1. Clone a Table from another DB. E.g. Assume we have 2 DBs – DEV\_DB and PRD\_DB and the Roles for these DBs are DEV\_ROLE and PRD\_ROLE. The Roles should be exclusive to the DB which means DEV\_ROLE should not be able to access the PROD\_DB and vice versa. Now Clone a Table from PRD\_DB to DEV\_DB.

USE ROLE ACCOUNTADMIN;

-- CREATING DBs

CREATE DATABASE DEV\_DB;

CREATE DATABASE PRD\_DB;

USE PRD\_DB;

-- CREATING TABLE in PRD\_DB

CREATE OR REPLACE TABLE PRD\_DB.PUBLIC.members(

id number(8) NOT NULL,

name varchar(255) default NULL,

fee number(3) NULL

);

-- INSERTING VALUES INSIDE THE Table

INSERT INTO PRD\_DB.PUBLIC.members (id,name,fee)

VALUES

(1,'Joe',0),

(2,'Jane',0),

(3,'George',0),

(4,'Betty',0),

(5,'Sally',0);

--CLONING THE TABLE to PRD\_DB

CREATE OR REPLACE Transient TABLE DEV\_DB.PUBLIC.members

CLONE PRD\_DB.PUBLIC.members;

SELECT \* FROM DEV\_DB.PUBLIC.members;

CREATE OR REPLACE ROLE DEV\_ROLE;

-- Granting access to the Role

GRANT USAGE ON DATABASE DEV\_DB TO ROLE DEV\_ROLE;

GRANT ALL ON ALL SCHEMAS IN DATABASE DEV\_DB TO ROLE DEV\_ROLE;

GRANT ALL ON ALL TABLES IN DATABASE DEV\_DB TO ROLE DEV\_ROLE;

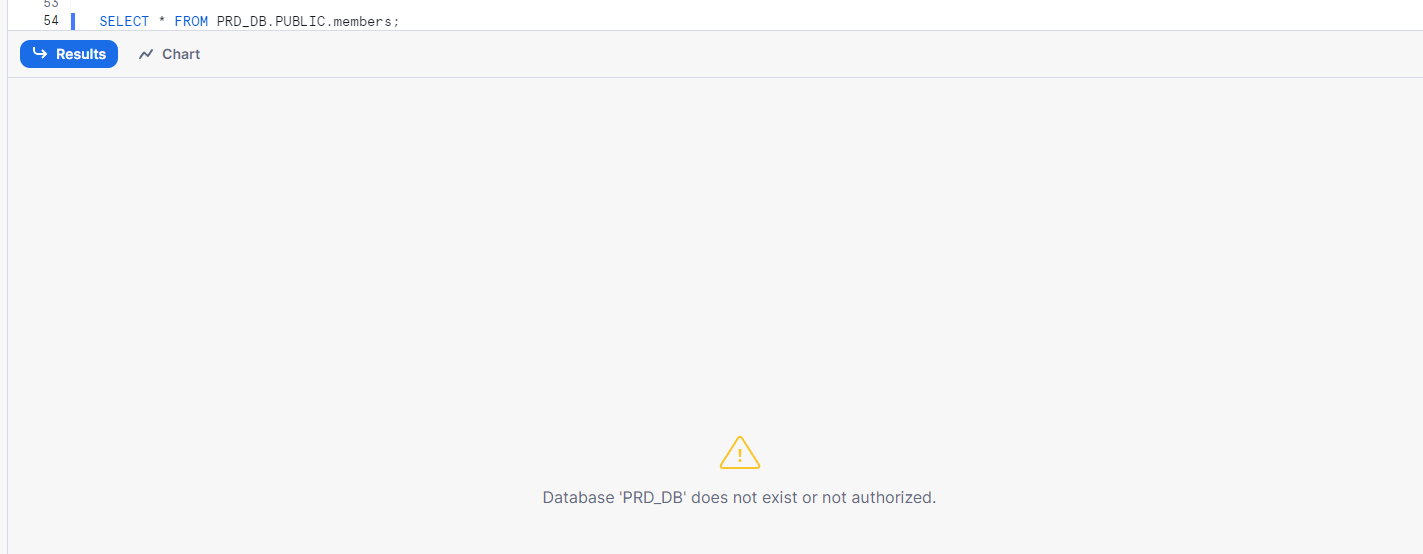
GRANT USAGE ON WAREHOUSE COMPUTE\_WH TO ROLE DEV\_ROLE;

GRANT ROLE DEV\_ROLE TO USER MELWINMPK;

USE ROLE DEV\_ROLE;

SELECT \* FROM DEV\_DB.PUBLIC.members;





2. Create a Temporary Table, Insert some data, do some Updates/Delete on the data and restore the Original data using Time Travel.

USE DEV\_DB;

-- Create a temporary table

CREATE Temporary TABLE DEV\_DB.PUBLIC.temp\_table AS

SELECT \* FROM DEV\_DB.PUBLIC.members;

INSERT INTO DEV\_DB.PUBLIC.temp\_table (id,name,fee)

VALUES

(6,'Alvin',0),

(7,'John',0);

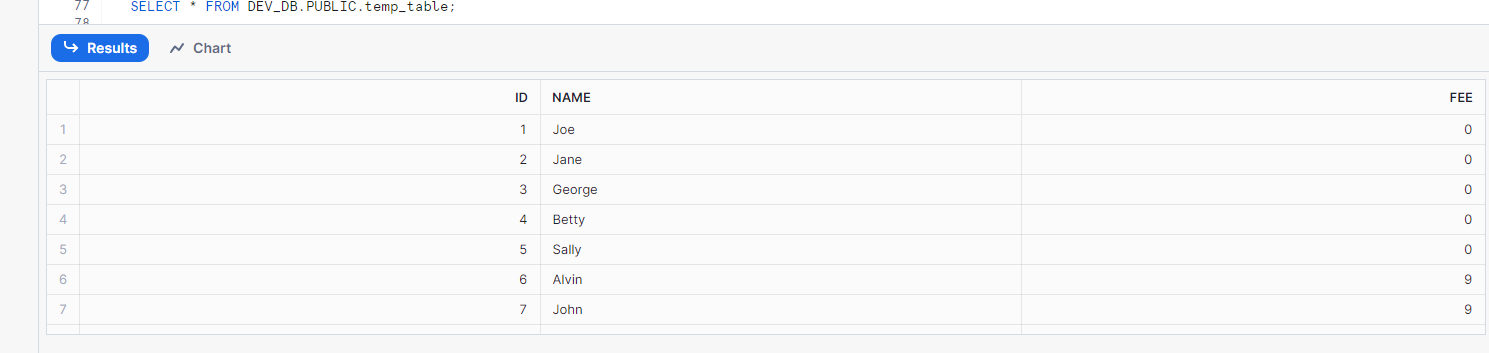
SELECT CURRENT\_TIMESTAMP(); -- 2024-05-28 01:04:51.259 -0700

UPDATE DEV\_DB.PUBLIC.temp\_table

SET fee = 9

WHERE id IN (6,7) ;

SELECT \* FROM DEV\_DB.PUBLIC.temp\_table;



ALTER SESSION SET TIMEZONE = 'UTC';

-- Restore original data using Time Travel

SELECT \*

FROM DEV\_DB.PUBLIC.temp\_table

AT (TIMESTAMP => TO\_TIMESTAMP('2024-05-28 01:04:51.259'));

-- Specify the timestamp before any changes



3. Create a Transient Table, Insert some data, do some Updates/Delete on the data and restore the Original data using Time Travel.

