# 1. WatchKeeper Common Prerequisites

## **1.1 Snowflake Prerequisite Query**

QUERY :   
//SETTING ENVIRONMENT FOR MONITOR

USE ROLE ACCOUNTADMIN;

CREATE ROLE MONITOR\_ADMIN;

GRANT ROLE MONITOR\_ADMIN TO ROLE ACCOUNTADMIN;

CREATE ROLE MONITOR\_USER;

CREATE WAREHOUSE MONITOR\_WH;

CREATE DATABASE MONITOR\_DB;

// MONITOR\_ADMIN PRIVILEGES

GRANT IMPORTED PRIVILEGES ON DATABASE SNOWFLAKE TO ROLE MONITOR\_ADMIN;

GRANT IMPORTED PRIVILEGES ON DATABASE SNOWFLAKE TO ROLE MONITOR\_USER;

GRANT MODIFY, OPERATE, USAGE, MONITOR ON WAREHOUSE "MONITOR\_WH" TO ROLE "MONITOR\_ADMIN";

GRANT ALL ON DATABASE "MONITOR\_DB" TO ROLE "MONITOR\_ADMIN";

GRANT EXECUTE TASK ON ACCOUNT TO ROLE "MONITOR\_ADMIN";

GRANT ROLE MONITOR\_ADMIN TO USER "<user\_name>";  
  
//MONITOR\_USER Privileges

GRANT ROLE MONITOR\_USER TO USER "<user\_name>";

GRANT USAGE ON WAREHOUSE MONITOR\_WH TO ROLE MONITOR\_USER;

GRANT USAGE ON DATABASE MONITOR\_DB TO ROLE MONITOR\_USER;

GRANT USAGE ON FUTURE SCHEMAS IN DATABASE MONITOR\_DB TO ROLE MONITOR\_USER;

GRANT SELECT ON FUTURE TABLES IN DATABASE MONITOR\_DB TO ROLE MONITOR\_USER;

# 2. Storage Monitor Queries

**Note : Flow of execution :**

Step 1 : Execute **2.1 Snowflake Prerequisite** Queries.

Step 2 : Execute **2.3 Create the Stored Procedures and tasks** Queries.

## **2****.1 Snowflake Prerequisite Query**

Please refer **Section 1.1 Snowflake Prerequisite Query** and execute those queries if you have not already executed them before you move on to the next steps.

## **2.2 List of Tables Used**

## **Tables used:**

Org\_Storage

Account\_Storage

DB\_Storage

Table\_and\_Schema\_Storage

## 

## **2.3 Create the Stored Procedure and Tasks**

**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 queries

**QUERY :**

USE role MONITOR\_ADMIN;

USE warehouse MONITOR\_WH;

USE database MONITOR\_DB;

CREATE schema STORAGE\_MONITOR\_SCHEMA;

USE schema STORAGE\_MONITOR\_SCHEMA;

// CREATE Stored Procedure

CREATE OR REPLACE PROCEDURE Storage\_Monitor\_SP()

RETURNS VARCHAR

LANGUAGE JAVASCRIPT

AS

$$

var orgCommand1="CREATE TRANSIENT TABLE if not exists ORG\_STORAGE as SELECT USAGE\_DATE,SERVICE\_TYPE, AVERAGE\_BYTES/(1024\*1024\*1024\*1024) as AVERAGE\_TB, ACCOUNT\_NAME, REGION FROM SNOWFLAKE.ORGANIZATION\_USAGE.PREVIEW\_STORAGE\_DAILY\_HISTORY;"

var orgCommand2="Truncate table ORG\_STORAGE;"

var orgCommand3="Insert into ORG\_STORAGE SELECT USAGE\_DATE,SERVICE\_TYPE, AVERAGE\_BYTES/(1024\*1024\*1024\*1024) as AVERAGE\_TB,ACCOUNT\_NAME,REGION FROM SNOWFLAKE.ORGANIZATION\_USAGE.PREVIEW\_STORAGE\_DAILY\_HISTORY;"

var accCommand1="CREATE TRANSIENT TABLE if not exists ACCOUNT\_STORAGE as select USAGE\_DATE,STORAGE\_BYTES/(1024\*1024\*1024\*1024) AS STORAGE\_TB ,STAGE\_BYTES/(1024\*1024\*1024\*1024) AS STAGE\_TB,FAILSAFE\_BYTES/(1024\*1024\*1024\*1024) AS FAILSAFE\_TB from SNOWFLAKE.ACCOUNT\_USAGE.STORAGE\_USAGE;"

var accCommand2="Truncate table ACCOUNT\_STORAGE;"

var accCommand3="Insert into ACCOUNT\_STORAGE select USAGE\_DATE,STORAGE\_BYTES/(1024\*1024\*1024\*1024) AS STORAGE\_TB ,STAGE\_BYTES/(1024\*1024\*1024\*1024) AS STAGE\_TB,FAILSAFE\_BYTES/(1024\*1024\*1024\*1024) AS FAILSAFE\_TB from SNOWFLAKE.ACCOUNT\_USAGE.STORAGE\_USAGE;"

var dbCommand1="CREATE TRANSIENT TABLE if not exists DB\_STORAGE as select DATABASE\_NAME,USAGE\_DATE,AVERAGE\_DATABASE\_BYTES/(1024\*1024\*1024\*1024) AS AVERAGE\_DATABASE\_TB ,AVERAGE\_FAILSAFE\_BYTES/(1024\*1024\*1024\*1024) AS AVERAGE\_FAILSAFE\_TB from SNOWFLAKE.ACCOUNT\_USAGE.DATABASE\_STORAGE\_USAGE\_HISTORY;"

var dbCommand2="Truncate table DB\_STORAGE;"

var dbCommand3="Insert into DB\_STORAGE select DATABASE\_NAME,USAGE\_DATE,AVERAGE\_DATABASE\_BYTES/(1024\*1024\*1024\*1024) AS AVERAGE\_DATABASE\_TB ,AVERAGE\_FAILSAFE\_BYTES/(1024\*1024\*1024\*1024) AS AVERAGE\_FAILSAFE\_TB from SNOWFLAKE.ACCOUNT\_USAGE.DATABASE\_STORAGE\_USAGE\_HISTORY;"

