SQL CASE STUDY : DATA MART ANALYSIS

INTRODUCTION:

Data Dart is my latest venture and I want your help to analyze the sales and performance of my venture. In June 2020 - large scale supply changes were made at Data Mart. All Data Mart products now use sustainable packaging methods in every single step from the farm all the way to the customer.

I need your help to quantify the impact of this change on the sales performance for Data Mart and its separate business areas.

CREATE DATABASE & TABLES QUERY_CODE:

Link :

SCHEMA USED: WEEKLY_SALES TABLE

Column name	Data type
week_date	date
region	varchar(20)
platform	varchar(20)
segment	varchar(10)
customer	varchar(20)
transactions	int
sales	int

CASE STUDY QUESTIONS

A. Data Cleansing Steps:

In a single query, perform the following operations and generate a new table in the data_mart schema named clean_weekly_sales:

1. Add a week_number as the second column for each week_date value, for example any value from the 1st of January to 7th of January will be 1, 8th to 14th will be 2, etc.
2. Add a month_number with the calendar month for each week_date value as the 3rd column
3. Add a calendar_year column as the 4th column containing either 2018, 2019 or 2020 values
4. Add a new column called age_band after the original segment column using the following mapping on the number inside the segment value

segment	age_band
1	Young Adults
2	Middle Aged
3 or 4	Retirees

5. Add a new demographic column using the following mapping for the first letter in the segment values:

segment	demographic
C	Couples
F	Families

6. Ensure all null string values with an "unknown" string value in the original segment column as well as the new age_band and demographic columns
7. Generate a new avg_transaction column as the sales value divided by transactions rounded to 2 decimal places for each record

❖ Query:

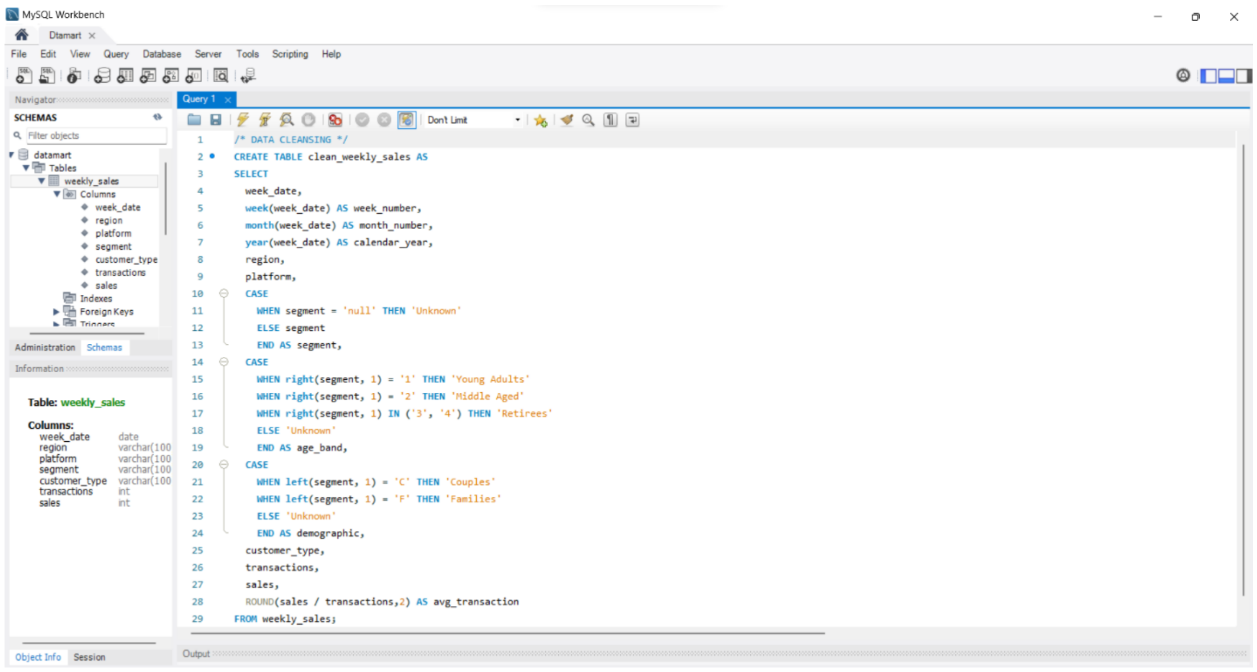


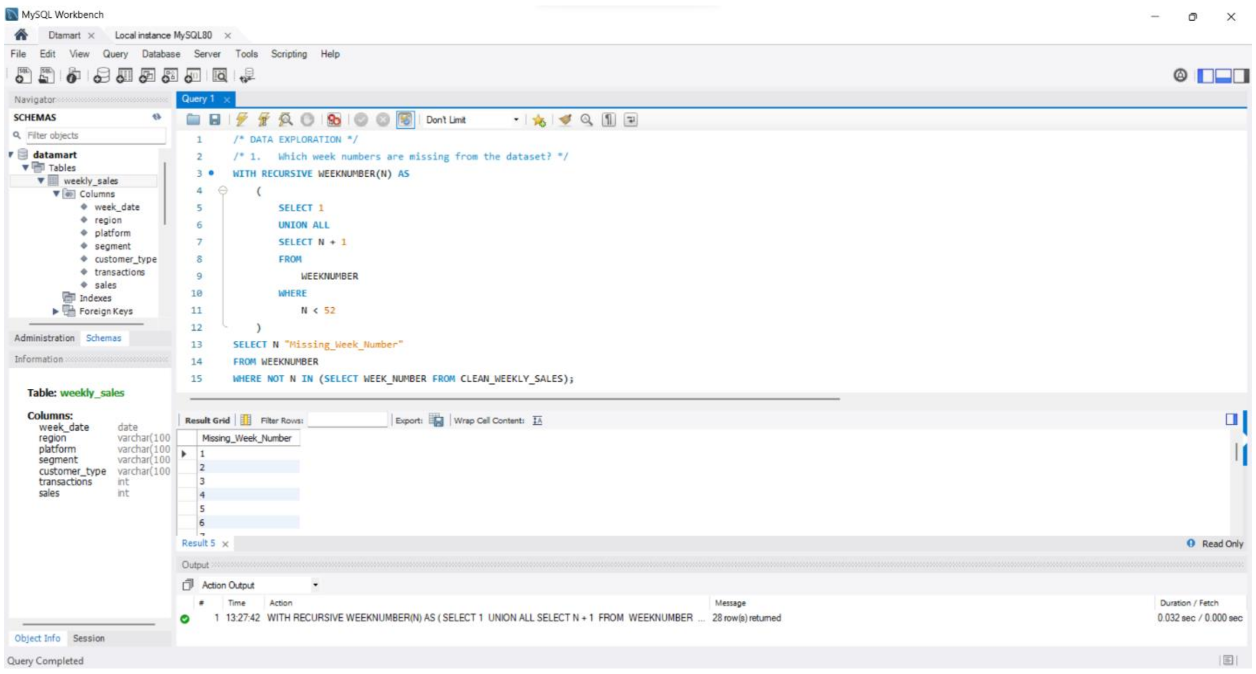
Table: weekly_sales

Columns:
```
week_date      date
region         varchar(100
platform       varchar(100
segment        varchar(100
customer_type  varchar(100
transactions   int
sales          int
```

```sql
/* DATA CLEANSING */
CREATE TABLE clean_weekly_sales AS
SELECT
  week_date,
  week(week_date) AS week_number,
  month(week_date) AS month_number,
  year(week_date) AS calendar_year,
  region,
  platform,
  CASE
    WHEN segment = 'null' THEN 'Unknown'
    ELSE segment
    END AS segment,
  CASE
    WHEN right(segment, 1) = '1' THEN 'Young Adults'
    WHEN right(segment, 1) = '2' THEN 'Middle Aged'
    WHEN right(segment, 1) IN ('3', '4') THEN 'Retirees'
    ELSE 'Unknown'
    END AS age_band,
  CASE
    WHEN left(segment, 1) = 'C' THEN 'Couples'
    WHEN left(segment, 1) = 'F' THEN 'Families'
    ELSE 'Unknown'
    END AS demographic,
  customer_type,
  transactions,
  sales,
  ROUND(sales / transactions,2) AS avg_transaction
FROM weekly_sales;
```

B. Data Exploration

1. Which week numbers are missing from the dataset?



```
/* DATA EXPLORATION */
/* 1.   Which week numbers are missing from the dataset? */
WITH RECURSIVE WEEKNUMBER(N) AS
    (
        SELECT 1
        UNION ALL
        SELECT N + 1
        FROM
            WEEKNUMBER
        WHERE
            N < 52
    )
SELECT N "Missing_Week_Number"
FROM WEEKNUMBER
WHERE NOT N IN (SELECT WEEK_NUMBER FROM CLEAN_WEEKLY_SALES);
```

```
Missing_Week_Number
1
2
3
4
5
6
```

```
#  Time      Action                                                                           Message               Duration / Fetch
1  13:27:42  WITH RECURSIVE WEEKNUMBER(N) AS ( SELECT 1  UNION ALL SELECT N + 1  FROM  WEEKNUMBER ...  28 row(s) returned   0.032 sec / 0.000 sec
```

# This MySQL query is a common example of using a recursive common table expression (CTE) to generate a list of week numbers from 1 to 52 and then filtering that list to exclude the week numbers that exist in a table called CLEAN_WEEKLY_SALES.

