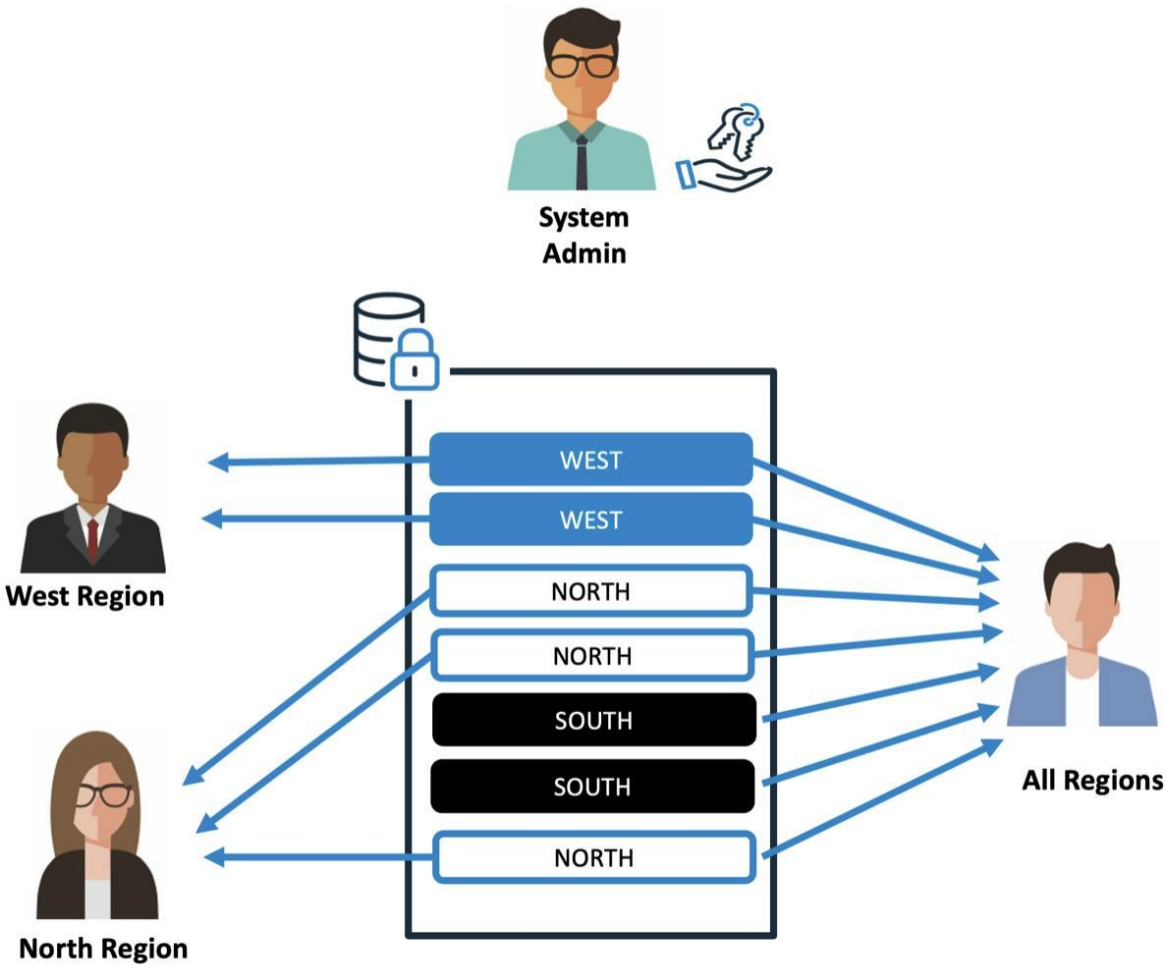
ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

1. What is Row-Level Security?

Row-Level Security is a security mechanism that limits the records returned from a database table based on the permissions provided to the currently logged-in user. Typically, this is done such that certain users can access only their data and are not permitted to view the data of other users.

