IMMUTA®

E-BOOK

The Snowflake Data Security Guide

Table of Contents

Introduction

As with any sound data infrastructure, you have to lay the foundation before you can start seeing results. When it comes to data security, however, access control and user management are often treated as afterthoughts. Security is a primary factor in almost every other aspect of our lives – so why should our data be treated any differently?

Think of it like Disney World. You can roam freely around the park – so long as you've paid and gone through the necessary security checkpoints – and only those with authorized access are able to go behind the scenes. With the wave of a bracelet, you can gain access to rides and concessions – but only if you've gone through the proper steps to acquire said bracelet. Your experience as a visitor seems dynamic and seamless, due in large part to the rigorous processes and controls put in place well before your arrival, and which remain invisible to you.

Data engineering and architecture teams should strive to deliver a Disney World-like experience for their data consumers – one that allows them to feel that they're free to access the data they need, while fine-grained access controls work behind the scenes to ensure that they remain within the proper boundaries. However, without the right strategy and technology in place, implementing these controls may be easier said than done.

In this guide, we'll explore how to:

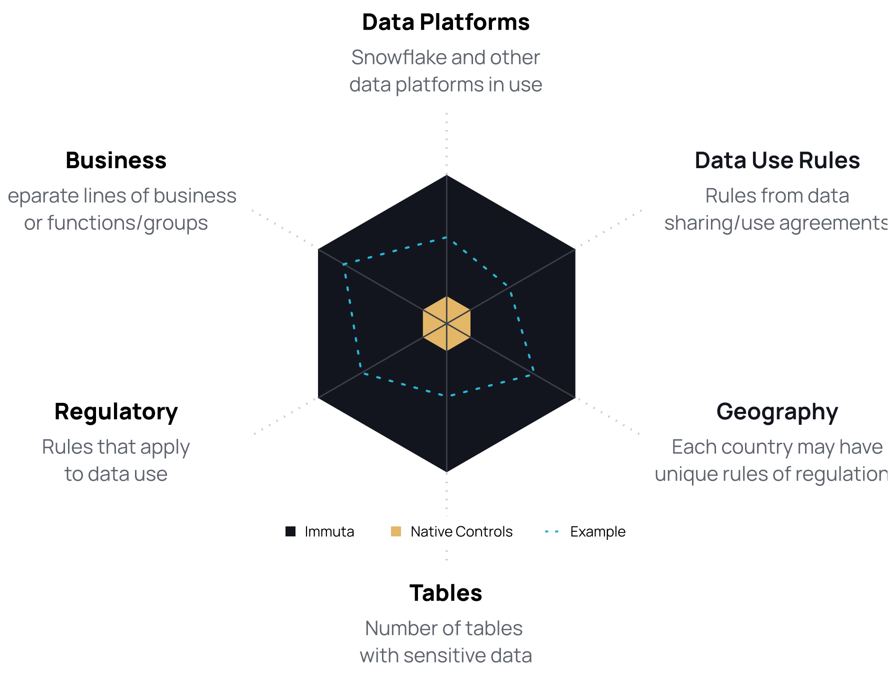
- Establish fine-grained access control for Snowflake Governance
- Implement attribute-based access control for added flexibility and scalability
- Enforce column-level security for sensitive data
- Apply data masking policies consistently across platforms for full-proof privacy
- Enable secure, dynamic data collaboration

Why Do You Need Data Security Safeguards for Snowflake Data?

Cloud data platforms such as Snowflake provide native security and data access controls engineered for platform-specific administrators to manage roles across the organization. Snowflake leverages concepts from discretionary access control (DAC) and role-based access control (RBAC) models to provide some agency over how users can access securable objects.

However, relying on a centralized data platform team to manage a hierarchy of roles and privileges can easily become a bottleneck. With different tenants and lines of business (LOBs) possessing domain-specific knowledge over their own data, policies, and users, scaling secure data access – particularly in complex data landscapes—is a substantial obstacle.

Below are examples of dimensions that contribute to this complexity:



This requires a diverse set of roles and stakeholders to implement and manage data access:

- Data Platform Architecture
- Data Engineering & Operations
- Data Owners (technical stewards organized by function or lines of business)
- Data Consumers (data scientists and analysts)
- Governance, Risk, and Compliance Teams (including legal)

Key Challenges to Managing Snowflake Data Security

These challenges are not necessarily specific to Snowflake, but apply to any centralized cloud data platform and span people, process, and technology. They are more common in large organizations where Snowflake is managed by a centralized platform team, and include:

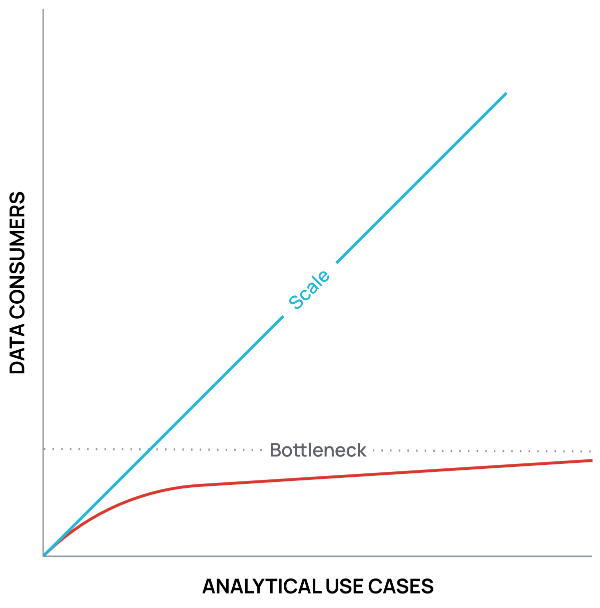
- Centralized platform teams' lack of domain expertise over the data
- Manual and slow processes to fulfill data access requests for internal customers
- Difficulty for business and security stakeholders to understand who can access what data and why

Many organizations are able to derive value from their data with a specific use case and small set of users, but as they look to grow those use cases and to scale user adoption, they run into the limitations of foundational access controls. Native access controls can't expand as new use cases, policies, and user cohorts are added, which requires data teams to rework access controls from scratch.

The good news is that with automated data security management and dynamic access control capabilities, technology is in a unique position to enable the people and process side of it as well.

Recommendations for Scaling Snowflake Data Security Management

Similar to data mesh architecture principles, decentralizing policy ownership is the most efficient way forward in complex data landscapes. The best way to manage this is by using a centralized access control infrastructure.



These are the common bottlenecks to cloud user adoption and key approaches to consider:

Bottleneck	Recommended Approach
Manual Role Management	Automate global security & privacy controls
Static Access Control	Leverage attribute-based access control using policy variables
Distributed Domain Stewardship	Empower each data owner to manage policies using data domain knowledge specific to a business line.

The following sections will explore how to execute each of these approaches so you can simplify Snowflake Governance and start getting more value from your data.

Laying the Groundwork: Fine-Grained Access Control

Deploying fine-grained access for Snowflake gives data teams agency over how data is accessed and used, without being too restrictive or permissive for users trying to access it. In this section, we'll lay the foundation for implementing Snowflake fine-grained access control using the Immuta policy engine. This involves:

1. Creating an open discoverability space with sanitized data for all employees
2. Enforcing strict controls over the raw data assets

With Immuta-enforced policies, this strategy allows you to open up your Snowflake platform and delegate control without concerns of leaking sensitive data to end users. Let's take a closer look.

