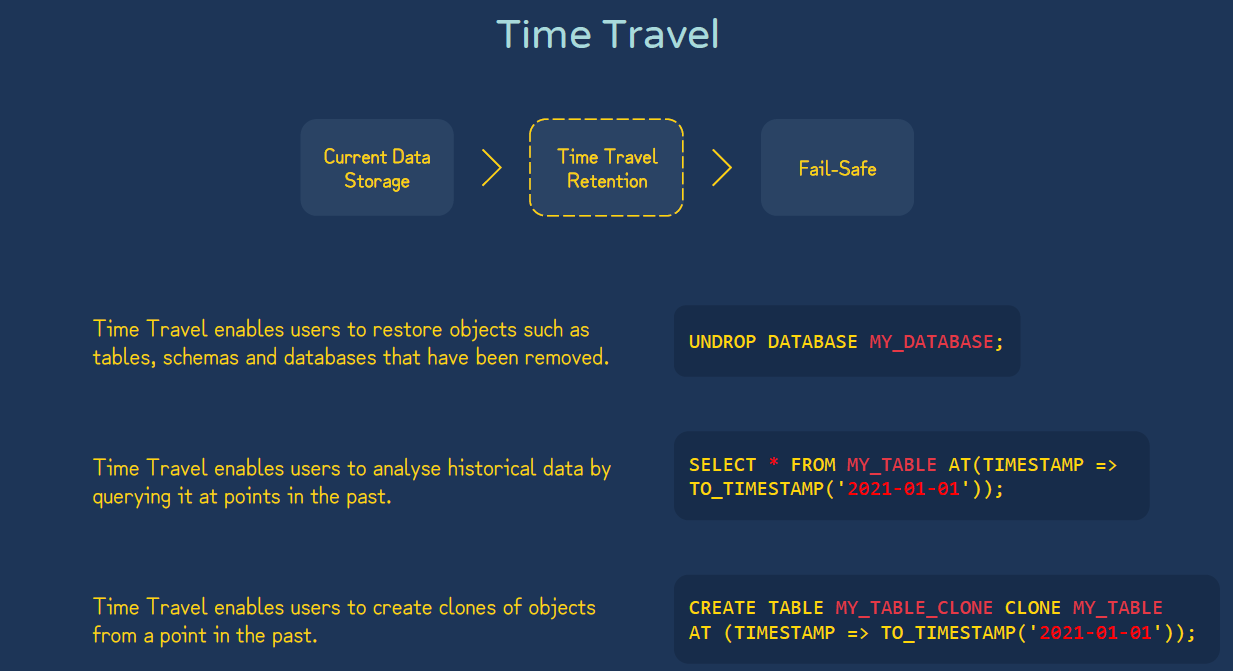
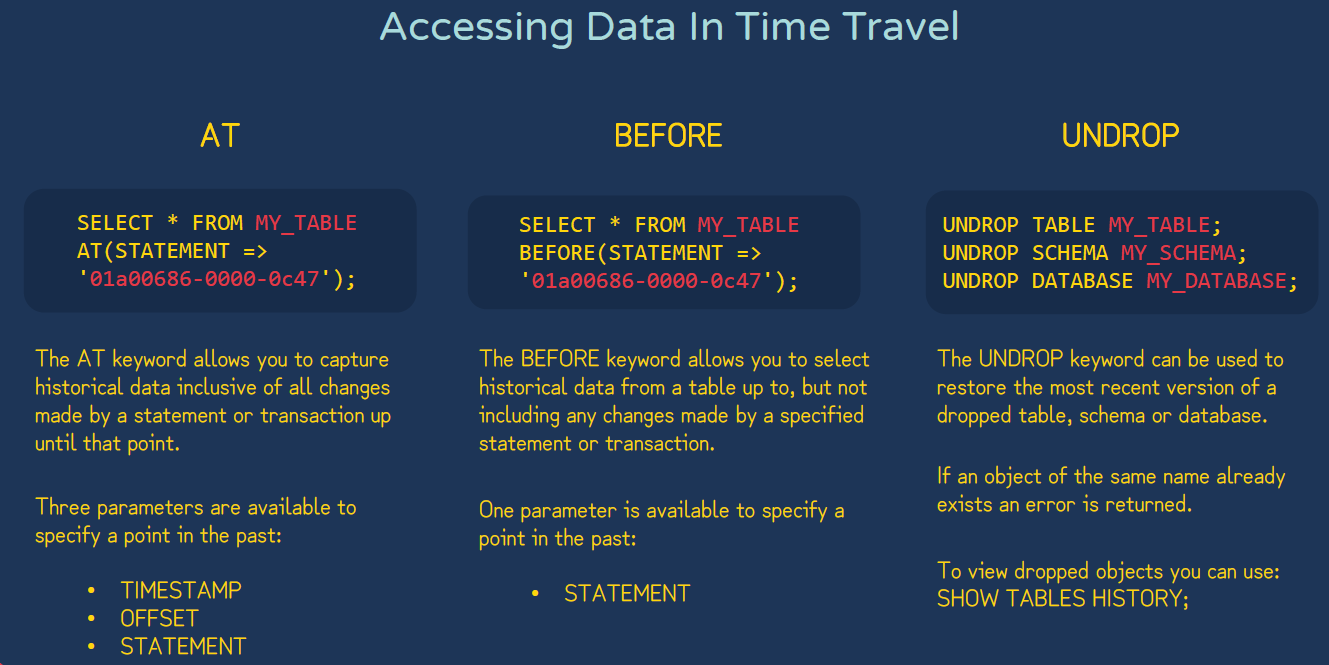
**Time Travel:**

**Time Travel** in **Snowflake** is a **powerful feature** that lets you **query, clone, or restore data** from the **past state** of tables, schemas, or databases — without having to maintain backups manually.

It’s extremely useful for **data recovery**, **accidental deletion rollback**, and **auditing**.





**Example:**

// Setting up table

CREATE OR REPLACE TABLE FIRST\_DB.public.test (

id int,

first\_name string,

last\_name string,

email string,

gender string,

Job string,

Phone string);

CREATE OR REPLACE FILE FORMAT FIRST\_DB.file\_formats.csv\_file

type = csv

field\_delimiter = ','

skip\_header = 1;

CREATE OR REPLACE STAGE FIRST\_DB.external\_stages.time\_travel\_stage

URL = 's3://data-snowflake-fundamentals/time-travel/'

file\_format = FIRST\_DB.file\_formats.csv\_file;

**//loaded data at evening 5**

COPY INTO FIRST\_DB.public.test

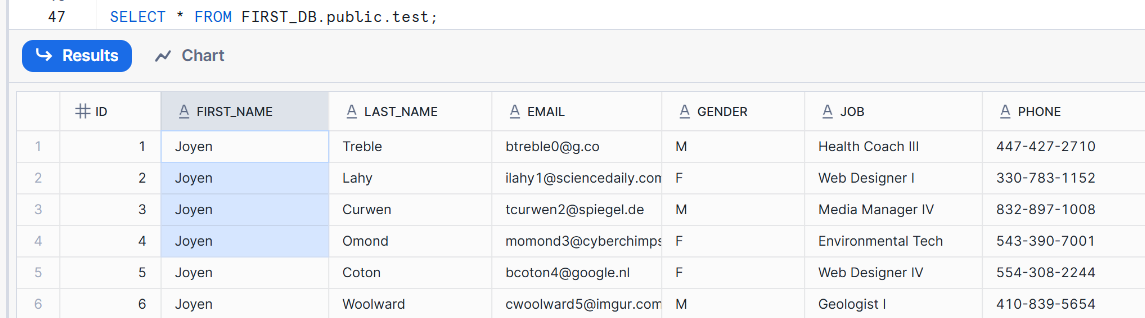
from @FIRST\_DB.external\_stages.time\_travel\_stage

files = ('customers.csv');

**//Executed update command at 5:03**

UPDATE FIRST\_DB.public.test

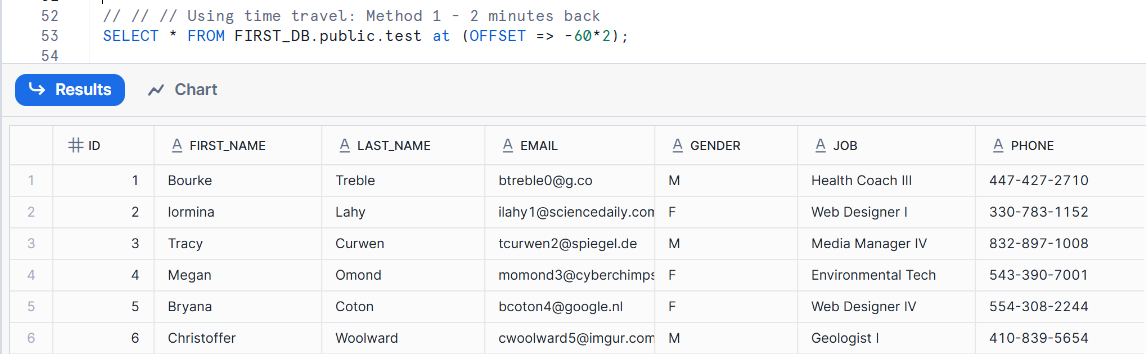
SET FIRST\_NAME = 'Joyen' ;



