# **Security Monitor Queries**

## **Note : Flow of execution :**

Step 1 : Execute **Snowflake Prerequisite** Queries.

Step 2 : Execute **Table Creation** Queries.

Step 3 : Execute **Stored Procedures and tasks to update above table** Queries.

## **1 Snowflake Prerequisite Query**

Please refer **Snowflake Prerequisite Query** and execute those queries if you have not already executed them before you move on to the next steps

## **2 List of Tables Used**

| **Table Name** | **Dashboards** | **Latency** |
| --- | --- | --- |
| ROLES\_INFO\_TB | Security Monitoring | None |
| AUTHENTICATION\_BREAKDOWN\_TB | Security Monitoring | Up to 120 minutes |
| NETWORK\_POLICIES\_TB | Security Monitoring | None |

## **3 Table Creation**

**Note :** Run the below queries to make sure that you use the appropriate role, warehouse, database and schema before you create the tables and run the further queries.

**QUERY :**

USE role MONITOR\_ADMIN;

USE warehouse MONITOR\_WH;

USE database MONITOR\_DB;

CREATE schema SECURITY\_MONITOR\_SCHEMA;

USE schema SECURITY\_MONITOR\_SCHEMA;

### 

### **3.1 ROLES\_INFO\_TB**

**Note** : This table uses the command show roles to get the data.

Refresh Rate depends upon the scheduled time for the task ROLES\_INFO\_TB\_TASK which calls the stored procedure for inserting the latest data to the table.

CREATE

OR REPLACE TRANSIENT TABLE ROLES\_INFO\_TB (

CREATED\_ON TIMESTAMP\_LTZ,

NAME VARCHAR,

IS\_DEFAULT VARCHAR,

IS\_CURRENT VARCHAR,

IS\_INHERITED VARCHAR,

ASSIGNED\_TO\_USERS NUMBER,

GRANTED\_TO\_ROLES NUMBER,

GRANTED\_ROLES NUMBER,

OWNER VARCHAR,

RCOMMENT VARCHAR,

REFRESH\_DATE TIMESTAMP\_LTZ DEFAULT CURRENT\_TIMESTAMP(),

ROLE\_TYPE varchar(30)

) COMMENT = 'stores snapshot of current snowflake roles';

### **3.2 NETWORK\_POLICIES\_TB**

**Note** : This table uses the command ‘**show network policies**’ to get the data.

Refresh Rate depends upon the scheduled time for the task call\_insert\_network\_policies\_tb\_task which calls the stored procedure for inserting the latest data to the table.

CREATE

OR REPLACE TABLE network\_policies\_tb(

CREATED\_ON TIMESTAMP\_LTZ,

NAME VARCHAR,

COMMENT varchar(100),

ENTRIES\_IN\_ALLOWED\_IP\_LIST NUMBER,

ENTRIES\_IN\_BLOCKED\_IP\_LIST NUMBER,

CURRENT\_TIMESTAMPS TIMESTAMP\_LTZ

) COMMENT = 'STORES NETWORK POLICIES INCLUDING BLOCKED AND ALLOWED IP';

### **3.3 AUTHENTICATION\_BREAKDOWN\_TB**

**Note** : This table uses the view “snowflake”.”account\_usage”.”login\_history” and as per the snowflake documentation, Latency for the view may be up to 120 minutes.

Refresh Rate depends upon the scheduled time for the task Authentication\_breakdown\_task which calls the stored procedure for inserting the latest data to the table.

CREATE OR REPLACE TRANSIENT TABLE Authentication\_Breakdown\_TB (

Event\_Count NUMBER(10),

Authentication\_Factor Varchar(30),

Second\_Authentication\_Factor Varchar(30)

);

## **4 Stored procedures and Task to update above TABLES**

### **4.1 SP - SNAPSHOT\_ROLES()**

//Stored procedure to insert latest details into table ROLES\_INFO\_TB :

CREATE OR REPLACE PROCEDURE SNAPSHOT\_ROLES()

RETURNS VARCHAR

LANGUAGE JAVASCRIPT

COMMENT = 'Captures the snapshot of roles and inserts the records into ROLES\_INFO\_TB'

EXECUTE AS CALLER

AS

$$

var result = "SUCCESS";

try {

snowflake.execute( {sqlText: "alter table ROLES\_INFO\_TB drop column ROLE\_TYPE;"} );

snowflake.execute( {sqlText: "truncate table ROLES\_INFO\_TB;"} );

snowflake.execute( {sqlText: "show roles;"} );

var dcroles\_tbl\_sql = `insert into ROLES\_INFO\_TB select \*,CURRENT\_TIMESTAMP() from table(result\_scan(last\_query\_id()));`;

snowflake.execute( {sqlText: dcroles\_tbl\_sql} );

snowflake.execute( {sqlText: "alter table ROLES\_INFO\_TB add column ROLE\_TYPE varchar(30);"} );

snowflake.execute( {sqlText: "update ROLES\_INFO\_TB set ROLE\_TYPE = 'Disconnected Role' where Granted\_to\_roles = 0 and Granted\_roles = 0;"} );

snowflake.execute( {sqlText: "update ROLES\_INFO\_TB set ROLE\_TYPE = 'Connected Role' where Granted\_to\_roles > 0 or Granted\_roles > 0;"} );

}

catch (err) {

result = "FAILED: Code: " + err.code + "\n State: " + err.state;result += "\n Message: " + err.message;result += "\nStack Trace:\n" + err.stackTraceTxt;

}

return result;

$$;

call SNAPSHOT\_ROLES();

