**Snowflake Metadata Share Guide**



|  |  |
| --- | --- |
| Product Version | 4.8.0 |
| Document Type | Snowshare Guide |
| Authors | Snowflake Data source Team |
| Reviewer | Red Team & Architects |
| Approver | CTO |
| Total Pages | 10 |
| Document Status | Draft |

Table Of Contents

[1.1 Objectives 2](#_Toc141993283)

[1.2 Architecture 2](#_Toc141993284)

[1.3 Pre-requisite 2](#_Toc141993285)

[1.4 Download Snowflake Metadata for health check. 3](#_Toc141993286)

[1.5 Output Files 5](#_Toc141993287)

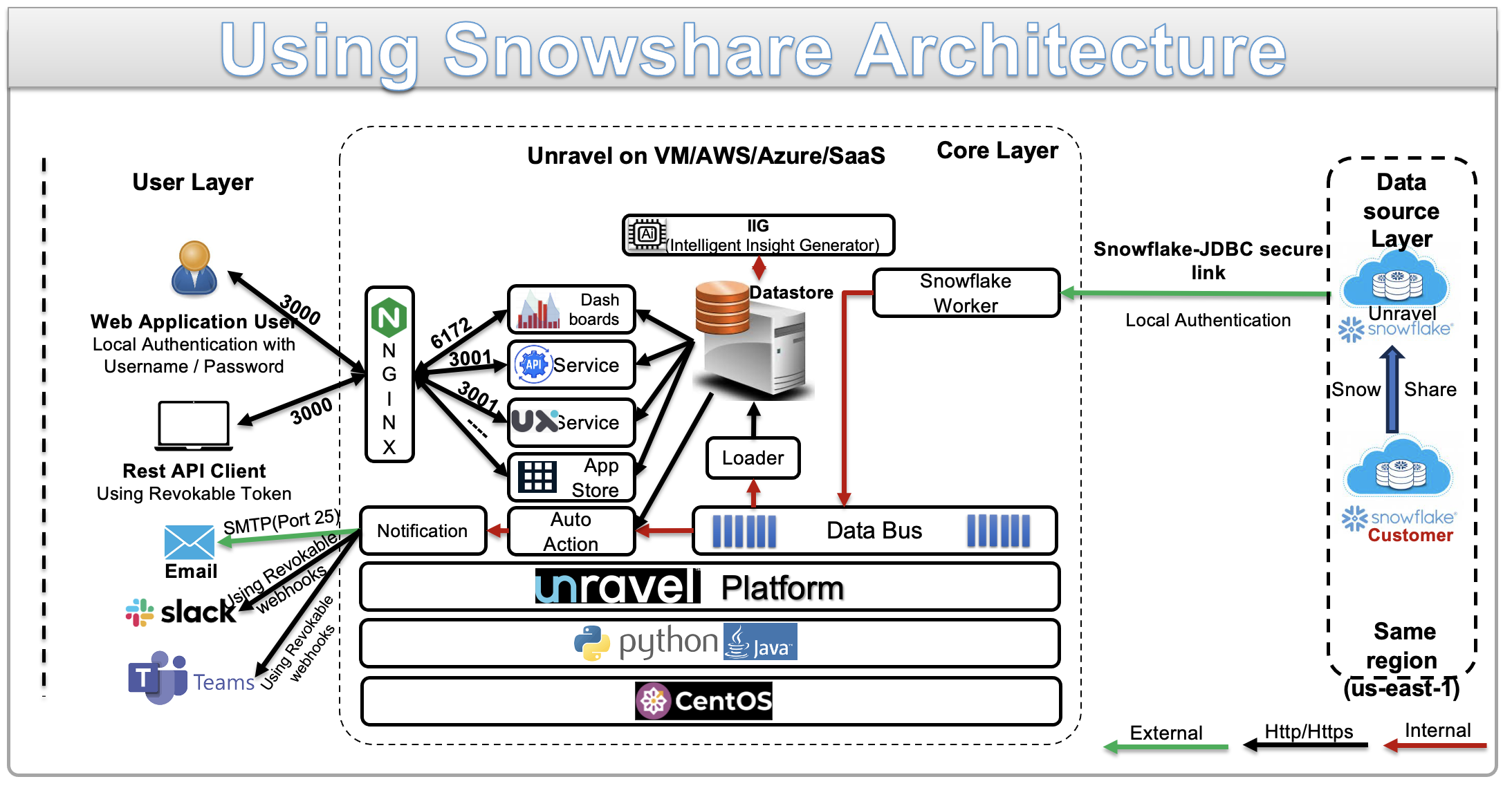
Document Version Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version # | Author | Remarks / Reason |
| 02-May-23 | 1.0 | Dev Team | New Document |

## Objectives

Health check download for snowflake unravel product.

