ConductorOne

Snowflake Authorization and Permission Model Deep Dive

Snowflake is a popular cloud data warehouse known for its ability to handle storage and analytics at scale. In the decade since its launch, Snowflake has grown into a mature data warehouse with enterprise grade security and authorization management functionality.

Snowflake's robust authorization and permission model is central to how a company secures data in the platform. It provides a flexible method of authorization management utilizing role-based permissions, discretionary privileges, and a hierarchical structure for roles, users, and the objects they can access.

This article takes a deep dive into the entities and methodologies that comprise Snowflake's permission model and its relative strengths and limitations. We'll also explore common issues and best practices for your organization.

Snowflake's Permission Model

Snowflake uses two approaches for granting permissions: discretionary access control (DAC) and role-based access control (RBAC). The DAC model simply states that each Snowflake object has an owner or creator, and that this owner has access to the object and can, in turn, grant others access to it. With RBAC, on the other hand, access to Snowflake objects is first granted to roles, and those roles can then be assigned to individual users.

Terms Used in Snowflake Permissions

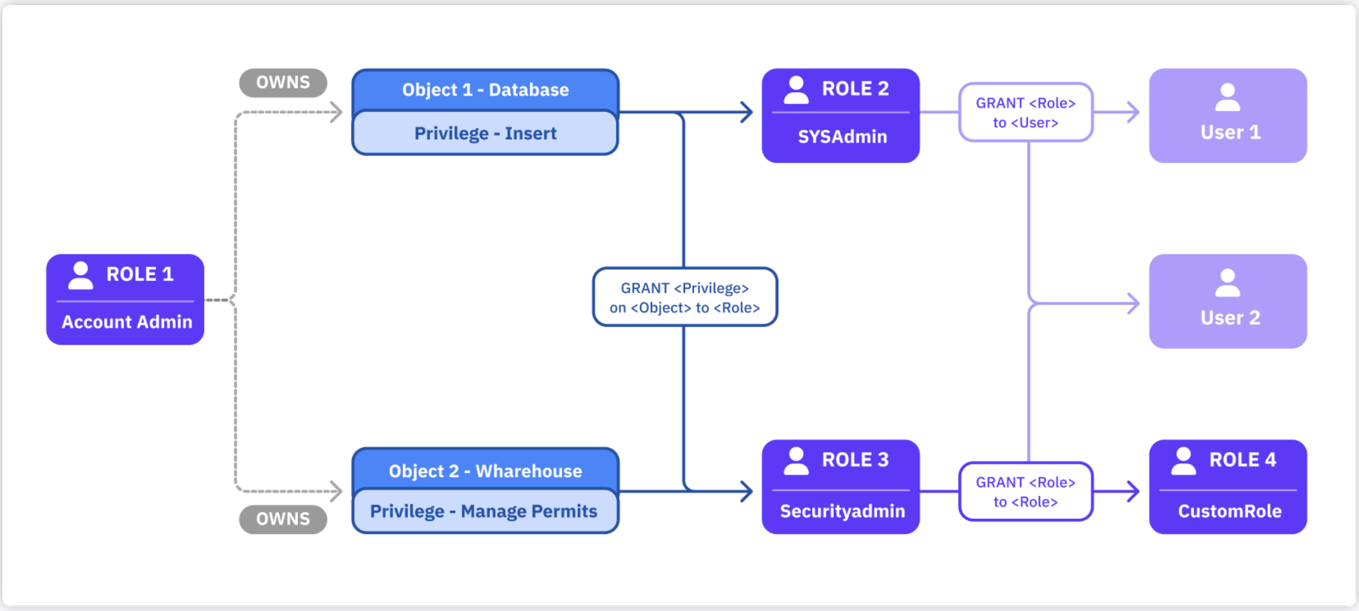
In Snowflake's permission model, some novel concepts are used that could be easily misinterpreted without proper context. Here are a few of the key concepts you need to take into account and their corresponding definition:

- **Securable objects:** These refer to entities within the Snowflake platform that can be secured and accessed with the appropriate permission. They include databases, warehouses, schemas, tables, and views. Access to these objects is granted or denied based on a role, which can be assigned to users and other roles.
- **Roles:** Roles in Snowflake are entities that enable users to perform various actions and/or grant permissions on objects. Users can be assigned multiple roles, allowing them to switch roles during a session to execute different actions that might require different permission levels. Additionally, roles can be granted to other roles, forming a hierarchical structure. This hierarchy enables inheritance, where privileges are inherited by roles lower in the structure.