The diagram below illustrates the primary challenge: ensuring data is only visible to specific individuals who need to know. If the user (or, more likely, their role) is not allowed to view the data, it's simply not visible. Even the DBAs, account admins, and table owners cannot access the data unless authorized.



While RBAC secures access to Tables, row level access is used to control access to sub-sets of the data. In the above diagram users are authorised to view data for one or more REGIONS, and the System Administrator has no access to the data at all.

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

Ideal Row Level Security Features

The ideal features of a Row Access Security system include:

- **Central Management:** The ability to define a single, centrally managed set of rules or policies controlling sensitive data access.
- **Easy Deployment:** Ideally, we'd like to define a policy once and deploy it against as many tables, schemas, or databases as needed.
- **Simplified Change Management:** With centrally defined policies, we'd like to change the data access rules without having to reapply them again.
- **Segregation of Duties:** This includes the ability of a central administrator to decide which data needs to be protected independently of the data owner.
- **Integration with RBAC:** The solution must integrate with the overall role-based access control (RBAC) architecture without adding significant complexity.

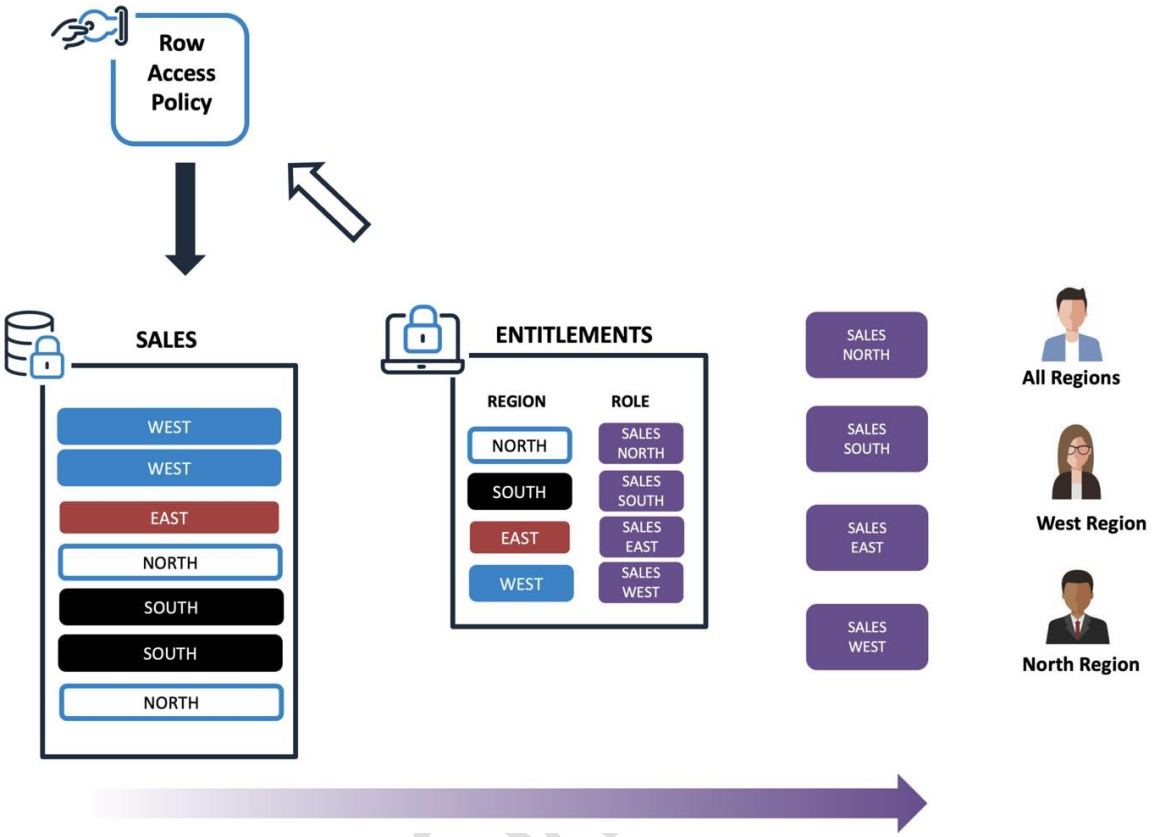
How does Snowflake implement Row Level Security?

