Ingestion Utility Streamlit App

Overview :

This application is a utility for viewing and managing data in specific manual dimension tables within Snowflake. It supports updating ,inserting & deleting records, focusing on maintaining data history (**Slowly Changing Dimension - SCD Type 2**).

Key Features :

1.App Startup and Initial View

- **Select Table:** On the left sidebar, use the "**Select the Tables**" dropdown to choose one of the manual dimension tables (e.g., DIM_MAP_BRANCH).
- **Preview Mode (Default):** The main screen loads into Preview Mode.
    - It displays the "**Top 5 Latest Active Rows**" for the selected table.
    - These rows represent the current, active version of the data.
- **Action:**
    - Click "**Edit**": Switches the application to Edit Mode, allowing you to modify the data.
    - Click "**Show full table history**": Expands the section to show all records for the table, including historical (expired) versions.

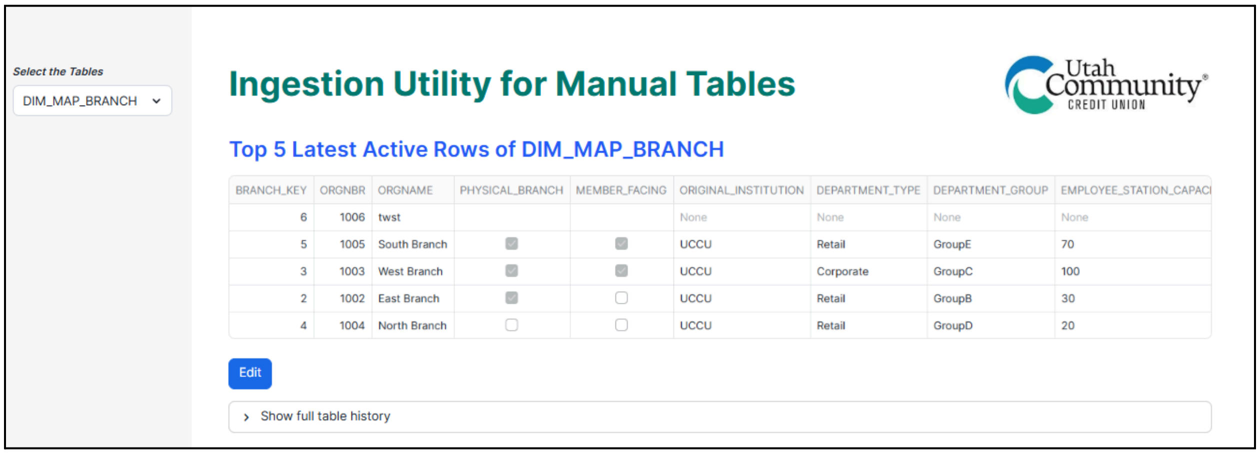
2.Edit Mode:

When you click "**Edit**," the view changes to allow data manipulation.

- **Data View:** The full set of active records is loaded into an editable table.
    - System columns (like dates, keys, and status flags) are hidden from editing.
- **Search:** Use the search bar to quickly filter records based on the table's Primary Key (e.g., searching by ORGNBR).
- **Data Editing:**
    - **Modify a Value:** Click on any cell in the editable columns to change its value.
    - **Add a New Record:** Scroll to the bottom and click on **+** in any editable columns at the last row to insert a new record.
    - **Mark for Deletion:** Check the box in the last column, "**DELETE**," for any record which needs to get removed.
- **Action**:
- Click "**Save Changes**": The app processes all changes:
    - **Deletions:** Records marked for delete are expired (their **IS_CURRENT** flag is set to **FALSE** and an **END_DATE** is applied).
    - **Updates:** The old version of the record is expired, and a new version with your changes is inserted (SCD Type 2).
    - **Insertions:** New records are inserted as the current active version.
    - Once saved, a success message appears, and the app automatically switches back to Preview Mode.

