Snowflake : Dynamic Data Masking

Pooja Kelgaonkar Follow 5 min read · Jul 28, 2023

Thank you all for reading my earlier blogs on Snowflake. I am starting a new series on Implementing data governance with Snowflake. Data governance implementation has various pillars. The data governance umbrella consist of many subcategories. Some of the pillars include — Access control, Data protection (classification), Data security (encryption) , Data Quality, Data sharing etc. This blog will help you learn one of the most important security feature — Data Protection.

Data protection is the process of protecting data from loss, misuse or data being exposed to users. Data is protected based on the data classification. Data classification is one of the critical pillar of data governance. Data is classified based on the category of the information being stored in the form of data. Data can be classified into Sensitive or confidential when it stores information like name, phone number, mail id, address, blood group, account number, ssn, tin etc is considered as PII — Person identifiable information. Data can also be classified as per the classifications like confidential, internal, public, classified etc depending on the information and consumers using them. To protect data from being misused data masking or tokenization is implemented. This is essential to protect data from being misused or exposed to users.

Snowflake supports dynamic data masking. This is enterprise edition feature in GA. You can define the policies once and apply to the tables, columns, views to protect data from being viewed by all users. You can follow below steps to implement dynamic data masking -

1. Create a custom role to define masking policies
2. Grant custom role to required users
3. Security officer or architect or admin creates these policies and apply to the relevant tables hence data
4. Validating rules or policies applied

You will learn more about each of these steps in subsequent section of this blog.

1. **Create a custom role to define masking policies**

As you all know, snowflake recommends RBAC access control implementation. You can use existing role to create security policies or create a new custom role. ACCOUNTADMIN or SECURITYADMIN can create these policies. You need to allow below privileges to the custom user — CREATE MASKING POLICY, APPLY MASKING POLICY and APPLY ON MASKING POLICY.

Use below commands to create role ->

use useradmin;

create role masking_admin;

Use below commands to grant permissions ->

use role securityadmin;

GRANT CREATE MASKING POLICY on schema demo_db.poc to role masking_admin;

GRANT APPLY MASKING POLICY on ACCOUNT to role masking_admin;

2. Grant custom role to required users

Now, you have role created and permissions applied. you can grant this role to the user who can create policies and apply.

GRANT ROLE masking_policy to user poojakelgaonkar;

3. Create MASKING policies

Now, you can use the role granted and create policies using below command —

CREATE OR REPLACE MASKING POLICY email_visibility **AS** (val **string**) **RETURNS string ->**

CASE

WHEN CURRENT_ROLE() **IN** ('ACCOUNTADMIN') **THEN** val

ELSE '******@***.com'

END;

This policy defines a rule that if the user is ACCOUNTADMIN then user can see the original value. if the user is not ACCOUNTADMIN then user will see

masked values as per the policy.

You can also create a policy to mask the account number, ssn or tin using below command —

CREATE OR REPLACE MASKING POLICY identification_visibility **AS** (val **string) RETURNS string ->**

CASE

WHEN CURRENT_ROLE() **IN** ('ACCOUNTADMIN') **THEN** val

ELSE '********'

END;

4. Apply the policies to the objects

You have policies defined and this is one time activity. You can apply them to your data model or database objects. You can use them while creating a object with DDL or use ALTER to assign policies —

— apply masking policy to a table column

ALTER TABLE IF EXISTS customer_details **MODIFY COLUMN** email **SET MASKING POLICY** email_visibility**;**

— apply masking policy to a table column

ALTER TABLE IF EXISTS account_details **MODIFY COLUMN** accountnumber **SET MASKING POLICY** identification_visibility;

ALTER TABLE IF EXISTS account_details **MODIFY COLUMN** ssn **SET MASKING POLICY** identification_visibility;

— apply the masking policy to a view column

ALTER VIEW v_account_details **MODIFY COLUMN** email **SET MASKING POLICY** email_visibility;

5. Validate data masked

You can create two sample tables customer_details and account_details. use below table DDLs to create them. Load some sample records to see masking getting applied dynamically.

use role sysadmin;

use database demo_db;

use schema poc;

create table customer_details (cust_id integer, cust_name string, cust_email string, addr string);

create table account_details(acct_id integer, acc_name string, accountnumber integer, ssn string, email string);

load some sample records using insert statements —

```
insert into customer_details(10120,'smith','smith.c@gmail.com','north york');

insert into customer_details(11230,'joe','joems@gmail.com','NJ');

insert into account_details(110111,'smith
c',0056937900,3937983,smith.c@gmail.com');
```

Now, you can use SQL queries to validate the policies applied. You can use default role to run select queries on the data and use accountadmin to run the same select queries and check data visibility.

```
use role sysadmin;

use database demo_db;

use schema poc;

select * from customer_details;

select * from account_details;
```

Note — you will see a masked values for the columns where policies are applied.

```
use role accountadmin;

use database demo_db;

use schema poc;
```

select * from customer_details;

select * from account_details;

Now, you will see all records as -is without any masking as you are using accountadmin role and as per policy accountadmin can see original value.

