Row Access Policy Design Document

Document Control

Document Status

Date	Version	Status

Document Authors

Name	Role
Naresh Kala	Platform Team Member
Koustubh Kasalkar	Offshore Lead
Syed Fayaz	Onshore Lead

Reviewers and Approvers

Name	Role	Review Section	Jira Ticket /Status	Version

Endorsers and Approvers

Name	Role	Purpose	Version	Date	Status

Key Revision History

Version	Date	Description	Author(s)	Status
0.1	15/10/2025	First Draft	Naresh Kala	Draft

Introduction

Purpose

This document describes the design and implementation of Row Access Policies (RAP) in Snowflake to enforce zone-based data segregation for the FinOps platform. The solution ensures that users can only access cost and usage data relevant to their assigned organizational zones.

Scope

A simple application of a row access policy is to specify an attribute in the policy and a role that is allowed to see that attribute in the query result. The advantage of simple policies like this is that there is a negligible performance cost for Snowflake to evaluate these policies to return query results compared to using row access policies with mapping tables.

Database: PLATFORM__AUDIT__PREPROD
Schema: FINOPS_CURATED
Tables:

- credit_usage
- storage_usage

View

- warehouse_recommender
- warehouse_rightsizing
- batch_job_optimization
- user_tag_access_detailes
- table_privileges
- warehouse_summary

Policy Type: Row-Level Security (RLS) using Row Access Policies

Key Benefits

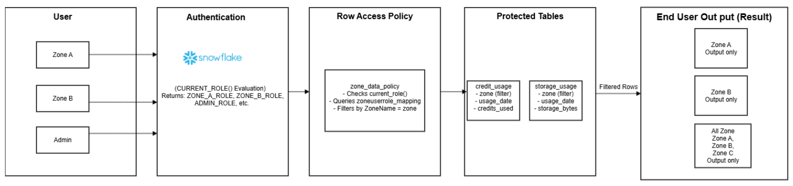
Data Privacy: Ensures users see only authorized zone data.
Compliance: Meets regulatory requirements for data segregation.
Centralized Control: Single mapping table manages all access.
Audit Trail: All access attempts are logged
Scalability: Easy to add new zones and roles

Overview of Row Access Policies

Row Access Policies are schema-level objects that enable granular, row-level security on tables and views. They provide a mechanism to control which rows a user or role can access within a given table or view, based on defined conditions.

Technical Architecture

Provides a visual and conceptual understanding of how the RAP system works:



Components Overview

A table describing each technical component (policies, tables, schemas) with their types and purposes. This serves as a quick reference guide.

Components	Type	Purpose
credit_usage	Protected Table	FinOps Credit consumption data
storage_usage	Protected Table	FinOps Storage utilization data
warehouse_recommender	Protected View	Identify oversized and undersized warehouses
batch_job_optimization	Protected View	Optimize 70% of credit spend from automated jobs

user_tag_access_detailes	Protected View	Finding Access by Classification Tags
table_privileges	Protected View	Finding all Tables and Fields a user has access
warehouse_summary	Protected View	Finding warehouses and credits consumed

Implementation Details

Contains the technical specifications and actual code for implementing the RAP solution:

Database Objects

Row Access Policy

The security logic that based on CURRENT_ROLE() a OR IS_ROLE_IN_SESSION()

```
 CREATE OR REPLACE ROW ACCESS POLICY
    platform__audit__preprod.finops_curated.zone_data_policy
AS (zone VARCHAR(30)) RETURNS BOOLEAN ->
    -- Multi-zone: Matches BDH_SF_FR_PLATFRM_*_DEVELOPER/ADMIN
    REGEXP_LIKE(CURRENT_ROLE(),
             '^BDH_SF_FR_PLATFRM_.*_(DEVELOPER|ADMIN)$',
             'i')
    -- Single zone: Matches BDH_SF_FR_{ZONE}_*_SupportAdmin
    OR REGEXP_LIKE(CURRENT_ROLE(),
               '^BDH_SF_FR_' || zone || '_.*_SUPPORTADMIN$',
               'i')
    -- System admins
    OR CURRENT_ROLE() IN ('ACCOUNTADMIN', 'SYSADMIN', 'SECURITYADMIN','BDH_SF_SVC_FR_PLATFRM_PREPROD_AIRFLOW')
      -- Check all enabled secondary roles
    OR IS_ROLE_IN_SESSION('BDH_SF_FR_' || zone || '_DEV_SUPPORTADMIN')
    OR IS_ROLE_IN_SESSION('BDH_SF_FR_' || zone || '_SYST_SUPPORTADMIN')
    OR IS_ROLE_IN_SESSION('BDH_SF_FR_' || zone || '_PPTE_SUPPORTADMIN')
    OR IS_ROLE_IN_SESSION('BDH_SF_FR_' || zone || '_PROD_SUPPORTADMIN')
    ;
```

