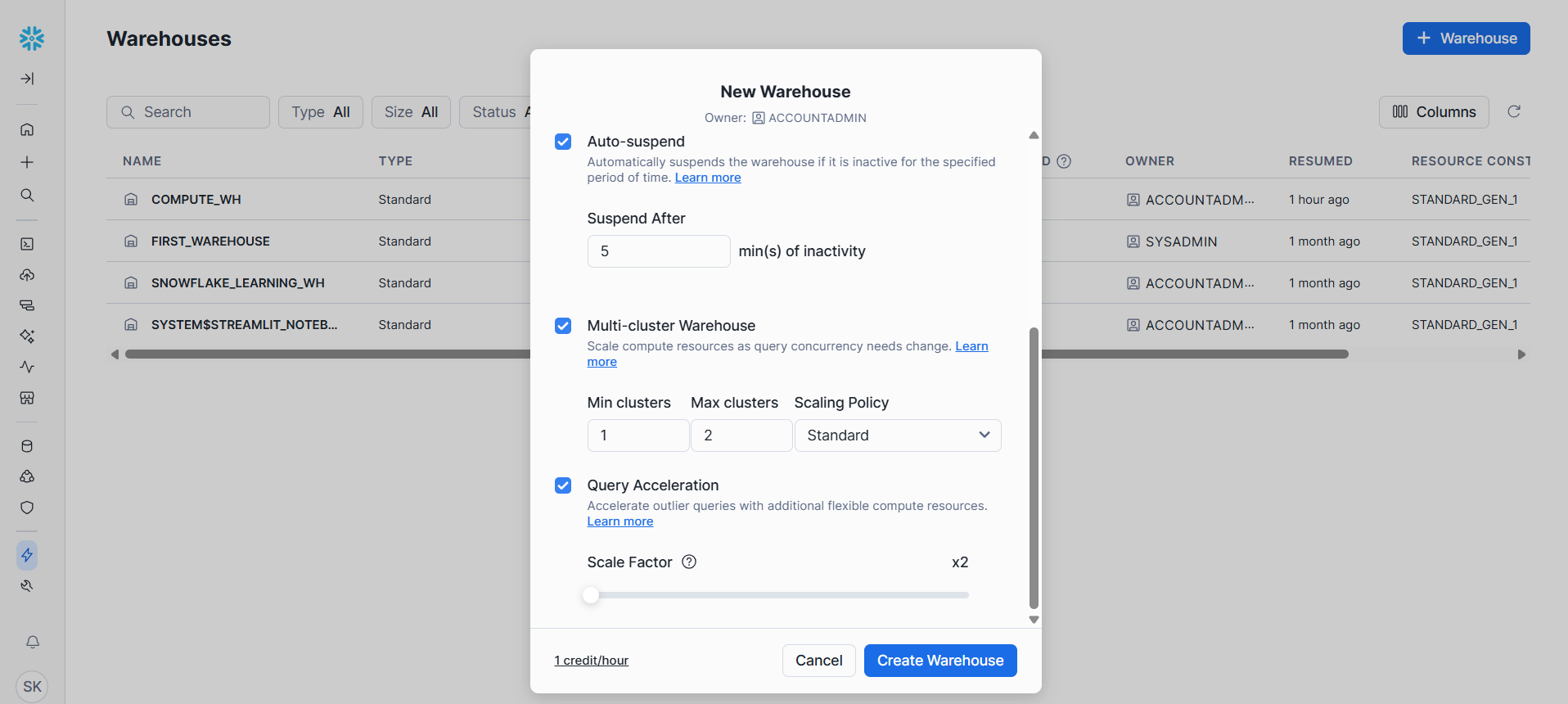
**Warehouse creation:**

**In Web UI**





**Using Snowsight**

****

**Example:**

**CREATE OR REPLACE WAREHOUSE SAI\_WH**

**WITH**

**WAREHOUSE\_SIZE = XSMALL**

**MIN\_CLUSTER\_COUNT = 1**

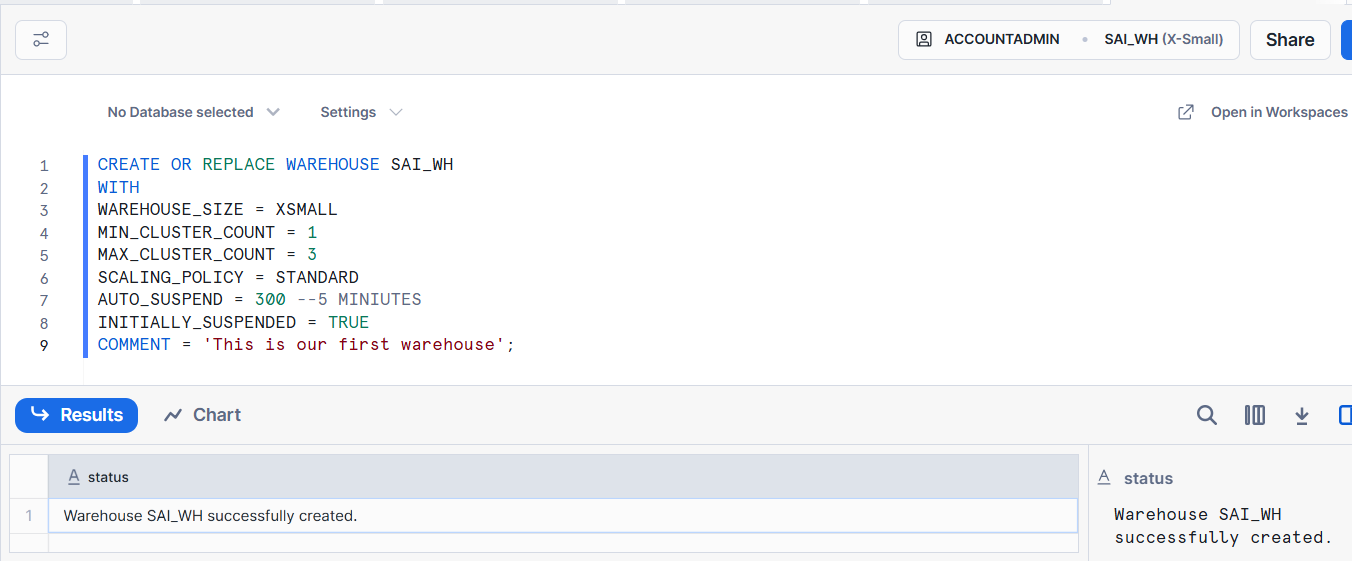
**MAX\_CLUSTER\_COUNT = 3**

**SCALING\_POLICY = STANDARD**

**AUTO\_SUSPEND = 300 --5 MINIUTES**

**INITIALLY\_SUSPENDED = TRUE**

**COMMENT = 'This is our first warehouse';**

****

**Data Types:**

| Data Type | Description | Example |
| --- | --- | --- |
| NUMBER / DECIMAL / NUMERIC | Fixed-point decimal numbers (can specify precision and scale). | NUMBER(10,2) → 12345.67 |
| INT / INTEGER / BIGINT / SMALLINT / TINYINT / BYTEINT | Integer numbers (all are stored as NUMBER). | 100 |
| FLOAT / FLOAT4 / FLOAT8 / DOUBLE / DOUBLE PRECISION / REAL | Floating-point numbers. | 123.4567 |
| VARCHAR / CHAR / CHARACTER / STRING / TEXT | Variable-length string. Default length is unlimited. | 'Hello' |
| CHAR / CHARACTER | Fixed-length string (padded with spaces). | 'A ' |
| BINARY / VARBINARY | Binary strings (used for files, encrypted values, etc.). | 0xDEADBEEF |
| DATE | Stores calendar date (no time). | 2025-10-14 |
| TIME | Stores time of day (no date). | 12:30:45 |
| TIMESTAMP | Alias for TIMESTAMP\_NTZ. | 2025-10-14 12:30:45 |
| TIMESTAMP\_NTZ | Timestamp without time zone. | 2025-10-14 12:30:45 |
| TIMESTAMP\_LTZ | Timestamp with local time zone. | 2025-10-14 12:30:45 |
| TIMESTAMP\_TZ | Timestamp with explicit time zone. | 2025-10-14 12:30:45 +05:30 |
| BOOLEAN | True/False values | TRUE, FALSE |
| VARIANT | Can store JSON, Avro, ORC, Parquet, XML, etc. | {'name': 'Sai', 'age': 30} |
| OBJECT | Key-value pairs (similar to JSON objects). | {'a':1, 'b':2} |
| ARRAY | Ordered list of elements. | [1,2,3,4] |
| GEOGRAPHY | Stores geospatial data in WGS 84 standard. | 'POINT(-122.35 37.55)' |