var tabCommand1="CREATE TRANSIENT TABLE IF NOT EXISTS TABLE\_AND\_SCHEMA\_STORAGE as select TABLE\_CREATED,TABLE\_NAME, TABLE\_SCHEMA, SCHEMA\_CREATED AS SCHEMA\_CREATION\_DATE, TABLE\_CATALOG AS TABLE\_DATABASE, DELETED AS IS\_TABLE\_DELETED, IS\_TRANSIENT, ACTIVE\_BYTES/(1024\*1024\*1024) AS ACTIVE\_GB,TIME\_TRAVEL\_BYTES/(1024\*1024\*1024) AS TIME\_TRAVEL\_GB,FAILSAFE\_BYTES/(1024\*1024\*1024) AS FAILSAFE\_GB,RETAINED\_FOR\_CLONE\_BYTES/(1024\*1024\*1024) AS RETAINED\_FOR\_CLONE\_GB from SNOWFLAKE.ACCOUNT\_USAGE.TABLE\_STORAGE\_METRICS;"

var tabCommand2="Truncate table TABLE\_AND\_SCHEMA\_STORAGE;"

var tabCommand3="Insert into TABLE\_AND\_SCHEMA\_STORAGE select TABLE\_CREATED,TABLE\_NAME, TABLE\_SCHEMA, SCHEMA\_CREATED AS SCHEMA\_CREATION\_DATE, TABLE\_CATALOG AS TABLE\_DATABASE, DELETED AS IS\_TABLE\_DELETED, IS\_TRANSIENT, ACTIVE\_BYTES/(1024\*1024\*1024) AS ACTIVE\_GB,TIME\_TRAVEL\_BYTES/(1024\*1024\*1024) AS TIME\_TRAVEL\_GB,FAILSAFE\_BYTES/(1024\*1024\*1024) AS FAILSAFE\_GB,RETAINED\_FOR\_CLONE\_BYTES/(1024\*1024\*1024) AS RETAINED\_FOR\_CLONE\_GB from SNOWFLAKE.ACCOUNT\_USAGE.TABLE\_STORAGE\_METRICS;"

var orgCmd\_dict1 = {sqlText: orgCommand1};

var orgCmd\_dict2 = {sqlText: orgCommand2};

var orgCmd\_dict3 = {sqlText: orgCommand3};

var accCmd\_dict1 = {sqlText: accCommand1};

var accCmd\_dict2 = {sqlText: accCommand2};

var accCmd\_dict3 = {sqlText: accCommand3};

var dbCmd\_dict1 = {sqlText: dbCommand1};

var dbCmd\_dict2 = {sqlText: dbCommand2};

var dbCmd\_dict3 = {sqlText: dbCommand3};

var tabCmd\_dict1 = {sqlText: tabCommand1};

var tabCmd\_dict2 = {sqlText: tabCommand2};

var tabCmd\_dict3 = {sqlText: tabCommand3};

var org\_stmt1 = snowflake.createStatement(orgCmd\_dict1);

var org\_rs1 = org\_stmt1.execute();

var org\_stmt2 = snowflake.createStatement(orgCmd\_dict2);

var org\_rs2 = org\_stmt2.execute();

var org\_stmt3 = snowflake.createStatement(orgCmd\_dict3);

var org\_rs3 = org\_stmt3.execute();

var acc\_stmt1 = snowflake.createStatement(accCmd\_dict1);

var acc\_rs1 = acc\_stmt1.execute();

var acc\_stmt2 = snowflake.createStatement(accCmd\_dict2);

var acc\_rs2 = acc\_stmt2.execute();

var acc\_stmt3 = snowflake.createStatement(accCmd\_dict3);

var acc\_rs3 = acc\_stmt3.execute();

var db\_stmt1 = snowflake.createStatement(dbCmd\_dict1);

var db\_rs1 = db\_stmt1.execute();

var db\_stmt2 = snowflake.createStatement(dbCmd\_dict2);

var db\_rs2 = db\_stmt2.execute();

var db\_stmt3 = snowflake.createStatement(dbCmd\_dict3);

var db\_rs3 = db\_stmt3.execute();

var tab\_stmt1 = snowflake.createStatement(tabCmd\_dict1);

var tab\_rs1 = tab\_stmt1.execute();

var tab\_stmt2 = snowflake.createStatement(tabCmd\_dict2);

var tab\_rs2 = tab\_stmt2.execute();

var tab\_stmt3 = snowflake.createStatement(tabCmd\_dict3);

var tab\_rs3 = tab\_stmt3.execute();

//CREATE TASK : CHANGE CRON EXPRESSION TO SCHEDULE SP

snowflake.execute({sqlText:`CREATE OR REPLACE TASK Monitor\_Storage\_Monitor\_Task WAREHOUSE = 'MONITOR\_WH' SCHEDULE = 'USING CRON 0 2 \* \* SUN UTC' AS call Storage\_Monitor\_SP();`});

snowflake.execute({sqlText:`alter task Monitor\_Storage\_Monitor\_Task resume;`});

return 'TRANSIENT TABLES CREATED FOR STORAGE COST MONITORING. TABLES NAME : 1.ORG\_STORAGE\_COST 2.ACCOUNT\_STORAGE\_COST 3.DB\_STORAGE\_COST 4.TABLE\_STORAGE\_COST';

$$;

**Note :** Call the storage monitor once to create all the tables and to create the task that would call the stored procedure Storage\_monitor\_SP() every day.

call Storage\_Monitor\_SP();

**Note :** Alter the task Monitor\_Storage\_Monitor\_Task in order to schedule the calling of the stored procedure Storage\_Monitor\_SP() in order to refresh the table as per the user requirements.

# 

# **3. Compute Monitor Queries**

**Note : Flow of execution :**

Step 1 : Execute **3.1 Snowflake Prerequisite** Queries.

Step 2 : Execute **3.3 Table Creation** Queries.

Step 3 : Execute **3.4 Stored Procedures and tasks to update above table** Queries.