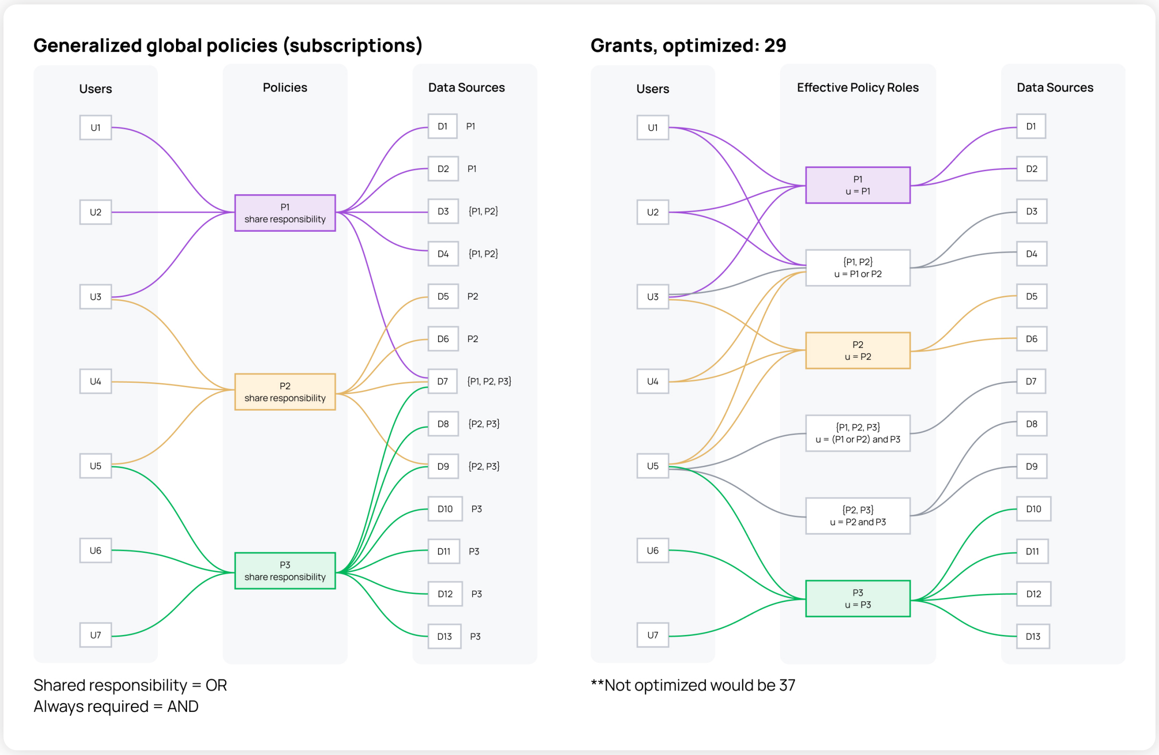
Abstract Policy from Users

Data consumers do not have an intrinsic demand for "analytics," "marketing," and "executive" roles in your data platform. Their priority is to get access to useful data that will deliver value—roles are just an access control implementation detail they (and data platform owners) have to live with because, well, that's just how it's done.

The alternative is to base access to data assets on user attributes—facts about users, which include, but are not limited to, groups and roles.

With fine-grained access controls, configured as part of each user, abstracted from multiple physical roles, each user has their own role and is personalized for their access to each table, down to the cell level. This dramatically simplifies both the platform owner's administrative burden and the end user's experience. It also unlocks many more possibilities for differentiating user access or protecting sensitive data.

The diagram below shows how Immuta centralizes and implements fine-grained access control in Snowflake.



By separating the data access control decision (what data should be released?) from the specific database objects (who is querying what data?), Immuta makes access controls more flexible and context-aware. In this example, a role hierarchy represents full access descisions, but from a user's perspective, they only need to know that they are using their single personal role for table access (pictured) and further fine-grained access control is layered in (not pictured, but we will address shortly). This presents a few advantages:

- Policies can be layered on, rather than baked in. If you want to start masking user names, you don't have to create a specific new policy for all the affected tables. This allows for delegation of control.
- Exceptions can be dealt with as they arise, rather than having to be planned for ahead of time. For example, a decision may be that users should only have access to the last year of Salesforce data, except for the analytics team. A second policy can be easily layered on to minimize access, even long after the data set is made "golden."

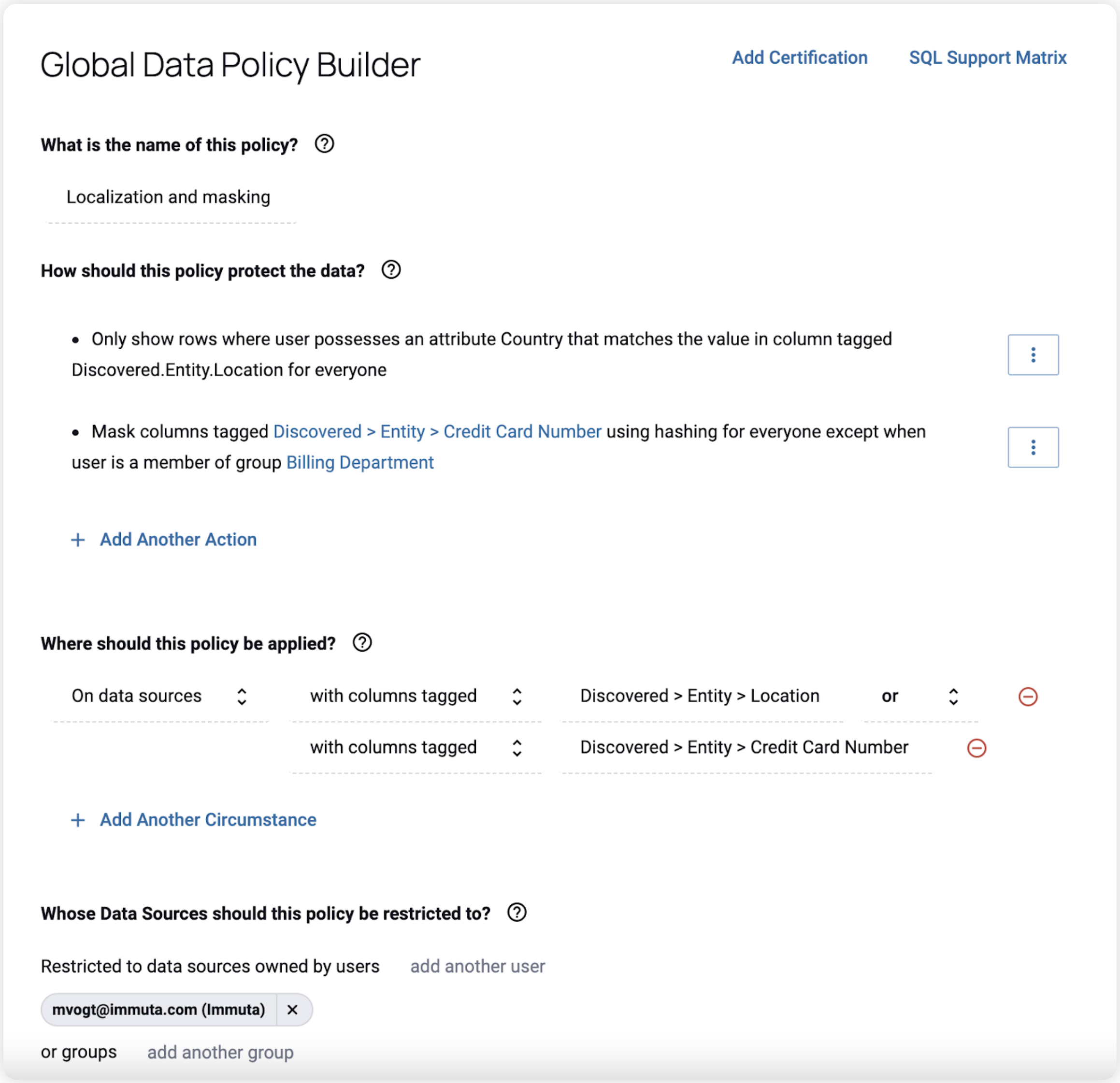
Focusing efforts on building out this common access area with the appropriate protections brings clarity to what data needs to be protected, why, and for whom.

