of tuples
    tuples = [tuple(x) for x in df.to_numpy()]
    cols = ','.join(list(df.columns))

    # Create SQL query
    query = f"INSERT INTO {table_name} ({cols}) VALUES %s"

    # Execute query
    with conn.cursor() as cur:
        execute_values(cur, query, tuples)
        conn.commit()
```

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

In this query, I analyzed the data to identify the top five brands based on the number of receipts scanned by users aged 21 and older. I combined the transactions, users, and products datasets by linking user IDs and barcodes. The data was filtered to exclude users with missing birth dates (to ensure valid age calculations) and products with missing or invalid brand names. The age of each user was determined using their birth date, and only those 21 years or older were considered. Finally, I grouped the data by brand and ranked them by the count of associated receipts, retrieving the top five brands.

In [157…

```
try:
    # Connect to the database
    conn = psycopg2.connect(**db_params)

    # Drop and create tables
    create_tables(conn)

    # Validate and inspect data
    print("Checking for duplicate RECEIPT_ID values...")
    if transactions_df['RECEIPT_ID'].duplicated().any():
        raise ValueError("Duplicate RECEIPT_ID values found in transactions data!")
    print("No duplicates found. Proceeding with insertion.")

    # Insert data into the database
    insert_data_to_db(products_df, 'products', conn)
    insert_data_to_db(transactions_df, 'transactions', conn)
    insert_data_to_db(users_df, 'users', conn)

    print("Data inserted successfully!")

    with conn.cursor() as cur:
        # Query: Top 5 brands by receipts scanned among users 21 and over
        cur.execute("""
            SELECT p.brand, COUNT(t.receipt_id) AS receipt_count
            FROM transactions t
            JOIN users u ON t.user_id = u.id
            JOIN products p ON t.barcode = p.barcode
            WHERE u.birth_date IS NOT NULL
              AND DATE_PART('year', AGE(u.birth_date)) >= 21
              AND p.brand IS NOT NULL AND p.brand != 'NaN'
            GROUP BY p.brand
            ORDER BY receipt_count DESC
            LIMIT 5;
        """)
        top_brands = cur.fetchall()
        print("Top 5 Brands by Receipts Scanned (21 and Over):")
        for brand in top_brands:
            print(brand)
except Exception as e:
    print(f"Error occurred: {e}")
```

```
Tables created successfully!
Checking for duplicate RECEIPT_ID values...
No duplicates found. Proceeding with insertion.
Data inserted successfully!
Top 5 Brands by Receipts Scanned (21 and Over):
('COCA-COLA', 314)
("ANNIE'S HOMEGROWN GROCERY", 288)
('DOVE', 279)
('BAREFOOT', 276)
('ORIBE', 252)
```

The top five brands by receipts scanned were Coca-Cola (314 receipts), Annie's Homegrown Grocery (288 receipts), Dove (279 receipts), Barefoot (276 receipts), and Oribe (252 receipts). These brands have the highest engagement from users aged 21 and older, indicating their popularity within this demographic.

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

This query analyzed sales in the Health & Wellness category across different generations. Users were assigned to generational groups based on their age, calculated using their birth dates. The query considered only transactions involving products in the Health & Wellness category and excluded users with missing birth dates. Sales data was aggregated by generation, and the percentage of total category

sales contributed by each generation was calculated. Generations included Silent Generation (77+ years), Boomers (58–76 years), Gen X (42–57 years), Millennials (26–41 years), and Gen Z (10–25 years), with users outside these age ranges categorized as "Other."

In [158…

