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

Submitted by Shramana Sinha, 23f1002703

1. Dataproc Cluster Setup

A Dataproc cluster on Compute Engine was created to run the Spark job, with the following configuration:

Manager Node:

- Machine Series: E2
- Machine Type: e2-standard-2
- Primary Disk Size: 30 GB

Worker Nodes:

- Number of Nodes: 2
- Machine Series: E2
- Machine Type: e2-standard-2
- Primary Disk Size: 30 GB

ter: cluster-1536 / Cluster configuration

Cluster details | SUBMIT JOB | REFRESH | START | STOP | DE

Autoscaling	Off
Performance Enhancements	
Advanced optimizations	Off
Advanced execution layer	Off
Google Cloud Storage caching	Off
Dataproc Metastore	None
Scheduled deletion	Off
Confidential computing enabled?	Disabled
Master node	Standard (1 master, N workers)
Machine type	e2-standard-2
Number of GPUs	0
Primary disk type	pd-balanced
Primary disk size	30GB
Local SSDs	0
Worker nodes	2
Machine type	e2-standard-2
Number of GPUs	0
Primary disk type	pd-balanced
Primary disk size	30GB
Local SSDs	0
Secondary worker nodes	0

Screenshot 1: Dataproc Cluster Configuration

2. File Upload

The required input files were created based on the example data from the lecture. The following files were uploaded to the cluster's Master Node using SSH via the cloud console:

Files:

- `main.py` – The Spark script implementing SCD Type II logic.
- `source.csv` – Source dataset containing customer data with `customer_id` as the primary key.
- `target.csv` – Target dataset containing all columns from the source dataset, along with `effective_start_date`, `effective_end_date`, and `record_id` (its own primary key), as required by the SCD Type II implementation.

The files from local file system were transferred to hdfs using the following command:

```
Unset
hadoop fs -put -f source.csv source.csv
hadoop fs -put -f target.csv target.csv
```

```
sinhashrutaba@cluster-1536-m:~$ ls
main.py  source.csv  target.csv
sinhashrutaba@cluster-1536-m:~$ hadoop fs -put -f source.csv source.csv
sinhashrutaba@cluster-1536-m:~$ hadoop fs -put -f target.csv target.csv
sinhashrutaba@cluster-1536-m:~$ hdfs dfs -ls
Found 2 items
-rw-r--r--   2 sinhashrutaba hadoop        174 2025-03-01 18:47 source.csv
-rw-r--r--   2 sinhashrutaba hadoop        745 2025-03-01 18:47 target.csv
```

Screenshot 2: Terminal showing the files in Master Node's local system and hdfs

```
sinhashrutaba@cluster-1536-m:~$ hdfs dfs -cat source.csv
customer_id,name,city
1,Name1,City1
2,Name2,City2
3,Name3,City3
4,Name4,City4
5,Name5,City5
6,Name6,City6
7,Name7,City7
8,Name8,City8
9,Name9,City9
10,Name10,City10sinhashrutaba@cluster-1536-m:~$
```

Screenshot 3: The source.csv

```
sinhashrutaba@cluster-1536-m:~$ hdfs dfs -cat target.csv
record_id,customer_id,name,city,effective_start_date,effective_end_date
1,1,Name1,City1,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Z
2,2,Name2,City2,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Z
3,3,Name3,City3,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Z
4,4,Name4,City4,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Z
5,5,Name5,City5,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Z
6,6,Name6,City6,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Z
7,7,Name7,City7,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Z
8,8,Name8,City8,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Z
9,9,Name9,City9,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Z
10,10,Name10,City10,1735-01-01T00:00:00.000Z,9999-12-31T00:00:00.000Zsin
```

Screenshot 4: The target.csv

3. Input Data Preparation

The `source.csv` in the hdfs was modified to:

- Delete a row (row with customer id 5).
- Add a new row (row with customer id 11).
- Change details for an existing row (row with customer id 3).

