DE ASSIGNMENT(Pyspark)

# **query:1**

from pyspark.sql import SparkSession

from pyspark.sql.functions import sum, avg

from pyspark.sql.window import Window

from pyspark.sql import functions as F

import snowflake.connector

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

# Create a SparkSession

from pyspark.sql import SparkSession

from pyspark.sql.functions import avg

spark = SparkSession.builder \

.appName("Customer Total Return Analysis") \

.getOrCreate()

# Read data from Snowflake into Spark DataFrame

store\_returns\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_returns").load()

date\_dim\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "date\_dim").load()

store\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store").load()

customer\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer").load()

# Perform necessary transformations

joined\_data = store\_returns\_df.join(date\_dim\_df, store\_returns\_df["SR\_RETURNED\_DATE\_SK"] == date\_dim\_df["D\_DATE\_SK"]) \

.filter(date\_dim\_df["D\_YEAR"] == 2001) \

.select("SR\_CUSTOMER\_SK", "SR\_STORE\_SK", "SR\_RETURN\_AMT")

customer\_total\_return = joined\_data.groupBy("SR\_CUSTOMER\_SK", "SR\_STORE\_SK") \

.sum("SR\_RETURN\_AMT") \

.withColumnRenamed("SR\_CUSTOMER\_SK", "CTR\_CUSTOMER\_SK") \

.withColumnRenamed("SR\_STORE\_SK", "CTR\_STORE\_SK") \

.withColumnRenamed("sum(SR\_RETURN\_AMT)", "CTR\_TOTAL\_RETURN")

avg\_return\_per\_store = customer\_total\_return.groupBy("CTR\_STORE\_SK") \

.agg((avg("CTR\_TOTAL\_RETURN") \* 1.2).alias("AVG\_TOTAL\_RETURN"))

# Filter customers based on the condition

filtered\_customers = customer\_total\_return.join(store\_df, store\_df["S\_STORE\_SK"] == customer\_total\_return["CTR\_STORE\_SK"]) \

.join(customer\_df, customer\_df["C\_CUSTOMER\_SK"] == customer\_total\_return["CTR\_CUSTOMER\_SK"]) \

.join(avg\_return\_per\_store, avg\_return\_per\_store["CTR\_STORE\_SK"] == customer\_total\_return["CTR\_STORE\_SK"]) \

.filter(customer\_total\_return["CTR\_TOTAL\_RETURN"] > avg\_return\_per\_store["AVG\_TOTAL\_RETURN"]) \

.filter(store\_df["S\_STATE"] == "TN") \

.select("C\_CUSTOMER\_ID") \

.orderBy("C\_CUSTOMER\_ID") \

.limit(100)

try:

    filtered\_customers.show(n=100, truncate=False) # Adjust 'n' to the desired number of rows

except Exception as e:

    print("An error occurred while showing the result:", e)

# query:2

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, round, sum, when

# Create a SparkSession

spark = SparkSession.builder \

    .appName("Weekly Sales Increase Analysis") \

    .getOrCreate()

# Read web\_sales and catalog\_sales into DataFrame

web\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "web\_sales").load()

catalog\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "catalog\_sales").load()

date\_dim\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "date\_dim").load()

# Combine web\_sales and catalog\_sales into a single DataFrame with common columns

wscs\_df = web\_sales\_df.select("WS\_SOLD\_DATE\_SK", "WS\_EXT\_SALES\_PRICE") \

            .unionAll(catalog\_sales\_df.select("CS\_SOLD\_DATE\_SK", "CS\_EXT\_SALES\_PRICE")) \

            .withColumnRenamed("WS\_SOLD\_DATE\_SK", "sold\_date\_sk") \

            .withColumnRenamed("WS\_EXT\_SALES\_PRICE", "sales\_price")

# Join with date\_dim DataFrame to get day name and week sequence

wswscs\_df = wscs\_df.join(date\_dim\_df, wscs\_df["sold\_date\_sk"] == date\_dim\_df["D\_DATE\_SK"]) \