## **3.1 Snowflake Prerequisite Query**

Please refer **Section 1.1 Snowflake Prerequisite Query** and execute those queries if you have not already executed them before you move on to the next steps

## **3.2 List of Tables Used**

| **Table Name** | **Dashboards** | **Latency** |
| --- | --- | --- |
| TABLE\_ROLE\_MONITOR | Detailed Compute Credit Analysis | Up to 45 minutes |
| TABLE\_USER\_MONITOR | Detailed Compute Credit Analysis | Up to 45 minutes |
| TABLE\_WAREHOUSE\_MONITOR | Compute Credit Overview,  Compute Credit Forecasting | Up To 180 minutes |
| TABLE\_PIPE\_MONITOR | Detailed Compute Credit Analysis | Up To 180 minutes |
| TABLE\_SERVERLESSTASK\_MONITOR | Detailed Compute Credit Analysis | Up To 180 minutes |
| TABLE\_REPLICATION\_MONITOR | Detailed Compute Credit Analysis | Up To 180 minutes |
| TABLE\_AUTOMATICCLUSTERING\_MONITOR | Detailed Compute Credit Analysis | Up To 180 minutes |
| TABLE\_MATERIALIZEDVIEWREFRESH\_MONITOR | Detailed Compute Credit Analysis | Up To 180 minutes |
| TABLE\_SEARCHOPTIMIZATION\_MONITOR | Detailed Compute Credit Analysis | Up To 180 minutes |

## 

## 

## **3.3 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 queries.

**QUERY :**

USE role MONITOR\_ADMIN;

USE warehouse MONITOR\_WH;

USE database MONITOR\_DB;

CREATE schema COMPUTE\_CREDIT\_MONITOR\_SCHEMA;

Use schema COMPUTE\_CREDIT\_MONITOR\_SCHEMA;

### **3.3.1 TABLE\_ROLE\_MONITOR**

**Note** : This table uses the view “snowflake”.”account\_usage”.”query\_history” and as per the snowflake documentation, Latency for the view may be up to 45 minutes.

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

create or replace TRANSIENT TABLE TABLE\_ROLE\_MONITOR (

DATE DATETIME,

ROLE\_NAME VARCHAR(16777216),

USER\_NAME VARCHAR(16777216),

WAREHOUSE\_NAME VARCHAR(16777216),

STMT\_CNT NUMBER(18,0),

ESTIMATED\_CREDITS NUMBER(38,12));

### **3.3.2 TABLE\_USER\_MONITOR**

**Note** : This table uses the view “snowflake”.”account\_usage”.”query\_history” and as per the snowflake documentation, Latency for the view may be up to 45 minutes. Refresh Rate depends upon the scheduled time for the task credit\_user\_task which calls the stored procedure for inserting the latest data to the table.

create or replace TRANSIENT TABLE TABLE\_USER\_MONITOR (

DATE DATETIME,

USER\_NAME VARCHAR(16777216),

STMT\_CNT NUMBER(18,0),

ESTIMATED\_CREDITS NUMBER(38,12));

### **3.3.3 TABLE\_WAREHOUSE\_MONITOR**

**Note :** This table uses the view “snowflake”.”account\_usage”.”warehouse\_metering\_history” and as per the snowflake documentation, Latency for the view may be up to 180 minutes (3 hours), except for the CREDITS\_USED\_CLOUD\_SERVICES column. Latency for CREDITS\_USED\_CLOUD\_SERVICES may be up to 6 hours.Refresh Rate depends upon the scheduled time for the task credit\_warehouse\_task which calls the stored procedure for inserting the latest data to the table.

create or replace TRANSIENT TABLE TABLE\_WAREHOUSE\_MONITOR (

WAREHOUSE\_NAME VARCHAR(16777216),

START\_DATE DATETIME,

DATE DATE,

DAILY\_CREDITS\_USED NUMBER(38,9),

DAILY\_CREDITS\_USED\_COMPUTE NUMBER(38,9),

DAILY\_CREDITS\_USED\_CLOUD NUMBER(38,9)

);

### 

### **3.3.4 TABLE\_PIPE\_MONITOR**

**Note :** This table uses the view “snowflake”.”account\_usage”.”pipe\_usage\_history” and as per the snowflake documentation, Latency for the view may be up to 180 minutes (3 hours). Refresh Rate depends upon the scheduled time for the task credit\_pipe\_task which calls the stored procedure for inserting the latest data to the table.

create or replace TRANSIENT TABLE TABLE\_PIPE\_MONITOR (

DATE DATETIME,

PIPE\_NAME VARCHAR(16777216),

DAILY\_CREDITS\_USED NUMBER(38,9)

);

### **3.3.5 TABLE\_SERVERLESSTASK\_MONITOR**

**Note :** This table uses the view “snowflake”.”account\_usage”.”serverless\_task\_history” and as per the snowflake documentation, Latency for the view may be up to 180 minutes (3 hours). Refresh Rate depends upon the scheduled time for the task credit\_stask\_task which calls the stored procedure for inserting the latest data to the table.

create or replace TRANSIENT TABLE TABLE\_SERVERLESSTASK\_MONITOR (

DATE DATETIME,

TASK\_NAME VARCHAR(16777216),

DAILY\_CREDITS\_USED NUMBER(38,9)

);

### **3.3.6 TABLE\_REPLICATION\_MONITOR**

**Note :** This table uses the view “snowflake”.”account\_usage”.”replication\_usage\_history” and as per the snowflake documentation, Latency for the view may be up to 180 minutes (3 hours). Refresh Rate depends upon the scheduled time for the task credit\_replication\_task which calls the stored procedure for inserting the latest data to the table.

create or replace TRANSIENT TABLE TABLE\_REPLICATION\_MONITOR (

DATE DATETIME,

DATABASE\_NAME VARCHAR(16777216),

DAILY\_CREDITS\_USED NUMBER(38,9)

);

### **3.3.7 TABLE\_AUTOMATICCLUSTERING\_MONITOR**

**Note :** This table uses the view “snowflake”.”account\_usage”.”automatic\_clustering\_history” and as per the snowflake documentation, Latency for the view may be up to 180 minutes (3 hours). Refresh Rate depends upon the scheduled time for the task credit\_autocluster\_task which calls the stored procedure for inserting the latest data to the table.

create or replace TRANSIENT TABLE TABLE\_AUTOMATICCLUSTERING\_MONITOR (

DATE DATETIME,

DATABASE\_NAME VARCHAR(16777216),

TABLE\_NAME VARCHAR(16777216),

DAILY\_CREDITS\_USED NUMBER(38,9)

