# **SHORT AND LONG QUERY ANALYSIS MONITOR QUERY DEVELOPMENT**

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

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

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

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

## **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

## **List of tables used**

| **Table Name** | **Dashboards** | **Latency** |
| --- | --- | --- |
| Timeout\_ref | Short and Long Query Analysis | Not applicable |
| Long\_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 |

## **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;

### **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'));

### **Long\_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\_long\_query which calls the stored procedure for inserting the latest data to the table.

create or replace table Long\_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);

### **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”;.

## **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.LONG\_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.

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

#### **Stored procedure - Insert\_long\_query()**

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

create or replace procedure insert\_long\_query()

returns string language javascript execute as caller as

$$

snowflake.execute({sqlText: `

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

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

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

snowflake.execute({sqlText:

`insert into MONITOR\_DB.PERFORMANCE\_MONITOR\_SCHEMA.LONG\_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.LONG\_QUERY)`

});

return 'INSERTED'

$$;

call insert\_long\_query();

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

create or replace task call\_insert\_long\_query

warehouse = MONITOR\_WH

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

As

call insert\_long\_query();

ALTER TASK call\_insert\_long\_query 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.
* For optimal performance in analyzing Short and Long queries dashboard, it is advisable to store only the most recent three months (or less) of query history data in the Long\_query table.