DWBI Project Report

The Organization sellls thousands of products in 100's of retail stores to its customers from last 10 years.
The Company also runs a loyalty program based on customers sales.
They wants to generate meaningful insights from this data.

What will you learn??

1. Generate test data using Python
2. Extract data using Pythin
3. Create Objects in Snowflake
4. Load data into Snowflake
5. Query data using SQL in Snowflake
6. Build Reports in Power BI

Tools used :

1. Python
2. Notepad++
3. Excel
4. Snowflake Account
5. Power BI Desktop

Python Code to Generate Random Files

To generate Store Data:

```
#import python libraries
import pandas as pd
import random
import csv
from faker import Faker

#Initialize
fake= Faker()

#input the number of rows that the csv file should have
num_rows = int(input( " Enter the number of rows that you want to generate in the csv file : "))


#input the name of the csv file (e.g data.csv)
csv_file = input ( " Enter the name of the csv file : ")

# details of the excel file that has the lookup data , File Path and Name , Sheet Name and column names where the data is present
excel_file_path_name = "D:/dwbi project/Lookup Data/LookupFile.xlsx"
excel_sheet_name = "Store Name Data"
adjective_column_name  = "Adjectives"
noun_column_name = "Nouns"


#fetch this sheet data in a dataframe
df = pd.read_excel(excel_file_path_name,sheet_name=excel_sheet_name)

#open the csv file
with open(csv_file,mode='w',newline='') as file:
```

```
    writer=csv.writer(file)

    #create the header
    header=['StoreName','StoreType','StoreOpeningDate','Address','City','State','Country','Region','Manager Name']

    #write the header to the csv file
    writer.writerow(header)

    #loop and generate multiple rows
    for _ in range(num_rows):

        #Select a random Adjective and Noun and we are going to concatenate it with the word "The" and finally use that as our
store name
        random_adjective=df[adjective_column_name].sample(n=1).values[0]
        random_noun=df[noun_column_name].sample(n=1).values[0]
        store_name= f"The {random_adjective} {random_noun}"
        # print(store_name)

        #Generate a Single row
        row = [
        store_name,
        random.choice(['Exclusive','MBO','SMB','Outlet Stores']),
        fake.date(),
        fake.address().replace("\n"," ").replace(","," "),
        fake.city(),
        fake.state(),
        fake.country(),
        random.choice(['North','South','East','West']),
        fake.first_name()
        ]

        # write the row to the csv file
        writer.writerow(row)

#print success statement
print(" The process completed Successfully")
```

To Generate Product Data:

```
#import python libraries
import pandas as pd
import random
import csv


#input the number of rows that the csv file should have
num_rows = int(input( " Enter the number of rows that you want to generate in the csv file : "))


#input the name of the csv file (e.g data.csv)
csv_file = input ( " Enter the name of the csv file : ")

# details of the excel file that has the lookup data , File Path and Name , Sheet Name and column names where the data is
present
excel_file_path_name = "D:/dwbi project/Lookup Data/LookupFile.xlsx"

excel_sheet_name_product = "Raw Product Names"
product_column_name  = "Product Name"
```

```
excel_sheet_name_category = "Product Categories"
category_column_name = "Category Name"


#fetch this sheet data in a dataframe
df = pd.read_excel(excel_file_path_name,sheet_name=excel_sheet_name_product)
df_cat = pd.read_excel(excel_file_path_name,sheet_name=excel_sheet_name_category)


#open the csv file
with open(csv_file,mode='w',newline='') as file:
    writer=csv.writer(file)


    #create the header
    header=['ProductName','Category','Brand','UnitPrice']


    #write the header to the csv file
    writer.writerow(header)


    #loop and generate multiple rows
    for _ in range(num_rows):

        #Generate a Single row
        row = [
        df[product_column_name].sample(n=1).values[0],#product Name
        df_cat[category_column_name].sample(n=1).values[0],#Category
        random.choice(['FakeLuxeAura','FakeUrbanGlow','FakeEtherealEdge','FakeVelvetVista','FakeZenithStyle']),
        random.randint(100,1000)
        ]

        # write the row to the csv file
        writer.writerow(row)


#print success statement
print(" The process completed Successfully")
```