**Note :** Alter the task ROLES\_INFO\_TB\_TASK in order to schedule the calling of the stored procedure call SNAPSHOT\_ROLES(); as per the user requirements.

create or replace task ROLES\_INFO\_TB\_TASK

warehouse = MONITOR\_WH

schedule = 'USING CRON 0 3 \* \* \* UTC'

As

call SNAPSHOT\_ROLES();

ALTER TASK ROLES\_INFO\_TB\_TASK resume;

### **4.2 SP - NETWORK\_POL\_SP()**

//Stored procedure to insert latest details into table network\_policies\_tb :

**Note :** To execute the stored procedure below, the user needs to have the ACCOUNTADMIN role assigned to them. This role is necessary to retrieve information about network policies using the '**SHOW network policies**' command. Therefore, please ensure that the user executing this stored procedure has the ACCOUNTADMIN role before running it.

CREATE OR REPLACE PROCEDURE network\_pol\_sp()

RETURNS VARCHAR

LANGUAGE JAVASCRIPT

COMMENT = 'Captures the snapshot of policies and inserts the records into network\_policies\_tb'

EXECUTE AS CALLER

AS

$$

var result = "SUCCESS";

try {

snowflake.execute( {sqlText: "truncate table network\_policies\_tb;"} );

snowflake.execute( {sqlText: "use role ACCOUNTADMIN;"} );

snowflake.execute( {sqlText: "show network policies;"} );

var network\_policies\_tb\_sql = `insert into network\_policies\_tb select \*,CURRENT\_TIMESTAMP() from table(result\_scan(last\_query\_id()));`;

snowflake.execute( {sqlText: network\_policies\_tb\_sql} );

snowflake.execute( {sqlText: `use role MONITOR\_ADMIN;`} );

}

catch (err) {

result = "FAILED: Code: " + err.code + "\n State: " + err.state;result += "\n Message: " + err.message;result += "\nStack Trace:\n" + err.stackTraceTxt;

}

return result;

$$;

call network\_pol\_sp();

**Note :** Alter the task call\_insert\_network\_policies\_tb\_data in order to schedule the calling of the stored procedure call NETWORK\_POL\_SP(); as per the user requirements.

create or replace task call\_insert\_network\_policies\_tb\_task

warehouse = MONITOR\_WH

schedule = 'USING CRON 0 3 \* \* \* UTC'

As

call network\_pol\_sp();

ALTER TASK call\_insert\_network\_policies\_tb\_task resume;

### **4.3 SP - AUTHENTICATION\_BREAKDOWN\_SP()**

//Stored procedure to create a table Authentication\_Breakdown and update the same.

CREATE OR REPLACE PROCEDURE Authentication\_breakdown\_sp()

RETURNS VARCHAR

LANGUAGE JAVASCRIPT

COMMENT = 'Create a new custom table having authentication factors and its event count'

EXECUTE AS CALLER

AS

$$

var result = "SUCCESS";

try {

snowflake.execute( {sqlText: `

truncate table MONITOR\_DB.SECURITY\_MONITOR\_SCHEMA.AUTHENTICATION\_BREAKDOWN\_TB;`} );

snowflake.execute( {sqlText: `

Insert into

MONITOR\_DB.SECURITY\_MONITOR\_SCHEMA.AUTHENTICATION\_BREAKDOWN\_TB (

Event\_Count,

Authentication\_Factor,

Second\_Authentication\_Factor

)

SELECT

count(First\_Authentication\_Factor) as Event\_Count,

First\_Authentication\_Factor as Authentication\_Factor,

Second\_Authentication\_Factor

from

SNOWFLAKE.ACCOUNT\_USAGE.LOGIN\_HISTORY

group by

First\_Authentication\_Factor,

Second\_Authentication\_Factor;

`} );

snowflake.execute( {sqlText: `

update

MONITOR\_DB.SECURITY\_MONITOR\_SCHEMA.AUTHENTICATION\_BREAKDOWN\_TB

set

AUTHENTICATION\_FACTOR = (

SELECT

CONCAT(

AUTHENTICATION\_FACTOR,

CONCAT('\_', second\_authentication\_factor)

)

from

MONITOR\_DB.SECURITY\_MONITOR\_SCHEMA.AUTHENTICATION\_BREAKDOWN\_TB

where

second\_authentication\_factor IS NOT NULL

)

where

second\_authentication\_factor is not null;

`} );

}

catch (err) {

result = "FAILED: Code: " + err.code + "\n State: " + err.state;result += "\n Message: " + err.message;result += "\nStack Trace:\n" + err.stackTraceTxt;

}

return result;

$$;

call Authentication\_breakdown\_sp();

**Note :** Alter the task Authentication\_breakdown\_task in order to schedule the calling of the stored procedure call Authentication\_breakdown\_sp(); as per the user requirements.

create or replace task Authentication\_breakdown\_task

warehouse = MONITOR\_WH

schedule = 'USING CRON 0 3 \* \* \* UTC'

As

call Authentication\_breakdown\_sp();

ALTER TASK Authentication\_breakdown\_task resume;

# **Best Practices :-**

* It is mandatory to execute all the insert statements for historical load wherever required before executing SPs for incremental load of data.
* Always run historical load commands only once. If in any case you want to run it again then first truncate the respective custom table.
* You can always schedule the task as per your requirement . Reference for setting up task using cron expression - <https://docs.snowflake.com/en/sql-reference/sql/create-task.html#examples>
* If you want to customize bad query table results then always start by replacing the old table and do historical and incremental load again.