Scaling User Adoption: Attribute-Based Access Control

Let's say you are ready to expand how your data is being used and by whom. The first step will be to implement global access controls for the new analytical use case.

Implementing Global Controls

This process starts with your data owners, who are responsible for delivering data in Snowflake. They will work with business stakeholders to understand the use case and align on the requirements for using relevant data in the cloud. For this example, they'll use these inputs to set up global data access policies so users can only see data in their country, and all credit card numbers are masked for everyone except privileged Billing Department users.



What is the name of this policy?

Localization and masking

How should this policy protect the data?

- Only show rows where user possesses an attribute Country that matches the value in column tagged Discovered.Entity.Location for everyone
- Mask columns tagged Discovered > Entity > Credit Card Number using hashing for everyone except when user is a member of group Billing Department

Where should this policy be applied?

On data sources	with columns tagged	Discovered > Entity > Location	or
	with columns tagged	Discovered > Entity > Credit Card Number	

Whose Data Sources should this policy be restricted to?

Restricted to data sources owned by users add another user

mvogt@immuta.com (Immuta) ×

or groups add another group

A few things to point out from this policy:

1. It will only show rows based on the querying user's country attribute as compared to the data in the column tagged Discovered.Entity.Location. This is a dynamic evaluation, so if the user's country changes, the policy remains correct. With an RBAC model, you would have to create a new role and policy for EVERY INDIVIDUAL PERMUTATION of access (remember, some users would have more than one country).
2. This policy will apply on tables with columns tagged with either Discovered.Entity.Location or Discovered.Entity.Credit Card Number tags. This is not possible in an RBAC system where the simple policy shown above could be literally hundreds of roles and policies configured with layers of complex SQL.
3. This also shows that the user building this policy has been given delegation control—as seen at the bottom. The policy will only apply to the tables that mvogt@immuta.com - (the policy builder) owns.

As you can see from this single, simple policy, when governance, risk, and compliance (GRC) stakeholders receive requests from auditors to answer basic questions on what personally identifiable information (PII) is accessible and whether it's compliant with data collection rules — requests that are usually in legal language, which might sound Greek to data engineers — those data engineers can share the explainable policies and audit reports in real time with the GRC team. These policies and data audit trails are easy to understand, even without background knowledge of Snowflake access control modules and declarative commands. This is not possible in an RBAC system where the simple policy shown above could be literally hundreds of roles and policies.

Under the covers, this policy authored in Immuta is pushed out to Snowflake as native row access and masking policies, acting as a translation layer between plain language polices and native Snowflake governance controls maintained in SQL.

Protecting Sensitive Data: Column-Level Masking for PII & PHI

As sensitive data, like personally identifiable information (PII) and protected health information (PHI), becomes more widely used across industries, it is incumbent upon data teams to have mechanisms in place to protect it. We showed this briefly in this credit card number example in the previous section but let's dig in deeper to Snowflake's column-level security capability that enables dynamic data masking of this type of sensitive data through policies that are applied at query time to mask plain-text data in the table and view columns. How the data appears to users is based on the policy conditions, SQL execution context, and role entitlements.

Immuta's native integration with Snowflake also enables dynamic data masking, but using an automated, attribute-based approach to policy enforcement. In this section, we'll walk through two ways to implement Snowflake column-level security policies in order to contrast them. The first shows how to mask PII using Snowflake's built-in role-based controls, and the second demonstrates how to do the same task with Snowflake's Immuta integration using attribute-based access control, as discussed in the previous section.

Steps 1–7 demonstrate column-level security with native Snowflake controls, but you can also jump to page 16 to see the single ABAC policy approach with Immuta.

Scenario: Masking Snowflake Customer Data Labeled PII

In this scenario, you have sensitive customer and call center data that must be masked for all users, except those with an override authorization. There are four Snowflake users in the company's central office, but only two are authorized to see the unmasked data.

Note: the tables used are from the TPC-DS sample data available from Snowflake.

Creating Column-Level Masking Policies for PII in Snowflake

1. Create Masking Policies

First, you'll create the policy **PII_N_NULLIFY** to mask **Number** to null.

```
USE ROLE ACCOUNTADMIN;
```

```
    CREATE OR REPLACE MASKING POLICY PII_N_NULLIFY AS
```

```
(VAL NUMBER) RETURNS NUMBER -> CASE WHEN INVOKER_ROLE() IN ('CENTRAL_OFFICE_
ADMIN') THEN VAL ELSE NULL
```

```
END;
```

Next, you'll create a similar policy, **PII_C_NULLIFY**, to mask **VARCHAR** to null.

```
CREATE OR REPLACE MASKING POLICY PII_C_REPLACE AS
```

```
(VAL VARCHAR) RETURNS VARCHAR -> CASE WHEN INVOKER_ROLE() IN ('CENTRAL_OFFICE_
ADMIN') THEN VAL ELSE 'REDACTED'
```

```
END;
```

As you can see, when building the masking policies, you must account for every possible data type. In this scenario, we want to show the word "REDACTED", but since a numeric column can't contain a string, we must NULL the numeric columns with a different masking policy.

2. Apply Masking Policies to CUSTOMER table

Once the policies are built, you can enforce them on your customer table so that personal information, like names, email addresses, and birthdays, are masked. The user managing the policy must know this table contains PII columns, and which columns specifically contain PII, so they can apply the policy one by one. The user managing the policy must also know when to apply the number masking policy versus the varchar masking policy by understanding the column types.

```
USE ROLE ACCOUNTADMIN;
```

```
    ALTER TABLE customer MODIFY COLUMN c_first_name SET MASKING POLICY
PII_C_REPLACE;
```

```
    ALTER TABLE customer MODIFY COLUMN c_last_name SET MASKING POLICY
PII_C_REPLACE;
```

```
    ALTER TABLE customer MODIFY COLUMN c_birth_country SET MASKING POLICY
PII_C_REPLACE;
```

```
    ALTER TABLE customer MODIFY COLUMN c_birth_day SET MASKING POLICY
PII_N_NULLIFY;
```

```
    ALTER TABLE customer MODIFY COLUMN c_birth_month SET MASKING POLICY
PII_N_NULLIFY;
```

```
    ALTER TABLE customer MODIFY COLUMN c_birth_year SET MASKING POLICY
PII_N_NULLIFY;
```