# Here's a breakdown of the query step by step: Recursive CTE (Common Table Expression):

# The query starts with a recursive CTE named WEEKNUMBER. This CTE generates a list of week numbers starting from 1.

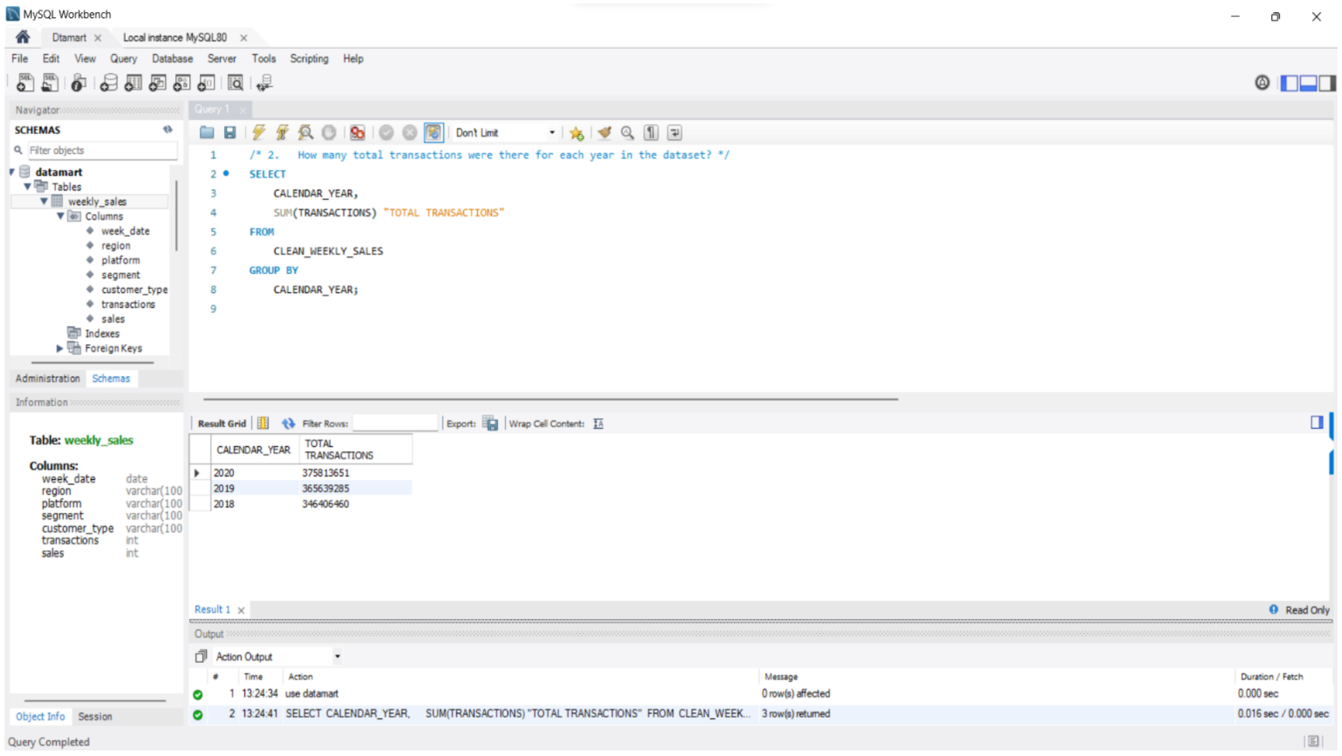
# It begins with an anchor member, which is the SELECT 1. This sets the initial value of N to 1. The recursive member, SELECT N + 1 FROM WEEKNUMBER WHERE N < 52, adds 1 to the current N value and continues to generate rows as long as N is less than 52.

# Main Query: The main query selects the values of N from the WEEKNUMBER CTE but filters out the week numbers that are present in the CLEAN_WEEKLY_SALES table.

# It retrieves all values of N from the WEEKNUMBER CTE. The WHERE NOT N IN (SELECT WEEK_NUMBER FROM CLEAN_WEEKLY_SALES) clause filters the week numbers that are not found in the CLEAN_WEEKLY_SALES table.

# In other words, it excludes the week numbers that already exist in the CLEAN_WEEKLY_SALES table. In summary, this query generates a list of week numbers from 1 to 52 and then filters this list to return only the week numbers that are not already present in the CLEAN_WEEKLY_SALES table. This can be useful for finding missing or unused week numbers in the context of weekly sales data.

2. How many total transactions were there for each year in the dataset?



```sql
/* 2.  How many total transactions were there for each year in the dataset? */
SELECT
    CALENDAR_YEAR,
    SUM(TRANSACTIONS) "TOTAL TRANSACTIONS"
FROM
    CLEAN_WEEKLY_SALES
GROUP BY
    CALENDAR_YEAR;
```

Table: weekly_sales

Columns:

week_date	date
region	varchar(100
platform	varchar(100
segment	varchar(100
customer_type	varchar(100
transactions	int
sales	int

CALENDAR_YEAR	TOTAL TRANSACTIONS
2020	375813651
2019	365639285
2018	346406460

```
#  Time      Action                                                                        Message              Duration / Fetch
1  13:24:34  use datamart                                                                  0 row(s) affected    0.000 sec
2  13:24:41  SELECT CALENDAR_YEAR,   SUM(TRANSACTIONS) "TOTAL TRANSACTIONS"  FROM CLEAN_WEEK...  3 row(s) returned    0.016 sec / 0.000 sec
```

Query Completed

# This SQL query is designed to retrieve and summarize data from a dataset called "CLEAN WEEKLY SALES." Let's break down the query and provide a description for it:

# SELECT CALENDAR YEAR, SUM(TRANSACTIONS) TOTAL TRANSACTIONS

# This part of the query specifies what data you want to retrieve. It selects two columns: "CALENDAR YEAR" and the sum of "TRANSACTIONS." The latter is renamed as "TOTAL TRANSACTIONS" in the result set.

# FROM CLEAN WEEKLY SALES

# This part of the query indicates the data source. It tells the database to fetch data from the "CLEAN WEEKLY SALES" dataset or table.