// Using time travel: Method 1 - 2 minutes back

**//ran this query at 05:04**

SELECT \* FROM FIRST\_DB.public.test at (OFFSET => -60\*2);



-- Setting up UTC time for convenience

ALTER SESSION SET TIMEZONE ='UTC';

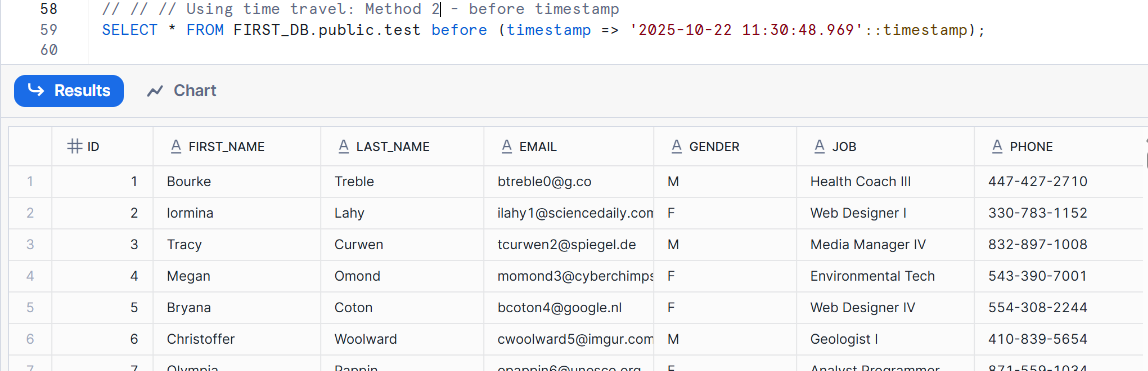
**//ran this command at 5:05**

SELECT current\_timestamp;



// Using time travel: Method 2 - before timestamp

SELECT \* FROM FIRST\_DB.public.test before (timestamp => '2025-10-22 11:30:48.969'::timestamp);



// Altering table (by mistake)

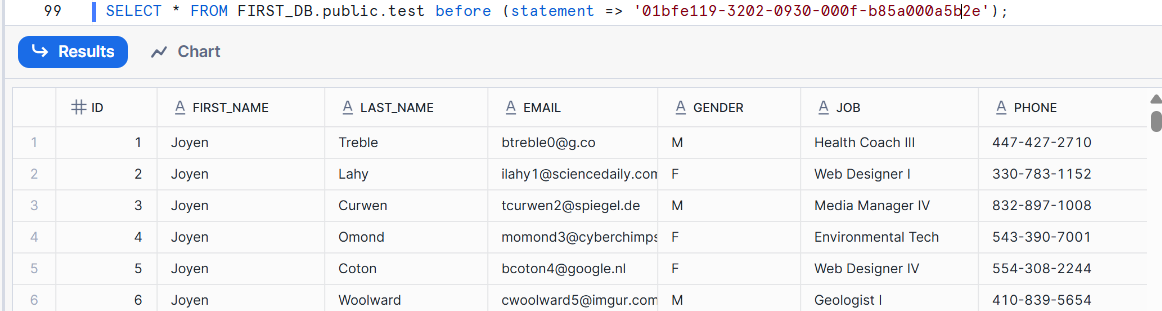
**//copy the query ID after running below query**

UPDATE FIRST\_DB.public.test

SET EMAIL = null;

// Using time travel: Method 3 - before Query ID

SELECT \* FROM FIRST\_DB.public.test before (statement => '01bfe119-3202-0930-000f-b85a000a5b2e');



**Example: UNDROP**

// Setting up table

CREATE OR REPLACE STAGE FIRST\_DB.external\_stages.time\_travel\_stage

URL = 's3://data-snowflake-fundamentals/time-travel/'

file\_format = FIRST\_DB.file\_formats.csv\_file;

CREATE OR REPLACE TABLE FIRST\_DB.public.customers (

id int,

first\_name string,

last\_name string,

email string,

gender string,

Job string,

Phone string);

COPY INTO FIRST\_DB.public.customers

from @FIRST\_DB.external\_stages.time\_travel\_stage

files = ('customers.csv');

// UNDROP command - Tables

DROP TABLE FIRST\_DB.public.customers;

SELECT \* FROM FIRST\_DB.public.customers;

UNDROP TABLE FIRST\_DB.public.customers;

// UNDROP command - Schemas

DROP SCHEMA FIRST\_DB.public;

SELECT \* FROM FIRST\_DB.public.customers;

UNDROP SCHEMA FIRST\_DB.public;

// UNDROP command - Database

DROP DATABASE FIRST\_DB;

SELECT \* FROM FIRST\_DB.public.customers;

UNDROP DATABASE FIRST\_DB;

// Undroping a with a name that already exists

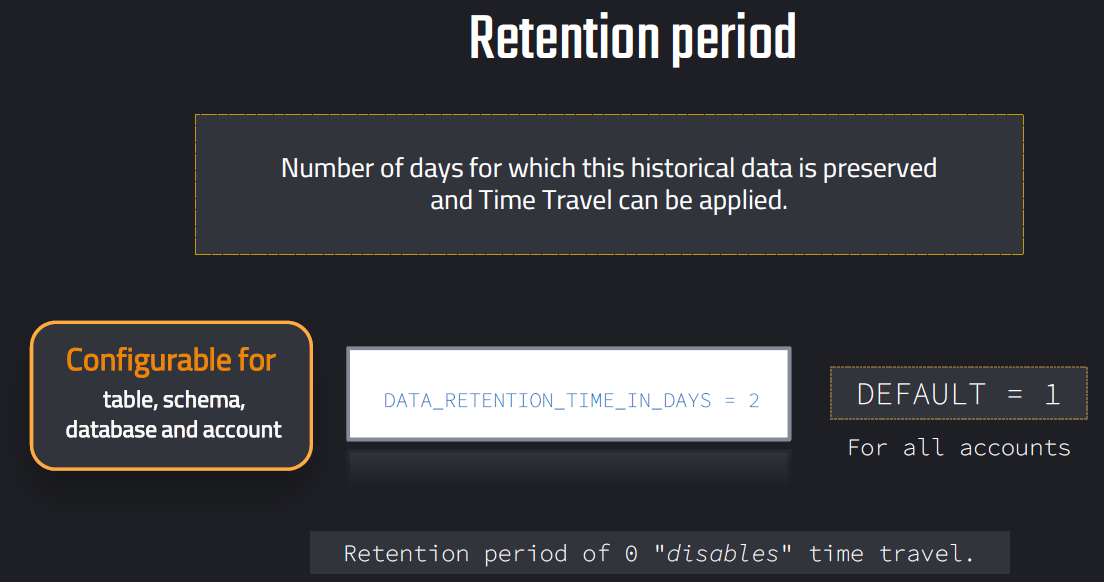
SELECT \* FROM FIRST\_DB.public.customers;

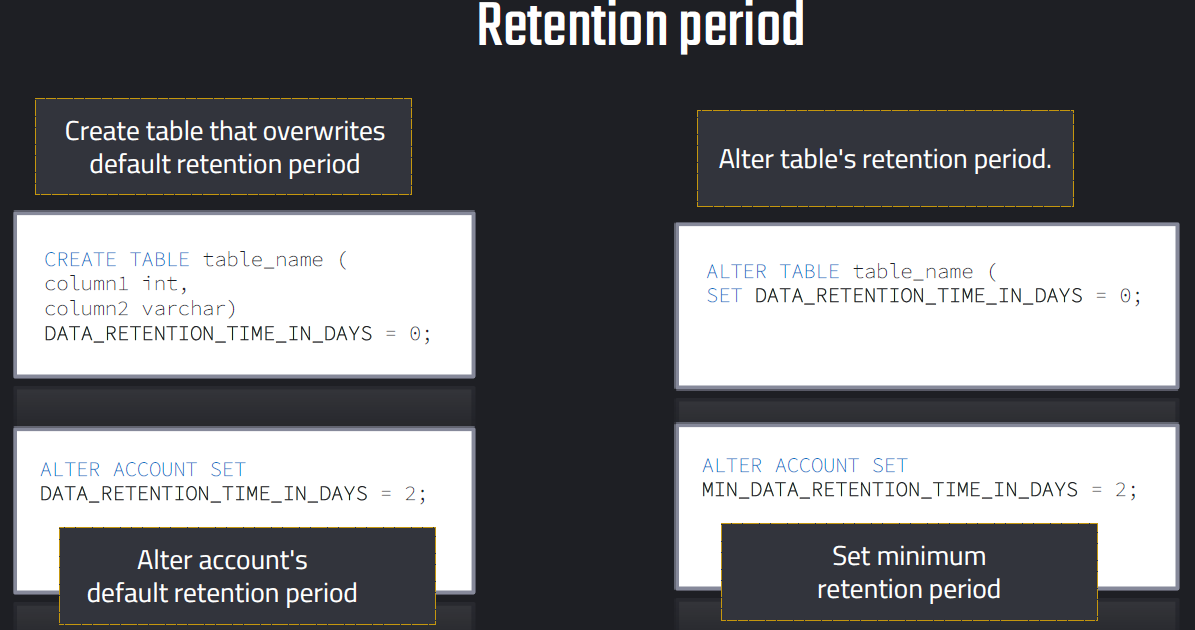
UNDROP table FIRST\_DB.public.customers;--**if we try to undrop already existing table we will get error "Object 'CUSTOMERS' already exists."**

