**MATILLION - SECURITY MONITORING DASHBOARD**

Matillion security monitoring dashboard is used to track user configuration data, Audit logs, Permission groups of the users. Analysis of security-related Audit logs can help to identify violations of security policies and potential threats to an organization's security. Analysis of users assigned to different Permission groups can help organizations to more efficiently assign appropriate privileges.

**Create Tables in Snowflake To Store Matillion Security/User Related Data:**

USE ROLE MATILLION\_MONITOR\_ADMIN;

//-------------AUDIT TABLE----------------

CREATE or REPLACE TABLE MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.AUDIT (

id int,

specifier varchar(255),

commandType varchar(255),

user varchar(255),

timestamp datetime

);

SELECT \* FROM AUDIT;

//-------------USERCONFIG TABLE----------------

CREATE OR REPLACE TABLE MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.USERCONFIG(

TYPE VARCHAR(255),

USERNAME VARCHAR(255),

ISSERVERADMIN BOOLEAN,

ISAPI BOOLEAN,

ISPROJECTADMIN BOOLEAN,

ISREADONLY BOOLEAN

);

//-------------USERCONFIG\_1 TABLE----------------

CREATE OR REPLACE TABLE MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.USERCONFIG\_1(

TYPE VARCHAR(255),

USERNAME VARCHAR(255),

ROLE\_ASSIGNED VARCHAR(255),

isOpenIDConfigured BOOLEAN

);

//-------------OAUTH TABLE----------------

CREATE OR REPLACE TABLE MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.OAUTH(

OAUTH\_NAME VARCHAR(255)

);

//-------------PERMISSION\_GROUPS TABLE----------------

CREATE OR REPLACE TABLE MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.PERMISSION\_GROUPS(

PERMISSION\_GROUP VARCHAR(255)

);

//-------------USER\_PERMISSION\_GROUPS TABLE----------------

CREATE OR REPLACE TABLE MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.USER\_PERMISSION\_GROUPS(

USERNAME VARCHAR(255),

PERMISSION\_GROUP VARCHAR(255));

//-------------AUDIT\_NEW & AUDIT\_STREAM TABLE---------------------------------

CREATE or REPLACE TABLE MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.AUDIT\_NEW (

id int,

specifier varchar(400),

commandType varchar(255),

user varchar(255),

timestamp datetime

);

SELECT \* FROM AUDIT\_NEW;

create or replace stream MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.AUDIT\_STREAM on TABLE MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.AUDIT

//-------------------------task to load data from stream table to audit new table-------------------

CREATE OR REPLACE TASK AUDIT\_STREAM\_TASK

WAREHOUSE = MATILLION\_MONITOR\_WH

SCHEDULE = 'USING CRON 14 15 \* \* \* Asia/Kolkata'

WHEN

SYSTEM$STREAM\_HAS\_DATA('AUDIT\_STREAM')

AS

INSERT INTO AUDIT\_NEW(id,

specifier,

commandType,

user,

timestamp) SELECT id, specifier,commandType, user, timestamp FROM AUDIT\_STREAM;

**Create Python Functions to Load Security/User Related Data into Snowflake tables:**

import snowflake.connector

import requests

import json

import datetime

# Snowflake connection details

snowflake\_user = "<user>"

snowflake\_password = "<Password>"

snowflake\_account = "<account>"

snowflake\_warehouse = "<warehouse>"

snowflake\_database = "<database>"

snowflake\_schema = "<schema>"

# Snowflake connection

conn = snowflake.connector.connect(

user=snowflake\_user,

password=snowflake\_password,

account=snowflake\_account,

warehouse=snowflake\_warehouse,

database=snowflake\_database,

schema=snowflake\_schema

)

url = "http://54.168.223.24/rest/v1/audit/export?offset=0&limit=2000"

username = "ec2-user"

password = "i-058d860b02ef898f4"

#------------------TABLE: AUDIT-----------------------

# defining the request

def DATA\_AUDIT():

r = requests.get(url, auth=(username, password))

try:

job\_runs = r.json()

except json.JSONDecodeError as e:

print(f"Error decoding JSON response: {e}")

print(f"Response content: {r.content}")

job\_runs = None

insert\_query = "INSERT INTO AUDIT (id, specifier, commandType, user, timestamp) VALUES (%s, %s, %s, %s, %s)"

if job\_runs is not None:

for job\_run in job\_runs["list"]:

# print(job\_run["id"])

id = job\_run["id"]

specifier = job\_run["specifier"]

commandType = job\_run["commandType"]

user = job\_run["user"]

timestamp\_1 = job\_run["timestamp"]

timestamp = datetime.datetime.fromtimestamp( timestamp\_1 / 1000)

conn.cursor().execute(insert\_query, (id, specifier, commandType, user, timestamp))

# Commit the changes

conn.commit()

# Close the Snowflake connection

conn.close()

#------------------TABLE: USERCONFIG-----------------------

def DATA\_USERCONFIG\_NEW():

url = "http://54.168.223.24/rest/v1/userconfig/export"

username = "ec2-user"

password = "i-058d860b02ef898f4"

# defining the request

r = requests.get(url, auth=(username, password))

# extracting JSON

json\_response = r.json()

print(json\_response)

try:

job\_runs = r.json()

except json.JSONDecodeError as e:

print(f"Error decoding JSON response: {e}")

print(f"Response content: {r.content}")

job\_runs = None

insert\_query = "INSERT INTO USERCONFIG (TYPE, USERNAME, ISSERVERADMIN, ISAPI, ISPROJECTADMIN,ISREADONLY) VALUES (%s, %s, %s, %s, %s,%s)"

insert\_query\_1 = "INSERT INTO USERCONFIG\_1 (TYPE, USERNAME, ROLE\_ASSIGNED,isOpenIDConfigured) VALUES (%s, %s, %s, %s)"

truncate\_1="Truncate table USERCONFIG"

truncate\_2="Truncate table USERCONFIG\_1"

conn.cursor().execute(truncate\_1)

conn.cursor().execute(truncate\_2)

if job\_runs is not None:

for job\_run in job\_runs["objects"]:

TYPE = job\_run["type"]

if TYPE=='Internal':

isOpenIDConfigured = job\_run["isOpenIDConfigured"]

else:

isOpenIDConfigured is False

for i in job\_run["users"]:

USERNAME = i["username"]

ISSERVERADMIN = i["isServerAdmin"]

ISAPI = i["isApi"]

ISPROJECTADMIN = i["isProjectAdmin"]

ISREADONLY = i["isReadOnly"]

if i["isServerAdmin"] is True:

ROLE\_ASSIGNED = 'ServerAdmin'

conn.cursor().execute(insert\_query\_1, (TYPE, USERNAME, ROLE\_ASSIGNED, isOpenIDConfigured))

if i["isApi"] is True:

ROLE\_ASSIGNED = 'Api'

conn.cursor().execute(insert\_query\_1, (TYPE, USERNAME, ROLE\_ASSIGNED, isOpenIDConfigured))

if i["isProjectAdmin"] is True:

ROLE\_ASSIGNED = 'ProjectAdmin'

conn.cursor().execute(insert\_query\_1, (TYPE, USERNAME, ROLE\_ASSIGNED, isOpenIDConfigured))

if i["isReadOnly"] is True:

ROLE\_ASSIGNED = 'ReadOnly'

conn.cursor().execute(insert\_query\_1, (TYPE, USERNAME, ROLE\_ASSIGNED, isOpenIDConfigured))

conn.cursor().execute(insert\_query, (TYPE, USERNAME, ISSERVERADMIN, ISAPI, ISPROJECTADMIN,ISREADONLY))

# Commit the changes

conn.commit()

# Close the Snowflake connection

conn.close()

#------------------TABLE: PERMISSION\_GROUPS-----------------------

def DATA\_GROUP\_PERMISSIONS():

url = "http://54.168.223.24/rest/v1/permission/global/group"

username = "ec2-user"

password = "i-058d860b02ef898f4"

# defining the request

r = requests.get(url, auth=(username, password))

try:

job\_runs = r.json()

except json.JSONDecodeError as e:

print(f"Error decoding JSON response: {e}")

print(f"Response content: {r.content}")

job\_runs = None

truncate\_2="Truncate table PERMISSION\_GROUPS"

conn.cursor().execute(truncate\_2)