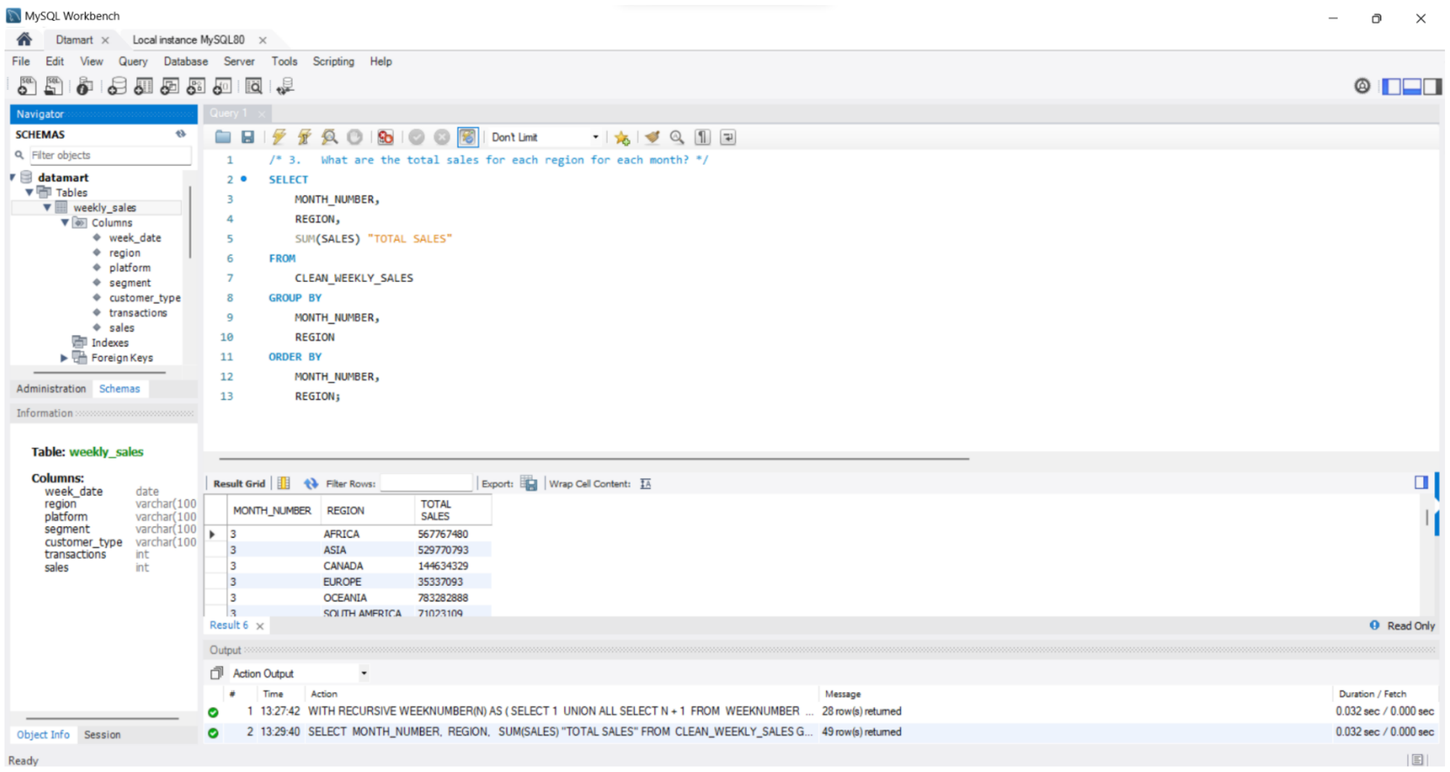
# GROUP BY CALENDAR_YEAR

# This part of the query instructs the database to group the data based on the values in the "CALENDAR YEAR" column. In other words, it will group the transactions by each unique year.

# Description: This SQL query is used for data analysis and reporting. It retrieves information from the "CLEAN WEEKLY SALES" dataset and provides a summary of the total transactions for each distinct calendar year. By grouping the data by calendar year, it allows you to see the total transactions for each year separately. This type of query is commonly used for generating reports and gaining insights into transaction data over time.

# The result of this query will be a table with two columns: "CALENDAR YEAR" and "TOTAL TRANSACTIONS," where each row represents a calendar year and its corresponding total transactions.This can be useful for tracking and analyzing transaction trends over different years, which can be valuable for making business decisions and identifying patterns in sales data.

3. What are the total sales for each region for each month?



```
/* 3.   What are the total sales for each region for each month? */
SELECT
    MONTH_NUMBER,
    REGION,
    SUM(SALES) "TOTAL SALES"
FROM
    CLEAN_WEEKLY_SALES
GROUP BY
    MONTH_NUMBER,
    REGION
ORDER BY
    MONTH_NUMBER,
    REGION;
```

MONTH_NUMBER	REGION	TOTAL SALES
3	AFRICA	567767480
3	ASIA	529770793
3	CANADA	144634329
3	EUROPE	35337093
3	OCEANIA	783282888
3	SOUTH AMERICA	71023109

```
#   Time      Action                                                                        Message              Duration / Fetch
1   13:27:42  WITH RECURSIVE WEEKNUMBER(N) AS ( SELECT 1  UNION ALL SELECT N + 1  FROM  WEEKNUMBER ...   28 row(s) returned   0.032 sec / 0.000 sec
2   13:29:40  SELECT  MONTH_NUMBER,  REGION,  SUM(SALES) "TOTAL SALES" FROM  CLEAN_WEEKLY_SALES G...    49 row(s) returned   0.032 sec / 0.000 sec
```

# This SQL query appears to retrieve data from a table called "CLEAN WEEKLY SALES" and perform some calculations on it. Here's a description of the query:

# The SELECT statement is used to specify the columns that will be included in the result set.

# "MONTH NUMBER" seems to represent the month of the sales data.

# "REGION" represents the region where the sales data is collected.

# "SUM(SALES) 'TOTAL SALES'" calculates the total sales for each combination of month and region.

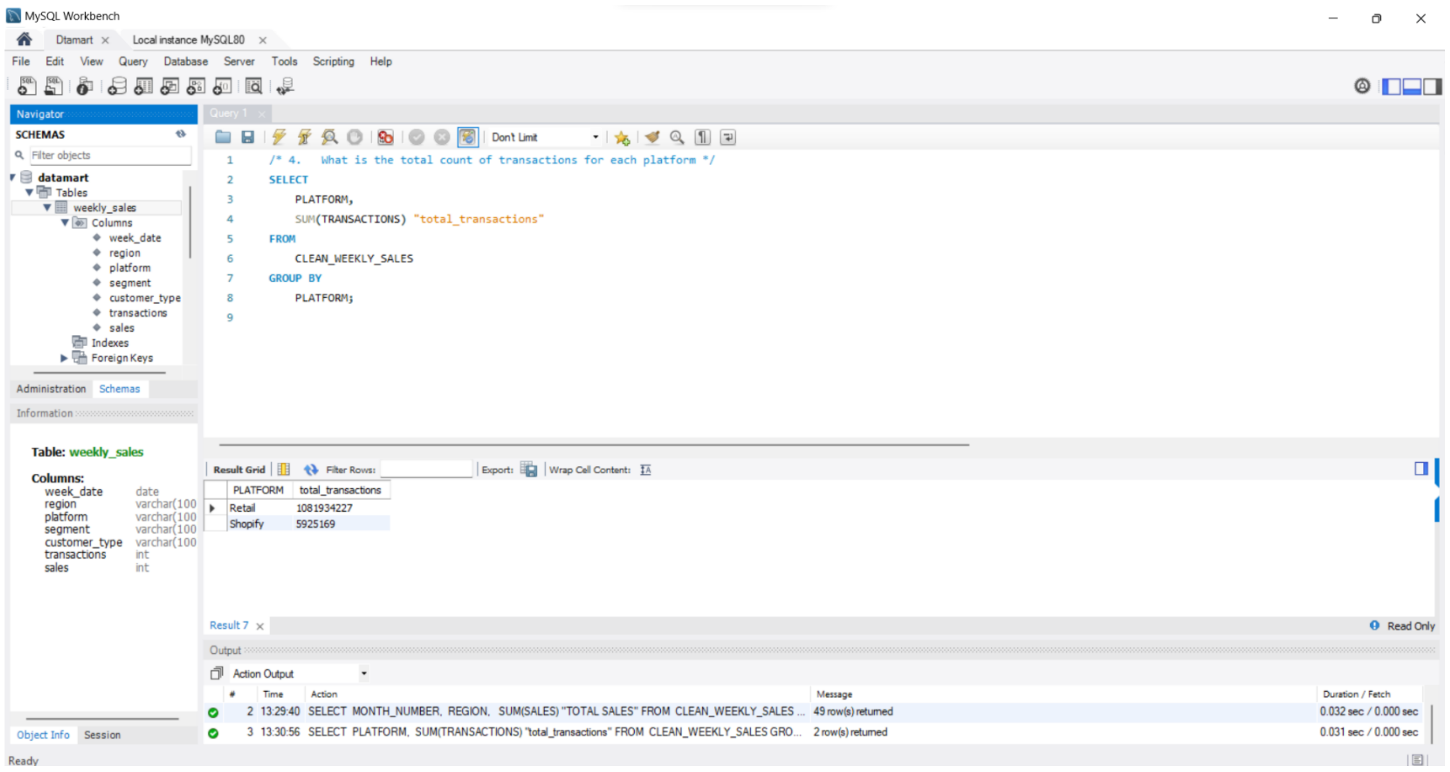
# The FROM clause indicates that the data is being retrieved from the "CLEAN WEEKLY SALES" table.

# The GROUP BY clause is used to group the data based on the specified columns, in this case, "MONTH NUMBER" and "REGION." This means that the query will calculate the total sales for each unique combination of month and region.

# The ORDER BY clause arranges the results in ascending order of "MONTH NUMBER" and "REGION," ensuring that the output is organized in a structured manner.

# In summary, this query is aggregating and organizing sales data by month and region, providing the total sales for each combination of these two criteria.

4. What is the total count of transactions for each platform?



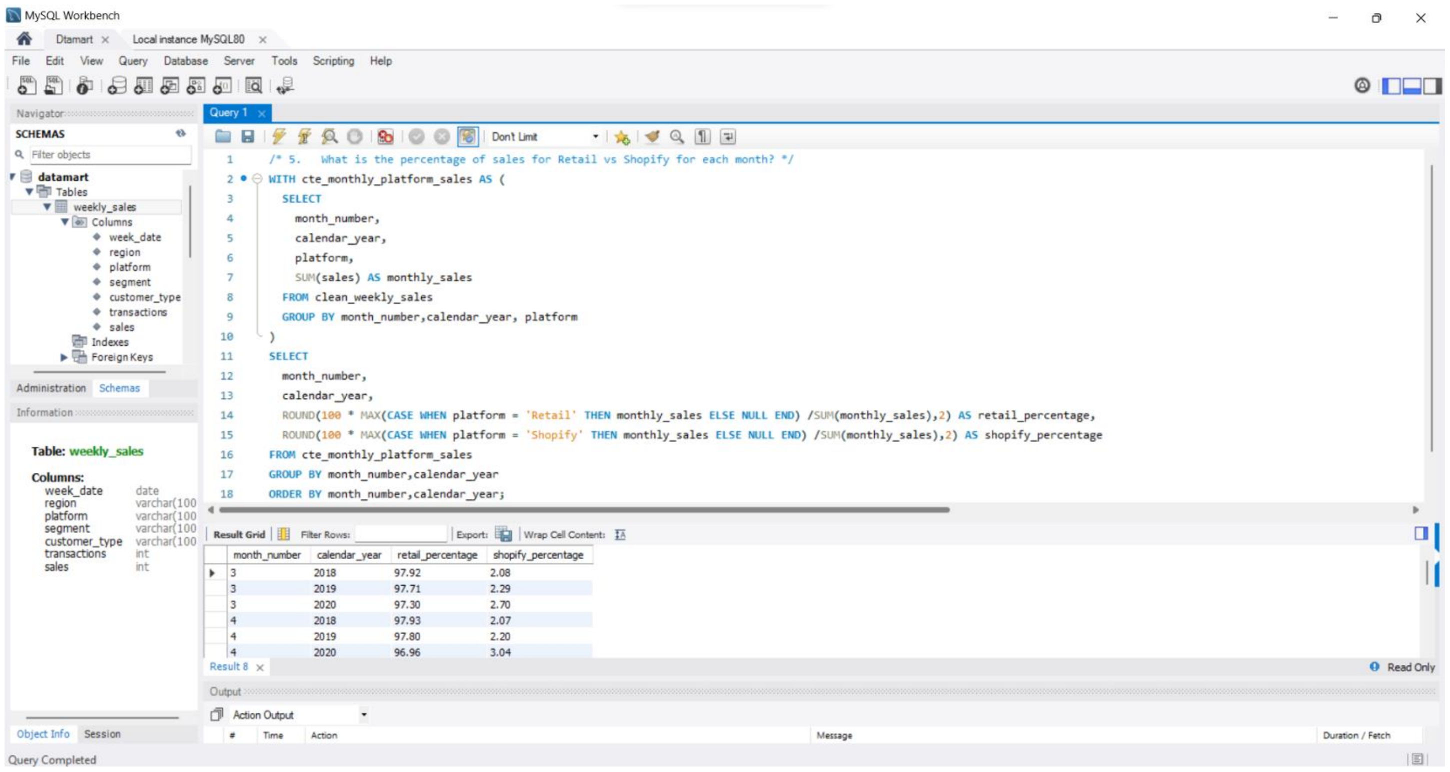
```sql
/* 4.  What is the total count of transactions for each platform */
SELECT
    PLATFORM,
    SUM(TRANSACTIONS) "total_transactions"
FROM
    CLEAN_WEEKLY_SALES
GROUP BY
    PLATFORM;
```