- **Privileges:** For each securable object in Snowflake, there is a set of privileges that can be granted to it. A privilege refers to a specific level of access granted to a securable object. It determines who can access and perform operations on the object. Privileges are essential for controlling the granularity of access allowed, and multiple separate privileges might be used to manage different aspects of object access.

- **Users:** Users in Snowflake are distinct identities that represent individuals or machines that interact with the platform, access data, and execute various actions. Each user is linked to one or more roles, which dictate their access level. The privileges assigned to users are the combined privileges granted by all the roles assigned to them. Users can be associated with both built-in roles provided by Snowflake and custom roles created for specific purposes.

How Does It Work?



Snowflake effectively blends the DAC and RBAC approaches by utilizing the "ownership" privilege for securable objects like databases and assigning them to roles. In addition to "ownership," specific privileges like "insert" or "manage" can be individually assigned to other roles. These roles (along with their associated privileges) can be granted to users or other roles, resulting in an inheritance of all the assigned privileges. This hierarchical approach creates a robust system of permissions, granting varying levels of access to different roles and users.

Snowflake also implements a hierarchical structure to organize data effectively through three main types of data containers:

- **Databases:** These serve as top-level containers for data and can house multiple schemas.

- **Schemas:** Within databases, schemas act as logical containers that further organize data.

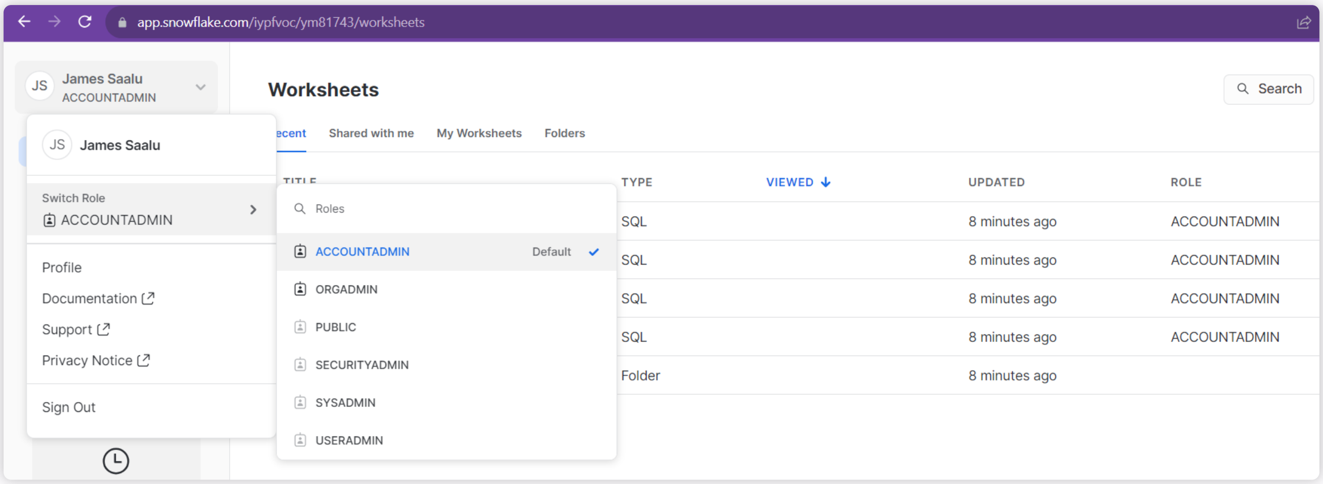
- **Tables:** These objects store the actual data and are organized within schemas.

- **Views:** These are virtual tables presenting data from one or more underlying tables. Views are commonly used to simplify complex queries or offer specific subsets of data.