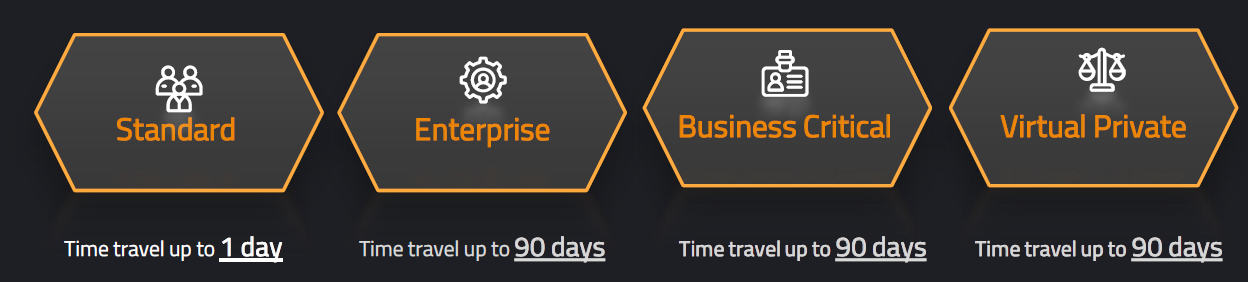
ALTER TABLE FIRST\_DB.public.customers

RENAME TO FIRST\_DB.public.customers\_new;

UNDROP table FIRST\_DB.public.customers;--**if we try to undrop after renaming table we will get error "Table CUSTOMERS did not exist or was purged."**







Example:

//creating table

CREATE OR REPLACE TABLE FIRST\_DB.public.customers (

id int,

first\_name string,

last\_name string,

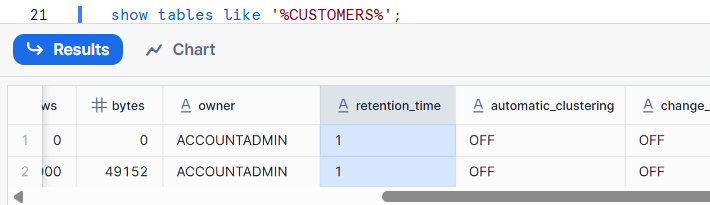
email string,

gender string,

Job string,

Phone string);

**Default retention time for newly created table is 1 day**



// Change retention period

alter table FIRST\_DB.public.customers

set data\_retention\_time\_in\_days = 0;

// Drop and undrop table

drop table first\_db.public.customers;

undrop table first\_db.public.customers; --ERROR: “Table CUSTOMERS did not exist or was purged.”

alter account set data\_retention\_time\_in\_days = 2;

//recreating table

CREATE OR REPLACE TABLE FIRST\_DB.public.customers (

id int,

first\_name string,

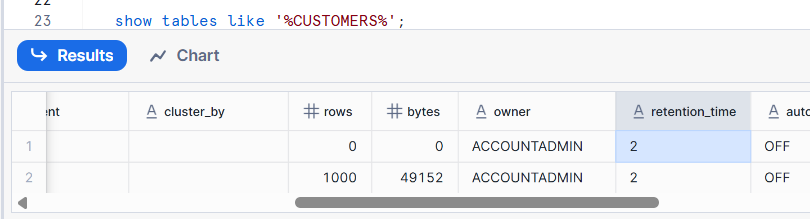
last\_name string,

email string,

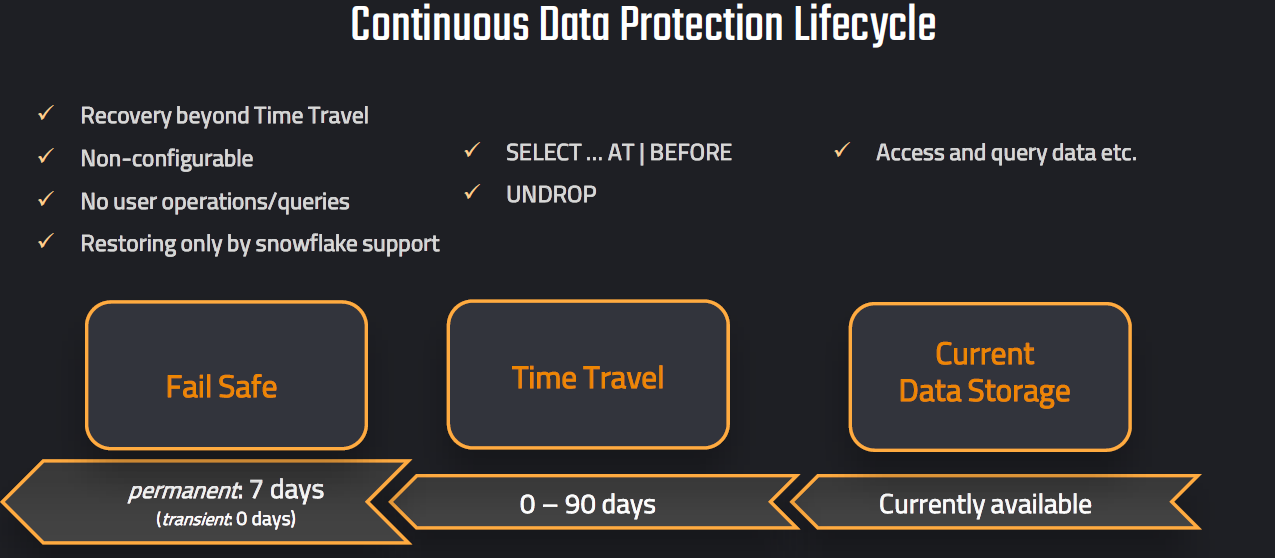
gender string,

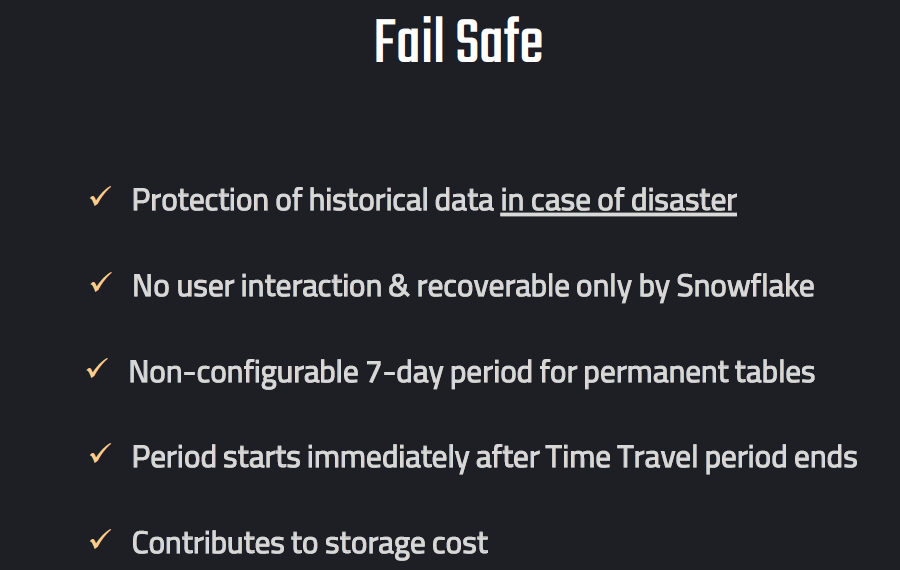
Job string,

Phone string);



**Fail Safe:**

****

****

**Storage Usage:**

// Storage usage on account level

SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.STORAGE\_USAGE ORDER BY USAGE\_DATE DESC;

// Storage usage on table level

SELECT \* FROM SNOWFLAKE.ACCOUNT\_USAGE.TABLE\_STORAGE\_METRICS;

