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	Release

Table Of Contents

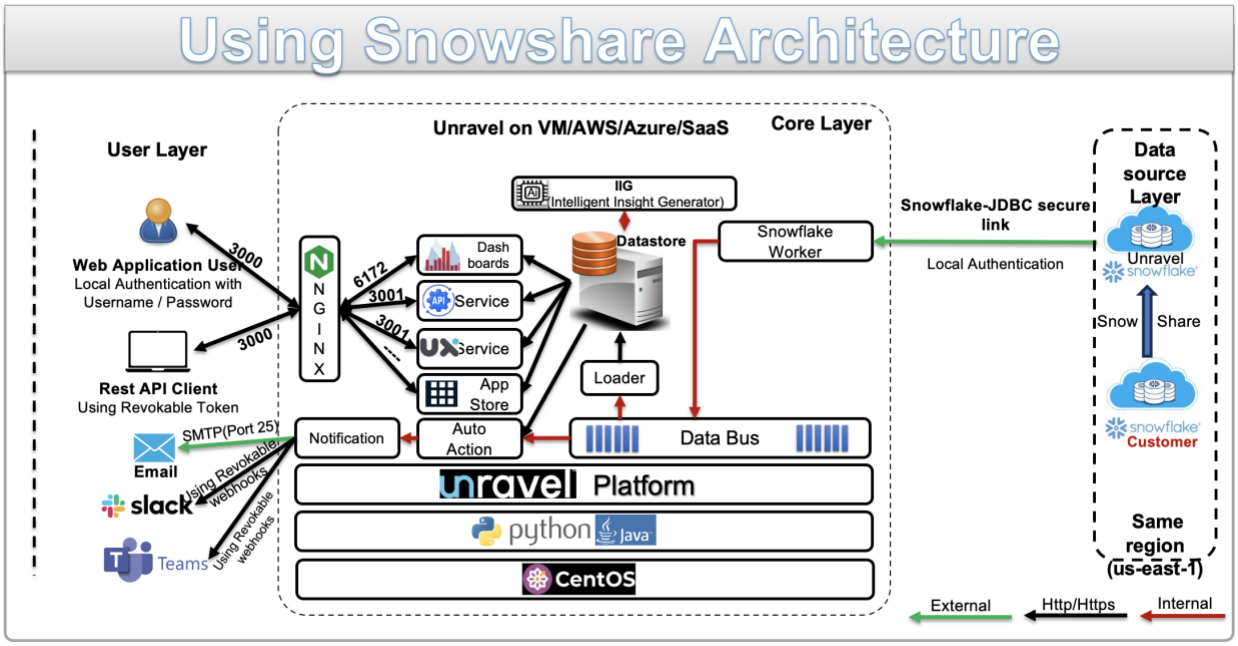
Document Version Record

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

1. Objectives

Health check download for snowflake unravel product.

2. Architecture



3. Pre-requisite

A. **Snowflake account Access through snowshare**

i. Create a unravel user in snowflake.
ii. Grant select for unravel user on schema SNOWFLAKE.ACCOUNT_USAGE.
iii. 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.
iv. Execute procedure to create tables and serverless task to populate the metadata from the account metadata.
v. 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”.
vi. Based on the latency required the serverless/warehouse task will cost.

4. Share Snowflake Metadata with Snowflake secure share.

4.1. 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_T;

USE UNRAVEL_SHARE.SCHEMA_4823_T;

CREATE OR REPLACE PROCEDURE CREATE_TABLES(DB STRING, SCHEMA STRING)
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);
CREATE OR REPLACE TABLE replication_log (
  eventDate DATE DEFAULT current_date,
  executionStatus VARCHAR(1000) DEFAULT NULL,
  remarks VARCHAR(1000)
);
CREATE OR REPLACE TRANSIENT TABLE WAREHOUSE_METERING_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE WAREHOUSE_EVENTS_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_EVENTS_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE WAREHOUSE_LOAD_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_LOAD_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE TABLES WITH DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.TABLES;
CREATE OR REPLACE TRANSIENT TABLE METERING_DAILY_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE SNOWFLAKE.ACCOUNT_USAGE.METERING_DAILY_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE METERING_HISTORY WITH DATA_RETENTION_TIME_IN_DAYS=0
LIKE SNOWFLAKE.ACCOUNT_USAGE.METERING_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE DATABASE_REPLICATION_USAGE_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.DATABASE_REPLICATION_USAGE_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE REPLICATION_GROUP_USAGE_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.REPLICATION_GROUP_USAGE_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE DATABASE_STORAGE_USAGE_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.DATABASE_STORAGE_USAGE_HISTORY;
```

```
CREATE OR REPLACE TRANSIENT TABLE STAGE_STORAGE_USAGE_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.STAGE_STORAGE_USAGE_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE SEARCH_OPTIMIZATION_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.SEARCH_OPTIMIZATION_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE DATA_TRANSFER_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE SNOWFLAKE.ACCOUNT_USAGE.DATA_TRANSFER_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE AUTOMATIC_CLUSTERING_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.AUTOMATIC_CLUSTERING_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY WITH
DATA_RETENTION_TIME_IN_DAYS=0 LIKE
SNOWFLAKE.ACCOUNT_USAGE.SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE TAG_REFERENCES WITH DATA_RETENTION_TIME_IN_DAYS=0
LIKE SNOWFLAKE.ACCOUNT_USAGE.TAG_REFERENCES;
CREATE OR REPLACE TRANSIENT TABLE QUERY_HISTORY WITH DATA_RETENTION_TIME_IN_DAYS=0
LIKE SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE ACCESS_HISTORY WITH DATA_RETENTION_TIME_IN_DAYS=0
LIKE SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY;
CREATE OR REPLACE TRANSIENT TABLE IS_QUERY_HISTORY WITH DATA_RETENTION_TIME_IN_DAYS=0
AS SELECT * FROM TABLE(INFORMATION_SCHEMA.QUERY_HISTORY()) WHERE 1=0;
RETURN 'SUCCESS';