Managing Access and Authorization in Snowflake

Snowflake offers an extensive system for managing access within your account, ensuring only authorized users and applications can access data and perform actions at each level of your Snowflake environment.

You can access and modify aspects of your Snowflake account permissions through the user interface or with explicit SQL commands. For example, you can explore the roles your user has access to through the account dropdown menu at the left of your Snowflake home page:

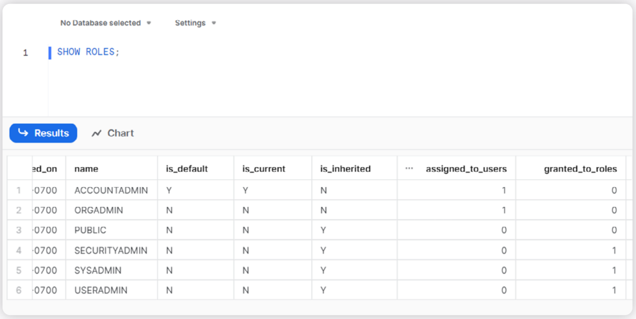


TITLE	TYPE	VIEWED	UPDATED	ROLE
	SQL		8 minutes ago	ACCOUNTADMIN
	SQL		8 minutes ago	ACCOUNTADMIN
	SQL		8 minutes ago	ACCOUNTADMIN
	SQL		8 minutes ago	ACCOUNTADMIN
	Folder		8 minutes ago	

You can also do this by executing the following command in a Snowflake worksheet:

```
SHOW ROLES;
```

This shows the available roles for your user account as well as information on the number of roles and users with connections to each role:



```
SHOW ROLES;
```

	ed_on	name	is_default	is_current	is_inherited	…	assigned_to_users	granted_to_roles
1	0700	ACCOUNTADMIN	Y	Y	N		1	0
2	0700	ORGADMIN	N	N	N		1	0
3	0700	PUBLIC	N	N	Y		0	0
4	0700	SECURITYADMIN	N	N	Y		0	1
5	0700	SYSADMIN	N	N	Y		0	1
6	0700	USERADMIN	N	N	Y		0	1

You can use the **GRANT** and **REVOKE** commands to manage privileges, giving or denying access to users and roles in your Snowflake environment.

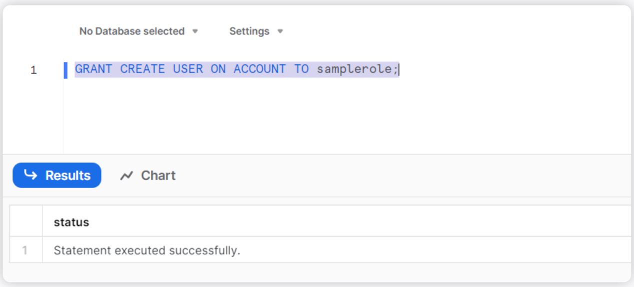
The following command creates a new role:

```
CREATE ROLE samplerole
```

The following code will grant this new role some privileges, though note that you can only grant privileges from authorized accounts that already have access to the target securable objects:

```
GRANT CREATE USER ON ACCOUNT TO samplerole;
```

This allows users with the **samplerole** role to create more users on your account:



```
GRANT CREATE USER ON ACCOUNT TO samplerole;
```

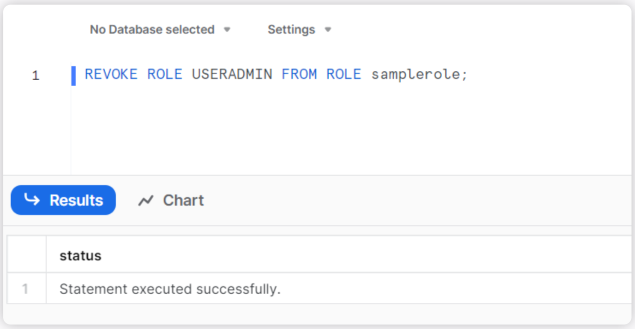
	status
1	Statement executed successfully.