# Store DBT job data in Snowflake

insert\_query = "INSERT INTO PERMISSION\_GROUPS (Permission\_group) VALUES (%s)"

if job\_runs is not None:

for i in job\_runs:

# print(job\_run["id"])

Permission\_group = i

conn.cursor().execute(insert\_query, (Permission\_group))

# Commit the changes

conn.commit()

# Close the Snowflake connection

conn.close()

#------------------TABLE: OAUTH------------------------

def DATA\_OAUTH():

url = "http://54.168.223.24/rest/v1/oauth"

username = "ec2-user"

password = "i-058d860b02ef898f4"

# defining the request

r = requests.get(url, auth=(username, password))

try:

job\_runs = r.json()

except json.JSONDecodeError as e:

print(f"Error decoding JSON response: {e}")

print(f"Response content: {r.content}")

job\_runs = None

truncate\_2="Truncate table OAUTH"

conn.cursor().execute(truncate\_2)

# Store DBT job data in Snowflake

insert\_query = "INSERT INTO OAUTH (OAUTH\_NAME) VALUES (%s)"

if job\_runs is not None:

for i in job\_runs:

OAUTH\_NAME = i

conn.cursor().execute(insert\_query, (OAUTH\_NAME))

# Commit the changes

conn.commit()

# Close the Snowflake connection

conn.close()

#Scheduling functions to run at 12:00 AM daily

while True:

time = datetime.datetime.now()

if time.hour==24 and time.minute==00:

print("RUN THE CODE")

# Call your function here

DATA\_AUDIT()

DATA\_USERCONFIG\_NEW()

DATA\_GROUP\_PERMISSIONS()

DATA\_OAUTH()

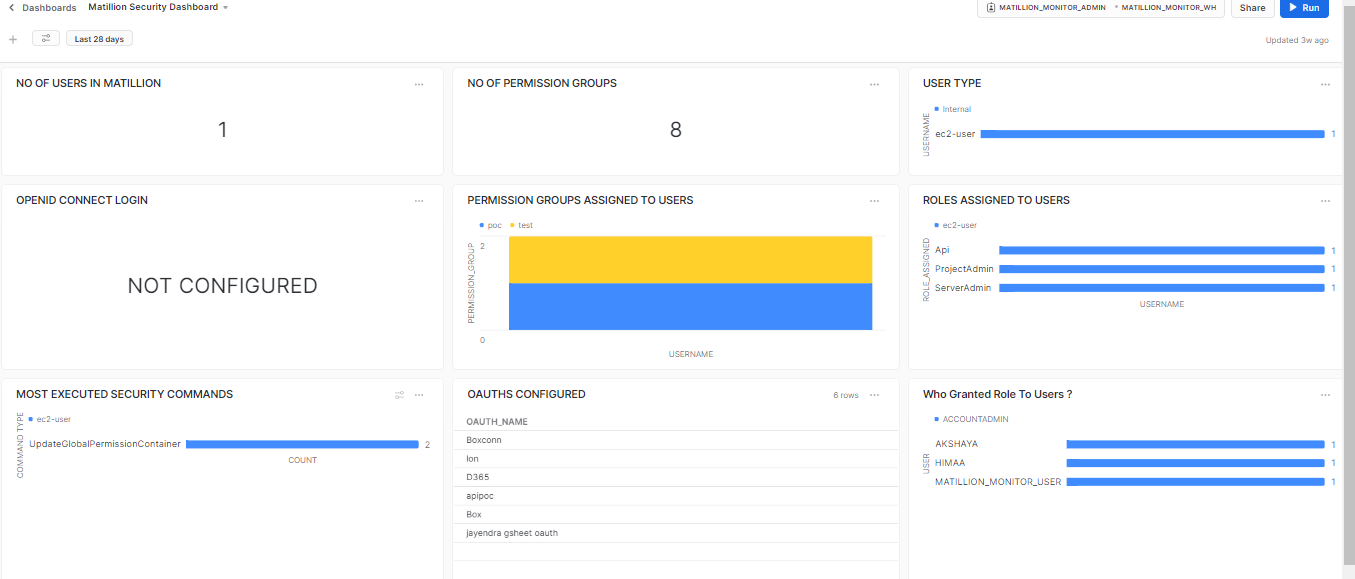
print("END")