END;



-- PROCEDURE FOR REPLICATE ACCOUNT_USAGE
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);

TRUNCATE TABLE IF EXISTS WAREHOUSE_METERING_HISTORY;
INSERT INTO WAREHOUSE_METERING_HISTORY  SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY HIS WHERE HIS.START_TIME >
DATEADD(Day ,-:LOOK_BACK_DAYS, current_date)  ;

TRUNCATE TABLE IF EXISTS WAREHOUSE_EVENTS_HISTORY ;
INSERT INTO WAREHOUSE_EVENTS_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_EVENTS_HISTORY HIS WHERE HIS.TIMESTAMP >
DATEADD(Day ,-:LOOK_BACK_DAYS, current_date)  ;
```

```
TRUNCATE TABLE IF EXISTS  WAREHOUSE_LOAD_HISTORY ;
INSERT INTO WAREHOUSE_LOAD_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_LOAD_HISTORY HIS WHERE HIS.START_TIME >
DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  TABLES ;
INSERT INTO TABLES SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.TABLES;

TRUNCATE TABLE IF EXISTS  METERING_DAILY_HISTORY ;
INSERT INTO METERING_DAILY_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.METERING_DAILY_HISTORY HIS WHERE HIS.USAGE_DATE >
DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  METERING_HISTORY ;
INSERT INTO METERING_HISTORY SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.METERING_HISTORY
HIS WHERE HIS.START_TIME > DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  DATABASE_REPLICATION_USAGE_HISTORY ;
INSERT INTO DATABASE_REPLICATION_USAGE_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.DATABASE_REPLICATION_USAGE_HISTORY HIS WHERE HIS.START_TIME
> DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  REPLICATION_GROUP_USAGE_HISTORY ;
INSERT INTO REPLICATION_GROUP_USAGE_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.REPLICATION_GROUP_USAGE_HISTORY HIS WHERE HIS.START_TIME >
DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  DATABASE_STORAGE_USAGE_HISTORY ;
INSERT INTO DATABASE_STORAGE_USAGE_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.DATABASE_STORAGE_USAGE_HISTORY HIS WHERE HIS.USAGE_DATE >
DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  STAGE_STORAGE_USAGE_HISTORY ;
INSERT INTO STAGE_STORAGE_USAGE_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.STAGE_STORAGE_USAGE_HISTORY HIS WHERE HIS.USAGE_DATE >
DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  SEARCH_OPTIMIZATION_HISTORY ;
INSERT INTO SEARCH_OPTIMIZATION_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.SEARCH_OPTIMIZATION_HISTORY HIS WHERE HIS.START_TIME >
DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  DATA_TRANSFER_HISTORY ;
INSERT INTO DATA_TRANSFER_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.DATA_TRANSFER_HISTORY HIS WHERE HIS.START_TIME > DATEADD(Day
,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  AUTOMATIC_CLUSTERING_HISTORY ;
INSERT INTO AUTOMATIC_CLUSTERING_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.AUTOMATIC_CLUSTERING_HISTORY HIS WHERE HIS.START_TIME >
DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY ;
```

```
INSERT INTO SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY SELECT * FROM
SNOWFLAKE.ACCOUNT_USAGE.SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY HIS WHERE
HIS.START_TIME > DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  TAG_REFERENCES ;
INSERT INTO TAG_REFERENCES SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.TAG_REFERENCES ;


TRUNCATE TABLE IF EXISTS  QUERY_HISTORY ;
INSERT INTO QUERY_HISTORY SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY HIS
WHERE HIS.START_TIME > DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

TRUNCATE TABLE IF EXISTS  ACCESS_HISTORY ;
INSERT INTO ACCESS_HISTORY SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY HIS
WHERE HIS.QUERY_START_TIME > DATEADD(Day ,-:LOOK_BACK_DAYS, current_date);

RETURN 'SUCCESS';

END;

--PROCEDURE FOR REPLICATE REALTIME QUERY
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);

TRUNCATE TABLE IF EXISTS  IS_QUERY_HISTORY ;
INSERT INTO IS_QUERY_HISTORY SELECT * FROM
TABLE(INFORMATION_SCHEMA.QUERY_HISTORY(dateadd('hours',-:LOOK_BACK_HOURS
,current_timestamp()),current_timestamp(),10000)) order by start_time ;


RETURN 'SUCCESS';

END;



CCREATE OR REPLACE PROCEDURE create_query_profile(dbname string, schemaname string, cost string,
days String)
   returns VARCHAR(25200)
   LANGUAGE javascript
```