-- Create a transient table

CREATE TRANSIENT TABLE DEV\_DB.PUBLIC.my\_transient\_table AS

SELECT \* FROM DEV\_DB.PUBLIC.members;

-- Insert data into the transient table

INSERT INTO DEV\_DB.PUBLIC.my\_transient\_table (id,name,fee)

VALUES

(6,'Alvin',0),

(7,'John',0);

SELECT CURRENT\_TIMESTAMP(); -- 2024-05-28 08:53:10.012 +0000

-- Perform updates and deletes

UPDATE DEV\_DB.PUBLIC.my\_transient\_table

SET fee = 9

WHERE id IN (6,7) ;

SELECT CURRENT\_TIMESTAMP(); -- 2024-05-28 08:54:06.127 +0000

SELECT \*

FROM DEV\_DB.PUBLIC.my\_transient\_table;

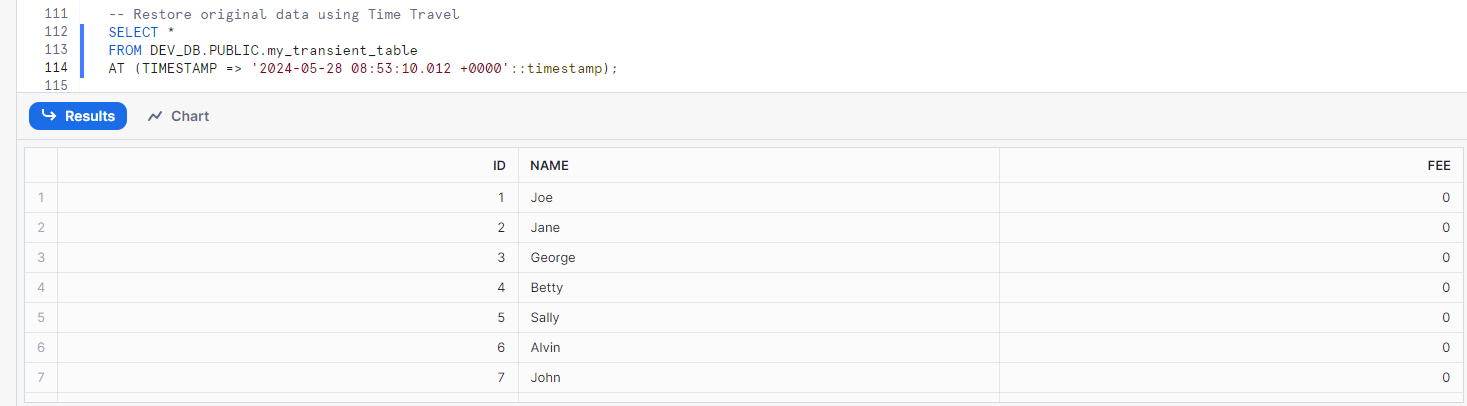


-- Restore original data using Time Travel

SELECT \*

FROM DEV\_DB.PUBLIC.my\_transient\_table

AT (TIMESTAMP => '2024-05-28 08:53:10.012 +0000'::timestamp);



4. Create a normal (Permanent) Table, set the Time Travel Interval to 1 day, drop the Table and restore it using the FailSafe option.

-- Create a permanent table

CREATE TABLE DEV\_DB.PUBLIC.my\_permanent\_table (

id INT,

name STRING

);

INSERT INTO DEV\_DB.PUBLIC.my\_permanent\_table (id, name) VALUES (1,'Tom'),(2,'Bob'),(2,'Alice');

-- Set the Time Travel interval to 1 day (default)

-- For Standard Endtions there is Only 1 Day od Time travel

-- For Enterprice Editions up to 90 days of timetravel

-- DATA\_RETENTION\_TIME\_IN\_DAYS = 1

-- alter Table my\_permanent\_table set DATA\_RETENTION\_TIME\_IN\_DAYS=1;

-- Drop the table (simulate accidental deletion)

DROP TABLE my\_permanent\_table;

5. Create a Table (TABLE1) and load data from multiple sources on hourly basis for 4 - 5 hours. On the 5th hour restore the data to how it was at the end of 1st hour. Clone the TABLE1 to TABLE2 based on how the data was at the end of 2nd hour.

6. Create a stream on Table to log the changes and capture the change logs in a separate Table.

-- Create a table to store the names and fees paid by members of a gym

CREATE OR REPLACE TABLE DEV\_DB.PUBLIC.members (

id number(8) NOT NULL,

name varchar(255) default NULL,

fee number(3) NULL

);

CREATE SCHEMA DEV\_DB.STREAMS;

-- Create a stream to track changes to date in the MEMBERS table

CREATE OR REPLACE STREAM DEV\_DB.STREAMS.member\_check ON TABLE DEV\_DB.PUBLIC.members;

INSERT INTO members (id,name,fee)

VALUES

(1,'Joe',0),

(2,'Jane',0),

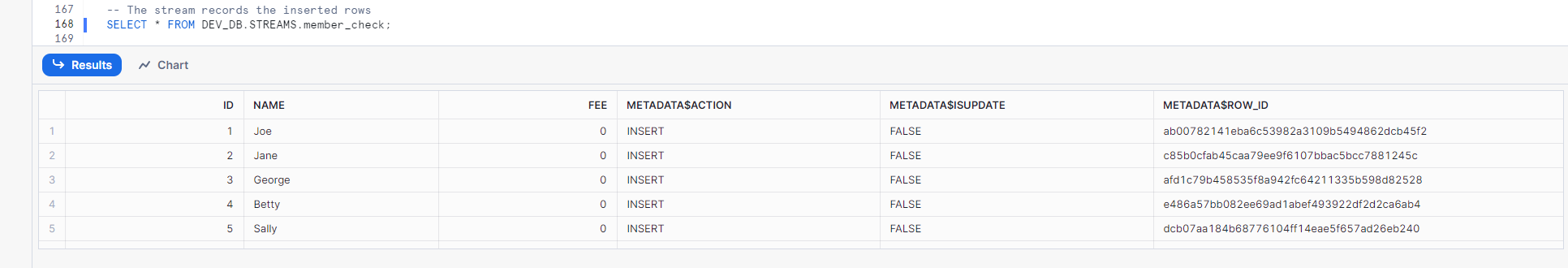
(3,'George',0),

(4,'Betty',0),

(5,'Sally',0);

-- The stream records the inserted rows

SELECT \* FROM DEV\_DB.STREAMS.member\_check;



7. Create an External Table and auto refresh the data using Snowpipe.

CREATE OR REPLACE STORAGE INTEGRATION ACCESS\_TO\_S3BUCKET

TYPE=EXTERNAL\_STAGE

STORAGE\_PROVIDER='S3'

STORAGE\_AWS\_ROLE\_ARN='arn:aws:iam::533267410818:role/S3\_read\_access\_for\_snowflake'

ENABLED=TRUE

STORAGE\_ALLOWED\_LOCATIONS=('s3://lz-snowflake/Snowflake/Employees/');

DESCRIBE STORAGE INTEGRATION ACCESS\_TO\_S3BUCKET;

-- CREATE EXTERNAL STAGE

CREATE OR REPLACE STAGE DEV\_DB.PUBLIC.employees\_stage

url='s3://lz-snowflake/Snowflake/Employees/'

file\_format = (type = 'CSV' field\_delimiter = ',' field\_optionally\_enclosed\_by = '"' skip\_header = 1)

STORAGE\_INTEGRATION = ACCESS\_TO\_S3BUCKET;

DROP TABLE DEV\_DB.PUBLIC.employees;

CREATE OR REPLACE TABLE DEV\_DB.PUBLIC.employees\_snowpipe (

id INT,

first\_name STRING,

last\_name STRING,

email STRING,

location STRING,

department STRING

);

SELECT COUNT(\*) FROM DEV\_DB.PUBLIC.employees\_snowpipe;

COPY INTO DEV\_DB.PUBLIC.employees\_snowpipe FROM @DEV\_DB.PUBLIC.employees\_stage;

LIST @DEV\_DB.PUBLIC.employees\_stage;

CREATE SCHEMA DEV\_DB.PIPES;

// Define pipe

CREATE OR REPLACE pipe DEV\_DB.PIPES.employee\_pipe

auto\_ingest = TRUE

AS

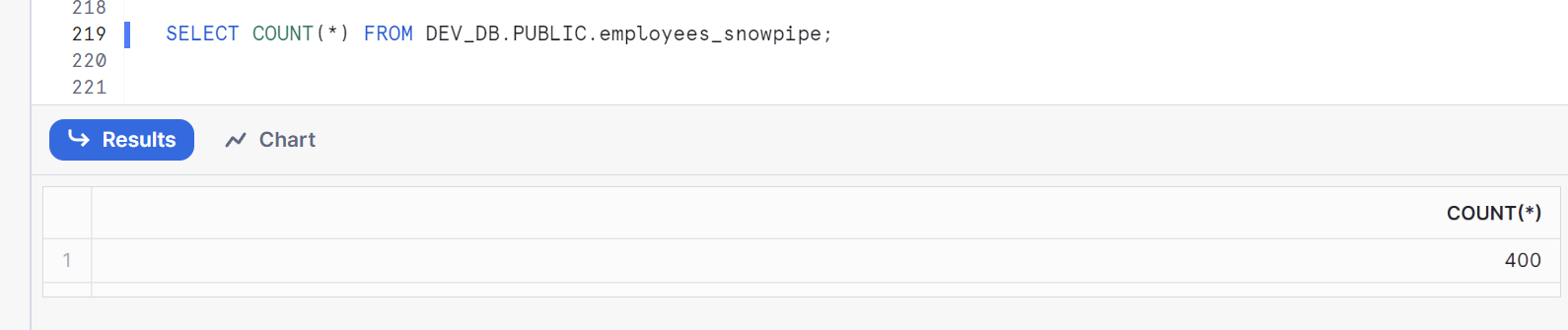
COPY INTO DEV\_DB.PUBLIC.employees\_snowpipe FROM @DEV\_DB.PUBLIC.employees\_stage;

