

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.

[illegible]

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

[illegible]

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.

```
68     col('name'),
69     col('id'),
70     col('dob'),
71     col('validity_start_prev').alias('validity_start'),
72     col('validity_end_prev').alias('validity_end')
73 )
74
75 # Union the dataframes
76 final_df = fullrowupdates_df.union(closeprev_df).dropDuplicates()
77
78 # Order the final dataframe by id
79 final_df = final_df.orderBy(asc("validity_end"),asc("validity_start"),asc("id"))
80 final_df.show()
81
82
```

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

```

$ docker build --no-cache --tag myproj:cd myproj:cd . --output=images
Step 1/10: FROM docker.io/library/python:3.8.10-slim@sha256:1000000000000000000000000000000000000000000000000000000000000000
Step 2/10: WORKDIR /app
Step 3/10: COPY requirements.txt ./
Step 4/10: RUN pip install --no-cache-dir -r requirements.txt
Step 5/10: COPY . .
Step 6/10: RUN python3 -m pip install --no-cache-dir -r requirements.txt
Step 7/10: RUN python3 -m pip install --no-cache-dir -r requirements.txt
Step 8/10: RUN python3 -m pip install --no-cache-dir -r requirements.txt
Step 9/10: RUN python3 -m pip install --no-cache-dir -r requirements.txt
Step 10/10: CMD ["python3", "main.py"]