```
AS
$$

var create_query_profile_task = "create_query_profile ---> Getting Query Profile data and inserting into
Query_profile table";
var task="profile_task";
function logError(err, taskName)
{
   var fail_sql = "INSERT INTO REPLICATION_LOG VALUES (current_timestamp,'FAILED', "+"'"+ err +"'"+",
"+"'"+ taskName +"'"+");" ;
   sql_command1 = snowflake.createStatement({sqlText: fail_sql} );
   sql_command1.execute();
}

function insertToReplicationLog(status, message, taskName)
{
   var query_profile_status = "INSERT INTO REPLICATION_LOG VALUES (current_timestamp, "+"'"+status
+"'"+", "+"'"+ message +"'"+", "+"'"+ taskName +"'"+");" ;
   sql_command1 = snowflake.createStatement({sqlText: query_profile_status} );
   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, create_query_profile_task)
   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, create_query_profile_task)
   return "Failed to create the schema "+ SCHEMANAME + ", error: " + err;
}

var schemaName = SCHEMANAME;
var dbName = DBNAME;
var cost = parseFloat(COST);
var lookBackDays = -parseInt(DAYS);
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, "+ lookBackDays
+", current_date) ORDER BY start_time) where cost is not null AND cost > " +cost+";";


queries[2] = "SELECT count(1) FROM "+ dbName + "." + schemaName + ".query_history_temp";


var returnVal = "SUCCESS";
var error = "";
var total_query_count = 0;
var failed_query_count = 0;
for (let i = 0; i < queries.length; i++) {
   var stmt = snowflake.createStatement({sqlText:queries[i]});
   try
   {
      var res = stmt.execute();

      if(i==2)
      {
       res.next();
       total_query_count = res.getColumnValue(1)
       var message ="Total records = "+ total_query_count;
       insertToReplicationLog("started",message,task);
      }


   }
   catch (err)
   {
      logError(err, create_query_profile_task)
      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});
 var query_count = 0;
   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();
      query_count++;
      if (query_count % 100 == 0){
       var message ="Total records = "+ total_query_count +", completed = "+query_count+", failed =
"+failed_query_count;
       insertToReplicationLog("running", message, task);
       }
      }

   }
   catch (err)
   {
       logError(err, create_query_profile_task)
       error += "Failed: " + err;
   }

var message ="Total records = "+ total_query_count +", completed = "+query_count+", failed =
"+failed_query_count;
insertToReplicationLog("completed", message, task);

return returnVal;
$$;


CREATE OR REPLACE PROCEDURE warehouse_proc(dbname STRING, schemaname STRING)
  RETURNS VARCHAR(252)
  LANGUAGE JAVASCRIPT
  EXECUTE AS CALLER
AS
$$

var warehouse_proc_task = "warehouse_proc ---> Warehouses and Warehouse_Parameter Table
Creation";
function logError(err, taskName)
{
   var fail_sql = "INSERT INTO REPLICATION_LOG VALUES (current_timestamp,'FAILED', "+"""+ err +"""+",
"+"""+ taskName +"""+");" ;
   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, warehouse_proc_task);
        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, warehouse_proc_task);
        return "Failed to create the schema " + SCHEMANAME + ", error: " + err;
}


var returnVal = "SUCCESS";
var error = "";

try {
   // 1. create warehouse table if not exist
   var createWarehouseTable = 'CREATE TRANSIENT TABLE IF NOT EXISTS ' + DBNAME + '.' +
SCHEMANAME + '.WAREHOUSES(NAME VARCHAR(16777216), STATE VARCHAR(16777216), TYPE
VARCHAR(16777216), SIZE VARCHAR(16777216), MIN_CLUSTER_COUNT NUMBER(38,0),
MAX_CLUSTER_COUNT NUMBER(38,0), STARTED_CLUSTERS NUMBER(38,0), RUNNING NUMBER(38,0),
QUEUED NUMBER(38,0), IS_DEFAULT VARCHAR(1), IS_CURRENT VARCHAR(1), AUTO_SUSPEND
NUMBER(38,0), AUTO_RESUME VARCHAR(16777216), AVAILABLE VARCHAR(16777216), PROVISIONING
VARCHAR(16777216), QUIESCING VARCHAR(16777216), OTHER VARCHAR(16777216), CREATED_ON
TIMESTAMP_LTZ(9), RESUMED_ON TIMESTAMP_LTZ(9), UPDATED_ON TIMESTAMP_LTZ(9), OWNER
VARCHAR(16777216), COMMENT VARCHAR(16777216), ENABLE_QUERY_ACCELERATION
VARCHAR(16777216), QUERY_ACCELERATION_MAX_SCALE_FACTOR NUMBER(38,0),
RESOURCE_MONITOR VARCHAR(16777216), ACTIVES NUMBER(38,0), PENDINGS NUMBER(38,0), FAILED
NUMBER(38,0), SUSPENDED NUMBER(38,0), UUID VARCHAR(16777216), SCALING_POLICY
VARCHAR(16777216), BUDGET VARCHAR(16777216));';


var createWarehouseTableStmt = snowflake.createStatement({
                sqlText: createWarehouseTable
        });
   createWarehouseTableStmt.execute();

   // 2. truncate table
   var truncateWarehouse = 'TRUNCATE TABLE IF EXISTS ' + DBNAME + '.' + SCHEMANAME +
'.WAREHOUSES;';
   var truncateWarehouseStmt = snowflake.createStatement({
```

```
                    sqlText: truncateWarehouse
            });
    truncateWarehouseStmt.execute();


    // 3. run show warehouses
    var showWarehouse = 'SHOW WAREHOUSES;';
            var showWarehouseStmt = snowflake.createStatement({
                    sqlText: showWarehouse
            });
    var resultSet = showWarehouseStmt.execute();

    // 4. insert to warehouse
    var insertToWarehouse = 'INSERT INTO ' + DBNAME + '.' + SCHEMANAME + '.WAREHOUSES  SELECT *
FROM TABLE(result_scan(last_query_id()));';
    var insertToWarehouseStmt = snowflake.createStatement({
                            sqlText: insertToWarehouse
                    });
            insertToWarehouseStmt.execute();

