**Use case 1:** State few issues with shared disk and shared nothing architectures data warehouses or Databases.

**Solution:**

Few issues with shared disk and shared nothing architectures for data warehouses or databases in Snowflake architecture:

* Shared disk architecture is generally easier to set up and manage than shared nothing architecture. However,
  + It can be less scalable and performant, especially for large datasets.
  + If a node in a shared disk cluster fails, the entire cluster can be unavailable until the node is repaired or replaced. This can lead to downtime and data loss.
* Shared nothing architecture is generally more scalable and performant than shared disk architecture. However,
  + It can be more complex to set up and manage.
  + It architecture requires more hardware than a shared disk architecture. This can make it more expensive to set up and maintain.

The best architecture for a particular application will depend on the specific requirements of that application.

**Use case 2:** On which clouds , Snowflake can be hosted

**Solution:**

Snowflake can be hosted on three different clouds: Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

**Use case 3:** What are the different layers in Snowflake architecture and use of each layer?

**Solution:**

The different layers in Snowflake architecture and the use of each layer:

* The outermost layer of Snowflake's architecture is the **Services Layer.** This layer provides all the services that are needed to run Snowflake, such as authentication, authorization, metadata management, and infrastructure management.
* The middle layer of Snowflake's architecture is the Multi-cluster **Compute Layer**: This layer is responsible for processing all the queries that are submitted to Snowflake.
* The innermost layer of Snowflake's architecture is the **Storage Layer**: This layer stores all of the data in Snowflake. The Storage Layer is a distributed file system that is optimized for performance and scalability.

**Use case 4:** What the different types of Snowflake editions.

**Solution:**

Different Snowflake Editions:

* Standard Edition: The most basic edition of Snowflake, with limited features and functionality.
* Enterprise Edition: A more powerful edition of Snowflake, with additional features and functionality, such as multi-cluster warehouses, time travel, and dynamic data masking.
* Business Critical Edition: The most powerful edition of Snowflake, with all of the features and functionality of the Enterprise Edition, plus additional features, such as encryption key rotation, materialized views, and search optimization services.
* Virtual Private Snowflake (VPS): A dedicated Snowflake instance that is isolated from other Snowflake instances.

Snowflake offers multiple editions to choose from, ensuring that your usage fits your organization’s specific requirements.

**Use case 5:** What are the different types of tables in Snowflake and what is difference between them?

**Solution:**

The different types of tables in Snowflake and the difference between them:

* **Permanent Tables**:
  + Permanent tables are the default table type in Snowflake.
  + They are stored in the Snowflake database and are accessible to all users.
  + Permanent tables are persisted, which means that they are not deleted when the session ends.
* **Transient Tables**:
  + Transient tables are temporary tables that are only stored in memory.
  + Transitory data is maintained beyond each session.
  + Exists until explicitly dropped.
  + Available to all users with appropriate privileges.
  + Can be cloned as transient or temporary tables.
* **Temporary Tables**:
  + Only exists in the session it is created in and not visible to other sessions/users.
  + Once the session ends, data cannot be retrieved by anyone.
  + time-travel feature. (Time Travel Retention Period - 0/1 default-1 day)

**Use case 6:** Create additional partitioning on table to improve query performance.

**Solution:**

Snowflake can be hosted on three different clouds: Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

**Use case 7:** What is difference between Economy and standard scaling policy for query processing engine?

**Solution:**

The Economy scaling policy in Snowflake's query processing engine provides a low-cost option for running queries with minimal resources. “Economy” only adds a cluster if there's enough query load to keep the new cluster busy for 6 minutes. This means that the Economy scaling policy may cause some queries to queue, but it will help to keep costs down.

The Standard scaling policy provides a balanced approach to resource allocation for more demanding workloads. It will add a cluster as soon as there is a query that is waiting to be executed. This means that the Standard scaling policy may cause costs to increase, but it will help to ensure that queries are executed as quickly as possible.

**Use case 8:** Create a multi-cluster and cost effective warehouse with auto suspend after 3 minutes by using SQL queries.

