Big Data Processing Lab-5

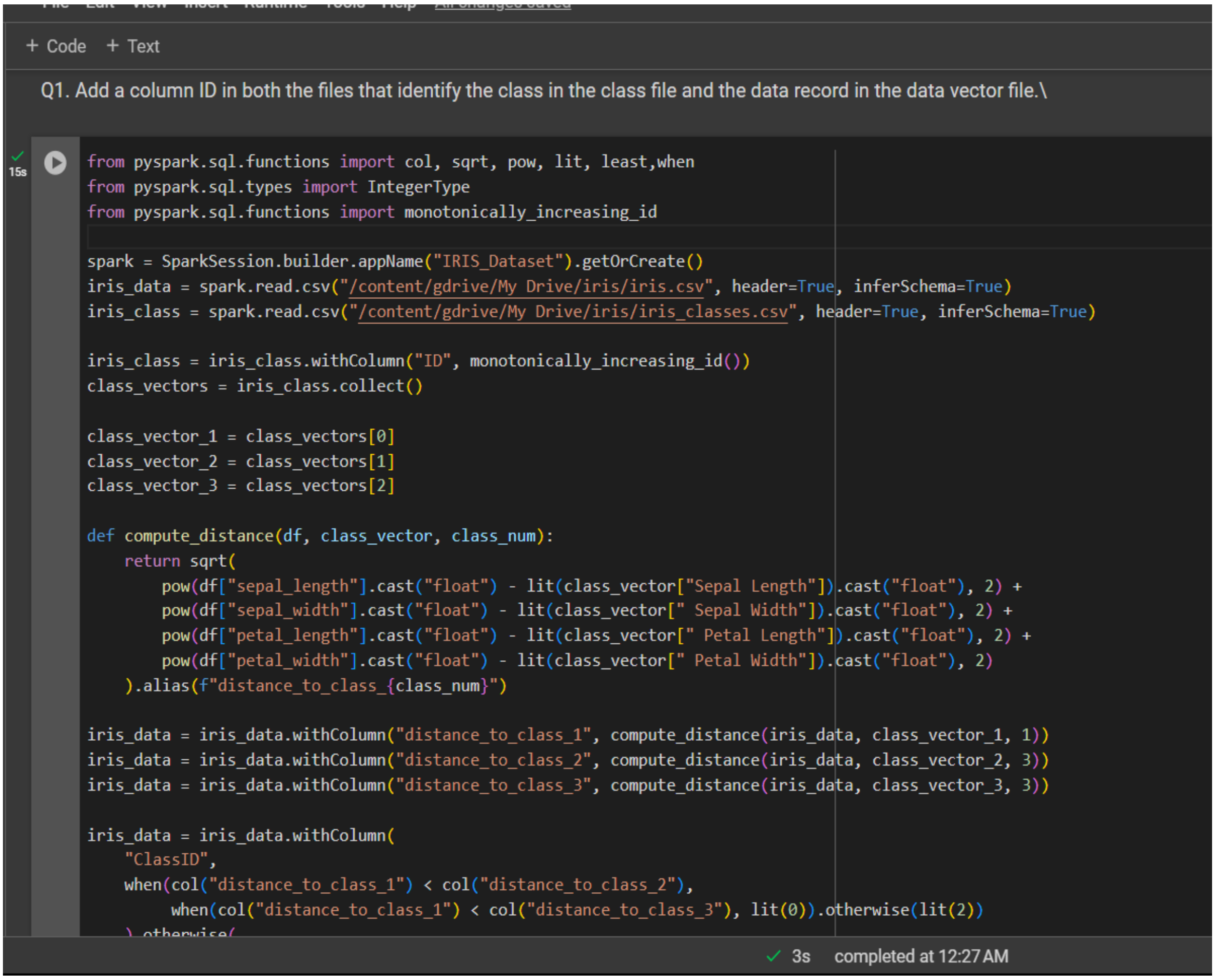
Name: Sumit Vishwakarma

Roll No: 202201320

Colab Link:https://colab.research.google.com/drive/1qoWgXo2rUcqI-ZVKYnXp_uRl_PazjuW8 - scrollTo=YKAMqYiUM7cK

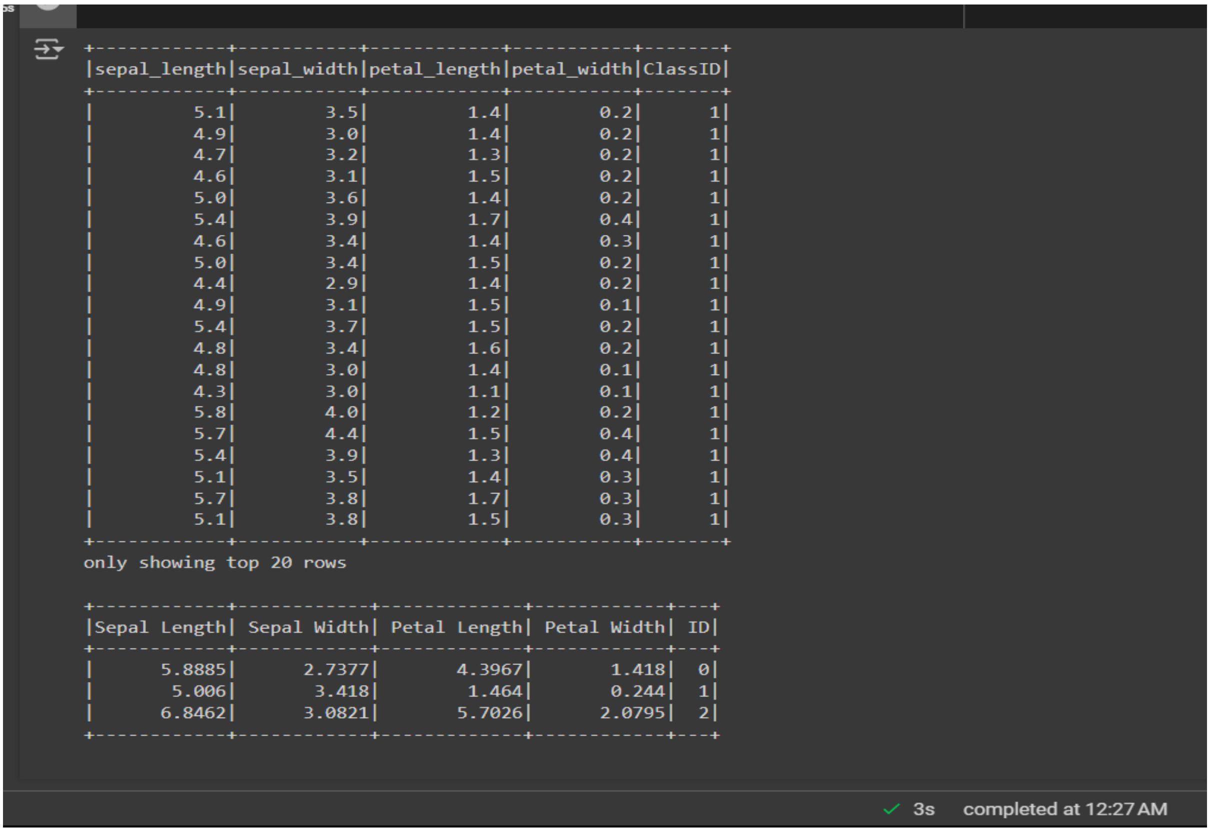
Q1: Add a column ID in both the files that identify the class in the class file and the data record in the data vector file.