Logic:

- Accepts **zone** column value as input parameter
- Returns **TRUE** if current role has mapping for the zone
- Returns **FALSE** otherwise (row is hidden)

Policy Application

SQL statements that attach the policy to protected tables.

```
-- Associate the policy to the table
ALTER table platform__audit__preprod.finops_curated.credit_usage
     ADD ROW ACCESS POLICY platform__audit__preprod.finops_curated.zone_data_policy ON (zone);


ALTER table platform__audit__preprod.finops_curated.storage_usage
     ADD ROW ACCESS POLICY platform__audit__preprod.finops_curated.zone_data_policy ON (zone);
```

Remove old Row access policy

```
use database platform__audit__preprod;
use schema finops_curated;
ALTER TABLE platform__audit__preprod.finops_curated.credit_usage
 DROP ROW ACCESS POLICY zone_data_policy;



ALTER TABLE platform__audit__preprod.finops_curated.storage_usage
 DROP ROW ACCESS POLICY zone_data_policy;
DROP ROW ACCESS POLICY IF EXISTS platform__audit__preprod.finops_curated.zone_data_policy;
```

Role Hierarchy

Shows the Snowflake role structure and inheritance chain, helping understand privilege delegation and administrative boundaries.

ACCOUNTADMIN (Full Access)
|
|**SECURITYADMIN** (Policy Management)
|
|**SYSADMIN** (Object Ownership)
|
Zone-Specific Roles
BDH_SF_FR_PLATFRM_PREPROD_ADMIN(Multi-Zone)
BDH_SF_FR_LOANS_DEV_SUPPORTADMIN (Zone: LOANS)
BDH_SF_FR_PTP_DEV_SUPPORTADMIN (Zone: PTP)
BDH_SF_FR_CUST_DEV_SUPPORTADMIN (Zone: CUST)

Privilege Model

A table showing which roles have what permissions and whether RAP filtering applies to them.

Role	Privileges	RAP Applied?
Account Admin	All privileges	No (bypasses)
SysAdmin	Object Ownership	No (bypasses)
SecurityAdmin	Policy Management	No (bypasses)
Zone-specific roles	SELECT (filtered)	YES

Security Model

Describes how security is enforced and supported

Authentication Flow

Step-by-step breakdown of how a user query is evaluated, from login through role assumption to filtered results.

1. User authenticates to Snowflake
2. User assumes a role: USE ROLE <role_name>
3. User queries protected table
4. RAP evaluates: CURRENT_ROLE() against mapping table
5. Query returns filtered results based on zone access

Authorization Matrix

Below table shows exactly what data each role type can access across all zones. This is essential for security audits and access reviews.