To Generate Customer Data:

```
#import python libraries
import csv
import random
from faker import Faker


#Initialize

fake=Faker()

#input the number of rows that the csv file should have

num_rows=int(input(" Enter the number of rows the csv file should have : "))


#input the name of the csv file (e.g data.csv)
```

```
csv_file = input ( " Enter the name of the csv file like data.csv : ")


#open the csv file
with open(csv_file,mode='w',newline='') as file:
   writer=csv.writer(file)

#create the header
   header = ['First Name','Last Name','Gender','DateOfBirth', 'Email', 'Phone Number', 'Address', 'City', 'State', 'Postal Code',
'Country','LoyaltyProgramID']

#write the header to the csv file
   writer.writerow(header)

#loop and generate multiple rows
   for _ in range(num_rows):
      #Generate a Single row
      row=[
         fake.first_name(),
         fake.last_name(),
         random.choice(['M','F','Others','Not Specified']),
         fake.date(),
         fake.email(),
         fake.phone_number(),
         fake.address().replace("\n"," ").replace(","," "),
         fake.city(),
         fake.state(),
         fake.postcode(),
         fake.country(),
         random.randint(1,5)
      ]

      writer.writerow(row)

#print success statement

print(" The file has been loaded successfully")
```

To Generate Date Data:

```
# import pandas library
import pandas as pd

# start date and end date between which we need to geenrate our dates
start_date ='2014-01-01'
end_date='2024-12-31'

# generate a series of dates between the start and the end date
date_range=pd.date_range(start=start_date, end = end_date)
#print(date_range)

# convert these series of dates into a data frame
date_dimension = pd.DataFrame(date_range,columns = ['Date'])
#print(date_dimension)
```

```
# add new columns to our dataframe DayofWeek Month Quarter Year Isweekend DateID
date_dimension['DayofWeek'] = date_dimension['Date'].dt.dayofweek
date_dimension['Month'] = date_dimension['Date'].dt.month
date_dimension['Quarter'] = date_dimension['Date'].dt.quarter
date_dimension['Year'] = date_dimension['Date'].dt.year
date_dimension['Isweekend'] = date_dimension['DayofWeek'].isin([5,6])
date_dimension['DateID'] = date_dimension['Date'].dt.strftime('%Y%m%d').astype(int)

# reorder our data frame so that the dateid becomes the 1st column
cols = ['DateID'] + [col for col in date_dimension.columns if col != 'DateID']
date_dimension=date_dimension[cols]

# export it into a csv index column to be ignored
date_dimension.to_csv('DimDate.csv',index=False)

#print success statement
print(" The process completed Successfully")
```

To Generate Orders Data :

```
import pandas as pd
import numpy as np


num_rows = int(input(" Enter the number of rows for orders : "))

#generate the series of dates between 2014 and 2024
random_dates = np.random.choice(np.arange(np.datetime64('2014-01-01'),np.datetime64('2024-07-28')),size = num_rows)


formatted_rows = pd.to_datetime(random_dates).strftime('%Y%m%d')

data = {
   'DateID': formatted_rows,
   'ProductID': np.random.randint(1,1001,size=num_rows),
   'StoreID': np.random.randint(1,101,size=num_rows),
   'CustomerID': np.random.randint(1,1001,size=num_rows),
   'QuantityOrderded': np.random.randint(1,21,size=num_rows),
   'OrderAmount': np.random.randint(100,1001,size=num_rows)
   }

df = pd.DataFrame(data)

discount_perc = np.random.uniform(0.02,0.15,size=num_rows)
shipping_cost = np.random.uniform(0.05,0.15,size=num_rows)


# calculate columns
df['DiscountAmount'] = df['OrderAmount'] * discount_perc
df['Shipping Cost'] = df['OrderAmount'] * shipping_cost
df['TotalAmount'] = df['OrderAmount'] -(df['DiscountAmount']+df['Shipping Cost'])
print(df)

df.to_csv('factorders.csv',index=False)
```