**Solution:**

CREATE OR REPLACE WAREHOUSE USECASE\_D1 WITH WAREHOUSE\_SIZE = 'XSMALL'

WAREHOUSE\_TYPE= 'STANDARD' --STANDARD | SNOWPARK-OPTIMIZED

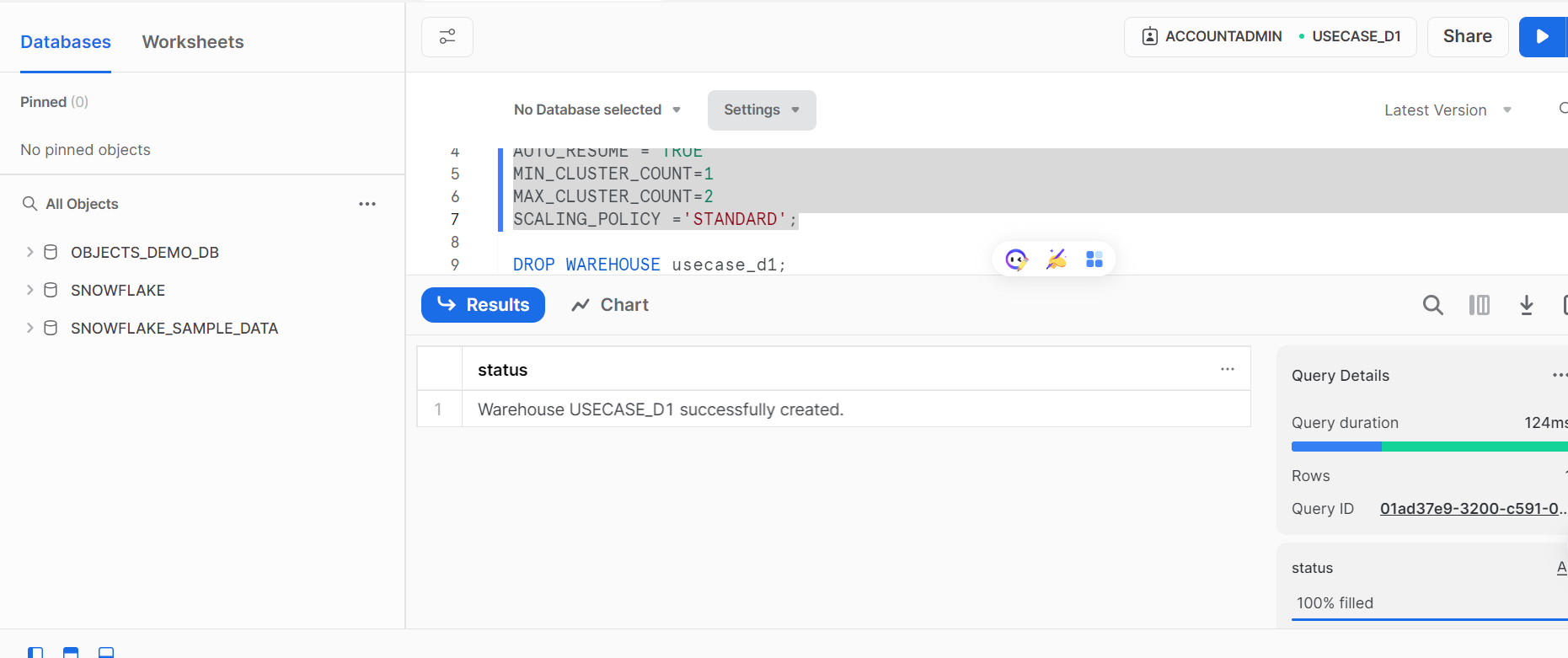
AUTO\_SUSPEND= 180

AUTO\_RESUME = TRUE

MIN\_CLUSTER\_COUNT=1

MAX\_CLUSTER\_COUNT=2

SCALING\_POLICY ='STANDARD';



**Use case 9:** Create two medium size virtual warehouses, one with maximize mode and another with auto-scale mode.

**Solution:**

CREATE OR REPLACE WAREHOUSE MAXIMIZED\_WH

WITH WAREHOUSE\_SIZE = 'XSMALL'