Code and Output:



Q1. Add a column ID in both the files that identify the class in the class file and the data record in the data vector file.\

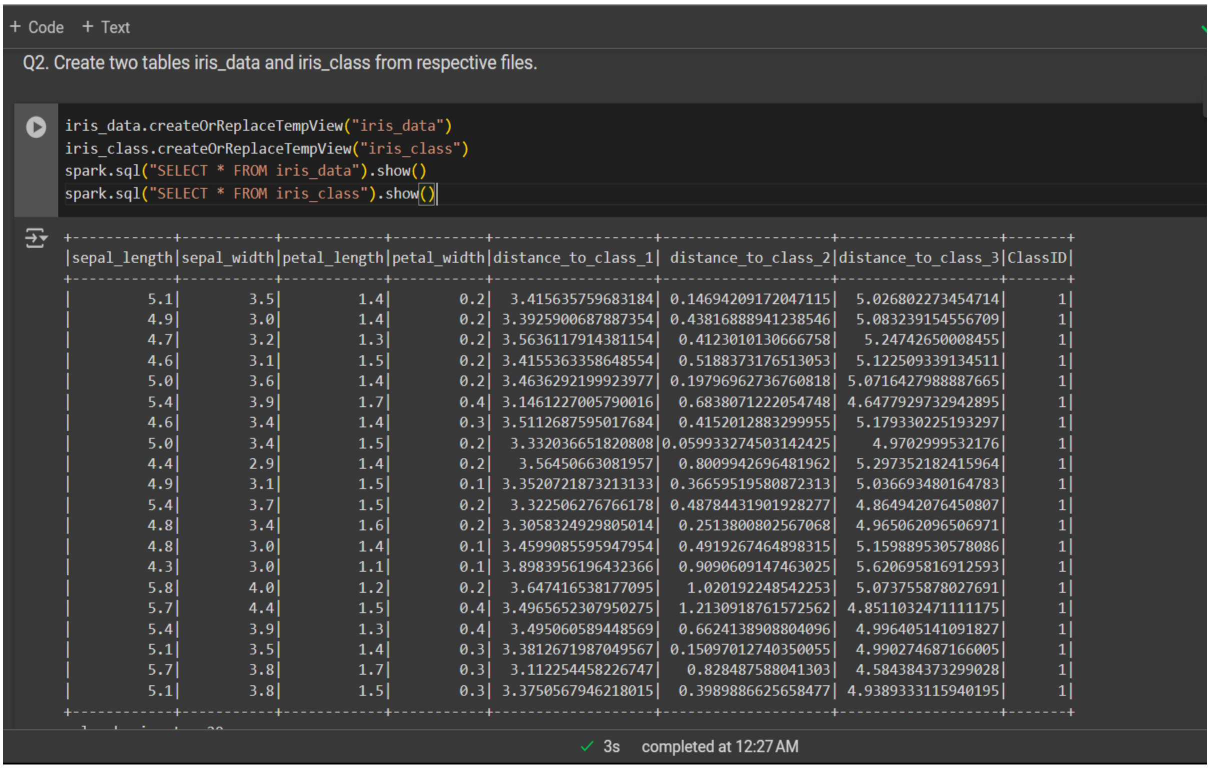
```
from pyspark.sql.functions import col, sqrt, pow, lit, least,when
from pyspark.sql.types import IntegerType
from pyspark.sql.functions import monotonically_increasing_id

spark = SparkSession.builder.appName("IRIS_Dataset").getOrCreate()
iris_data = spark.read.csv("/content/gdrive/My Drive/iris/iris.csv", header=True, inferSchema=True)
iris_class = spark.read.csv("/content/gdrive/My Drive/iris/iris_classes.csv", header=True, inferSchema=True)

iris_class = iris_class.withColumn("ID", monotonically_increasing_id())
class_vectors = iris_class.collect()

class_vector_1 = class_vectors[0]
class_vector_2 = class_vectors[1]
class_vector_3 = class_vectors[2]

def compute_distance(df, class_vector, class_num):
    return sqrt(
        pow(df["sepal_length"].cast("float") - lit(class_vector["Sepal Length"]).cast("float"), 2) +
        pow(df["sepal_width"].cast("float") - lit(class_vector[" Sepal Width"]).cast("float"), 2) +
        pow(df["petal_length"].cast("float") - lit(class_vector[" Petal Length"]).cast("float"), 2) +
        pow(df["petal_width"].cast("float") - lit(class_vector[" Petal Width"]).cast("float"), 2)
    ).alias(f"distance_to_class_{class_num}")

iris_data = iris_data.withColumn("distance_to_class_1", compute_distance(iris_data, class_vector_1, 1))
iris_data = iris_data.withColumn("distance_to_class_2", compute_distance(iris_data, class_vector_2, 3))
iris_data = iris_data.withColumn("distance_to_class_3", compute_distance(iris_data, class_vector_3, 3))

iris_data = iris_data.withColumn(
    "ClassID",
    when(col("distance_to_class_1") < col("distance_to_class_2"),
         when(col("distance_to_class_1") < col("distance_to_class_3"), lit(0)).otherwise(lit(2))
    ).otherwise(
```



```
+------------+-----------+------------+-----------+-------+
|sepal_length|sepal_width|petal_length|petal_width|ClassID|
+------------+-----------+------------+-----------+-------+
|         5.1|        3.5|         1.4|        0.2|      1|
|         4.9|        3.0|         1.4|        0.2|      1|
|         4.7|        3.2|         1.3|        0.2|      1|
|         4.6|        3.1|         1.5|        0.2|      1|
|         5.0|        3.6|         1.4|        0.2|      1|
|         5.4|        3.9|         1.7|        0.4|      1|
|         4.6|        3.4|         1.4|        0.3|      1|
|         5.0|        3.4|         1.5|        0.2|      1|
|         4.4|        2.9|         1.4|        0.2|      1|
|         4.9|        3.1|         1.5|        0.1|      1|
|         5.4|        3.7|         1.5|        0.2|      1|
|         4.8|        3.4|         1.6|        0.2|      1|
|         4.8|        3.0|         1.4|        0.1|      1|
|         4.3|        3.0|         1.1|        0.1|      1|
|         5.8|        4.0|         1.2|        0.2|      1|
|         5.7|        4.4|         1.5|        0.4|      1|
|         5.4|        3.9|         1.3|        0.4|      1|
|         5.1|        3.5|         1.4|        0.3|      1|
|         5.7|        3.8|         1.7|        0.3|      1|
|         5.1|        3.8|         1.5|        0.3|      1|
+------------+-----------+------------+-----------+-------+
only showing top 20 rows

+------------+------------+-------------+------------+---+
|Sepal Length| Sepal Width| Petal Length| Petal Width| ID|
+------------+------------+-------------+------------+---+
|      5.8885|      2.7377|       4.3967|       1.418|  0|
|       5.006|       3.418|        1.464|       0.244|  1|
|      6.8462|      3.0821|       5.7026|      2.0795|  2|
+------------+------------+-------------+------------+---+
```