## Architecture



## Pre-requisite

1. **Snowflake account Access through snowshare**
   * 1. Create a unravel user in snowflake
     2. Grant select for unravel user on schema SNOWFLAKE.ACCOUNT\_USAGE.
     3. Create a database unraveldb and a schema unravelschema to create a stored procedure required for metadata collection from account\_usage, information\_schema & Profile and store in local schema
     4. Execute procedure to create tables and serverless task to populate the metadata from the account metadata.
     5. Need to have Snowshare access to share the local schema to unravel snowflake account , Execute the snowshare script to give access to ”Unravel Snowflake account”.
     6. Based on the latency required the serverless/warehouse task will cost.

## Share Snowflake Metadata with Snowflake secure share.

Execute below statement to create the procedure and necessary functions.

CREATE DATABASE IF NOT EXISTS UNRAVEL\_SHARE;  
  
USE UNRAVEL\_SHARE;  
  
CREATE SCHEMA IF NOT EXISTS SCHEMA\_4823;  
  
USE UNRAVEL\_SHARE.SCHEMA\_4823;  
  
  
CREATE OR REPLACE TABLE replication\_log (  
 eventDate DATE DEFAULT current\_date,  
 executionStatus VARCHAR(1000) DEFAULT NULL,  
 remarks VARCHAR(1000)  
);  
  
CREATE OR REPLACE PROCEDURE REPLICATE\_ACCOUNT\_USAGE(DB STRING, SCHEMA STRING, LOOK\_BACK\_DAYS INTEGER)  
RETURNS STRING NOT NULL  
LANGUAGE SQL  
EXECUTE AS CALLER  
AS  
DECLARE  
use\_statement VARCHAR;  
res RESULTSET;  
BEGIN  
  
use\_statement := 'USE ' || DB || '.' || SCHEMA;  
res := (EXECUTE IMMEDIATE :use\_statement);  
  
DROP TABLE IF EXISTS WAREHOUSE\_METERING\_HISTORY ;  
CREATE TRANSIENT TABLE WAREHOUSE\_METERING\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.WAREHOUSE\_METERING\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date) ;  
  
DROP TABLE IF EXISTS WAREHOUSE\_EVENTS\_HISTORY ;  
CREATE TRANSIENT TABLE WAREHOUSE\_EVENTS\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.WAREHOUSE\_EVENTS\_HISTORY HIS WHERE HIS.TIMESTAMP > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date) ;  
  
DROP TABLE IF EXISTS WAREHOUSE\_LOAD\_HISTORY ;  
CREATE TRANSIENT TABLE WAREHOUSE\_LOAD\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.WAREHOUSE\_LOAD\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS TABLES ;  
CREATE TRANSIENT TABLE TABLES WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.TABLES;  
  
DROP TABLE IF EXISTS METERING\_DAILY\_HISTORY ;  
CREATE TRANSIENT TABLE METERING\_DAILY\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.METERING\_DAILY\_HISTORY HIS WHERE HIS.USAGE\_DATE > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS METERING\_HISTORY ;  
CREATE TRANSIENT TABLE METERING\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.METERING\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS DATABASE\_REPLICATION\_USAGE\_HISTORY ;  
CREATE TRANSIENT TABLE DATABASE\_REPLICATION\_USAGE\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.DATABASE\_REPLICATION\_USAGE\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS REPLICATION\_GROUP\_USAGE\_HISTORY ;  
CREATE TRANSIENT TABLE REPLICATION\_GROUP\_USAGE\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.REPLICATION\_GROUP\_USAGE\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS DATABASE\_STORAGE\_USAGE\_HISTORY ;  
CREATE TRANSIENT TABLE DATABASE\_STORAGE\_USAGE\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.DATABASE\_STORAGE\_USAGE\_HISTORY HIS WHERE HIS.USAGE\_DATE > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS STAGE\_STORAGE\_USAGE\_HISTORY ;  
CREATE TRANSIENT TABLE STAGE\_STORAGE\_USAGE\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.STAGE\_STORAGE\_USAGE\_HISTORY HIS WHERE HIS.USAGE\_DATE > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS SEARCH\_OPTIMIZATION\_HISTORY ;  
CREATE TRANSIENT TABLE SEARCH\_OPTIMIZATION\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.SEARCH\_OPTIMIZATION\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS DATA\_TRANSFER\_HISTORY ;  
CREATE TRANSIENT TABLE DATA\_TRANSFER\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.DATA\_TRANSFER\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS AUTOMATIC\_CLUSTERING\_HISTORY ;  
CREATE TRANSIENT TABLE AUTOMATIC\_CLUSTERING\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.AUTOMATIC\_CLUSTERING\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS SNOWPIPE\_STREAMING\_FILE\_MIGRATION\_HISTORY ;  
CREATE TRANSIENT TABLE SNOWPIPE\_STREAMING\_FILE\_MIGRATION\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.SNOWPIPE\_STREAMING\_FILE\_MIGRATION\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS TAG\_REFERENCES ;  
CREATE TRANSIENT TABLE TAG\_REFERENCES WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.TAG\_REFERENCES ;  
  