WAREHOUSE\_TYPE= 'STANDARD' --STANDARD | SNOWPARK-OPTIMIZED

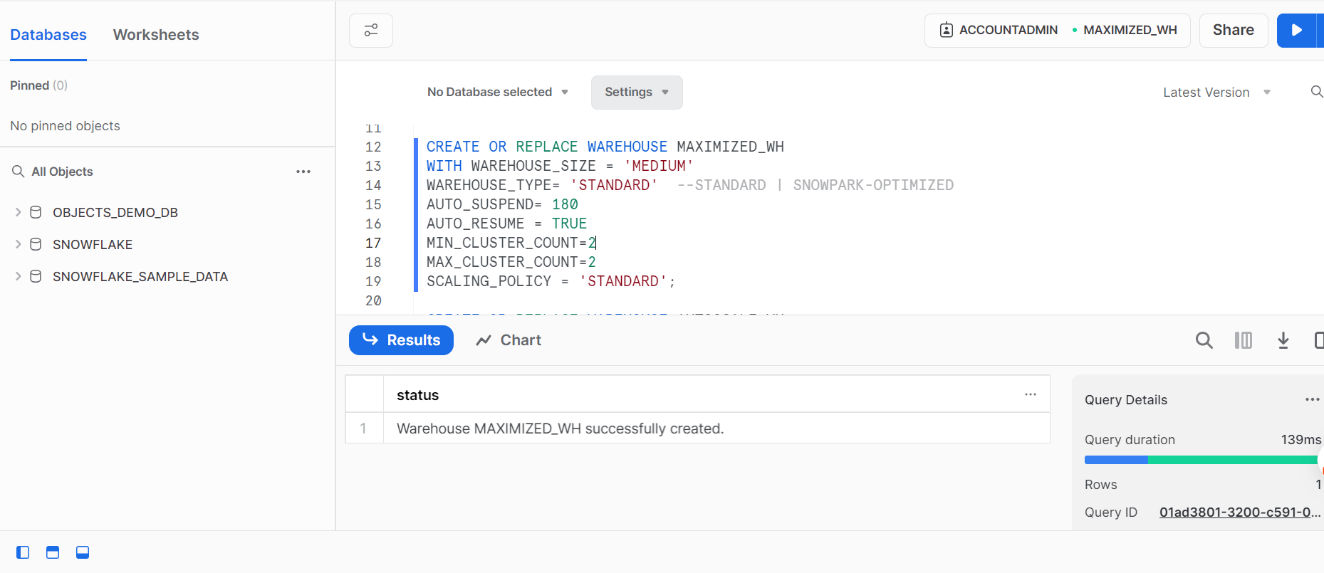
AUTO\_SUSPEND= 180

AUTO\_RESUME = TRUE

MIN\_CLUSTER\_COUNT=2

MAX\_CLUSTER\_COUNT=2

SCALING\_POLICY = 'STANDARD';



CREATE OR REPLACE WAREHOUSE AUTOSCALE\_WH

WITH WAREHOUSE\_SIZE = 'XSMALL'

WAREHOUSE\_TYPE= 'STANDARD' --STANDARD | SNOWPARK-OPTIMIZED

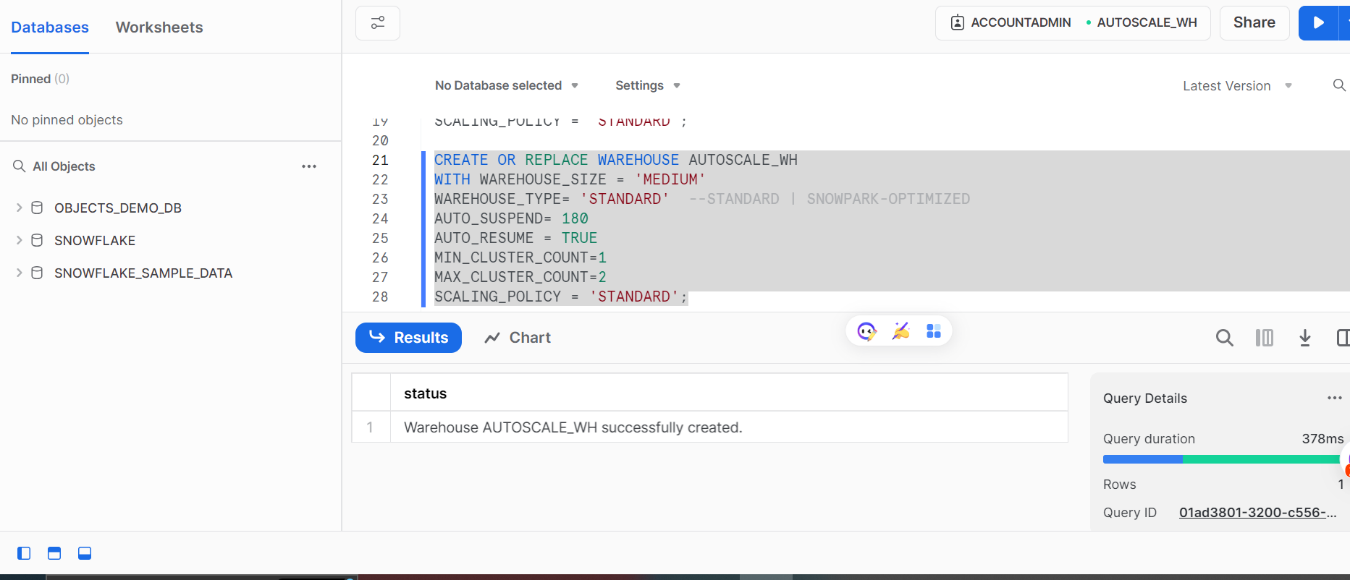
AUTO\_SUSPEND= 180

AUTO\_RESUME = TRUE

MIN\_CLUSTER\_COUNT=1

MAX\_CLUSTER\_COUNT=2

SCALING\_POLICY = 'STANDARD';



**Use case 10:** What is mean by vertical scaling and horizontal scaling. In which cases, we should go with vertical scaling and horizontal scaling?