DESCRIBE pipe DEV\_DB.PIPES.employee\_pipe;

-- getting the notification Channel to set up the S3 bucket Notification

-- arn:aws:sqs:us-east-1:851725646910:sf-snowpipe-AIDA4MTWN7Q7DAHEQB3QX-vadq\_r2TdV9BEEYU0QRPIA

SELECT COUNT(\*) FROM DEV\_DB.PUBLIC.employees\_snowpipe;

Before uploading the Data to S3 bucket  
  
After Uploading the Data   


8. Create an External Table and instead of using Snowpipe, use a Stored Procedure and Task to refresh the data.

CREATE OR REPLACE STORAGE INTEGRATION ACCESS\_TO\_S3BUCKET

TYPE=EXTERNAL\_STAGE

STORAGE\_PROVIDER='S3'

STORAGE\_AWS\_ROLE\_ARN='arn:aws:iam::533267410818:role/S3\_read\_access\_for\_snowflake'

ENABLED=TRUE

STORAGE\_ALLOWED\_LOCATIONS=('s3://lz-snowflake/Snowflake/Employees/');

DESCRIBE STORAGE INTEGRATION ACCESS\_TO\_S3BUCKET;

-- CREATE EXTERNAL STAGE

CREATE OR REPLACE STAGE DEV\_DB.PUBLIC.employees\_stage

url='s3://lz-snowflake/Snowflake/Employees/'

file\_format = (type = 'CSV' field\_delimiter = ',' field\_optionally\_enclosed\_by = '"' skip\_header = 1)

STORAGE\_INTEGRATION = ACCESS\_TO\_S3BUCKET;

SELECT COUNT(\*) FROM DEV\_DB.PUBLIC.employees;

DROP STAGE DEV\_DB.PUBLIC.employees\_stage;

CREATE OR REPLACE EXTERNAL TABLE DEV\_DB.PUBLIC.employees (

id INT AS (value:c1::INTEGER),

first\_name STRING AS (value:c2::TEXT),

last\_name STRING AS (value:c3::TEXT),

email STRING AS (value:c4::TEXT),

location STRING AS (value:c5::TEXT),

department STRING AS (value:c6::TEXT)

)

WITH LOCATION = @DEV\_DB.PUBLIC.employees\_stage

FILE\_FORMAT = (TYPE = CSV field\_delimiter = ',' field\_optionally\_enclosed\_by = '"' SKIP\_HEADER = 1);

LIST @DEV\_DB.PUBLIC.employees\_stage;

CREATE OR REPLACE PROCEDURE DEV\_DB.PUBLIC.EMPLOYEES\_DATA\_PROCEDURE ()

RETURNS STRING NOT NULL

LANGUAGE SQL

AS

$$

ALTER EXTERNAL TABLE DEV\_DB.PUBLIC.employees REFRESH;

$$;

CREATE OR REPLACE TASK DEV\_DB.PUBLIC.EMPLOYEES\_DATA\_TASK\_PROCEDURE\_CALL

WAREHOUSE = COMPUTE\_WH

SCHEDULE = 'USING CRON \* \* \* \* \* UTC'

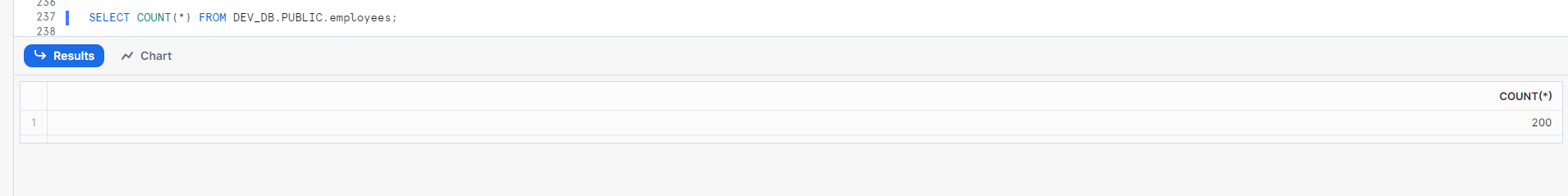
AS CALL DEV\_DB.PUBLIC.EMPLOYEES\_DATA\_PROCEDURE();

ALTER TASK DEV\_DB.PUBLIC.EMPLOYEES\_DATA\_TASK\_PROCEDURE\_CALL resume;

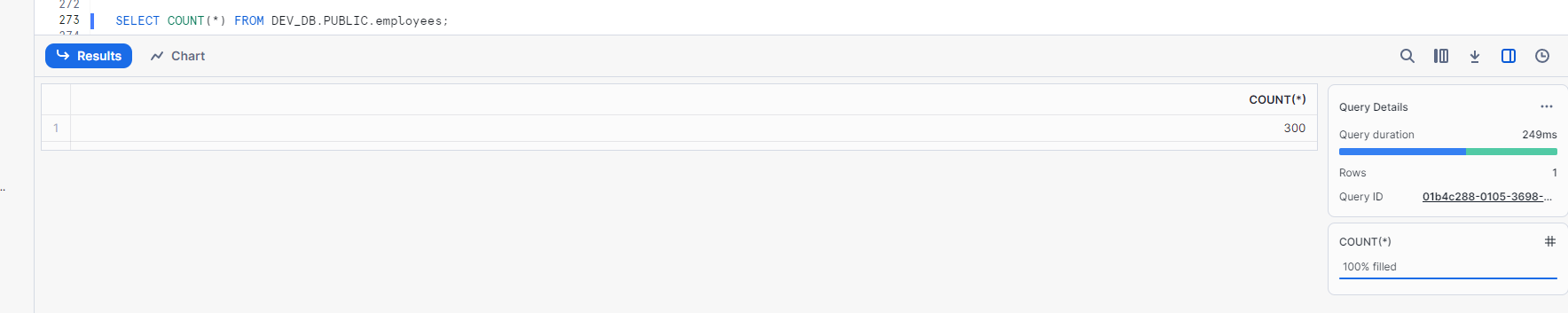
LIST @DEV\_DB.PUBLIC.employees\_stage;

SELECT COUNT(\*) FROM DEV\_DB.PUBLIC.employees;

Before running with the task



After Creating the Task   
  
After Uploading the Data to S3 bucket   
after a minutes



9. Create a Masking policy to mask PII data on an External Table.

GRANT USAGE ON DATABASE DEV\_DB TO ROLE ANALYST\_MASKED;

GRANT ALL ON ALL SCHEMAS IN DATABASE DEV\_DB TO ROLE ANALYST\_MASKED;

GRANT ALL ON ALL TABLES IN DATABASE DEV\_DB TO ROLE ANALYST\_MASKED;

GRANT USAGE ON DATABASE DEV\_DB TO ROLE ANALYST\_FULL;

GRANT ALL ON ALL SCHEMAS IN DATABASE DEV\_DB TO ROLE ANALYST\_FULL;

GRANT ALL ON ALL TABLES IN DATABASE DEV\_DB TO ROLE ANALYST\_FULL;

-- grant warehouse access to roles

GRANT USAGE ON WAREHOUSE COMPUTE\_WH TO ROLE ANALYST\_MASKED;

GRANT USAGE ON WAREHOUSE COMPUTE\_WH TO ROLE ANALYST\_FULL;

-- assign roles to a user

GRANT ROLE ANALYST\_MASKED TO USER MELWINMPK;

GRANT ROLE ANALYST\_FULL TO USER MELWINMPK;

-- Set up masking policy

create or replace masking policy PII

as (val varchar) returns varchar ->

case

when current\_role() in ('ANALYST\_FULL', 'ACCOUNTADMIN') then val

else '##-###-##'

end;

