**STREAMLIT APP FOR UPDATING BUSINESS METADATA TABLE**

* This streamlit app is designed to create a Streamlit web application that allows users to update metadata for tables in a Snowflake database. The application provides an interactive interface for selecting databases, schemas, and tables, and then displays the metadata for the selected table in an editable format.

**Key Uses:**

1. **Interactive Metadata Management:**

* The application allows users to select a database, schema, and table from dropdown menus.
* It fetches and displays metadata for the selected table, including business terms, descriptions, data classification, data owner, data steward, approval status, and approved by.

1. **Editable Data Table:**

* The metadata is displayed in an editable table using Streamlit's data editor.
* Users can update the metadata directly within the web application.

1. **Submit Changes:**

* A submit button allows users to save their changes.
* When the submit button is clicked, the updated metadata is written back to the Snowflake database.

1. **Tag Management:**

* The application fetches tags assigned to columns and displays them in the metadata table.
* Tags are displayed but not editable within the application.

**Code Explanation:**

1. **Initialization:**

* The code imports necessary libraries (streamlit, snowflake.snowpark.context, and pandas).
* It gets the current Snowflake session using get\_active\_session().

1. Database, Schema and Table Selection:

* The application fetches available databases and displays them in a dropdown menu for selection.
* Based on the selected database, the application fetches and displays available schemas and tables in dropdown menus.

1. Metadata Fetching:

* The application fetches metadata for the selected table from the BUSINESS\_METADATA\_TABLE.
* It also fetches tags assigned to columns from the TAG\_REFERENCES table.

1. Data Display and Editing:

* The fetched metadata is displayed in an editable table.
* Users can update the metadata fields, except for tags which are displayed but not editable.

1. Submitting Changes:

* When the submit button is clicked, the updated metadata is written back to the BUSINESS\_METADATA\_TABLE in Snowflake.
* A success message is displayed upon successful update.

import streamlit as st

from snowflake.snowpark.context import get\_active\_session

import pandas as pd

# Get the current Snowflake session

session = get\_active\_session()

# Set page config to use wide layout

st.set\_page\_config(layout="wide")

# Streamlit App Title

st.title("Update Snowflake Metadata Table")

# Custom CSS to control table width and button positioning

st.markdown("""

<style>

/\* Adjust table width \*/

.stDataEditor div[data-testid="stVerticalBlock"] {

width: 90vw !important;

max-width: 90vw !important;

}

/\* Reduce spacing below table \*/

.stDataEditor {

margin-bottom: 0px !important;

}

/\* Keep Submit button close to table \*/

.submit-container {

display: flex;

justify-content: center;

margin-top: -10px; /\* Pull button closer \*/

}

</style>

""", unsafe\_allow\_html=True)

# Fetch available databases

database\_query = "SHOW DATABASES;"

databases = [row["name"] for row in session.sql(database\_query).collect()]

# Select database

selected\_database = st.selectbox(":open\_file\_folder: Select Database", databases, index=0)

# Fetch schemas from information schema

schema\_query = f"""

SELECT DISTINCT table\_schema

FROM {selected\_database}.information\_schema.tables

WHERE table\_catalog = '{selected\_database}'

AND table\_schema != 'INFORMATION\_SCHEMA';

"""

schemas = [row["TABLE\_SCHEMA"] for row in session.sql(schema\_query).collect()]

# Select schema

selected\_schema = st.selectbox(":open\_file\_folder: Select Schema", schemas, index=0)

# Fetch tables for selected schema

table\_query = f"""

SELECT DISTINCT table\_name

FROM {selected\_database}.information\_schema.tables

WHERE table\_schema = '{selected\_schema}'

AND table\_type='BASE TABLE';

"""

tables = [row["TABLE\_NAME"] for row in session.sql(table\_query).collect()]

# Select table

selected\_table = st.selectbox(":page\_facing\_up: Select Table", tables, index=0)

# Fetch metadata from business metadata table

if selected\_database and selected\_schema and selected\_table:

metadata\_query = f"""

SELECT database\_name, schema\_name, table\_name, column\_name,

BUSINESS\_TERM, DESCRIPTION, DATA\_CLASSIFICATION, DATA\_OWNER, DATA\_STEWARD,

APPROVAL\_STATUS, APPROVED\_BY

FROM DATADICTIONARY\_DEV.DEV\_DICTIONARY.BUSINESS\_METADATA\_TABLE

WHERE database\_name = '{selected\_database}'

AND schema\_name = '{selected\_schema}'

AND table\_name = '{selected\_table}';

"""

metadata\_result = session.sql(metadata\_query).collect()

# Fetch tags assigned to columns

tag\_query = f"""

SELECT column\_name, tag\_name

FROM SNOWFLAKE.ACCOUNT\_USAGE.TAG\_REFERENCES

WHERE object\_database = '{selected\_database}'

AND object\_schema = '{selected\_schema}'

AND object\_name = '{selected\_table}'