```

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.

```

7 from pyspark.sql.functions import col, when, lit
8
9 from pyspark.sql import SparkSession
10 import pandas as pd
11
12 import numpy as np
13
14 # select a dataframe of customer records, where
15 # validity_start is not to a date from a long time ago - e.g. 40-41-1970
16 # and validity_end will set to a long time in the future
17 # such that there is only 1 record per customer that is valid as of now
18 customer_table["validity_start"] = when(
19     col("validity_start") < "40-41-1970", lit("12-31-9999")) | when(
20     col("validity_start") > "40-41-1970", lit("01-01-2099")) | col("validity_start")
21
22 # ... and given some updates to some of the records
23
24 # e.g. as of now
25 updates=[{"customer_id": "40-40-1970"}]
26
27 # update the SparkSession dataframe columns: "name", "updated_at"
28 updates_df=SparkSession.createDataFrame(
29     updates, ["customer_id", "updated_at"])
30
31 # create SparkSession
32 spark = SparkSession.builder.appname("CDS Type 1").getOrCreate()
33
34 # convert the pandas data frame to pyspark data frame
35 customer_table = spark.createDataFrame(customer_table)
36 updates_df = spark.createDataFrame(updates)
37
38 customer_table.createTableIfNotExists(customer_table)
39 updates_df.createTableIfNotExists(updates_df)
40
41 # make the dataframe with the newly updated data
42 full_dataframe = spark.sql("""
43     SELECT cu.name, cu.id,
44     up.updated_at
45     FROM customer_table cu
46     LEFT JOIN updates up
47     ON cu.customer_id = up.customer_id
48 """)
49
50 # write the dataframe to parquet
51 full_dataframe.write.parquet("data/customer_table.parquet")
52
53 # read the dataframe back
54 df = spark.read.parquet("data/customer_table.parquet")
55
56 # write the dataframe back to parquet
57 df.write.parquet("data/customer_table.parquet")
58
59 # read the dataframe back
60 df = spark.read.parquet("data/customer_table.parquet")
61
62 # write the dataframe back to parquet
63 df.write.parquet("data/customer_table.parquet")
64
65 # read the dataframe back
66 df = spark.read.parquet("data/customer_table.parquet")
67
68 # write the dataframe back to parquet
69 df.write.parquet("data/customer_table.parquet")
70
71 # read the dataframe back
72 df = spark.read.parquet("data/customer_table.parquet")
73
74 # write the dataframe back to parquet
75 df.write.parquet("data/customer_table.parquet")
76
77 # read the dataframe back
78 df = spark.read.parquet("data/customer_table.parquet")
79
80 # write the dataframe back to parquet
81 df.write.parquet("data/customer_table.parquet")
82
83 # read the dataframe back
84 df = spark.read.parquet("data/customer_table.parquet")
85
86 # write the dataframe back to parquet
87 df.write.parquet("data/customer_table.parquet")
88
89 # read the dataframe back
90 df = spark.read.parquet("data/customer_table.parquet")
91
92 # write the dataframe back to parquet
93 df.write.parquet("data/customer_table.parquet")
94
95 # read the dataframe back
96 df = spark.read.parquet("data/customer_table.parquet")
97
98 # write the dataframe back to parquet
99 df.write.parquet("data/customer_table.parquet")
100
101 # read the dataframe back
102 df = spark.read.parquet("data/customer_table.parquet")
103
104 # write the dataframe back to parquet
105 df.write.parquet("data/customer_table.parquet")
106
107 # read the dataframe back
108 df = spark.read.parquet("data/customer_table.parquet")
109
110 # write the dataframe back to parquet
111 df.write.parquet("data/customer_table.parquet")
112
113 # read the dataframe back
114 df = spark.read.parquet("data/customer_table.parquet")
115
116 # write the dataframe back to parquet
117 df.write.parquet("data/customer_table.parquet")
118
119 # read the dataframe back
120 df = spark.read.parquet("data/customer_table.parquet")
121
122 # write the dataframe back to parquet
123 df.write.parquet("data/customer_table.parquet")
124
125 # read the dataframe back
126 df = spark.read.parquet("data/customer_table.parquet")
127
128 # write the dataframe back to parquet
129 df.write.parquet("data/customer_table.parquet")
130
131 # read the dataframe back
132 df = spark.read.parquet("data/customer_table.parquet")
133
134 # write the dataframe back to parquet
135 df.write.parquet("data/customer_table.parquet")
136
137 # read the dataframe back
138 df = spark.read.parquet("data/customer_table.parquet")
139
140 # write the dataframe back to parquet
141 df.write.parquet("data/customer_table.parquet")
142
143 # read the dataframe back
144 df = spark.read.parquet("data/customer_table.parquet")
145
146 # write the dataframe back to parquet
147 df.write.parquet("data/customer_table.parquet")
148
149 # read the dataframe back
150 df = spark.read.parquet("data/customer_table.parquet")
151
152 # write the dataframe back to parquet
153 df.write.parquet("data/customer_table.parquet")
154
155 # read the dataframe back
156 df = spark.read.parquet("data/customer_table.parquet")
157
158 # write the dataframe back to parquet
159 df.write.parquet("data/customer_table.parquet")
160
161 # read the dataframe back
162 df = spark.read.parquet("data/customer_table.parquet")
163
164 # write the dataframe back to parquet
165 df.write.parquet("data/customer_table.parquet")
166
167 # read the dataframe back
168 df = spark.read.parquet("data/customer_table.parquet")
169
170 # write the dataframe back to parquet
171 df.write.parquet("data/customer_table.parquet")
172
173 # read the dataframe back
174 df = spark.read.parquet("data/customer_table.parquet")
175
176 # write the dataframe back to parquet
177 df.write.parquet("data/customer_table.parquet")
178
179 # read the dataframe back
180 df = spark.read.parquet("data/customer_table.parquet")
181
182 # write the dataframe back to parquet
183 df.write.parquet("data/customer_table.parquet")
184
185 # read the dataframe back
186 df = spark.read.parquet("data/customer_table.parquet")
187
188 # write the dataframe back to parquet
189 df.write.parquet("data/customer_table.parquet")
190
191 # read the dataframe back
192 df = spark.read.parquet("data/customer_table.parquet")
193
194 # write the dataframe back to parquet
195 df.write.parquet("data/customer_table.parquet")
196
197 # read the dataframe back
198 df = spark.read.parquet("data/customer_table.parquet")
199
200 # write the dataframe back to parquet
201 df.write.parquet("data/customer_table.parquet")
202
203 # read the dataframe back
204 df = spark.read.parquet("data/customer_table.parquet")
205
206 # write the dataframe back to parquet