// Storage usage on table level - formatted

SELECT ID,

TABLE\_NAME,

TABLE\_SCHEMA,

TABLE\_CATALOG,

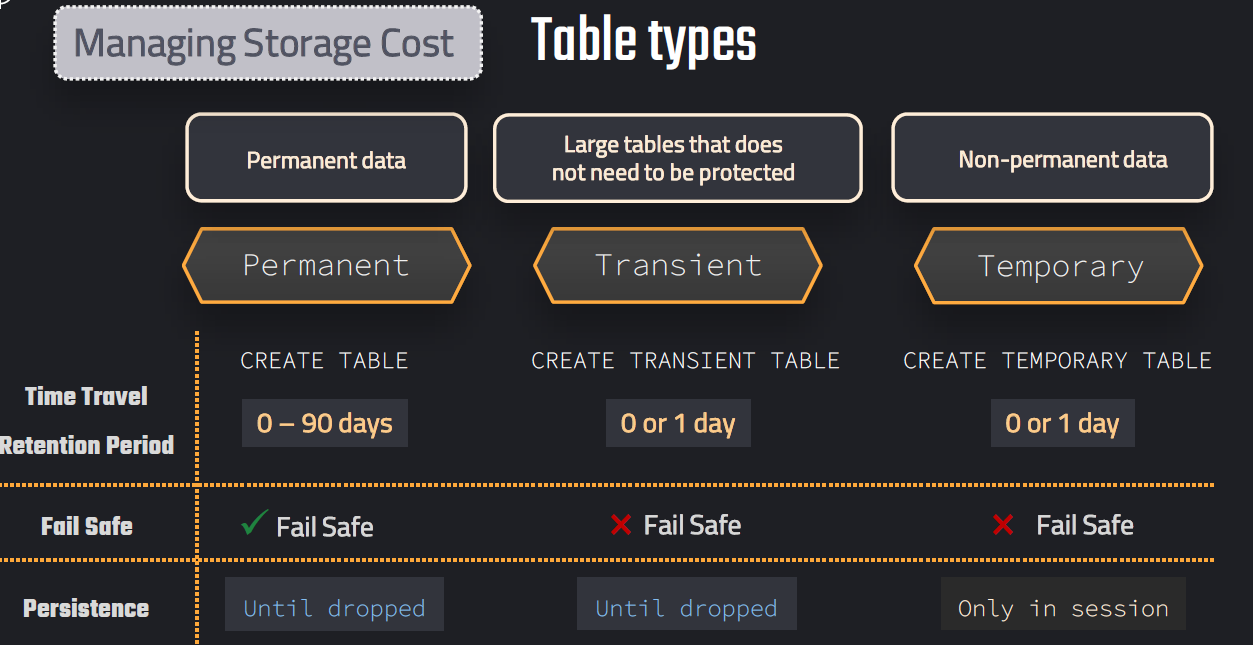
ACTIVE\_BYTES / (1024\*1024\*1024) AS STORAGE\_USED\_GB,

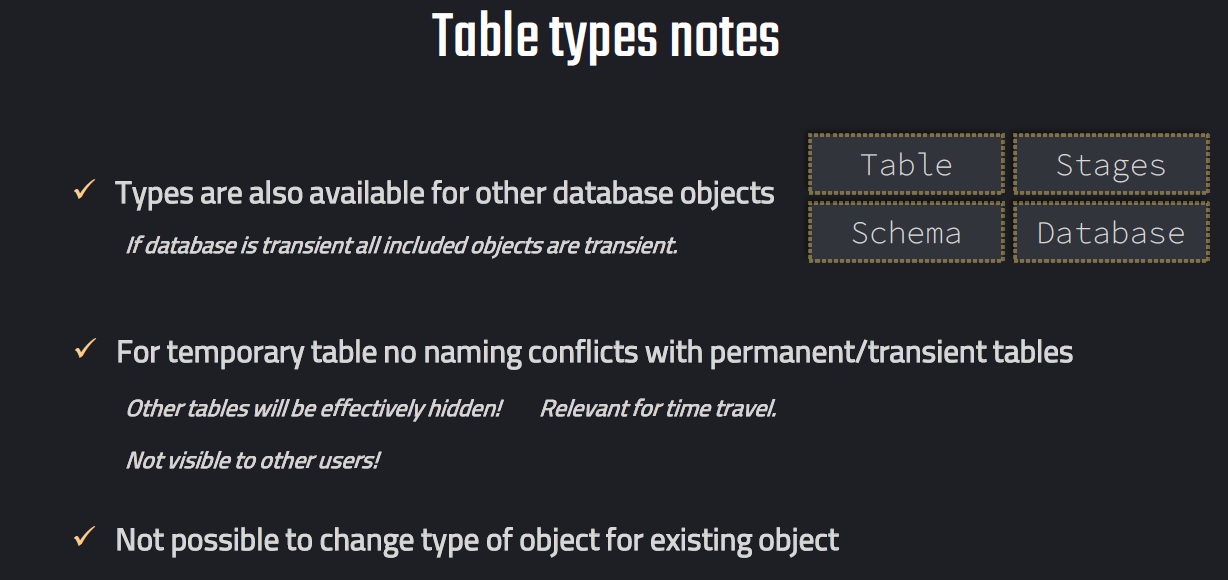
TIME\_TRAVEL\_BYTES / (1024\*1024\*1024) AS TIME\_TRAVEL\_STORAGE\_USED\_GB,

FAILSAFE\_BYTES / (1024\*1024\*1024) AS FAILSAFE\_GB

FROM SNOWFLAKE.ACCOUNT\_USAGE.TABLE\_STORAGE\_METRICS

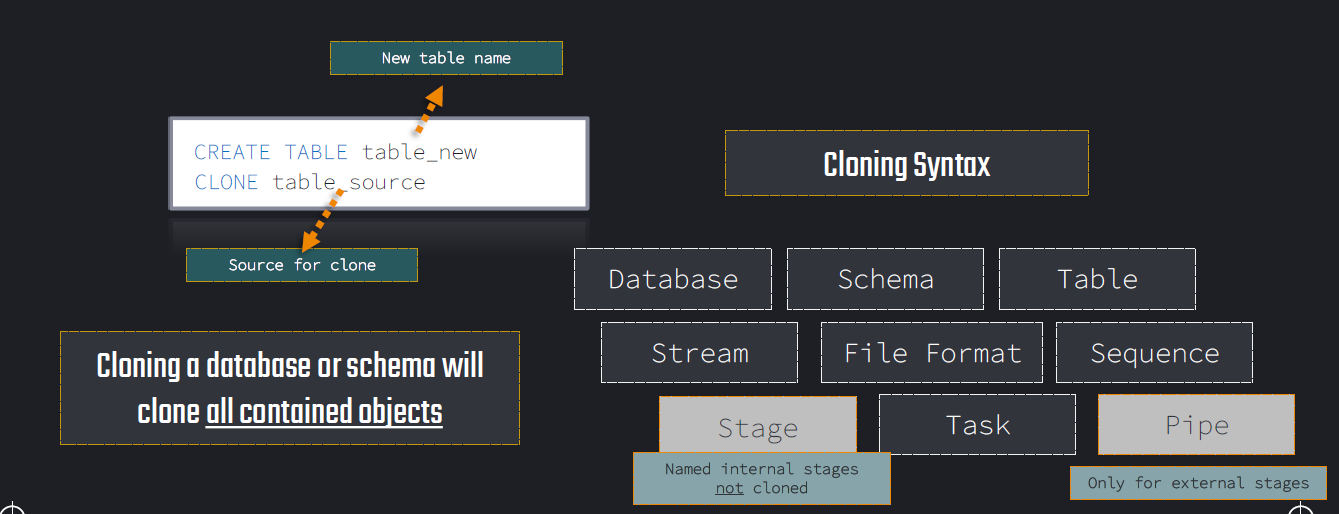
ORDER BY STORAGE\_USED\_GB DESC,TIME\_TRAVEL\_STORAGE\_USED\_GB DESC;