**Stages:**

In Snowflake, stages are locations where data files are stored before being loaded into tables or after being unloaded from tables.  
They act as intermediate storage areas for data ingestion and extraction.

| **Feature / Property** | **User Stage** | **Table Stage** | **Named Stage** |
| --- | --- | --- | --- |
| 🏗 **Creation** | Automatically created for each user | Automatically created for each table | Manually created by user |
| 👥 **Scope / Ownership** | Specific to each user | Specific to a single table | Shared — can be used by multiple users/tables |
| 🗂 **Usage Purpose** | Temporary, personal file staging | Staging files linked to one table | Centralized or external data staging |
| 📦 **File Location** | Internal only (Snowflake-managed) | Internal only (Snowflake-managed) | Internal or external (e.g., S3, Azure, GCS) |
| 🔐 **Access Control** | Only the owner (user) can access | Only table owner / granted roles | Full RBAC (can grant privileges) |
| 🧭 **Reference Syntax** | @~ | @%table\_name | @stage\_name |
| ⬆️ **PUT/GET Support** | Yes | Yes | Yes (internal), not needed for external |
| 🧾 **File Persistence** | Temporary | Temporary | Persistent (managed manually) |
| 🌐 **Integration with External Storage** | ❌ No | ❌ No | ✅ Yes |
| 💡 **Typical Use Case** | Testing, quick uploads | One-table loading | Production pipelines, shared data, external data |

**Steps to create stage:**

//first create database

CREATE OR REPLACE DATABASE FIRST\_DB;

//use the database

USE DATABASE FIRST\_DB;

//create stage schema to store stages

CREATE OR REPLACE SCHEMA external\_stages;

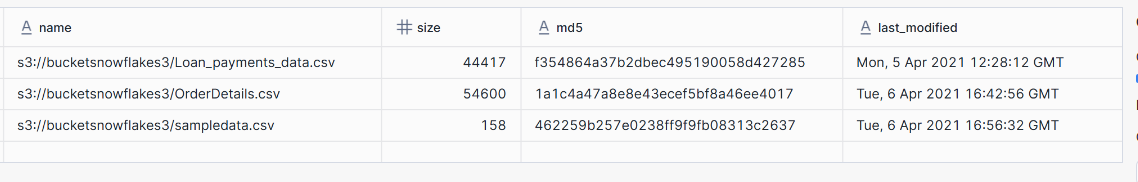
// creating stage object from publicly available s3 bucket

CREATE OR REPLACE STAGE FIRST\_DB.external\_stages.aws\_stage

url='s3://bucketsnowflakes3';

// List files in stage

LIST @FIRST\_DB.EXTERNAL\_STAGES.aws\_stage;



// Creating ORDERS table to load data from staging area

CREATE OR REPLACE TABLE FIRST\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Copy command with pattern for file names

COPY INTO FIRST\_DB.PUBLIC.ORDERS

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

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

pattern='.\*Order.\*';



**Example to load only specific columns:**

//creating table

CREATE OR REPLACE TABLE FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT

);

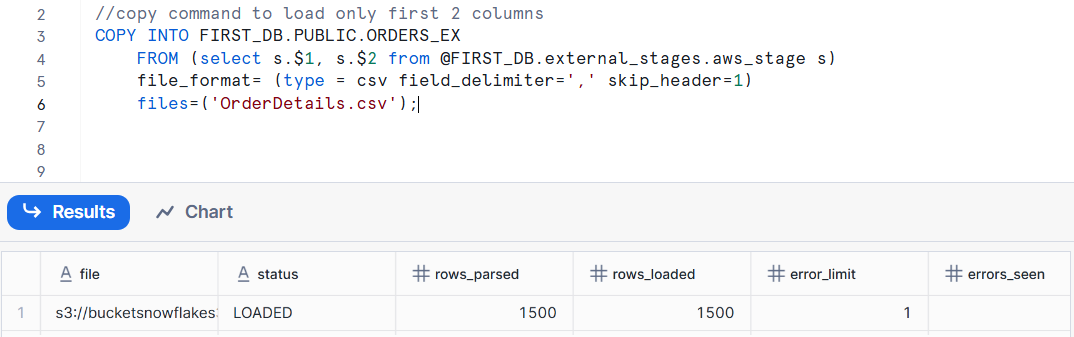
//copy command to load only first 2 columns

COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

FROM (select s.$1, s.$2 from @FIRST\_DB.external\_stages.aws\_stage s)

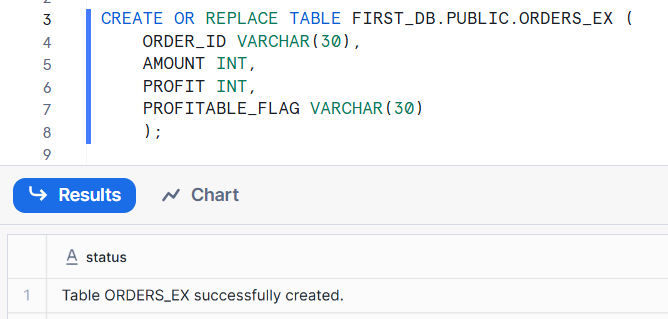
file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files=('OrderDetails.csv');





**Example 2 to load only specific columns:**



COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX (ORDER\_ID,PROFIT)

FROM (select

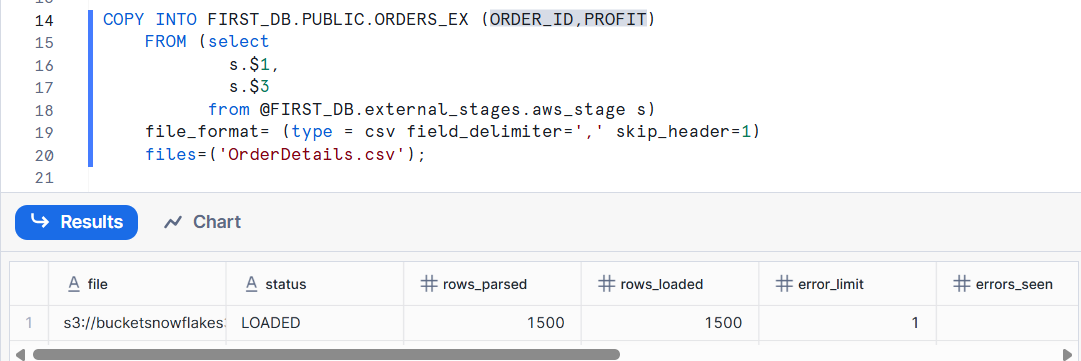
s.$1,

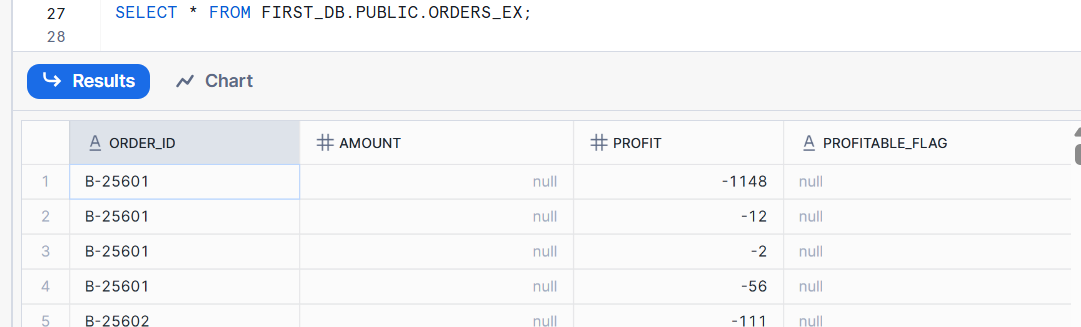
s.$3

from @FIRST\_DB.external\_stages.aws\_stage s)

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

files=('OrderDetails.csv');