Snowflake achieves all of the above using Row Access Policies. A row access policy is a small, centrally defined procedure that returns a Boolean value (TRUE or FALSE) depending on whether the user can view the specific row. The steps to defining row-level security include:

1. Decide which table or view needs to be secured. For example, the SALES table, which has access controlled by REGION.
2. Create an ENTITLEMENTS table that records the roles allowed to view the data for each REGION.
3. Create a **Row Access Policy** to implement the rule. This will typically query the ENTITLEMENTS table and return TRUE if the executing user has the relevant sensitive data role.
4. Deploy the **Row Access Policy** against the SALES table.

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

The diagram below illustrates the key components of the solution.



The components include:

- **Sales Table** - This holds the data we need to secure.
- **Entitlements Table** - Which records for each REGION which roles are allowed to view the data.
- **Row Access Policy** - Which enforces the security access rules.
- **Sensitive Data Roles** - These are granted to users to integrate the solution into the overall RBAC architecture.

Effectively, the ENTITLEMENTS table maps which ROLES are allowed to view data for each REGION, and the Row Access Policy implements the rule. Once a Row Access Policy is applied to the SALES table, nobody will have access unless they are granted the appropriate Sensitive Data Role, and even then, only to the data they are authorized to view.

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

In this article let us understand how to set up Row-Level Security on a database table using Row Access Policies in Snowflake.

2. What are Row Access Policies in Snowflake?

A Row Access Policy is a schema-level object that determines whether a given row in a table or view can be viewed by a user using the following types of statements.
1. SELECT statements
2. Rows selected by UPDATE, DELETE, and MERGE statements.

The row access policy should be added to a table or a view binding with a column present inside them. A row access policy can be added to a table or view either when the object is created or after the object is created.

3. Steps to implement Row-Level Security using Row Access Policies in Snowflake

Follow below steps to implement Row-Level Security using Row Access Policies in Snowflake.

1. Create a table to apply Row-Level Security
2. Create a Role Mapping table
3. Create a Row Access Policy
4. Add the Row Access Policy to a table
5. Create Custom Roles and their Role Hierarchy
6. Grant SELECT privilege on table to custom roles
7. Grant USAGE privilege on virtual warehouse to custom roles
8. Assign Custom Roles to Users
9. Query and verify Row-Level Security on table using custom roles
10. Revoke privileges on role mapping table to custom roles

3.1. Create a table to apply Row-Level Security

Let us consider a sample employees table as an example for the demonstration of row-level security using secure views.

The below SQL statements creates a table named *employees* with required sample data in *hr* schema of *analytics* database.

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

```
USE ROLE SYSADMIN;

CREATE OR REPLACE WAREHOUSE DEMO_WAREHOUSE;

CREATE OR REPLACE DATABASE ANALYTICS;

CREATE OR REPLACE SCHEMA ANALYTICS.HR;


-- Create a table to apply Row-Level Security

CREATE OR REPLACE TABLE ANALYTICS.HR.EMPLOYEES
(   employee_id number,
    first_name varchar(50),
    last_name varchar(50),
    email varchar(50),
    hire_date date,
    country varchar(50)
);


-- INSERT VALUES

INSERT INTO
analytics.hr.employees(employee_id,first_name,last_name,email,hire_date,country)
VALUES
(100,'Steven','King','SKING@outlook.com','2013-06-17','US'),
(101,'Neena','Kochhar','NKOCHHAR@outlook.com','2015-09-21','US'),
(102,'Lex','De Haan','LDEHAAN@outlook.com','2011-01-13','US'),
(103,'Alexander','Hunold','AHUNOLD@outlook.com','2016-01-03','UK'),

(104,'Bruce','Ernst','BERNST@outlook.com','2017-05-21','UK'),
(105,'David','Austin','DAUSTIN@outlook.com','2015-06-25','UK'),
(106,'Valli','Pataballa','VPATABAL@outlook.com','2016-02-05','CA'),
```