App UI

Preview Mode



Select the Tables

DIM_MAP_BRANCH

Ingestion Utility for Manual Tables

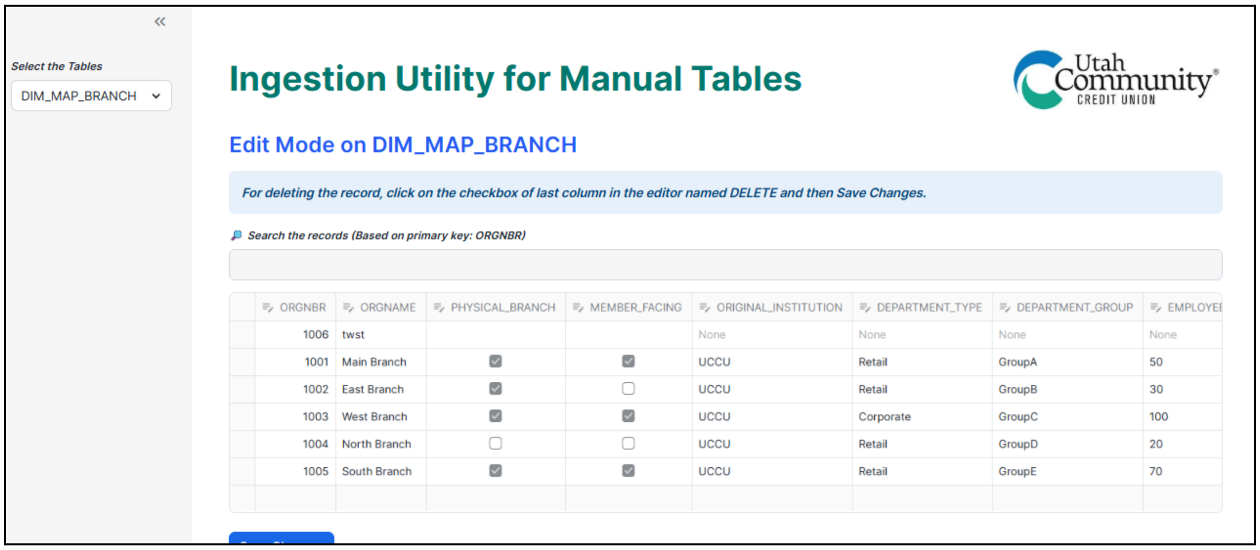
Top 5 Latest Active Rows of DIM_MAP_BRANCH

BRANCH_KEY	ORGNBR	ORGNAME	PHYSICAL_BRANCH	MEMBER_FACING	ORIGINAL_INSTITUTION	DEPARTMENT_TYPE	DEPARTMENT_GROUP	EMPLOYEE_STATION_CAPACI
6	1006	twst			None	None	None	None
5	1005	South Branch	✓	✓	UCCU	Retail	GroupE	70
3	1003	West Branch	✓	✓	UCCU	Corporate	GroupC	100
2	1002	East Branch	✓	☐	UCCU	Retail	GroupB	30
4	1004	North Branch	☐	☐	UCCU	Retail	GroupD	20

Edit

Show full table history

Edit Mode



Select the Tables

DIM_MAP_BRANCH

Ingestion Utility for Manual Tables

Edit Mode on DIM_MAP_BRANCH

For deleting the record, click on the checkbox of last column in the editor named DELETE and then Save Changes.

Search the records (Based on primary key: ORGNBR)

ORGNBR	ORGNAME	PHYSICAL_BRANCH	MEMBER_FACING	ORIGINAL_INSTITUTION	DEPARTMENT_TYPE	DEPARTMENT_GROUP	EMPLOYEE
1006	twst			None	None	None	None
1001	Main Branch	✓	✓	UCCU	Retail	GroupA	50
1002	East Branch	✓	☐	UCCU	Retail	GroupB	30
1003	West Branch	✓	✓	UCCU	Corporate	GroupC	100
1004	North Branch	☐	☐	UCCU	Retail	GroupD	20
1005	South Branch	✓	✓	UCCU	Retail	GroupE	70