To Generate Scheduled data:

```
import pandas as pd
```

```
import numpy as np
import os

DATEID='20240728'
directory="D:/dwbi project/Landing Directory/"

for i in range(1,101):
    num_rows = np.random.randint(100,1000)
    data = {
        'DateID': [DATEID] * num_rows ,
        'ProductID': np.random.randint(1,1001,size=num_rows),
        'StoreID': [i] * num_rows,
        'CustomerID': np.random.randint(1,1001,size=num_rows),
        'QuantityOrderded': np.random.randint(1,21,size=num_rows),
        'OrderAmount': np.random.randint(100,1001,size=num_rows)
        }

    df = pd.DataFrame(data)

    discount_perc = np.random.uniform(0.02,0.15,size=num_rows)
    shipping_cost = np.random.uniform(0.05,0.15,size=num_rows)


    # calculate columns

    df['DiscountAmount'] = df['OrderAmount'] * discount_perc
    df['Shipping Cost'] = df['OrderAmount'] * shipping_cost
    df['TotalAmount'] = df['OrderAmount'] -(df['DiscountAmount']+df['Shipping Cost'])
    print(df)

    file_name=f'Store_{i}_{DATEID}.csv'
    file_path=os.path.join(directory,file_name)

    # if file exists remove and write again
    if os.path.exists(file_path):
        os.remove(file_path)

    df.to_csv(file_path,index=False)


""" These are the column we need to generate
'DateID' -- random date in a date range in a specified format
'ProductID' -- random
'StoreID' -- random
'CustomerID' -- random
'QuantityOrderded' -- random
'OrderAmount' -- random
'DiscountAmount' -- Calculated dependent on OrderAmount
'Shipping Cost' -- Calculated dependent on OrderAmount
'TotalAmount' -- Calculated dependent on OrderAmount and others
"""
```

Snowflake Page Link : 24-queries - Snowflake

→ Objects Creation in Snowflake

```
-- Create database
```

```
create database test_db;

-- Create schema

create schema test_db_schema

-- Dimension Table: DimDate

CREATE TABLE DimDate (

    DateID INT PRIMARY KEY,

    Date DATE,

    DayOfWeek VARCHAR(10),

    Month VARCHAR(10),

    Quarter INT,

    Year INT,

    IsWeekend BOOLEAN

);

-- Dimension Table: DimCustomer

CREATE TABLE DimCustomer (

    CustomerID INT PRIMARY KEY autoincrement start 1 increment 1,

    FirstName VARCHAR(50),

    LastName VARCHAR(50),

    Gender VARCHAR(10),

    DateOfBirth DATE,

    Email VARCHAR(100),

    Address VARCHAR(255),

    City VARCHAR(50),

    State VARCHAR(50),

    ZipCode VARCHAR(10),

    Country VARCHAR(50),

    LoyaltyProgramID INT

);

-- Dimension Table: DimProduct

CREATE TABLE DimProduct (

    ProductID INT PRIMARY KEY autoincrement start 1 increment 1,
```

```
    ProductName VARCHAR(100),
    Category VARCHAR(50),
    Brand VARCHAR(50),
    UnitPrice DECIMAL(10, 2)
);
-- Dimension Table: DimStore
CREATE TABLE DimStore (
    StoreID INT PRIMARY KEY autoincrement start 1 increment 1,
    StoreName VARCHAR(100),
    StoreType VARCHAR(50),
    StoreOpeningDate DATE,
    Address VARCHAR(255),
    City VARCHAR(50),
    State VARCHAR(50),
    ZipCode VARCHAR(10),
    Country VARCHAR(50),
    ManagerName VARCHAR(100)
);
-- Dimension Table: DimLoyaltyProgram
CREATE TABLE DimLoyaltyProgram (
    LoyaltyProgramID INT PRIMARY KEY,
    ProgramName VARCHAR(100),
    ProgramTier VARCHAR(50),
    PointsAccrued INT
);
-- Fact Table: FactOrders
CREATE TABLE FactOrders (
    OrderID INT PRIMARY KEY autoincrement start 1 increment 1,
    DateID INT,
    ProductID INT,
    StoreID INT,
```

```
    CustomerID INT,
    QuantityOrdered INT,
    OrderAmount DECIMAL(10, 2),
    DiscountAmount DECIMAL(10, 2),
    ShippingCost DECIMAL(10, 2),
    TotalAmount DECIMAL(10, 2),
    FOREIGN KEY (DateID) REFERENCES DimDate(DateID),
    FOREIGN KEY (CustomerID) REFERENCES DimCustomer(CustomerID),
    FOREIGN KEY (ProductID) REFERENCES DimProduct(ProductID),
    FOREIGN KEY (StoreID) REFERENCES DimStore(StoreID)
);
```