break

**MATILLION SECURITY MONITORING DASHBOARD:**

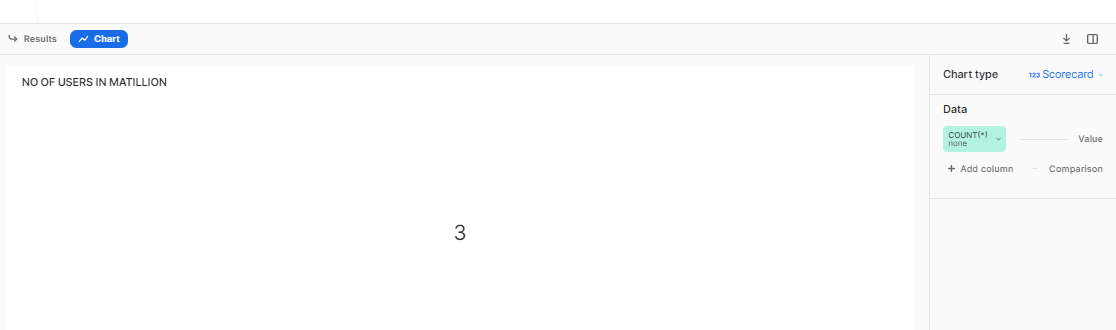
**FILTERS:**

* DATERANGE(Default Filter)



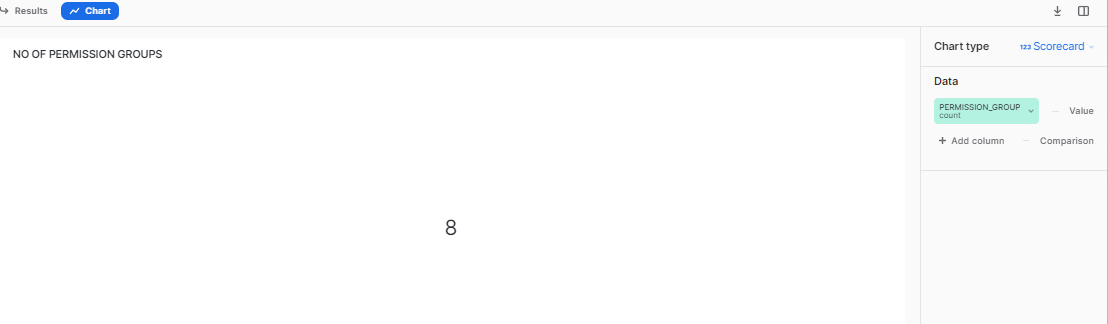
**1 NO OF USERS IN MATILLION:**

SELECT COUNT(\*) FROM MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.USERCONFIG;