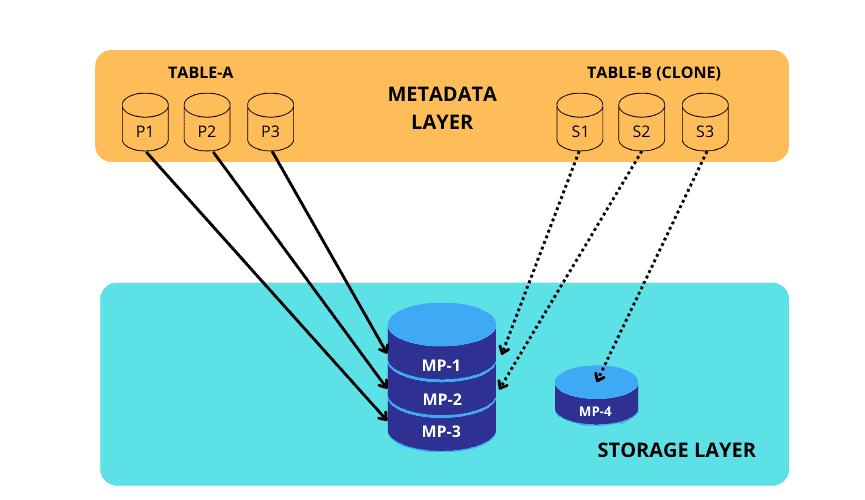


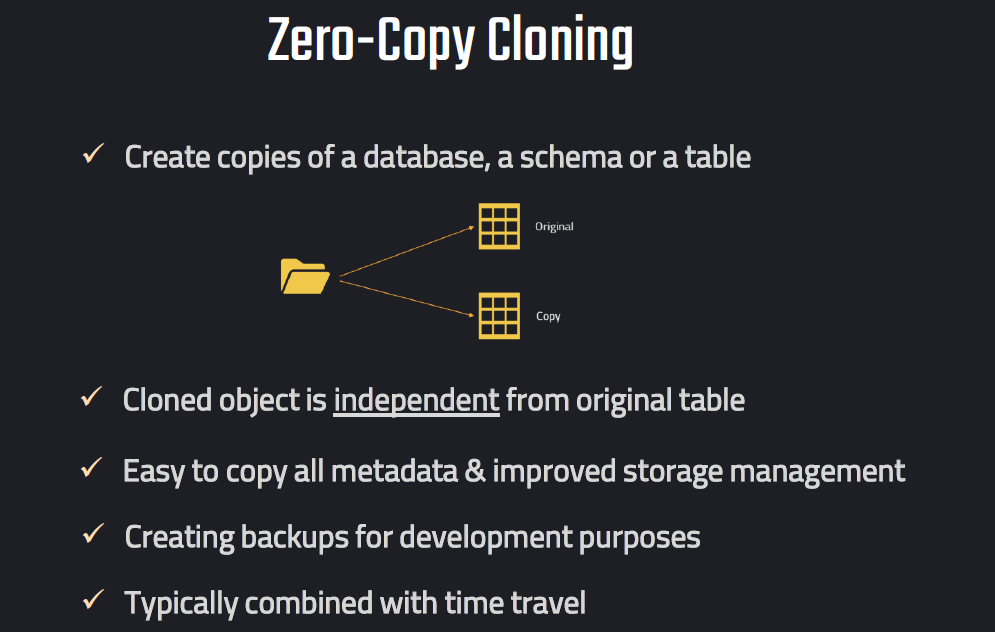
**ZERO COPY CLONING:**

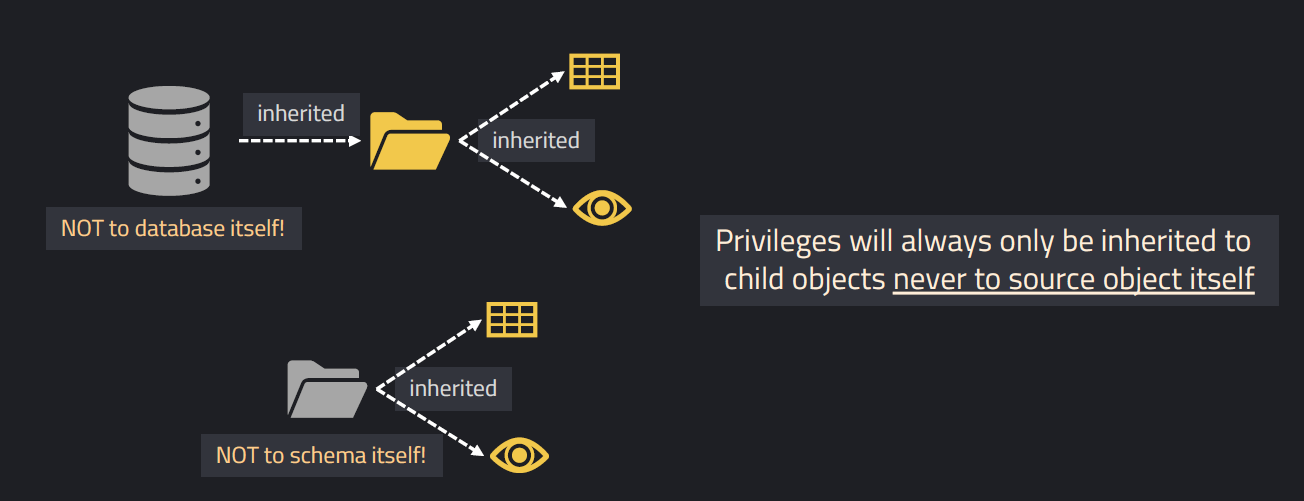
Snowflake’s Zero Copy Cloning is a feature which provides a quick and easy way to create a copy of any table, schema, or an entire database without incurring any additional costs as the derived copy shares the underlying storage with the original object.

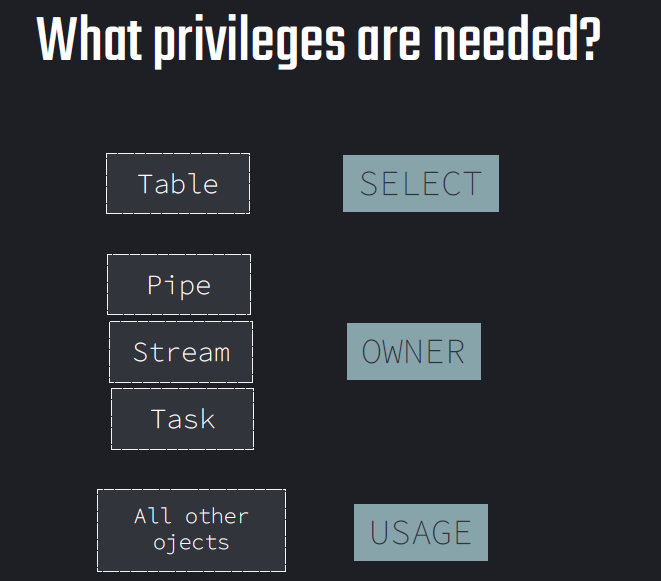


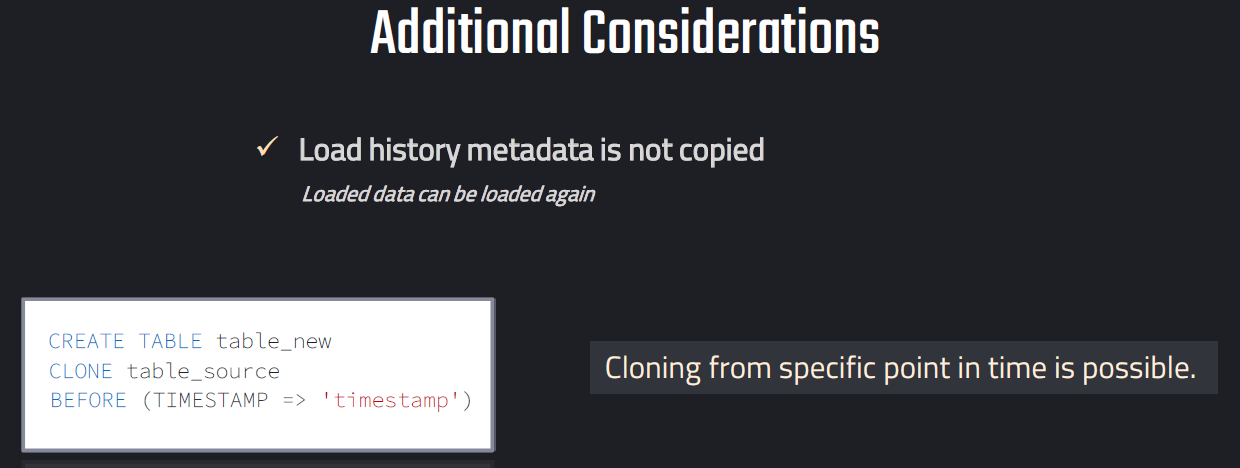
Micro-partitions in Snowflake are immutable i.e., once created lasts in the same state until table is dropped. Hence, for any change in the data of a micro-partition, a new micro-partition is created with updated changes and metadata will point to the newly created micro partition. The older partition is retained for time-travel and Fail-safe purposes.











**Secure Data Sharing**

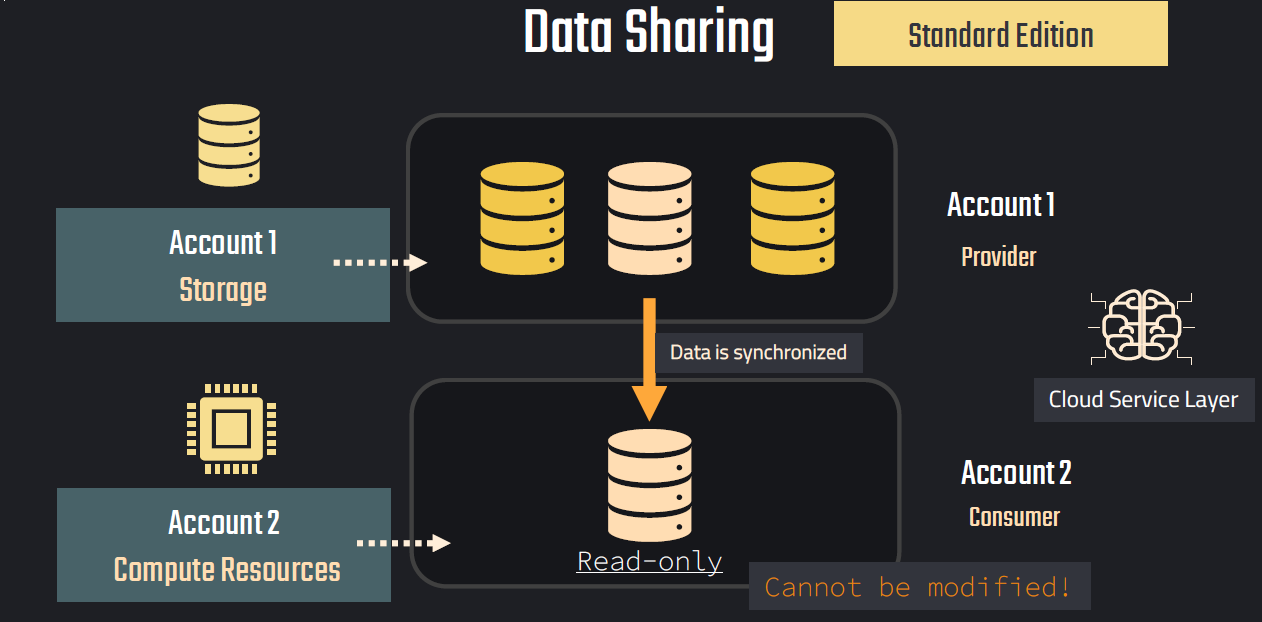
Secure Data Sharing in Snowflake enables account-to-account sharing of selected database objects in your account with other. It is an extremely powerful, yet easy-to-use feature.

