# Snowflake Role-Based Access Control simplified

Syed Faisal · Follow
Published in Cognizant Servian · 5 min read · May 5, 2020




Role-Based Access Control or RBAC is part of Snowflake's Access Control Framework which allows privileges to be granted by Object Owners to Roles, and Roles, in turn, can be associated with Users to restrict/allow

actions to be performed on objects. Phew! Yes, this is what it takes to describe RBAC in a single sentence. Let us now break it down to understand and appreciate the components mentioned.

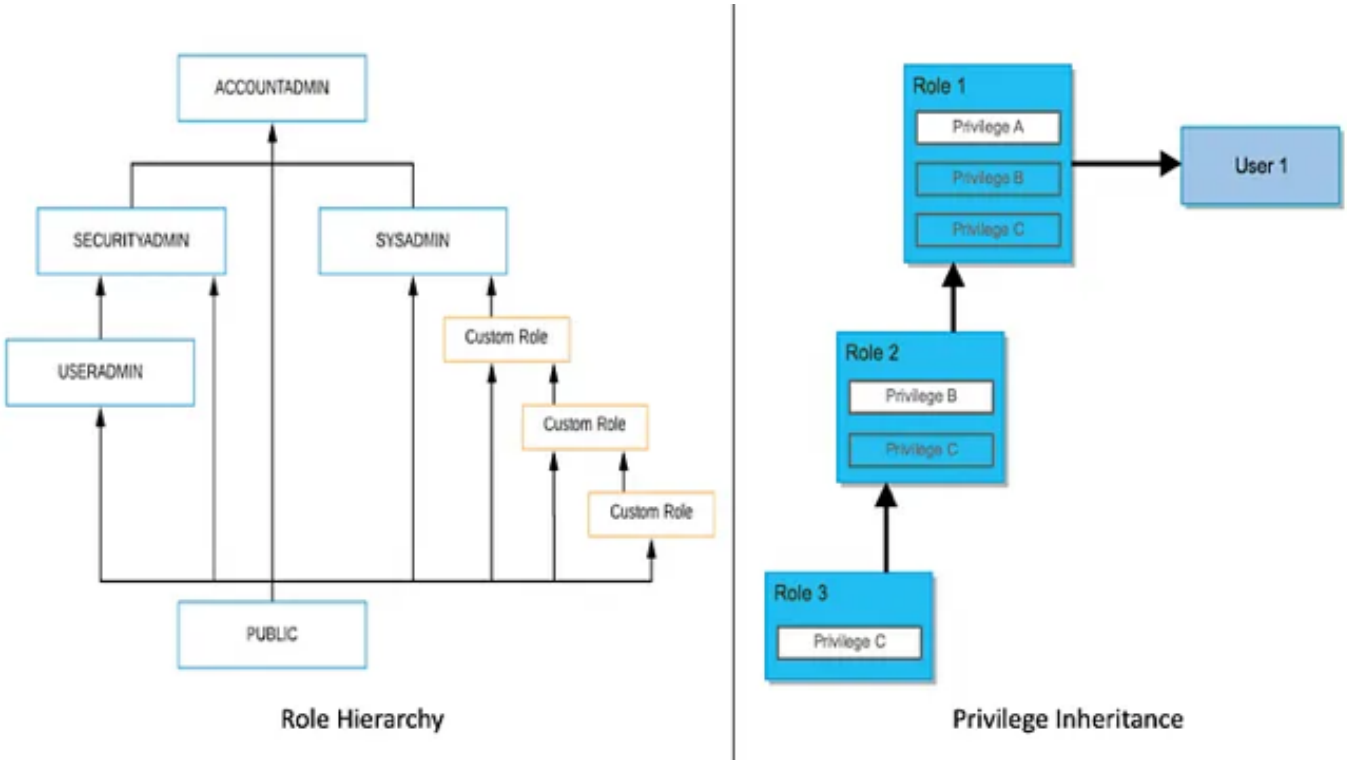
The central idea of the Access Control Framework is that every component on a Snowflake database is an Object and the privileges to these objects rest with Roles. The Role with which Object is created will be the owner and has Discretionary Access Control (DAC) on it. Once the object is created, RBAC comes into picture where the access privileges on the Object are assigned to Role(s) which in turn can be assigned to Users. Each user can be assigned multiple roles (and vice versa) but can assume only one role at a time. In simple terms we can summarize it as — actions on Snowflake Objects can be performed only by a user based on the privileges associated with the user's current role.

Now that we have a brief understanding of RBAC, it is a good time to note that Roles can be assigned to other Roles which means Role hierarchies are allowed in Snowflake. RBAC enables granular control on objects and also complex access privilege inheritance to accommodate every evolving organizational needs. All this sounds really exciting and RBAC surely has a lot of potentials to suit every need out there. Once we start to implement it, we generally have more questions than answers. How do we start with an RBAC set up on Snowflake? How can we have separate production, UAT, and dev environments and control it within the same Snowflake Account? How can we simplify on-going RBAC maintenance? We shall be answering all of these questions going forward.

Before I start discussing my perspective on how to approach these questions, let us look at the RBAC considerations which are part of Snowflake's official documentation.

For the benefit of the readers, I shall be using some excerpts and diagrams from the official site to explain the details. There are 5 system-defined roles and Snowflake defines what "role" each system role shall perform and it is recommended that users adhere to this.