);

### **3.3.8 TABLE\_MATERIALIZEDVIEWREFRESH\_MONITOR**

**Note :** This table uses the view “snowflake”.”account\_usage”.”materialized\_view\_refresh\_history” and as per the snowflake documentation, Latency for the view may be up to 180 minutes (3 hours). Refresh Rate depends upon the scheduled time for the task credit\_mview\_task which calls the stored procedure for inserting the latest data to the table.

create or replace TRANSIENT TABLE TABLE\_MATERIALIZEDVIEWREFRESH\_MONITOR (

DATE DATETIME,

DATABASE\_NAME VARCHAR(16777216),

TABLE\_NAME VARCHAR(16777216),

DAILY\_CREDITS\_USED NUMBER(38,9)

);

### **3.3.9 TABLE\_SEARCHOPTIMIZATION\_MONITOR**

**Note :** This table uses the view “snowflake”.”account\_usage”.”search\_optimization\_history” and as per the snowflake documentation, Latency for the view may be up to 180 minutes (3 hours). Refresh Rate depends upon the scheduled time for the task credit\_SEARCHOPTIMIZATION\_task which calls the stored procedure for inserting the latest data to the table.

create or replace TRANSIENT TABLE TABLE\_SEARCHOPTIMIZATION\_MONITOR (

DATE DATETIME,

DATABASE\_NAME VARCHAR(16777216),

TABLE\_NAME VARCHAR(16777216),

DAILY\_CREDITS\_USED NUMBER(38,9)

);

## **3.4 STORED PROCEDURES and TASKS to update above TABLES**

Note: All these stored procedures are used for incremental load. Inorder to insert all previous(historical) records you can run the following insert statements written in section **3.4.1**

### **3.4.1 To Load all the previous records :**

Run the below statements to insert all the previous records to the custom tables.

#### **3.4.1.1 Table\_User\_Monitor :**

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA".TABLE\_USER\_MONITOR (DATE ,

USER\_NAME ,

STMT\_CNT ,

ESTIMATED\_CREDITS)

Select convert\_timezone('UTC', START\_TIME)::datetime as date,

user\_name,

count(\*) as stmt\_cnt,

sum(execution\_time/1000 \*

case warehouse\_size

when 'X-Small' then 1/60/60

when 'Small' then 2/60/60

when 'Medium' then 4/60/60

when 'Large' then 8/60/60

when 'X-Large' then 16/60/60

when '2X-Large' then 32/60/60

when '3X-Large' then 64/60/60

when '4X-Large' then 128/60/60

else 0

end) as estimated\_credits

from snowflake.account\_usage.query\_history

group by 1,2

order by 1 desc,4 desc,2 ;

#### **3.4.1.2 Table\_Role\_Monitor :**

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA".TABLE\_ROLE\_MONITOR(DATE ,ROLE\_NAME ,USER\_NAME ,WAREHOUSE\_NAME ,STMT\_CNT ,ESTIMATED\_CREDITS)

select convert\_timezone('UTC', start\_time)::datetime as date,role\_name,user\_name,warehouse\_name ,count(\*) as stmt\_cnt,

sum(execution\_time/1000 \*

case warehouse\_size

when 'X-Small' then 1/60/60

when 'Small' then 2/60/60

when 'Medium' then 4/60/60

when 'Large' then 8/60/60

when 'X-Large' then 16/60/60

when '2X-Large' then 32/60/60

when '3X-Large' then 64/60/60

when '4X-Large' then 128/60/60

else 0

end) as estimated\_credits

from snowflake.account\_usage.query\_history

group by 1,2,3,4

order by 1 desc,4 desc,2;

#### **3.4.1.3 Table\_Warehouse\_Monitor :**

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA".TABLE\_WAREHOUSE\_MONITOR

(WAREHOUSE\_NAME,START\_DATE,DATE,DAILY\_CREDITS\_USED, DAILY\_CREDITS\_USED\_COMPUTE, DAILY\_CREDITS\_USED\_CLOUD)

SELECT WAREHOUSE\_NAME,

START\_TIME as START\_DATE,

convert\_timezone('UTC', a.start\_time)::datetime as date,

SUM(CREDITS\_USED) AS DAILY\_CREDITS\_USED,

SUM(CREDITS\_USED\_COMPUTE) AS DAILY\_CREDITS\_USED\_COMPUTE,

SUM(CREDITS\_USED\_CLOUD\_SERVICES) AS DAILY\_CREDITS\_USED\_CLOUD

FROM SNOWFLAKE.ACCOUNT\_USAGE.WAREHOUSE\_METERING\_HISTORY AS a

GROUP BY a.WAREHOUSE\_NAME, a.START\_TIME;

#### **3.4.1.4 Table\_Pipe\_Monitor :**

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA".TABLE\_PIPE\_MONITOR(DATE, PIPE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

pipe\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."PIPE\_USAGE\_HISTORY"

group by 1,2

order by 1 desc,3 desc,2;

#### **3.4.1.5 Table\_Serverlesstask\_Monitor :**

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_SERVERLESSTASK\_MONITOR"(DATE, TASK\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

task\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."SERVERLESS\_TASK\_HISTORY"

group by 1,2

order by 1 desc,3 desc,2;

#### **3.4.1.6 Table\_Replication\_Monitor :**

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_REPLICATION\_MONITOR"(DATE, DATABASE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

database\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."REPLICATION\_USAGE\_HISTORY"

group by 1,2

order by 1 desc,3 desc,2;

#### **3.4.1.7 Table\_Materializedviewrefresh\_Monitor:**

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_MATERIALIZEDVIEWREFRESH\_MONITOR"(DATE, DATABASE\_NAME,TABLE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

database\_name,

table\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."MATERIALIZED\_VIEW\_REFRESH\_HISTORY"

group by 1,2,3

order by 1 desc,3 desc,2;

#### **3.4.1.8 Table\_AutomaticClustering\_Monitor :**

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_AUTOMATICCLUSTERING\_MONITOR"(DATE, DATABASE\_NAME,TABLE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

database\_name,

table\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."AUTOMATIC\_CLUSTERING\_HISTORY"

group by 1,2,3

order by 1 desc,3 desc,2;