Q2: Create two tables iris_data and iris_class from respective files.

Code and Output:

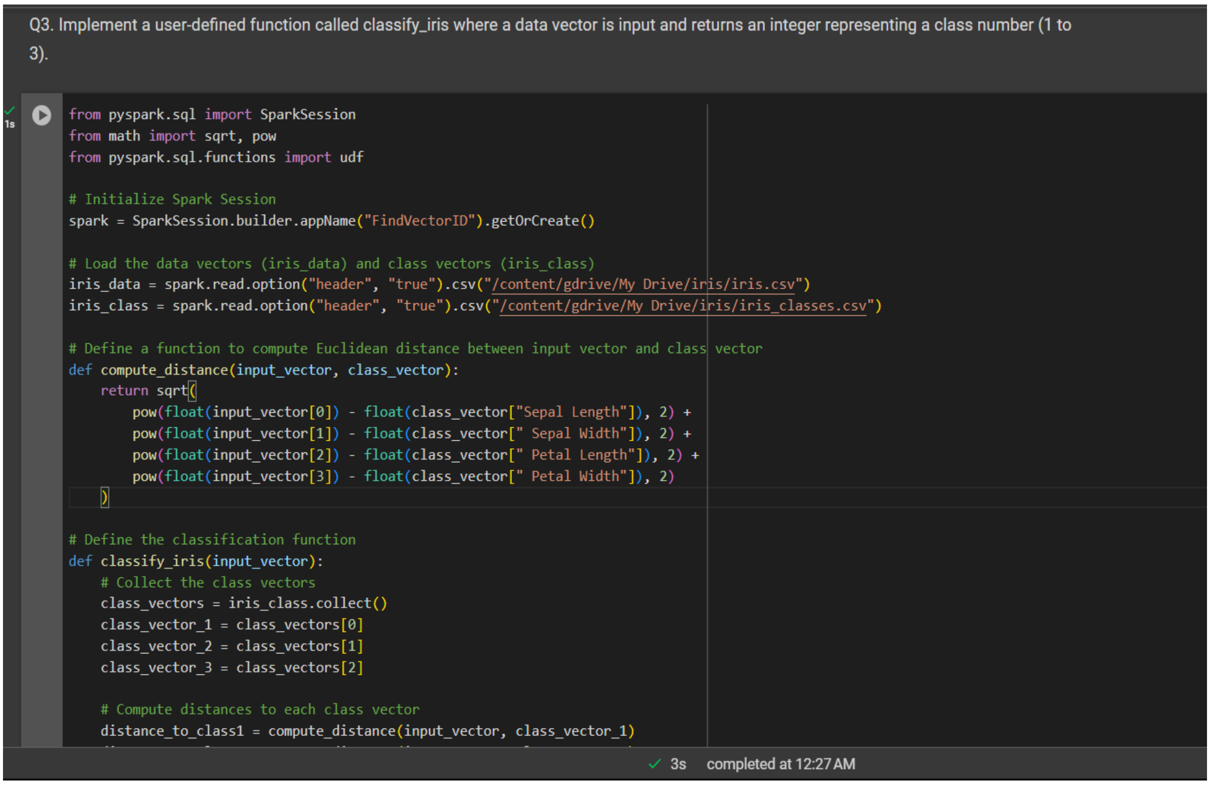


Q2. Create two tables iris_data and iris_class from respective files.

```
iris_data.createOrReplaceTempView("iris_data")
iris_class.createOrReplaceTempView("iris_class")
spark.sql("SELECT * FROM iris_data").show()
spark.sql("SELECT * FROM iris_class").show()
```

```
+------------+-----------+------------+-----------+-------------------+--------------------+-------------------+-------+
|sepal_length|sepal_width|petal_length|petal_width|distance_to_class_1| distance_to_class_2|distance_to_class_3|ClassID|
+------------+-----------+------------+-----------+-------------------+--------------------+-------------------+-------+
|         5.1|        3.5|         1.4|        0.2|  3.415635759683184| 0.14694209172047115|  5.026802273454714|      1|
|         4.9|        3.0|         1.4|        0.2| 3.3925900687887354| 0.43816888941238546|  5.083239154556709|      1|
|         4.7|        3.2|         1.3|        0.2| 3.5636117914381154|   0.412301013066758|   5.24742650008455|      1|
|         4.6|        3.1|         1.5|        0.2| 3.4155363358648554|  0.5188373176513053|  5.122509339134511|      1|
|         5.0|        3.6|         1.4|        0.2| 3.4636292199923977| 0.19796962736760818| 5.0716427988887665|      1|
|         5.4|        3.9|         1.7|        0.4| 3.1461227005790016|  0.6838071222054748| 4.6477929732942895|      1|
|         4.6|        3.4|         1.4|        0.3| 3.5112687595017684|  0.4152012883299955|  5.179330225193297|      1|
|         5.0|        3.4|         1.5|        0.2|  3.332036651820808|0.059933274503142425|    4.9702999532176|      1|
|         4.4|        2.9|         1.4|        0.2|   3.56450663081957|  0.8009942696481962|  5.297352182415964|      1|
|         4.9|        3.1|         1.5|        0.1| 3.3520721873213133| 0.36659519580872313|  5.036693480164783|      1|
|         5.4|        3.7|         1.5|        0.2|  3.322506276766178| 0.48784431901928277|  4.864942076450807|      1|
|         4.8|        3.4|         1.6|        0.2| 3.3058324929805014|  0.2513800802567068|  4.965062096506971|      1|
|         4.8|        3.0|         1.4|        0.1| 3.4599085595947954|  0.4919267464898315|  5.159889530578086|      1|
|         4.3|        3.0|         1.1|        0.1| 3.8983956196432366|  0.9090609147463025|  5.620695816912593|      1|
|         5.8|        4.0|         1.2|        0.2|  3.647416538177095|   1.020192248542253|  5.073755878027691|      1|
|         5.7|        4.4|         1.5|        0.4| 3.4965652307950275|  1.2130918761572562| 4.8511032471111175|      1|
|         5.4|        3.9|         1.3|        0.4|  3.495060589448569|  0.6624138908804096|  4.996405141091827|      1|
|         5.1|        3.5|         1.4|        0.3| 3.3812671987049567| 0.15097012740350055|  4.990274687166005|      1|
|         5.7|        3.8|         1.7|        0.3|  3.112254458226747|   0.828487588041303|  4.584384373299028|      1|
|         5.1|        3.8|         1.5|        0.3| 3.3750567946218015|  0.3989886625658477| 4.9389333115940195|      1|
+------------+-----------+------------+-----------+-------------------+--------------------+-------------------+-------+
```