This allowed testing the correctness of the SCD Type II implementation.

```
sinhashrutaba@cluster-1536-m:~$ hdfs dfs -cat source.csv
customer_id,name,city
1,Name1,City1
2,Name2,City2
3,Name3,City
4,Name4,City4
6,Name6,City6
7,Name7,City7
8,Name8,City8
9,Name9,City9
10,Name10,City10
11,Name11,City11
```

Screenshot 5: The modified source.csv

4. Spark Job Execution

The spark job was submitted to the cluster, using the following command:

```
Unset
spark-submit main.py
```

5. Code Explanation

Import Required Libraries

```python
from pyspark.sql import SparkSession
```

Data Loading and Preparation

The source and target data are read from CSV files into temporary tables. The maximum existing record ID of the target table is determined, along with queries to select business keys, target columns, and source columns.

```python
    spark.read.csv(source_path, header=True,
inferSchema=True).createOrReplaceTempView("source_table")
    spark.read.csv(target_path, header=True,
inferSchema=True).createOrReplaceTempView("target_table")

    business_keys_sql = ", ".join([f"t.{key} = s.{key}" for key in
business_keys])
    business_keys_select = ", ".join([f"s.{key}" for key in business_keys])
    target_columns_sql = ", ".join([col for col in
spark.table("target_table").columns])
    source_columns_sql = ", ".join([f"s.{col}" for col in
spark.table("source_table").columns])
    target_columns_sql_without_end_date = ", ".join( [ f"t.{col}" for col in
spark.table("target_table").columns if col != end_date_col ])

    max_id = spark.sql(f"""SELECT COALESCE(MAX({unique_id_col}), 0) as max_id
FROM target_table""").collect()[0]["max_id"]
```

Identify Records

Identifying Active Records: Filters the target table to retrieve only the currently active records. Active records will have their end date set to `9999-12-31`.

```python
spark.sql(f"""CREATE OR REPLACE TEMPORARY VIEW active_records AS
    SELECT * FROM target_table t
    WHERE t.{end_date_col} = '9999-12-31'""")
```

Detecting Changed Records: Joins the source and active records on business keys and filters for changes in the tracked columns.

```python
spark.sql(f"""CREATE OR REPLACE TEMPORARY VIEW changed_records AS
    SELECT t.*
    FROM active_records t
    JOIN source_table s ON {business_keys_sql}
    WHERE {' OR '.join([f't.{col} <> s.{col}' for col in tracked_columns])}""")
```

Identifying Deleted Records: Performs a left anti-join to detect records that exist in the active records but are missing from the source. These records are marked as deleted by setting their end date to the current timestamp.

```python
spark.sql(f"""CREATE OR REPLACE TEMPORARY VIEW deleted_records AS
    SELECT {target_columns_sql_without_end_date}, CURRENT_TIMESTAMP() as
{end_date_col}
    FROM active_records t
    LEFT ANTI JOIN source_table s ON {business_keys_sql} """)
```

Identifying Unchanged Records: Combines business keys of changed and deleted records, then uses a left anti-join with the target table to retrieve records that remain unchanged.

```python
spark.sql(f"""CREATE OR REPLACE TEMPORARY VIEW unchanged_records AS
    SELECT t.*
    FROM target_table t
    LEFT ANTI JOIN (
        SELECT {business_keys_select.replace('s.', '')} FROM changed_records
        UNION
        SELECT {business_keys_select.replace('s.', '')} FROM deleted_records
    ) combined ON {business_keys_sql.replace('s.', 'combined.')}""")
```

Manage Versions

Expiring Changed Records: Joins active records with changed records to expire outdated records by setting their end date to the current timestamp.

```python
spark.sql(f"""CREATE OR REPLACE TEMPORARY VIEW records_to_expire AS
    SELECT {target_columns_sql_without_end_date}, CURRENT_TIMESTAMP() as
{end_date_col}
    FROM active_records t
    JOIN changed_records c ON {business_keys_sql.replace('s.', 'c.')}""")
```