PLATFORM	total_transactions
Retail	1081934227
Shopify	5925169

```
#  Time      Action                                                                        Message             Duration / Fetch
2  13:29:40  SELECT MONTH_NUMBER, REGION, SUM(SALES) "TOTAL SALES" FROM CLEAN_WEEKLY_SALES ...  49 row(s) returned  0.032 sec / 0.000 sec
3  13:30:56  SELECT PLATFORM, SUM(TRANSACTIONS) "total_transactions" FROM CLEAN_WEEKLY_SALES GRO...  2 row(s) returned  0.031 sec / 0.000 sec
```

# This SQL query is used to retrieve and summarize data from a table called "CLEAN WEEKLY_SALES." The purpose of this query is to provide insights into transaction data on different platforms. Here's a description of the query:
# SELECT PLATFORM, SUM(TRANSACTIONS) "total_transactions":
# This part of the query selects two columns from the table. It retrieves the "PLATFORM" column, which likely contains information about the platforms where transactions occurred, and it calculates the sum of the "TRANSACTIONS" column, naming the result as "total_transactions."
# FROM CLEAN WEEKLY_SALES:
# This specifies the source table, which is named "CLEAN WEEKLY_SALES." The query will fetch data from this table for further processing.
# GROUP BY PLATFORM:
# This part of the query groups the data by the "PLATFORM" column. It means that the sum of transactions will be calculated for each unique platform listed in the "PLATFORM" column.
# In summary, this query retrieves data from the "CLEAN WEEKLY_SALES" table, calculates the total transactions for each platform, and groups the results by platform. The output will show a list of platforms with their corresponding total transaction counts.

5. What is the percentage of sales for Retail vs Shopify for each month?



```sql
/* 5.   What is the percentage of sales for Retail vs Shopify for each month? */
WITH cte_monthly_platform_sales AS (
  SELECT
    month_number,
    calendar_year,
    platform,
    SUM(sales) AS monthly_sales
  FROM clean_weekly_sales
  GROUP BY month_number,calendar_year, platform
)
SELECT
  month_number,
  calendar_year,
  ROUND(100 * MAX(CASE WHEN platform = 'Retail' THEN monthly_sales ELSE NULL END) /SUM(monthly_sales),2) AS retail_percentage,
  ROUND(100 * MAX(CASE WHEN platform = 'Shopify' THEN monthly_sales ELSE NULL END) /SUM(monthly_sales),2) AS shopify_percentage
FROM cte_monthly_platform_sales
GROUP BY month_number,calendar_year
ORDER BY month_number,calendar_year;
```

month_number	calendar_year	retail_percentage	shopify_percentage
3	2018	97.92	2.08
3	2019	97.71	2.29
3	2020	97.30	2.70
4	2018	97.93	2.07
4	2019	97.80	2.20
4	2020	96.96	3.04