#### **3.4.1.9 Table\_Searchoptimization\_Monitor :**

#### 

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_SEARCHOPTIMIZATION\_MONITOR"(DATE, DATABASE\_NAME,TABLE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

database\_name,

table\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."SEARCH\_OPTIMIZATION\_HISTORY"

group by 1,2,3

order by 1 desc,3 desc,2;

**Note** : Run the stored procedures for performing incremental load into the custom tables.

### **3.4.2 Stored procedure - SP\_USER\_MONITOR()**

--Stored procedure to update custom table table\_user\_monitor

CREATE OR REPLACE PROCEDURE SP\_USER\_MONITOR()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

snowflake.execute( {sqlText: `

delete from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_USER\_MONITOR

where date = (Select max(date)

from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_USER\_MONITOR);`} );

var sql\_command = `

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA".TABLE\_USER\_MONITOR (DATE ,

USER\_NAME ,

STMT\_CNT ,

ESTIMATED\_CREDITS)

Select convert\_timezone('UTC', START\_TIME)::datetime as date,

user\_name,

count(\*) as stmt\_cnt,

sum(execution\_time/1000 \*

case warehouse\_size

when 'X-Small' then 1/60/60

when 'Small' then 2/60/60

when 'Medium' then 4/60/60

when 'Large' then 8/60/60

when 'X-Large' then 16/60/60

when '2X-Large' then 32/60/60

when '3X-Large' then 64/60/60

when '4X-Large' then 128/60/60

else 0

end) as estimated\_credits

from snowflake.account\_usage.query\_history

WHERE convert\_timezone('UTC', DATE) > (select max(convert\_timezone('UTC', DATE)) from monitor\_db.compute\_credit\_monitor\_schema.table\_user\_monitor)

group by 1,2

order by 1 desc,4 desc,2;`;

try {

snowflake.execute({sqlText: sql\_command});

return "Success";

}

catch (err) {

return "Failed" + err;

}

$$

;

call SP\_USER\_MONITOR();

**Note :** Alter the task credit\_user\_task in order to schedule the calling of the stored procedure SP\_USER\_MONITOR() as per the user requirements.

--Task to call SP\_USER\_MONITOR()

create or replace task credit\_user\_task

warehouse = MONITOR\_WH

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

as

call SP\_USER\_MONITOR();

ALTER TASK credit\_user\_task resume;

show tasks;

### **3.4.3 Stored procedure - SP\_ROLE\_MONITOR()**

--Stored procedure to update custom table table\_role\_monitor

CREATE OR REPLACE PROCEDURE SP\_ROLE\_MONITOR()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

snowflake.execute( {sqlText: `

delete from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_ROLE\_MONITOR

where date = (Select max(date)

from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_ROLE\_MONITOR);`} );

var sql\_command = `

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA".TABLE\_ROLE\_MONITOR(DATE ,ROLE\_NAME ,USER\_NAME ,WAREHOUSE\_NAME ,STMT\_CNT ,ESTIMATED\_CREDITS)

select convert\_timezone('UTC', start\_time)::datetime as date,ROLE\_NAME,user\_name,warehouse\_name ,count(\*) as stmt\_cnt,

sum(execution\_time/1000 \*

case warehouse\_size

when 'X-Small' then 1/60/60

when 'Small' then 2/60/60

when 'Medium' then 4/60/60

when 'Large' then 8/60/60

when 'X-Large' then 16/60/60

when '2X-Large' then 32/60/60

when '3X-Large' then 64/60/60

when '4X-Large' then 128/60/60

else 0

end) as estimated\_credits

from snowflake.account\_usage.query\_history

WHERE convert\_timezone('UTC', DATE) > (select max(convert\_timezone('UTC', DATE)) from monitor\_db.compute\_credit\_monitor\_schema.table\_role\_monitor)

group by 1,2,3,4

order by 1 desc,4 desc,2;`;

try {

snowflake.execute({sqlText: sql\_command});

return "Success";

}

catch (err) {

return "Failed" + err;

}

$$

;

call SP\_ROLE\_MONITOR();

**Note :** Alter the task credit\_role\_task in order to schedule the calling of the stored procedure SP\_ROLE\_MONITOR() in order to change the Refresh Rate of the tables as per the user requirements

--Task to call SP\_ROLE\_MONITOR()

create or replace task credit\_role\_task

warehouse = MONITOR\_WH

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

as

call SP\_ROLE\_MONITOR();

ALTER TASK credit\_role\_task resume;

### **3.4.4 Stored procedure - SP\_WAREHOUSE\_MONITOR()**

--Stored procedure to update custom table table\_warehouse\_monitor

CREATE OR REPLACE PROCEDURE SP\_WAREHOUSE\_MONITOR()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

snowflake.execute( {sqlText: `

delete from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_WAREHOUSE\_MONITOR

where start\_date = (Select max(start\_date)

from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_WAREHOUSE\_MONITOR);`} );

var sql\_command = `

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA".TABLE\_WAREHOUSE\_MONITOR

(WAREHOUSE\_NAME,START\_DATE,DATE,DAILY\_CREDITS\_USED, DAILY\_CREDITS\_USED\_COMPUTE, DAILY\_CREDITS\_USED\_CLOUD)

SELECT WAREHOUSE\_NAME,

START\_TIME as START\_DATE,

convert\_timezone('UTC', a.start\_time)::datetime as date,

SUM(CREDITS\_USED) AS DAILY\_CREDITS\_USED,

SUM(CREDITS\_USED\_COMPUTE) AS DAILY\_CREDITS\_USED\_COMPUTE,

SUM(CREDITS\_USED\_CLOUD\_SERVICES) AS DAILY\_CREDITS\_USED\_CLOUD

FROM SNOWFLAKE.ACCOUNT\_USAGE.WAREHOUSE\_METERING\_HISTORY AS a

WHERE convert\_timezone('UTC', start\_date) > (select max(convert\_timezone('UTC', start\_date)) from monitor\_db.compute\_credit\_monitor\_schema.table\_warehouse\_monitor)

GROUP BY a.WAREHOUSE\_NAME, a.START\_TIME;`;

try {

snowflake.execute({sqlText: sql\_command});

return "Success";

}

catch (err) {

return "Failed" + err;

}

$$

;

call SP\_WAREHOUSE\_MONITOR();

**Note :** Alter the task credit\_warehouse\_task in order to schedule the calling of the stored procedure SP\_WAREHOUSE\_MONITOR() as per the user requirements.