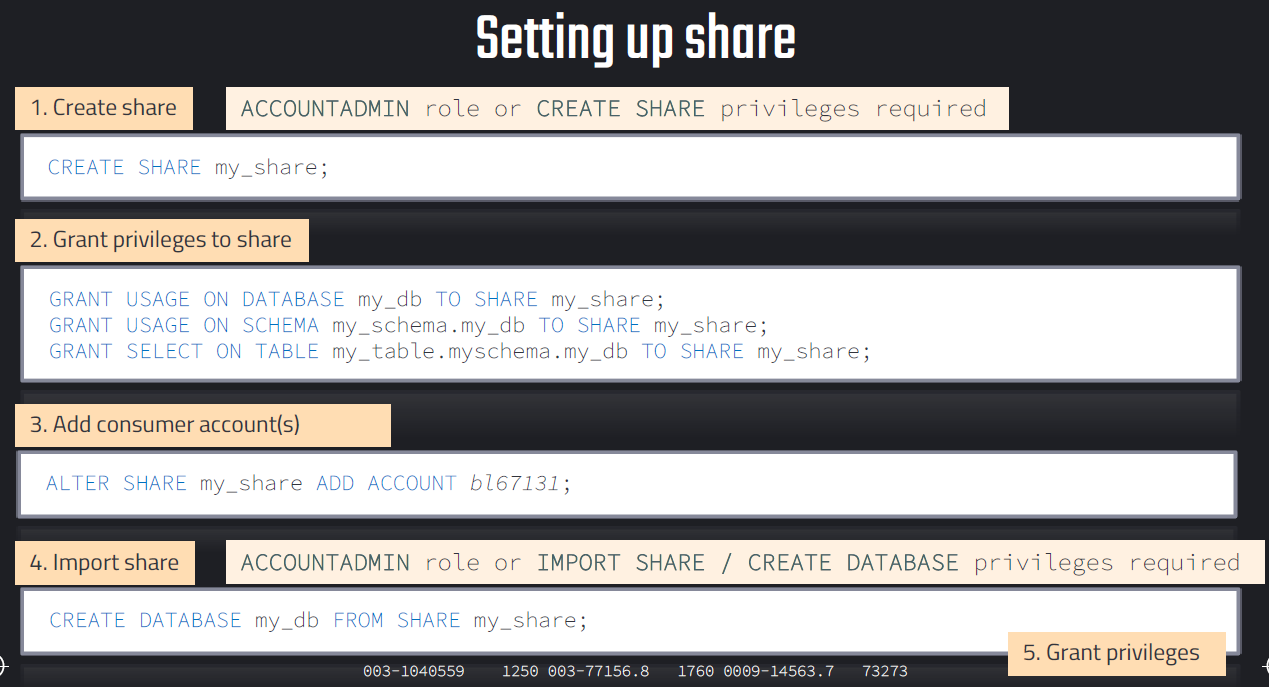
**Full Consumer Accounts** enable providers to share data with consumers who are already Snowflake customers.

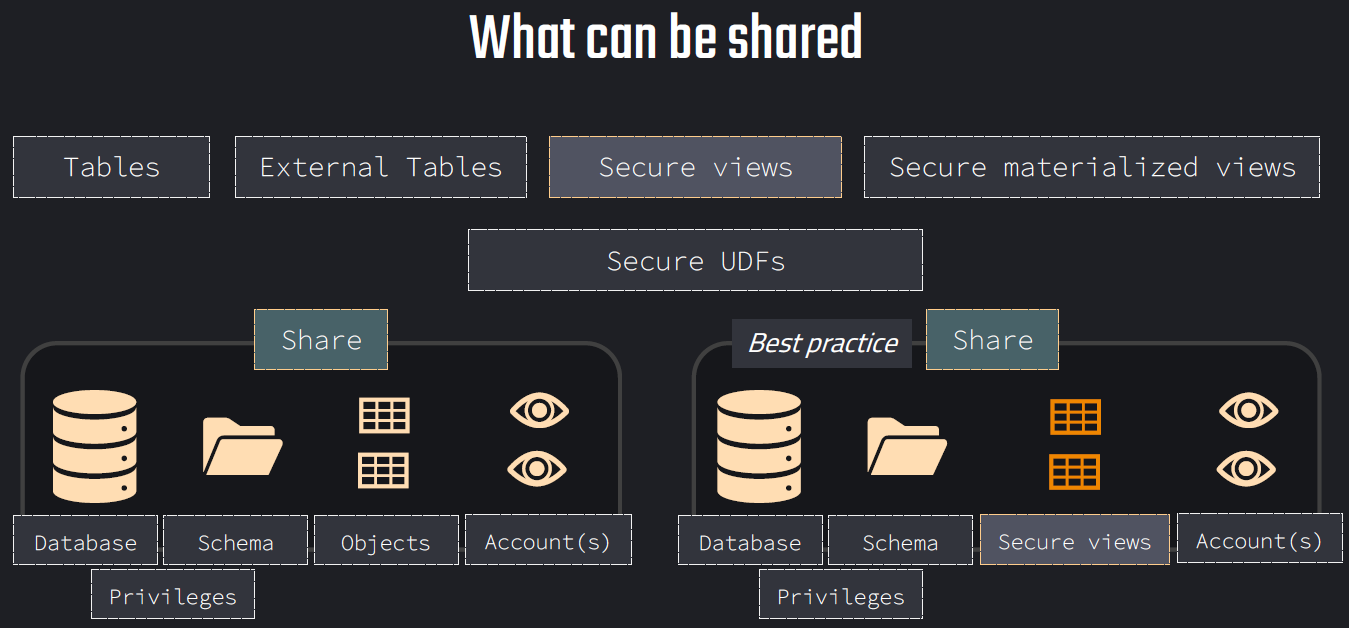
* If the consumer is already a Snowflake customer, the data can be shared directly to the consumer’s existing account.
* In this case, the consumer would pay for all compute resources incurred by querying the shared databases.

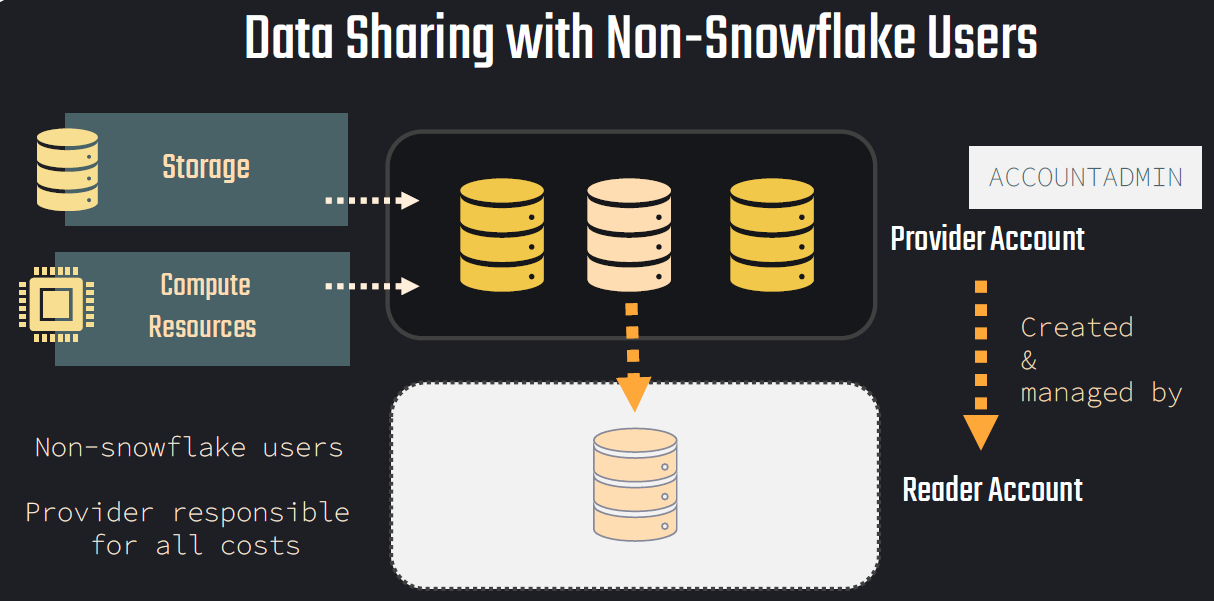
**Reader Accounts** enable providers to share data with consumers who are not already Snowflake customers, without requiring the consumers to become Snowflake customers.

