BIG DATA LAB ASSIGNMENT THREE (WEEK5)

Devansh Sanghvi
Department Of Biotechnology
Indian Institute of Technology, Madras
Chennai, India
be19b002@smail.iitm.ac.in

***Abstract*—This document aims to answer the two questions in Assignment Three of the course: Big Data Lab. The python scripts used in the submission can be found at Submission Files**

IMPORTANT NOTE: All outputs in this assignment will have the ID: **sanghvidevansh23** instead of Roll No. This is because I have signed into Google Cloud Console using my personal email ID.

I. Question One

Question: Write PySpark code to implement SCD Type II on the sample customer master data frame.

Solution

1) Start the cloud shell. Enable the dataproc API, and create a new cluster. Set the various variables - such as bucket name, region name, etc. Follow the instructions given at the start of the tutorial.
2) Once this is done, get the python script ready. The required python script has also been attached with the submission.

Fig. 1. Python File for the PySPark Task: Part 1

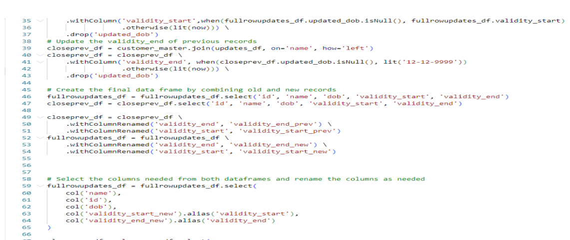

Fig. 2. Python File for the PySPark Task: Part 2

3) Now that the python script is ready, run the pyspark code to perform the required operation.

```
    col('name'),
    col('id'),
    col('dob'),
    col('validity_start_prev').alias('validity_start'),
    col('validity_end_prev').alias('validity_end')
)

# Union the dataframes
final_df = fullrowupdates_df.union(closeprev_df).dropDuplicates()

# Order the final dataframe by id
final_df = final_df.orderBy(asc("validity_end"),asc("validity_start"),asc("id"))
final_df.show()
```

Fig. 3. Python File for the PySPark Task: Part 3

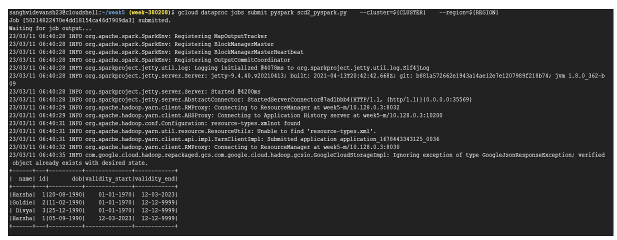

Fig. 4. Run the PySpark Code

name	id	dob	validity_start	validity_end
Harsha	1	20-08-1990	01-01-1970	12-03-2023
Goldie	2	11-02-1990	01-01-1970	12-12-9999
Divya	3	25-12-1990	01-01-1970	12-12-9999
Harsha	1	05-09-1990	12-03-2023	12-12-9999

Fig. 5. Output for the PySpark Job

4) You can view the output of the PySpark Job now.
5) As can be seen from the output, the results are correct.

II. Question Two

Question: Write SparkSQL code to implement SCD Type II on the sample customer master data frame.

Solution

1) Start the cloud shell. Enable the dataproc API, and create a new cluster. Set the various variables - such as bucket name, region name, etc. Follow the instructions given at the start of the tutorial.
2) Once this is done, ready the python script using Spark-SQL. The required python script has also been attached with the submission.

```python
from pyspark.sql.functions import col, when, lit
from pyspark.sql import SparkSession
import pandas as pd
import numpy as np
# Given a dataframe of customer records, where
# validity_start is set to a date from a long time ago - e.g. 01-01-1970
# while validity_end will be set to a long time in the future
# such that there is only 1 record per customer that is valid as on date
customer_master=[[1,'Harsha','20-08-1990','01-01-1970','12-12-9999'],[2,'Goldie','11-02-1990','01-01-1970','12-12-9999'],[3,'Divya','25-12-1990','01-01-1970','12-12-9999']]
cm_df=pd.DataFrame(customer_master,columns=['id','name','dob','validity_start','validity_end'])

# ... and given some updates to some of the records
# as of now
updates=[['Harsha','05-09-1990']]
updates_df=pd.DataFrame(updates,columns=['name','updated_dob'])
now='12-03-2023'

# Create SparkSession
spark = SparkSession.builder.appName('SCD Type II').getOrCreate()

# Convert the pandas data frame to PySpark data frame
customer_master = spark.createDataFrame(cm_df)
updates_df = spark.createDataFrame(updates_df)
customer_master.createOrReplaceTempView("customer_master")
updates_df.createOrReplaceTempView("updates_df")

#Make the dataframe with the newly updated data
fullrowupdates_df = spark.sql("""
    SELECT cm.name, cm.id,
```