Streamlit Code:

```python
import streamlit as st
from snowflake.snowpark.context import get_active_session
from snowflake.snowpark import functions as F
import hashlib
from datetime import date, datetime

# --- Get active Snowflake session ---
```

```
session = get_active_session()

# --- Custom page style (padding adjustments) ---
st.markdown(
            """
            <style>
                .block-container {
                    padding-top: 2.5rem;
                    # padding-bottom: 0rem;
                    padding-right: 3.5rem;
                    padding-left: 3rem;
                }
            </style>
            """, unsafe_allow_html=True
        )

# --- Set page layout to wide ---
st.set_page_config(layout="wide")
# --- Mapping of tables to their primary key columns ---
TABLE_PK_MAP = {
    "RAW.MANUAL.DIM_MAP_BRANCH": "ORGNBR",
    "RAW.MANUAL.DIM_MAP_MEMBERSHIP_CHANGE_REASON": "CHANGE_REASON",
    "RAW.MANUAL.DIM_MAP_MEMBERSHIP_TYPE": "MEMBERSHIP_TYPE",
    "RAW.MANUAL.DIM_MAP_PRODUCT": "MIACCTTYPCD",
    "RAW.MANUAL.DIM_MAP_PRODUCT_PURCHASED_AUTO": "MIACCTTYP"
}
# --- Mapping of tables to their surrogate key columns ---
SURROGATE_KEY_MAP = {
    "DIM_MAP_BRANCH": "BRANCH_KEY",
    "DIM_MAP_MEMBERSHIP_CHANGE_REASON": "MEMBERSHIP_CHANGE_REASON_KEY",
    "DIM_MAP_MEMBERSHIP_TYPE": "MEMBERSHIP_TYPE_KEY",
    "DIM_MAP_PRODUCT": "PRODUCT_KEY",
    "DIM_MAP_PRODUCT_PURCHASED_AUTO": "PRODUCT_PURCHASED_AUTOS_KEY"
}
# --- Columns excluded from MD5 hash calculation ---
EXCLUDE_COLS = {
    "EFFECTIVE_DATE", "EFFECTIVE_TIME", "END_DATE", "END_TIME",
    "IS_CURRENT", "CREATED_DATE", "CREATED_TIME",
    "UPDATED_DATE", "UPDATED_TIME", "IS_DELETED"
}
# --- UI: Select Table ---
DATABASE = "RAW"
SCHEMA = "MANUAL"
```

```
# --- Fetch all tables from schema dynamically ---
tables = session.sql(f"SHOW TABLES IN {DATABASE}.{SCHEMA}").collect()
table_names = [row["name"] for row in tables]

# --- Build TABLE_OPTIONS dict dynamically ---
# Label = table name (can be adjusted if you want pretty labels)
TABLE_OPTIONS = {name: f"{DATABASE}.{SCHEMA}.{name}" for name in table_names}

t1,t2=st.columns([7,2])
with t1:
    st.markdown(f'<h1 style="color:#01796F;">Ingestion Utility for Manual
Tables</h1>', unsafe_allow_html=True)
with t2:
    session.file.get("@RAW.MANUAL.LOGO/uccu.png","/tmp")
    st.image("/tmp/uccu.png",width=175,use_container_width=True)
# --- Sidebar: Table Selector ---
selected_label = st.sidebar.selectbox("***Select the Tables***",
list(TABLE_OPTIONS.keys()))
TABLE = TABLE_OPTIONS[selected_label]
# --- Get surrogate key column for selected table ---
table_name_short = TABLE.split('.')[-1]
SK_COL = SURROGATE_KEY_MAP.get(table_name_short)
# --- Helper: Get next surrogate key value ---
def get_next_key(table_name, sk_col):
    return session.sql(
        f"SELECT COALESCE(MAX({sk_col}),0)+1 AS NEXT_KEY FROM {table_name}"
    ).collect()[0][0]
# --- Helper: Compute MD5 hash column dynamically ---
def compute_md5(df, table_name):
    cols = [c.name for c in df.schema.fields]
    pk_col = TABLE_PK_MAP.get(table_name)
    hash_cols = [c for c in cols if c not in EXCLUDE_COLS and c != pk_col and c
!= SK_COL]
    if not hash_cols:
        return df.with_column("MD5_HASH_KEY", F.lit(None))
    concat_expr = F.col(hash_cols[0]).cast("string")
    for c in hash_cols[1:]:
        concat_expr = F.concat(concat_expr, F.lit("||"),
F.col(c).cast("string"))
    return df.with_column("MD5_HASH_KEY", F.md5(concat_expr))
# --- Load active (latest) data with MD5 ---
def load_active_data(table_name: str, for_edit: bool = False):

    df = session.table(table_name)
    cols = [c.name for c in df.schema.fields]
    df = compute_md5(df, table_name)

    if "IS_CURRENT" in cols:
        df = df.filter(F.col("IS_CURRENT") == True)

     # Sort by EFFECTIVE_DATE/TIME
    if "EFFECTIVE_DATE" in cols:
        df = df.sort(F.col("EFFECTIVE_DATE").desc(),
                     F.col("EFFECTIVE_TIME").desc())

    if not for_edit:
        # Limit rows in preview mode
        df = df.limit(5)

    return df.to_pandas(), cols
# --- Same loader but without MD5 ---
def load_active_data1(table_name: str, for_edit: bool = False):

    df = session.table(table_name)
    cols = [c.name for c in df.schema.fields]
    # df = compute_md5(df, table_name)

    if "IS_CURRENT" in cols:
        df = df.filter(F.col("IS_CURRENT") == True)