**Copy Command: ON\_ERROR options**

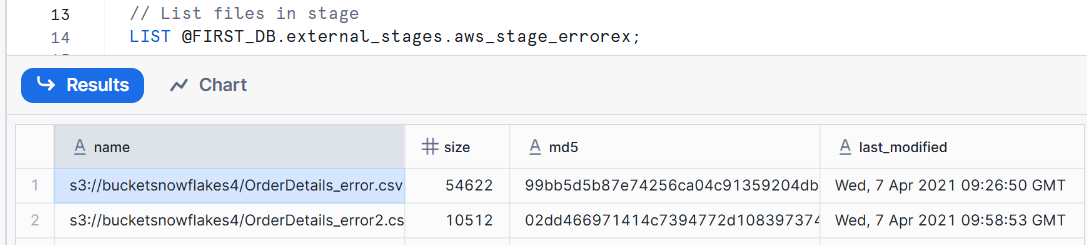
| Option | Behavior | When to Use |
| --- | --- | --- |
| ABORT\_STATEMENT *(default)* | Stops the entire load if any error occurs. | When you want strict data quality. |
| CONTINUE | Loads valid records and skips the bad ones. | When you can tolerate some bad records. |
| SKIP\_FILE | Skips the entire file if any error occurs in that file. | When you want to skip problematic files entirely. |
| SKIP\_FILE\_<n> | Skips the file if the number of errors in that file exceeds n. | When some errors are okay, but not too many. |
| SKIP\_FILE\_<n>\_PERCENT | Skips the file if the error rate exceeds n%. | Useful for large files where a few errors are acceptable. |

**Example : CONTINUE**

**// Create new stage**

CREATE OR REPLACE STAGE FIRST\_DB.external\_stages.aws\_stage\_errorex

url='s3://bucketsnowflakes4';



// Create example table

CREATE OR REPLACE TABLE FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Error handling using the ON\_ERROR CONTINUE option

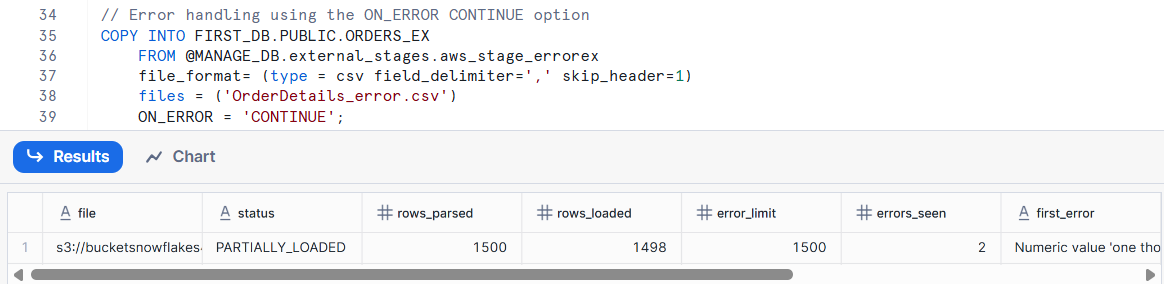
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

FROM @MANAGE\_DB.external\_stages.aws\_stage\_errorex

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

files = ('OrderDetails\_error.csv')

ON\_ERROR = 'CONTINUE';



**Example : ABORT\_STATEMENT**

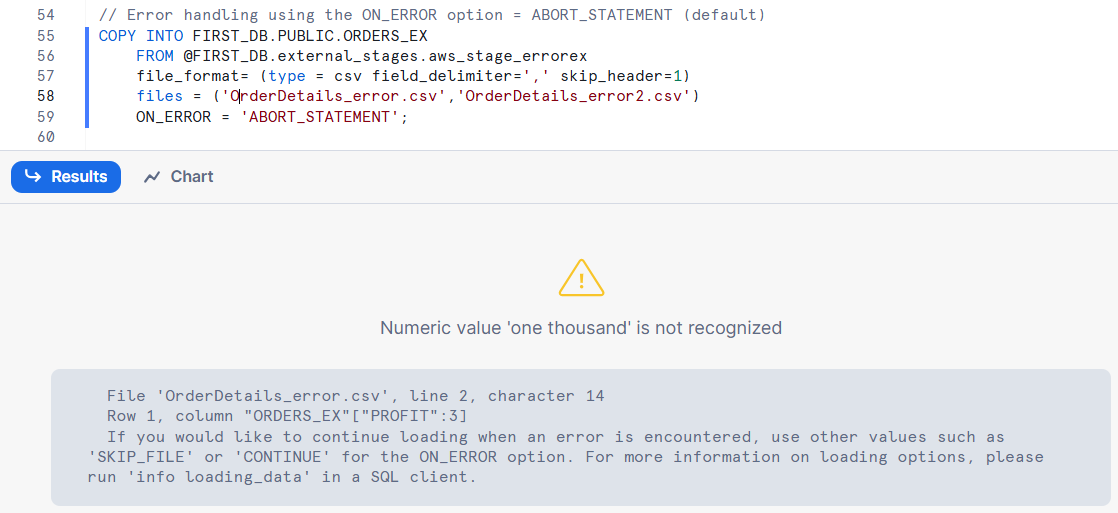
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