Instead of individual privileges, you can also grant a role to another role or user to have them inherit a collection of privileges:

```
GRANT ROLE USERADMIN TO ROLE samplerole;
```

Revoking the privileges you granted uses the same command syntax. Replace **GRANT** with **REVOKE** and **TO** with **FROM**:

```
REVOKE ROLE USERADMIN FROM ROLE samplerole;
```

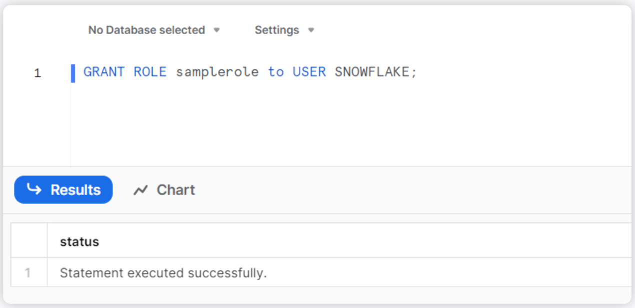


```
REVOKE ROLE USERADMIN FROM ROLE samplerole;
```

```
   status
1  Statement executed successfully.
```

You can also assign users to a specific role. For example, you can assign **samplerole** to a new user to have them gain the privileges attached to the role:

```
GRANT ROLE samplerole to USER SNOWFLAKE;
```



```
GRANT ROLE samplerole to USER SNOWFLAKE;
```

```
   status
1  Statement executed successfully.
```

The **SNOWFLAKE** user here is a default user on your account. You can utilize the following code to check all users in your Snowflake environment, just like with the role statement earlier:

```
SHOW USERS;
```

This helps you manage your users and the roles assigned to them.

Snowflake allows you to grant privileges at different levels of the account hierarchy (i.e. global, database, schema, and object level). The global level involves privileges that apply across your Snowflake account, such as the ability to create users and roles, as you saw earlier. You can also grant specific privileges for specific objects, such as databases and schemas.

Here are examples of Snowflake commands at different hierarchy levels of privilege:

```
--database level grant
GRANT USAGE ON DATABASE SAMPLE_DATA to ROLE
samplerole;

--schema level grant
GRANT USAGE ON SCHEMA SAMPLE_DATA.SAMPLE_
SCHEMA TO ROLE samplerole;

--object(table) level grant
GRANT SELECT ON TABLE SAMPLE_DATA.SAMPLE_
SCHEMA.SAMPLE_TABLE to ROLE samplerole;
```

When a privilege is granted at a higher object level, it can be inherited by lower-level objects within the hierarchy. For example, if a privilege is granted at the database level, it will apply to all schemas and objects within that database unless specifically overridden with different privileges at the schema or object level. This hierarchical / inheritance-driven approach eliminates the need to grant privileges for each individual object, making access control and privilege management more efficient.

Evaluation of Snowflake’s Permission Model

Snowflake’s permission model is a flexible system for managing access to your data and resources. However, it also has significant weaknesses.

Strengths

The following are Snowflake's main strengths:

- **Granularity and flexibility in managing access:** Snowflake offers intricate control over access management, allowing you to tailor permissions at varying levels of granularity. You can adjust to the individual needs of your users and ensure they have the necessary access without unnecessary privileges. This helps you align with the principle of least privilege, which reduces potential vulnerabilities and unauthorized access.
- **Ease of role-based management:** The use of roles simplifies your access management since they serve as a collection of privileges that you can grant to and revoke from users as needed. With Snowflake's role inheritance and hierarchy of permissions, you can reduce repetitive administrative tasks and effectively control access rights.
- **Support for complex hierarchical structures:** Snowflake offers extensive support for complex hierarchical structures in data management and organizational collaboration. The use of a hierarchical permission model with multilevel inheritance streamlines and centralizes permission management across your data architecture. It does so by reducing repetitive and redundant permission assignments and relying on the cascade of permissions through your hierarchical structure.