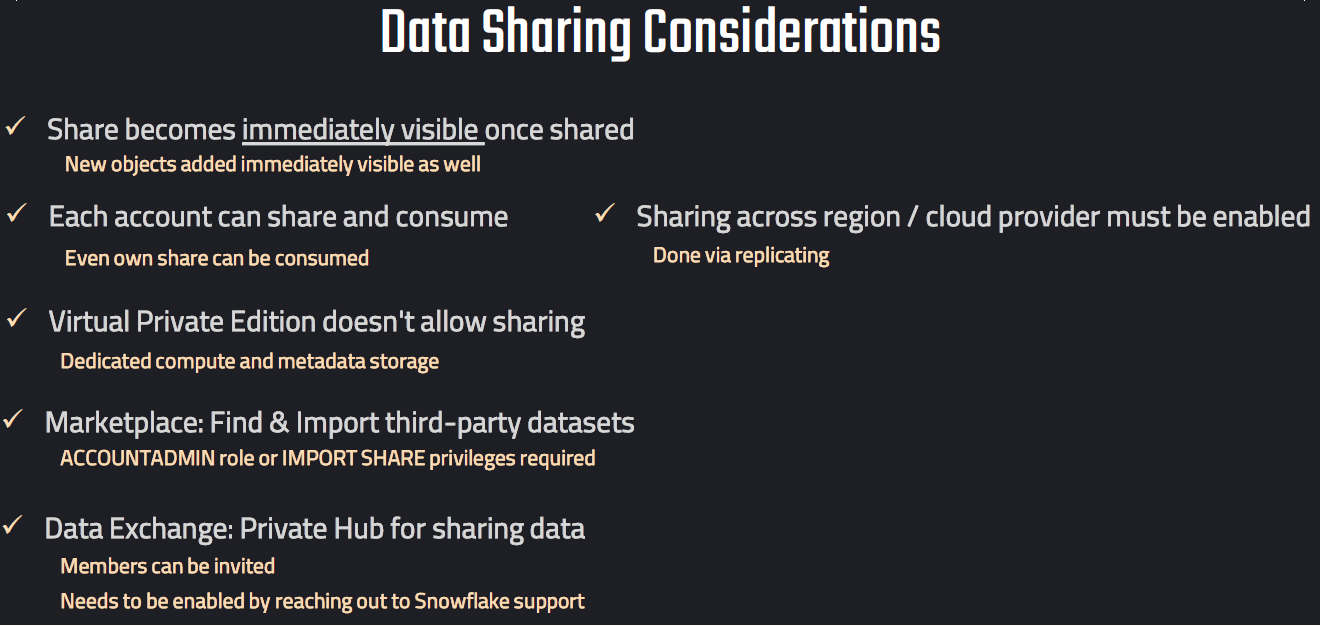
* If the consumer is not a Snowflake customer, the provider can create Reader Accounts that the consumer can use.
* In the case of Reader Accounts, all costs incurred by users in the reader account would be borne by the provider, although those costs could be tracked and invoiced back to the consumer.

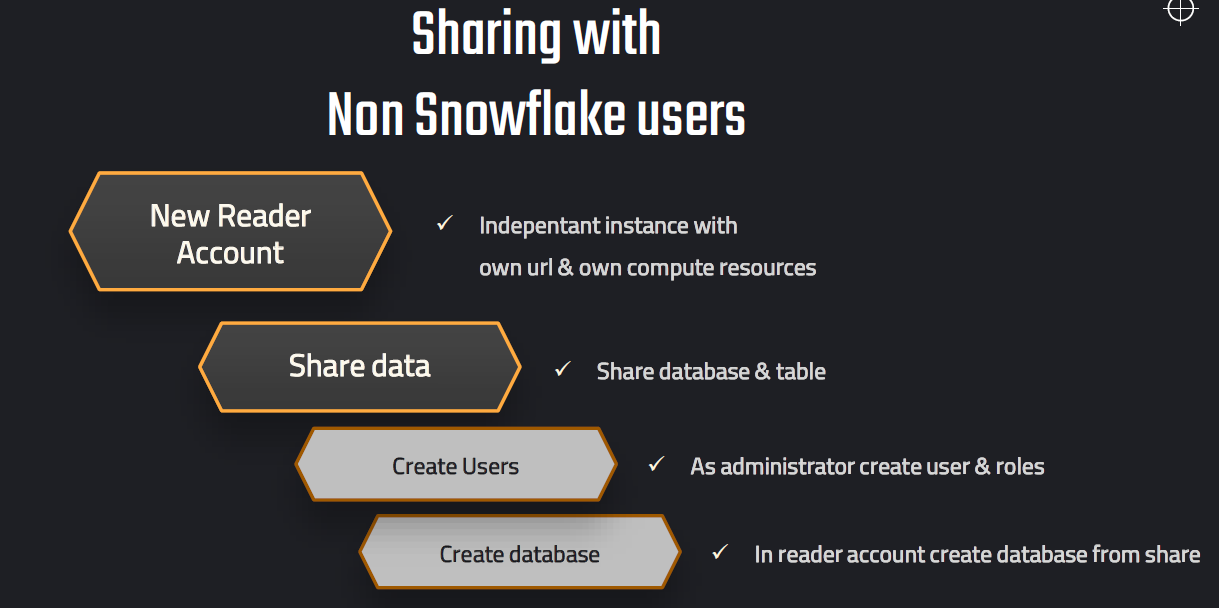












**Example:**

//create database

CREATE OR REPLACE DATABASE DATA\_S;

//create stage

CREATE OR REPLACE STAGE aws\_stage

url='s3://bucketsnowflakes3';

// List files in stage

LIST @aws\_stage;

// Create table

CREATE OR REPLACE TABLE ORDERS (

ORDER\_ID VARCHAR(30)

,AMOUNT NUMBER(38,0)

,PROFIT NUMBER(38,0)

,QUANTITY NUMBER(38,0)

,CATEGORY VARCHAR(30)

,SUBCATEGORY VARCHAR(30));

// Load data using copy command

COPY INTO ORDERS

FROM @FIRST\_DB.external\_stages.aws\_stage

file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*OrderDetails.\*';

SELECT \* FROM ORDERS;

--- Why Secure view is recommended? ---

CREATE OR REPLACE VIEW ORDERS\_VIEW AS

SELECT

ORDER\_ID,

AMOUNT,

QUANTITY

FROM ORDERS

WHERE CATEGORY != 'Furniture';

SHOW VIEWS LIKE '%ORDER%';

-- Create Secure View

CREATE OR REPLACE SECURE VIEW ORDERS\_VIEW\_SECURE AS

SELECT

ORDER\_ID,

AMOUNT,

QUANTITY

FROM ORDERS

WHERE CATEGORY != 'Furniture';

//// STEP 1: Create a share object

-- You need the ACCOUNTADMIN role

USE ROLE ACCOUNTADMIN;

-- Create Share

CREATE OR REPLACE SHARE ORDERS\_SHARE;

//// STEP 2: Setup Grants

// Grant usage on database

GRANT USAGE ON DATABASE DATA\_S TO SHARE ORDERS\_SHARE;

// Grant usage on schema

GRANT USAGE ON SCHEMA DATA\_S.PUBLIC TO SHARE ORDERS\_SHARE;

// Grant SELECT on table

--GRANT SELECT ON TABLE DATA\_S.PUBLIC.ORDERS TO SHARE ORDERS\_SHARE;

// Grant select on view

GRANT SELECT ON VIEW DATA\_S.PUBLIC.ORDERS\_VIEW TO SHARE ORDERS\_SHARE;--**ERROR: Non-secure object can only be granted to shares with "secure\_objects\_only" property set to false.**

GRANT SELECT ON VIEW DATA\_S.PUBLIC.ORDERS\_VIEW\_SECURE TO SHARE ORDERS\_SHARE;

// Validate Grants

SHOW GRANTS TO SHARE ORDERS\_SHARE;

-- Create Reader Account --

CREATE MANAGED ACCOUNT reader\_account

ADMIN\_NAME = read\_acc\_admin,

ADMIN\_PASSWORD = 'Pwd\_1234567890',

TYPE = READER;

--- To drop the account again: DROP MANAGED ACCOUNT reader\_account;

// Show accounts

SHOW MANAGED ACCOUNTS;

-- Share the data --

ALTER SHARE ORDERS\_SHARE

ADD ACCOUNT = ya15332;

-- Sharing to a lower edition

ALTER SHARE ORDERS\_SHARE

ADD ACCOUNT = ya15332

SHARE\_RESTRICTIONS=false;

//// STEP 4:Create database from share ////

**--- By using reader account ---**

// Show all shares (consumer & producers)

SHOW SHARES;

// See details on share

DESC SHARE WIDAYUS.QJ74933.ORDERS\_SHARE;

// Create a database in consumer account using the share

CREATE DATABASE DATA\_SHARE\_DB FROM SHARE WIDAYUS.QJ74933.ORDERS\_SHARE;