➔ **To load the data in the internal stage, follow the below steps:**

```
CREATE OR REPLACE FILE FORMAT CSV_SOURCE_FILE_FORMAT
TYPE = 'CSV'
SKIP_HEADER = 1
FIELD_OPTIONALLY_ENCLOSED_BY = ' '' '
DATE_FORMAT = 'YYYY-MM-DD';

CREATE OR REPLACE STAGE TESTSTAGE;

-- Run for local using SnowSQL
PUT local_file_path Stage @TEST_DB.TEST_DB_SCHEMA.TESTSTAGE
```

➔ **To load the data from internal stage to warehouse: (COPY command)**

```
COPY INTODimLoyaltyProgram
FROM @TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimLoyaltyInfo/DimLoyaltyInfo.csv
FILE_FORMAT = (FORMAT_NAME = 'CSV_SOURCE_FILE_FORMAT');

SELECT * FROM DimLoyaltyProgram;

COPY INTO dIMcUSTOMER(FirstName,LastNAme, Gender, DateOfBirth, Emai, PhoneNumber, Address, City,
State, ZipCode, Country, LoyaltyProgramID)
FROM @TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimCustomerData/DimCustomerData.csv
FILE_FORMAT = (FORMAT_NAME = 'CSV_SOURCE_FILE_FORMAT');

SELECT * FROM DimCustomerData;

COPY INTO DimProduct(ProductName, Category, Brand, UnitPrice)
FROM @TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimProductData/DimProductData.csv
FILE_FORMAT = (FORMAT_NAME = 'CSV_SOURCE_FILE_FORMAT');

SELECT * FROM DimProductData;
```

```
COPY INTO DimDate(DateID, Date, DayOfWeek, Month, Quarter,Year, IsWeekend)
FROM @TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimDate/DimDate.csv
FILE_FORMAT = (FORMAT_NAME = 'CSV_SOURCE_FILE_FORMAT');

SELECT * FROM DimDate;

COPY INTO DimStore(StoreName, StoreType, StoreOpeningDate, Address, City, State, Country, Region,
ManagerName)
FROM @TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimStore/DimStore.csv
FILE_FORMAT = (FORMAT_NAME = 'CSV_SOURCE_FILE_FORMAT');

SELECT * FROM DimStore limit 100;

COPY INTO FactOrders(DateID, ProductID, StoreID, CustomerID, QuantityOrdered, OrderAmount,
DiscountAmount, ShippingCost, TotalAmount)
FROM @TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/Factorders /Factorders.csv
FILE_FORMAT = (FORMAT_NAME = 'CSV_SOURCE_FILE_FORMAT');

SELECT * FROM FactOrders limit 100;
```

➔ **Create a new user**

```
CREATE OR REPLACE USER Raj
   PASSWORD = 'Test_PowerBI_User'
   LOGIN_NAME = 'PowerBI User'
   DEFAULT_ROLE = 'ACCOUNTADMIN'
   DEFAULT_WAREHOUSE = 'COMPUTE_WH'
   MUST_CHANGE_PASSWORD = TRUE;
```