} catch (err) {
            logError(err, warehouse_proc_task);
    error += "Failed: " + err;
}

try {

    //1. create warehouse parameters table
            var createWP = 'CREATE TRANSIENT TABLE IF NOT EXISTS ' + 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();

    //2. trunate warehouse parameter tables
    var truncateWarehouseParameter = 'TRUNCATE TABLE IF EXISTS ' + DBNAME + '.' + SCHEMANAME +
'.WAREHOUSE_PARAMETERS;';
    var truncateWarehouseParameterStmt = snowflake.createStatement({
                    sqlText: truncateWarehouseParameter
            });
    truncateWarehouseParameterStmt.execute();

} catch (err) {
            logError(err, warehouse_proc_task);
    error += "Failed: " + err;
}


try {
```

```
  //3.Get warehouse details
        var wn = 'SELECT * FROM ' + DBNAME + '.' + SCHEMANAME + '.WAREHOUSES;';
        var wnStmt = snowflake.createStatement({
                sqlText: wn
        });
        var resultSet1 = wnStmt.execute();
        while (resultSet1.next()) {
                var whName = resultSet1.getColumnValue(1);
    //4. show warehouse parameters
                var showWP = 'SHOW PARAMETERS IN WAREHOUSE ' + whName + ';';
                var showWPStmt = snowflake.createStatement({
                        sqlText: showWP
                });
                showWPStmt.execute();

     //5. insert into WAREHOUSE_PARAMETERS table
                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, warehouse_proc_task);

}

if (error.length > 0) {
        return error;
}

return returnVal;
$$;
```

4.2. One time execution

```
CALL CREATE_TABLES('UNRAVEL_SHARE','SCHEMA_4823_T');
CALL REPLICATE_ACCOUNT_USAGE('UNRAVEL_SHARE','SCHEMA_4823_T',2);
CALL WAREHOUSE_PROC('UNRAVEL_SHARE','SCHEMA_4823_T');
CALL REPLICATE_REALTIME_QUERY('UNRAVEL_SHARE','SCHEMA_4823_T',10);
CALL CREATE_QUERY_PROFILE(dbname => 'UNRAVEL_SHARE',schemaname => 'SCHEMA_4823_T', cost
=> '1', days => '1');
```


4.3. 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_T',2);

-- create warehouse replicate Task
CREATE OR REPLACE TASK createWarehouseTable
  WAREHOUSE = UNRAVELDATA
  SCHEDULE = '60 MINUTE'
AS
CALL warehouse_proc('UNRAVEL_SHARE','SCHEMA_4823_T');

-- create profile replicate task
CREATE OR REPLACE TASK createProfileTable
  WAREHOUSE = UNRAVELDATA
  SCHEDULE = '60 MINUTE'
AS
CALL create_query_profile(dbname => 'UNRAVEL_SHARE',schemaname => 'SCHEMA_4823_T', cost =>
'1', days => '1');

-- create Task for replicating information schema query history
CREATE OR REPLACE TASK replicate_realtime_query
  WAREHOUSE = UNRAVELDATA
  SCHEDULE = '30 MINUTE'
AS
CALL REPLICATE_REALTIME_QUERY('UNRAVEL_SHARE','SCHEMA_4823_T',10);

-- START ALL THE TASKS

ALTER TASK replicate_metadata RESUME;

ALTER TASK createWarehouseTable RESUME;

ALTER TASK createProfileTable RESUME;

ALTER TASK replicate_realtime_query RESUME;
```

5. Share the Transient tables to unravel account.

From SQL (Recommended):

```
-- Share tables
Create share RICOH_UNRAVEL_SHARE;
```

```
Grant Usage on database UNRAVEL_SHARE to share RICOH_UNRAVEL_SHARE;
Grant Usage on schema SCHEMA_4823_T to share RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE WAREHOUSE_METERING_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE WAREHOUSE_EVENTS_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE WAREHOUSE_LOAD_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE TABLES  to share RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE METERING_DAILY_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE METERING_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE DATABASE_REPLICATION_USAGE_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE REPLICATION_GROUP_USAGE_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE DATABASE_STORAGE_USAGE_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE STAGE_STORAGE_USAGE_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE SEARCH_OPTIMIZATION_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE DATA_TRANSFER_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE AUTOMATIC_CLUSTERING_HISTORY  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY  to
share RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE TAG_REFERENCES  to share RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE QUERY_HISTORY  to share RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE ACCESS_HISTORY  to share RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE IS_QUERY_HISTORY  to share RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE WAREHOUSE_PARAMETERS  to share
RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE WAREHOUSES  to share RICOH_UNRAVEL_SHARE;
GRANT SELECT ON TABLE QUERY_PROFILE  to share RICOH_UNRAVEL_SHARE;
alter share RICOH_UNRAVEL_SHARE add accounts = HFB47355;
```

6. Receive the data in recipient account (Sql mode).

```
--Validate that the inbound share is available to the consumer account.
```