**Solution:**

**Vertical scaling**, also known as scaling up, refers to adding more resources to a single server, such as increasing the CPU, RAM, or storage capacity. This is a good option if you need to increase the capacity of a single server without adding more servers. However, vertical scaling can be expensive, and it can only increase the capacity of a single server up to a certain point.

**Horizontal scaling**, also known as scaling out, involves adding more servers to distribute the workload. This is a good option if you need to increase the capacity of a system beyond what a single server can handle. Horizontal scaling is more cost-effective than vertical scaling, and it can scale a system to much larger sizes.

**Use case 11:** Explain types of caches in Snowflake (in brief only) and what is the life of each type of cache.

**Solution:**

Snowflake has three types of caches: Result Cache, Metadata Cache, and Query Cache.

* **Result Cache** stores the results of previously executed query. Result Cache lasts until the result is no longer needed.
* **Metadata Cache** stores metadata about tables, views, and other objects in the database. Metadata Cache lasts until the object is altered or dropped.
* **Local Disk Cache:**  Which is used to cache data used by SQL queries.  Whenever data is needed for a given query it's retrieved from the Remote Disk storage and cached in SSD and memory.

The life of each cache type varies depending on the type of cache.

**Use case 12:** How can we secure/protect our data WITH MANAGED ACCESS schema option. Create a scenario to implement it.

**Scenario:**

You are a developer working for a company that stores sensitive customer data in Snowflake. You need to ensure that the data is secure and that only authorized users can access it.

**Solution:**

use role USERADMIN;

create or replace role developer;

create or replace user alex;

drop user analyst;

use role sysadmin;

CREATE OR REPLACE WAREHOUSE USECASE\_D3 WITH WAREHOUSE\_SIZE = 'XSMALL'

WAREHOUSE\_TYPE= 'STANDARD' --STANDARD | SNOWPARK-OPTIMIZED

AUTO\_SUSPEND= 600

AUTO\_RESUME = TRUE

MIN\_CLUSTER\_COUNT=1

MAX\_CLUSTER\_COUNT=2

SCALING\_POLICY ='STANDARD';

show warehouses;

create or replace database db\_sales;

create or replace schema employees;

create or replace table emp\_info(name varchar,age int);

--Granting Privileges to roles

Use role securityadmin; ---- manage and grants priviliges

grant usage on database db\_sales to role developer;

//grant usage on database demo\_db to user alex;//throws error

grant usage on all schemas in database db\_sales to role developer;

grant select on all tables in database db\_sales to role developer; //only read access granted here

grant usage,operate on warehouse USECASE\_D3 to role developer;

grant role developer to user alex;

grant role developer to user Tusharr08;

use role analyst;

