Here is a prompt you could use to generate the Snowpark Python code you provided. It's designed to be detailed and comprehensive, ensuring all the key logic is included.

**Snowpark Bi-Temporal Merge Prompt**

Please write a Python function main(session: snowpark.Session) for a Snowflake Snowpark Python Worksheet. The function should demonstrate a full **bi-temporal merge process**, taking a historical dataset and a new delta dataset, and applying the changes to produce a unified, fully historical view.

**Requirements:**

1. **Define Schemas and Sample Data:**
   * Create two StructType schemas: history\_schema and delta\_schema.
   * The history\_schema should include bi-temporal columns like START\_TMS, END\_TMS, LAST\_CHG\_TMS, TXN\_START\_TMS, TXN\_END\_TMS, IS\_CURRENT, and IS\_LATEST\_TXN.
   * The delta\_schema should be a subset of the history schema, containing only the business key and the new/corrected data.
   * Create sample history\_data and delta\_data lists that contain various scenarios, including:
     + **New versions:** A new START\_TMS for an existing asset.
     + **Corrections:** A new LAST\_CHG\_TMS for an existing START\_TMS.
     + **New assets:** An ASSET\_ID that does not exist in the history.
     + **Deletions:** Use specific sentinel values ($$DELETED$$ for strings, a special number for decimals, and datetime(1, 1, 1) for dates/timestamps) to indicate a field should be nullified.
2. **Core Merge Logic:**
   * **Unify and Propagate:**
     + Combine the history and delta dataframes into a single event stream using union\_by\_name.
     + Use a Window function with last\_value(..., ignore\_nulls=True) to perform a **Last Observation Carried Forward (LOCF)**. This should be applied to all data columns to fill in missing values chronologically based on START\_TMS and LAST\_CHG\_TMS.
   * **Filter Non-Changing Records:**
     + After propagation, filter the event stream to keep only the records that represent an actual change in the data. A record should be kept if any of its data columns (ASSET\_PRICE, etc.) have a different value than the previous record for the same START\_TMS.
   * **Reconstruct Bi-Temporal Axes:**
     + Rebuild the transaction-time axis (TXN\_START\_TMS, TXN\_END\_TMS, IS\_LATEST\_TXN) by partitioning by the key columns and START\_TMS, and ordering by LAST\_CHG\_TMS.
     + Rebuild the valid-time axis (START\_TMS, END\_TMS, IS\_CURRENT) by partitioning by the key columns and ordering by START\_TMS.
   * **Clean Up Deletion Sentinels:**
     + After the final join, replace the sentinel values from the delta data ($$DELETED$$, etc.) with NULL to properly represent a deletion.
3. **Final Output:**
   * The function should return the final dataframe, sorted by ASSET\_ID, DATA\_PROVIDER, START\_TMS, and TXN\_START\_TMS.

This code needs to do the following.

* If we receive a delta record, pull all the history table records for that asset\_id and data provider.
* Find if there are any history records with the same START\_TMS for that asset\_id and data\_provider.
  + If yes, then end the current matching history record with TXN\_END\_TMS = LAST\_CHG\_TMS of the delta record minus 1 microsecond and IS\_LATEST\_TXN = FALSE. Also, insert a new row for that asset\_id, data\_provider, start\_tms with TXN\_START\_TMS = LAST\_CHG\_TMS of the delta record and IS\_LATEST\_TXN = TRUE. Apart from the data columns which have non-null values in the delta record, for remaining data columns, use the values from the existing history record we just updated with the TXN\_END\_TMS
    - If there are contiguous future dated rows (based on START\_TMS) for the same asset\_id and data\_provider, and they had the same value as the current row (based on DELTA record START\_TMS) that got updated, then the updated value for those columns from the DELTA record should be propagated to those rows too. Instead of updating them directly, we need to use the same logic of updating TXN\_END\_TMS = LAST\_CHG\_TMS of the delta record minus 1 microsecond and IS\_LATEST\_TXN = FALSE and also inserting a clone of this row with TXN\_START\_TMS = LAST\_CHG\_TMS of the delta record and IS\_LATEST\_TXN = TRUE with the updated value in those impacted columns based on the delta record.
  + If no,
    - Find if there are any past-dated rows for that asset\_id and data\_provider.
      * If yes, then check if there are any future dated rows for that asset\_id and data\_provider.
        + If no, then insert a new record with START\_TMS = START\_TMS from DELTA record, END\_TMS = NULL, TXN\_START\_TMS = LAST\_CHG\_TMS of the delta record and IS\_LATEST\_TXN = TRUE. Apart from the data columns which have non-null values in the delta record, for remaining data columns, use the values from the most recent past dated record.
        + If yes, then

update the most recent past dated record with TXN\_END\_TMS = LAST\_CHG\_TMS of the delta record minus 1 microsecond. Before updating this row, clone it and insert a new row with END\_TMS = LAST\_CHG\_TMS of the delta record minus 1 microsecond, TXN\_START\_TMS = LAST\_CHG\_TMS of the delta record and IS\_LATEST\_TXN = TRUE.

Insert a new record with START\_TMS = START\_TMS from DELTA record, END\_TMS = START\_TMS of the next immediate record in the history for the same ASSET\_ID and DATA\_PROVIDER minus 1 microsecond, TXN\_START\_TMS = LAST\_CHG\_TMS of the delta record and IS\_LATEST\_TXN = TRUE. Apart from the data columns which have non-null values in the delta record, for remaining data columns, use the values from the most recent past dated record.

* + - * If no, then check if there are any future dated records for that asset\_id and data\_provider.
        + If yes, then skip the delta record since we don’t want to insert history records prior to the existing history records
        + If no (i.e., no past or future dated records for the same asset\_id and data\_provider), then insert a new record with START\_TMS = START\_TMS from DELTA record, END\_TMS = NULL, TXN\_START\_TMS = LAST\_CHG\_TMS of the delta record and IS\_LATEST\_TXN = TRUE.