The following columns are masked in the Customer table as **PII_N_NULLIFY** or **PII_C_REPLACE**:

Table	Column	Masking Type
customer	c_first_name	PII_C_REPLACE
customer	c_last_name	PII_C_REPLACE
customer	c_birth_country	PII_C_REPLACE
customer	c_birth_day	PII_C_NULLIFY
customer	c_birth_month	PII_C_NULLIFY
customer	c_birth_year	PII_C_NULLIFY

3. Apply Masking Policies to CALL_CENTER table

Next, you'll repeat the same process for the call center table, so that employees' and managers' names are masked. Again, the user managing the policy must know this table contains PII columns and which columns specifically contain PII so they can apply the policy one by one.

```
    ALTER TABLE call_center MODIFY COLUMN cc_name SET MASKING POLICY
PII_C_REPLACE;
```

```
ALTER TABLE call_center MODIFY COLUMN cc_employees SET MASKING POLICY
PII_N_REPLACE;
```

```
    ALTER TABLE call_center MODIFY COLUMN cc_manager SET MASKING POLICY
PII_C_REPLACE;
```

```
    ALTER TABLE call_center MODIFY COLUMN cc_market_manager SET MASKING POLICY
PII_C_REPLACE;
```

The following columns are tagged in call_center table as **PII_N_NULLIFY** or **PII_C_NULLIFY**:

Table	Column	Masking Types
call_center	cc_name	PII_C_REPLACE
call_center	cc_employees	PII_C_NULLIFY
call_center	cc_manager	PII_C_REPLACE
call_center	cc_market_manager	PII_C_REPLACE

4. Create Two Users *_a for Authorized Users and Two Users *_e for Unauthorized Users

In this scenario, you have four users in the central office, but only two should be allowed to access the unmasked data. Therefore, you'll create the user roles for each of the four users with their corresponding permission levels, add them to the central office role, and finally add the two users with authorization to see unmasked data to the admin role.

```
--Grant 2 new users to 'CENTRAL_OFFICE' role
```

```
GRANT ROLE CENTRAL_OFFICE TO USER jeff_e;
```

```
GRANT ROLE CENTRAL_OFFICE TO USER bill_e;
```

```
--Grant 2 new users to 'CENTRAL_OFFICE_ADMIN' role
```

```
GRANT ROLE CENTRAL_OFFICE_ADMIN TO USER john_a;
```

```
GRANT ROLE CENTRAL_OFFICE_ADMIN TO USER jane_a;
```

```
  GRANT select on all tables in schema [database.schema] to role
CENTRAL_OFFICE_ADMIN
```

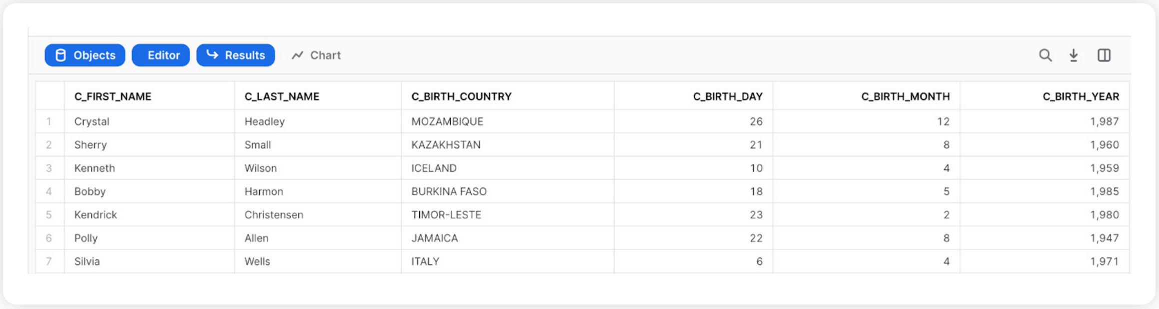
```
  GRANT select on all tables in schema [database.schema] to role
CENTRAL_OFFICE
```

5. View Two Authorized Users (CENTRAL_OFFICE_ADMIN role) Who See Unmasked Data

To verify that your authorized users, John and Jane, are able to see unmasked data, you'll log in and run a query under their permissions. After logging in as JOHN_A, you can click on Worksheets and run the following SQL statements:

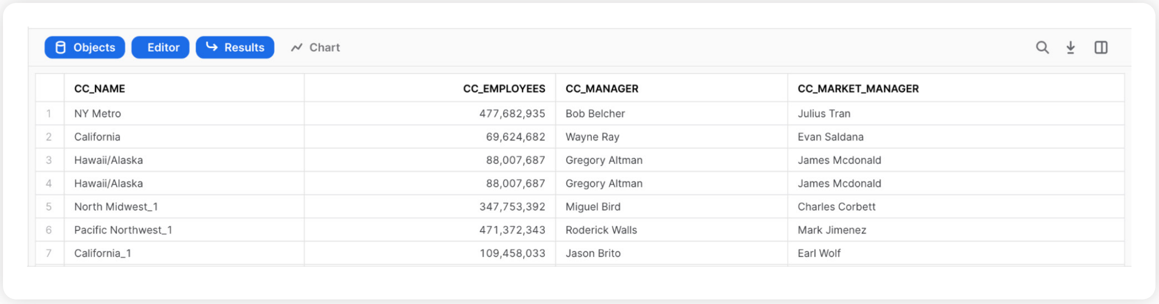
```
USE ROLE CENTRAL_OFFICE_ADMIN;
```

```
select c_first_name,c_last_name,c_birth_country,c_birth_day,c_birth_month,c_birth_
year, from customer;
```



	C_FIRST_NAME	C_LAST_NAME	C_BIRTH_COUNTRY	C_BIRTH_DAY	C_BIRTH_MONTH	C_BIRTH_YEAR
1	Crystal	Headley	MOZAMBIQUE	26	12	1,987
2	Sherry	Small	KAZAKHSTAN	21	8	1,960
3	Kenneth	Wilson	ICELAND	10	4	1,959
4	Bobby	Harmon	BURKINA FASO	18	5	1,985
5	Kendrick	Christensen	TIMOR-LESTE	23	2	1,980
6	Polly	Allen	JAMAICA	22	8	1,947
7	Silvia	Wells	ITALY	6	4	1,971

```
select cc_name,cc_employees,cc_manager,cc_market_manager, from call_center;
```



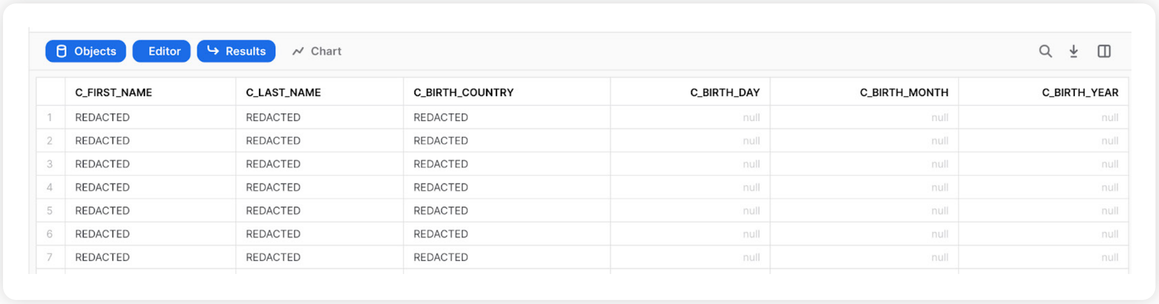
	CC_NAME	CC_EMPLOYEES	CC_MANAGER	CC_MARKET_MANAGER
1	NY Metro	477,682,935	Bob Belcher	Julius Tran
2	California	69,624,682	Wayne Ray	Evan Saldana
3	Hawaii/Alaska	88,007,687	Gregory Altman	James Mcdonald
4	Hawaii/Alaska	88,007,687	Gregory Altman	James Mcdonald
5	North Midwest_1	347,753,392	Miguel Bird	Charles Corbett
6	Pacific Northwest_1	471,372,343	Roderick Walls	Mark Jimenez
7	California_1	109,458,033	Jason Brito	Earl Wolf