-- Apply policy on a specific column

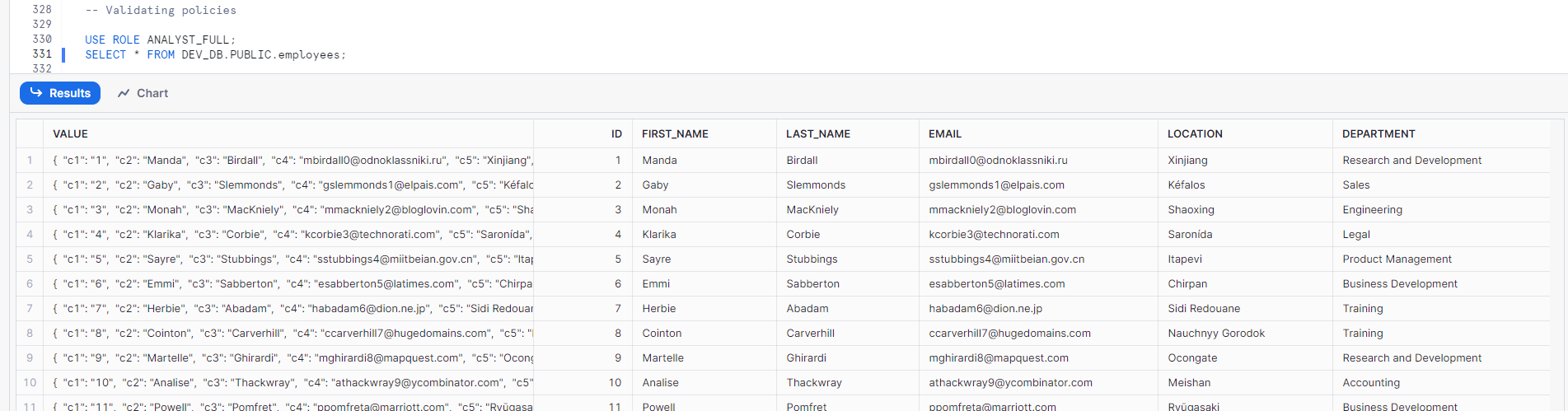
ALTER TABLE IF EXISTS DEV\_DB.PUBLIC.employees MODIFY COLUMN email

SET MASKING POLICY PII;

-- Validating policies

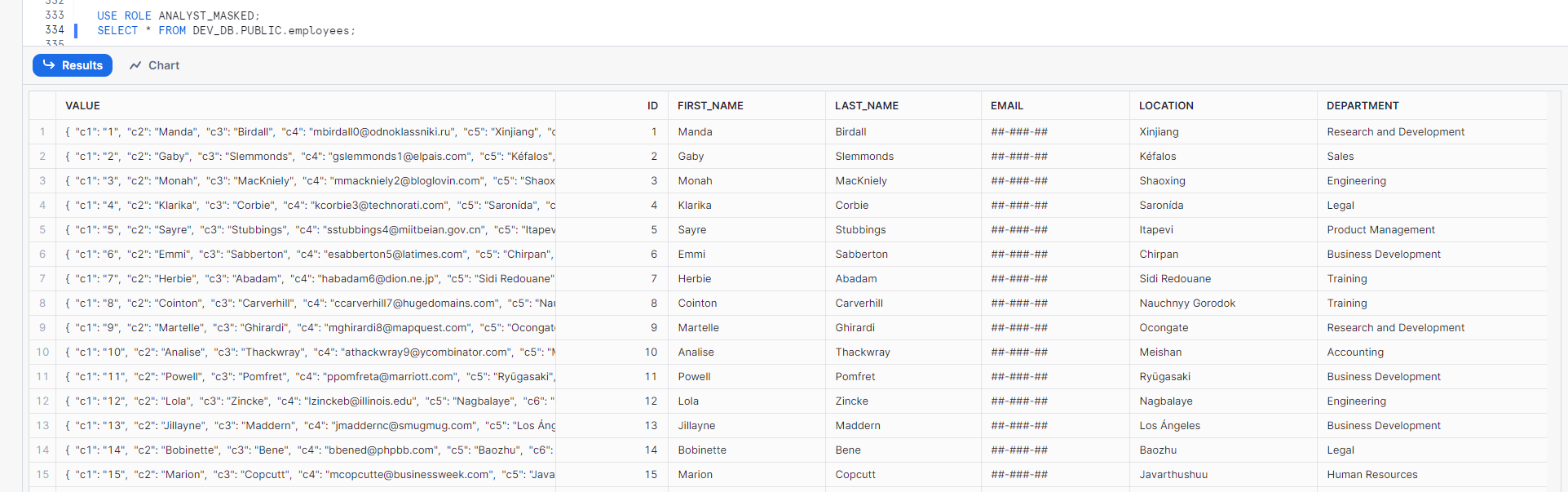
USE ROLE ANALYST\_FULL;

SELECT \* FROM DEV\_DB.PUBLIC.employees;



USE ROLE ANALYST\_MASKED;

SELECT \* FROM DEV\_DB.PUBLIC.employees;



10. Create a Secure view to restrict data based on current Role.

-- Secure view

CREATE secure view prd\_db.Public.secure\_dest as

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF1.CUSTOMER limit 100;

USE ROLE DEV\_ROLE;

SELECT \* FROM prd\_db.Public.secure\_dest limit 10;



11. Create a Materialized view.

-- CREATE OR REPLACE MATERIALIZED VIEW

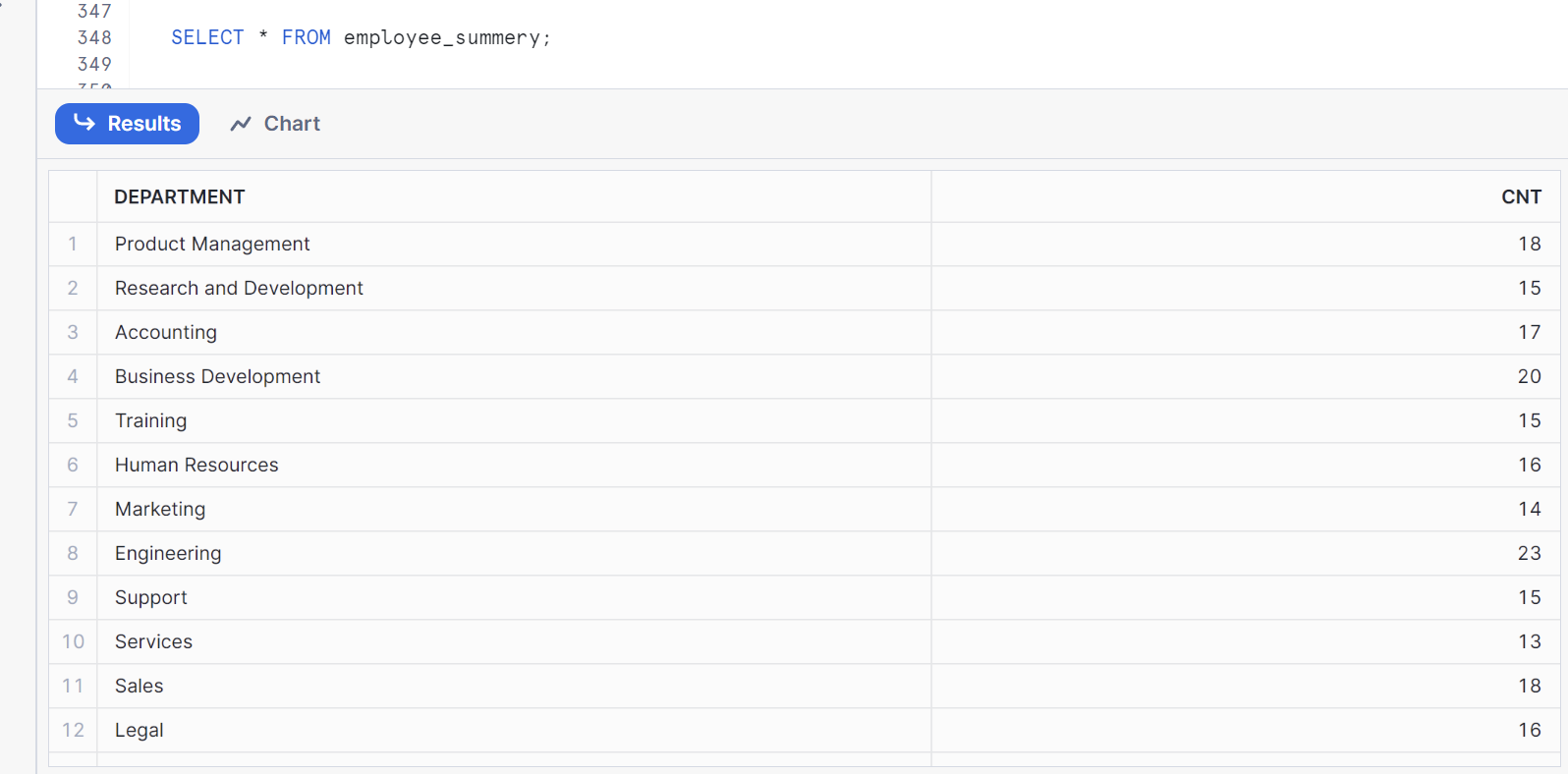
SELECT \* FROM employees;

CREATE OR REPLACE MATERIALIZED VIEW employee\_summery as

SELECT DEPARTMENT, count(\*) as cnt FROM employees

GROUP BY DEPARTMENT;

SELECT \* FROM employee\_summery;