You can also create full masking or partial masking or tokenization while defining the policy. Refer to below command —

CASE

WHEN current_role() **IN** (‘ACCOUNTADMIN’) **THEN** val

WHEN current_role() **IN** (‘ANALYST’) **THEN**
regexp_replace(val,’.+\@’,’*****@’) — leave email domain unmasked

ELSE ‘********’

END;

Hope this blog helps you to learn and implement dynamic masking in Snowflake. Follow this blog series to know and learn more about other data governance pillars.

> ***About Me :***
>
> *I am one of the Snowflake Data Superheroes 2023. I am also one of the Snowflake SnowPro Core SME- Certification Program. I am a DWBI and Cloud Architect! I am currently working as Senior Data Architect — GCP, Snowflake. I have been*

> *working with various Legacy data warehouses, Bigdata Implementations, and Cloud platforms/Migrations. I am SnowPro Core certified Data Architect as well as Google certified Google Professional Cloud Architect. You can reach out to me LinkedIn if you need any further help on certification, Data Solutions, and Implementations!*

Snowflake | Data Superhero | Data Masking

Responses (1)

Eng Omar Essam

What are your thoughts?

Brahma, The Data Engineer.
Aug 3, 2023

Hi Pooja,

Nice article. I hope you're well. I'm a data engineer with a passion for Snowflake and have written several articles on the medium platform. I've been following your work closely and find it inspiring.

I'm interested in contributing to the... more

1 reply Reply

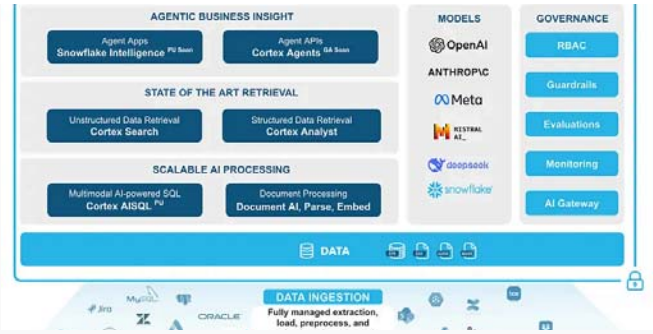
More from Pooja Kelgaonkar and Snowflake Builders Blog: Data Engineers, App Developers, AI, & Data Science

Pooja Kelgaonkar

SnowPro Advanced: Architect Certification Study Guide

Recently, I have completed my advance architect certification and its a great...

Mar 22 6



In Snowflake Builders Blog: Data Engine... by Ume...

Build Snowflake Cost Savings and Performance Agent in 5 minutes

Snowflake Cortex Agent, building agentic AI solutions is no longer about complexity—it's...

Sep 30 44 3

In Snowflake Builders Blog: Data Enginee... by Sai...

Agent Instruction Best Practices for Snowflake Intelligence

Prototyping AI agents is easy. However, successfully launching reliable agents to...

Sep 25 54 3

In Snowflake Builders Blog: Data Engi... by Pooja ...

Architecting Data Warehousing solutions with Snowflake

Thanks for reading out the earlier blog in the series. The earlier blog was focused on...

May 13, 2023 34

See all from Pooja Kelgaonkar

See all from Snowflake Builders Blog: Data Engineers, App Developers, AI, & Data Science

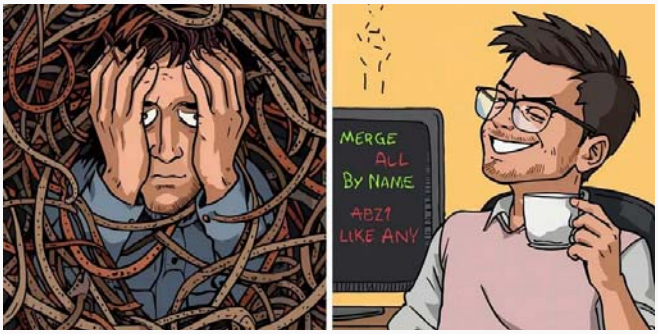
Recommended from Medium

John Ryan

Snowflake Gen-2 Warehouses: Faster Performance or Just Highe...

Introduction

Oct 1 6



Vishal Kaushal

Top 10 Snowflake SQL Functions Every Data Professional Should...

Are you spending too much time writing and maintaining long SQL queries in Snowflake?...

Oct 16 34 1

Karthik Goutam

Enterprise Data Pipeline: Jira Atlassian Cloud to Snowflake ETL...

Abstract

In Snowflake Builders Blog: Data Engin... by M...

Role-based Access Control (RBAC) with Snowflake Database Roles

Managing role-based access control (RBAC) in Snowflake can become overwhelming as...

Jun 9 4

In Python in Plain English by Amey A.

CAST & TRY_CAST In Snowflake

‘05-jan-1996’ is string and in current state not able to provide any date related information...

Sep 23

Apr 24 55 2

Kamalakannan R

Snowflake

— use role sysadmin; —use warehouse compute_wh; —create or replace database...

Aug 20

See more recommendations