6. Verify that Unauthorized User (CENTRAL_OFFICE role) Can Only See Masked Data

Next, you'll run the same query logged in as one of your unauthorized users, Jeff or Bill, to ensure they are able to see only masked data. To do this, log in as JEFF_E, click on Worksheets and run the following SQL statements:

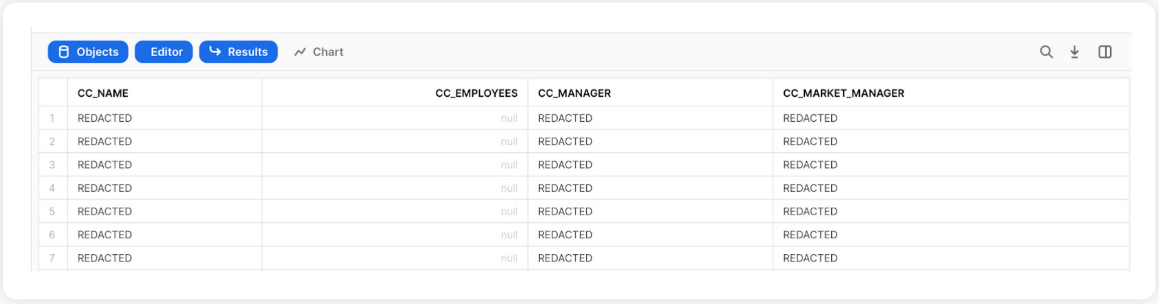
```
USE ROLE CENTRAL_OFFICE;
```

```
select c_first_name,c_last_name,c_birth_country,c_birth_day,c_birth_month,c_
birth_year, from customer;
```



	C_FIRST_NAME	C_LAST_NAME	C_BIRTH_COUNTRY	C_BIRTH_DAY	C_BIRTH_MONTH	C_BIRTH_YEAR
1	REDACTED	REDACTED	REDACTED	null	null	null
2	REDACTED	REDACTED	REDACTED	null	null	null
3	REDACTED	REDACTED	REDACTED	null	null	null
4	REDACTED	REDACTED	REDACTED	null	null	null
5	REDACTED	REDACTED	REDACTED	null	null	null
6	REDACTED	REDACTED	REDACTED	null	null	null
7	REDACTED	REDACTED	REDACTED	null	null	null

```
select cc_name,cc_employees,cc_manager,cc_market_manager, from call_center;
```



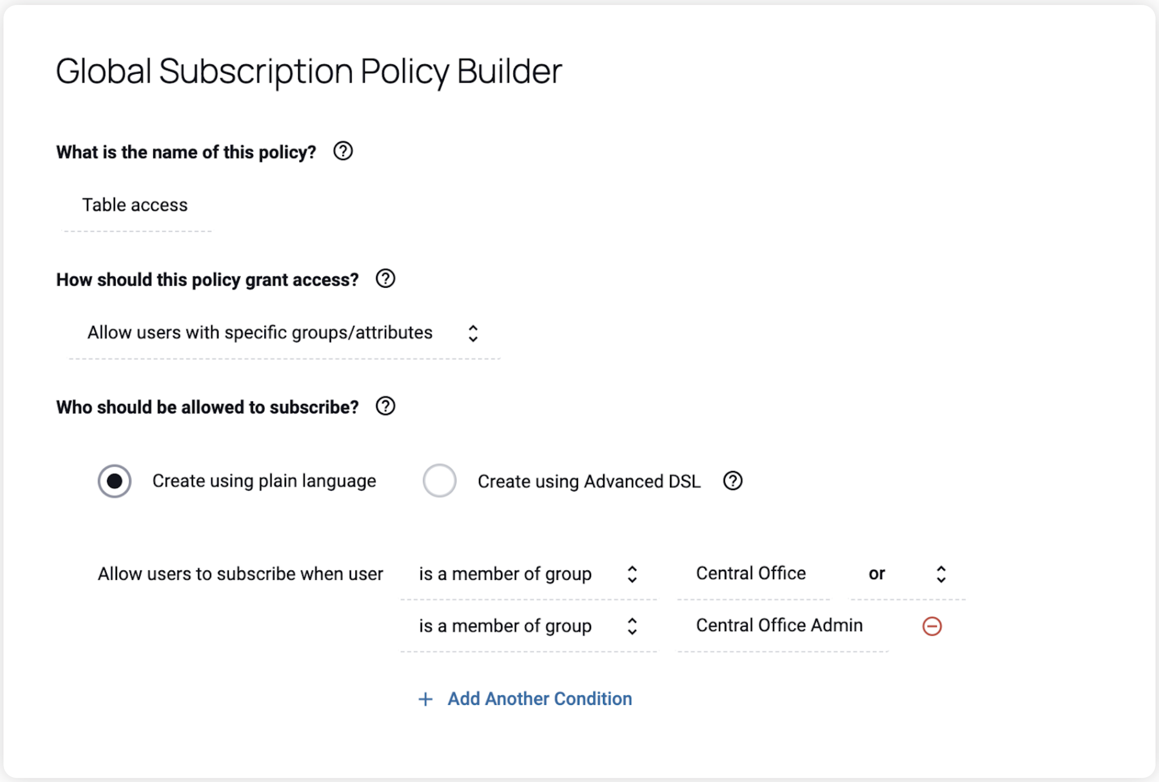
	CC_NAME	CC_EMPLOYEES	CC_MANAGER	CC_MARKET_MANAGER
1	REDACTED	null	REDACTED	REDACTED
2	REDACTED	null	REDACTED	REDACTED
3	REDACTED	null	REDACTED	REDACTED
4	REDACTED	null	REDACTED	REDACTED
5	REDACTED	null	REDACTED	REDACTED
6	REDACTED	null	REDACTED	REDACTED
7	REDACTED	null	REDACTED	REDACTED

As you can see, your authorized users are able to access unmasked data, but it remains masked for unauthorized users. Now, let's see how it's done when Immuta is integrated with Snowflake.

Applying Column-Level Masking Policies with Snowflake + Immuta

To accomplish the same objective in Snowflake using two simple polices in Immuta. You start by creating new groups called "central-office" and "central-office-admins" and add the appropriate users. Note that these groups could have come from your identity manager and do not necessarily have to be created manually in Immuta.

Next, you'll create a global access policy that allows all users in that group to subscribe to the database's tables.


Global Subscription Policy Builder

What is the name of this policy?

Table access

How should this policy grant access?