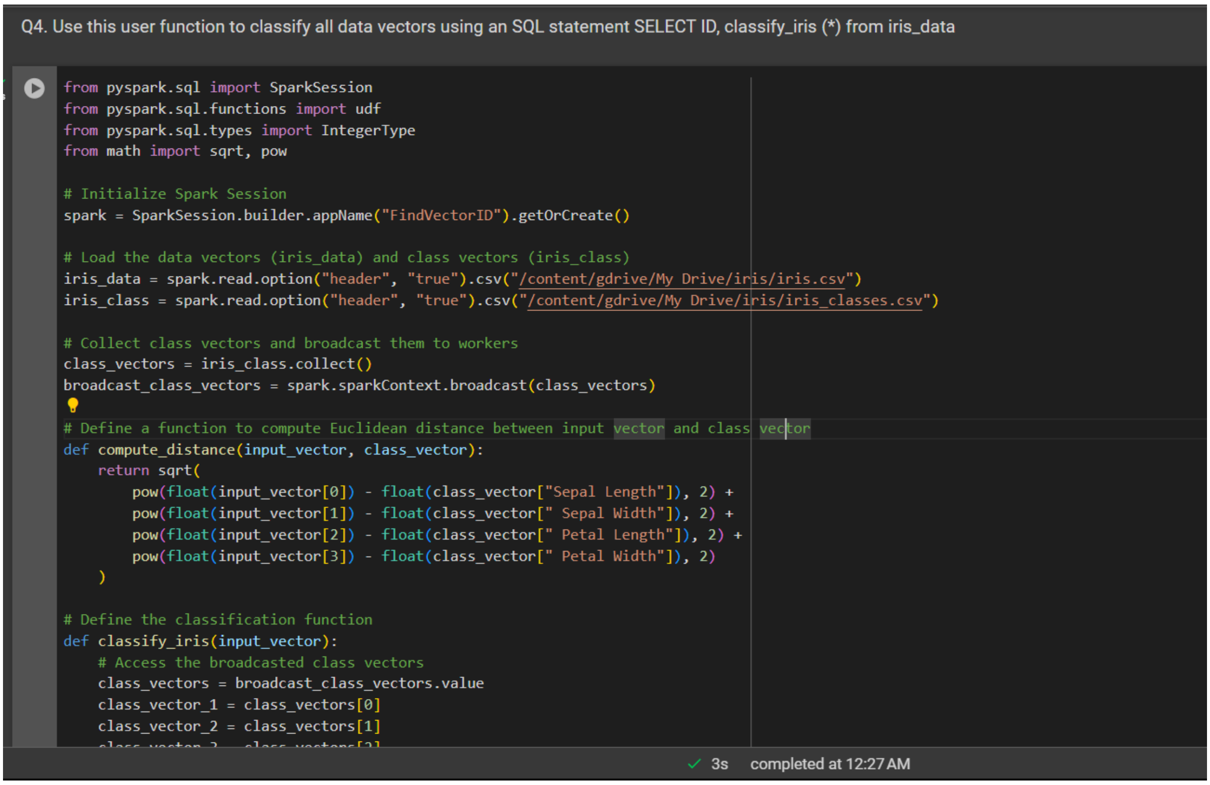
Q3: Implement a user-defined function called classify_iris where a data vector is input and returns an integer representing a class number (1 to 3).



Q3. Implement a user-defined function called classify_iris where a data vector is input and returns an integer representing a class number (1 to 3).

```
from pyspark.sql import SparkSession
from math import sqrt, pow
from pyspark.sql.functions import udf

# Initialize Spark Session
spark = SparkSession.builder.appName("FindVectorID").getOrCreate()

# Load the data vectors (iris_data) and class vectors (iris_class)
iris_data = spark.read.option("header", "true").csv("/content/gdrive/My Drive/iris/iris.csv")
iris_class = spark.read.option("header", "true").csv("/content/gdrive/My Drive/iris/iris_classes.csv")

# Define a function to compute Euclidean distance between input vector and class vector
def compute_distance(input_vector, class_vector):
    return sqrt(
        pow(float(input_vector[0]) - float(class_vector["Sepal Length"]), 2) +
        pow(float(input_vector[1]) - float(class_vector[" Sepal Width"]), 2) +
        pow(float(input_vector[2]) - float(class_vector[" Petal Length"]), 2) +
        pow(float(input_vector[3]) - float(class_vector[" Petal Width"]), 2)
    )

# Define the classification function
def classify_iris(input_vector):
    # Collect the class vectors
    class_vectors = iris_class.collect()
    class_vector_1 = class_vectors[0]
    class_vector_2 = class_vectors[1]
    class_vector_3 = class_vectors[2]

    # Compute distances to each class vector
    distance_to_class1 = compute_distance(input_vector, class_vector_1)
```

Q4: Use this user function to classify all data vectors using an SQL statement SELECT ID, classify_iris (*) from iris_data



Q4. Use this user function to classify all data vectors using an SQL statement SELECT ID, classify_iris (*) from iris_data

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType
from math import sqrt, pow

# Initialize Spark Session
spark = SparkSession.builder.appName("FindVectorID").getOrCreate()

# Load the data vectors (iris_data) and class vectors (iris_class)
iris_data = spark.read.option("header", "true").csv("/content/gdrive/My Drive/iris/iris.csv")
iris_class = spark.read.option("header", "true").csv("/content/gdrive/My Drive/iris/iris_classes.csv")