Table: weekly_sales

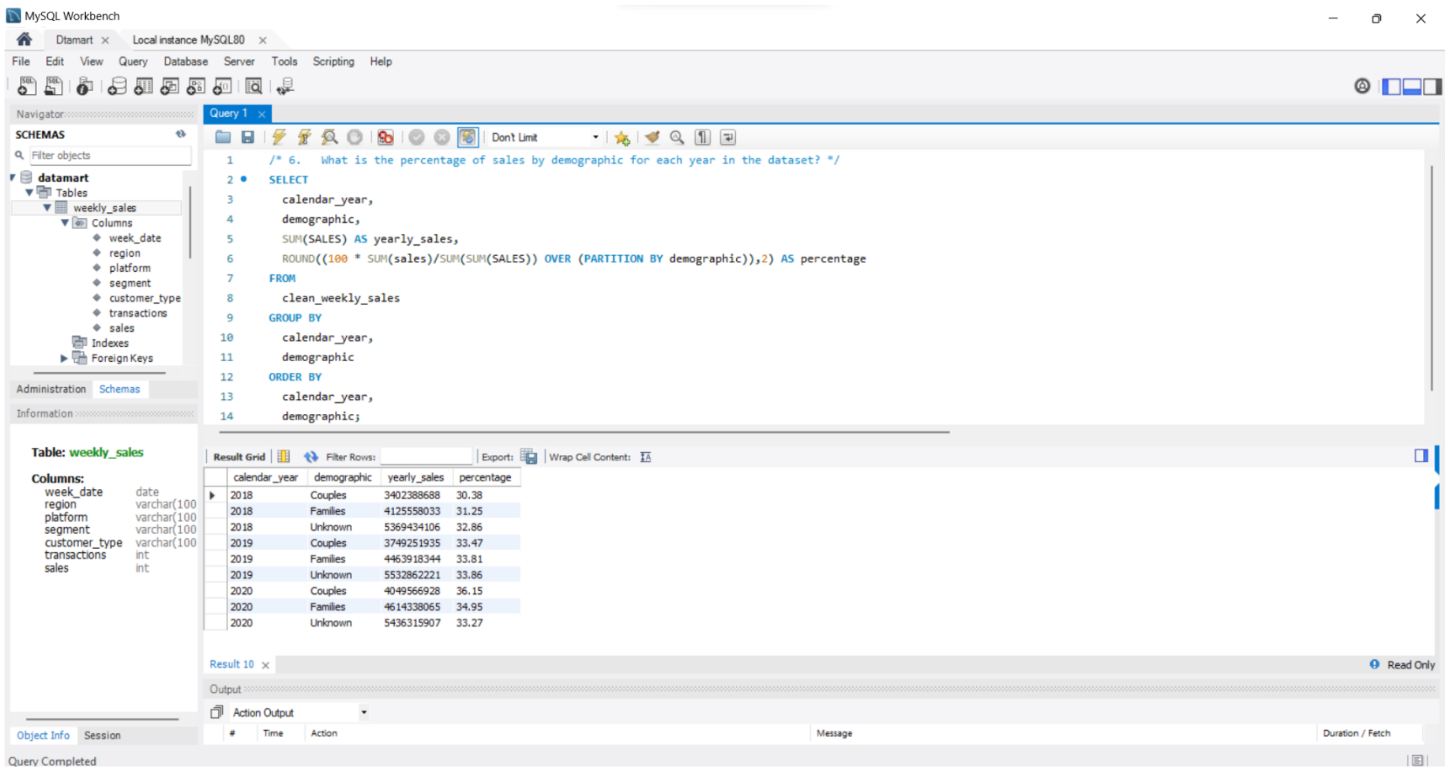
Columns:

week_date	date
region	varchar(100
platform	varchar(100
segment	varchar(100
customer_type	varchar(100
transactions	int
sales	int

Query Completed

# This SQL query calculates the monthly sales percentages for two different platforms, "Retail" and "Shopify," based on data from a table named "clean_weekly_sales." The query utilizes a Common Table Expression (CTE) named "cte_monthly_platform_sales" to structure and process the data as follows:
# The CTE selects data from the "clean_weekly_sales" table, grouping it by month, calendar year, and platform.
# It calculates the total monthly sales for each combination of month, calendar year, and platform and renames this value as "monthly sales."
# After processing the data in the CTE, the main query performs the following actions:
# It selects the month number, calendar year, and two additional columns:
# "retail_percentage," which represents the percentage of sales for the "Retail" platform out of the total monthly sales for each month and year.
# "shopify_percentage," which represents the percentage of sales for the "Shopify" platform out of the total monthly sales for each month and year.
# The percentages are rounded to two decimal places.
# The result set is then grouped by month number and calendar year.
# Finally, the query orders the results by month number and calendar year.
# In summary, this query is used to analyze the monthly sales distribution between the "Retail" and "Shopify" platforms for each month and year. It provides the percentage of sales for each platform relative to the total sales for that specific month and year, helping to track and compare platform performance over time.

6. What is the percentage of sales by demographic for each year in the dataset?



```
/* 6.  What is the percentage of sales by demographic for each year in the dataset? */
SELECT
  calendar_year,
  demographic,
  SUM(SALES) AS yearly_sales,
  ROUND((100 * SUM(sales)/SUM(SUM(SALES)) OVER (PARTITION BY demographic)),2) AS percentage
FROM
  clean_weekly_sales
GROUP BY
  calendar_year,
  demographic
ORDER BY
  calendar_year,
  demographic;
```

calendar_year	demographic	yearly_sales	percentage
2018	Couples	3402388688	30.38
2018	Families	4125558033	31.25
2018	Unknown	5369434106	32.86
2019	Couples	3749251935	33.47
2019	Families	4463918344	33.81
2019	Unknown	5532862221	33.86
2020	Couples	4049566928	36.15
2020	Families	4614338065	34.95
2020	Unknown	5436315907	33.27