     # Sort by EFFECTIVE_DATE/TIME
    if "EFFECTIVE_DATE" in cols:
        df = df.sort(F.col("EFFECTIVE_DATE").desc(),
                     F.col("EFFECTIVE_TIME").desc())

    if not for_edit:
        # Limit rows in preview mode
        df = df.limit(5)

    return df.to_pandas(), cols
# --- Helper to safely format values for SQL ---
def format_value(v):
    if v is None:
        return "NULL"
    if isinstance(v, str):
        v_safe = v.replace("'", "''")
        return f"'{v_safe}'"
    if isinstance(v, bool):
        return "TRUE" if v else "FALSE"
    if isinstance(v, (date, datetime)):
        return f"DATE '{v:%Y-%m-%d}'"            # DATE '2018-11-01'
    return str(v)
# --- Initialize edit mode in session state ---
if "edit_mode" not in st.session_state:
    st.session_state.edit_mode = False
# --- Load preview data ---
try:
    data, cols = load_active_data(TABLE)
    display_count = len(data)
except Exception as e:
    st.warning('No SCD & Audit columns found.')
    st.stop()
# --- Preview Mode (default) ---
if not st.session_state.edit_mode:
    st.markdown(f'<h3 style="color:#275EF5;">Top {display_count} Latest Active
Rows of {selected_label} </h3>', unsafe_allow_html=True)
    # hist_df = session.table(TABLE)           # no column filter
    # st.dataframe(hist_df.to_pandas(), use_container_width=True)
    data, cols = load_active_data1(TABLE, for_edit=False)
    st.dataframe(data, use_container_width=True, hide_index=True)
    if st.button("Edit", type="primary"):
        st.session_state.edit_mode = True
        st.rerun()
else:
    # --- Edit Mode ---
    st.markdown(f'<h3 style="color:#275EF5;">Edit Mode on
{selected_label}</h3>', unsafe_allow_html=True)

    st.info('***For deleting the record, click on the checkbox of last column
in the editor named DELETE and then Save Changes.***')
    pk_col = TABLE_PK_MAP[TABLE]
    data, cols = load_active_data(TABLE, for_edit=True)
    # Allow editing only on non-system columns
    editable_cols = [c for c in cols if c not in EXCLUDE_COLS |
{"MD5_HASH_KEY", SK_COL}]
    filtered_data = data.copy()

    # --- Search filter (on Primary Key column) ---
    search_query = st.text_input(
    f":mag_right: ***Search the records (Based on primary key: {pk_col})***",
    ""
    )
    filtered_data = data
    if search_query.strip():
        filtered_data = data[
            data[pk_col].astype(str).str.contains(search_query, case=False,
na=False)
        ]

    if filtered_data.empty:
        st.info("No matching records found.")
        st.stop()

    # --- Editable DataFrame ---
    editable_cols = [c for c in cols if c not in EXCLUDE_COLS |
{"MD5_HASH_KEY", SK_COL}]
    data_editor_df = filtered_data[editable_cols].copy()
    data_editor_df["DELETE"] = False
    edited_df = st.data_editor(
        data_editor_df,
        num_rows="dynamic",
        use_container_width=True,
        hide_index=True,
    )

     # --- Save Changes Logic ---
    if st.button("Save Changes", type="primary"):
        for _, new_row in edited_df.iterrows():
            # identify PK column and value
            pk_col = TABLE_PK_MAP[TABLE]
            pk_val = new_row[pk_col]

            # find old row in original dataset
            old_row = data[data[pk_col] == pk_val]

            # insert/update/delete logic
            # --- Case 1: Delete ---
            if new_row.get("DELETE", False) and not old_row.empty:
                # if "IS_CURRENT" in cols:
                session.sql(f"""
                    UPDATE {TABLE}
                       SET IS_CURRENT = FALSE,
                           UPDATED_DATE   = CURRENT_DATE,
                           UPDATED_TIME   = CURRENT_TIME,
                           END_DATE   = CURRENT_DATE,
                           END_TIME   = CURRENT_TIME,
                           IS_DELETED = TRUE
                     WHERE {pk_col} = '{pk_val}'
                       AND IS_CURRENT = TRUE
                """).collect()
                continue
            # --- Case 2: Update existing row ---
            if not old_row.empty:
                old_values = old_row[editable_cols].iloc[0].to_dict()
                new_values = new_row[editable_cols].to_dict()
                if old_values != new_values:
                    if "IS_CURRENT" in cols:
                        # expire old row
                        b=(f"""
                            UPDATE {TABLE}