            .groupBy("D\_WEEK\_SEQ") \

            .agg(

                round(sum(when(col("D\_DAY\_NAME") == "Sunday", col("sales\_price")).otherwise(0)), 2).alias("sun\_sales"),

                round(sum(when(col("D\_DAY\_NAME") == "Monday", col("sales\_price")).otherwise(0)), 2).alias("mon\_sales"),

                round(sum(when(col("D\_DAY\_NAME") == "Tuesday", col("sales\_price")).otherwise(0)), 2).alias("tue\_sales"),

                round(sum(when(col("D\_DAY\_NAME") == "Wednesday", col("sales\_price")).otherwise(0)), 2).alias("wed\_sales"),

                round(sum(when(col("D\_DAY\_NAME") == "Thursday", col("sales\_price")).otherwise(0)), 2).alias("thu\_sales"),

                round(sum(when(col("D\_DAY\_NAME") == "Friday", col("sales\_price")).otherwise(0)), 2).alias("fri\_sales"),

                round(sum(when(col("D\_DAY\_NAME") == "Saturday", col("sales\_price")).otherwise(0)), 2).alias("sat\_sales")

            )

# Join with itself for comparison between years

comparison\_df = wswscs\_df.alias("y").join(wswscs\_df.alias("z"), col("y.D\_WEEK\_SEQ") == col("z.D\_WEEK\_SEQ") - 53) \

                .select("y.D\_WEEK\_SEQ",

                        round(col("y.sun\_sales") / col("z.sun\_sales"), 2).alias("sun\_sales\_increase"),

                        round(col("y.mon\_sales") / col("z.mon\_sales"), 2).alias("mon\_sales\_increase"),

                        round(col("y.tue\_sales") / col("z.tue\_sales"), 2).alias("tue\_sales\_increase"),

                        round(col("y.wed\_sales") / col("z.wed\_sales"), 2).alias("wed\_sales\_increase"),

                        round(col("y.thu\_sales") / col("z.thu\_sales"), 2).alias("thu\_sales\_increase"),

                        round(col("y.fri\_sales") / col("z.fri\_sales"), 2).alias("fri\_sales\_increase"),

                        round(col("y.sat\_sales") / col("z.sat\_sales"), 2).alias("sat\_sales\_increase")

                ) \

                .orderBy("y.D\_WEEK\_SEQ")

# Show the results

comparison\_df.show()

**# query: 3**

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

from pyspark.sql import SparkSession

from pyspark.sql.functions import sum as spark\_sum

# Create SparkSession

spark = SparkSession.builder \

    .appName("Convert SQL to PySpark") \

    .getOrCreate()

# Read tables as DataFrames

date\_dim\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "date\_dim").load()

store\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_sales").load()

item\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "item").load()

# Join DataFrames and apply filters

joined\_df = date\_dim\_df.join(store\_sales\_df, date\_dim\_df["d\_date\_sk"] == store\_sales\_df["ss\_sold\_date\_sk"]) \

    .join(item\_df, store\_sales\_df["ss\_item\_sk"] == item\_df["i\_item\_sk"]) \

    .where((date\_dim\_df["d\_moy"] == 11) & (item\_df["i\_manufact\_id"] == 128))

# Perform aggregation

agg\_df = joined\_df.groupBy(date\_dim\_df["d\_year"], item\_df["i\_brand\_id"], item\_df["i\_brand"]) \

    .agg(spark\_sum("ss\_ext\_sales\_price").alias("sum\_agg"))

# Order by specified columns

ordered\_df = agg\_df.orderBy("d\_year", "sum\_agg", "i\_brand\_id", ascending=[True, False, True])

# Show DataFrame

ordered\_df.show()

**#query: 4**

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

from pyspark.sql import SparkSession

from pyspark.sql.functions import sum, col, when

# Create a SparkSession

spark = SparkSession.builder \

    .appName("Customer Spending Analysis") \

    .getOrCreate()

# Read data from Snowflake into Spark DataFrame

customer\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer").load()

store\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_sales").load()

catalog\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "catalog\_sales").load()

# Define the qualification substitution parameters

YEAR = 2001

# Calculate year total for store sales

from pyspark.sql.functions import sum, col, lit

# Calculate year total for store sales

store\_sales\_year\_total = store\_sales\_df.groupBy("ss\_customer\_sk").agg(

    sum(((store\_sales\_df["ss\_ext\_list\_price"] - store\_sales\_df["ss\_ext\_wholesale\_cost"] - store\_sales\_df["ss\_ext\_discount\_amt"]) + store\_sales\_df["ss\_ext\_sales\_price"]) / 2).alias("year\_total")