  
DROP TABLE IF EXISTS QUERY\_HISTORY ;  
CREATE TRANSIENT TABLE QUERY\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY HIS WHERE HIS.START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
DROP TABLE IF EXISTS ACCESS\_HISTORY ;  
CREATE TRANSIENT TABLE ACCESS\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.ACCESS\_HISTORY HIS WHERE HIS.QUERY\_START\_TIME > DATEADD(Day ,-:LOOK\_BACK\_DAYS, current\_date);  
  
RETURN 'SUCCESS';  
  
END;  
  
CREATE OR REPLACE PROCEDURE REPLICATE\_REALTIME\_QUERY(DB STRING, SCHEMA STRING, LOOK\_BACK\_HOURS INTEGER)  
RETURNS STRING NOT NULL  
LANGUAGE SQL  
EXECUTE AS CALLER  
AS  
DECLARE  
use\_statement VARCHAR;  
res RESULTSET;  
BEGIN  
  
use\_statement := 'USE ' || DB || '.' || SCHEMA;  
res := (EXECUTE IMMEDIATE :use\_statement);  
  
DROP TABLE IF EXISTS IS\_QUERY\_HISTORY ;  
CREATE TRANSIENT TABLE IS\_QUERY\_HISTORY WITH DATA\_RETENTION\_TIME\_IN\_DAYS=0 AS SELECT \* FROM TABLE(INFORMATION\_SCHEMA.QUERY\_HISTORY(dateadd('hours',-:LOOK\_BACK\_HOURS ,current\_timestamp()),current\_timestamp(),10000)) order by start\_time ;  
  
  
RETURN 'SUCCESS';  
  
END;  
  
  
  
  
CREATE OR REPLACE PROCEDURE create\_query\_profile(dbname string, schemaname string)  
 returns VARCHAR(25200)  
 LANGUAGE javascript  
  
AS  
$$  
  
function logError(err)  
{  
 var fail\_sql = "INSERT INTO REPLICATION\_LOG VALUES (current\_date,'FAIL', " + "'"+ err +"'"+");" ;  
 sql\_command1 = snowflake.createStatement({sqlText: fail\_sql} );  
 sql\_command1.execute();  
}  
  
try  
{  
 var query = 'CREATE DATABASE IF NOT EXISTS ' + DBNAME + ';';  
 var stmt = snowflake.createStatement({sqlText:query})  
 stmt.execute();  
 result = "Database: " + DBNAME + " creation is success";  
}  
catch (err)  
{  
 logError(err)  
 return "Failed to create DB " + DBNAME + ", error: " + err;  
}  
  
try  
{  
 var query = 'CREATE SCHEMA IF NOT EXISTS ' + DBNAME + '.' + SCHEMANAME + ';';  
 var stmt = snowflake.createStatement({sqlText:query})  
 stmt.execute();  
 result += "\nSchema: " + SCHEMANAME + " creation is success";  
}  
catch (err)  
{ logError(err)  
 return "Failed to create the schema "+ SCHEMANAME + ", error: " + err;  
}  
  
var schemaName = SCHEMANAME;  
var dbName = DBNAME;  
  
const queries = [];  
queries[0] = 'CREATE TRANSIENT TABLE IF NOT EXISTS ' + dbName + '.' + schemaName + '.QUERY\_PROFILE (QUERY\_ID VARCHAR(16777216),STEP\_ID NUMBER(38, 0),OPERATOR\_ID NUMBER(38,0),PARENT\_OPERATORS ARRAY, OPERATOR\_TYPE VARCHAR(16777216),OPERATOR\_STATISTICS VARIANT,EXECUTION\_TIME\_BREAKDOWN VARIANT, OPERATOR\_ATTRIBUTES VARIANT);';  
  