# Collect class vectors and broadcast them to workers
class_vectors = iris_class.collect()
broadcast_class_vectors = spark.sparkContext.broadcast(class_vectors)

# Define a function to compute Euclidean distance between input vector and class vector
def compute_distance(input_vector, class_vector):
    return sqrt(
        pow(float(input_vector[0]) - float(class_vector["Sepal Length"]), 2) +
        pow(float(input_vector[1]) - float(class_vector[" Sepal Width"]), 2) +
        pow(float(input_vector[2]) - float(class_vector[" Petal Length"]), 2) +
        pow(float(input_vector[3]) - float(class_vector[" Petal Width"]), 2)
    )

# Define the classification function
def classify_iris(input_vector):
    # Access the broadcasted class vectors
    class_vectors = broadcast_class_vectors.value
    class_vector_1 = class_vectors[0]
    class_vector_2 = class_vectors[1]
    class_vector_3 = class_vectors[2]
```

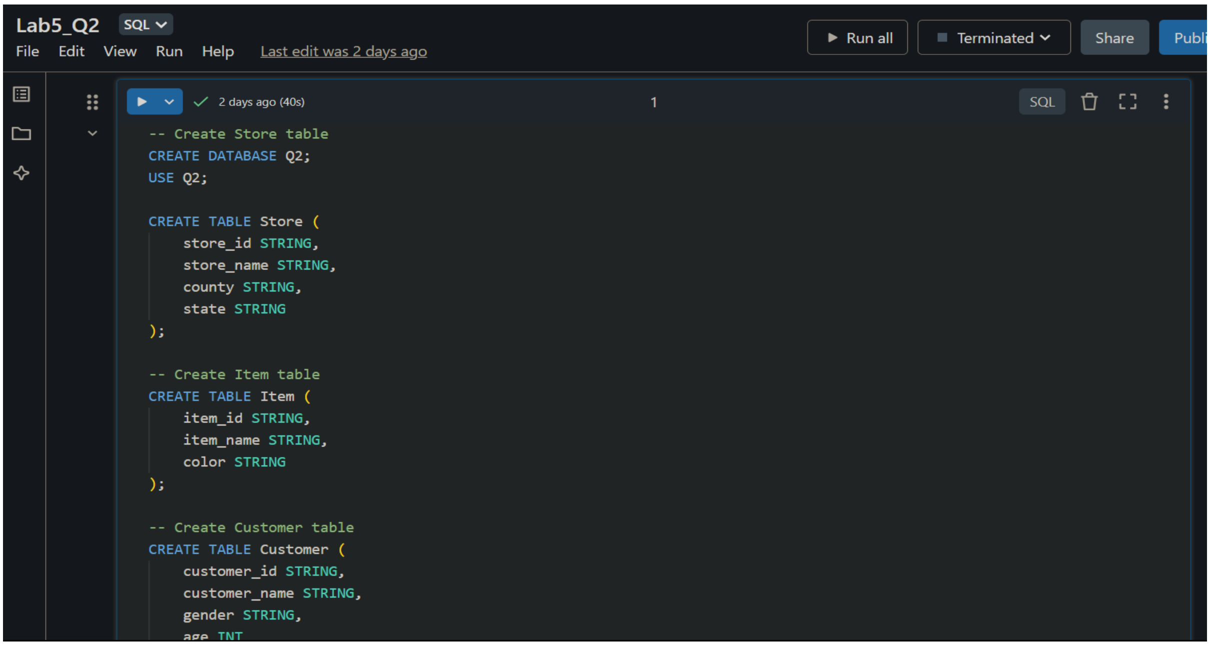
Part 2: SQL Notebook in Databricks

Do this exercise in a SQL notebook of databricks. Here let us do some OLAP exercises in Spark-SQL. Exercises here are inspired by an OLAP demo[1] from Jennifer Widom, Stanford University. It works on the same star schema discussed in the lectures. DDL and insert scripts are available here.

Perform the following operations on this dataset.

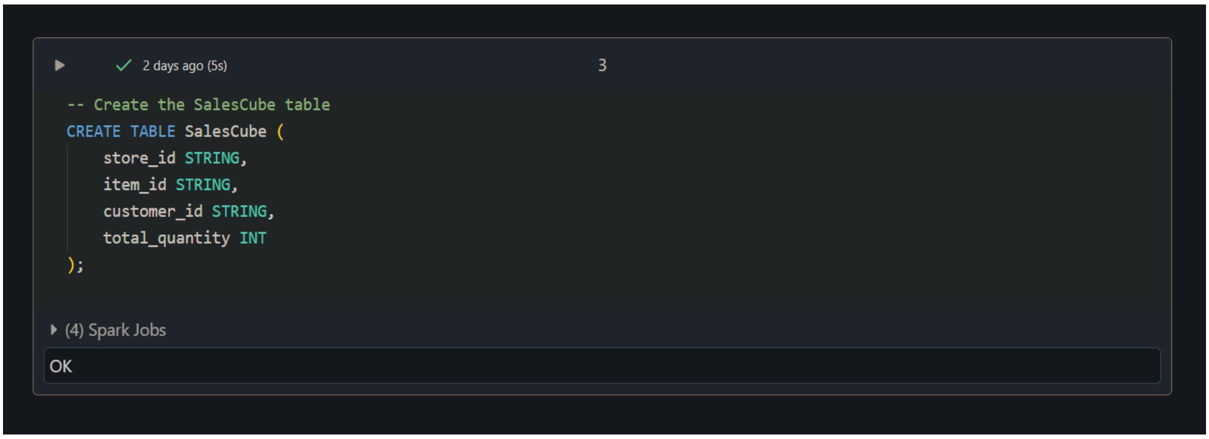
Q1. Create tables for the schema as defined in DDL script and add data using the INSERT scripts for all tables.

Code and Output:



```sql
-- Create Store table
CREATE DATABASE Q2;
USE Q2;

CREATE TABLE Store (
    store_id STRING,
    store_name STRING,
    county STRING,
    state STRING
);

-- Create Item table
CREATE TABLE Item (
    item_id STRING,
    item_name STRING,
    color STRING
);