).withColumn("sale\_type", lit("s"))

# Calculate year total for catalog sales

catalog\_sales\_year\_total = catalog\_sales\_df.groupBy("cs\_bill\_customer\_sk").agg(

    sum((((catalog\_sales\_df["cs\_ext\_list\_price"] - catalog\_sales\_df["cs\_ext\_wholesale\_cost"] - catalog\_sales\_df["cs\_ext\_discount\_amt"]) + catalog\_sales\_df["cs\_ext\_sales\_price"]) / 2)).alias("year\_total")

).withColumn("sale\_type", lit("c"))

# Calculate year total for catalog sales

catalog\_sales\_year\_total = catalog\_sales\_df.groupBy("cs\_bill\_customer\_sk").agg(

    sum((((catalog\_sales\_df["cs\_ext\_list\_price"] - catalog\_sales\_df["cs\_ext\_wholesale\_cost"] - catalog\_sales\_df["cs\_ext\_discount\_amt"]) + catalog\_sales\_df["cs\_ext\_sales\_price"]) / 2)).alias("year\_total")

).withColumn("sale\_type", lit("c"))

# Find customers who spent more via catalog than in stores

preferred\_customers = store\_sales\_year\_total.join(catalog\_sales\_year\_total, store\_sales\_year\_total["ss\_customer\_sk"] == catalog\_sales\_year\_total["cs\_bill\_customer\_sk"], "inner") \

    .join(customer\_df, store\_sales\_year\_total["ss\_customer\_sk"] == customer\_df["c\_customer\_sk"], "inner") \

    .filter((store\_sales\_year\_total["year\_total"] > 0) & (catalog\_sales\_year\_total["year\_total"] > 0) & \

            (catalog\_sales\_year\_total["year\_total"] / store\_sales\_year\_total["year\_total"] > 1)) \

    .select(

        customer\_df["c\_customer\_id"].alias("customer\_id"),

        customer\_df["c\_first\_name"].alias("customer\_first\_name"),

        customer\_df["c\_last\_name"].alias("customer\_last\_name"),

        customer\_df["c\_birth\_country"].alias("customer\_birth\_country"),

        customer\_df["c\_preferred\_cust\_flag"].alias("customer\_preferred\_cust\_flag")

    )

# Show the result

preferred\_customers.show()

**# 5. query**

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

from pyspark.sql import SparkSession

from pyspark.sql.functions import sum as spark\_sum, col, when, lit

# Create SparkSession

spark = SparkSession.builder \

    .appName("Convert SQL to PySpark") \

    .getOrCreate()

# Read tables as DataFrames

store\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_sales").load()

store\_returns\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_returns").load()

catalog\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "catalog\_sales").load()

catalog\_returns\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "catalog\_returns").load()

web\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "web\_sales").load()

web\_returns\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "web\_returns").load()

store\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store").load()

catalog\_page\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "catalog\_page").load()

web\_site\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "web\_site").load()

date\_dim\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "date\_dim").load()

# Create temporary views for tables

store\_sales\_df.createOrReplaceTempView("store\_sales")

store\_returns\_df.createOrReplaceTempView("store\_returns")

catalog\_sales\_df.createOrReplaceTempView("catalog\_sales")

catalog\_returns\_df.createOrReplaceTempView("catalog\_returns")

web\_sales\_df.createOrReplaceTempView("web\_sales")

web\_returns\_df.createOrReplaceTempView("web\_returns")

store\_df.createOrReplaceTempView("store")

catalog\_page\_df.createOrReplaceTempView("catalog\_page")

web\_site\_df.createOrReplaceTempView("web\_site")

date\_dim\_df.createOrReplaceTempView("date\_dim")

# Execute the PySpark code

result\_df = store\_sales\_df.join(

    date\_dim\_df,

    store\_sales\_df["ss\_sold\_date\_sk"] == date\_dim\_df["d\_date\_sk"]