// Validate table access

SELECT \* FROM DATA\_SHARE\_DB.PUBLIC.ORDERS\_VIEW\_SECURE;

// Setup virtual warehouse

CREATE WAREHOUSE READ\_WH WITH

WAREHOUSE\_SIZE='X-SMALL'

AUTO\_SUSPEND = 180

AUTO\_RESUME = TRUE

INITIALLY\_SUSPENDED = TRUE;

//// STEP 5: Grant privileges optionally

-- Create and set up users --

// Create user

CREATE USER MYRIAM PASSWORD = 'difficult\_passw@ord=123';

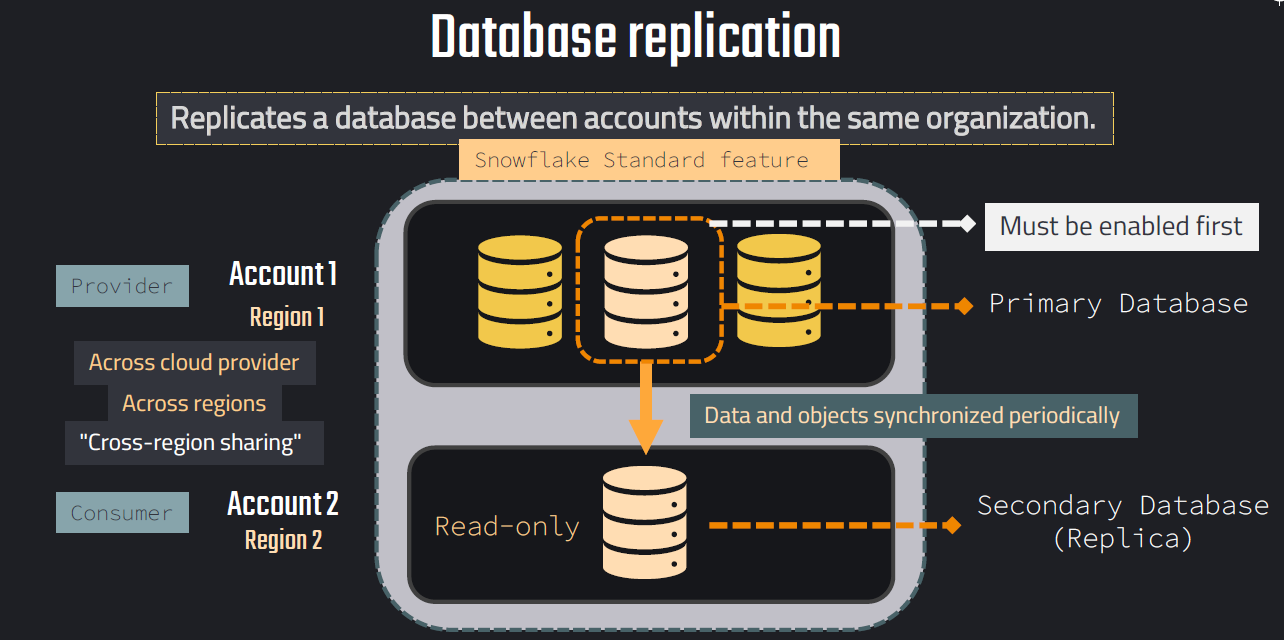
// Grant usage on warehouse

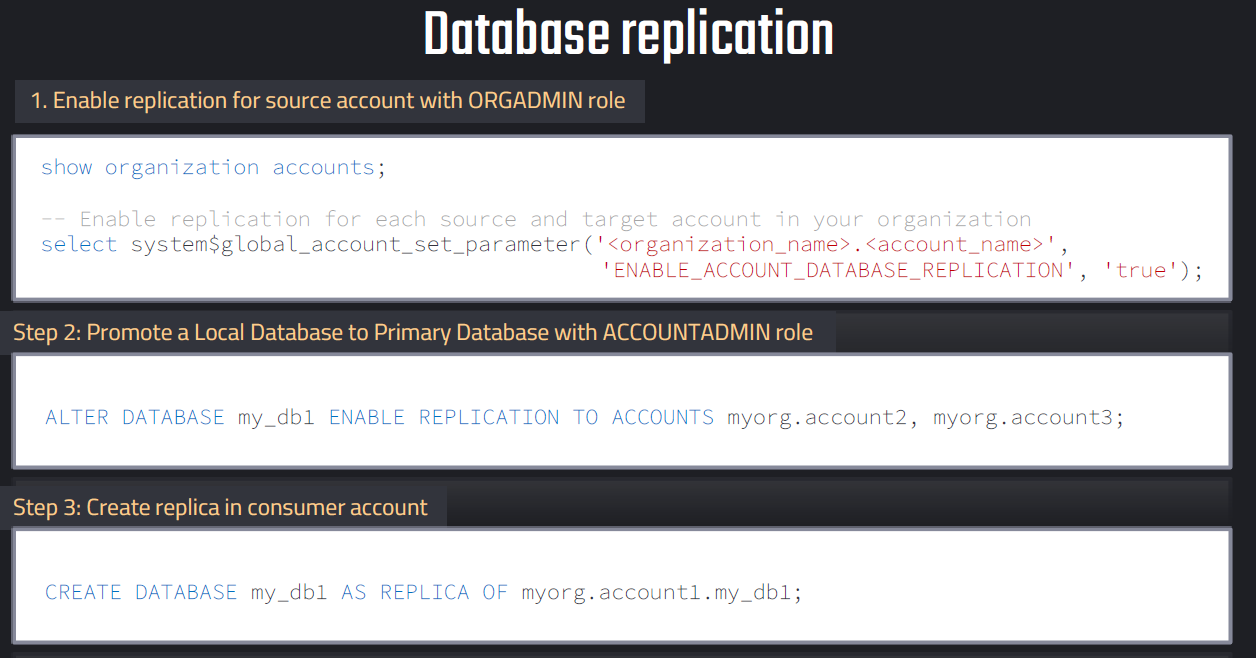
GRANT USAGE ON WAREHOUSE READ\_WH TO ROLE PUBLIC;

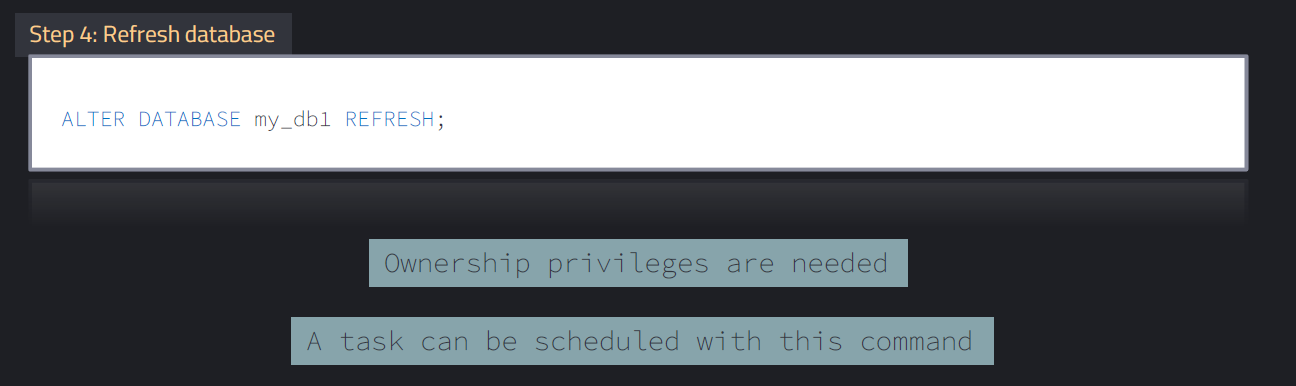
// Grating privileges on a Shared Database for other users

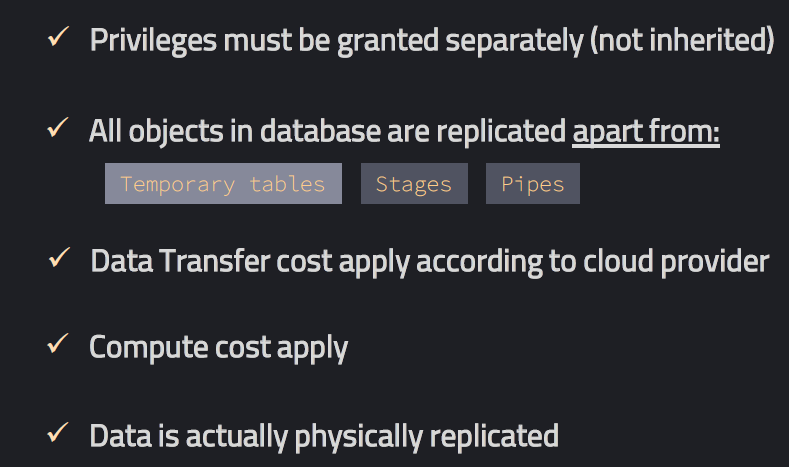
GRANT IMPORTED PRIVILEGES ON DATABASE DATA\_SHARE\_DB TO ROLE PUBLIC;

**Data Replication:**

****

****

****

****