

## Context

Tracking read/write operations	2 – 3
Secure View	3 – 5
Row-level Security	6 – 13
Tagging	14 – 16

# Tracking read/write operations

1. Querying QUERY\_HISTORY view:
2. This view contains a record of all queries executed in your account.
3. You can filter queries based on their type (e.g., SELECT for read operations, INSERT/UPDATE/DELETE for write operations).
4. Query to find recent read operations.

Query: SELECT \*

FROM "SNOWFLAKE"."ACCOUNT\_USAGE"."QUERY\_HISTORY"

WHERE start\_time >= DATEADD(HOUR, -72, CURRENT\_TIMESTAMP()) -- Filter for last 72 hour

AND query\_type = 'SELECT';

The screenshot shows the Snowflake Query Editor interface. At the top, there's a dropdown menu set to "SAMPLE\_SNOW" and "Snowflake". Below it, a SQL query is entered in a text area:

```
1 SELECT *
2 FROM "SNOWFLAKE"."ACCOUNT_USAGE"."QUERY_HISTORY"
3 WHERE start_time >= DATEADD(HOUR, -72, CURRENT_TIMESTAMP()) -- Filter for last 72 hour
4 AND query_type = 'SELECT';
```

Below the query editor, there's a "Results" tab selected. It shows a table with two columns: "QUERY\_ID" and "QUERY\_TEXT". The table contains 11 rows of query results. To the right of the table, there's a "Query Details" panel showing "Query duration" as 5.2s, "Rows" as 121, and "Query ID" as 01b51035-0001-2586-... Below this, there are two sections: "QUERY\_ID" and "QUERY\_TEXT", each with a "100% filled" progress bar. At the bottom right, there's a "DATABASE\_ID" section with a "#" symbol.

QUERY_ID	QUERY_TEXT
01b50024-0001-2513-0000-0004e88e21f1	SELECT system\$GET_NPS_FEEDBACK_TIMESTAMP(), system\$GET_NPS_DISMI
01b5003f-0001-2513-0000-0004e88e22f1	CALL SYSTEM\$SHOW_SCHEMAS_IN_ACCOUNT_WITH_CREATE_FILE_BASED_EI
01b4ff98-0001-2512-0000-0004e88e3085	SELECT system\$GET_NPS_FEEDBACK_TIMESTAMP(), system\$GET_NPS_DISMI
01b5003f-0001-2513-0000-0004e88e22ed	SELECT SYSTEM\$ARE_ANACONDA_TERMS_ACKNOWLEDGED()
01b4ff9b-0001-2513-0000-0004e88e20cd	Select * from EMPLOYEE_DETAILS
01b50041-0001-2512-0000-0004e88e332f	select tag_id, tag_name, tag_database, tag_schema, count(*), concat(tag_data
01b5002c-0001-2513-0000-0004e88e224f	SELECT sale_id, sale_date, customer_id, product_id, amount FROM sales WHER
01b5003f-0001-2512-0000-0004e88e3295	SELECT system\$GET_NPS_FEEDBACK_TIMESTAMP(), system\$GET_NPS_DISMI
01b5003f-0001-2512-0000-0004e88e32a1	SELECT SYSTEM\$SHOW_NOTEBOOKS_IN_ACCOUNT()
01b50041-0001-2513-0000-0004e88e238f	select SYSTEM\$LIST_ENTITIES_BOTTOMUP_CHECK('TAG', 5000)
01b50041-0001-2513-0000-0004e88e239f	select query_parameterized_hash, sum(t.num_queries) as num_queries, max

5. Query to find recent write operations.

Query: SELECT \*

FROM "SNOWFLAKE"."ACCOUNT\_USAGE"."QUERY\_HISTORY"

WHERE start\_time >= DATEADD(HOUR, -72, CURRENT\_TIMESTAMP()) -- Filter for last 72 hour

AND query\_type IN ('INSERT', 'UPDATE', 'DELETE');

## 6. Check the data.

The screenshot shows a Snowflake query editor with a query that filters for the last 72 hours of INSERT, UPDATE, and DELETE queries. The results pane shows a table with two columns: QUERY\_ID and QUERY\_TEXT. The table contains 8 rows of data, all of which are INSERT statements into the Employee\_Details table. The right-hand pane shows query details, including a duration of 2.0s and 8 rows returned.

```
1 SELECT *
2 FROM "SNOWFLAKE"."ACCOUNT_USAGE"."QUERY_HISTORY"
3 WHERE start_time >= DATEADD(HOUR, -72, CURRENT_TIMESTAMP()) -- Filter for last 72 hour
4 AND query_type IN ('INSERT', 'UPDATE', 'DELETE');
5
```