➔ **Grant it accountadmin access**

```
grant role accountadmin to user Raj
```

Query 1: **Update the Store Table So that all stores have opening date on or after 1-Jan-2014, Populate random dates.**

```
SELECT * FROM DIMSTORE;

SELECT DATEIFF(DAY, '2014-01-01',CURRENT_DATE)  → 3863

UPDATE DIMSTORE SET STOREOPENINGDATE = DATEADD(DAY,UNIFORM(0,3800,RANDOM()),'2014-01-01')

COMMIT;
```

Query 2:

Update the store table so that stores with store id between 91 and 100 are opened in the last 12 months.

```
SELECT * FROM DIMSTORE WHERE STOREID BETWEEN 91 AND 100;

UPDATE DIMSTORESET STOREOPENINGDATE = SELECT DATEADD(DAY, UNIFORM(0,360, RANDOM()), '2023-07-30') → WHERESTOREID BETWEEN
91 AND 100;

SELECT DATEADD(year, -1, CURRENT_DATE) → '2023-07-30'
```

```
SELECT DATEADD(DAY, UNIFORM(0,360, RANDOM()), '2023-07-30')

COMMIT;
```

Query 3: **Update the Customer Table so that all customers are at least 12 years old- Any customer that is less than 12 years old. Subtract 12 years from their DOB.**

```
SELECT * FROM DIMCUSTOMER WHERE dateofbirth >= dateadd(year, -12,current_date);

UPDATE DIMCUSTOMER SET dateofbirth = dateadd(year, -12,dateofbirth) where dateofbirth >= dateadd(year, -12,current_date);

COMMIT;

SELECT dateadd(year, -12,current_date);
```

Query 4: **We may have some orders in the Fact Table that may have a DAteID which contains a value even before the store was opened.**

For example: a store was opened last year but we have an order from 10 years ago which is incorrect.

Update dateid in order table for such rows with to have random dateid after the opening date of their respective stores.

```
Step 1: we identify the records that have a problem
Step 2: Identify a valid date that we can enter
Step 3: we need to convert the date into dateid and update
Update FACTORDERS f
Set f.dateid = r.dateid from (select orderid, d.dateid from
(
 SELECT orderid, Dateadd(day,
  (DATEDIFF(DAY, S.STOREOPENINGDATE,CURRENT_DATE)* UNIFORM(1,10,RANDOM())*.1, S.STOREOPENINGDATE) AS new_Date
FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMSTORE S ON F.STOREID = S.STOREID
WHERE D.DATE < S.STOREOPENINGDATE) o
Join dimdate d on o.new_Date= d.date) r
Where  f.orderid  = r.orderid
COMMIT;
```

Query 5: **List customers who haven't placed an order in the last 30 days**

```
Select * from dimcustomer where customerid not in (
Select distinct c.Customerid from dimcustomer c
Join factorders f on c.customerid = f.customerid
Join dimdate d on f.dateid = d.dateid
Where d.date >= dateadd(month, -1,current_date));
```

Query 6: **List the store that was opened most recently along with its sales since then**

```
Step1: fetch the most recent store among the stores
Step2: fetch the sales since then

With store_rank as
(
SELECT storied, storeopeningdate, row_number() over(order by storeopeningdate desc) as final_Rank FROM DIMSTORE
),
most_recent_store as
(
```

```
select storied from store_Rank where final_rank=1
),
Store_amount as
(
select o.storeid, sum(totalamount) from factorders o join most_recent_store s on o.storeid = s.storeid
group by o.storeid
)
Select  s.*, a.totalamount from dimstore s join store_amount a on s.storeid = a.storeid
```

Query 7: Find customers who have ordered product from more than 3 categories in the last 6 months

```
WITH BASE_DATA AS
(
SELECT O.CUSTOMERID, P.CATEGORY FROM FACTORDERS O JOIN DIMDATE D ON O.DATEID = D.DATEID
JOIN DIMPRODUCT P ON O.PRODUCTID
WHERE D.DATE >= DATEADD(MONTH, -6, CURRENT_DATE)
GROUP BY O.CUSTOMERID,P. CATEGORY
)
SELECT CUSTOMERID
FROM BASE_DATA
GROUP BY CUSTOMERID
HAVING COUNT(DISTINCT CATEGORY) > 3
```