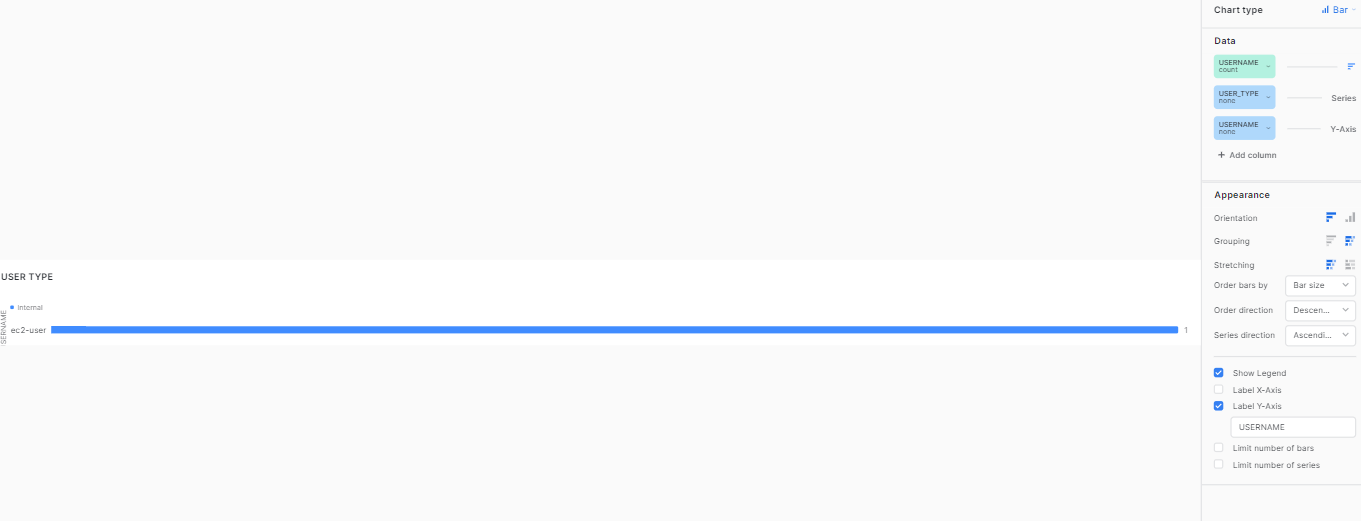
**2 NO OF PERMISSION GROUPS:**

SELECT \* FROM MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.PERMISSION\_GROUPS;



**3.3 USER TYPE:**

SELECT TYPE AS USER\_TYPE,USERNAME FROM MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.USERCONFIG;



**4 OPENID CONNECT LOGIN:**

SELECT

case

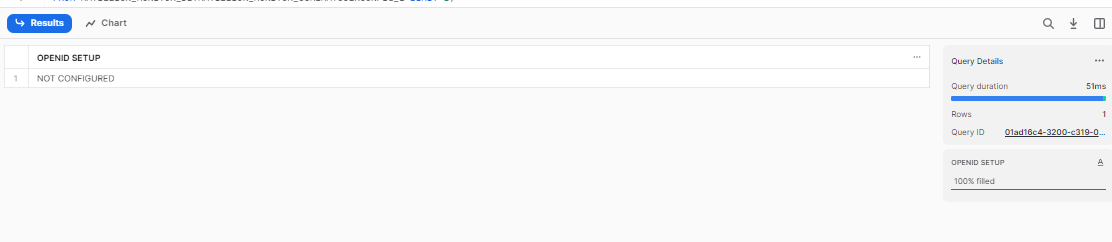
when TYPE='Internal' AND ISOPENIDCONFIGURED ='FALSE' THEN 'NOT CONFIGURED'

WHEN TYPE='Internal' AND ISOPENIDCONFIGURED ='TRUE' THEN 'CONFIGRUED'

WHEN TYPE ='External' THEN 'NOT POSSIBLE FOR EXTERNAL USERS'

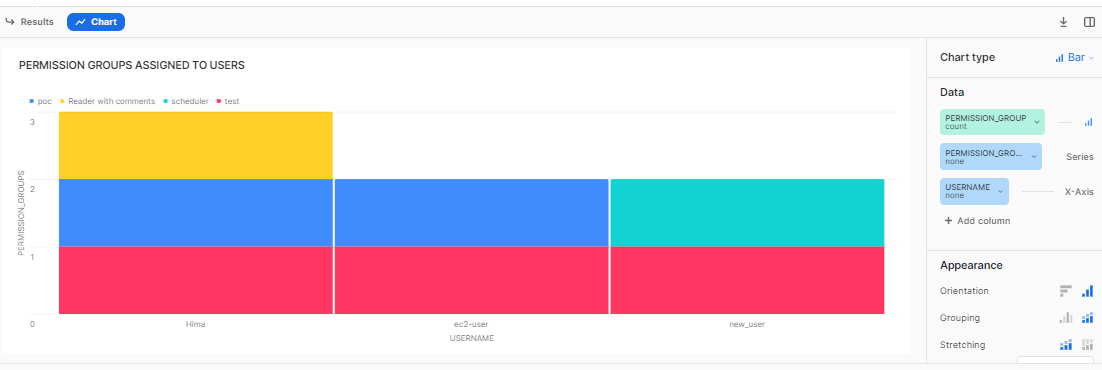
END AS "OPENID SETUP"