12. Create a Role and provide ReadOnly privileges to PROD\_DB.

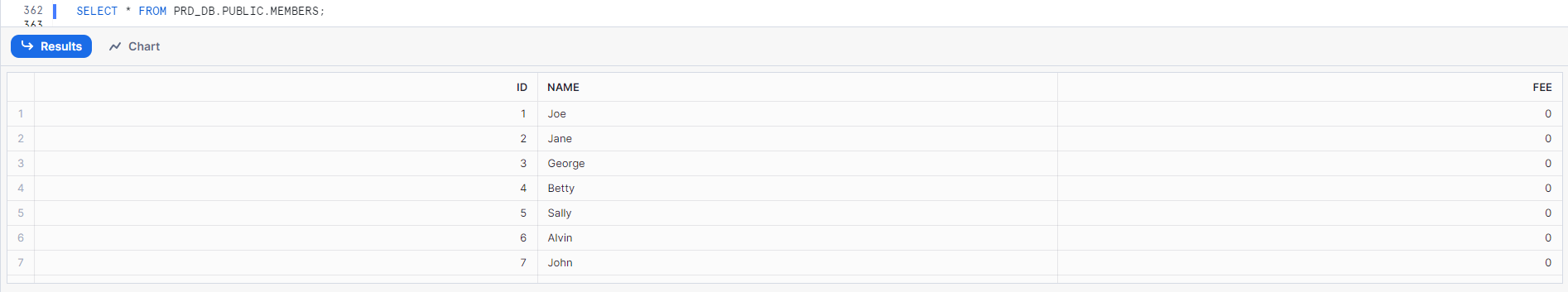
-- Granting access to the Role

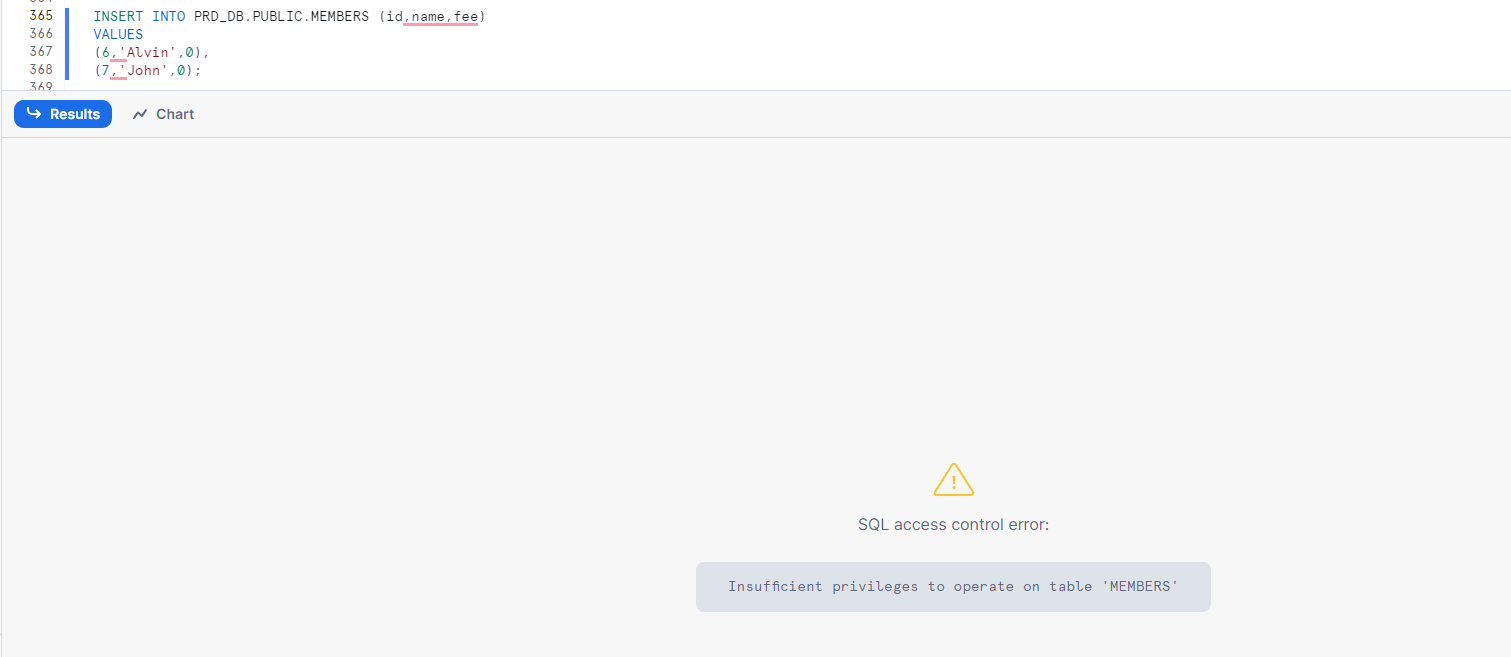
GRANT USAGE ON DATABASE PRD\_DB TO ROLE DEV\_ROLE;

GRANT ALL ON ALL SCHEMAS IN DATABASE PRD\_DB TO ROLE DEV\_ROLE;

GRANT SELECT ON ALL TABLES IN DATABASE PRD\_DB TO ROLE DEV\_ROLE;

USE ROLE DEV\_ROLE;

SELECT \* FROM PRD\_DB.PUBLIC.MEMBERS;  
  




13. Resource Monitor: Create a Resource Monitor to be triggered on consumption of 90 Snowflake credits and assign it to a Warehouse.

CREATE OR REPLACE RESOURCE MONITOR credit\_monitor WITH CREDIT\_QUOTA=90

TRIGGERS ON 100 PERCENT DO SUSPEND;

ALTER WAREHOUSE FIRST\_WAREHOUSE SET RESOURCE\_MONITOR = credit\_monitor;

14. Identify the WH COST and Storage costs associated with your Snowflake Account.

15. Create an Internal and External Stage, load data from S3 to these stages and Insert the data into a Table.

-- Create INTERNAL Stage

CREATE OR REPLACE STAGE DEV\_DB.PUBLIC.amazone\_books\_stage

FILE\_FORMAT = (TYPE= csv FIELD\_DELIMITER = ',' FIELD\_OPTIONALLY\_ENCLOSED\_BY = '"', SKIP\_HEADER=1);

CREATE OR REPLACE TABLE DEV\_DB.PUBLIC.amazone\_books (

BOOK\_ID NUMBER(38,0) NOT NULL,

BOOK\_TITLE VARCHAR(16777216),

BOOK\_AMOUNT FLOAT,

BOOK\_AUTHOR VARCHAR(16777216),

BOOK\_RATING FLOAT,

BOOK\_LINK VARCHAR(16777216),

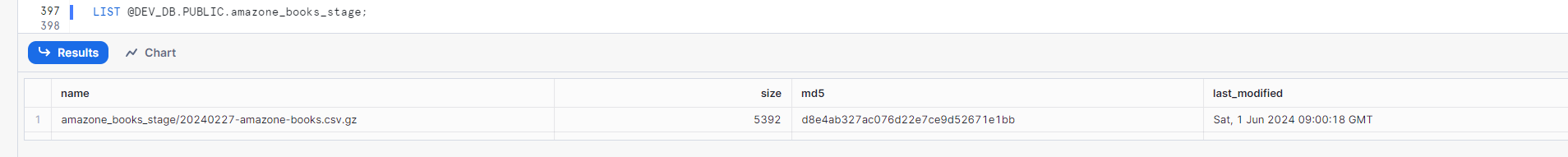
BUSINESS\_DATE DATE

);

-- load the Data using the python connector snowflake.connector

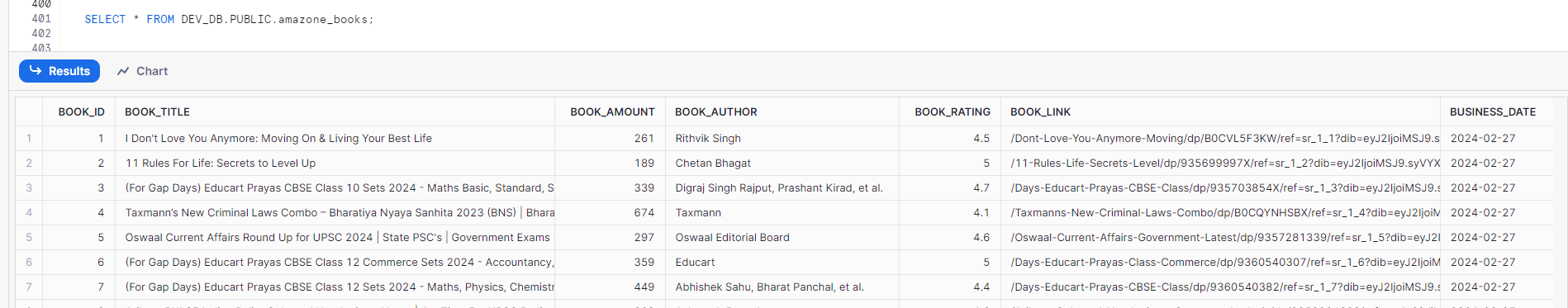
SELECT $1,$2,$3,$4,$5,$6,$7 FROM @DEV\_DB.PUBLIC.amazone\_books\_stage;