--now alex has access to database db\_sales

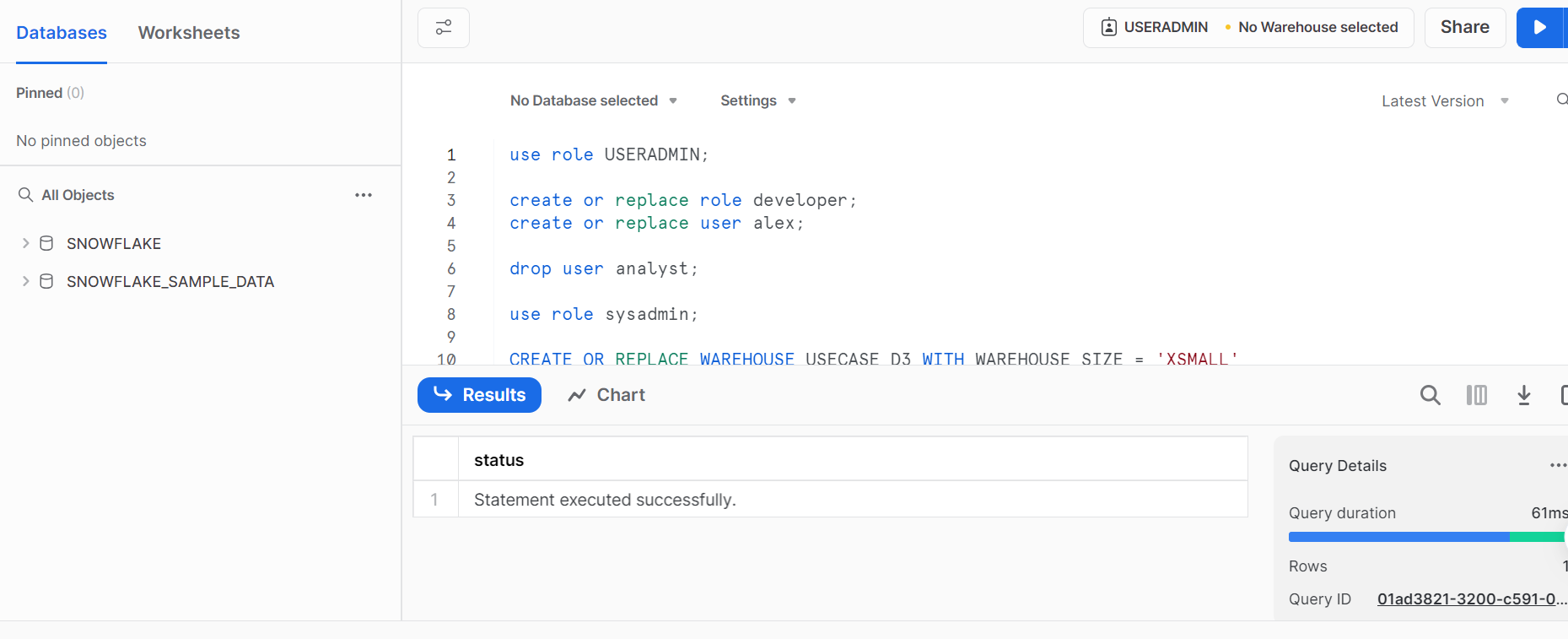
--Assigning default objects to the role Alex

use role useradmin;

Alter user ALEX set default\_Role = 'Public' default\_warehouse='USECASE\_D3' default\_namespace='db\_sales.employees';

-- DROP USER ALEX;

-- DROP ROLE ANALYST



**Use case 13:** What is difference between standard view and secured view. Explain with examples.

**Solution:**

**Standard views** are the most common type of view in Snowflake. They are created using the CREATE VIEW statement, and they can be used to simplify or summarize data from one or more tables. Created by executing the query at the time that the view is referenced in a query. The results are not stored for future use. Performance is slower than with materialized views.

e.g.:

CREATE or replace VIEW OBJECTS\_DEMO\_DB.OBJECTS\_DEMO\_SCHEMA.Fin\_view

COMMENT = 'original finance team as a view' AS select name, age from table\_demo where dept='Fin';

select \* from Fin\_view

**Secure views**: Both non-materialized and materialized views can be defined as *secure*. Secure views have advantages over standard views, including improved data privacy and data sharing. They also have some performance impacts to take into consideration.

e.g.: --Secure View Syntax:

create or replace secure view age\_sview

comment = 'Test view for age'

as select name, dept from table\_demo where age>30;

select \* from age\_sview;

**Use case 14:** What is difference between standard view and Materialized view. Explain with examples.

**Solution:**

The difference between standard views and materialized views in Snowflake:

**Standard views** are virtual tables that are created using the CREATE VIEW statement. They are not stored physically on disk, and they are not updated automatically when the underlying tables are updated.

**Materialized views are** physical tables that are created from the results of a SQL query. When you query a materialized view, the results of the query are stored in the database and are available even if the view is dropped or the underlying tables are modified.

Here is an example of a standard view:

CREATE VIEW customers\_summary AS

SELECT customer\_id, customer\_name, COUNT(\*) AS total\_orders

FROM orders

GROUP BY customer\_id, customer\_name;

This view would create a view called customers\_summary that summarizes the customer data in the orders table. The results of this view **would not be stored** in the database, so they would not be available if the view was dropped or the underlying tables were modified.

Here is an example of a materialized view:

CREATE MATERIALIZED VIEW customers\_summary AS

SELECT customer\_id, customer\_name, COUNT(\*) AS total\_orders

FROM orders

GROUP BY customer\_id, customer\_name;

This view would create a materialized view called customers\_summary that summarizes the customer data in the orders table. The results of this view **would be stored** in the database and would be available even if the view was dropped or the underlying tables were modified.