queries[1] = "CREATE OR REPLACE TEMPORARY TABLE "+ dbName + "." + schemaName + ".query\_history\_temp AS SELECT query\_id, unit \* execution\_time \* query\_load\_percent / 100 / (3600 \* 1000) as cost from( SELECT query\_id, query\_load\_percent, CASE WHEN WAREHOUSE\_SIZE = 'X-Small' THEN 1 WHEN WAREHOUSE\_SIZE = 'Small' THEN 2 WHEN WAREHOUSE\_SIZE = 'Medium' THEN 4 WHEN WAREHOUSE\_SIZE = 'Large' THEN 6 WHEN WAREHOUSE\_SIZE = 'X-Large' THEN 8 WHEN WAREHOUSE\_SIZE = '2X-Large' THEN 10 WHEN WAREHOUSE\_SIZE = '3X-Large' THEN 12 WHEN WAREHOUSE\_SIZE = '4X-Large' THEN 14 WHEN WAREHOUSE\_SIZE = '5X-Large' THEN 16 WHEN WAREHOUSE\_SIZE = '6X-Large' THEN 18 ELSE 1 END as unit, execution\_time FROM SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY WHERE START\_TIME > dateadd(day, -1, current\_date) ORDER BY start\_time) where cost is not null AND cost > 0.1;";  
  
  
var returnVal = "SUCCESS";  
var error = "";  
for (let i = 0; i < queries.length; i++) {  
 var stmt = snowflake.createStatement({sqlText:queries[i]});  
 try  
 {  
 stmt.execute();  
 }  
 catch (err)  
 {  
 logError(err)  
 error += "Failed: " + err;  
 }  
}  
if(error.length > 0 ) {  
 return error;  
}  
  
var actualQueryId = 'SELECT tmp.query\_id FROM '+ dbName + '.' + schemaName + '.query\_history\_temp tmp WHERE NOT EXISTS (SELECT query\_id FROM QUERY\_PROFILE WHERE query\_id = tmp.query\_id);';  
  
var profileInsert = 'INSERT INTO ' + dbName + '.' + schemaName + '.QUERY\_PROFILE select \* from table(get\_query\_operator\_stats(?));';  
var stmt = snowflake.createStatement({sqlText: actualQueryId});  
  
 try  
 {  
 var result\_set1 = stmt.execute();  
 while (result\_set1.next()) {  
 var queryId = result\_set1.getColumnValue(1);  
 var profileInsertstmt = snowflake.createStatement({sqlText: profileInsert, binds:[queryId]});  
 profileInsertstmt.execute();  
  
 }  
 }  
 catch (err)  
 {  
 logError(err)  
 error += "Failed: " + err;  
 }  
  
return returnVal;  
$$;  
  
CREATE OR REPLACE PROCEDURE warehouse\_proc(dbname STRING, schemaname STRING)  
 RETURNS VARCHAR(252)  
 LANGUAGE JAVASCRIPT  
 EXECUTE AS CALLER  
AS  
$$  
  
function logError(err)  
{  
 var fail\_sql = "INSERT INTO REPLICATION\_LOG VALUES (current\_date,'FAIL', " + "'"+ err +"'"+");" ;  
 sql\_command1 = snowflake.createStatement({sqlText: fail\_sql} );  
 sql\_command1.execute();  
}  
  
try {  
 var query = 'CREATE DATABASE IF NOT EXISTS ' + DBNAME + ';';  
 var stmt = snowflake.createStatement({  
 sqlText: query  
 })  
 stmt.execute();  
 result = "Database: " + DBNAME + " creation is success";  
} catch (err) {  
 logError(err)  
 return "Failed to create DB " + DBNAME + ", error: " + err;  
}  
  
try {  
 var query = 'CREATE SCHEMA IF NOT EXISTS ' + DBNAME + '.' + SCHEMANAME + ';';  
 var stmt = snowflake.createStatement({  
 sqlText: query  
 })  
 stmt.execute();  
 result += "\nSchema: " + SCHEMANAME + " creation is success";  
} catch (err) {  
 logError(err)  
 return "Failed to create the schema " + SCHEMANAME + ", error: " + err;  
}  
  
var showWarehouse = 'SHOW WAREHOUSES;';  
var createWarehouseTable = 'CREATE OR REPLACE TRANSIENT TABLE ' + DBNAME + '.' + SCHEMANAME + '.WAREHOUSES AS SELECT \* FROM TABLE(result\_scan(last\_query\_id()));';  
var returnVal = "SUCCESS";  
var error = "";  
  