LIST @DEV\_DB.PUBLIC.amazone\_books\_stage;



COPY INTO DEV\_DB.PUBLIC.amazone\_books FROM @DEV\_DB.PUBLIC.amazone\_books\_stage;

SELECT \* FROM DEV\_DB.PUBLIC.amazone\_books;



-- Create external Stage

CREATE OR REPLACE STORAGE INTEGRATION ACCESS\_TO\_S3BUCKET\_AWS

TYPE=EXTERNAL\_STAGE

STORAGE\_PROVIDER='S3'

STORAGE\_AWS\_ROLE\_ARN='arn:aws:iam::533267410818:role/S3\_read\_access\_for\_snowflake'

ENABLED=TRUE

STORAGE\_ALLOWED\_LOCATIONS=('s3://lz-snowflake/Snowflake/');

CREATE OR REPLACE STAGE DEV\_DB.PUBLIC.amazone\_books\_ext\_stage

url='s3://lz-snowflake/Snowflake/amazonebooks\_external/amazone\_books/'

file\_format = (type = 'CSV' field\_delimiter = ',' field\_optionally\_enclosed\_by = '"' skip\_header = 1)

STORAGE\_INTEGRATION = ACCESS\_TO\_S3BUCKET\_AWS;

-- CREATE EXTERNAL TABLE

CREATE OR REPLACE EXTERNAL TABLE DEV\_DB.PUBLIC.AMAZONE\_BOOKS\_EXT(

BOOK\_ID NUMBER(38, 0) AS (value:c1::INTEGER)

,BOOK\_TITLE VARCHAR(16777216) AS (value:c2::TEXT)

,BOOK\_AMOUNT FLOAT AS (value:c3::FLOAT)

,BOOK\_AUTHOR VARCHAR(16777216) AS (value:c4::TEXT)

,BOOK\_RATING FLOAT AS (value:c5::float)

,BOOK\_LINK VARCHAR(16777216) AS (value:c6::TEXT)

,BUSINESS\_DATE DATE AS TO\_DATE(value:c7::VARCHAR,'YYYY-MM-DD')

,FILE\_DATE\_PARTITION NUMBER(10,0) as (split\_part(METADATA$FILENAME,'/',4)::int)

)

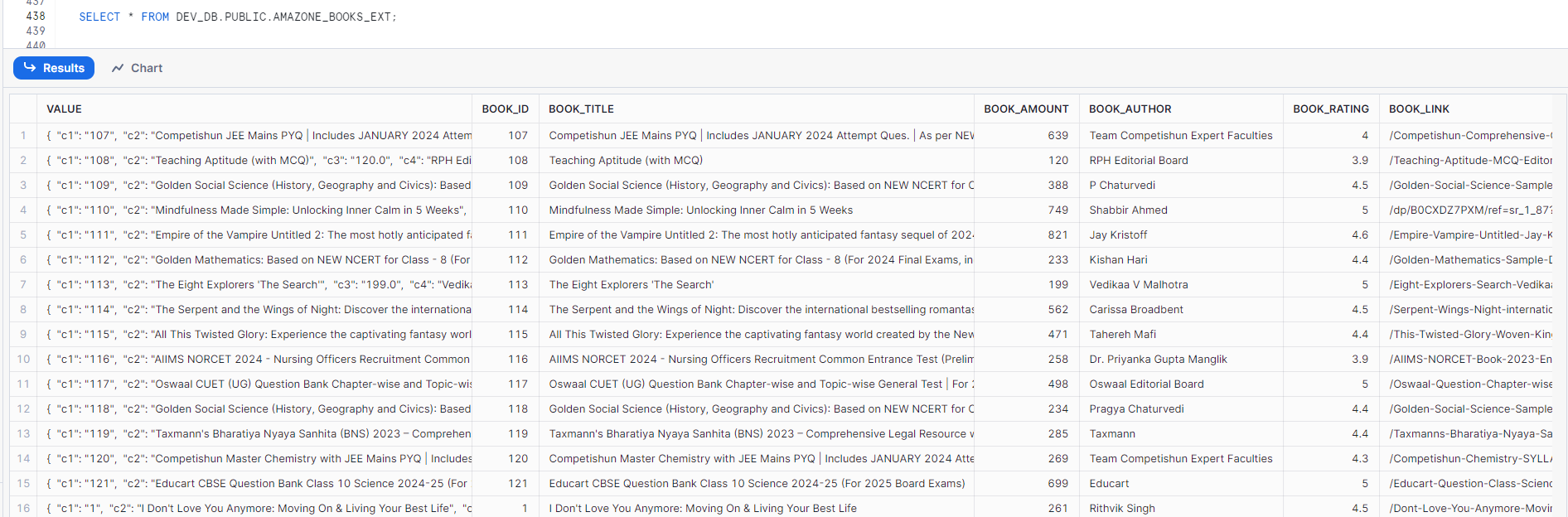
PARTITION BY (FILE\_DATE\_PARTITION)

WITH LOCATION = @amazone\_books\_ext\_stage

FILE\_FORMAT = (TYPE = CSV field\_delimiter = ',' field\_optionally\_enclosed\_by = '"' SKIP\_HEADER = 1);

-- ALTER EXTERNAL TABLE IF EXISTS DEV\_DB.PUBLIC.amazone\_books\_ext\_stage ADD FILES ('20240227/20240227-amazone-books.csv');

SELECT \* FROM DEV\_DB.PUBLIC.AMAZONE\_BOOKS\_EXT;



16. Create a stored procedure, switch role inside the stored and drop a Table.

E.g. Assume Stored Procedure SP1 is executed using ROLE1, Switch to ROLE2 inside the Stored Procedure and drop a Table using ROLE2.

CREATE OR REPLACE PROCEDURE DEV\_DB.PUBLIC.DROP\_TABLE()

RETURNS STRING NOT NULL

LANGUAGE SQL

EXECUTE AS OWNER

AS

$$

DROP TABLE DEV\_DB.PUBLIC.members;

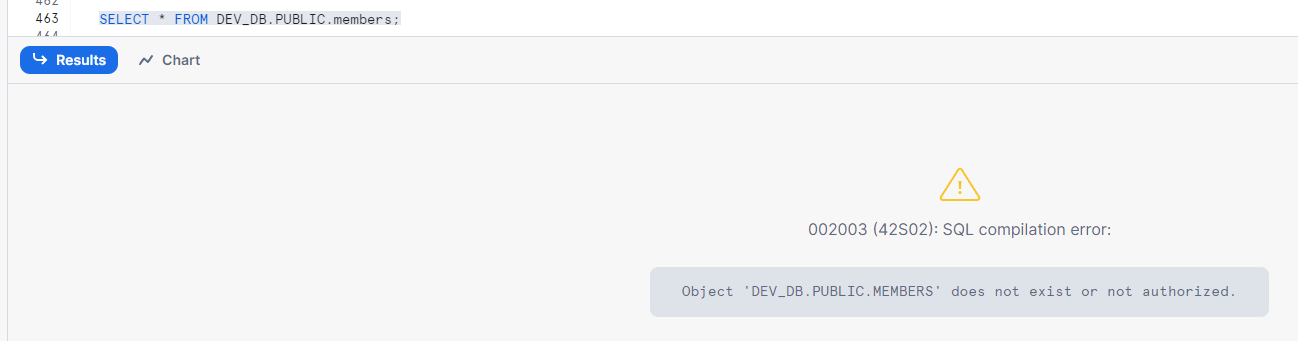
$$;

GRANT ALL ON ALL PROCEDURES IN DATABASE DEV\_DB TO ROLE DEV\_ROLE;

USE ROLE DEV\_ROLE;

CALL DEV\_DB.PUBLIC.DROP\_TABLE();

SELECT \* FROM DEV\_DB.PUBLIC.members;



17. Identify the Queries executed on DEV\_DB and which User\Role has executed most Queries.

SELECT

QUERY\_TEXT

FROM

SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY

WHERE DATABASE\_NAME = 'DEV\_DB' limit 10;