Allow users with specific groups/attributes

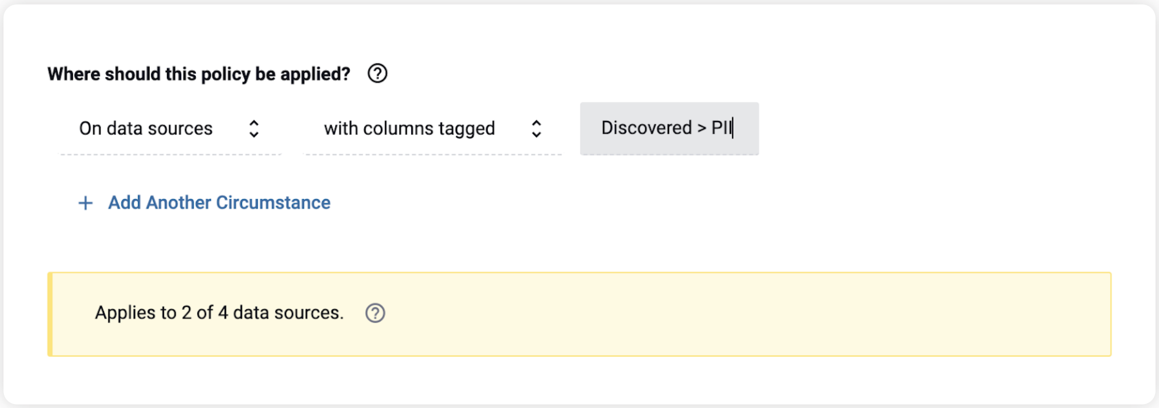
Who should be allowed to subscribe?

[✓ Create using plain language] Create using Advanced DSL

Allow users to subscribe when user is a member of group Central Office or
is a member of group Central Office Admin

You see that either Central Office or Central Office Admin users will gain access to this table. It's important to mention that complex boolean logic is allowed here, unlike RBAC systems which always ORs roles together.

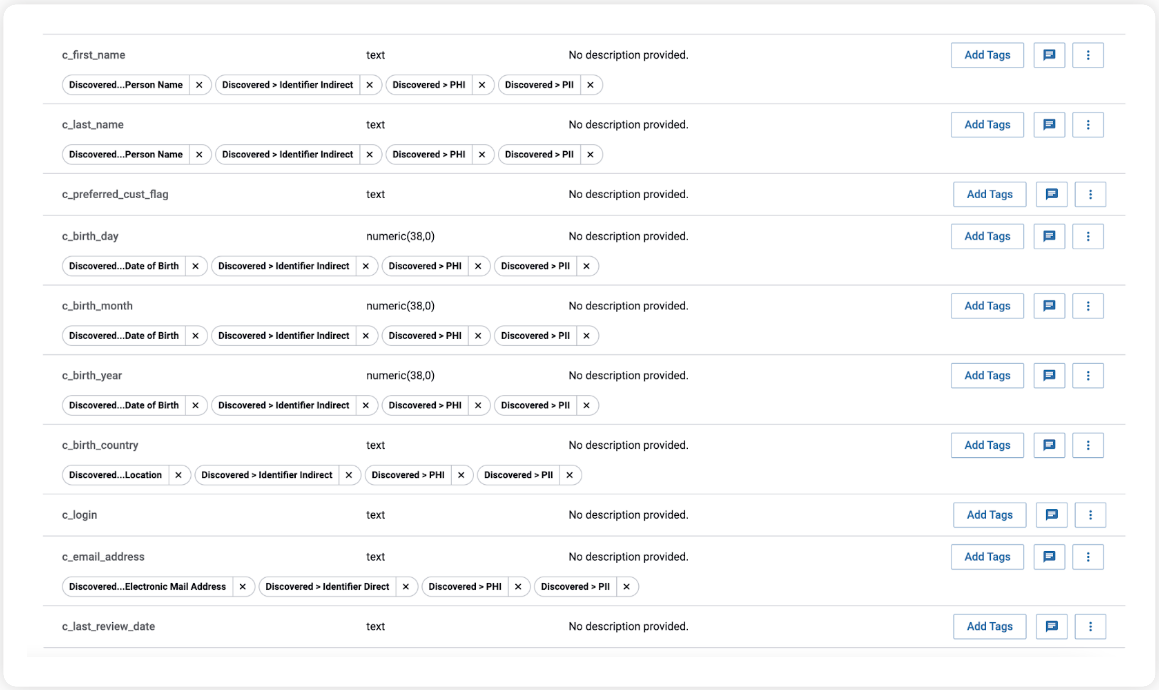
This policy will be applied anywhere we've discovered PII.


Where should this policy be applied?

On data sources with columns tagged Discovered > PII

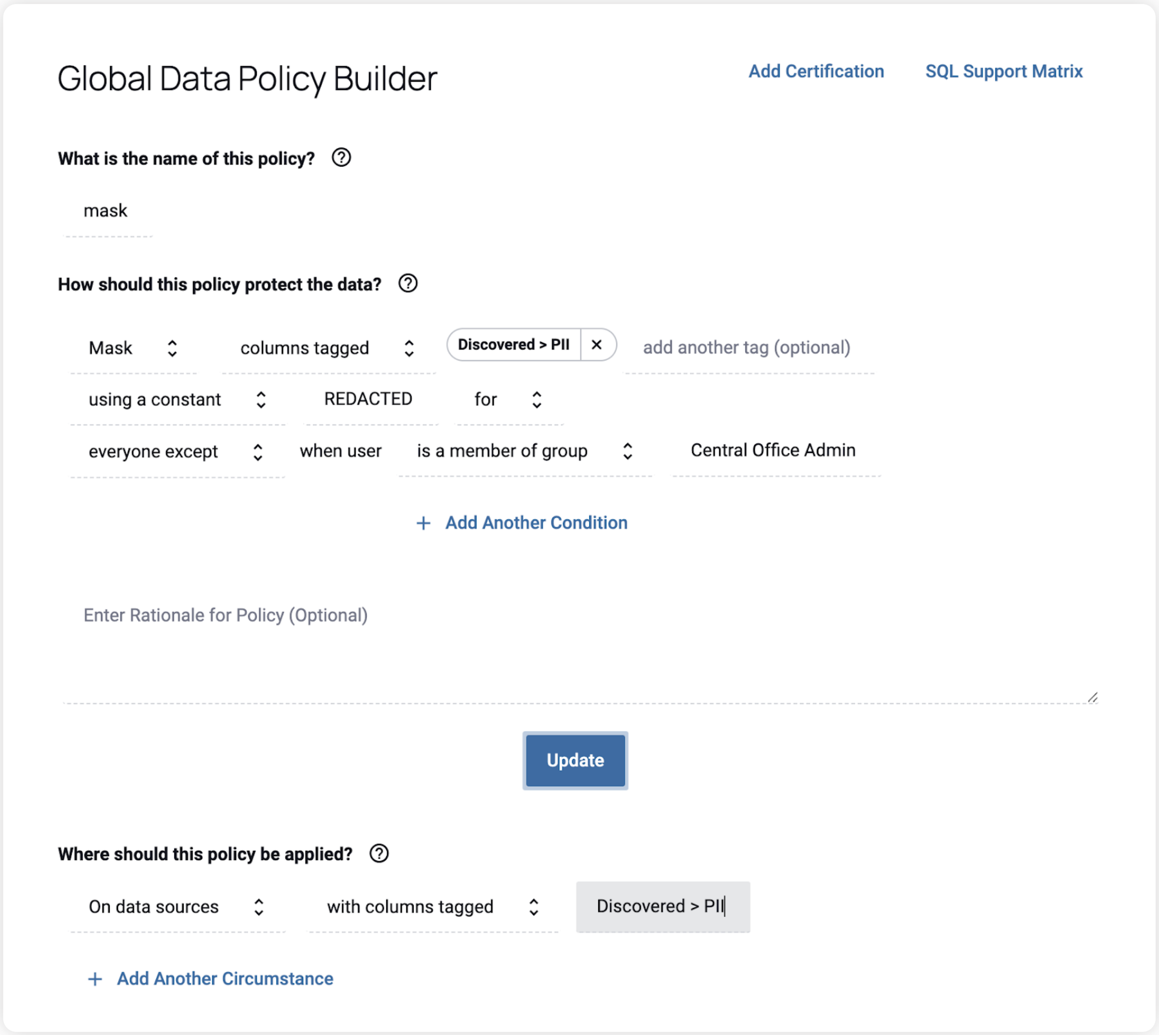
Applies to 2 of 4 data sources.