Creating New Versions of Changed Records: Generates new records for changed entries with an updated start date, an end date of `9999-12-31`, and a unique ID.

```python
spark.sql(f"""CREATE OR REPLACE TEMPORARY VIEW new_changed_records AS
    SELECT
        {source_columns_sql},
        CURRENT_TIMESTAMP() as {start_date_col},
        CAST('9999-12-31' AS TIMESTAMP) as {end_date_col},
        (ROW_NUMBER() OVER (ORDER BY {business_keys_select}) + {max_id}) AS
{unique_id_col}
    FROM source_table s
    JOIN changed_records c ON {business_keys_sql.replace('t.', 'c.')}""")
```

Identifying New Records: Identifies new records from the source that do not exist in the target, assigning appropriate start, end dates and unique IDs.

```python
spark.sql(f"""CREATE OR REPLACE TEMPORARY VIEW new_records AS
    SELECT
        {source_columns_sql},
        CURRENT_TIMESTAMP() as {start_date_col},
        CAST('9999-12-31' AS TIMESTAMP) as {end_date_col},
        (ROW_NUMBER() OVER (ORDER BY {business_keys_select}) + {max_id}) AS
{unique_id_col}
    FROM source_table s
    LEFT ANTI JOIN target_table t ON {business_keys_sql}""")
```

Final Dataset Assembly

Combines all processed record types into a final DataFrame.

```python
final_df = spark.sql(f"""SELECT * FROM unchanged_records
    UNION ALL
```

```
    SELECT {target_columns_sql} FROM records_to_expire
    UNION ALL
    SELECT {target_columns_sql} FROM new_changed_records
    UNION ALL
    SELECT {target_columns_sql} FROM new_records
    UNION ALL
    SELECT {target_columns_sql} FROM deleted_records""")
```

6. Output Verification

The output was displayed in the terminal and written to a CSV file. The output dataset accurately reflected the changes made to the source data:

- The deleted row in the source dataset (row with customer id 5) was correctly marked as expired in the output.
- The new row from the source dataset (row with customer id 11) was added with current timestamps.
- For the modified row (row with customer id 3), the older version was marked as expired, and a new version with updated details was added.

```
+---------+-----------+------+------+--------------------------+--------------------------+
|record_id|customer_id|name  |city  |effective_start_date      |effective_end_date        |
+---------+-----------+------+------+--------------------------+--------------------------+
|1        |1          |Name1 |City1 |1735-01-01 00:00:00       |9999-12-31 00:00:00       |
|2        |2          |Name2 |City2 |1735-01-01 00:00:00       |9999-12-31 00:00:00       |
|3        |3          |Name3 |City3 |1735-01-01 00:00:00       |2025-03-01 18:55:51.878449|
|4        |4          |Name4 |City4 |1735-01-01 00:00:00       |9999-12-31 00:00:00       |
|5        |5          |Name5 |City5 |1735-01-01 00:00:00       |2025-03-01 18:55:51.878449|
|6        |6          |Name6 |City6 |1735-01-01 00:00:00       |9999-12-31 00:00:00       |
|7        |7          |Name7 |City7 |1735-01-01 00:00:00       |9999-12-31 00:00:00       |
|8        |8          |Name8 |City8 |1735-01-01 00:00:00       |9999-12-31 00:00:00       |
|9        |9          |Name9 |City9 |1735-01-01 00:00:00       |9999-12-31 00:00:00       |
|10       |10         |Name10|City10|1735-01-01 00:00:00       |9999-12-31 00:00:00       |
|11       |3          |Name3 |City  |2025-03-01 18:55:51.878449|9999-12-31 00:00:00       |
|12       |11         |Name11|City11|2025-03-01 18:55:51.878449|9999-12-31 00:00:00       |
+---------+-----------+------+------+--------------------------+--------------------------+
```

Screenshot 6: The output of the execution