(107,'Diana','Lorentz','DLORENTZ@outlook.com','2017-02-07','CA'),

(108,'Nancy','Greenberg','NGREENBE@outlook.com','2012-08-17','CA');

```
select * from analytics.hr.employees;
```

Results | Chart

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	COUNTRY
1	100	Steven	King	SKING@outlook.com	2013-06-17	US
2	101	Neena	Kochhar	NKOCHHAR@outlook.com	2015-09-21	US
3	102	Lex	De Haan	LDEHAAN@outlook.com	2011-01-13	US
4	103	Alexander	Hunold	AHUNOLD@outlook.com	2016-01-03	UK
5	104	Bruce	Ernst	BERNST@outlook.com	2017-05-21	UK
6	105	David	Austin	DAUSTIN@outlook.com	2015-06-25	UK
7	106	Valli	Pataballa	VPATABAL@outlook.com	2016-02-05	CA
8	107	Diana	Lorentz	DLORENTZ@outlook.com	2017-02-07	CA
9	108	Nancy	Greenberg	NGREENBE@outlook.com	2012-08-17	CA

3.2. Create a Role Mapping table

The below SQL statements creates mapping table named ***role_mapping*** which stores the country and corresponding role to be assigned for the users of that country as shown below.

-- Create a role mapping table

```
CREATE OR REPLACE TABLE  ANALYTICS.HR.ROLE_MAPPING
(
   country varchar(50),
   role_name varchar(50)
);
```

--Insert Values Into Tables

```
INSERT INTO ANALYTICS.HR.ROLE_MAPPING(country, role_name)
VALUES
('US','DATA_ANALYST_ROLE_US'),
('UK','DATA_ANALYST_ROLE_UK'),
('CA','DATA_ANALYST_ROLE_CA');
```

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

```
45  select * from role_mapping;
46
```

Results | Chart

	COUNTRY	ROLE_NAME
1	US	DATA_ANALYST_ROLE_US
2	UK	DATA_ANALYST_ROLE_UK
3	CA	DATA_ANALYST_ROLE_CA

3.3. Create a Row Access Policy

The below SQL statement creates a Row Access Policy with following two conditions.

1. User with SYSADMIN role can query all rows of the table.
2. User with DATA_ANALYST roles can query only rows belonging to their country based on the role mapping table.

```
use role SYSADMIN;

create or replace row access policy analytics.hr.country_role_policy as (country_name varchar) returns boolean ->

 'SYSADMIN' = current_role()

    or exists (

        select 1 from role_mapping

         where role_name = current_role()

           and country = country_name

       )

;
```

In the above statement:

country_role_policy specifies the name of the policy.

country_name is the signature of the row access policy which specifies the field and data type of the mapping table to which it links.

returns boolean -> specifies the application of the row access policy.

'SYSADMIN' = current_role() is the first condition of row access policy which allows users with SYSDAMIN role to view all rows of the table.

or exists … is the second condition of the row access policy expression which uses a subquery. The subquery requires the CURRENT_ROLE to be the custom role which specifies the country through role mapping table. This is used by row access policy to limit the rows to be returned for the query executed by user.

3.4. Add the Row Access Policy to a table

The below SQL statement adds the row access policy named *country_role_policy* to the table *employees* on *country* field.

```
use role SYSADMIN;

alter table analytics.hr.employees

add row access policy analytics.hr.country_role_policy on (country);
```

3.5. Create Custom Roles and their Role Hierarchy

The below SQL statements creates custom roles mentioned in the role mapping table to assign to the users in later stage.

```
use role SECURITYADMIN;

create or replace role DATA_ANALYST_ROLE_US;

create or replace role DATA_ANALYST_ROLE_UK;

create or replace role DATA_ANALYST_ROLE_CA;
```

When the roles are created, they exist in isolation not allowing the other roles (even the roles which create and grant privileges to them) to access the objects created by them. So, it is required to set up a role hierarchy for the custom roles we created.