**Use case 15:** What is responsibility (at high level) of each system role in Snowflake?

**Solution:**

1. ACCOUNTADMIN: (aka ACCOUNT ADMINISTRATOR)
   1. Role that encapsulates the SYSADMIN, SECURITYADMIN system defined roles.
   2. It is the top-level role in the system and should be granted only to a limited/controlled no of users in account.
2. SECURITYADMIN: (aka SECURITY ADMINISTRATOR)
   1. Role that can manage any object grant globally.
   2. As well as CREATE, MONITOR and MANAGE **USERS and ROLES.**
   3. This role is granted the **MANAGE GRANTS** security privileges to be able to modify any grant, including revoking it.
   4. It inherits privileges of USERADMIN role via system role hierarchy.
3. USERADMIN: (aka USER ADMINISTRATOR)
   1. Role that is dedicated to **USER and ROLE management only.**
   2. This role is granted the CREATE USERS and CREATE ROLE security privileges.
   3. This role also manages users and roles that it owns. Only the role with ownership privilege on an object or higher role can modify object properties.
4. SYSADMIN: (aka SYSTEM ADMINISTRATOR)
   1. Role that has privileges to create WAREHOUSES AND DATABASE OBJECTS in an account.
   2. If as, recommended you create a role hierarchy that ultimately assigns all custom roles to SYSADMIN, this role has ability to grant privileges on warehouses and other database objects.

**Use case 16:** Create users and roles - at least 2 users and functional roles by using SQL queries then assign the roles to users. Also , create roles for read-only and read-write and assign these roles to functional roles and test it by accessing objects by using these roles.

**Solution:**

use role USERADMIN;

create or replace role manager;

create or replace role analyst;

create or replace user tom;

create or replace user henry;

create or replace role read\_only; **//only read access**

create or replace role read\_write; **//both read and write data**

-- use role sysadmin;

-- CREATE OR REPLACE WAREHOUSE USECASE\_D3 WITH WAREHOUSE\_SIZE = 'XSMALL'

-- WAREHOUSE\_TYPE= 'STANDARD' **--STANDARD | SNOWPARK-OPTIMIZED**

-- AUTO\_SUSPEND= 600

-- AUTO\_RESUME = TRUE

-- MIN\_CLUSTER\_COUNT=1

-- MAX\_CLUSTER\_COUNT=2

-- SCALING\_POLICY ='STANDARD';

-- show warehouses;

-- create or replace database db\_sales;

-- create or replace schema employees;

-- create or replace table emp\_info(name varchar,age int);

**--Granting Privileges to roles**

Use role securityadmin; ---- manage and grants priviliges

**//grants for read-write**

grant usage on database db\_sales to role read\_write;

grant usage on all schemas in database db\_sales to role read\_write;

grant usage,operate on warehouse USECASE\_D3 to role read\_write;

**//grants for read-only**

grant usage on database db\_sales to role read\_only;

grant usage on all schemas in database db\_sales to role read\_only;

grant select on all tables in database db\_sales to role read\_only;

grant role read\_write to role manager;

grant role read\_only to role analyst;

grant role manager to user tom;

grant role analyst to user henry;

use role analyst;

**--Assigning default objects to the role Alex**

use role useradmin;

Alter user tom set default\_Role = 'Public' default\_warehouse='USECASE\_D3' default\_namespace='db\_sales.employees';

**A screenshot of a computer

Description automatically generated with medium confidence**

**Use case 17:** Create a scenario to apply future grants.

**Scenario:**

You are a developer and you need to create a new table to store data from a new data source. You want to grant the SELECT privilege on the new table to a group of users who will be using the data for analysis.

You can use the GRANT SELECT ON FUTURE TABLES IN SCHEMA statement to grant the SELECT privilege on all future tables in the schema. This will allow the users to access the new table as soon as it is created.

use role sysadmin;

**//creating a new warehouse**

create or replace WAREHOUSE USECASE\_D3 WITH WAREHOUSE\_SIZE = 'XSMALL'

WAREHOUSE\_TYPE = 'STANDARD' AUTO\_SUSPEND = 3600 AUTO\_RESUME = TRUE MIN\_CLUSTER\_COUNT = 1 MAX\_CLUSTER\_COUNT = 2 SCALING\_POLICY = 'STANDARD';

**//database: db\_sales**

**//schema : employees**

**//table: emp\_info**

insert into emp\_info values ('Alex',25), ('Angela',32), ('Oscar',31),('Toby',36),('Jim',31),('Kevin',30)

select \* from emp\_info

use role securityadmin;

grant usage on database db\_sales to role developer;

grant usage on all schemas in database db\_sales to role developer;

grant select on all tables in database db\_sales to role developer;

**//Grant SELECT privilege on all future tables for a particular schema:**

grant select on future tables in schema db\_sales.employees to role developer;

**//Grant SELECT privilege on all future tables for a particular database:**

grant select on future tables in database db\_sales to role developer;

**Use case 18:** Apply data masking policy on sensitive data in your table

**Solution:**

use role sysadmin;

create or replace database confidential;

create or replace schema security;

USE ROLE SECURITYADMIN;

--claims.pharmacy(namespace=database.schema)

GRANT CREATE MASKING POLICY ON SCHEMA confidential.security TO ROLE SYSADMIN;

create or replace role admin;

GRANT ROLE admin TO ROLE SYSADMIN;

create or replace user agent password='agent123' must\_change\_password = false;

--drop user developer

--create or replace user developer password='abc123' default\_secondary\_roles = ('ALL') must\_change\_password = false;

grant role admin to user agent;

-- Grant usage access on warehouse & database.