try {  
 var warehouseStmt = snowflake.createStatement({  
 sqlText: showWarehouse  
 });  
 var resultSet = warehouseStmt.execute();  
 // Checking if the SHOW WAREHOUSES statement returned any rows before creating the table  
 if (resultSet.next() == false) {  
 error += "No warehouses found.";  
 } else {  
 var warehouseTableStmt = snowflake.createStatement({  
 sqlText: createWarehouseTable  
 });  
 warehouseTableStmt.execute();  
 }  
} catch (err) {  
 logError(err)  
 error += "Failed: " + err;  
}  
  
try {  
 var createWP = 'CREATE OR REPLACE TRANSIENT TABLE ' + DBNAME + '.' + SCHEMANAME + '.WAREHOUSE\_PARAMETERS (WAREHOUSE VARCHAR(1000), KEY VARCHAR(1000), VALUE VARCHAR(1000), DEFUALT VARCHAR(1000),LEVEL VARCHAR(1000), DESCRIPTION VARCHAR(10000),TYPE VARCHAR(100));';  
  
 var createWPStmt = snowflake.createStatement({  
 sqlText: createWP  
 });  
 createWPStmt.execute();  
} catch (err) {  
 logError(err)  
 error += "Failed: " + err;  
}  
var showWP = '';  
try {  
  
 var wn = 'SELECT \* FROM ' + DBNAME + '.' + SCHEMANAME + '.WAREHOUSES;';  
 var wnStmt = snowflake.createStatement({  
 sqlText: wn  
 });  
 var resultSet1 = wnStmt.execute();  
 while (resultSet1.next()) {  
 var whName = resultSet1.getColumnValue('name');  
  
 showWP = 'SHOW PARAMETERS IN WAREHOUSE ' + whName + ';';  
  
 var showWPStmt = snowflake.createStatement({  
 sqlText: showWP  
 });  
 showWPStmt.execute();  
  
 var wpInsert = 'INSERT INTO ' + DBNAME + '.' + SCHEMANAME + '.WAREHOUSE\_PARAMETERS SELECT ' + "'" + whName + "'" + ',\* FROM TABLE (result\_scan(last\_query\_id()));';  
  
 var wpInsertStmt = snowflake.createStatement({  
 sqlText: wpInsert  
 });  
 wpInsertStmt.execute();  
  
 }  
} catch (err) {  
  
 error += "Failed: " + err;  
 return logError(err)  
  
}  
  
if (error.length > 0) {  
 return error;  
}  
  
return returnVal;  
$$;

To continue execution on certain interval this is created as tasks.

-- create account usage tables Task  
CREATE OR REPLACE TASK replicate\_metadata  
 WAREHOUSE = UNRAVELDATA  
 SCHEDULE = '60 MINUTE'  
AS  
call REPLICATE\_ACCOUNT\_USAGE('UNRAVEL\_SHARE','SCHEMA\_4823',2);  
  
-- create warehouse replicate Task  
CREATE OR REPLACE TASK createWarehouseTable  
 WAREHOUSE = UNRAVELDATA  
 SCHEDULE = '60 MINUTE'  
AS  
call warehouse\_proc('UNRAVEL\_SHARE','SCHEMA\_4823');  
  
-- create profile replicate task  
CREATE OR REPLACE TASK createProfileTable  
 WAREHOUSE = UNRAVELDATA  
 SCHEDULE = '60 MINUTE'  
AS  
call create\_query\_profile('UNRAVEL\_SHARE','SCHEMA\_4823');  
  
-- create Task for replicating information schema query history  
CREATE OR REPLACE TASK replicate\_realtime\_query  
 WAREHOUSE = UNRAVELDATA  
 SCHEDULE = '10 MINUTE'  
AS  
call REPLICATE\_REALTIME\_QUERY('UNRAVEL\_SHARE','SCHEMA\_4823',10);  
  
ALTER TASK replicate\_metadata RESUME;  
  
ALTER TASK createWarehouseTable RESUME;  
  
ALTER TASK createProfileTable RESUME;  
  
ALTER TASK replicate\_realtime\_query RESUME;

## Share the Transient tables to unravel account.

Eg url : <https://app.snowflake.com/fwttice/primary_pg/#/data/shared/outbound>

Replace your account in above url intead of “fwttice”

From UI :

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Now the tables are shared to **HFB47355 Account**

## Receive the data in recipient account.

Login as account admin.

A screenshot of a computer

Description automatically generated

Click here.

Select the role

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Now data is available in recipient account

A screenshot of a computer

Description automatically generated

## Configure recipient account in unravel.

Database eg : UNRAVEL\_SHARE\_SNOWFLAKE\_SECURE\_SHARE\_1704953581612

Other configurations are same.