The below SQL statements assigns the custom roles to the role SYSADMIN so that the SYSADMIN can inherit all the privileges assigned to custom role.

```
use role SECURITYADMIN;

grant role DATA_ANALYST_ROLE_US to role SYSADMIN;

grant role DATA_ANALYST_ROLE_UK to role SYSADMIN;

grant role DATA_ANALYST_ROLE_CA to role SYSADMIN;
```

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

use role SYSADMIN;

grant usage on database analytics to role DATA_ANALYST_ROLE_US;

grant usage on schema analytics.hr to role DATA_ANALYST_ROLE_US;

grant select on all tables in schema analytics.hr to role DATA_ANALYST_ROLE_US;

grant usage on database analytics to role DATA_ANALYST_ROLE_UK;

grant usage on schema analytics.hr to role DATA_ANALYST_ROLE_UK;

grant select on all tables in schema analytics.hr to role DATA_ANALYST_ROLE_UK;

grant usage on database analytics to role DATA_ANALYST_ROLE_CA;

grant usage on schema analytics.hr to role DATA_ANALYST_ROLE_CA;

grant select on all tables in schema analytics.hr to role DATA_ANALYST_ROLE_CA;

Grant USAGE privilege on virtual warehouse to custom roles

The below SQL statements provides usage privileges on warehouse ***DEMO_WAREHOUSE*** to the custom roles to query tables.

use role ACCOUNTADMIN;

grant usage on warehouse DEMO_WAREHOUSE to role DATA_ANALYST_ROLE_US;

grant usage on warehouse DEMO_WAREHOUSE to role DATA_ANALYST_ROLE_UK;

grant usage on warehouse DEMO_WAREHOUSE to role DATA_ANALYST_ROLE_CA;

3.8. Assign Custom Roles to Users

Let us consider there are three users TONY, STEVE and BRUCE belonging to US, UK and CA respectively.

The below SQL statements assigns the custom roles to the users belonging to the respective countries.

use role SECURITYADMIN;

grant role DATA_ANALYST_ROLE_US to user TONY;

grant role DATA_ANALYST_ROLE_UK to user STEVE;

grant role DATA_ANALYST_ROLE_CA to user BRUCE;

3.9. Query and verify Row-Level Security on table using custom roles

Let us verify the data returned for each user when queried on the same table.

The below image shows that for user with role *DATA_ANALYST_ROLE_US* when queried on the table *employees,* the data returned is only from country US.

```
110  -- Query and verify Row-Level Security on table using custom roles
111  USE ROLE DATA_ANALYST_ROLE_US;
112  SELECT * FROM ANALYTICS.HR.EMPLOYEES;   -- VISIBLE ONLY US DATA
```

Results | Chart

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	COUNTRY
1	100	Steven	King	SKING@outlook.com	2013-06-17	US
2	101	Neena	Kochhar	NKOCHHAR@outlook.com	2015-09-21	US
3	102	Lex	De Haan	LDEHAAN@outlook.com	2011-01-13	US

Query returning only US data when queried with DATA_ANALYST_ROLE_US role

The below image shows that for user with role *DATA_ANALYST_ROLE_UK* when queried on the table *employees,* the data returned is only from country UK.

```
114  USE ROLE DATA_ANALYST_ROLE_UK;
115  SELECT * FROM ANALYTICS.HR.EMPLOYEES; --  VISIBLE ONLY UK DATA
```

Results | Chart

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	COUNTRY
1	103	Alexander	Hunold	AHUNOLD@outlook.com	2016-01-03	UK
2	104	Bruce	Ernst	BERNST@outlook.com	2017-05-21	UK
3	105	David	Austin	DAUSTIN@outlook.com	2015-06-25	UK

Query returning only UK data when queried with DATA_ANALYST_ROLE_UK role

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

The below image shows that for user with role *DATA_ANALYST_ROLE_CA* when queried on the table *employees,* the data returned is only from country CA.

```
117  USE ROLE DATA_ANALYST_ROLE_CA;
118  SELECT * FROM ANALYTICS.HR.EMPLOYEES; -- VISIBLE ONLY CA DATA
```