).join(

    store\_df,

    store\_sales\_df["ss\_store\_sk"] == store\_df["s\_store\_sk"]

).where(

    (col("d\_date").between("2000-08-23", date\_add("2000-08-23", 14))) &

    (col("store\_sales\_df") == "store")

).groupBy(

    "s\_store\_id"

).agg(

    spark\_sum("ss\_sales\_price").alias("sales"),

    spark\_sum("ss\_net\_profit").alias("profit"),

    spark\_sum(when(col("sr\_return\_amt").isNull(), 0).otherwise(col("sr\_return\_amt"))).alias("returns"),

    spark\_sum(when(col("sr\_net\_loss").isNull(), 0).otherwise(col("sr\_net\_loss"))).alias("profit\_loss")

).select(

    col("channel"),

    col("id"),

    spark\_sum("sales").alias("sales"),

    spark\_sum("returns").alias("returns"),

    (spark\_sum("profit") - spark\_sum("profit\_loss")).alias("profit")

).groupBy(

    rollup("channel", "id")

).orderBy(

    "channel",

    "id"

)

# Show DataFrame

result\_df.show()

**#6.query**

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, avg

# Create SparkSession

spark = SparkSession.builder \

    .appName("Convert SQL to PySpark") \

    .getOrCreate()

# Read necessary tables

customer\_address\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer\_address").load()

customer\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer").load()

store\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_sales").load()

date\_dim\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "date\_dim").load()

item\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "item").load()

# Filter date\_dim for the specified year and month

date\_dim\_filtered\_df = date\_dim\_df.filter((col("d\_year") == 2001) & (col("d\_moy") == 1))

# Join the tables and perform necessary operations

result\_df = customer\_address\_df.alias("a") \

    .join(customer\_df.alias("c"), col("a.ca\_address\_sk") == col("c.c\_current\_addr\_sk")) \

    .join(store\_sales\_df.alias("s"), col("c.c\_customer\_sk") == col("s.ss\_customer\_sk")) \

    .join(date\_dim\_filtered\_df.alias("d"), col("s.ss\_sold\_date\_sk") == col("d.d\_date\_sk")) \

    .join(item\_df.alias("i"), col("s.ss\_item\_sk") == col("i.i\_item\_sk")) \

    .groupBy("a.ca\_state") \

    .agg({"\*": "count"}) \

    .withColumnRenamed("count(1)", "cnt") \

    .filter(col("cnt") >= 10) \

    .orderBy("cnt", "a.ca\_state")

# Show the result

result\_df.show()

**#query: 7**

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

from pyspark.sql import SparkSession

from pyspark.sql.functions import avg, col

# Create SparkSession

spark = SparkSession.builder \

    .appName("Convert SQL to PySpark") \

    .getOrCreate()

# Read necessary tables

store\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_sales").load()

customer\_demographics\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer\_demographics").load()

date\_dim\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "date\_dim").load()

item\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "item").load()

promotion\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "promotion").load()

# Filter date\_dim for the specified year

date\_dim\_filtered\_df = date\_dim\_df.filter(col("d\_year") == 2000)

# Join the tables and perform necessary operations

result\_df = store\_sales\_df.alias("ss") \

    .join(customer\_demographics\_df.alias("cd"), col("ss.ss\_cdemo\_sk") == col("cd.cd\_demo\_sk")) \

    .join(date\_dim\_filtered\_df.alias("d"), col("ss.ss\_sold\_date\_sk") == col("d.d\_date\_sk")) \

    .join(item\_df.alias("i"), col("ss.ss\_item\_sk") == col("i.i\_item\_sk")) \

    .join(promotion\_df.alias("p"), col("ss.ss\_promo\_sk") == col("p.p\_promo\_sk")) \

    .filter((col("cd.cd\_gender") == "M") &

            (col("cd.cd\_marital\_status") == "S") &

            (col("cd.cd\_education\_status") == "College") &

            ((col("p.p\_channel\_email") == "N") | (col("p.p\_channel\_event") == "N"))) \

    .groupBy("i.i\_item\_id") \

    .agg(avg("ss.ss\_quantity").alias("agg1"),

         avg("ss.ss\_list\_price").alias("agg2"),

         avg("ss.ss\_coupon\_amt").alias("agg3"),

         avg("ss.ss\_sales\_price").alias("agg4")) \

    .orderBy("i.i\_item\_id")