	QUERY_ID	QUERY_TEXT
1	01b4ff9c-0001-2512-0000-0004e88e30dd	insert into Employee_Details(id, Name, age, Salary, Gender,contact_number, Err
2	01b4ff9c-0001-2512-0000-0004e88e30c9	insert into Employee_Details(id, Name, age, Salary, Gender,contact_number, Err
3	01b4ff9c-0001-2512-0000-0004e88e30d9	insert into Employee_Details(id, Name, age, Salary, Gender,contact_number, Err
4	01b4ff9c-0001-2512-0000-0004e88e30d1	insert into Employee_Details(id, Name, age, Salary, Gender,contact_number, Err
5	01b4ff9c-0001-2512-0000-0004e88e20d5	insert into Employee_Details(id, Name, age, Salary, Gender,contact_number, Err
6	01b4ff9c-0001-2512-0000-0004e88e30b9	insert into Employee_Details(id, Name, age, Salary, Gender,contact_number, Err
7	01b50039-0001-2512-0000-0004e88e3261	INSERT INTO MY_TABLE (ID, NAME) SELECT MY_CUSTOM_SEQUENCE.NEXTVAL
8	01b50025-0001-2512-0000-0004e88e31f9	INSERT INTO sales (sale_id, sale_date, customer_id, product_id, amount) VALUE

## Secure View

### 1. Create a table and insert the data with the below query.

Query: create or replace table Patient(  
Id int,  
Full\_Name varchar(40),  
City varchar(50),  
Org\_Id varchar(50)  
)

Insert into Patient(Id, Full\_Name, City, Org\_Id) values  
(1, 'Sanjay Kumar', 'Dallas', 'Baylor Hospital'),  
(2, 'Suraj Jain', 'New York', 'Baylor Hospital'),  
(3, 'Jim DCosta', 'New York', 'Mayo Clinic'),  
(4, 'John Leboyd', 'New York, Dallas', 'Mayo Clinic');

The screenshot shows a Snowflake query editor with a query that creates a table named Patient and inserts four rows of data. The results pane shows a table with two columns: number of rows inserted and Query Details. The table contains 1 row of data, with 4 rows inserted. The right-hand pane shows query details, including a duration of 2.0s and 4 rows returned.

```
1 create or replace table Patient(
2 Id int,
3 Full_Name varchar(40),
4 City varchar(50),
5 Org_Id varchar(50)
6 )
7
8 Insert into Patient(Id, Full_Name, City, Org_Id) values
9 (1, 'Sanjay Kumar', 'Dallas', 'Baylor Hospital'),
10 (2, 'Suraj Jain', 'New York', 'Baylor Hospital'),
11 (3, 'Jim DCosta', 'New York', 'Mayo Clinic'),
12 (4, 'John Leboyd', 'New York, Dallas', 'Mayo Clinic');
13
```

	number of rows inserted	Query Details
1	4	Query duration

2. Create a secure view with the below query.

Query: CREATE OR REPLACE SECURE VIEW ORG\_PATIENTS AS  
SELECT \* FROM PATIENT ;

The screenshot shows the Snowflake query editor interface. The query text is: `1 CREATE OR REPLACE SECURE VIEW ORG_PATIENTS AS`  
`2 SELECT * FROM PATIENT ;`  
`3`. Below the query editor, the 'Results' tab is selected, showing a single row with the status: 'View ORG\_PATIENTS successfully created.' The right sidebar shows 'Query Details' and 'Query duration'.

3. Check the view data.

The screenshot shows the Snowflake query editor interface. The query text is: `1 select * from ORG_PATIENTS;`. Below the query editor, the 'Results' tab is selected, displaying a table with 4 rows and 5 columns: ID, FULL\_NAME, CITY, and ORG\_ID. The data is as follows:

	ID	FULL_NAME	CITY	ORG_ID
1	1	Sanjay Kumar	Dallas	Baylor Hospital
2	2	Suraj Jain	New York	Baylor Hospital
3	3	Jim DCosta	New York	Mayo Clinic
4	4	John Leboyd	New York, Dallas	Mayo Clinic

The right sidebar shows 'Query Details', 'Query duration', 'Rows', and 'Query ID'.

4. Select the current account. Copy the account name.

The screenshot shows the Snowflake query editor interface. The query text is: `1 select CURRENT_ACCOUNT();`. Below the query editor, the 'Results' tab is selected, showing a single row with the current account name: 'HR09072'. The right sidebar shows 'Query Details' and 'Query duration'.