Limitations and Challenges

Some of Snowflake's notable limitations and challenges include the following:

- **Lack of fine-grained object-level permissions:** Snowflake's permission model does not natively support fine-grained access control at the object level. For example, it does not allow you to control access to specific columns within a table or individual elements within an object. That is, different users cannot access the same table while only seeing the specific columns they have access to. While the Snowflake Enterprise Edition does offer dynamic data masking and external tokenization, these features are geared more towards obfuscating or tokenizing data on the schema level to protect sensitive information than fine-grained access control at the individual column level.
- **Complexity in managing large numbers of roles and users:** In large organizations with numerous users and required roles, the management of permissions can become complex and challenging within Snowflake's unique framework. The more roles and users there are, the more difficult it becomes to ensure accurate and secure permission assignments. This is especially true with mishandled role inheritance, where roles are assigned without exploring all the privileges attached and how they might be misused.
- **Potential difficulties in auditing and tracking permissions:** As the number of roles and users grows, tracking and auditing permissions can become difficult within your Snowflake architecture. Without proper monitoring and audit trails, it may be challenging to identify and rectify any potential security breaches or unauthorized access.

Common Problems and Pain Points with Snowflake Access Control

While Snowflake stands out as a mature and effective solution, it's important to acknowledge the existence of certain common challenges and pain points within its access management. Let's delve into a closer examination of these aspects:

- **Overly permissive access:** With role inheritance, there is a high potential for your system to be overly permissive. When roles are granted to other roles, privileges may unintentionally propagate to users who shouldn't have them. You need to be vigilant and regularly review role assignments to ensure that access remains appropriately controlled.

- **Managing access across multiple environments and stages:** In organizations with multiple development, testing, and production environments, managing access control consistently across all stages can become complex. Different environments may have unique requirements, and coordinating access control settings across them may be challenging. Proper documentation, version control, and automated deployment processes are essential in order to maintain consistency and avoid misconfigurations within your Snowflake environment.

- **Multitenant environment:** Snowflake is a multitenant cloud service where multiple Snowflake accounts share the same underlying infrastructure. This has a significant influence on data privacy and security, as ensuring data isolation becomes critical in order to prevent unauthorized access to sensitive data. You should set up strong access controls and implement best practices for securing data within a multitenant setup.

- **Compliance challenges and regulatory requirements:** Different industries and regions may have specific compliance requirements and regulatory standards that your organization must adhere to. Ensuring that your access controls meet these requirements can be challenging, especially when data spans multiple geographic regions or involves data subject to various regulations.

Best Practices for Managing Authorization and Permissions in Snowflake

Let's explore a few best practices for ensuring the security of your data and utilizing Snowflake's authorization and permission model effectively:

- **Define a clear and logical role hierarchy:** Co-opt the Snowflake structure in your organization. Design a role hierarchy that reflects your organization's structure and access requirements. Create and organize your roles based on job functions, teams, or data access levels to ensure a logical and scalable hierarchy. Use role inheritance to simplify permissions management and avoid duplication of privileges.

- **Regularly review and audit permissions:** It's important to routinely monitor and review your permissions and security. This helps you better identify and rectify any permissive access cases and minimize your security risks. Tools like ConductorOne can help you manage and audit your permissions over your organization's infrastructure. ConductorOne provides you with a clear overview of permissions across roles and objects, helping you maintain a secure infrastructure.

- **Utilize the principle of least privilege:** By following the principle of least privilege, you only grant the minimum security privileges required at any point in time for a certain task. This reduces unnecessary or excessive permissions within your organization and minimizes the risk of data breaches or unauthorized access. ConductorOne is built around this principle, with "just-in-time" and short-lived access control to resources to reduce risk.

- **Leverage Snowflake's built-in security features:** Snowflake offers various security features that can enhance access control and data protection, such as end-to-end data encryption at rest and in transit, multifactor authentication, and secure data sharing. You can leverage these features to make your architecture secure and mitigate unauthorized access.

Conclusion

This article introduced the Snowflake environment, its authorization and permission model, and its features. You also explored its strengths and weaknesses, common problems, and best practices.

ConductorOne is an identity security solution for the modern workforce. It helps you simplify and automate your authorization workflows, implement JIT access control, centralize permissions management, and all the authorization best practices we discussed in this article, all while providing a seamless user experience.