--Task to call SP\_WAREHOUSE\_MONITOR()

create or replace task credit\_warehouse\_task

warehouse = MONITOR\_WH

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

as

call SP\_WAREHOUSE\_MONITOR();

ALTER TASK credit\_warehouse\_task resume;

### 

### **3.4.5 Stored procedure - SP\_PIPE\_MONITOR()**

--Stored procedure to update custom table table\_pipe\_monitor

CREATE OR REPLACE PROCEDURE SP\_PIPE\_MONITOR()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

snowflake.execute( {sqlText: `

delete from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_PIPE\_MONITOR

where date = (Select max(date)

from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_PIPE\_MONITOR);`} );

var sql\_command = `

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA".TABLE\_PIPE\_MONITOR(DATE, PIPE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

pipe\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."PIPE\_USAGE\_HISTORY"

WHERE convert\_timezone('UTC', DATE) > (select max(convert\_timezone('UTC', DATE)) from monitor\_db.compute\_credit\_monitor\_schema.table\_pipe\_monitor)

group by 1,2

order by 1 desc,3 desc,2;

`;

try {

snowflake.execute({sqlText: sql\_command});

return "Success";

}

catch (err) {

return "Failed" + err;

}

$$

;

call SP\_PIPE\_MONITOR();

**Note :** Alter the task credit\_pipe\_task in order to schedule the calling of the stored procedure SP\_PIPE\_MONITOR() as per the user requirements.

--Task to call SP\_PIPE\_MONITOR()

create or replace task credit\_pipe\_task

warehouse = MONITOR\_WH

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

as

call SP\_PIPE\_MONITOR();

ALTER TASK credit\_pipe\_task resume;

### **3.4.6 Stored procedure - SP\_STASK\_MONITOR()**

--Stored procedure to update custom table table\_serverlesstask\_monitor

CREATE OR REPLACE PROCEDURE SP\_STASK\_MONITOR()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

snowflake.execute( {sqlText: `

delete from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_SERVERLESSTASK\_MONITOR

where date = (Select max(date)

from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_SERVERLESSTASK\_MONITOR);`} );

var sql\_command = `

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_SERVERLESSTASK\_MONITOR"(DATE, TASK\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

task\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."SERVERLESS\_TASK\_HISTORY"

WHERE convert\_timezone('UTC', DATE) > (select max(convert\_timezone('UTC', DATE)) from monitor\_db.compute\_credit\_monitor\_schema.table\_serverlesstask\_monitor)

group by 1,2

order by 1 desc,3 desc,2;

`;

try {

snowflake.execute({sqlText: sql\_command});

return "Success";

}

catch (err) {

return "Failed" + err;

}

$$

;

call SP\_STASK\_MONITOR();

**Note :** Alter the task credit\_stask\_task in order to schedule the calling of the stored procedure SP\_STASK\_MONITOR() as per the user requirements.

--Task to call SP\_STASK\_MONITOR()

create or replace task credit\_stask\_task

warehouse = MONITOR\_WH

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

as

call SP\_STASK\_MONITOR();

ALTER TASK credit\_stask\_task resume;

### **3.4.7 Stored procedure - SP\_REPLICATION\_MONITOR()**

--Stored procedure to update custom table\_replication\_monitor

CREATE OR REPLACE PROCEDURE SP\_REPLICATION\_MONITOR()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

snowflake.execute( {sqlText: `

delete from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_REPLICATION\_MONITOR

where date = (Select max(date)

from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_REPLICATION\_MONITOR);`} );

var sql\_command = `

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_REPLICATION\_MONITOR"(DATE, DATABASE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

database\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."REPLICATION\_USAGE\_HISTORY"

WHERE convert\_timezone('UTC', DATE) > (select max(convert\_timezone('UTC', DATE)) from monitor\_db.compute\_credit\_monitor\_schema.table\_replication\_monitor)

group by 1,2

order by 1 desc,3 desc,2;

`;

try {

snowflake.execute({sqlText: sql\_command});

return "Success";

}

catch (err) {

return "Failed" + err;

}

$$;

call SP\_REPLICATION\_MONITOR();

**Note :** Alter the task credit\_replication\_task in order to schedule the calling of the stored procedure SP\_REPLICATION\_MONITOR() as per the user requirements.

--Task to call SP\_REPLICATION\_MONITOR();

create or replace task credit\_replication\_task

warehouse = MONITOR\_WH

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

as

call SP\_REPLICATION\_MONITOR();

ALTER TASK credit\_replication\_task resume;

### **3.4.8 Stored procedure - SP\_MVIEW\_MONITOR()**

--Stored procedure to update custom table table\_materializedviewrefresh\_monitor

CREATE OR REPLACE PROCEDURE SP\_MVIEW\_MONITOR()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

snowflake.execute( {sqlText: `

delete from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_MATERIALIZEDVIEWREFRESH\_MONITOR

where date = (Select max(date)

from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_MATERIALIZEDVIEWREFRESH\_MONITOR);`} );

var sql\_command = `

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_MATERIALIZEDVIEWREFRESH\_MONITOR"(DATE, DATABASE\_NAME,TABLE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

database\_name,

table\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."MATERIALIZED\_VIEW\_REFRESH\_HISTORY"

WHERE convert\_timezone('UTC', DATE) > (select max(convert\_timezone('UTC', DATE)) from monitor\_db.compute\_credit\_monitor\_schema.table\_materializedviewrefresh\_monitor)

group by 1,2,3

order by 1 desc,3 desc,2;

`;

try {

snowflake.execute({sqlText: sql\_command});

return "Success";

}

catch (err) {

return "Failed" + err;

}

$$

;

call SP\_MVIEW\_MONITOR();

**Note :** Alter the task credit\_mview\_task in order to schedule the calling of the stored procedure SP\_MVIEW\_MONITOR() as per the user requirements.