Query 8: Get the monthly sales for the current year

```
SELECT MONTH,SUM(TOTALAMOUNT) AS MONTHLY_AMOUNT FROM FAACTORDERS O JOIN DIMDATE D ON O.DATEID = D.DATEID WHERE
D.YEAR = EXTRACT(YEAR FROM CURRENT_DATE)
GROUP BY MONTH
ORDER BY MONTH
```

Query 9: Find the highest discount given on any order in the last 1 year

```
WITH base_data as
(
SELECT discountAMOUNT, ROWNUMBER() OVER(ORDER BY discountAMOUNT desc) as discountAMOUNT_rank FROM FACTORDERS O JOIN
DIMDATE D ON O.DATEID = D.DATEID
WHERE D.DATE >= DATEADD(YEAR, -1,CURRENT_DATE)
)
SELECT * FROM base_data WHERE discountAMOUNT_rank =1
```

Query 10: Calculate total sales by multiplying the unit price from product column with quantity ordered from fact orders

```
SELECT SUM(quantityordered*unitprice) from FACTORDERS O JOIN DIMPRODUCT P ON O.PRODUCTID =  P.PRODUCTID
```

Query 11: Show the customer id of the customer who has taken the maximum discount in their lifetime

```
SELECT CUSTOMERID, SUM(DISCOUNTAMOUNT) AS TOTAL_DISCOUNTFROM FACTORDERS F
GROUP BY CUSTOMERID
ORDER BY TOTAL_DISCOUNT DESC LIMIT 1
```

Query 12: List the customer who was placed maximum number of orders till date

```
WITH base_data AS
(
SELECT CUSTOMERID, COUNT(ORDERID) AS ORDER_COUNT FROM FACTORDERS F
GROUP BY CUSTOMERID
```

```
),
ORDER_RANK_DATA AS
(
SELECT b.*, ROW_NUMBER() OVER(ORDER BY order_count DESC) AS order_rank FROM base_data b
)
SELECT customerid, order_count FROM order_rank where order_rank=1
```

Query 13: Show the top 3 brands on there sales in the last 1 year

```
WITH brand_sales
AS (
SELECT brand, sum(totalamount) FROM
FACTORDERS F JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMPRODUCT P ON F.PRODUCTID = P.PRODUCTID
WHERE D.DATE >= DATEADD(YEAR, -1, CURRENT_DATE)
GROUP BY  brand
),
brand_sales_rank AS
(
SELECT S.*, ROW_NUMBER() OVER(ORDER BY total_sales DESC) AS sales_rank FROM brand_sales s
)  SELECT brand, total_sales FROM brand_sales_rank WHERE sales_rank<=3
```

Query 14: IF the discount amount and the shipping cost was made static at 5 and 8 % respectively will the sum of new total amount be greater than the total amount we have

```
SELECT CASE WHEN SUM(ORDERAMOUNT – ORDERAMOUNT*.05 -ORDERAMOUNT*.08) > SUM(TOTALAMOUNT) THEN ‘yes’ ELSE ‘no END
FROM FACTORDERS F → LIMIT 10
```

Query 15: Share the number of customers and their current loyalty program status.

```
SELECT L.PROGRAMTIER, COUNT(CUSTOMERID) AS CUSTOMER_COUNT FROM DIMCUSTOMER D  JOIN DIMLOYALTYPROGRAM L ON
D.LOYALTYPROGRAMID = L.LOYALTYPROGRAMID
GROUP BY L.PROGRAMTIER
```