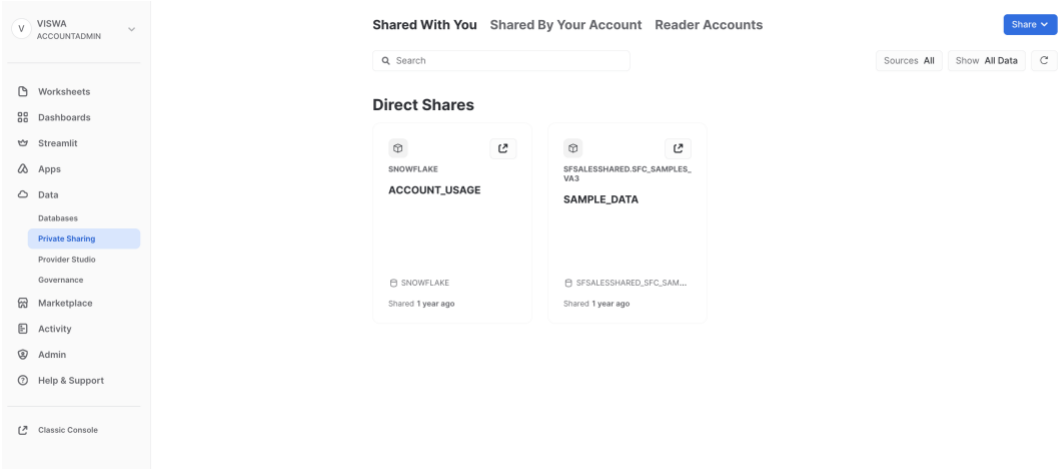
```
Show shares like '%UNRAVEL%';

Use role AccountAdmin;
create database RICOH_SHARE from share
FWTTICE.PRIMARY_PG.RICOH_UNRAVEL_SHARE;
```

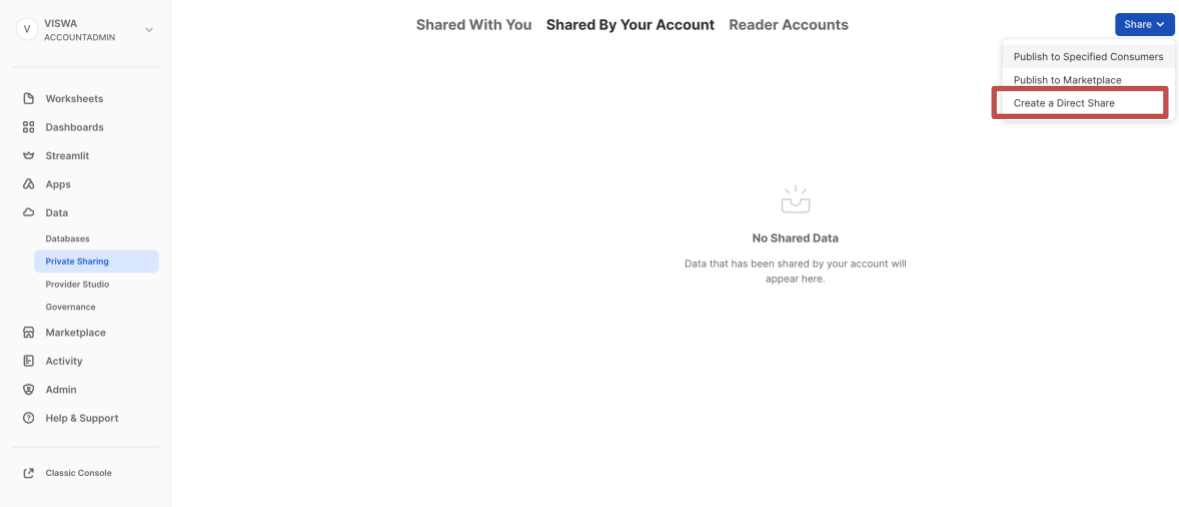
7. Share the Transient tables to unravel account (UI Mode).

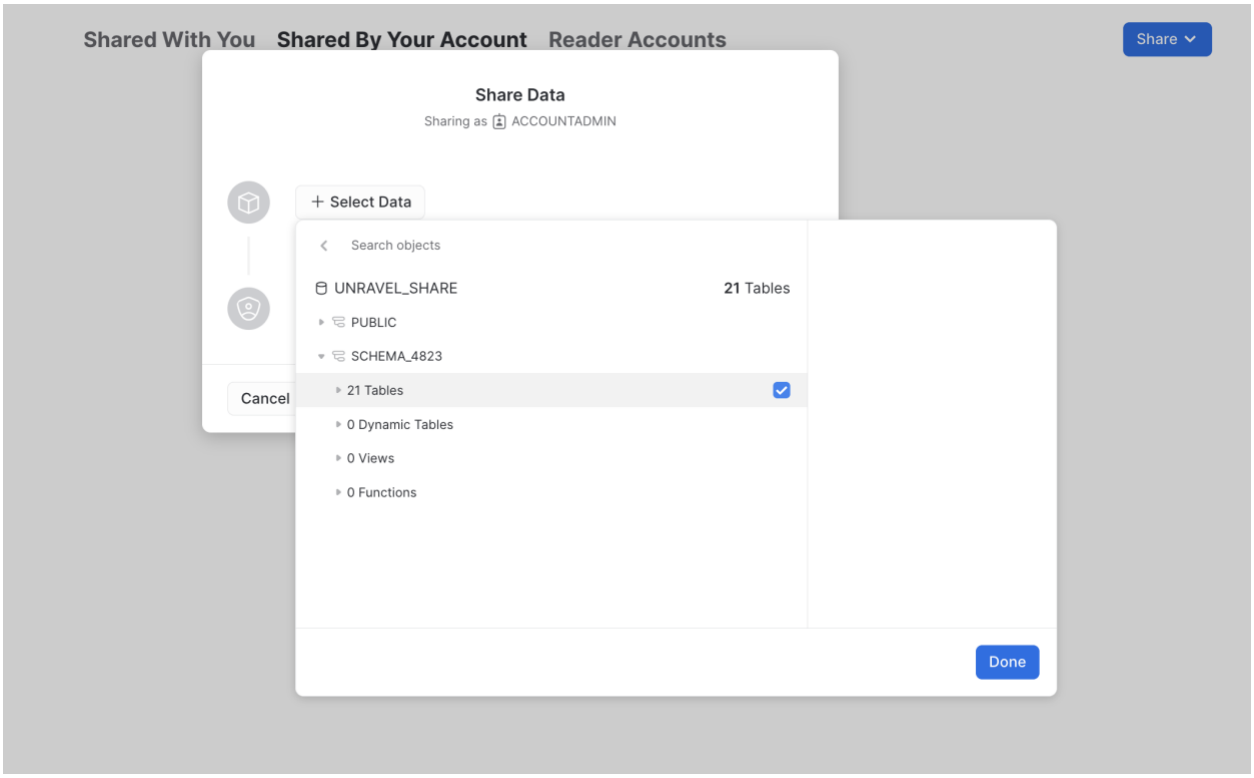
Sharing through UI :