Results | Chart

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	COUNTRY
1	106	Valli	Pataballa	VPATABAL@outlook.com	2016-02-05	CA
2	107	Diana	Lorentz	DLORENTZ@outlook.com	2017-02-07	CA
3	108	Nancy	Greenberg	NGREENBE@outlook.com	2012-08-17	CA

Query returning only CA data when queried with DATA_ANALYST_ROLE_CA role

The below image shows that when the user with role *SYSADMIN* queries on the table *employees,* all rows are returned.

```
120  USE ROLE SYSADMIN;
121  SELECT * FROM ANALYTICS.HR.EMPLOYEES; -- ALL DATA VISIBLE
```

Results | Chart

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	COUNTRY
1	100	Steven	King	SKING@outlook.com	2013-06-17	US
2	101	Neena	Kochhar	NKOCHHAR@outlook.com	2015-09-21	US
3	102	Lex	De Haan	LDEHAAN@outlook.com	2011-01-13	US
4	103	Alexander	Hunold	AHUNOLD@outlook.com	2016-01-03	UK
5	104	Bruce	Ernst	BERNST@outlook.com	2017-05-21	UK
6	105	David	Austin	DAUSTIN@outlook.com	2015-06-25	UK
7	106	Valli	Pataballa	VPATABAL@outlook.com	2016-02-05	CA
8	107	Diana	Lorentz	DLORENTZ@outlook.com	2017-02-07	CA
9	108	Nancy	Greenberg	NGREENBE@outlook.com	2012-08-17	CA

Query returning all rows when queried with SYSADMIN role

3.10. Revoke privileges on role mapping table to custom roles

Since we have provided SELECT privilege on all tables, the user can also access the role mapping table which is used to limit the access to the users.

To avoid this you could either create this role mapping table in a different schema to which the users do not have access or simply revoke the access on this particular table.

The below SQL statement revokes all privileges on table *role_mapping* to the custom roles.

```
use role SYSADMIN;

revoke all privileges on table analytics.hr.role_mapping from role
DATA_ANALYST_ROLE_US;

revoke all privileges on table analytics.hr.role_mapping from role
DATA_ANALYST_ROLE_UK;
```

revoke all privileges on table analytics.hr.role_mapping from role DATA_ANALYST_ROLE_CA;

4. How to Remove a Row Access Policy on a table in Snowflake?

The below SQL statement removes a row access policy on a table.

alter table <table_name> drop row access policy <policy_name>;

The below SQL statement removes all row access policy associations from a table.

alter table <table_name> drop all row access policies;

5. How to Extract information of existing Row Access Policies in Snowflake?

5.1. SHOW ROW ACCESS POLICIES

Lists the row access policies for which the user have access privileges. It returns information of creation date, database and schema names, owner, and any available comments.

The below SQL statement extracts the row access policies present in the database and schema of the current session.

show row access policies;

```
131  show row access policies;
132
```

Results | Chart

	created_on	name	database_name	schema_name	kind	owner	comment	owner_role_type	options
1	2025-01-13 02:43:31.205 -0800	COUNTRY_ROLE_POLICY	ANALYTICS	HR	ROW_ACCESS_POLICY	SYSADMIN		ROLE	

Show Row Access Policies – Listing all Row Access Policies

5.2. DESCRIBE ROW ACCESS POLICY

Describes the current definition of a row access policy, including the creation date, name, data type, and SQL expression.

The below SQL statement extracts information of the row access policy *country_role_policy*.

describe row access policy country_role_policy;

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

```
describe row access policy country_role_policy;
```

Results | Chart

	name	signature	return_type	body
1	COUNTRY_ROLE_POLICY	(COUNTRY_NAME VARCHAR)	BOOLEAN	'SYSADMIN' = current_role() or exists (select 1 from role_mapping

body

```
'SYSADMIN' = current_role()
      or exists (
            select 1 from
role_mapping
               where
role_name = current_role()
                  and country
= country_name
            )
```

Describe Row Access Policy – Extracting details of a Row Access Policy

6. How to Rename a Row Access Policy in Snowflake?

The below SQL statement renames row access policy from *row_policy1* to *row_policy2*.

alter row access policy row_policy1 rename to row_policy2;

7. How to Update a Row Access Policy in Snowflake?

To update an existing row access policy,

- If you need to see the current definition of the policy, run the DESCRIBE ROW ACCESS POLICY command.
- The row access policy expression can then be updated with the ALTER ROW ACCESS POLICY command.