--Task to call SP\_MVIEW\_MONITOR()

create or replace task credit\_mview\_task

warehouse = MONITOR\_WH

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

as

call SP\_MVIEW\_MONITOR();

ALTER TASK credit\_mview\_task resume;

### **3.4.9 Stored procedure - SP\_AUTOCLUSTER\_MONITOR()**

--Stored procedure to update custom table table\_automaticclustering\_monitor

CREATE OR REPLACE PROCEDURE SP\_AUTOCLUSTER\_MONITOR()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

snowflake.execute( {sqlText: `

delete from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_AUTOMATICCLUSTERING\_MONITOR

where date = (Select max(date)

from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_AUTOMATICCLUSTERING\_MONITOR);`} );

var sql\_command = `

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_AUTOMATICCLUSTERING\_MONITOR"(DATE, DATABASE\_NAME,TABLE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

database\_name,

table\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."AUTOMATIC\_CLUSTERING\_HISTORY"

WHERE convert\_timezone('UTC', DATE) > (select max(convert\_timezone('UTC', DATE)) from monitor\_db.compute\_credit\_monitor\_schema.table\_automaticclustering\_monitor)

group by 1,2,3

order by 1 desc,3 desc,2;

`;

try {

snowflake.execute({sqlText: sql\_command});

return "Success";

}

catch (err) {

return "Failed" + err;

}

$$

;

call SP\_AUTOCLUSTER\_MONITOR();

**Note :** Alter the task credit\_autocluster\_task in order to schedule the calling of the stored procedure SP\_autocluster\_MONITOR() as per the user requirements.

--Task to call SP\_autocluster\_MONITOR()

create or replace task credit\_autocluster\_task

warehouse = MONITOR\_WH

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

as

call SP\_AUTOCLUSTER\_MONITOR();

ALTER TASK credit\_autocluster\_task resume;

### **3.4.10 Stored procedure - SP\_SEARCHOPTIMIZATION\_MONITOR()**

--Stored procedure to update custom table table\_searchoptimization\_monitor

CREATE OR REPLACE PROCEDURE SP\_SEARCHOPTIMIZATION\_MONITOR()

RETURNS STRING

LANGUAGE JAVASCRIPT

AS

$$

snowflake.execute( {sqlText: `

delete from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_SEARCHOPTIMIZATION\_MONITOR

where date = (Select max(date)

from MONITOR\_DB.COMPUTE\_CREDIT\_MONITOR\_SCHEMA.TABLE\_SEARCHOPTIMIZATION\_MONITOR);`} );

var sql\_command = `

insert into "MONITOR\_DB"."COMPUTE\_CREDIT\_MONITOR\_SCHEMA"."TABLE\_SEARCHOPTIMIZATION\_MONITOR"(DATE, DATABASE\_NAME,TABLE\_NAME, DAILY\_CREDITS\_USED)

select convert\_timezone('UTC', start\_time)::datetime as date,

database\_name,

table\_name,

sum(credits\_used) as DAILY\_CREDITS\_USED

from "SNOWFLAKE"."ACCOUNT\_USAGE"."SEARCH\_OPTIMIZATION\_HISTORY"

WHERE convert\_timezone('UTC', DATE) > (select max(convert\_timezone('UTC', DATE)) from monitor\_db.compute\_credit\_monitor\_schema.table\_searchoptimization\_monitor)

group by 1,2,3

order by 1 desc,3 desc,2;

`;

try {

snowflake.execute({sqlText: sql\_command});

return "Success";

}

catch (err) {

return "Failed" + err;

}

$$

;

call SP\_SEARCHOPTIMIZATION\_MONITOR();

**Note :** Alter the task credit\_SEARCHOPTIMIZATION\_task in order to schedule the calling of the stored procedure SP\_SEARCHOPTIMIZATION\_MONITOR() as per the user requirements.

--Task to call SP\_SEARCHOPTIMIZATION\_MONITOR()

create or replace task credit\_SEARCHOPTIMIZATION\_task

warehouse = MONITOR\_WH

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

as

call SP\_SEARCHOPTIMIZATION\_MONITOR();

ALTER TASK credit\_SEARCHOPTIMIZATION\_task resume;

# 4. Performance Monitor Queries

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

Step 1 : Execute **4.1 Snowflake Prerequisite** Queries.

Step 2 : Execute **4.3 Table Creation** Queries.

Step 3 : Execute **4.4 Stored Procedures and tasks** Queries.

## **4.1 Snowflake Prerequisite Query**

Please refer **Section 1.1 Snowflake Prerequisite Query** and execute those queries if you have not already executed them before you move on to the next steps

## **4.2 List of tables used**

| **Table Name** | **Dashboards** | **Latency** |
| --- | --- | --- |
| Timeout\_ref | Short and Long Query Analysis | Not applicable |
| bad\_query | Short and Long Query Analysis | Upto 45 minutes |
| Query\_History | Slowest running queries  User adoption  Snowflake Query Utilization  Error Tracking  Performance Overview | Upto 45 minutes |

## **4.3 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 queries.

QUERY :

USE role MONITOR\_ADMIN;

USE warehouse MONITOR\_WH;

USE database MONITOR\_DB;

CREATE schema PERFORMANCE\_MONITOR\_SCHEMA;

Use schema MONITOR\_DB.PERFORMANCE\_MONITOR\_SCHEMA;

### **4.3.1 Timeout\_ref Table**

**Create the table timeout\_ref :**

CREATE OR REPLACE TABLE timeout\_ref (

wh\_size VARCHAR(16777216),

timeout\_1 NUMBER(38,0),

timeout\_2 NUMBER(38,0),

priority\_1 array,

priority\_2 array

);

**Insert values to the table timeout\_ref :**

**Note :** The values in the timeout\_1 and timeout\_2 columns are provided in minutes.

INSERT INTO timeout\_ref(wh\_size, TIMEOUT\_1,TIMEOUT\_2)

VALUES

('X-Small',20,16),

('Small',18,14),

('Medium',15,12),

('Large',12,10),

('X-Large',12,10),

('2X-Large',10,8),

('3X-Large',8,6),

('4X-Large',8,6),

('5X-Large',5,3),

('6X-Large',5,3);

**Update the table timeout\_ref :**

UPDATE "MONITOR\_DB"."PERFORMANCE\_MONITOR\_SCHEMA"."TIMEOUT\_REF" SET PRIORITY\_1 = (select array\_construct('ACCOUNTADMIN','SYSADMIN','SECURITYADMIN'));