FROM MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.USERCONFIG\_1 LIMIT 1;



**5 PERMISSION GROUPS ASSIGNED TO USERS:**

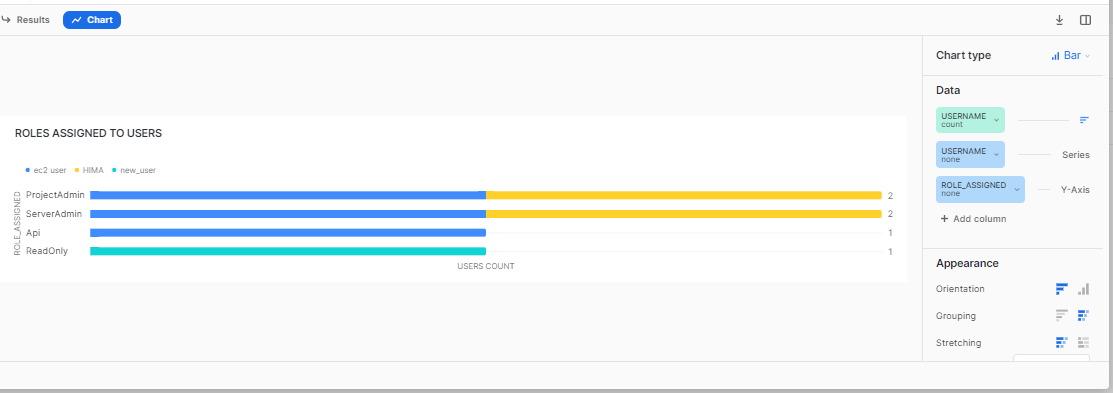
SELECT \* FROM MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.USER\_PERMISSION\_GROUPS;

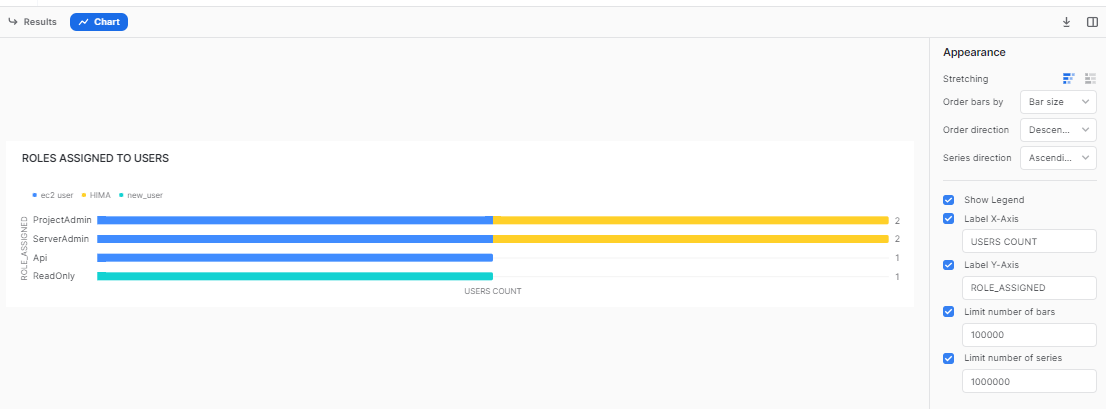




**6 ROLES ASSIGNED TO USERS:**

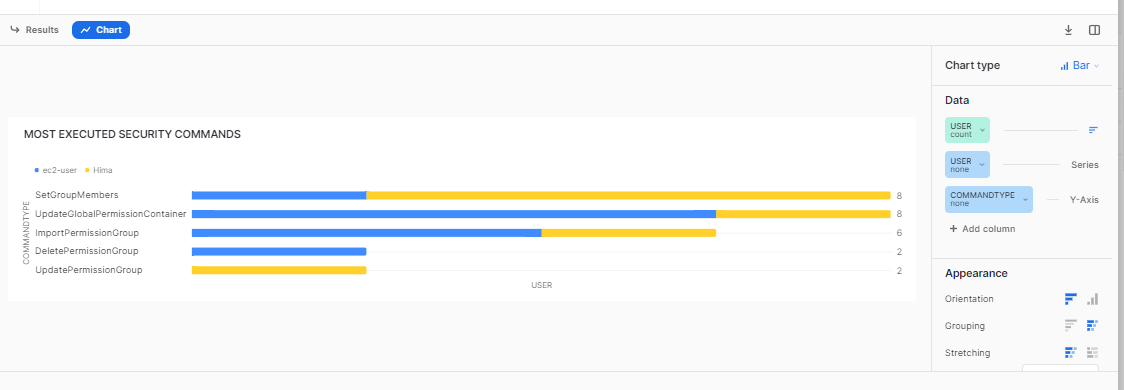
SELECT \* FROM MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.USERCONFIG\_1;