Fig. 6. Python File for the SparkSQL Task: Part 1

```python

#Make the dataframe with the newly updated data
fullrowupdates_df = spark.sql("""
    SELECT cm.name, cm.id,
        CASE WHEN ud.updated_dob IS NULL THEN cm.dob ELSE ud.updated_dob END AS dob,
        CASE WHEN ud.updated_dob IS NULL THEN cm.validity_start ELSE '{0}' END AS validity_start,
        cm.validity_end AS validity_end
    FROM customer_master cm
    LEFT JOIN updates_df ud ON cm.name = ud.name
""".format('12-03-2023'))

# Update the validity_end of previous records
closeprev_df = spark.sql("""
    SELECT cm.name, cm.id, cm.dob, cm.validity_start,
        CASE WHEN ud.updated_dob IS NULL THEN '12-12-9999' ELSE '{0}' END AS validity_end
    FROM customer_master cm
    LEFT JOIN updates_df ud ON cm.name = ud.name
""".format('12-03-2023'))

#Create the final dataframe using SQL to merge historical data with the new data
fullrowupdates_df.createOrReplaceTempView("fullrowupdates")
closeprev_df.createOrReplaceTempView("closeprev")
final_df = spark.sql("""
    SELECT DISTINCT *
    FROM (
        SELECT * FROM fullrowupdates
        UNION ALL
        SELECT * FROM closeprev
```

Fig. 7. Python File for the SparkSQL Task: Part 2

```python
# Update the validity_end of previous records
closeprev_df = spark.sql("""
    SELECT cm.name, cm.id, cm.dob, cm.validity_start,
        CASE WHEN ud.updated_dob IS NULL THEN '12-12-9999' ELSE '{0}' END AS validity_end
    FROM customer_master cm
    LEFT JOIN updates_df ud ON cm.name = ud.name
""".format('12-03-2023'))

#Create the final dataframe using SQL to merge historical data with the new data
fullrowupdates_df.createOrReplaceTempView("fullrowupdates")
closeprev_df.createOrReplaceTempView("closeprev")
final_df = spark.sql("""
    SELECT DISTINCT *
    FROM (
        SELECT * FROM fullrowupdates
        UNION ALL
        SELECT * FROM closeprev
    ) temp
    ORDER BY validity_end, validity_start, id
""")

final_df.show()
```

Fig. 8. Python File for the SparkSQL Task: Part 3

3) Now that the python script is ready, run the python code to perform the required operation.
4) You can view the output of the SparkSQL Job now.

The two tables can be compared with the python script provided for reference: "week5.py" whose results are the following and match exactly with the two given results above:

```
sanghvidevansh23@cloudshell:~/week5/q2 (week-380208)$ gcloud dataproc jobs submit pyspark scd2_sparksql_copy.py    --cluster=${CLUSTER}    --region=${REGION}
Job [6c96b8dfbf2a496cb39e83147dba6997] submitted.
Waiting for job output...
23/03/11 13:16:33 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
23/03/11 13:16:33 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
23/03/11 13:16:33 INFO org.apache.spark.SparkEnv: Registering BlockManagerMasterHeartbeat
23/03/11 13:16:33 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
23/03/11 13:16:33 INFO org.sparkproject.jetty.util.log: Logging initialized @3946ms to org.sparkproject.jetty.util.log.Slf4jLog
23/03/11 13:16:33 INFO org.sparkproject.jetty.server.Server: jetty-9.4.40.v20210413; built: 2021-04-13T20:42:42.668Z; git: b881a572662e1943a14ae12e7e1207989f218b74; jvm 1.8.0_362-b09
23/03/11 13:16:33 INFO org.sparkproject.jetty.server.Server: Started @4047ms
23/03/11 13:16:33 INFO org.sparkproject.jetty.server.AbstractConnector: Started ServerConnector@1151b172{HTTP/1.1, (http/1.1)}{0.0.0.0:46801}
23/03/11 13:16:34 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at week5-m/10.128.0.5:8032
23/03/11 13:16:34 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at week5-m/10.128.0.5:10200
23/03/11 13:16:36 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
23/03/11 13:16:36 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
23/03/11 13:16:36 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1678530119266_0023
23/03/11 13:16:37 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at week5-m/10.128.0.5:8030
23/03/11 13:16:40 INFO com.google.cloud.hadoop.repackaged.gcs.com.google.cloud.hadoop.gcsio.GoogleCloudStorageImpl: Ignoring exception of type GoogleJsonResponseException; verified object already exis
ts with desired state.
```

Fig. 9. Run the python Code

```
+------+---+----------+--------------+------------+
|  name| id|       dob|validity_start|validity_end|
+------+---+----------+--------------+------------+
|Harsha|  1|20-08-1990|    01-01-1970|  12-03-2023|
|Goldie|  2|11-02-1990|    01-01-1970|  12-12-9999|
| Divya|  3|25-12-1990|    01-01-1970|  12-12-9999|
|Harsha|  1|05-09-1990|    12-03-2023|  12-12-9999|
+------+---+----------+--------------+------------+
```

Fig. 10. Output for the SparkSQL Job

name	id	dob	validity_start	validity_end
Harsha	1	20-08-1990	01-01-1970	12-03-2023
Goldie	2	11-02-1990	01-01-1970	12-12-9999
Divya	3	25-12-1990	01-01-1970	12-12-9999
Harsha	1	05-09-1990	12-03-2023	12-12-9999

Fig. 11. Output from the reference .py file