USE ROLE SYSADMIN;

GRANT USAGE ON WAREHOUSE USECASE\_D3 TO ROLE ADMIN;

GRANT OPERATE ON WAREHOUSE USECASE\_D3 TO ROLE ADMIN;

GRANT USAGE ON DATABASE CONFIDENTIAL TO ROLE ADMIN;

GRANT USAGE ON SCHEMA CONFIDENTIAL.SECURITY TO ROLE ADMIN;

--Create masking policy.

USE ROLE SYSADMIN;

CREATE OR REPLACE MASKING POLICY CONFIDENTIAL.SECURITY.AGENT\_MASK AS (VAL STRING) RETURNS STRING ->

CASE

WHEN CURRENT\_ROLE()NOT IN ('SYSADMIN') THEN

CASE WHEN SUBSTRING(VAL, 1, 4) = '008-' THEN CONCAT('\*\*\*\*\*\*\*\*', SUBSTRING(VAL, 7)) ELSE VAL

END

ELSE VAL

END;

CREATE or replace TABLE AGENTS

(

AGENT\_CODE varchar NOT NULL PRIMARY KEY,

AGENT\_NAME varchar,

WORKING\_AREA CHAR(35),

COMMISSION NUMBER(10,2),

PHONE\_NO varchar MASKING POLICY CONFIDENTIAL.SECURITY.AGENT\_MASK,--applying masking policy,

COUNTRY varchar

);

INSERT INTO AGENTS VALUES ('A007', 'Ramasundar', 'Bangalore', '0.15', '077-25814763', '');

INSERT INTO AGENTS VALUES ('A003', 'Alex ', 'London', '0.13', '075-12458969', '');

INSERT INTO AGENTS VALUES ('A008', 'Alford', 'New York', '0.12', '008-25874365', '');

INSERT INTO AGENTS VALUES ('A011', 'Ravi Kumar', 'Bangalore', '0.15', '077-45625874', '');

INSERT INTO AGENTS VALUES ('A010', 'Santakumar', 'Chennai', '0.14', '007-22388644', '');

INSERT INTO AGENTS VALUES ('A012', 'Lucida', 'San Jose', '0.12', '044-52981425', '');

INSERT INTO AGENTS VALUES ('A005', 'Anderson', 'Brisban', '0.13', '045-21447739', '');

INSERT INTO AGENTS VALUES ('A001', 'Subbarao', 'Bangalore', '0.14', '077-12346674', '');

INSERT INTO AGENTS VALUES ('A002', 'Mukesh', 'Mumbai', '0.11', '029-12358964', '');

INSERT INTO AGENTS VALUES ('A006', 'McDen', 'London', '0.15', '078-22255588', '');

INSERT INTO AGENTS VALUES ('A004', 'Ivan', 'Torento', '0.15', '008-22544166', '');

INSERT INTO AGENTS VALUES ('A009', 'Benjamin', 'Hampshair', '0.11', '008-22536178', '');

select \* from agents;

GRANT SELECT ON confidential.security.agents TO ROLE admin

show masking policies;

**A screenshot of a computer

Description automatically generated**

**Use case 19:** Create a row level data security on your table data.

**Solution:**

create or replace row access policy CONFIDENTIAL.SECURITY.ROW\_AGENT\_MASK as (working\_area varchar) returns boolean ->

CASE

WHEN CURRENT\_ROLE()NOT IN ('SYSADMIN') THEN

CASE **WHEN working\_area='Bangalore' THEN FALSE** ELSE TRUE

END

ELSE TRUE

END;

CREATE or replace TABLE AGENTS

(

AGENT\_CODE varchar NOT NULL PRIMARY KEY,

AGENT\_NAME varchar,

WORKING\_AREA CHAR(35),

COMMISSION NUMBER(10,2),

PHONE\_NO varchar MASKING POLICY CONFIDENTIAL.SECURITY.AGENT\_MASK,--applying masking policy,

COUNTRY varchar

);

alter table confidential.security.agents add row access policy ROW\_AGENT\_MASK on (working\_area);

**After logging in through the agent user, all the entries with Bangalore as working\_area is cleared.**

A screenshot of a computer

Description automatically generated with medium confidence

**Use case 20:** Can we apply both row level and data masking both at a time on table?

**Solution:**

**Yes,** as you can see in the above example both the masking policies row-level as well as column-level policy is implemented. After logging in through the agent user, not only the Bangalore as working\_area rows is visible but also the phone\_no starting with ‘080-‘ are encrypted.