Query 16: Show the region category wise total amount for the last 6 months.

```
SELECT REGION,CATEGORY, SUM(TOTALAMOUNT) AS TOTAL_SALES
FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMPRODUCT P ON F.PRODUCTID = P.PRODUCTID
JOIN DIMSTORE S ON F.STOREID = S.STOREID
WHERE D.STORE S ON F.STOREID = S.STOREID
WHERE D.DATE >= DATEADD(MONTH, -6, CURRENT_DATE)
GROUP BY REGION, CATEGORY
```

Query 17: Show the top 5 products based on quantity ordered in the last 3 years

```
WITH QUANTITY_DATA AS
(
SELECT F.PRODUCTID, SUM(QUANTITYORDERED) AS TOTAL_QUANTITY FROM FACTORDERS F JOIN DIMDATE D ON F.DATEID = D.DATEID
WHERE D.DATE >=DATEADD(YEAR,-3, CURRNET_DATE)
GROUP BY F.PRODUCTID
),
Quantity_rank_data  AS
(
SELECT q.*, ROW_NUMBER() OVER(ORDER BY TOTAL_QUANTITY DESC) AS quantity_wise_rank FROM QUANTITY_DATA q
)
```

```
SELECT productid,TOTAL_QUANTITY FROM Quantity_rank_data WHERE qantity_wise_rank <=5
```

Query 18: List the total amount for each loyalty program tier since year 2023

```
SELECT P.PROGRAMNAME, SUM(TOTALAMOUNT) AS TOTAL_SALES FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMCUSTOMER C ON F.CUSTOMERID = C.CUSTOMERID
JOIN DIMLOYALTYPROGRAM P ON C.LOYALTYPROGRAMID= P.LOYALTYPROGRAMID
WHERE D.YEAR >=2023
GROUP BY P.PROGRAMNAME
```

Query 19: Calculate the revenge generated by each store manager in june 2024

```
SELECT S.MANAGERNAME, SUM(TOTALAMOUNT) AS TOAL_SALES FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMSTORE S ON F.STOREID = S.STOREID
WHERE D.YEAR = 2024 AND D.MONTH = 6
GROUP BY S.MANAGERNAME
```

Query 20: List the average order amount per store, along with the store name and type for the year 2024

```
SELECT S.STORENAME,S.STORETYPE, AVG(TOTALAMOUNT) AS TOAL_SALES FROM FACTORDERS F
JOIN DIMDATE D ON F.DATEID = D.DATEID
JOIN DIMSTORE S ON F.STOREID = S.STOREID
WHERE D.YEAR = 2024
GROUP BY S.STORENAME, S.STORETYPE
```

Query 21: Query data from the customer csv file that is present in the stage → Reading from file

```
SELECT $1, $2, $3
FROM
@TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimCustomerData/DimCustomerData.csv
(FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT');
```

Query 22: Aggregate data, share the count of record in the DimCustomer File from stage → Aggregation in file

```
SELECT count($1)
FROM
@TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimCustomerData/DimCustomerData.csv
(FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT');
```

Query 23: Filter data, share the records from DimCustomer file Where customerid is greater than 960 → Filter from files

```
SELECT $1, $2, $3, $4, $5, $6, $7, $8
FROM @TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimCustomerData/DimCustomerData.csv
(FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT')
WHERE $4 > '2000-01-01';
```

Query 24: JoinDimCustomer and DimLoyalty and show the customer 1st name along with the program tier they are part of → Join data from files

```
WITH customer_data AS
(
SELECT $1 AS First_Name, $12 AS Loyalty_Program_ID
FROM
@TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimCustomerData/DimCustomerData.csv
(FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT'),
```

```
Loyalty_data AS
(
SELECT $1 AS Loyalty_Program_ID, $3 AS program_tier
FROM
@TEST_DB.TEST_DB_SCHEMA.TESTSTAGE/DimCustomerData/DimCustomerData.csv
(FILE_FORMAT => 'CSV_SOURCE_FILE_FORMAT')
)
SELECT program_tier, count(1) AS total_Count FROM customer_data c join loyalty_data l on c.Loyalty_Program_id = l.Loyalty_Program_ID
GROUP BY program_tier
```