Immuta's sensitive data discovery and classification capability automatically tags data containing PII (and other characteristics), based on built-in classifiers (data teams have the ability to design their own custom classifiers as well). Therefore, any Snowflake data registered with Immuta is automatically tagged with Discovered.PII, without any manual investigating or input.



c_first_name	text	No description provided.	Add Tags
Discovered...Person Name ✕ · Discovered > Identifier Indirect ✕ · Discovered > PHI ✕ · Discovered > PII ✕			
c_last_name	text	No description provided.	Add Tags
Discovered...Person Name ✕ · Discovered > Identifier Indirect ✕ · Discovered > PHI ✕ · Discovered > PII ✕			
c_preferred_cust_flag	text	No description provided.	Add Tags
c_birth_day	numeric(38,0)	No description provided.	Add Tags
Discovered...Date of Birth ✕ · Discovered > Identifier Indirect ✕ · Discovered > PHI ✕ · Discovered > PII ✕			
c_birth_month	numeric(38,0)	No description provided.	Add Tags
Discovered...Date of Birth ✕ · Discovered > Identifier Indirect ✕ · Discovered > PHI ✕ · Discovered > PII ✕			
c_birth_year	numeric(38,0)	No description provided.	Add Tags
Discovered...Date of Birth ✕ · Discovered > Identifier Indirect ✕ · Discovered > PHI ✕ · Discovered > PII ✕			
c_birth_country	text	No description provided.	Add Tags
Discovered...Location ✕ · Discovered > Identifier Indirect ✕ · Discovered > PHI ✕ · Discovered > PII ✕			
c_login	text	No description provided.	Add Tags
c_email_address	text	No description provided.	Add Tags
Discovered...Electronic Mail Address ✕ · Discovered > Identifier Direct ✕ · Discovered > PHI ✕ · Discovered > PII ✕			
c_last_review_date	text	No description provided.	Add Tags

Further leveraging this capability, you'll create a new global data policy that masks all columns with the Discovered.PII tag and has the cell-level data with "REDACTED" except when a user has the central-office-admin group.



Global Data Policy Builder

Add Certification · SQL Support Matrix

What is the name of this policy?

mask

How should this policy protect the data?

Mask · columns tagged · Discovered > PII ✕ · add another tag (optional)

using a constant · REDACTED · for

everyone except · when user · is a member of group · Central Office Admin

+ Add Another Condition

Enter Rationale for Policy (Optional)

Update

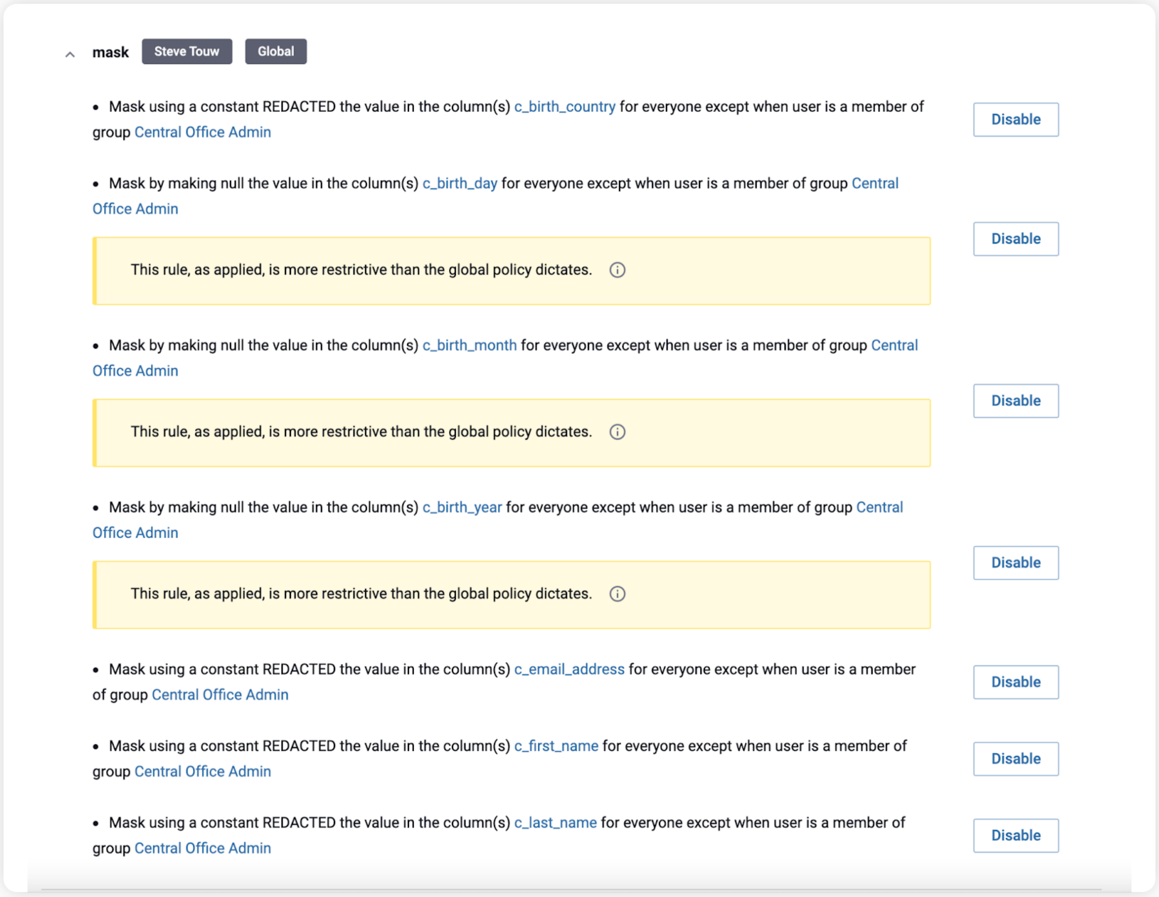
Where should this policy be applied?

On data sources · with columns tagged · Discovered > PII

+ Add Another Circumstance

Now, when central office employees with the central-office-admin attribute run a query on this table, they will be able to see the unmasked data, while sensitive data will appear as "REDACTED" to all other users.