-- Create Customer table
CREATE TABLE Customer (
    customer_id STRING,
    customer_name STRING,
    gender STRING,
    age INT
```

Q2: Create a Cube that Stores Sales Amount summaries. Let this Cube be materialized as a relational table and named as SalesCube.

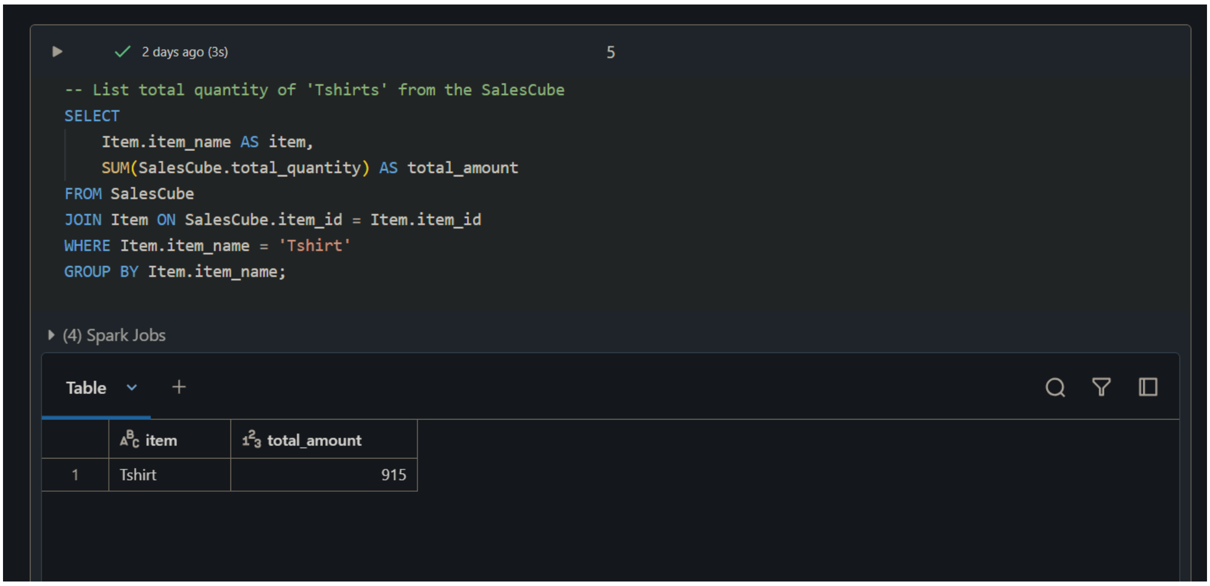


```sql
-- Create the SalesCube table
CREATE TABLE SalesCube (
    store_id STRING,
    item_id STRING,
    customer_id STRING,
    total_quantity INT
);
```

```
OK
```

Q-3: List total sales (amount) of all items of category 'Tshirts' Project: item , sum(amount)

Code and Output:



2 days ago (3s)

5

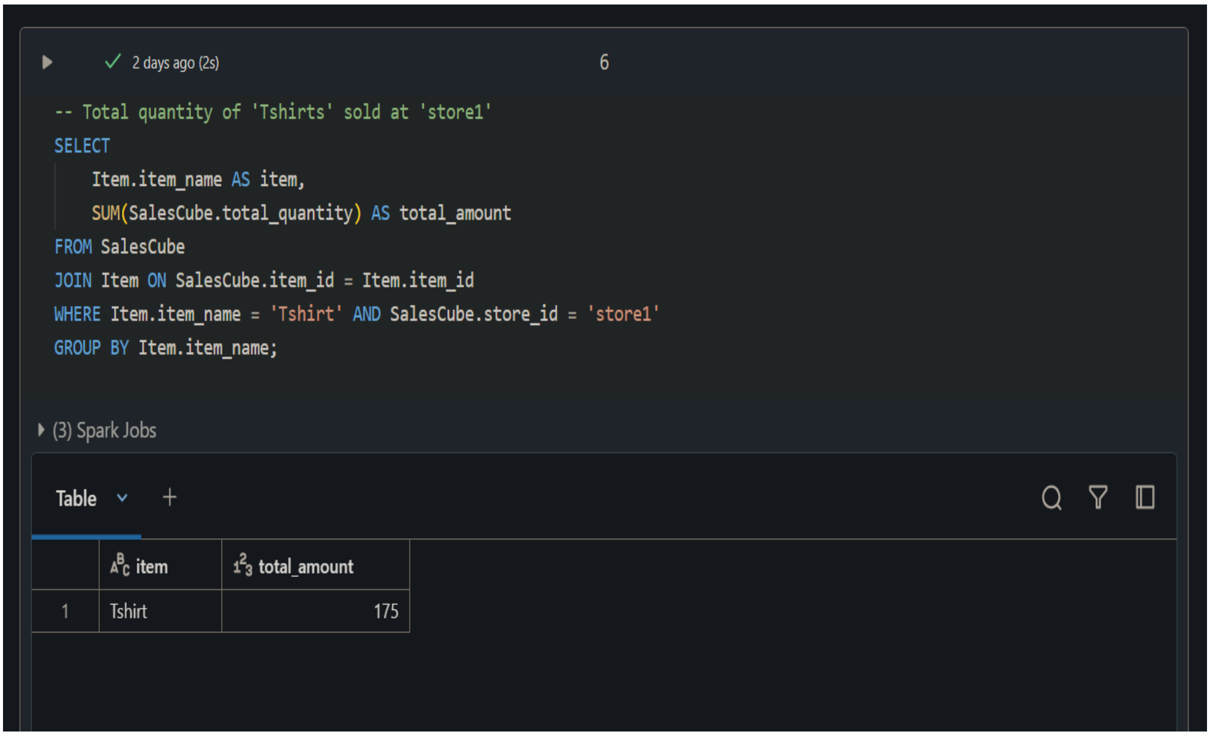
```
-- List total quantity of 'Tshirts' from the SalesCube
SELECT
    Item.item_name AS item,
    SUM(SalesCube.total_quantity) AS total_amount
FROM SalesCube
JOIN Item ON SalesCube.item_id = Item.item_id
WHERE Item.item_name = 'Tshirt'
GROUP BY Item.item_name;
```

(4) Spark Jobs

	item	total_amount
1	Tshirt	915

Q-4: List total sales (amount) of all items of category 'Tshirts' at 'store1' Project: item , sum(amount)

Code and Output:



2 days ago (2s)

6

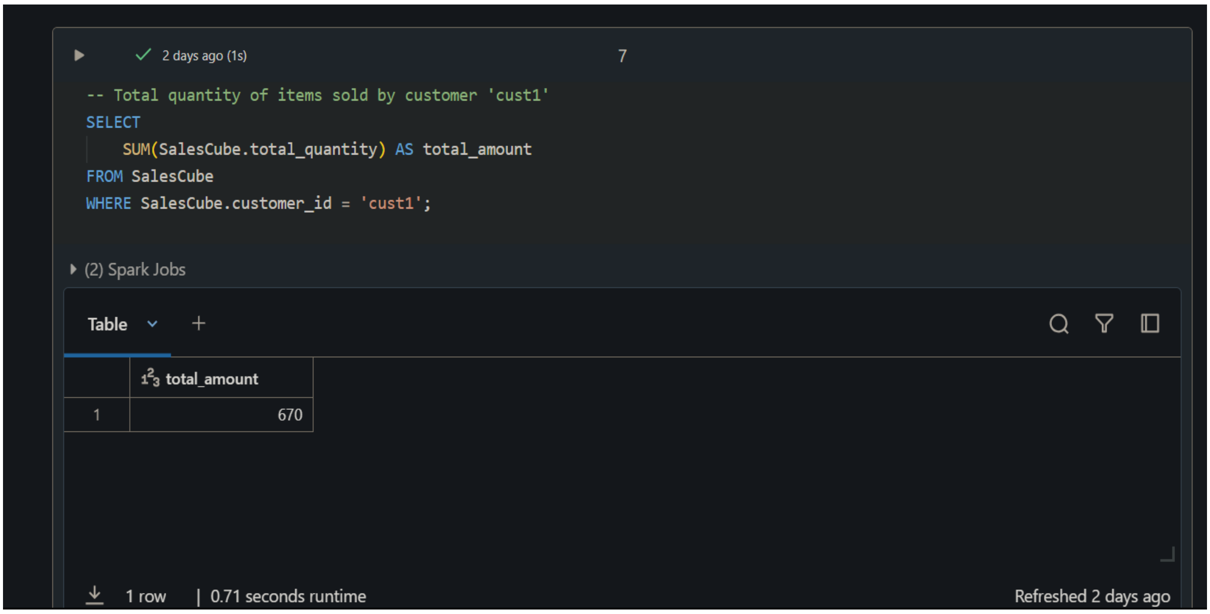
```
-- Total quantity of 'Tshirts' sold at 'store1'
SELECT
    Item.item_name AS item,
    SUM(SalesCube.total_quantity) AS total_amount
FROM SalesCube
JOIN Item ON SalesCube.item_id = Item.item_id
WHERE Item.item_name = 'Tshirt' AND SalesCube.store_id = 'store1'
GROUP BY Item.item_name;
```

(3) Spark Jobs

	item	total_amount
1	Tshirt	175

Q5: Give total sales (amount) of customer 'cust1' . Project: sum(amount)

Code and Output:



```
-- Total quantity of items sold by customer 'cust1'
SELECT
    SUM(SalesCube.total_quantity) AS total_amount
FROM SalesCube
WHERE SalesCube.customer_id = 'cust1';
```

(2) Spark Jobs

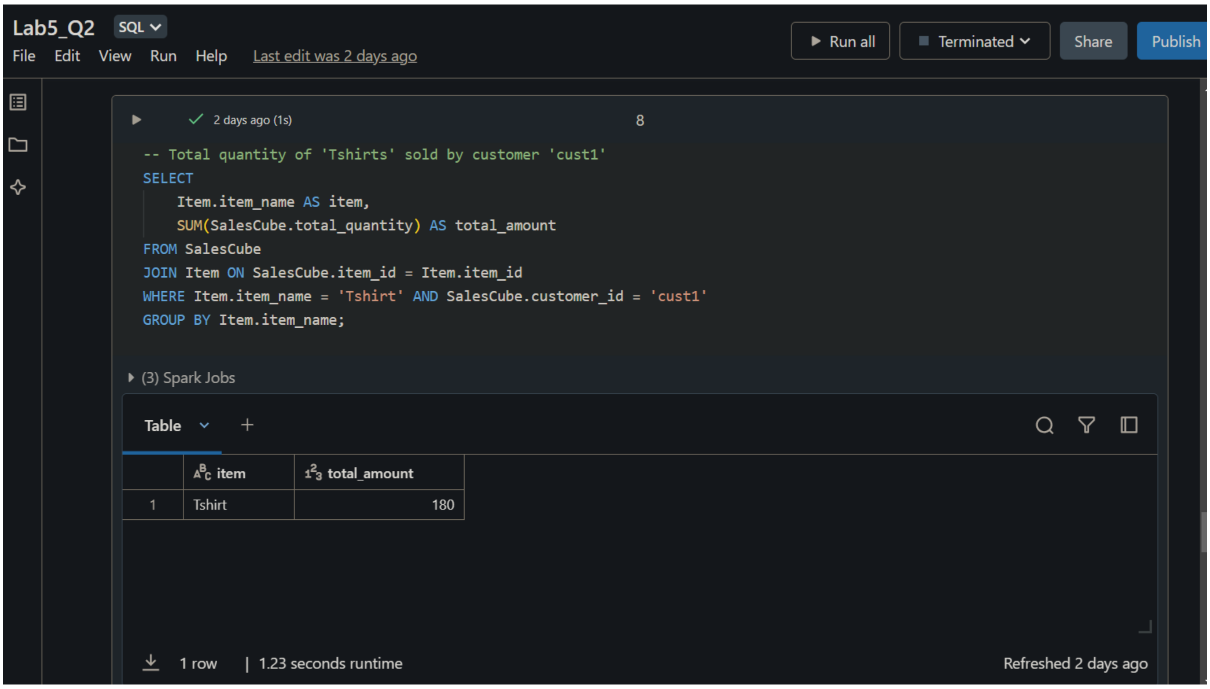
	total_amount
1	670

1 row | 0.71 seconds runtime

Refreshed 2 days ago

Q6: Give total sales (amount) of customer 'cust1' from items of category 'Tshirts' Project: item , sum(amount)

Code and Output:



```
-- Total quantity of 'Tshirts' sold by customer 'cust1'
SELECT
    Item.item_name AS item,
    SUM(SalesCube.total_quantity) AS total_amount
FROM SalesCube
JOIN Item ON SalesCube.item_id = Item.item_id
WHERE Item.item_name = 'Tshirt' AND SalesCube.customer_id = 'cust1'
GROUP BY Item.item_name;
```

(3) Spark Jobs

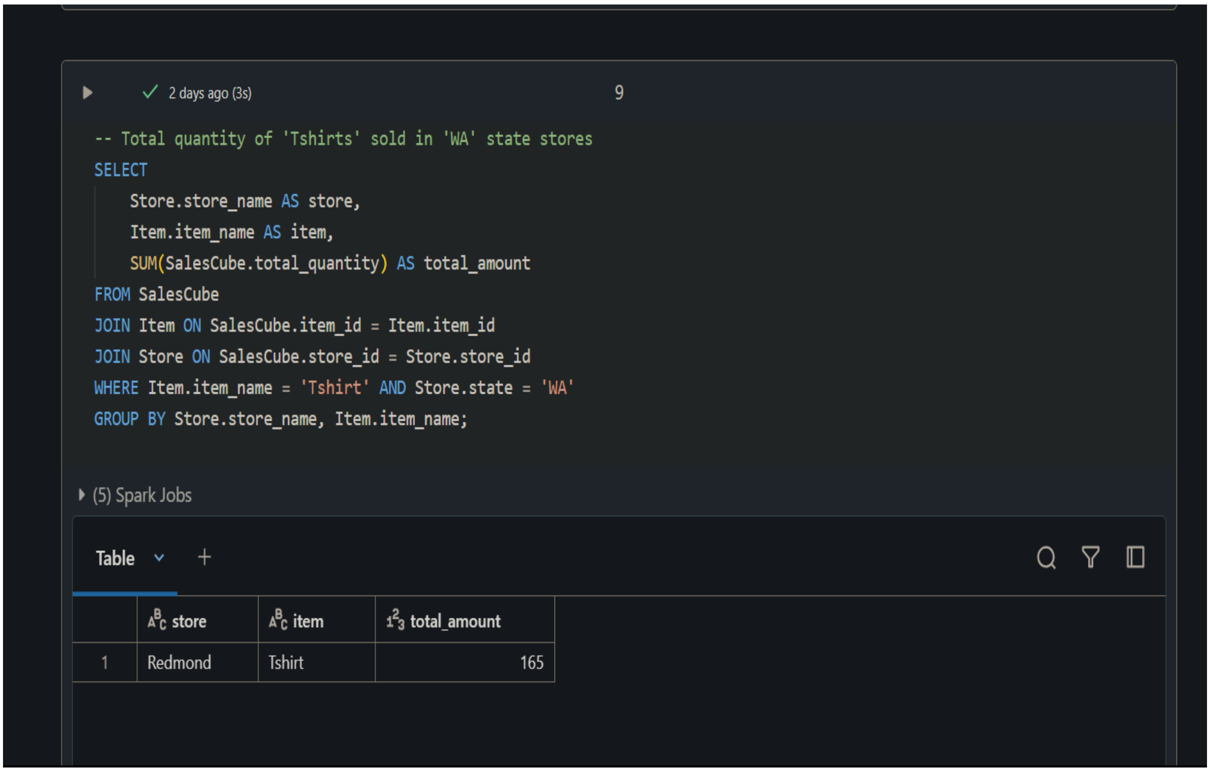
	item	total_amount
1	Tshirt	180

1 row | 1.23 seconds runtime

Refreshed 2 days ago

Q7: Give total sales (amount) of all stores of 'WA' state for items of category 'Tshirts' Project: store , item , sum(amount)

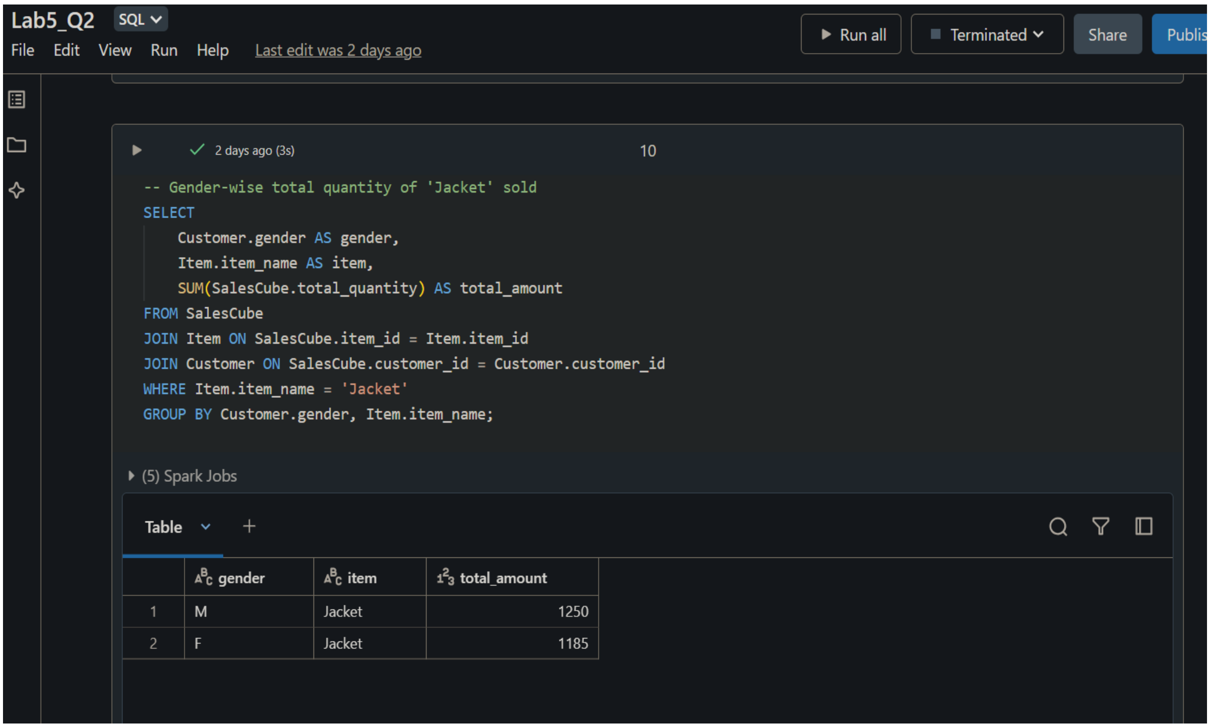
Code and output:



```sql
-- Total quantity of 'Tshirts' sold in 'WA' state stores
SELECT
    Store.store_name AS store,
    Item.item_name AS item,
    SUM(SalesCube.total_quantity) AS total_amount
FROM SalesCube
JOIN Item ON SalesCube.item_id = Item.item_id
JOIN Store ON SalesCube.store_id = Store.store_id
WHERE Item.item_name = 'Tshirt' AND Store.state = 'WA'
GROUP BY Store.store_name, Item.item_name;
```

	store	item	total_amount
1	Redmond	Tshirt	165

Q8: Give gender-wise total sales (amount) of items in the 'Jacket' category. Project: gender , item , sum(amount)



```sql
-- Gender-wise total quantity of 'Jacket' sold
SELECT
    Customer.gender AS gender,
    Item.item_name AS item,
    SUM(SalesCube.total_quantity) AS total_amount
FROM SalesCube
JOIN Item ON SalesCube.item_id = Item.item_id
JOIN Customer ON SalesCube.customer_id = Customer.customer_id
WHERE Item.item_name = 'Jacket'
GROUP BY Customer.gender, Item.item_name;
```

	gender	item	total_amount
1	M	Jacket	1250
2	F	Jacket	1185