207 df.write.parquet("data/customer_table.parquet")
208
209 # read the dataframe back
210 df = spark.read.parquet("data/customer_table.parquet")
211
212 # write the dataframe back to parquet
213 df.write.parquet("data/customer_table.parquet")
214
215 # read the dataframe back
216 df = spark.read.parquet("data/customer_table.parquet")
217
218 # write the dataframe back to parquet
219 df.write.parquet("data/customer_table.parquet")
220
221 # read the dataframe back
222 df = spark.read.parquet("data/customer_table.parquet")
223
224 # write the dataframe back to parquet
225 df.write.parquet("data/customer_table.parquet")
226
227 # read the dataframe back
228 df = spark.read.parquet("data/customer_table.parquet")
229
230 # write the dataframe back to parquet
231 df.write.parquet("data/customer_table.parquet")
232
233 # read the dataframe back
234 df = spark.read.parquet("data/customer_table.parquet")
235
236 # write the dataframe back to parquet
237 df.write.parquet("data/customer_table.parquet")
238
239 # read the dataframe back
240 df = spark.read.parquet("data/customer_table.parquet")
241
242 # write the dataframe back to parquet
243 df.write.parquet("data/customer_table.parquet")
244
245 # read the dataframe back
246 df = spark.read.parquet("data/customer_table.parquet")
247
248 # write the dataframe back to parquet
249 df.write.parquet("data/customer_table.parquet")
250
251 # read the dataframe back
252 df = spark.read.parquet("data/customer_table.parquet")
253
254 # write the dataframe back to parquet
255 df.write.parquet("data/customer_table.parquet")
256
257 # read the dataframe back
258 df = spark.read.parquet("data/customer_table.parquet")
259
260 # write the dataframe back to parquet
261 df.write.parquet("data/customer_table.parquet")
262
263 # read the dataframe back
264 df = spark.read.parquet("data/customer_table.parquet")
265
266 # write the dataframe back to parquet
267 df.write.parquet("data/customer_table.parquet")
268
269 # read the dataframe back
270 df = spark.read.parquet("data/customer_table.parquet")
271
272 # write the dataframe back to parquet
273 df.write.parquet("data/customer_table.parquet")
274
275 # read the dataframe back
276 df = spark.read.parquet("data/customer_table.parquet")
277
278 # write the dataframe back to parquet
279 df.write.parquet("data/customer_table.parquet")
280
281 # read the dataframe back
282 df = spark.read.parquet("data/customer_table.parquet")
283
284 # write the dataframe back to parquet
285 df.write.parquet("data/customer_table.parquet")
286
287 # read the dataframe back
288 df = spark.read.parquet("data/customer_table.parquet")
289
290 # write the dataframe back to parquet
291 df.write.parquet("data/customer_table.parquet")
292
293 # read the dataframe back
294 df = spark.read.parquet("data/customer_table.parquet")
295
296 # write the dataframe back to parquet
297 df.write.parquet("data/customer_table.parquet")
298
299 # read the dataframe back
300 df = spark.read.parquet("data/customer_table.parquet")
301
302 # write the dataframe back to parquet
303 df.write.parquet("data/customer_table.parquet")
304
305 # read the dataframe back
306 df = spark.read.parquet("data/customer_table.parquet")
307
308 # write the dataframe back to parquet
309 df.write.parquet("data/customer_table.parquet")
310
311 # read the dataframe back
312 df = spark.read.parquet("data/customer_table.parquet")
313
314 # write the dataframe back to parquet
315 df.write.parquet("data/customer_table.parquet")
316
317 # read the dataframe back
318 df = spark.read.parquet("data/customer_table.parquet")
319
320 # write the dataframe back to parquet
321 df.write.parquet("data/customer_table.parquet")
322
323 # read the dataframe back
324 df = spark.read.parquet("data/customer_table.parquet")
325
326 # write the dataframe back to parquet
327 df.write.parquet("data/customer_table.parquet")
328
329 # read the dataframe back
330 df = spark.read.parquet("data/customer_table.parquet")
331
332 # write the dataframe back to parquet
333 df.write.parquet("data/customer_table.parquet")
334
335 # read the dataframe back
336 df = spark.read.parquet("data/customer_table.parquet")
337
338 # write the dataframe back to parquet
339 df.write.parquet("data/customer_table.parquet")
340
341 # read the dataframe back
342 df = spark.read.parquet("data/customer_table.parquet")
343
344 # write the dataframe back to parquet
345 df.write.parquet("data/customer_table.parquet")
346
347 # read the dataframe back
348 df = spark.read.parquet("data/customer_table.parquet")
349
350 # write the dataframe back to parquet
351 df.write.parquet("data/customer_table.parquet")
352
353 # read the dataframe back
354 df = spark.read.parquet("data/customer_table.parquet")
355
356 # write the dataframe back to parquet
357 df.write.parquet("data/customer_table.parquet")
358
359 # read the dataframe back
360 df = spark.read.parquet("data/customer_table.parquet")
361
362 # write the dataframe back to parquet
363 df.write.parquet("data/customer_table.parquet")
364
365 # read the dataframe back
366 df = spark.read.parquet("data/customer_table.parquet")
367
368 # write the dataframe back to parquet
369 df.write.parquet("data/customer_table.parquet")
370
371 # read the dataframe back
372 df = spark.read.parquet("data/customer_table.parquet")
373
374 # write the dataframe back to parquet
375 df.write.parquet("data/customer_table.parquet")
376
377 # read the dataframe back
378 df = spark.read.parquet("data/customer_table.parquet")
379
380 # write the dataframe back to parquet
381 df.write.parquet("data/customer_table.parquet")
382
383 # read the dataframe back
384 df = spark.read.parquet("data/customer_table.parquet")
385
386 # write the dataframe back to parquet
387 df.write.parquet("data/customer_table.parquet")
388
389 # read the dataframe back
390 df = spark.read.parquet("data/customer_table.parquet")
391
392 # write the dataframe back to parquet
393 df.write.parquet("data/customer_table.parquet")
394
395 # read the dataframe back
396 df = spark.read.parquet("data/customer_table.parquet")
397
398 # write the dataframe back to parquet
399 df.write.parquet("data/customer_table.parquet")
400
401 # read the dataframe back
402 df = spark.read.parquet("data/customer_table.parquet")
403
404 # write the dataframe back to parquet
405 df.write.parquet("data/customer_table.parquet")
406
407 # read the dataframe back
408 df = spark.read.parquet("data/customer_table.parquet")
409
410 # write the dataframe back to parquet
411 df.write.parquet("data/customer_table.parquet")
412
413 # read the dataframe back
414 df =
```