5. Create a User Access table with the below query. Here give your account name for the Baylor Hospital.

Query: create table USER\_ACCESS  
(  
    ORG\_ID VARCHAR(50),  
    ACCOUNT\_ID varchar(50)  
);

insert into USER\_ACCESS values ('Baylor Hospital', HR09072);  
insert into USER\_ACCESS values ('Mayo Clinic','PO51568');

"SAMPLE\_SNOW": "Snowflake" Settings Code Versions

```

1 Create table USER_ACCESS
2 (
3     ORG_ID VARCHAR(50),
4     ACCOUNT_ID varchar(50)
5 );
6
7 insert into USER_ACCESS values ('Baylor Hospital', 'HR09072');
8 insert into USER_ACCESS values ('Mayo Clinic', 'P051568');

```

Results Chart

	number of rows inserted
1	1

Query Details

Query duration 43

6. Create a secure view using both the tables.

Query: CREATE OR REPLACE SECURE VIEW ORG\_PATIENTS AS  
 SELECT FULL\_NAME,CITY,USER\_ACCESS.ORG\_ID FROM patient  
 INNER JOIN USER\_ACCESS ON PATIENT.ORG\_ID=USER\_ACCESS.ORG\_ID  
 WHERE ACCOUNT\_ID=CURRENT\_ACCOUNT();

"SAMPLE\_SNOW": "Snowflake" Settings Code Versions

```

1 CREATE OR REPLACE SECURE VIEW ORG_PATIENTS AS
2 SELECT FULL_NAME,CITY,USER_ACCESS.ORG_ID FROM patient
3 INNER JOIN USER_ACCESS ON PATIENT.ORG_ID=USER_ACCESS.ORG_ID
4 WHERE ACCOUNT_ID=CURRENT_ACCOUNT();

```

Results Chart

	status
1	View ORG_PATIENTS successfully created.

Query Details

Query duration 137ms

7. Now check the data.

"SAMPLE\_SNOW": "Snowflake" Settings Code Versions

```

1 select * from ORG_PATIENTS;

```

Results Chart

	FULL_NAME	CITY	ORG_ID
1	Sanjay Kumar	Dallas	Baylor Hospital
2	Suraj Jain	New York	Baylor Hospital

Query Details

Query duration 777ms

Rows 2

## 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.

### Create a table to apply Row-Level Security

1. Let us consider a sample employee table as an example for the demonstration of row-level security using secure views.
2. The below SQL statements create a table named employees with required sample data in the HR schema of the analytics database.

Query: use role SYSADMIN;

```
create or replace database analytics;
```

```
create or replace schema analytics.hr;
```

```
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 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')  
;
```

ANALYTICS.HR Settings

```

1  use role SYSADMIN;
2
3  create or replace database analytics;
4
5  create or replace schema analytics.hr;
6
7  create or replace table analytics.hr.employees(
8      employee_id number,
9      first_name varchar(50),
10     last_name varchar(50),
11     email varchar(50),
12     hire_date date,
13     country varchar(50)
14 );
15
16 INSERT INTO analytics.hr.employees(employee_id,first_name,last_name,email,hire_date,country) VALUES
17 ('100','Steven','King','SKING@outlook.com','2013-06-17','US'),
18 ('101','Neena','Kochhar','NKOCHHAR@outlook.com','2015-09-21','US'),
19 ('102','Lex','De Haan','LDEHAAN@outlook.com','2011-01-13','US'),

```

Results Chart

	number of rows inserted
1	9

Query

## Create a Role Mapping table

- The below SQL statements create mapping table named role\_mapping which stores the country and corresponding role to be assigned for the users of that country as shown below.

Query: use role SYSADMIN;

```

create or replace table analytics.hr.role_mapping(
    country varchar(50),
    role_name varchar(50)
);

```

```

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')
;

```

ANALYTICS.HR Settings

```

1  use role SYSADMIN;
2
3  create or replace table analytics.hr.role_mapping(
4      country varchar(50),
5      role_name varchar(50)
6  );
7
8  INSERT INTO analytics.hr.role_mapping(country, role_name) VALUES
9      ('US','DATA_ANALYST_ROLE_US'),
10     ('UK','DATA_ANALYST_ROLE_UK'),
11     ('CA','DATA_ANALYST_ROLE_CA')
12 ;