SELECT

USER\_NAME, ROLE\_NAME, COUNT(\*) as Count

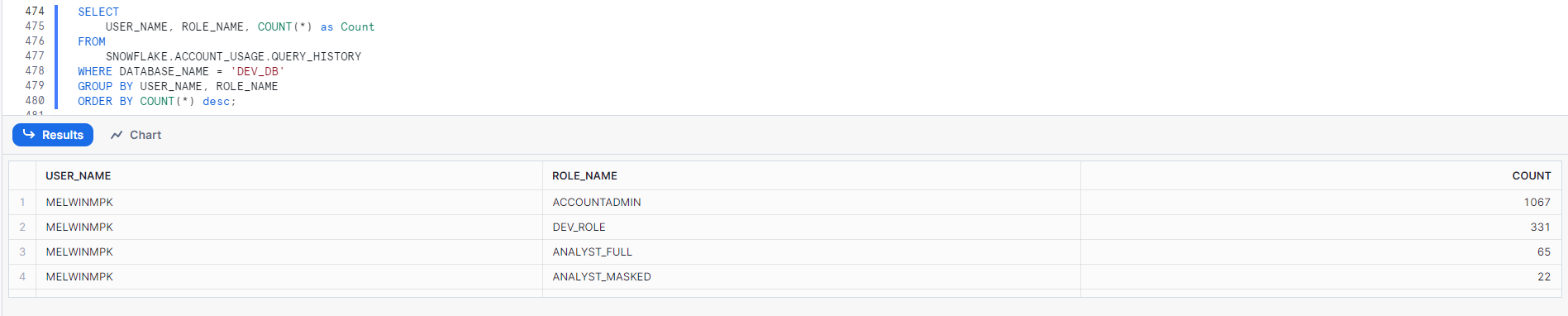
FROM

SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY

WHERE DATABASE\_NAME = 'DEV\_DB'

GROUP BY USER\_NAME, ROLE\_NAME

ORDER BY COUNT(\*) desc;



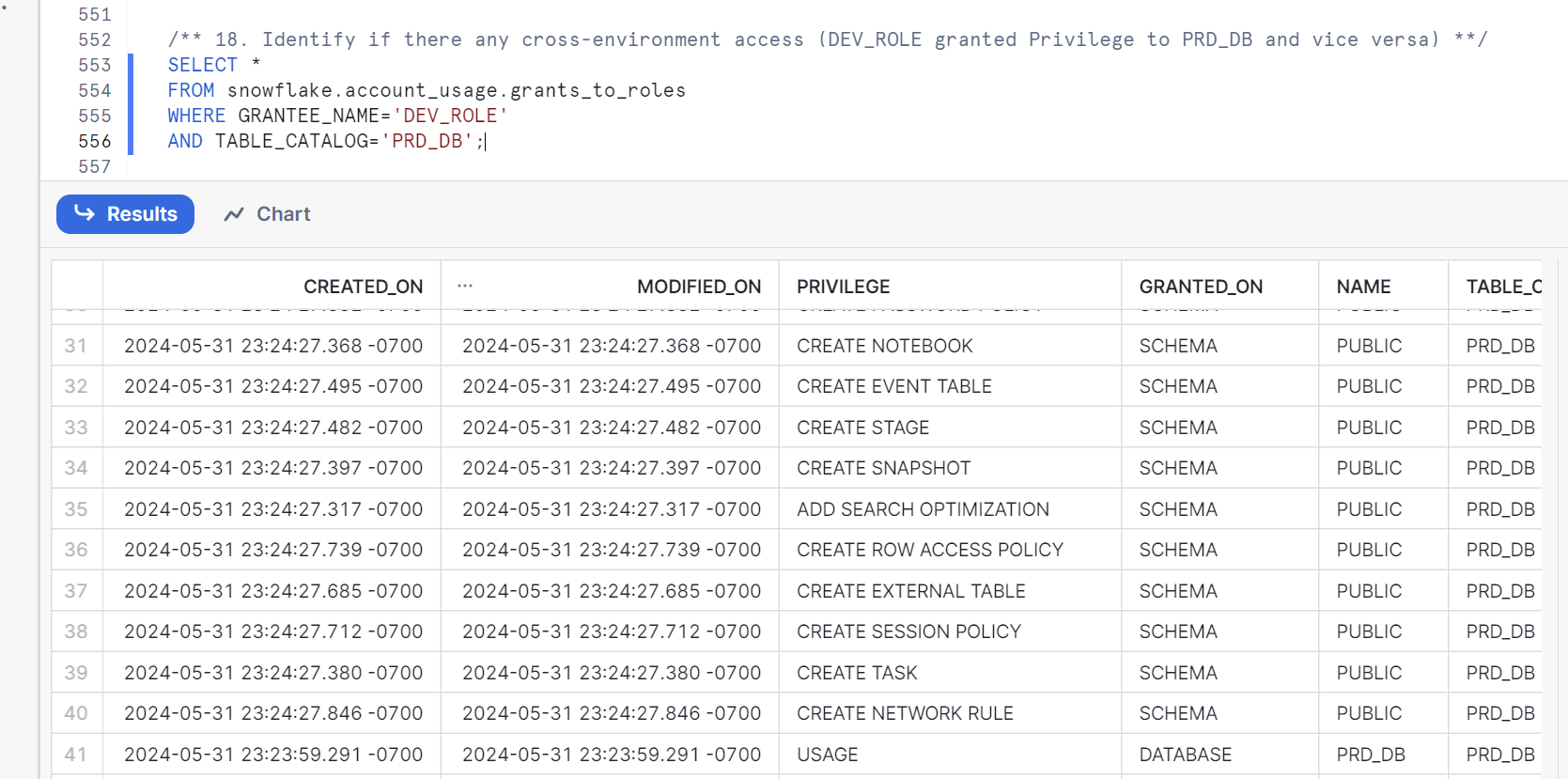
18. Identify if there any cross-environment access (DEV\_ROLE granted Privilege to PRD\_DB and vice versa)

SELECT \*

FROM snowflake.account\_usage.grants\_to\_roles

WHERE GRANTEE\_NAME='DEV\_ROLE'

AND TABLE\_CATALOG='PRD\_DB';

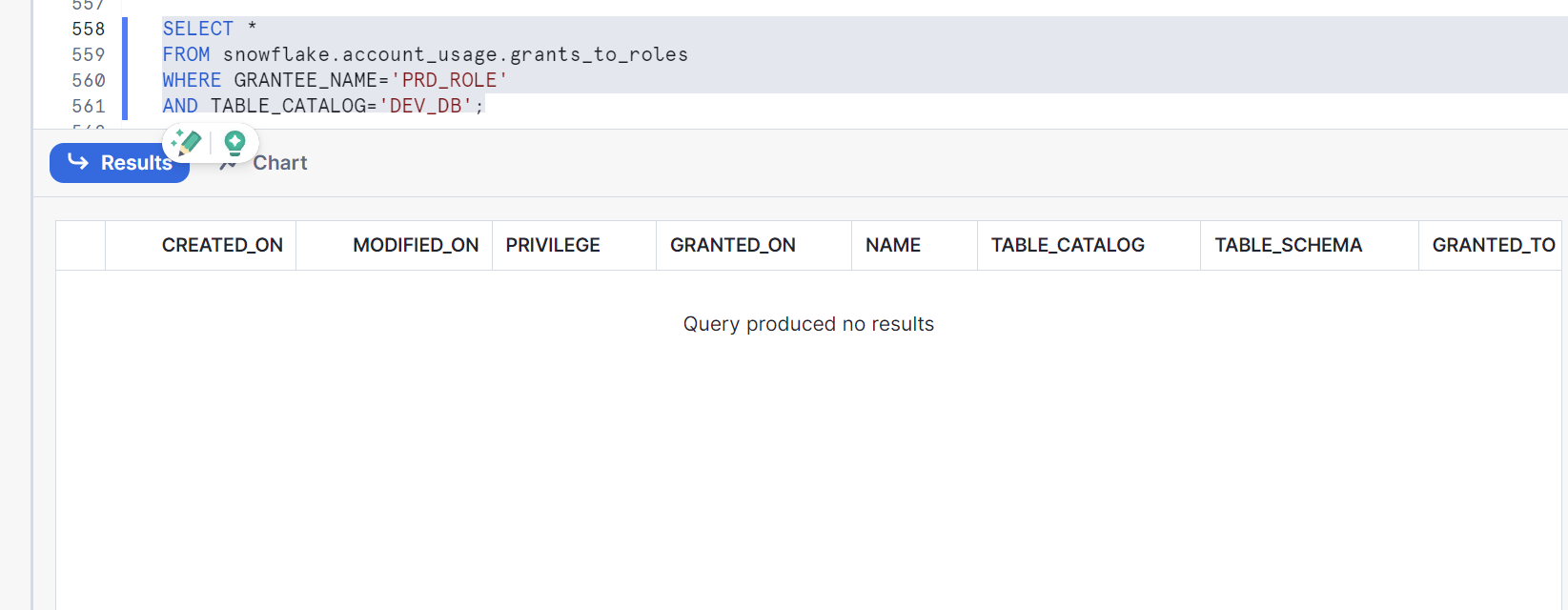


SELECT \*

FROM snowflake.account\_usage.grants\_to\_roles

WHERE GRANTEE\_NAME='PRD\_ROLE'