The below SQL statement updates the SQL expression that filters the data in the row access policy.

alter row access policy <policy name> set body -> <expression_on_val>;

The expression can include conditional expression functions to represent conditional logic, built-in functions, or UDFs to transform the data.

8. How to Drop a Row Access Policy in Snowflake?

A Row Access Policy cannot be dropped successfully if it is currently attached to a resource. Before executing a DROP statement, detach the row access policy from the table or view.

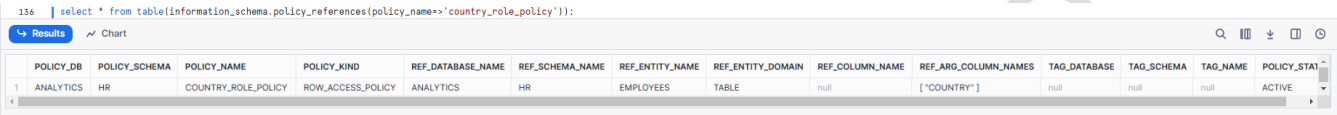
ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

Follow below steps to drop a row access policy in Snowflake

1. Find the objects on which the row access policy is attached.

The below SQL statement lists all the objects on which row access policy named *country_role_policy* is attached.

select * from table(information_schema.policy_references(policy_name=>'country_role_policy'));



Finding all objects on which Role Access Policy is applied

2. Remove the row access policy from all the tables and views to which it is associated. (*refer section-4 of the article*)

3. Drop the row access policy.

The below SQL statement drops row access policy named *country_role_policy*.

drop row access policy country_role_policy;

9. Closing Points

Few key points to keep in mind related to row access policies.

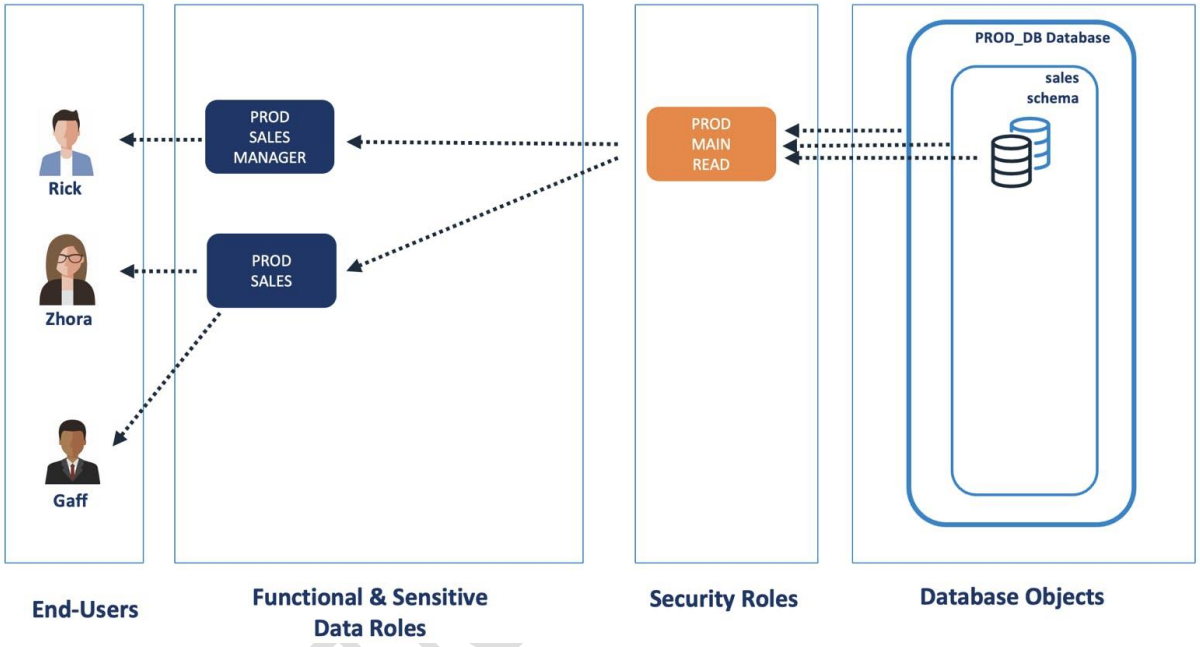
- If a table column has a row access policy attached to it, the column cannot be dropped from the table.
- Snowflake does not support UNDROP with row access policy objects.
- Snowflake does not support using external tables as a mapping table in a row access policy.
- A table or view column can only be protected by one row access policy at a time. Adding a policy fails if the policy body refers to a table or view column that is protected by a row access policy.
- If an object has both a row access policy and one or more Column-level Security masking policies, the row access policy is evaluated first.