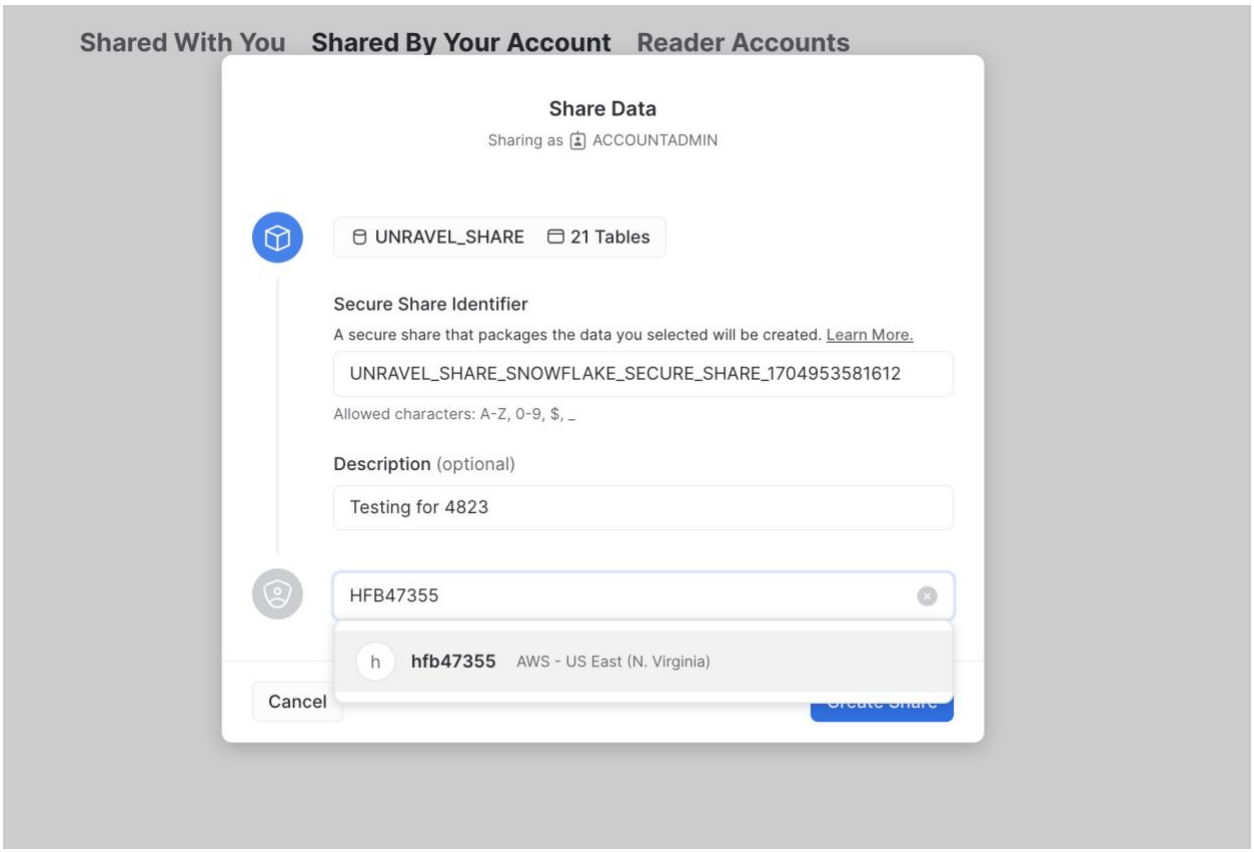
E.g. URL : https://app.snowflake.com/fwttice/primary_pg/#/data/shared/outbound