```

Results Chart

	number of rows inserted
1	3

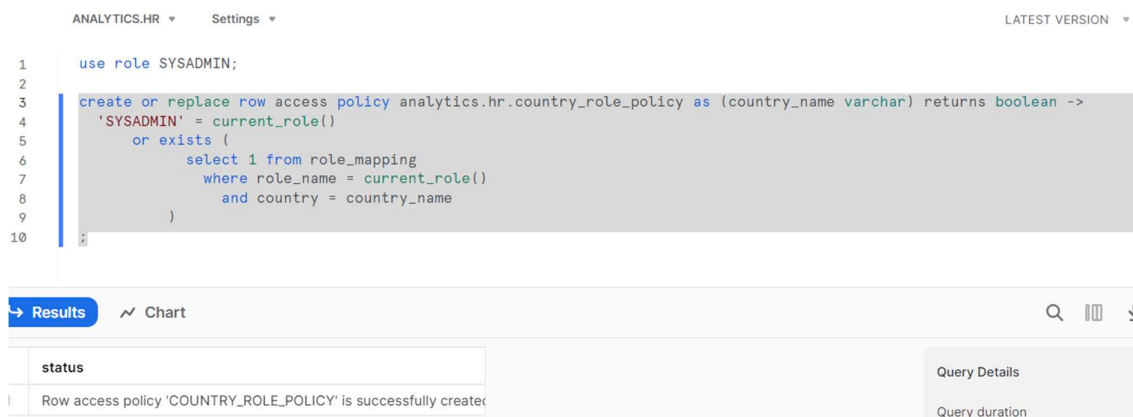
Query

## Create a Row Access Policy

4. The below SQL statement creates a Row Access Policy with following two conditions.
5. User with SYSADMIN role can query all rows of the table.
6. User with DATA\_ANALYST roles can query only rows belonging to their country based on the role mapping table.

Query: 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  
)  
;
```



7. In the above statement:
8. **country\_role\_policy** specifies the name of the policy.
9. **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.
10. **returns boolean** -> specifies the application of the row access policy.
11. **'SYSADMIN' = current\_role()** is the first condition of row access policy which allows users with SYSADMIN role to view all rows of the table.
12. **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.



## Add the Row Access Policy to a table

13. The below SQL statement adds the row access policy named country\_role\_policy to the table employees on country field.

Query: use role SYSADMIN;

alter table analytics.hr.employees

add row access policy analytics.hr.country\_role\_policy on (country);

The screenshot shows a SQL IDE interface with a toolbar at the top containing 'ANALYTICS.HR' and 'Settings'. The SQL editor contains four lines of code: `1 use role SYSADMIN;`, `2` (blank), `3 alter table analytics.hr.employees`, and `4 add row access policy analytics.hr.country_role_policy on (country);`. Below the editor is a 'Results' tab with a 'status' column. The first row shows the message 'Statement executed successfully.'

	status
1	Statement executed successfully.

## Create Custom Roles and their Role Hierarchy

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

Query: 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;

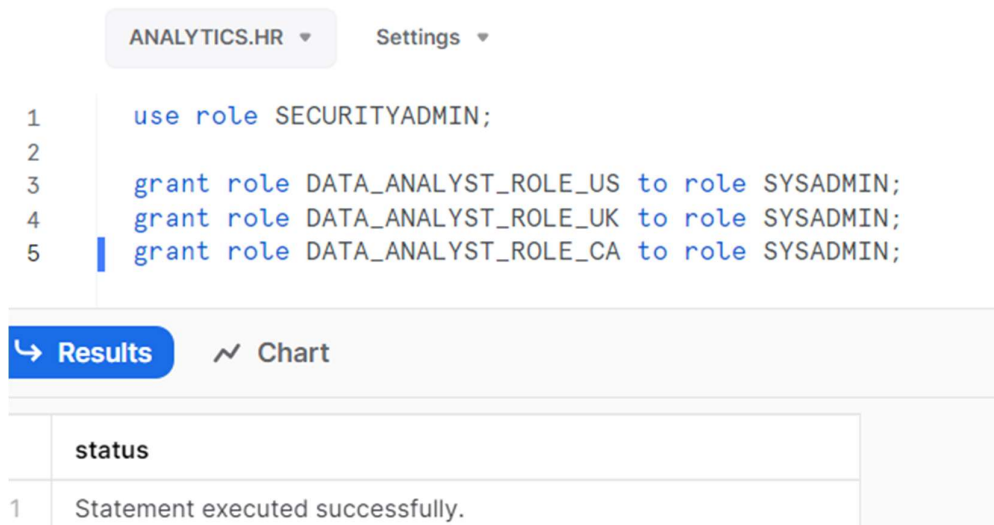
The screenshot shows a SQL IDE interface with a toolbar at the top containing 'ANALYTICS.HR' and 'Settings'. The SQL editor contains five lines of code: `1 use role SECURITYADMIN;`, `2` (blank), `3 create or replace role DATA_ANALYST_ROLE_US;`, `4 create or replace role DATA_ANALYST_ROLE_UK;`, and `5 create or replace role DATA_ANALYST_ROLE_CA;`. Below the editor is a 'Results' tab with a 'status' column. The first row shows the message 'Role DATA\_ANALYST\_ROLE\_CA successfully created.'

	status
1	Role DATA_ANALYST_ROLE_CA successfully created.

15. 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.
16. So, it is required to set up a role hierarchy for the custom roles we created.
17. 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.

Query: 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;
```