Table: weekly_sales

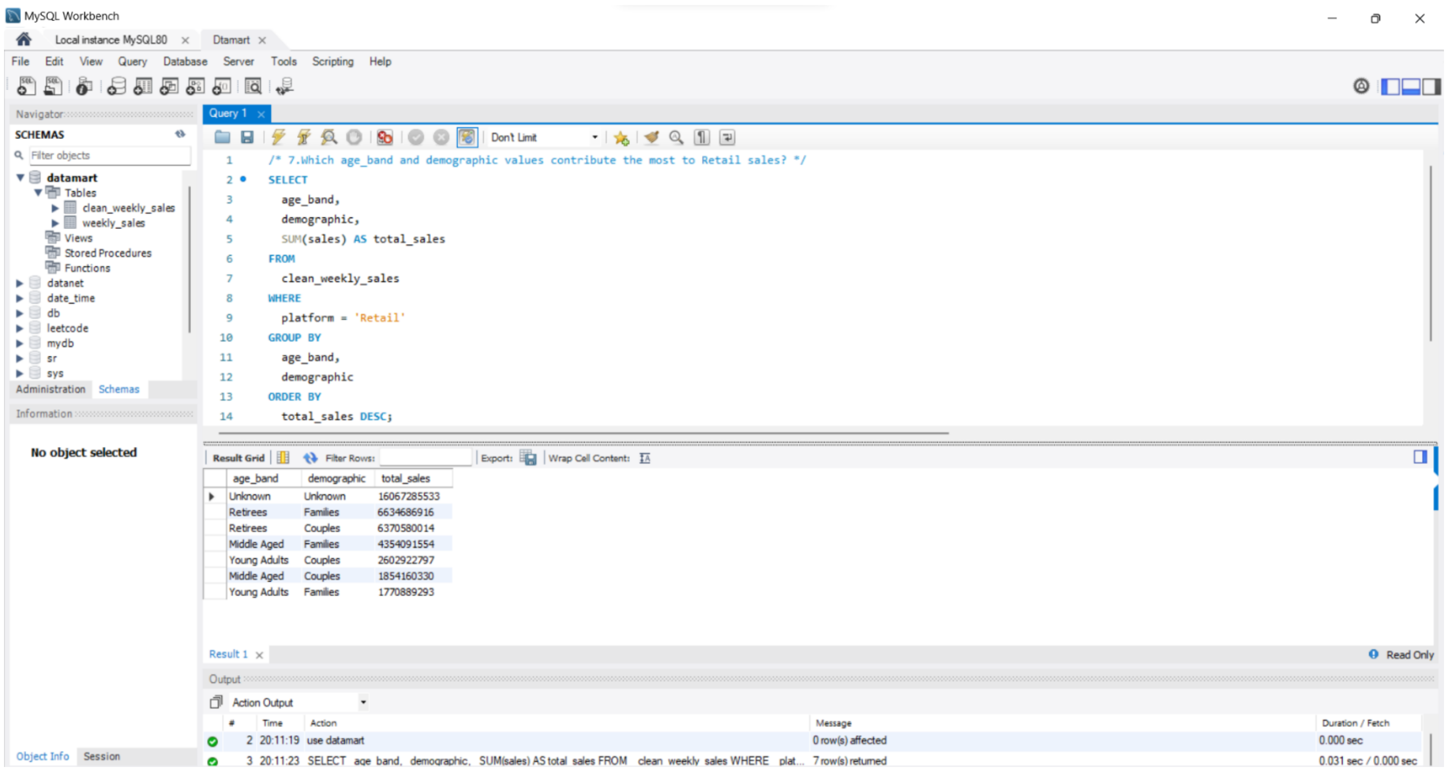
Columns:

week_date	date
region	varchar(100
platform	varchar(100
segment	varchar(100
customer_type	varchar(100
transactions	int
sales	int

Query Completed

# This SQL query is designed to analyze sales data from the "clean_weekly_sales" table. It calculates the yearly sales and percentage of sales for each demographic group, grouped by calendar year. Here's a breakdown of what the query does:
# It selects four columns for the result set:
# "calendar year": The year of the sales data.
# "demographic": The demographic group for which sales data is being analyzed.
# "yearly sales": The total sales for a specific year and demographic group.
# "percentage": The percentage of the total sales for the given demographic group in a specific year.
# The query uses the SUM() function to calculate the total sales for each combination of calendar year and demographic.
# It also uses the ROUND() function to round the calculated percentage to two decimal places.
# The query then uses the GROUP BY clause to group the results by "calendar year" and "demographic," which means it will show the total sales and percentage for each unique combination of year and demographic.
# Finally, the results are ordered by "calendar year" and "demographic" in ascending order, so the output is organized by year and demographic group.
# In summary, this query provides a breakdown of yearly sales and the percentage of total sales for different demographic groups, helping to analyze sales performance over time and across various demographics.

7. Which age_band and demographic values contribute the most to Retail sales?



```sql
/* 7.Which age_band and demographic values contribute the most to Retail sales? */
SELECT
    age_band,
    demographic,
    SUM(sales) AS total_sales
FROM
    clean_weekly_sales
WHERE
    platform = 'Retail'
GROUP BY
    age_band,
    demographic
ORDER BY
    total_sales DESC;
```

age_band	demographic	total_sales
Unknown	Unknown	16067285533
Retirees	Families	6634686916
Retirees	Couples	6370580014
Middle Aged	Families	4354091554
Young Adults	Couples	2602922797
Middle Aged	Couples	1854160330
Young Adults	Families	1770889293

```
#  Time      Action                                                                        Message              Duration / Fetch
2  20:11:19  use datamart                                                                  0 row(s) affected    0.000 sec
3  20:11:23  SELECT  age band,  demographic,  SUM(sales) AS total  sales FROM  clean  weekly  sales WHERE  plat...  7 row(s) returned    0.031 sec / 0.000 sec
```

- # This SQL query retrieves data from a table called "clean_weekly sales."
- # It calculates the total sales summed up as "total_sales" for different age bands and demographics, specifically within the "Retail" platform.
- # The results are grouped by "age_band" and "demographic," and then sorted in descending order of total sales.
- # In essence, it provides a breakdown of sales data by age band and demographic, focusing on the "Retail" platform.

THANK YOU