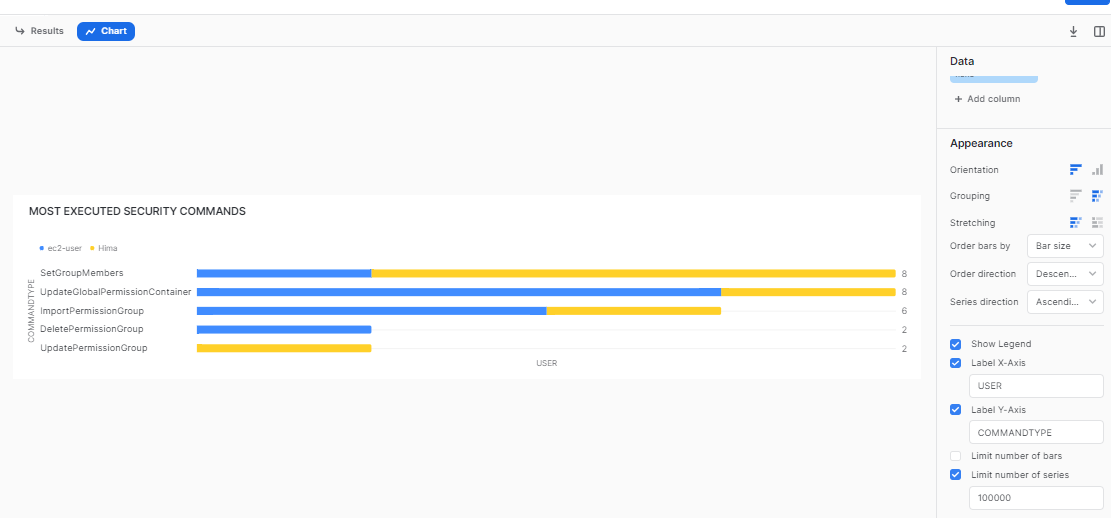




**7 MOST EXECUTED SECURITY COMMANDS:**

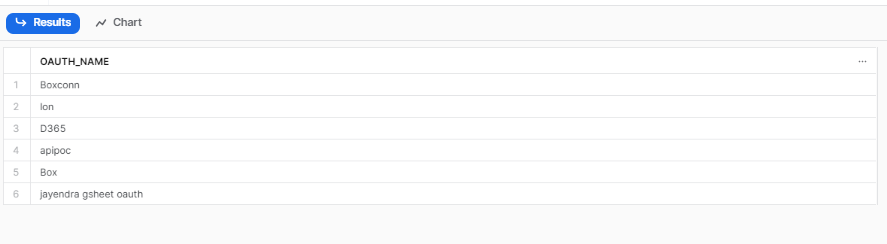
select \* from MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.AUDIT\_NEW WHERE SPECIFIER LIKE '%Security%' and timestamp=:daterange;





**8 OAUTHS CONFIGURED IN MATILLION:**

SELECT \* FROM MATILLION\_MONITOR\_DB.MATILLION\_MONITOR\_SCHEMA.OAUTH;



**9 Who Granted Role To Users ?:**

SELECT GRANTEE\_NAME AS USER,

IFNULL( GRANTED\_BY , 'NONE' ) AS GRANTED\_BY,

CREATED\_ON FROM SNOWFLAKE.ACCOUNT\_USAGE.GRANTS\_TO\_USERS WHERE GRANTED\_TO='USER'

AND

ROLE= CURRENT\_ROLE() AND DELETED\_ON IS NULL;