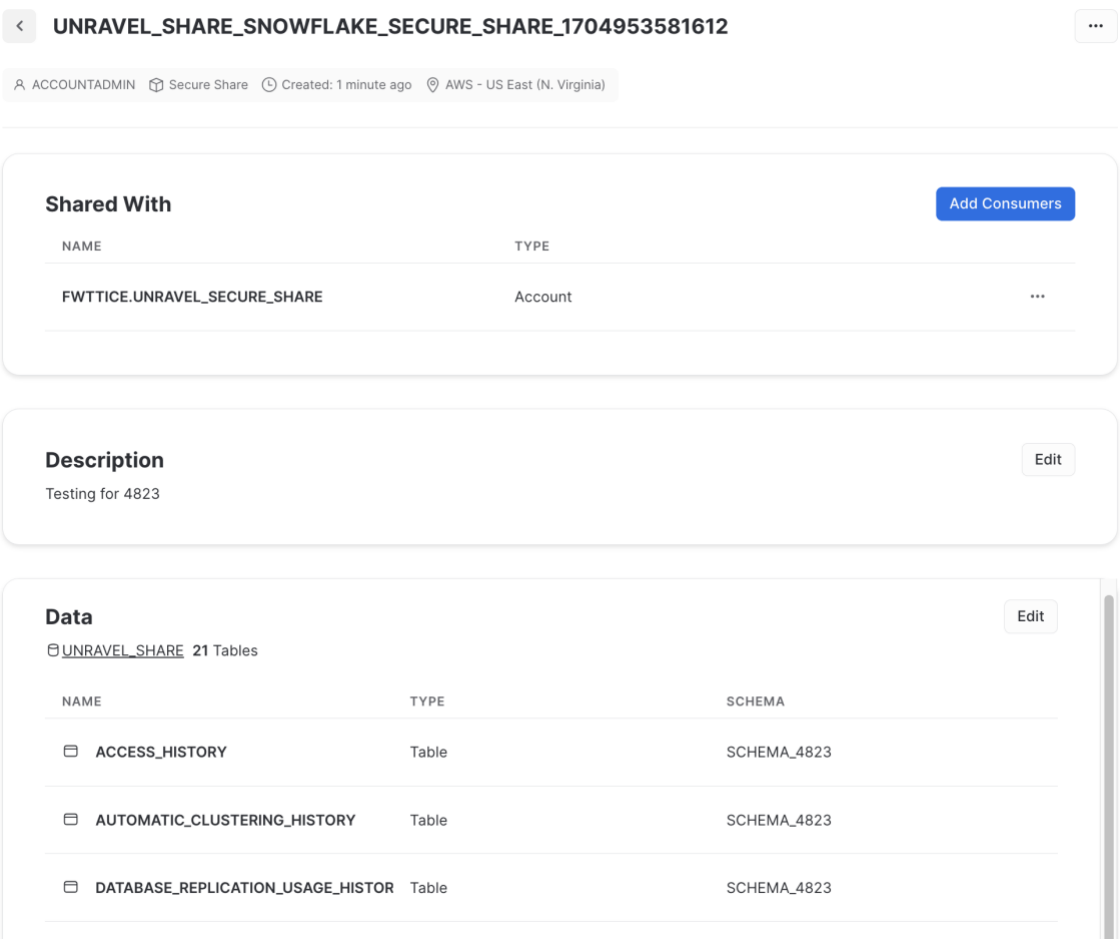
Replace your account in above URL instead of "fwttice"







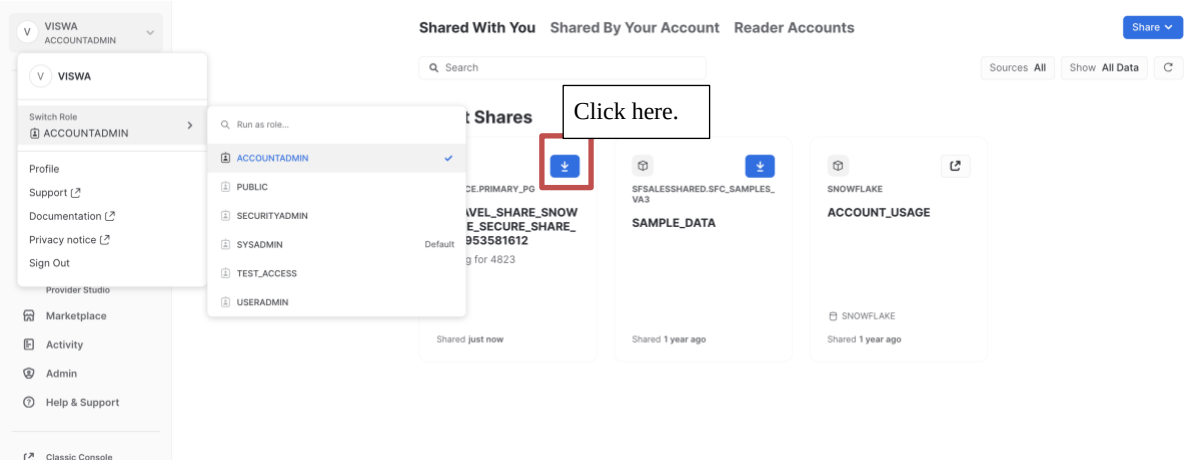




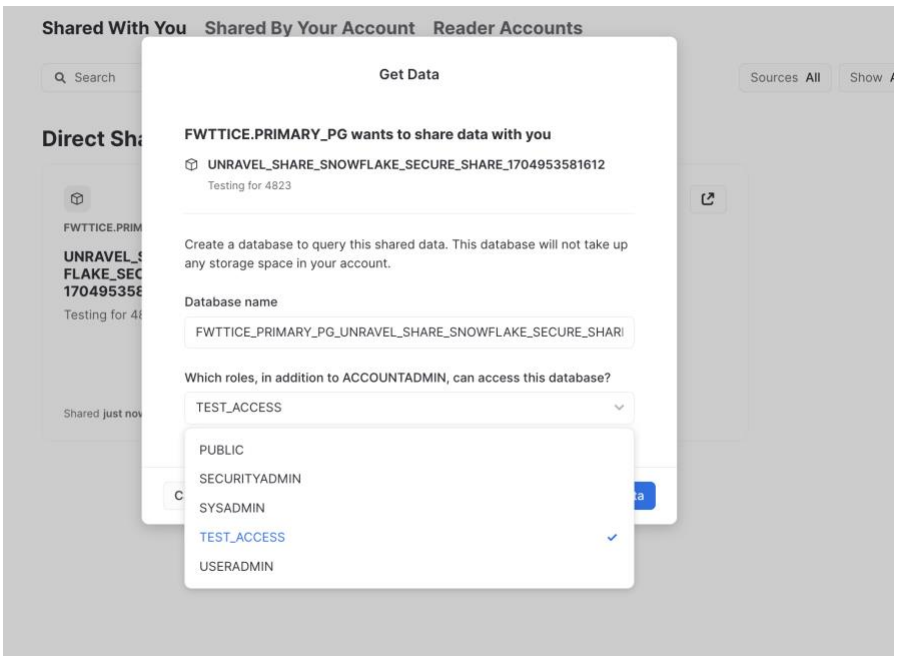
Now the tables are shared to **HFB47355 Account**