AND domain = 'COLUMN';

"""

tags\_result = session.sql(tag\_query).collect()

# Convert tags result into a dictionary {column\_name: tag\_name}

tag\_dict = {row["COLUMN\_NAME"]: row["TAG\_NAME"] for row in tags\_result}

if metadata\_result:

# Convert result to DataFrame

df = pd.DataFrame(metadata\_result)

df.fillna("", inplace=True) # Replace NULLs with empty strings

# Add the "TAGS" column based on fetched tags

df["TAGS"] = df["COLUMN\_NAME"].map(tag\_dict).fillna("-")

# Dropdown options

classification\_options = ["", "Public", "PII"]

owner\_options = ["", "capital-formation"]

steward\_options = ["", "capital-formation"]

approval\_options = ["", "Pending", "Approved", "Rejected"]

approver\_options = ["", "capital-formation"]

# Reorder DataFrame columns

df = df[[

"DATABASE\_NAME", "SCHEMA\_NAME", "TABLE\_NAME", "COLUMN\_NAME", "TAGS",

"BUSINESS\_TERM", "DESCRIPTION", "DATA\_CLASSIFICATION", "DATA\_OWNER",

"DATA\_STEWARD", "APPROVAL\_STATUS", "APPROVED\_BY"

]]

# Editable Table

edited\_df = st.data\_editor(

df,

column\_config={

"DATABASE\_NAME": st.column\_config.TextColumn("Database Name", disabled=True),

"SCHEMA\_NAME": st.column\_config.TextColumn("Schema Name", disabled=True),

"TABLE\_NAME": st.column\_config.TextColumn("Table Name", disabled=True),

"COLUMN\_NAME": st.column\_config.TextColumn("Column Name", disabled=True),

"BUSINESS\_TERM": st.column\_config.TextColumn("Business Term"),

"DESCRIPTION": st.column\_config.TextColumn("Description"),

"TAGS": st.column\_config.TextColumn("Tags", disabled=True), # Now non-editable

"DATA\_CLASSIFICATION": st.column\_config.SelectboxColumn("Data Classification", options=classification\_options),

"DATA\_OWNER": st.column\_config.SelectboxColumn("Data Owner", options=owner\_options),

"DATA\_STEWARD": st.column\_config.SelectboxColumn("Data Steward", options=steward\_options),

"APPROVAL\_STATUS": st.column\_config.SelectboxColumn("Approval Status", options=approval\_options),

"APPROVED\_BY": st.column\_config.SelectboxColumn("Approved By", options=approver\_options),

},

num\_rows="fixed",

use\_container\_width=True,

)

# Submit button directly below the table

st.markdown('<div class="submit-container">', unsafe\_allow\_html=True)

if st.button(":floppy\_disk: Submit Changes"):

updated\_rows = edited\_df.to\_dict(orient="records")

for row in updated\_rows:

update\_query = f"""

UPDATE DATADICTIONARY\_DEV.DEV\_DICTIONARY.BUSINESS\_METADATA\_TABLE

SET

BUSINESS\_TERM = CASE WHEN '{row['BUSINESS\_TERM']}' IN ('', 'None') THEN NULL ELSE '{row['BUSINESS\_TERM']}' END,

DESCRIPTION = CASE WHEN '{row['DESCRIPTION']}' IN ('', 'None') THEN NULL ELSE '{row['DESCRIPTION']}' END,

DATA\_CLASSIFICATION = CASE WHEN '{row['DATA\_CLASSIFICATION']}' IN ('', 'None') THEN NULL ELSE '{row['DATA\_CLASSIFICATION']}' END,

DATA\_OWNER = CASE WHEN '{row['DATA\_OWNER']}' IN ('', 'None') THEN NULL ELSE '{row['DATA\_OWNER']}' END,

DATA\_STEWARD = CASE WHEN '{row['DATA\_STEWARD']}' IN ('', 'None') THEN NULL ELSE '{row['DATA\_STEWARD']}' END,

APPROVAL\_STATUS = CASE WHEN '{row['APPROVAL\_STATUS']}' IN ('', 'None') THEN NULL ELSE '{row['APPROVAL\_STATUS']}' END,

APPROVED\_BY = CASE WHEN '{row['APPROVED\_BY']}' IN ('', 'None') THEN NULL ELSE '{row['APPROVED\_BY']}' END

WHERE database\_name = '{row['DATABASE\_NAME']}'

AND schema\_name = '{row['SCHEMA\_NAME']}'

AND table\_name = '{row['TABLE\_NAME']}'

AND column\_name = '{row['COLUMN\_NAME']}';

"""

session.sql(update\_query).collect()

st.success(":white\_check\_mark: Metadata updated successfully!")

st.rerun()

st.markdown('</div>', unsafe\_allow\_html=True)

else:

st.error(":x: No data found for the specified filters.")

else:

st.warning(":warning: Please select a database, schema, and table.")

