**REDUNDANT OBJECTS DOCUMENTATION**

The REDUNDANT\_OBJECTS stored procedure is designed to identify and document database objects (tables and views) that have not been accessed for a specified period. This procedure helps in maintaining an up-to-date data dictionary by flagging potentially redundant objects. It also sends an email notification upon successful execution, ensuring that relevant stakeholders are informed.

**Here's a step-by-step explanation :**

**Procedure Definition:**

* The procedure is defined with the name REDUNDANT\_OBJECTS in the DATADICTIONARY\_PROD.PROD\_DICTIONARY schema.
* It takes two parameters: DESTINATIONTABLE (a string) and TIMEDELTA (an integer).
* It returns a string and is written in Python 3.10 using the snowflake-snowpark-python and requests packages.

**Imports and Setup:**

* The procedure imports necessary modules: datetime and snowflake.snowpark.
* It defines the main function which takes a Snowflake session, a destination table name, and a time delta as arguments.

**Email Notification Setup:**

* The email integration name is set to email\_integration.
* The recipient email address is set to [vamsi.s.krishna@kipi.ai](mailto:vamsi.s.krishna@kipi.ai).

**Query to Fetch Access History:**

* A SQL query is constructed to fetch access history from the SNOWFLAKE.ACCOUNT\_USAGE.ACCESS\_HISTORY table.
* The query flattens the BASE\_OBJECTS\_ACCESSED and DIRECT\_OBJECTS\_ACCESSED arrays to extract object details.
* It filters for tables and views that have not been accessed for a specified number of days (TIMEDELTA).

**DataFrame Creation:**

* The query result is loaded into a DataFrame.
* A new column LOAD\_TIMESTAMP is added to the DataFrame with the current timestamp.

**Check if Destination Table Exists:**

* A query checks if the destination table exists in the SNOWFLAKE.INFORMATION\_SCHEMA.TABLES.
* If the table does not exist, the DataFrame is saved as a new table with the name specified in DESTINATIONTABLE.
* If the table exists, it checks if data for the current date already exists in the table.
* If data for the current date exists, it deletes the existing data for the current date.
* The DataFrame is then appended to the existing table.

**Send Email Notification:**

* An email notification is sent using the SYSTEM$SEND\_EMAIL function.
* The email informs the recipient that the object's accessed table has been successfully created or updated.

**Return Success Message:**

* The procedure returns the string "Success" upon completion.

**Below is the code which is used to create stored procedure in snowflake:**

CREATE OR REPLACE PROCEDURE DATADICTIONARY\_PROD.PROD\_DICTIONARY.REDUNDANT\_OBJECTS("DESTINATIONTABLE" VARCHAR, "TIMEDELTA" NUMBER(38,0))

RETURNS VARCHAR

LANGUAGE PYTHON

RUNTIME\_VERSION = '3.10'

PACKAGES = ('snowflake-snowpark-python','requests')

HANDLER = 'main'

EXECUTE AS OWNER

AS '

import datetime

import snowflake.snowpark as snowpark

from snowflake.snowpark.functions import lit

def main(session: snowpark.Session, destinationTable: str, timeDelta: int):

notifIntegration = "email\_integration"

sendTo = ["vamsi.s.krishna@kipi.ai"] # Fixed email syntax

query = f"""

WITH ACCESS\_HISTORY AS (

SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.ACCESS\_HISTORY

),

ACCESS\_HISTORY\_FLATTENED AS (

SELECT

ACCESS\_HISTORY.QUERY\_ID,

ACCESS\_HISTORY.QUERY\_START\_TIME,

ACCESS\_HISTORY.USER\_NAME,

OBJECTS\_ACCESSED.VALUE:objectId::INTEGER AS TABLE\_ID,

OBJECTS\_ACCESSED.VALUE:objectName::TEXT AS OBJECT\_NAME,

OBJECTS\_ACCESSED.VALUE:objectDomain::TEXT AS OBJECT\_DOMAIN,

OBJECTS\_ACCESSED.VALUE:columns AS COLUMNS\_ARRAY

FROM ACCESS\_HISTORY,

LATERAL FLATTEN(ACCESS\_HISTORY.BASE\_OBJECTS\_ACCESSED) AS OBJECTS\_ACCESSED

-- WHERE

-- SPLIT\_PART(OBJECT\_NAME, ''.'', 1) <> ''DEV\_GOOGLE\_ANALYTICS''

-- AND SPLIT\_PART(OBJECT\_NAME, ''.'', 3) NOT ILIKE ''%SNOWPARK\_TEMP\_TABLE%''

UNION

SELECT

ACCESS\_HISTORY.QUERY\_ID,

ACCESS\_HISTORY.QUERY\_START\_TIME,

ACCESS\_HISTORY.USER\_NAME,

OBJECTS\_ACCESSED.VALUE:objectId::INTEGER AS TABLE\_ID,

OBJECTS\_ACCESSED.VALUE:objectName::TEXT AS OBJECT\_NAME,

OBJECTS\_ACCESSED.VALUE:objectDomain::TEXT AS OBJECT\_DOMAIN,

OBJECTS\_ACCESSED.VALUE:columns AS COLUMNS\_ARRAY

FROM ACCESS\_HISTORY,

LATERAL FLATTEN(ACCESS\_HISTORY.DIRECT\_OBJECTS\_ACCESSED) AS OBJECTS\_ACCESSED

-- WHERE

-- SPLIT\_PART(OBJECT\_NAME, ''.'', 1) <> ''DEV\_GOOGLE\_ANALYTICS''