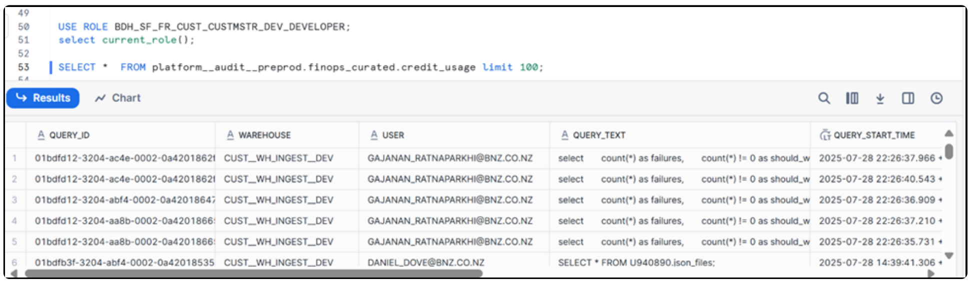
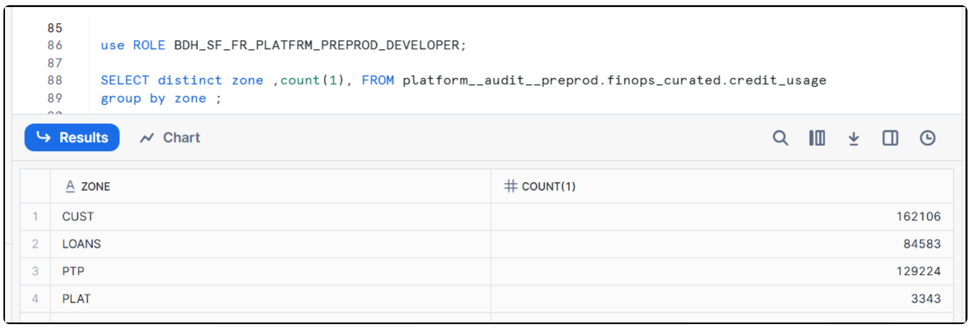
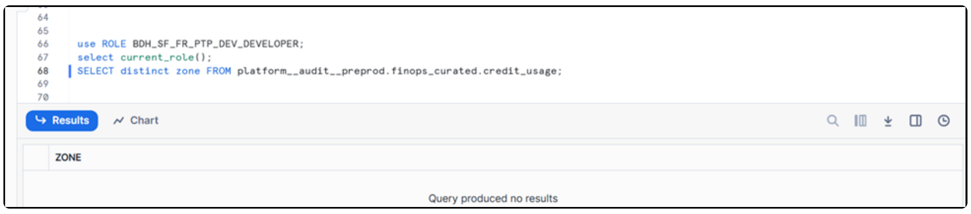
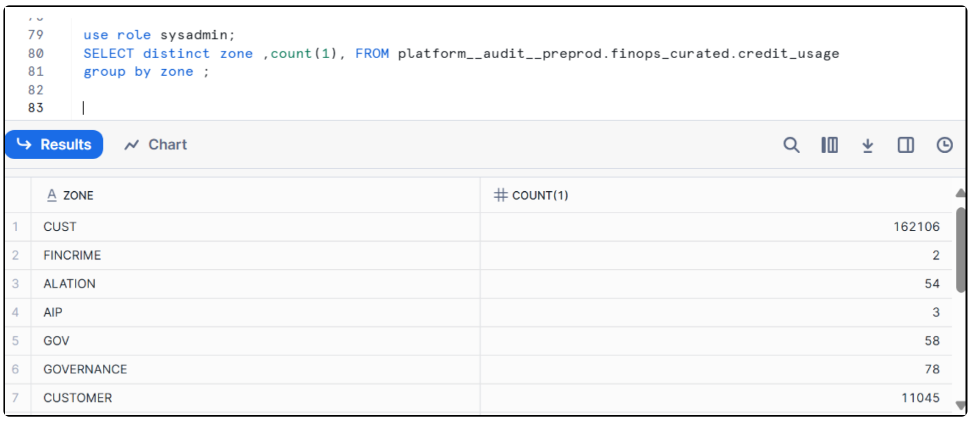
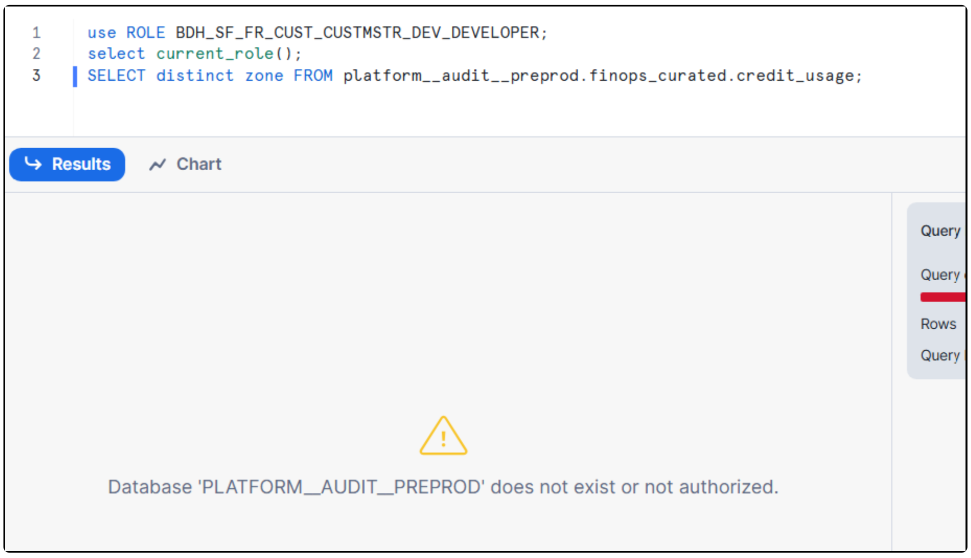
User Role	Zone - Cust	Zone - PTP	Zone - Loans	All -Zones
ACCOUNTADMIN	Yes	Yes	Yes	Yes
SYSADMIN	Yes	Yes	Yes	Yes
BDH_SF_FR_PLATFRM_PREPROD_ADMIN	Yes	Yes	Yes	Yes
BDH_SF_FR_LOANS_DEV_SUPPORTADMIN	No	No	Yes	No
BDH_SF_FR_PTP_DEV_SUPPORTADMIN	No	Yes	No	No
BDH_SF_FR_CUST_DEV_SUPPORTADMIN	Yes	No	No	No
Unmapped Role	No	No	No	No