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

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

files = ('OrderDetails\_error.csv','OrderDetails\_error2.csv')

ON\_ERROR = 'ABORT\_STATEMENT';



**Example : SKIP\_FILE**

// Error handling using the ON\_ERROR option = SKIP\_FILE

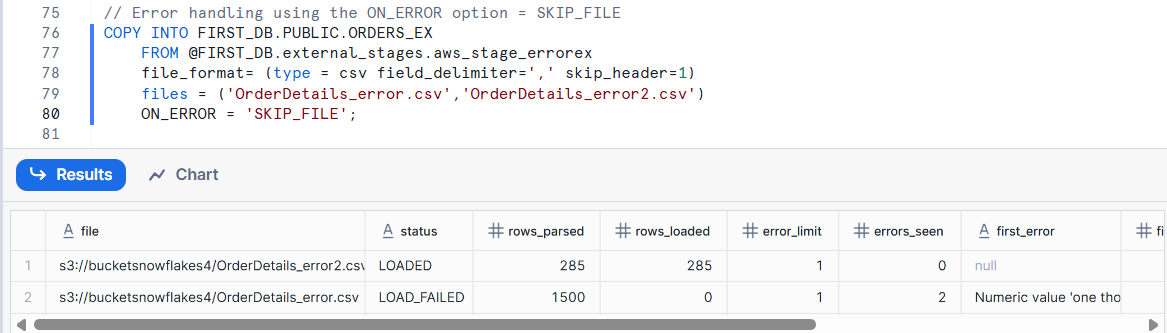
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

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

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

files = ('OrderDetails\_error.csv','OrderDetails\_error2.csv')

ON\_ERROR = 'SKIP\_FILE';



**Example : SKIP\_FILE\_<number>**

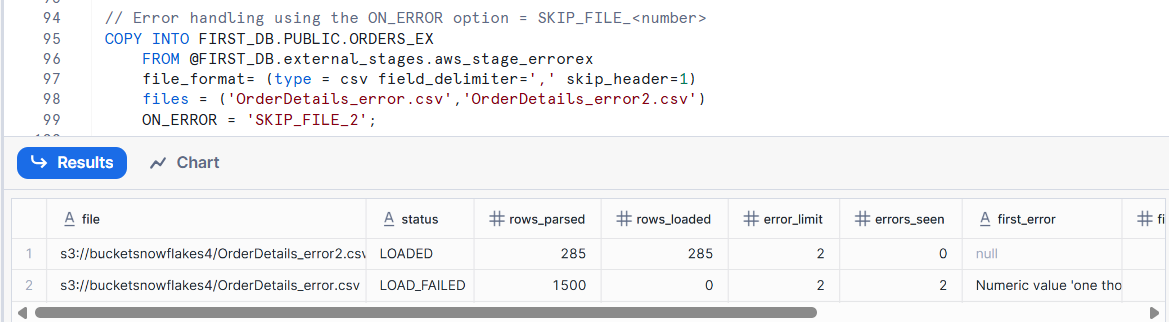
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