ANALYTICS.HR ▾ Settings ▾

```
1 use role SECURITYADMIN;  
2  
3 grant role DATA_ANALYST_ROLE_US to role SYSADMIN;  
4 grant role DATA_ANALYST_ROLE_UK to role SYSADMIN;  
5 grant role DATA_ANALYST_ROLE_CA to role SYSADMIN;
```

Results Chart

	status
1	Statement executed successfully.

## Grant SELECT privilege on table to custom roles

18. The below SQL statements grants usage privileges on database analytics and schema hr present inside it with only select privilege on all tables present inside them to the custom roles created.

Query: 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;
```

ANALYTICS.HR ▾ Settings ▾

```

1  use role SYSADMIN;
2
3  grant usage on database analytics to role DATA_ANALYST_ROLE_US;
4  grant usage on schema analytics.hr to role DATA_ANALYST_ROLE_US;
5  grant select on all tables in schema analytics.hr to role DATA_ANALYST_ROLE_US;
6
7  grant usage on database analytics to role DATA_ANALYST_ROLE_UK;
8  grant usage on schema analytics.hr to role DATA_ANALYST_ROLE_UK;
9  grant select on all tables in schema analytics.hr to role DATA_ANALYST_ROLE_UK;
10
11 grant usage on database analytics to role DATA_ANALYST_ROLE_CA;
12 grant usage on schema analytics.hr to role DATA_ANALYST_ROLE_CA;
13 grant select on all tables in schema analytics.hr to role DATA_ANALYST_ROLE_CA;

```

↩ Results Chart

	status
1	Statement executed successfully. 2 objects affected.

## Grant USAGE privilege on virtual warehouse to custom roles

19. The below SQL statements provides usage privileges on warehouse compute\_wh to the custom roles to query tables.

Query: use role ACCOUNTADMIN;

```

grant usage on warehouse compute_wh to role DATA_ANALYST_ROLE_US;
grant usage on warehouse compute_wh to role DATA_ANALYST_ROLE_UK;
grant usage on warehouse compute_wh to role DATA_ANALYST_ROLE_CA;

```

ANALYTICS.HR ▾ Settings ▾

```

1  use role ACCOUNTADMIN;
2
3  grant usage on warehouse compute_wh to role DATA_ANALYST_ROLE_US;
4  grant usage on warehouse compute_wh to role DATA_ANALYST_ROLE_UK;
5  grant usage on warehouse compute_wh to role DATA_ANALYST_ROLE_CA;
6

```

↩ Results Chart

	status
	Statement executed successfully.

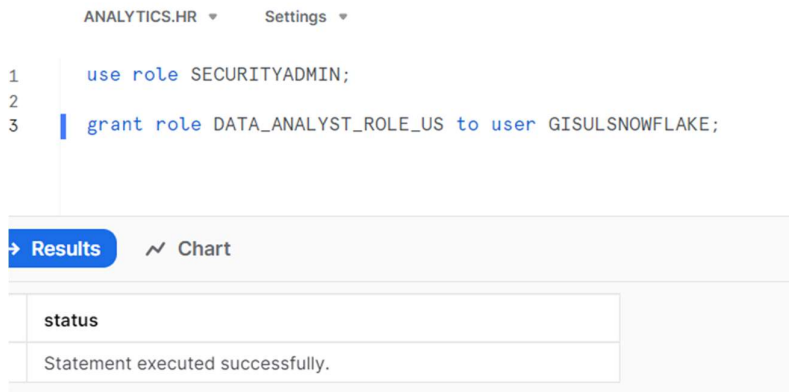
## Assign Custom Roles to Users

20. In my case I have only one user so, I am only assigning only DATA\_ANALYST\_ROLE\_US to my use.

21. If you have multiple users you can assign for other roles also.

Query: use role SECURITYADMIN;

grant role DATA\_ANALYST\_ROLE\_US to user GISULSNOWFLAKE;



The screenshot shows a SQL query editor with the following code:

```
1 use role SECURITYADMIN;
2
3 grant role DATA_ANALYST_ROLE_US to user GISULSNOWFLAKE;
```

Below the query editor, there is a 'Results' tab and a 'Chart' tab. The 'Results' tab is active, showing a table with one row:

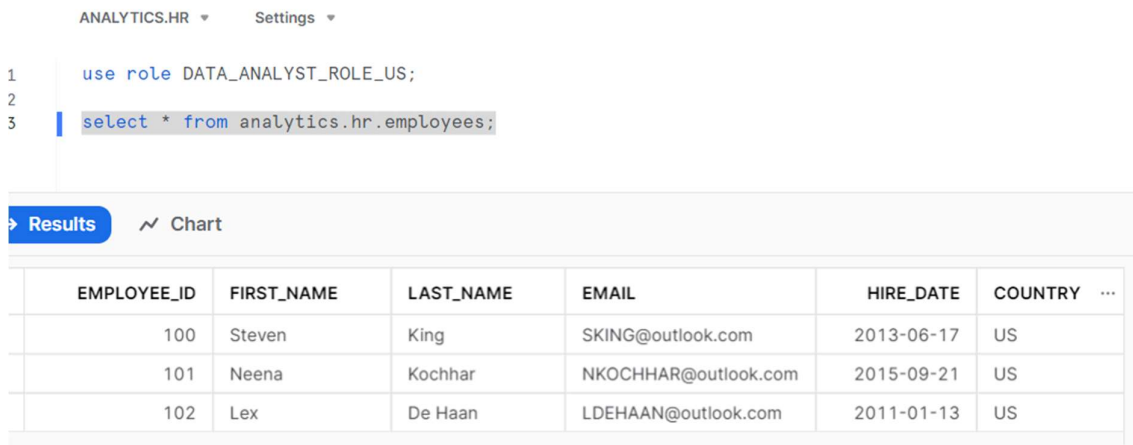
status
Statement executed successfully.

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

23. 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.

Query: Use role DATA\_ANALYST\_ROLE\_US;

Select \* from analytics.hr.employees;



The screenshot shows a SQL query editor with the following code:

```
1 use role DATA_ANALYST_ROLE_US;
2
3 select * from analytics.hr.employees;
```

Below the query editor, there is a 'Results' tab and a 'Chart' tab. The 'Results' tab is active, showing a table with the following data:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	COUNTRY	...
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	

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

Query: User role SYSADMIN;

Select \* from analytics.hr.employees;

ANALYTICS.HR ▾ Settings ▾

1 use role SYSADMIN;

2

3 select \* from analytics.hr.employees;

→ Results Chart

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	...	HIRE_DATE	COUNTRY
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

25. The below SQL statement revokes all privileges on table role\_mapping to the custom roles.

Query: use role SYSADMIN;

revoke all privileges on table analytics.hr.role\_mapping from role DATA\_ANALYST\_ROLE\_US;

ANALYTICS.HR ▾ Settings ▾

1 use role SYSADMIN;

2

3 revoke all privileges on table analytics.hr.role\_mapping from role DATA\_ANALYST\_ROLE\_US;

↶ Results Chart

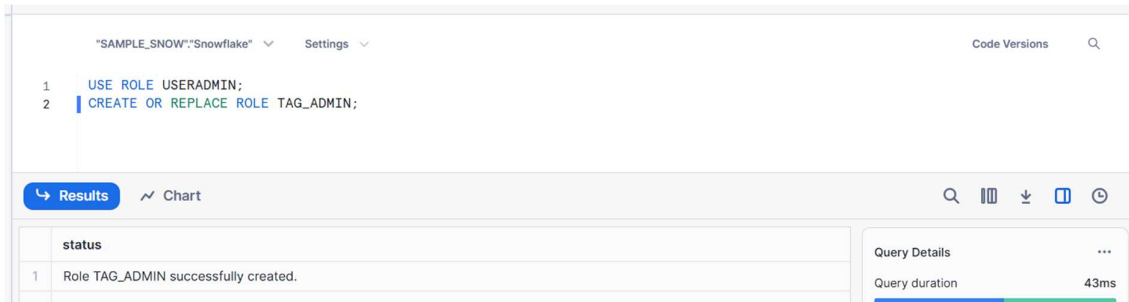
status
1 Statement executed successfully.

# Tagging

1. Create a Custom Role
2. The below SQL statement creates a custom role TAG\_ADMIN in Snowflake.

Query: USE ROLE USERADMIN;

CREATE OR REPLACE ROLE TAG\_ADMIN;



3. Assign Tagging Privileges to custom role
4. The below SQL statement grants privileges to create tags to the role TAG\_ADMIN.

Query: USE ROLE ACCOUNTADMIN;

GRANT CREATE TAG ON SCHEMA "SAMPLE\_SNOW"."Snowflake" TO ROLE TAG\_ADMIN;



7. The below SQL statement grants privileges to apply tags on Snowflake objects to the role TAG\_ADMIN.

Query: GRANT APPLY TAG ON ACCOUNT TO ROLE TAG\_ADMIN;



8. The below example shows creating a tag named PII\_DATA with Names, Contact Details and Email as allowed values.

Query: CREATE OR REPLACE TAG PII\_DATA ALLOWED\_VALUES 'Names', 'Contact Details','Email';

The screenshot shows a Snowflake query editor interface. At the top, there's a header with "SAMPLE\_SNOW"."Snowflake" and "Settings". Below the header, a query is entered: `CREATE OR REPLACE TAG PII_DATA ALLOWED_VALUES 'Names', 'Contact Details','Email';`. The results pane at the bottom shows a single row with the status "Tag PII\_DATA successfully created." and a "Query Details" sidebar on the right.

9. To set a tag on an existing column, use the ALTER TABLE ... MODIFY COLUMN command for a table column
10. The below example assigns the tag PII\_DATA on multiple columns with different tag values on the table named Employee\_Details.

Query: ALTER TABLE EMPLOYEE\_DETAILS MODIFY COLUMN NAME SET TAG PII\_DATA = 'Names';  
ALTER TABLE EMPLOYEE\_DETAILS MODIFY COLUMN CONTACT\_NUMBER SET TAG PII\_DATA = 'Contact Details';  
ALTER TABLE EMPLOYEE\_DETAILS MODIFY COLUMN EMAIL\_ID SET TAG PII\_DATA = 'Email';

The screenshot shows a Snowflake query editor interface. At the top, there's a header with "SAMPLE\_SNOW"."Snowflake" and "Settings". Below the header, three queries are entered: `ALTER TABLE EMPLOYEE_DETAILS MODIFY COLUMN NAME SET TAG PII_DATA = 'Names';`, `ALTER TABLE EMPLOYEE_DETAILS MODIFY COLUMN CONTACT_NUMBER SET TAG PII_DATA = 'Contact Details';`, and `ALTER TABLE EMPLOYEE_DETAILS MODIFY COLUMN EMAIL_ID SET TAG PII_DATA = 'Email';`. The results pane at the bottom shows a single row with the status "Statement executed successfully." and a "Query Details" sidebar on the right showing a duration of 49ms.