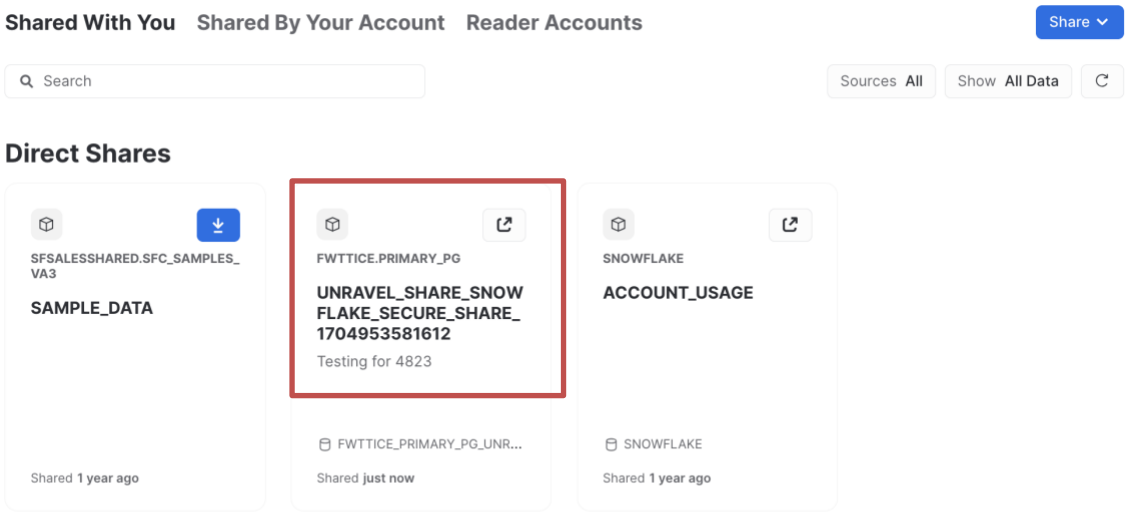
8. Receive the data in recipient account ui mode.

Login as account admin.

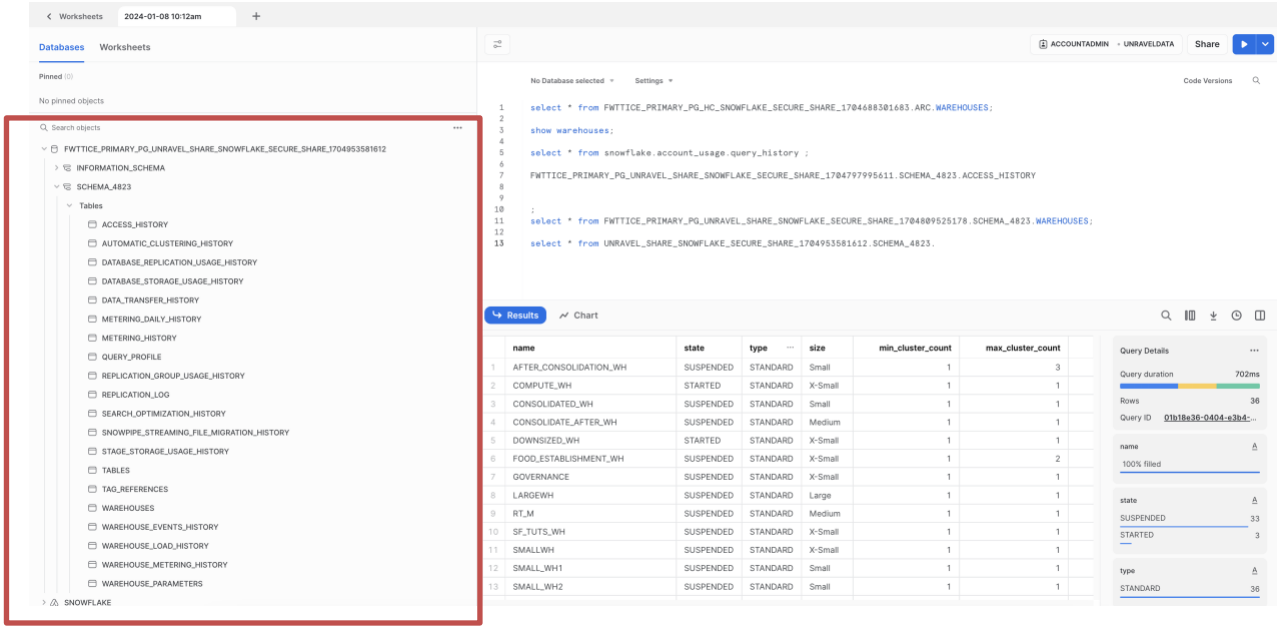


Select the role.





Now data is available in recipient account.



9. Configure recipient account in unravel.

Show shares like '%UNRAVEL%';

Use role AccountAdmin;

create database RICOH_SHARE from share
FWTTICE.PRIMARY_PG.RICOH_UNRAVEL_SHARE;

Database: RICOH_SHARE

Other configurations are same.