Testing Strategy

Test Scenarios

A table of test cases covering positive tests (authorized access), negative tests (unauthorized access), and administrative bypass scenarios. Each test has an ID, scenario description, and expected result.

Test ID	Scenario	Expected Result	Output Result

TC-01	User with single zone access	Sees only mapped zone data	
TC-02	User with multiple zone access	Sees all mapped zones	
TC-03	User with no zone mapping	Sees zero rows	
TC-04	SYSADMIN access	Sees all zones (bypasses RAP)	
TC-05	Invalid role name in mapping	No access granted	

```
USE ROLE BDH_SF_FR_CUST_CUSTMSTR_DEV_DEVELOPER;
select current_role();

SELECT *  FROM platform__audit__preprod.finops_curated.credit_usage limit 100;
```

	QUERY_ID	WAREHOUSE	USER	QUERY_TEXT	QUERY_START_TIME
1	01bdfd12-3204-ac4e-0002-0a42018621	CUST_WH_INGEST_DEV	GAJANAN_RATNAPARKHI@BNZ.CO.NZ	select count(*) as failures, count(*) != 0 as should_w	2025-07-28 22:26:37.966 +
2	01bdfd12-3204-ac4e-0002-0a42018621	CUST_WH_INGEST_DEV	GAJANAN_RATNAPARKHI@BNZ.CO.NZ	select count(*) as failures, count(*) != 0 as should_w	2025-07-28 22:26:40.543 +
3	01bdfd12-3204-abf4-0002-0a42018647	CUST_WH_INGEST_DEV	GAJANAN_RATNAPARKHI@BNZ.CO.NZ	select count(*) as failures, count(*) != 0 as should_w	2025-07-28 22:26:36.909 +
4	01bdfd12-3204-aa8b-0002-0a4201866	CUST_WH_INGEST_DEV	GAJANAN_RATNAPARKHI@BNZ.CO.NZ	select count(*) as failures, count(*) != 0 as should_w	2025-07-28 22:26:37.210 +
5	01bdfd12-3204-aa8b-0002-0a4201866	CUST_WH_INGEST_DEV	GAJANAN_RATNAPARKHI@BNZ.CO.NZ	select count(*) as failures, count(*) != 0 as should_w	2025-07-28 22:26:35.731 +
6	01bdfb3f-3204-abf4-0002-0a42018535	CUST_WH_INGEST_DEV	DANIEL_DOVE@BNZ.CO.NZ	SELECT * FROM U940890.json_files;	2025-07-28 14:39:41.306 +

```

use ROLE BDH_SF_FR_PLATFRM_PREPROD_DEVELOPER;

SELECT distinct zone ,count(1), FROM platform__audit__preprod.finops_curated.credit_usage
group by zone ;
```

	ZONE	COUNT(1)
1	CUST	162106
2	LOANS	84583
3	PTP	129224
4	PLAT	3343

```


use ROLE BDH_SF_FR_PTP_DEV_DEVELOPER;
select current_role();
SELECT distinct zone FROM platform__audit__preprod.finops_curated.credit_usage;
```

	ZONE

Query produced no results

```
use role sysadmin;
SELECT distinct zone ,count(1), FROM platform__audit__preprod.finops_curated.credit_usage
group by zone ;

```

	ZONE	COUNT(1)
1	CUST	162106
2	FINCRIME	2
3	ALATION	54
4	AIP	3
5	GOV	58
6	GOVERNANCE	78
7	CUSTOMER	11045

```
use ROLE BDH_SF_FR_CUST_CUSTMSTR_DEV_DEVELOPER;
select current_role();
SELECT distinct zone FROM platform__audit__preprod.finops_curated.credit_usage;
```

Database 'PLATFORM__AUDIT__PREPROD' does not exist or not authorized.

Conclusion

We added automatic security filters to our financial data. People in different zones specific role now see only their own data. It uses standard Snowflake features, so it's reliable and easy to maintain.