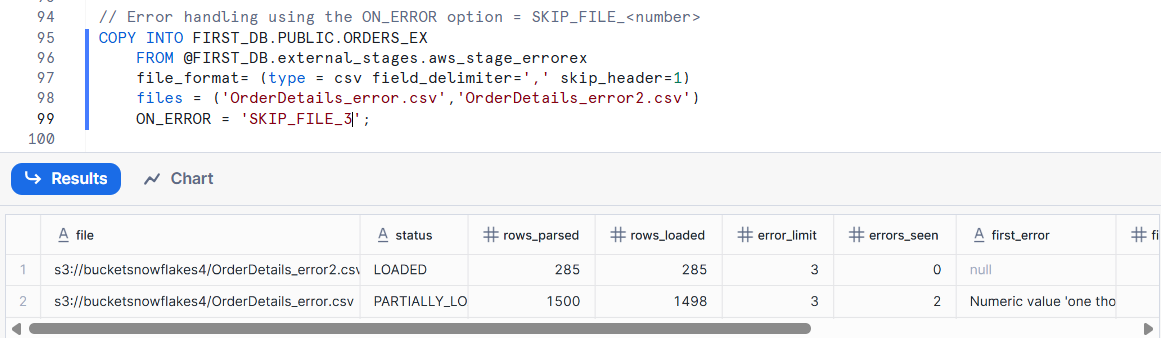
FROM @FIRST\_DB.external\_stages.aws\_stage\_errorex

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

files = ('OrderDetails\_error.csv','OrderDetails\_error2.csv')

ON\_ERROR = 'SKIP\_FILE\_2';





**Example : SKIP\_FILE\_<number>\_PERCENT**

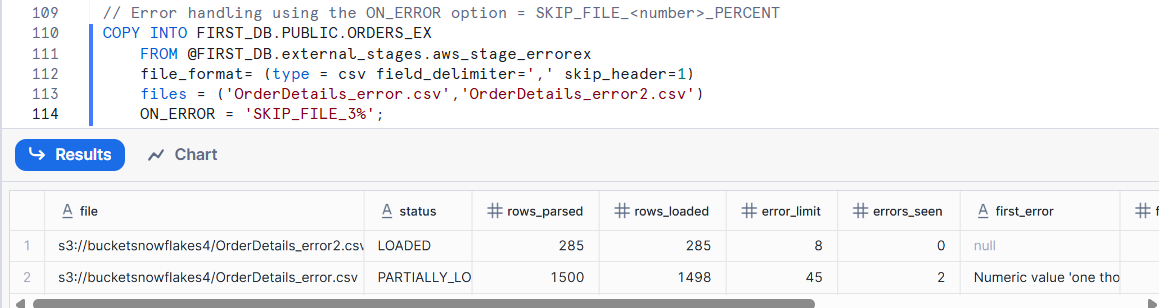
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

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

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

files = ('OrderDetails\_error.csv','OrderDetails\_error2.csv')

ON\_ERROR = 'SKIP\_FILE\_3%';



| **Limit Type** | **Limit** | **Best Practice** |
| --- | --- | --- |
| Per compressed file | 5 GB max | Prefer smaller files (100–250 MB) |
| Per uncompressed file | 50 GB max | Split files for speed |
| Per row | 16 MB | Avoid very wide records |
| Total COPY load size | No hard limit | Use parallel load with many files |
| Number of files per COPY | Thousands recommended, not millions | Batch loads if needed |

**File Formats:**

file formats define how data files are structured when you load (COPY INTO) or unload (COPY INTO … FROM) data between stages and tables.

| **File Format** | **Description** | **Typical Extension** |
| --- | --- | --- |
| **CSV** | Comma-separated values; plain text format widely used for tabular data. | .csv / .txt |
| **JSON** | Semi-structured data stored in JavaScript Object Notation format. | .json |
| **Avro** | Row-based binary format ideal for streaming and big data pipelines. | .avro |
| **ORC** | Optimized Row Columnar format, efficient for analytics workloads. | .orc |
| **Parquet** | Columnar format with good compression and performance. | .parquet |
| **XML** | Semi-structured data in Extensible Markup Language format. | .xml |