> ***ACCOUNTADMIN**: Account Administrator role which encapsulates the SYSADMIN and SECURITYADMIN roles.*
>
> ***SECURITYADMIN**: Role that can monitor, and manage users and roles.*
>
> ***USERADMIN**: Role that creates roles and users.*
>
> ***SYSADMIN**: Role that has privileges to create warehouses and databases (and other objects) in an account. All Custom roles are expected to roll up to SYSADMIN.*
>
> ***PUBLIC**: Role that is automatically granted to every user and every role in your account.*
>
> *Apart from System roles, we can have Custom roles which can be created by SECURITYADMIN and to be rolled up to SYSADMIN as a manual process.*




From the Official Documentation

As part of the official privilege inheritance depiction, you can see that Snowflake allows each role to be assigned privileges directly and it can also have privileges by inheritance. Based on this particular idea, the hierarchical RBAC set up is extrapolated and is usually implemented for a broader organizational structure.

At this point, I would like to appreciate your patience in staying with me to understand the whole context of RBAC and it's widely used implementation method. Engineers implementing Snowflake out there can relate to it when I say, RBAC can get too complex too fast. It also takes some effort to maintain a complex hierarchical setup. Having stated that, let us now look at setting up a simple RBAC model.

## A Simplified RBAC Model

The most important aspect of this is to keep access privileges to objects mutually exclusive of privilege inheritance for a given Role. We shall demonstrate this with an RBAC prototype shortly. Once you have come to terms with the idea, the next step will be to divide roles into logical levels. This is to simplify the RBAC requirement capture and also to segregate object access and inheritance of privileges.

Let's briefly describe the logical levels:

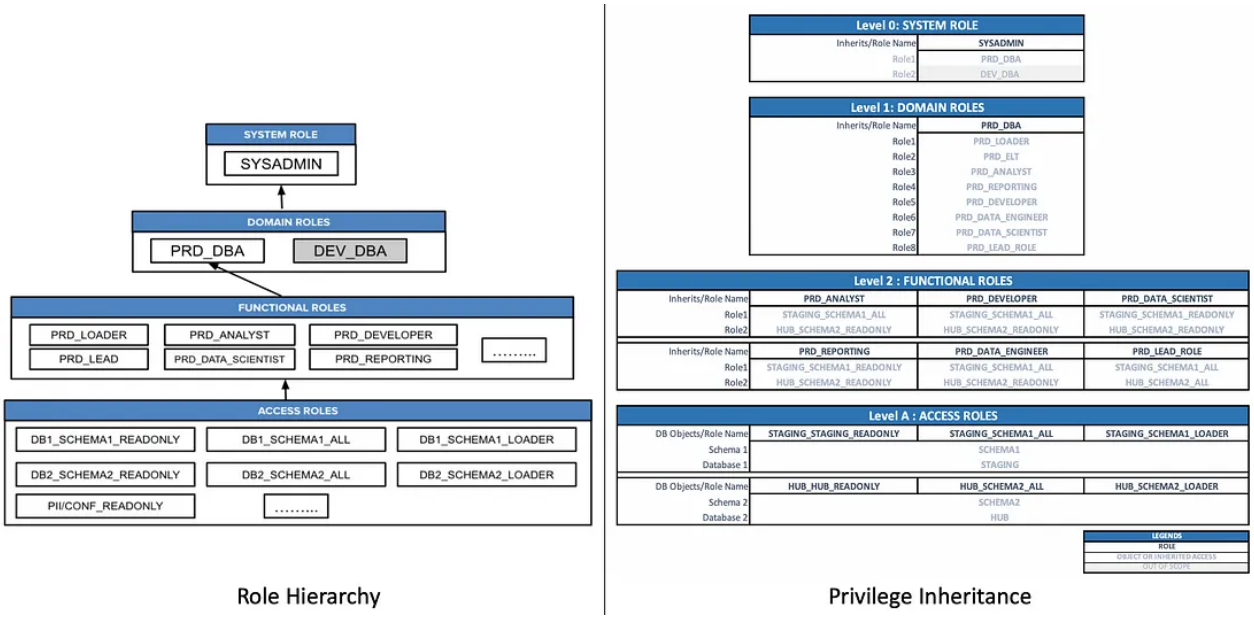
> ***Level A or Access Roles:*** *These roles will be the lowest level which will have actual access privileges on DB objects. Maintain distinct access roles based on user requirements at a schema level.*
>
> ***Level 2 or Functional Roles:*** *These roles map to the actual real-world roles of the users in the organization and are assigned to the Snowflake users.*
>
> ***Level 1 or Domain Roles:*** *If the organization has a requirement to have multiple independent domains under the same account, this will help to realize that. E.g. when there needs to be a separation of production or UAT/development domains.*
>
> ***Level 0 or System Roles:*** *Native Snowflake system role to which all the Domain roles should be rolling up to.*

To summarize, the level of granular access clients need shall define your Access Roles. The real-life roles will translate into Functional Roles. Based

on the number of environments, the Domain Roles shall be captured.

Now you know how to start your RBAC requirement capture!

Let us look at a prototype representation for a production environment. Having similar domain role hierarchies for Dev/UAT/Feature Store/Data Store as shown below will help us create logical domain segregations with granular object-level control on a single Snowflake account.




RBAC Model: Production Domain Prototype

> *In the above-discussed model, we have used Snowflake's flexibility in a simplified way. Inheritance is not present within a level and is allowed between levels in a controlled manner. Object access is provided as part of a flat structure and is not blended with inheritance, which sets this apart.*

This article on RBAC and the discussion of a simplified model is intended to facilitate a smooth onboarding of users to Snowflake's powerful Role-Based Access Control. It is important to note that the discussed approach is one of the ways to do the setup and it is not the only way to do it!

Snowflake | Rbac | Role Based Access Control | Snowflakedb | Logical Domains

# Responses (1)

PAVAN TEJA

What are your thoughts?

Sumit Rai
Aug 23, 2021

This is a very nice article.

One question, after creating domain roles, to whom do we assign it? I feel domain roles are optional, is it?

1 reply Reply

## More from Syed Faisal and Cognizant Servian