# Show the result

result\_df.show()

**# query: 8**

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, substring, sum as \_sum

# Initialize Spark session

spark = SparkSession.builder \

.appName("Snowflake to Spark") \

.getOrCreate()

# Read data from Snowflake into Spark DataFrames

store\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_sales").load()

date\_dim\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "date\_dim").load()

store\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store").load()

customer\_address\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer\_address").load()

customer\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer").load()

# Subquery to filter the zip codes

valid\_zips\_df = customer\_address\_df.select(substring("CA\_ZIP", 1, 5).alias("CA\_ZIP")) \

.filter(col("CA\_ZIP").isin(

'67436', '26121', '38443', '63157', '68856', '19485', '86425', '26741',

'70991', '60899', '63573', '47556', '56193', '93314', '87827', '62017',

'85067', '95390', '48091', '10261', '81845', '41790', '42853', '24675',

'12840', '60065', '84430', '57451', '24021', '91735', '75335', '71935',

'34482', '56943', '70695', '52147', '56251', '28411', '86653', '23005',

'22478', '29031', '34398', '15365', '42460', '33337', '59433', '73943',

'72477', '74081', '74430', '64605', '39006', '11226', '49057', '97308',

'42663', '18187', '19768', '43454', '32147', '76637', '51975', '11181',

'45630', '33129', '45995', '64386', '55522', '26697', '20963', '35154',

'64587', '49752', '66386', '30586', '59286', '13177', '66646', '84195',

'74316', '36853', '32927', '12469', '11904', '36269', '17724', '55346',

'12595', '53988', '65439', '28015', '63268', '73590', '29216', '82575',

'69267', '13805', '91678', '79460', '94152', '14961', '15419', '48277',

'62588', '55493', '28360', '14152', '55225', '18007', '53705', '56573',

'80245', '71769', '57348', '36845', '13039', '17270', '22363', '83474',

'25294', '43269', '77666', '15488', '99146', '64441', '43338', '38736',

'62754', '48556', '86057', '23090', '38114', '66061', '18910', '84385',

'23600', '19975', '27883', '65719', '19933', '32085', '49731', '40473',

'27190', '46192', '23949', '44738', '12436', '64794', '68741', '15333',

'24282', '49085', '31844', '71156', '48441', '17100', '98207', '44982',

'20277', '71496', '96299', '37583', '22206', '89174', '30589', '61924',

'53079', '10976', '13104', '42794', '54772', '15809', '56434', '39975',

'13874', '30753', '77598', '78229', '59478', '12345', '55547', '57422',

'42600', '79444', '29074', '29752', '21676', '32096', '43044', '39383',

'37296', '36295', '63077', '16572', '31275', '18701', '40197', '48242',

'27219', '49865', '84175', '30446', '25165', '13807', '72142', '70499',

'70464', '71429', '18111', '70857', '29545', '36425', '52706', '36194',

'42963', '75068', '47921', '74763', '90990', '89456', '62073', '88397',

'73963', '75885', '62657', '12530', '81146', '57434', '25099', '41429',

'98441', '48713', '52552', '31667', '14072', '13903', '44709', '85429',

'58017', '38295', '44875', '73541', '30091', '12707', '23762', '62258',

'33247', '78722', '77431', '14510', '35656', '72428', '92082', '35267',

'43759', '24354', '90952', '11512', '21242', '22579', '56114', '32339',

'52282', '41791', '24484', '95020', '28408', '99710', '11899', '43344',

'72915', '27644', '62708', '74479', '17177', '32619', '12351', '91339',

'31169', '57081', '53522', '16712', '34419', '71779', '44187', '46206',

'96099', '61910', '53664', '12295', '31837', '33096', '10813', '63048',