AND TABLE\_CATALOG='DEV\_DB';



19. Set up Future Grants on DEV\_DB so that once a Table is created, Select Privilege is provided to DEV\_ROLE2.

CREATE OR REPLACE ROLE DEV\_ROLE2;

-- Granting access to the Role

GRANT USAGE ON DATABASE DEV\_DB TO ROLE DEV\_ROLE2;

GRANT ALL ON ALL SCHEMAS IN DATABASE DEV\_DB TO ROLE DEV\_ROLE2;

GRANT ALL ON ALL TABLES IN DATABASE DEV\_DB TO ROLE DEV\_ROLE2;

GRANT ALL ON FUTURE TABLES IN DATABASE DEV\_DB TO ROLE DEV\_ROLE2;

20. Create a Task that’s executed on daily basis at 9.30 AM IST.

CREATE OR REPLACE TASK DEV\_DB.PUBLIC.my\_daily\_task

WAREHOUSE = COMPUTE\_WH

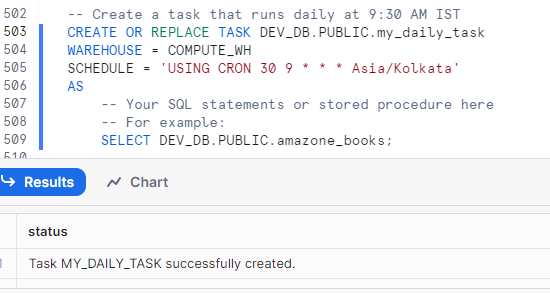
SCHEDULE = 'USING CRON 30 9 \* \* \* Asia/Kolkata'

AS

-- Your SQL statements or stored procedure here

-- For example:

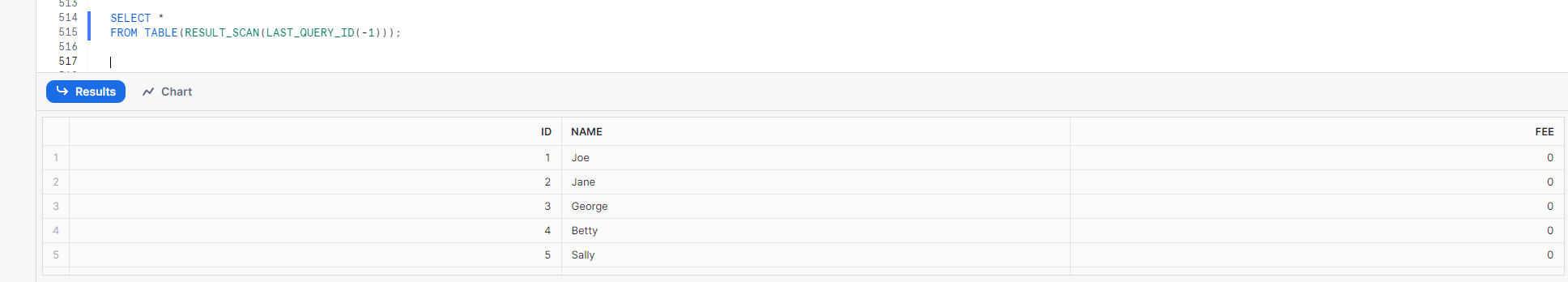
SELECT DEV\_DB.PUBLIC.amazone\_books;



21. Create a Schema with Managed Access and Grant access to Objects within the schema using a role other than SYSADMIN & Schema’s Owner.

22. Retrieve data from a previously executed Query using Result\_scan function.

SELECT \* FROM TABLE(RESULT\_SCAN(LAST\_QUERY\_ID(-1)));



23. Create a Share and provide access to specific Tables in the schema to the share.

CREATE SHARE my\_share;

GRANT usage on database dev\_db TO SHARE my\_share;

GRANT usage on schema dev\_db.public TO SHARE my\_share;

GRANT SELECT ON TABLE public.destination\_table TO SHARE my\_share;

GRANT SELECT ON TABLE public.emp\_tb TO SHARE my\_share;

24. Create a Share and share the entire schema (all Tables, Views, etc.) to the share.

CREATE SHARE my\_schema\_share;

GRANT USAGE ON DATABASE DEV\_DB TO SHARE my\_schema\_share;

GRANT USAGE ON SCHEMA DEV\_DB.PUBLIC TO SHARE my\_schema\_share;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA DEV\_DB.PUBLIC TO SHARE my\_schema\_share;

GRANT USAGE ON ALL SEQUENCES IN SCHEMA DEV\_DB.PUBLIC TO SHARE my\_schema\_share;

GRANT USAGE ON ALL STREAMS IN SCHEMA DEV\_DB.PUBLIC TO SHARE my\_schema\_share;

GRANT USAGE ON ALL TASKS IN SCHEMA DEV\_DB.PUBLIC TO SHARE my\_schema\_share;

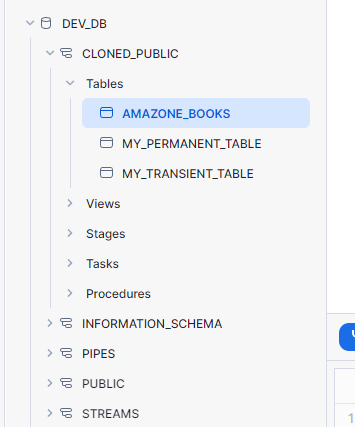
GRANT USAGE ON ALL VIEWS IN SCHEMA DEV\_DB.PUBLIC TO SHARE my\_schema\_share;

GRANT USAGE ON ALL MATERIALIZED VIEWS IN SCHEMA DEV\_DB.PUBLIC TO SHARE my\_schema\_share;

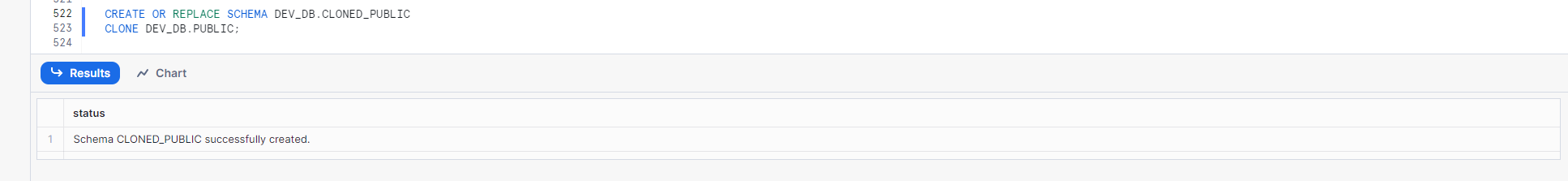
25. Clone a Schema with External Tables.

CREATE OR REPLACE SCHEMA DEV\_DB.CLONED\_PUBLIC

CLONE DEV\_DB.PUBLIC;



SELECT \* FROM DEV\_DB.CLONED\_PUBLIC.AMAZONE\_BOOKS limit 10;



-- External table cannot be cloned

-- we need to recreate it from the stage (as the respective stage is cloned)

-- CREATE EXTERNAL TABLE

CREATE OR REPLACE EXTERNAL TABLE DEV\_DB.CLONED\_PUBLIC.AMAZONE\_BOOKS\_EXT(

BOOK\_ID NUMBER(38, 0) AS (value:c1::INTEGER)

,BOOK\_TITLE VARCHAR(16777216) AS (value:c2::TEXT)

,BOOK\_AMOUNT FLOAT AS (value:c3::FLOAT)

,BOOK\_AUTHOR VARCHAR(16777216) AS (value:c4::TEXT)

,BOOK\_RATING FLOAT AS (value:c5::float)

,BOOK\_LINK VARCHAR(16777216) AS (value:c6::TEXT)

,BUSINESS\_DATE DATE AS TO\_DATE(value:c7::VARCHAR,'YYYY-MM-DD')

,FILE\_DATE\_PARTITION NUMBER(10,0) as (split\_part(METADATA$FILENAME,'/',4)::int)

)

PARTITION BY (FILE\_DATE\_PARTITION)

WITH LOCATION = @DEV\_DB.CLONED\_PUBLIC.amazone\_books\_ext\_stage

FILE\_FORMAT = (TYPE = CSV field\_delimiter = ',' field\_optionally\_enclosed\_by = '"' SKIP\_HEADER = 1);

SELECT \* FROM DEV\_DB.CLONED\_PUBLIC.AMAZONE\_BOOKS\_EXT limit 10;