But what about the different column types? Remember how we had to account for numeric versus varchar with the Snowflake policies? Immuta handles this through "smart fallbacks"—as you can see, the policy smartly fell back to nulling numeric columns because the "REDACTED" policy could not be applied (see where it switched to null and provides the warning):



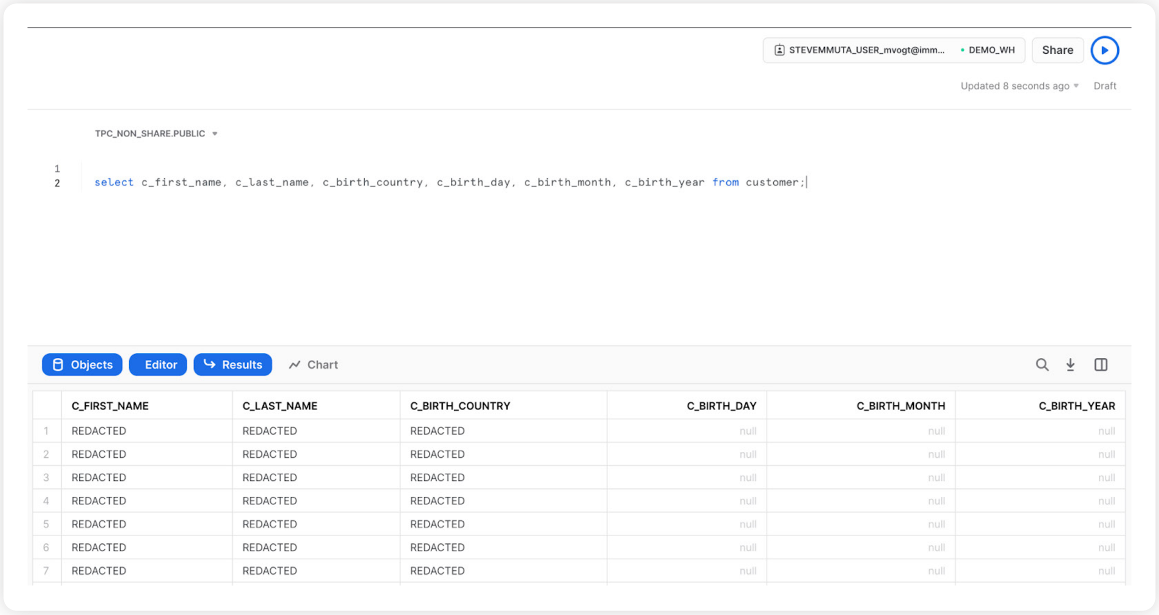
- Mask using a constant REDACTED the value in the column(s) c_birth_country for everyone except when user is a member of group Central Office Admin
- Mask by making null the value in the column(s) c_birth_day for everyone except when user is a member of group Central Office Admin

  This rule, as applied, is more restrictive than the global policy dictates.
- Mask by making null the value in the column(s) c_birth_month for everyone except when user is a member of group Central Office Admin

  This rule, as applied, is more restrictive than the global policy dictates.
- Mask by making null the value in the column(s) c_birth_year for everyone except when user is a member of group Central Office Admin

  This rule, as applied, is more restrictive than the global policy dictates.
- Mask using a constant REDACTED the value in the column(s) c_email_address for everyone except when user is a member of group Central Office Admin
- Mask using a constant REDACTED the value in the column(s) c_first_name for everyone except when user is a member of group Central Office Admin
- Mask using a constant REDACTED the value in the column(s) c_last_name for everyone except when user is a member of group Central Office Admin

Now, in Snowflake, instead of querying the data using the literal roles "Central Office Admin" and "Central Office Admin" the user can instead use the single role Immuta created for them; they don't have to think about which role to use to query what data anymore because of Immuta's dynamic enforcement. This screenshot depicts the single role in the upper right for this user and the policy enforced correctly:



```
select c_first_name, c_last_name, c_birth_country, c_birth_day, c_birth_month, c_birth_year from customer;
```

	C_FIRST_NAME	C_LAST_NAME	C_BIRTH_COUNTRY	C_BIRTH_DAY	C_BIRTH_MONTH	C_BIRTH_YEAR
1	REDACTED	REDACTED	REDACTED	null	null	null
2	REDACTED	REDACTED	REDACTED	null	null	null
3	REDACTED	REDACTED	REDACTED	null	null	null
4	REDACTED	REDACTED	REDACTED	null	null	null
5	REDACTED	REDACTED	REDACTED	null	null	null
6	REDACTED	REDACTED	REDACTED	null	null	null
7	REDACTED	REDACTED	REDACTED	null	null	null

Some key takeaways from the Immuta approach:

1. The user writing the policy doesn't have to know which physical tables or columns contain PII, they can simply write the policies abstractly.
2. Since the policy dynamically injects the user group at runtime, the Snowflake users don't need to worry about switching between appropriate roles. This is reflected in their single Immuta-managed user Role as you can see from the screenshot above.
3. The policy does not need to handle different column data types. Immuta has the capability to have "safe fallbacks" if a column type doesn't align with the masking policy For instance, we were able to take the "REDACTED" policy and dynamically switch it to nulling when it hit a numeric column type.

Snowflake Column-Level Security in Practice

For Snowflake users leveraging sensitive data, dynamic data access control and data masking capabilities are essential to ensuring the right users can access the right data at the right time, with minimal risk of unauthorized access or noncompliance.

Immuta's native integration with Snowflake eliminates the burdens of static, role-based access control, such as manually creating multiple policies and roles, managing those policies at the physical table-level, and ensuring that policy enforcement is consistent across platforms. This saves substantial time and risk, simplifies policy management and auditing, and allows data teams to onboard and scale workloads faster.

For organizations using Snowflake with Immuta, these capabilities have resulted in $1 million in data engineering savings and reduced time to data access from months to seconds.

Conclusion

By creating a safe general space and focusing protection on entry points and privileged areas, an amusement park enables the free movement and activity of its attendees. In the same way, Immuta's approach enables the safe exposure of all your Snowflake data assets, without solidifying today's governance decisions in the data model. Although your policies are certain to evolve, you will not have to re-architect the database or review a cadre of roles: all that will need to change is the relevant metadata and global policies.

Implementing fine-grained access controls for Snowflake is critical to achieve security standards, but understanding your data assets, how they're being accessed and used, and the potential risks to them are also necessary to enabling broad spectrum data security. Immuta's sensitive data discovery and activity monitoring augment the security features discussed in this eBook, as well as native Snowflake Data Governance capabilities. With discovery, security, and detection, Snowflake users can unlock more data at scale.

To see how this works for yourself and build a Snowflake access control policy, try our self-guided Snowflake demo. Or, to speak with a member of our team, request a demo today.

About Immuta

Immuta is the **leader in Data Security**, providing data teams one universal platform to control access to analytical data sets in the cloud. Only Immuta can automate access to data by discovering, protecting, and monitoring data. Data-driven organizations around the world trust Immuta to speed time to data, safely share more data with more users, and mitigate the risk of data leaks and breaches. Founded in 2015, Immuta is headquartered in Boston, MA.

IMMUTA®

25 Thomson Place, 4th Floor, Boston, MA 02210 immuta.com (800) 655-0982