UPDATE "MONITOR\_DB"."PERFORMANCE\_MONITOR\_SCHEMA"."TIMEOUT\_REF" SET PRIORITY\_2 = (select array\_construct('USERADMIN','PUBLIC'));

### **4.3.2 Bad\_query\_Table**

**Note** : This table uses the view “snowflake”.”account\_usage”.”query\_history” and as per the snowflake documentation, Latency for the view may be up to 45 minutes. Refresh Rate depends upon the scheduled time for the task call\_insert\_bad\_query which calls the stored procedure for inserting the latest data to the table.

create or replace table Bad\_query(QUERY\_ID string,QUERY\_TEXT string,START\_TIME datetime,END\_TIME datetime,DATABASE\_NAME varchar,SCHEMA\_NAME varchar,WAREHOUSE\_NAME varchar,WAREHOUSE\_SIZE varchar,USER\_NAME varchar,ROLE\_NAME varchar,TOTAL\_ELAPSED\_TIME float,EXECUTION\_STATUS varchar,Performance varchar);

### **4.3.3 Query\_ History\_Table**

**Note** : This table uses the view “snowflake”.”account\_usage”.”query\_history” and as per the snowflake documentation, Latency for the view may be up to 45 minutes.

No custom table is used for query history table, we can use the query SELECT \* FROM "SNOWFLAKE"."ACCOUNT\_USAGE"."QUERY\_HISTORY”;.

## **4.4 To Load all the previous records :**

Run the below statements to insert all the previous records (historical) to the custom table.

insert into MONITOR\_DB.PERFORMANCE\_MONITOR\_SCHEMA.BAD\_QUERY

select QUERY\_ID,QUERY\_TEXT,START\_TIME,END\_TIME,DATABASE\_NAME,SCHEMA\_NAME,WAREHOUSE\_NAME,WAREHOUSE\_SIZE,USER\_NAME,ROLE\_NAME,

case when START\_TIME<END\_TIME then total\_elapsed\_time\*1.667/100000

when START\_TIME>END\_TIME then timediff(minute,START\_TIME,current\_timestamp()) end as timediff,EXECUTION\_STATUS,

case when array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_1) and timediff>=TIMEOUT\_1 or

array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_2) and timediff>=TIMEOUT\_2 or

(NOT(array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_1)) AND NOT(array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_2)) and timediff>=TIMEOUT\_2)

then 'LONG'

when array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_1) and timediff<TIMEOUT\_1 or

array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_2) and timediff<TIMEOUT\_2 or

(NOT(array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_1)) AND NOT(array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_2)) and timediff<TIMEOUT\_2) or

WAREHOUSE\_SIZE is null then 'SHORT'

end as Performance

from (select \* from SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY) a

left join "MONITOR\_DB"."PERFORMANCE\_MONITOR\_SCHEMA"."TIMEOUT\_REF" b on a.WAREHOUSE\_SIZE = b.WH\_SIZE;

**Note** : Run the stored procedures for performing incremental load into the custom tables.

## **4.5 Stored procedures and Task to update above TABLE**

#### **4.5.1 Stored procedure - Insert\_bad\_query()**

//Stored procedure to insert latest query history details into table bad\_query :

create or replace procedure insert\_bad\_query()

returns string language javascript execute as caller as

$$

snowflake.execute({sqlText: `

delete from MONITOR\_DB.PERFORMANCE\_MONITOR\_SCHEMA.BAD\_QUERY

where START\_TIME = (Select max(START\_TIME)

from MONITOR\_DB.PERFORMANCE\_MONITOR\_SCHEMA.BAD\_QUERY);`});

snowflake.execute({sqlText:

`insert into MONITOR\_DB.PERFORMANCE\_MONITOR\_SCHEMA.BAD\_QUERY

select QUERY\_ID,QUERY\_TEXT,START\_TIME,END\_TIME,DATABASE\_NAME,SCHEMA\_NAME,WAREHOUSE\_NAME,WAREHOUSE\_SIZE,USER\_NAME,ROLE\_NAME,

case when START\_TIME<END\_TIME then total\_elapsed\_time\*1.667/100000

when START\_TIME>END\_TIME then timediff(minute,START\_TIME,current\_timestamp()) end as timediff,EXECUTION\_STATUS,

case when array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_1) and timediff>=TIMEOUT\_1 or

array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_2) and timediff>=TIMEOUT\_2 or

(NOT(array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_1)) AND NOT(array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_2)) and timediff>=TIMEOUT\_2)

then 'LONG'

when array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_1) and timediff<TIMEOUT\_1 or

array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_2) and timediff<TIMEOUT\_2 or

(NOT(array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_1)) AND NOT(array\_contains(ROLE\_NAME::VARIANT,PRIORITY\_2)) and timediff<TIMEOUT\_2) or

WAREHOUSE\_SIZE is null then 'SHORT'

end as Performance

from (select \* from SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY) a

left join "MONITOR\_DB"."PERFORMANCE\_MONITOR\_SCHEMA"."TIMEOUT\_REF" b on a.WAREHOUSE\_SIZE = b.WH\_SIZE

where START\_TIME > (select max(start\_time) from MONITOR\_DB.PERFORMANCE\_MONITOR\_SCHEMA.BAD\_QUERY)`

});

return 'INSERTED'

$$;

call insert\_bad\_query();

**Note :** Alter the task call\_insert\_bad\_query in order to schedule the calling of the stored procedure call insert\_bad\_query(); as per the user requirements.

create or replace task call\_insert\_bad\_query

warehouse = MONITOR\_WH

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

As

call insert\_bad\_query();

ALTER TASK call\_insert\_bad\_query resume;

# **5. Security Monitor Queries**

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

Step 1 : Execute **5.1 Snowflake Prerequisite** Queries.

Step 2 : Execute **5.3 Table Creation** Queries.

Step 3 : Execute **5.4 Stored Procedures and tasks to update above table** Queries.

## **5****.1 Snowflake Prerequisite Query**

Please refer **Section 1.1 Snowflake Prerequisite Query** and execute those queries if you have not already executed them before you move on to the next steps

## **5.2 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 |

## **5.3 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;

### 

### **5.3.1 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';

### **5.3.2 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';

### **5.3.2 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)

);

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

### **5.4.1 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;

### **5.4.2 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;

### **5.4.3 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.