                               SET IS_CURRENT = FALSE,
                                   UPDATED_DATE   = CURRENT_DATE,
                                   UPDATED_TIME   = CURRENT_TIME,
                                   END_DATE   = CURRENT_DATE,
                                   END_TIME   = CURRENT_TIME
                             WHERE {pk_col} = '{pk_val}'
                               AND IS_CURRENT = TRUE
                        """)

                        session.sql(b).collect()

                        # build new version
                        sk_val   = get_next_key(TABLE, SK_COL) if SK_COL else
None
                        row_dict = new_row.drop(labels=["DELETE"],
errors="ignore").to_dict()

                        # MD5 without PK and without surrogate key
                        md5_cols  = [c for c in row_dict if c not in
EXCLUDE_COLS and c != SK_COL]
                        md5_input = "|".join([str(row_dict[c]) if row_dict[c]
is not None else ""
                                              for c in md5_cols])
                        md5_hash  = hashlib.md5(md5_input.encode()).hexdigest()

                        col_list = ([SK_COL] if SK_COL else []) +
list(row_dict.keys()) + \
["EFFECTIVE_DATE","EFFECTIVE_TIME","IS_CURRENT","CREATED_DATE","CREATED_TIME","
UPDATED_DATE","UPDATED_TIME","IS_DELETED","MD5_HASH_KEY"]
                        val_list = ([str(sk_val)] if SK_COL else []) + \
                                   [format_value(v) for v in row_dict.values()]
+ \

["CURRENT_DATE","CURRENT_TIME","TRUE","CURRENT_DATE","CURRENT_TIME","CURRENT_DA
TE","CURRENT_TIME","FALSE",f"'{md5_hash}'"]

                        session.sql(f"""
                            INSERT INTO {TABLE} ({','.join(col_list)})
                            VALUES ({','.join(val_list)})
                        """).collect()

                    else:
                        # simple in-place update
                        set_clause = ", ".join([f"{c} =
{format_value(new_values[c])}"
                                                for c in editable_cols])
                        md5_input  = "|".join([str(new_values[c]) if
new_values[c] is not None else ""
                                               for c in editable_cols])
                        md5_hash   =
hashlib.md5(md5_input.encode()).hexdigest()

                        session.sql(f"""
                            UPDATE {TABLE}
                               SET {set_clause},
                                   MD5_HASH_KEY = '{md5_hash}'
                             WHERE {pk_col} = '{pk_val}'
                        """).collect()
            # --- Case 3: New row ---
            else:
                sk_next  = get_next_key(TABLE, SK_COL) if SK_COL else "NULL"
                row_dict = new_row.drop(labels=["DELETE"],
errors="ignore").to_dict()

                # MD5 without PK and without surrogate key
                md5_cols  = [c for c in row_dict if c not in EXCLUDE_COLS and c
!= SK_COL]
                # st.write(md5_cols,"MD5 COLS")
                md5_input = "|".join([str(row_dict[c]) if row_dict[c] is not
None else ""
```

```
                                  for c in md5_cols])
                # st.write(md5_input,"MD5 INPUT")

                md5_hash  = hashlib.md5(md5_input.encode()).hexdigest()
                # st.write(md5_hash,"MD5 HASH")

                col_list = ([SK_COL] if SK_COL else []) + list(row_dict.keys())
+ \

["EFFECTIVE_DATE","EFFECTIVE_TIME","IS_CURRENT","CREATED_DATE","CREATED_TIME","
UPDATED_DATE","UPDATED_TIME","IS_DELETED","MD5_HASH_KEY"]
                val_list = ([str(sk_next)] if SK_COL else []) + \
                           [format_value(v) for v in row_dict.values()] + \

["CURRENT_DATE","CURRENT_TIME","TRUE","CURRENT_DATE","CURRENT_TIME","CURRENT_DA
TE","CURRENT_TIME","FALSE",f"'{md5_hash}'"]

                sql = f"INSERT INTO {TABLE} ({','.join(col_list)}) VALUES
({','.join(val_list)})"
                session.sql(sql).collect()
        st.success(f"Changes saved to {selected_label}!")
        st.session_state.edit_mode = False
        st.rerun()
# --- Expandable section: Show full table history ---
with st.expander("Show full table history", expanded=False):
    hist_df = session.table(TABLE)           # no column filter
    st.dataframe(hist_df.to_pandas(), use_container_width=True)
```