11. The below example helps in identifying all the tags assigned on the columns of the table EMPLOYEE\_DETAILS.

Query: SELECT \* FROM TABLE( INFORMATION\_SCHEMA.TAG\_REFERENCES\_ALL\_COLUMNS('SAMPLE\_SNOW"."Snowflake". Employee\_Details', 'table'));

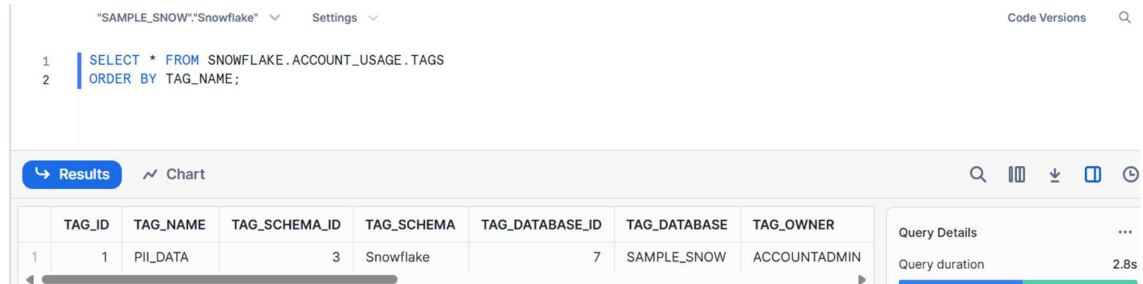
The screenshot shows a Snowflake query editor interface. At the top, there's a header with "SAMPLE\_SNOW"."Snowflake" and "Settings". Below the header, a query is entered: `SELECT * FROM TABLE( INFORMATION_SCHEMA.TAG_REFERENCES_ALL_COLUMNS('SAMPLE_SNOW"."Snowflake".Employee_Details', 'table'));`. The results pane at the bottom shows a table with 3 rows and 7 columns: TAG\_DATABASE, TAG\_SCHEMA, TAG\_NAME, TAG\_VALUE, LEVEL, OBJECT\_DATABASE, and OBJECT\_SCHEMA. The results are as follows:

	TAG_DATABASE	TAG_SCHEMA	TAG_NAME	TAG_VALUE	LEVEL	OBJECT_DATABASE	OBJECT_SCHEMA
1	SAMPLE_SNOW	Snowflake	PII_DATA	Contact Details	COLUMN	SAMPLE_SNOW	Snowflake
2	SAMPLE_SNOW	Snowflake	PII_DATA	Email	COLUMN	SAMPLE_SNOW	Snowflake
3	SAMPLE_SNOW	Snowflake	PII_DATA	Names	COLUMN	SAMPLE_SNOW	Snowflake

The "Query Details" sidebar on the right shows a duration of 685ms and 3 rows.

12. The TAGS view in the Account Usage schema of Snowflake database provides information of all the tags in your Snowflake account including the deleted tags.

Query: `SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.TAGS`  
`ORDER BY TAG_NAME;`

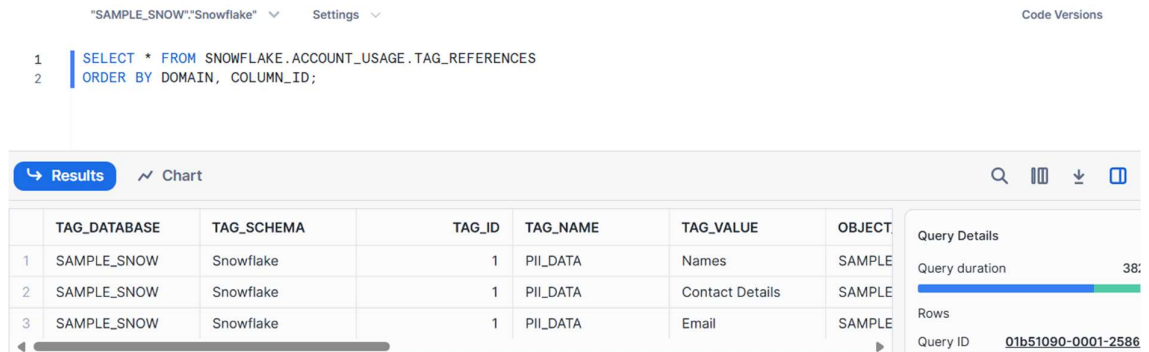


The screenshot shows the Snowflake query editor with the query `SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.TAGS ORDER BY TAG_NAME;` and its results. The results table has 7 columns: TAG\_ID, TAG\_NAME, TAG\_SCHEMA\_ID, TAG\_SCHEMA, TAG\_DATABASE\_ID, TAG\_DATABASE, and TAG\_OWNER. The first row shows TAG\_ID 1, TAG\_NAME PII\_DATA, TAG\_SCHEMA\_ID 3, TAG\_SCHEMA Snowflake, TAG\_DATABASE\_ID 7, TAG\_DATABASE SAMPLE\_SNOW, and TAG\_OWNER ACCOUNTADMIN. The query duration is 2.8s.

	TAG_ID	TAG_NAME	TAG_SCHEMA_ID	TAG_SCHEMA	TAG_DATABASE_ID	TAG_DATABASE	TAG_OWNER
1	1	PII_DATA	3	Snowflake	7	SAMPLE_SNOW	ACCOUNTADMIN

13. The TAG\_REFERENCES view in the Account Usage schema of Snowflake database provides information of all the objects in your Snowflake account that are assigned with a tag and a tag value.

Query: `SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.TAG_REFERENCES`  
`ORDER BY DOMAIN, COLUMN_ID;`



The screenshot shows the Snowflake query editor with the query `SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.TAG_REFERENCES ORDER BY DOMAIN, COLUMN_ID;` and its results. The results table has 7 columns: TAG\_DATABASE, TAG\_SCHEMA, TAG\_ID, TAG\_NAME, TAG\_VALUE, and OBJECT. The first three rows show TAG\_ID 1, TAG\_NAME PII\_DATA, and TAG\_VALUE Names, Contact Details, and Email, all under TAG\_DATABASE SAMPLE\_SNOW and TAG\_SCHEMA Snowflake. The query duration is 38s and the query ID is 01b51090-0001-2588.

	TAG_DATABASE	TAG_SCHEMA	TAG_ID	TAG_NAME	TAG_VALUE	OBJECT
1	SAMPLE_SNOW	Snowflake	1	PII_DATA	Names	SAMPLE
2	SAMPLE_SNOW	Snowflake	1	PII_DATA	Contact Details	SAMPLE
3	SAMPLE_SNOW	Snowflake	1	PII_DATA	Email	SAMPLE

14. The above output shows all the tags assigned at DATABASE, SCHEMA, TABLE and COLUMN levels at the account level.