**Use case 21:** Can we apply more than one row level access policy at a time on a table?

**Solution:**

Yes, you can apply more than one row level access policy at a time on a table. When you apply multiple row level access policies to a table, the policies are evaluated in the order in which they were created. The first policy that evaluates to TRUE determines whether the user is allowed to see the row. If the first policy evaluates to FALSE, then the second policy is evaluated, and so on.

For example, let's say we have a table named agents with two row level access policies:

* policy1: Allows users in the sales role to see all rows in the table.
* policy2: Masks the agent\_name column for all rows in the table.

If a user who is in the sales role tries to query the agents table, the policy1 row level access policy will be evaluated first. If the user is in the sales role, then the policy1 row level access policy will evaluate to TRUE, and the user will be allowed to see all rows in the table. The policy2 row level access policy will not be evaluated in this case.

However, if a user who is not in the sales role tries to query the agents table, the policy1 row level access policy will evaluate to FALSE. In this case, the policy2 row level access policy will be evaluated, and the agent\_name column will be masked for the user.

**Use case 22:** On what snowflake objects we can apply data masking and row level access policy?

**Solution:**

Data masking and row level access policies can be applied to the following Snowflake objects:

* **Tables:** You can apply masking and row level access policies to individual columns in a table. This allows you to control who can see what data in the table, and how the data is displayed.
* **Views:** You can also apply masking and row level access policies to views. This is useful if you want to apply the same masking or row level access policy to multiple tables.
* **Secure views:** Secure views are a special type of view that allows you to control access to the underlying tables based on the user's role. You can apply masking and row level access policies to secure views, but you cannot apply them to the underlying tables.
* **Virtual warehouses:** You can apply masking and row level access policies to virtual warehouses. This allows you to control who can access the data in the warehouse, and how the data is displayed.

It is important to note that you cannot apply masking and row level access policies to individual rows in a table. You can only apply them to columns or views.

**Use case 23:** limitations of data masking and row level access policies?

**Solution:**

**Row Level Access Policy:**

* Using the CHANGES clause on a view protected by a row access policy is not supported.
* Snowflake does not support using external tables as a mapping table in a row access policy.
* Snowflake does not support attaching a row access policy to the stream object itself, but does apply the row access policy to the table when the stream accesses a table protected by a row access policy.
* Future grants of privileges on row access policies are not supported.

**Data Masking:**

* It can be time-consuming and complex to implement.
* It can be difficult to maintain, as you need to update the masks whenever the data changes.
* It can be difficult to test, as you need to make sure that the masks do not interfere with the functionality of your applications.

**Use case 24:** Explain the Differences between user stage, table stage and name stage

**Solution:**

1. User Stage:

* User-specific storage location for temporary file storage.
* Used for data ingestion and export tasks.
* Associated with a specific user and has a limited lifespan.
* Accessible only by the user who owns it.

1. Table Stage:

* Dedicated storage location associated with a specific table.
* Used for efficient data loading and unloading operations.
* Allows for parallel data ingestion or export from multiple files.
* Managed by Snowflake and not directly accessible for general file storage.

1. Named Stage:

* Named storage location for files accessible across users, sessions, and warehouses.
* Provides centralized and persistent storage.
* Can be used for various data operations and file storage needs.

**Use case 25:** What are convenient scenerios for user stage, table stage & name stage?

**Solution:**

1. User stage:

This stage is a convenient option if your files will only be accessed by a single user, but need to be copied into multiple tables.

1. Table Stage:

This stage is a convenient option if your files need to be accessible to multiple users and only need to be copied into a single table.

1. Name Stage:

Centralized and persistent storage accessible across users. Ideal for collaborative data sharing and diverse data operations.

**Use case 26:** Create a file format for .txt file with delimiter as pipe (|)

**Solution:**

CREATE OR REPLACE FILE FORMAT text\_format

TYPE = 'CSV'

FIELD\_DELIMITER = '|'

SKIP\_HEADER = 0;

**Use case 27:** Load table (employee) with pipe(|) delimiter text file to user stage, try query it from stage , finally load to target table.

**Solution:**