```
try:
    with conn.cursor() as cur:
        # Query: Percentage of Sales in Health & Wellness Category by Generation
        cur.execute("""
        WITH generation_sales AS (
            SELECT
                CASE
                    WHEN DATE_PART('year', AGE(u.birth_date)) >= 77 THEN 'Silent Generation'
                    WHEN DATE_PART('year', AGE(u.birth_date)) BETWEEN 58 AND 76 THEN 'Boomers'
                    WHEN DATE_PART('year', AGE(u.birth_date)) BETWEEN 42 AND 57 THEN 'Gen X'
                    WHEN DATE_PART('year', AGE(u.birth_date)) BETWEEN 26 AND 41 THEN 'Millennials'
                    WHEN DATE_PART('year', AGE(u.birth_date)) BETWEEN 10 AND 25 THEN 'Gen Z'
                    WHEN DATE_PART('year', AGE(u.birth_date)) < 10 THEN 'Generation Alpha'
                    ELSE 'Unknown'
                END AS generation,
                SUM(CAST(t.final_sale AS FLOAT)) AS generation_sales
            FROM
                transactions t
            JOIN
                users u ON t.user_id = u.id
            JOIN
                products p ON t.barcode = p.barcode
            WHERE
                t.final_sale ~ '^\\d+(\\.\\d+)?$'
                AND p.category_1 = 'Health & Wellness'
                AND u.birth_date IS NOT NULL
            GROUP BY
                generation
        ),
        total_sales AS (
            SELECT
                SUM(generation_sales) AS total_health_wellness_sales
            FROM
                generation_sales
        )
        SELECT
            g.generation,
            g.generation_sales,
            ROUND(
                (g.generation_sales / t.total_health_wellness_sales)::NUMERIC,
                2
            ) * 100 AS percentage_of_sales
        FROM
            generation_sales g
        CROSS JOIN
            total_sales t
        ORDER BY
            percentage_of_sales DESC;
        """)

        results = cur.fetchall()

        if results:
            print("\nPercentage of Sales in Health & Wellness Category by Generation:")
            for row in results:
                print(f"Generation: {row[0]}, Sales: {row[1]:.2f}, Percentage: {row[2]:.2f}%")
        else:
            print("\nNo data found for the Health & Wellness category.")
except Exception as e:
    print(f"Error occurred: {e}")
```

```
Percentage of Sales in Health & Wellness Category by Generation:
Generation: Gen X, Sales: 27637.56, Percentage: 36.00%
Generation: Millennials, Sales: 26027.65, Percentage: 34.00%
Generation: Boomers, Sales: 22070.89, Percentage: 29.00%
```

Findings: The analysis revealed that Gen X contributed the most to Health & Wellness sales at 36%, followed by Millennials at 34% and Boomers at 29%. This indicates that the younger generations (Gen X and Millennials) drive most of the sales in this category, highlighting their preference for health-conscious products.

Who are Fetch's power users?

This query aimed to identify "power users" of Fetch Rewards by analyzing the receipt activity of users. The number of receipts scanned by each user was calculated by grouping the data by user IDs. A threshold for power users was established using the 99th percentile of receipt counts. Users whose receipt counts met or exceeded this threshold were identified as power users and ranked in descending order of receipt counts. The data excluded users with invalid or missing IDs to maintain accuracy.

In [142…

```
try:
    with conn.cursor() as cur:
        # Query: Fetch's Power Users
        cur.execute("""
            WITH user_receipt_counts AS (
```

```
                SELECT u.id AS user_id, COUNT(t.receipt_id) AS receipt_count
                FROM transactions t
                JOIN users u ON t.user_id = u.id
                GROUP BY u.id
            )
            SELECT user_id, receipt_count
            FROM user_receipt_counts
            WHERE receipt_count >= (SELECT PERCENTILE_CONT(0.99) WITHIN GROUP (ORDER BY receipt_count) FROM use
            ORDER BY receipt_count DESC;
        """)

        power_users = cur.fetchall()
        print("\nFetch's Power Users:")
        for user in power_users:
            print(user)
except Exception as e:
    print(f"Error occurred: {e}")
finally:
    if conn:
        conn.close()
        print("Database connection closed.")
```

```
Fetch's Power Users:
('62ffec490d9dbaff18c0a999', 3)
('5c366bf06d9819129dfa1118', 3)
('62c09104baa38d1a1f6c260e', 3)
('610a8541ca1fab5b417b5d33', 3)
('6528a0a388a3a884364d94dc', 3)
('61a58ac49c135b462ccddd1c', 3)
Database connection closed.
```

Findings: Fetch's power users were identified as users with three or more receipts scanned, corresponding to the 99th percentile. The analysis uncovered six power users, each with exactly three receipts scanned. These users represent the most engaged portion of the Fetch Rewards user base, likely contributing significantly to overall platform activity.