'31732', '79118', '73084', '72783', '84952', '46965', '77956', '39815',

'32311', '75329', '48156', '30826', '49661', '13736', '92076', '74865',

'88149', '92397', '52777', '68453', '32012', '21222', '52721', '24626',

'18210', '42177', '91791', '75251', '82075', '44372', '45542', '20609',

'60115', '17362', '22750', '90434', '31852', '54071', '33762', '14705',

'40718', '56433', '30996', '40657', '49056', '23585', '66455', '41021',

'74736', '72151', '37007', '21729', '60177', '84558', '59027', '93855',

'60022', '86443', '19541', '86886', '30532', '39062', '48532', '34713',

'52077', '22564', '64638', '15273', '31677', '36138', '62367', '60261',

'80213', '42818', '25113', '72378', '69802', '69096', '55443', '28820',

'13848', '78258', '37490', '30556', '77380', '28447', '44550', '26791',

'70609', '82182', '33306', '43224', '22322', '86959', '68519', '14308',

'46501', '81131', '34056', '61991', '19896', '87804', '65774', '92564'

))

# Second subquery to filter zip codes with preferred customers

preferred\_cust\_zips\_df = customer\_df.filter(col("C\_CURRENT\_HH\_DEMOGRAPHIC") == "preferred") \

.join(customer\_address\_df, customer\_df.C\_CURRENT\_ADDR\_SK == customer\_address\_df.CA\_ADDRESS\_SK) \

.select(substring(customer\_address\_df.CA\_ZIP, 1, 5).alias("CA\_ZIP"))

# Perform the intersection

common\_zips\_df = valid\_zips\_df.select("CA\_ZIP").intersect(preferred\_cust\_zips\_df.select("CA\_ZIP"))

# Join the DataFrames to filter sales by these zip codes

result\_df = store\_sales\_df.join(common\_zips\_df, store\_sales\_df.SS\_ADDR\_SK == customer\_address\_df.CA\_ADDRESS\_SK) \

.join(date\_dim\_df, store\_sales\_df.SS\_SOLD\_DATE\_SK == date\_dim\_df.D\_DATE\_SK) \

.join(store\_df, store\_sales\_df.SS\_STORE\_SK == store\_df.S\_STORE\_SK) \

.groupBy(store\_df.S\_STORE\_NAME) \

.agg(\_sum(store\_sales\_df.SS\_SALES\_PRICE).alias("TOTAL\_SALES"))

result\_df.show()

**# query: 9**

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

from pyspark.sql.functions import col, avg, count, when

# Initialize Spark session

spark = SparkSession.builder \

.appName("SQL to PySpark Conversion") \

.getOrCreate()

# Read data from Snowflake into Spark DataFrames

store\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_sales").load()

reason\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "reason").load()

# Function to calculate the metric based on the condition

def calculate\_metric(df, lower, upper, threshold):

  count\_value = df.filter((col("ss\_quantity") >= lower) & (col("ss\_quantity") <= upper)).count()

  if count\_value > threshold:

    return df.filter((col("ss\_quantity") >= lower) & (col("ss\_quantity") <= upper)).agg(avg("ss\_ext\_list\_price")).collect()[0][0]

  else:

    return df.filter((col("ss\_quantity") >= lower) & (col("ss\_quantity") <= upper)).agg(avg("ss\_net\_profit")).collect()[0][0]

# Calculate each bucket value

bucket1 = calculate\_metric(store\_sales\_df, 1, 20, 3672)

bucket2 = calculate\_metric(store\_sales\_df, 21, 40, 3392)

bucket3 = calculate\_metric(store\_sales\_df, 41, 60, 32784)

bucket4 = calculate\_metric(store\_sales\_df, 61, 80, 26032)

bucket5 = calculate\_metric(store\_sales\_df, 81, 100, 23982)

# Create a DataFrame with the calculated bucket values

bucket\_values = [(bucket1, bucket2, bucket3, bucket4, bucket5)]

buckets\_df = spark.createDataFrame(bucket\_values, ["bucket1", "bucket2", "bucket3", "bucket4", "bucket5"])

# Filter the reason table and join with the buckets DataFrame

result\_df = reason\_df.filter(col("r\_reason\_sk") == 1).crossJoin(buckets\_df)

result\_df.show()

