GA-5

Report: Implementation of SCD Type II using SparkSQL

Objective

The objective of this assignment is to implement Slowly Changing Dimension (SCD) Type II on a customer master data frame using SparkSQL, ensuring that historical changes are tracked and maintained in the data.

Input Data

Original Data (original.csv):

```
Index,Name,DOB,validity_start,validity_end
1,Jenny,16-04-2001,01-01-1970,31-12-9999
2,James,21-07-2002,01-01-1970,31-12-9999
3,Jacob,06-11-2001,01-01-1970,31-12-9999
```

Updated Data (updated.csv):

```
Name,updated_DOB
Jenny,17-09-2009
Nancy,13-02-2003
```

Expected Output

The output should reflect the changes in the `updated.csv` while maintaining historical records from the `original.csv`. The resulting data should look like this:

```
+-----+-----+----------+--------------+------------+
|Index| Name|       DOB|validity_start|validity_end|
+-----+-----+----------+--------------+------------+
|    1|Jenny|16-04-2001|    01-01-1970|  21-07-2024|
|    2|James|21-07-2002|    01-01-1970|  31-12-9999|
|    3|Jacob|06-11-2001|    01-01-1970|  31-12-9999|
|    4|Nancy|13-02-2003|    21-07-2024|  31-12-9999|
|    5|Jenny|17-09-2009|    21-07-2024|  31-12-9999|
+-----+-----+----------+--------------+------------+
```

Implementation Details

The implementation involves the following steps:

1. **Create Spark Session**: Initialize a Spark session to facilitate the processing of the data.
2. **Read CSV Files**: Read the original and updated data from CSV files into Spark DataFrames.
3. **Create Temporary Views**: Create temporary SQL views for these DataFrames to allow SQL queries.
4. **Generate SQL Query**: Create an SQL query to handle the SCD Type II logic.
5. **Execute SQL Query**: Execute the generated SQL query and display the results.

Here is the code used for the implementation:

```
from datetime import datetime
from pyspark.sql import SparkSession

# Helpers
def create_spark_session(app_name):
    return SparkSession.builder.appName(app_name).getOrCreate()

def read_csv(spark, path):
    return spark.read.csv(path, header=True, inferSchema=True)

def get_current_date():
    return datetime.now().strftime("%d-%m-%Y")

# Core
def create_scd_type_2_query(current_date, end_date, format):
    return f"""
    SELECT ROW_NUMBER() OVER (ORDER BY validity_start) as Index, *
    FROM (
        SELECT
            master.Name,
            master.updated_DOB AS DOB,
            master.validity_start,
            '{current_date}' as validity_end
        FROM master_data master
        INNER JOIN update_data update
            ON (master.Name = update.Name)
            AND (master.updated_DOB != update.updated_DOB)
```

```
        WHERE to_date(master.validity_end, '{format}') >
to_date('{current_date}', '{format}')

        UNION

        SELECT
            master.Name,
            master.updated_DOB AS DOB,
            master.validity_start,
            master.validity_end
        FROM master_data master
        LEFT JOIN update_data update
            ON update.Name = master.Name
        WHERE update.Name is NULL

        UNION

        SELECT
            update.Name,
            update.updated_DOB AS DOB,
            '{current_date}' as validity_start,
            '{end_date}' as validity_end
        FROM update_data update
    )
    """

# Runner
def main():
    spark = create_spark_session("SCD_Type_2")

    original = read_csv(spark, "gs://ibd-ga5/original.csv")
    original.createOrReplaceTempView("master_data")

    updated = read_csv(spark, "gs://ibd-ga5/updated.csv")
    updated.createOrReplaceTempView("update_data")

    current_date = get_current_date()
```

```
    sql_query = create_scd_type_2_query(current_date, "31-12-9999",
"dd-MM-yyyy")

    updated_data = spark.sql(sql_query)
    updated_data.show()

    spark.stop()

if __name__ == "__main__":
    main()
```

Explanation

1. **Read CSV Files**: The original and updated data are read from the provided Google Cloud Storage paths and converted into DataFrames.
2. **Create Temporary Views**: These DataFrames are registered as temporary SQL views (`master_data` and `update_data`), allowing SQL operations.
3. **Generate SQL Query**: The `create_scd_type_2_query` function constructs an SQL query with the following logic:
    - Selects records from the master where a corresponding update exists but with different `DOB` values, and sets the `validity_end` to the current date.
    - Selects records from the master where no corresponding update exists, retaining the original `validity_end`.
    - Selects new records from the update, setting the `validity_start` to the current date and `validity_end` to a future date.
4. **Execute SQL Query**: The query is executed to generate the final DataFrame, which is then displayed.

Conclusion

The SCD Type II implementation ensures that changes in the updated data are reflected while maintaining historical records from the original data. The provided solution is executed on a Dataproc cluster, showcasing the ability to handle large datasets efficiently with SparkSQL.