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

```

26 #make the dataframe with the newly updated data
27 fullroundupdates_df = spark.sql("""
28     SELECT cn.name, cm.id,
29           CASE WHEN cm.updated_date IS NULL THEN cn.mob ELSE ud.updated_date END AS dob,
30           CASE WHEN cm.updated_date IS NULL THEN cn.validity_start ELSE ('%' || ud.validity_start || '%') END AS validity_start,
31           CASE WHEN cm.updated_date IS NULL THEN cn.validity_end ELSE ('%' || ud.validity_end || '%') END AS validity_end
32     FROM cn_validity, ud
33     LEFT JOIN updates_df ON cn.name = ud.name
34     --format('12-09-2025')
35
36 -- update the validity end of previous records
37 closeprev_df = spark.sql("""
38     SELECT cn.name, cm.id, cn.validity_start,
39           CASE WHEN ud.updated_date IS NULL THEN '12-12-9999' ELSE ('%' || ud.updated_date || '%') END AS validity_end,
40     FROM cn_validity, ud
41     LEFT JOIN updates_df ON cn.name = ud.name
42     --format('12-09-2025')
43
44 #create the final dataframe using SQL to merge historical data with the new data
45 fullroundupdates_df.createOrReplaceTempView('fullroundupdates')
46 closeprev_df.createOrReplaceTempView('closeprev')
47 final_df = spark.sql("""
48     SELECT DISTINCT
49     FROM
50         SELECT * FROM fullroundupdates
51     UNION ALL
52         SELECT * FROM closeprev

```

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

```

37 # Update the validity_end of previous records
38 closeprev_df = spark.sql("""
39     SELECT cm.name, cm.id, cm.dob, cm.validity_start,
40     CASE WHEN ud.updated_dob IS NULL THEN '12-12-9999'
41     FROM customer_master cm
42     LEFT JOIN updates ud ON cm.name = ud.name
43     """)
44 #Format ('12-01-2023')
45
46 #Create the final dataframe using SQL to merge historical data with the new data
47 fullupdates_df.createOrReplaceTempView("fullupdates")
48 closeprev_df.createOrReplaceTempView("closeprev")
49 final_df = spark.sql("""
50     SELECT DISTINCT *
51     FROM (
52         SELECT * FROM fullupdates
53         UNION ALL
54         SELECT * FROM closeprev
55     ) temp
56     ORDER BY validity_end, validity_start, id
57     """)
58 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:

[illegible]

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
Diyya	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

	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