# query: 10

sfOptions = {

  "sfURL" : "https://uh98701.ap-southeast-1.snowflakecomputing.com",

  "sfAccount" : "uh98701.ap-southeast-1",

  "sfUser" : "UMESHCHANDRAKADEM",

  "sfPassword" : "Umesh.K@127",

  "sfDatabase" : "SNOWFLAKE\_SAMPLE\_DATA",

  "sfSchema" : "TPCDS\_SF10TCL",

  "sfWarehouse" : "COMPUTE\_WH"

}

from pyspark.sql.functions import col, count

# Initialize Spark session

spark = SparkSession.builder \

.appName("SQL to PySpark Conversion") \

.getOrCreate()

# Read data from Snowflake into Spark DataFrames

customer\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer").load()

customer\_address\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer\_address").load()

customer\_demographics\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "customer\_demographics").load()

store\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "store\_sales").load()

web\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "web\_sales").load()

catalog\_sales\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "catalog\_sales").load()

date\_dim\_df = spark.read.format("snowflake").options(\*\*sfOptions).option("dbtable", "date\_dim").load()

# Filter date\_dim for the required year and months

filtered\_date\_dim\_df = date\_dim\_df.filter((col("d\_year") == 2002) & (col("d\_moy").between(4, 7)))

# Filter store\_sales, web\_sales, and catalog\_sales for the required date range

filtered\_store\_sales\_df = store\_sales\_df.join(filtered\_date\_dim\_df, store\_sales\_df["ss\_sold\_date\_sk"] == filtered\_date\_dim\_df["d\_date\_sk"])

filtered\_web\_sales\_df = web\_sales\_df.join(filtered\_date\_dim\_df, web\_sales\_df["ws\_sold\_date\_sk"] == filtered\_date\_dim\_df["d\_date\_sk"])

filtered\_catalog\_sales\_df = catalog\_sales\_df.join(filtered\_date\_dim\_df, catalog\_sales\_df["cs\_sold\_date\_sk"] == filtered\_date\_dim\_df["d\_date\_sk"])

# Filter customers based on store\_sales, web\_sales, and catalog\_sales

store\_sales\_customers = filtered\_store\_sales\_df.select("ss\_customer\_sk").distinct()

web\_sales\_customers = filtered\_web\_sales\_df.select("ws\_bill\_customer\_sk").distinct()

catalog\_sales\_customers = filtered\_catalog\_sales\_df.select("cs\_ship\_customer\_sk").distinct()

# Union all customers

all\_customers = store\_sales\_customers.union(web\_sales\_customers).union(catalog\_sales\_customers).distinct()

# Filter customer and customer\_address tables

filtered\_customer\_address\_df = customer\_address\_df.filter(col("ca\_county").isin(

'Lycoming County', 'Sheridan County', 'Kandiyohi County', 'Pike County', 'Greene County'

))

# Join customer with filtered customer\_address

customer\_with\_address = customer\_df.join(filtered\_customer\_address\_df, customer\_df["c\_current\_addr\_sk"] == filtered\_customer\_address\_df["ca\_address\_sk"])

# Filter customers who are in the sales data

final\_customers = customer\_with\_address.join(all\_customers, customer\_with\_address["c\_customer\_sk"] == all\_customers["ss\_customer\_sk"])

# Join with customer\_demographics

final\_data = final\_customers.join(customer\_demographics\_df, customer\_demographics\_df["cd\_demo\_sk"] == final\_customers["c\_current\_cdemo\_sk"])

# Group by required columns and aggregate

result\_df = final\_data.groupBy(

"cd\_gender",

"cd\_marital\_status",

"cd\_education\_status",

"cd\_purchase\_estimate",

"cd\_credit\_rating",

"cd\_dep\_count",

"cd\_dep\_employed\_count",

"cd\_dep\_college\_count"

).agg(

count("\*").alias("cnt1"),

count("\*").alias("cnt2"),

count("\*").alias("cnt3"),

count("\*").alias("cnt4"),

count("\*").alias("cnt5"),

count("\*").alias("cnt6")

).orderBy(

"cd\_gender",

"cd\_marital\_status",

"cd\_education\_status",

"cd\_purchase\_estimate",

"cd\_credit\_rating",

"cd\_dep\_count",

"cd\_dep\_employed\_count",

"cd\_dep\_college\_count"

).limit(100)

# Show the result

result\_df.show()