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES

Integrating the solution with RBAC

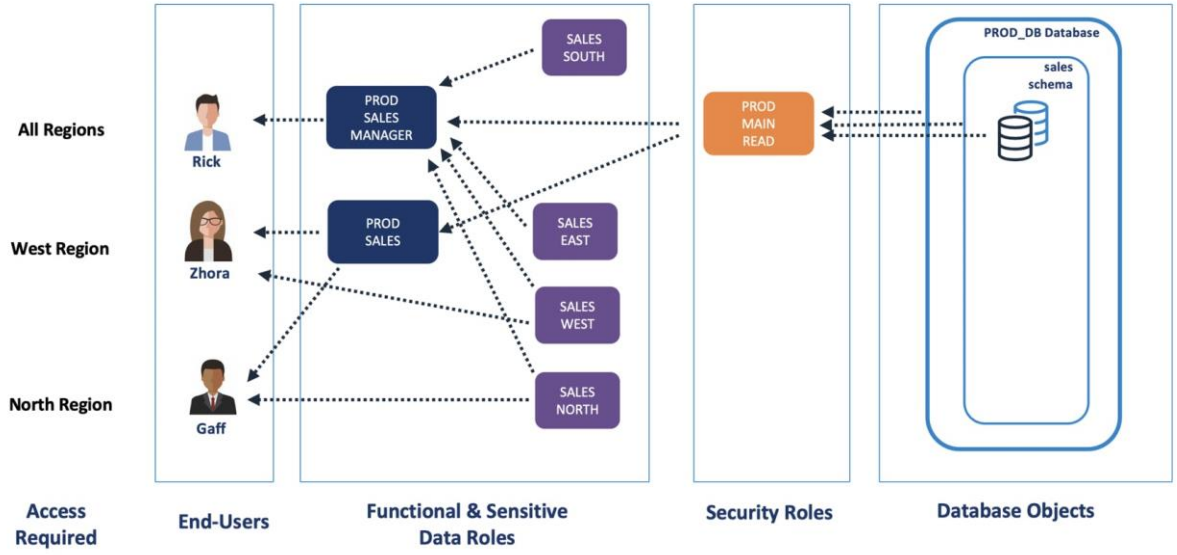
Role-based access Control (RBAC) can be extremely challenging to design and maintain, and it's important to avoid adding to an already complex problem. However, because of the way we implemented row-level security, it's remarkably easy to deploy.

The diagram below illustrates a simple RBAC architecture that allows the SALES team and their MANAGERS the ability to read data from the SALES schema.



When we deploy row-based access control, we need to deploy a sequence of ROLES to control access to the sensitive data. The diagram below illustrates a potential solution whereby the roles are either granted directly to individual users or (in the case of the management team) to the PROD_SALES_MANAGER role.

ROW-LEVEL SECURITY(RLS) USING ROW ACCESS POLICIES



Notice that the Sensitive Data Roles don't need to be directly granted access to the underlying data. This means we keep the RBAC solution utterly separate from handling sensitive data, which hugely simplifies the solution and allows us to separate these two challenges.