-- AND SPLIT\_PART(OBJECT\_NAME, ''.'', 3) NOT ILIKE ''%SNOWPARK\_TEMP\_TABLE%''

),

TABLE\_ACCESS\_HISTORY AS (

SELECT

QUERY\_ID,

QUERY\_START\_TIME,

USER\_NAME,

OBJECT\_NAME AS FULLY\_QUALIFIED\_TABLE\_NAME,

AHF.TABLE\_ID

FROM ACCESS\_HISTORY\_FLATTENED AHF

WHERE

OBJECT\_DOMAIN IN (''Table'', ''View'')

AND AHF.TABLE\_ID IS NOT NULL

),

TABLE\_ACCESS\_SUMMARY AS (

SELECT

TH.TABLE\_ID,

MAX(FULLY\_QUALIFIED\_TABLE\_NAME) AS FULLY\_QUALIFIED\_TABLE\_NAME,

MAX\_BY(USER\_NAME, QUERY\_START\_TIME) AS LAST\_ACCESSED\_BY,

MAX(QUERY\_START\_TIME) AS LAST\_ACCESSED\_AT,

MAX\_BY(QUERY\_ID, QUERY\_START\_TIME) AS LAST\_QUERY\_ID

FROM TABLE\_ACCESS\_HISTORY TH

GROUP BY 1

),

TABLE\_ACCESS\_SUMMARY\_WITHOUT\_DELETED AS (

SELECT

TS.TABLE\_ID,

FULLY\_QUALIFIED\_TABLE\_NAME,

LAST\_ACCESSED\_BY,

LAST\_ACCESSED\_AT,

LAST\_QUERY\_ID

FROM TABLE\_ACCESS\_SUMMARY TS

RIGHT JOIN SNOWFLAKE.ACCOUNT\_USAGE.TABLES AT

ON TS.TABLE\_ID = AT.TABLE\_ID

WHERE

LAST\_ACCESSED\_AT < (CURRENT\_DATE - {timeDelta})

AND AT.DELETED IS NULL

ORDER BY FULLY\_QUALIFIED\_TABLE\_NAME

)

SELECT DISTINCT

TA.TABLE\_ID,

TA.FULLY\_QUALIFIED\_TABLE\_NAME,

TA.LAST\_ACCESSED\_BY,

TA.LAST\_ACCESSED\_AT,

TA.LAST\_QUERY\_ID

FROM TABLE\_ACCESS\_SUMMARY\_WITHOUT\_DELETED TA

INNER JOIN SNOWFLAKE.ACCOUNT\_USAGE.SCHEMATA S

ON SPLIT\_PART(FULLY\_QUALIFIED\_TABLE\_NAME, ''.'', 2)::VARCHAR = S.SCHEMA\_NAME

INNER JOIN SNOWFLAKE.ACCOUNT\_USAGE.TABLES T

ON SPLIT\_PART(FULLY\_QUALIFIED\_TABLE\_NAME, ''.'', 3)::VARCHAR = T.TABLE\_NAME

WHERE

S.DELETED IS NULL

AND T.DELETED IS NULL

"""

df = session.sql(query)

df = df.with\_column("LOAD\_TIMESTAMP", lit(datetime.datetime.now()))

# Check if the destination table exists

table\_check = session.sql(f"""

SELECT TABLE\_CATALOG || ''.'' || TABLE\_SCHEMA || ''.'' || TABLE\_NAME AS FULL\_TABLE\_NAME

FROM SNOWFLAKE.INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_CATALOG || ''.'' || TABLE\_SCHEMA || ''.'' || TABLE\_NAME ILIKE ''{destinationTable}''

""").collect()

if not table\_check:

df.write.save\_as\_table(destinationTable, mode="overwrite") # :white\_tick: Corrected write method

else:

query = f"""

SELECT DATE(MAX(LOAD\_TIMESTAMP))::VARCHAR AS DATE

FROM {destinationTable}

"""

result = session.sql(query).collect()[0].as\_dict()

if result[''DATE''] == datetime.date.today().strftime("%Y-%m-%d"):

session.sql(f"DELETE FROM {destinationTable} WHERE DATE(LOAD\_TIMESTAMP) = CURRENT\_DATE").collect()

df.write.save\_as\_table(destinationTable, mode="append") # :white\_tick: Corrected write method

# Send email notification

emailNotif = f"""

CALL SYSTEM$SEND\_EMAIL(

''{notifIntegration}'',

''{", ".join(sendTo)}'',

''SUCCESS: Objects Accessed Table Created'',

''Successfully added objects that have not been accessed for {timeDelta} days in {destinationTable} table''

)

"""

session.sql(emailNotif).collect()

return "Success"

';