**Example to load CSV file:**

// Creating schema to keep things organized

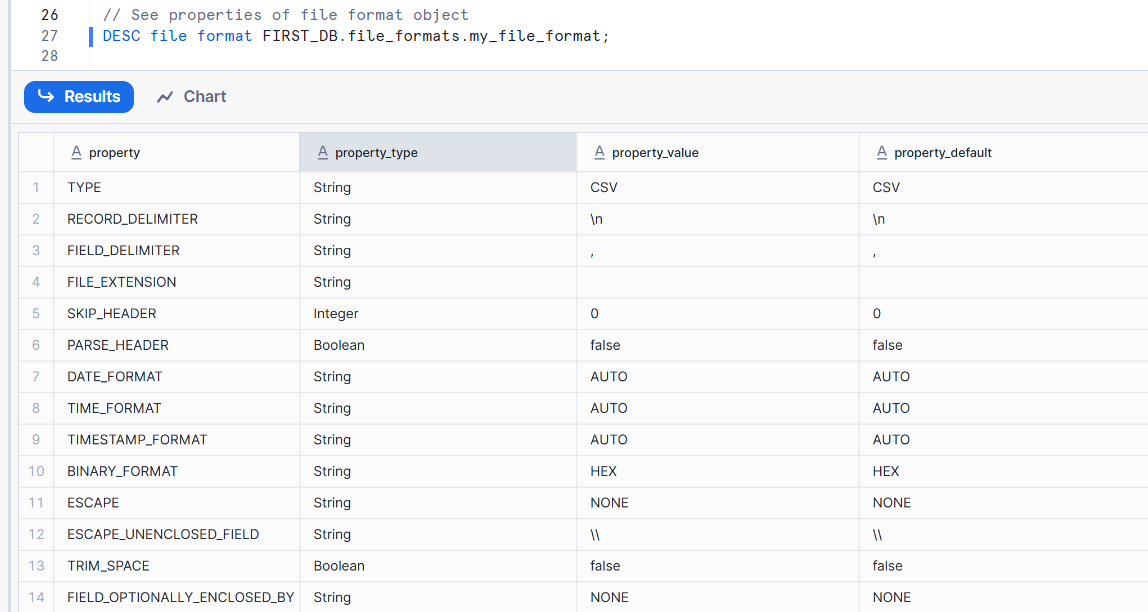
CREATE OR REPLACE SCHEMA FIRST\_DB.file\_formats;

// Creating file format object

CREATE OR REPLACE file format FIRST\_DB.file\_formats.my\_file\_format;

// See properties of file format object

DESC file format FIRST\_DB.file\_formats.my\_file\_format;



// Creating table

CREATE OR REPLACE TABLE FIRST\_DB.PUBLIC.ORDERS\_EX (

ORDER\_ID VARCHAR(30),

AMOUNT INT,

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Using file format object in Copy command

COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

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

file\_format= (FORMAT\_NAME=FIRST\_DB.file\_formats.my\_file\_format)

files = ('OrderDetails\_error.csv')

ON\_ERROR = 'SKIP\_FILE\_3';

**\* Load failed due to snowflake is considering header as first record.**

// Altering file format object to skip the header

ALTER file format FIRST\_DB.file\_formats.my\_file\_format

SET SKIP\_HEADER = 1;

After changing file format valid records are loaded into table.



**Note: Altering the type of a file format is not possible instead recreate the file format.**

**VALIDATION\_MODE:**

VALIDATION\_MODE parameter in the COPY INTO command allows you to validate your data files before actually loading them into a table.

| **Option** | **Description** | **Example Output** |
| --- | --- | --- |
| RETURN\_ALL\_ERRORS | Returns **all** errors found in the file(s). | Lists all error rows |
| RETURN\_ERRORS | Returns up to **1,000 error rows**. | Sample of error rows |
| RETURN\_N\_ROWS | Returns the **first N rows** from each file. | Quick preview of data |

// Prepare database & table

CREATE OR REPLACE DATABASE COPY\_DB;

//creating ordres table

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

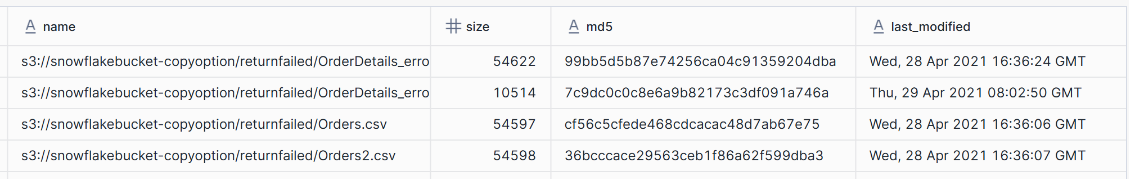
CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

create or replace stage copy\_db.public.aws\_stage\_copy

url ='s3://snowflakebucket-copyoption/returnfailed/';

list @copy\_db.public.aws\_stage\_copy;



-- show all errors --

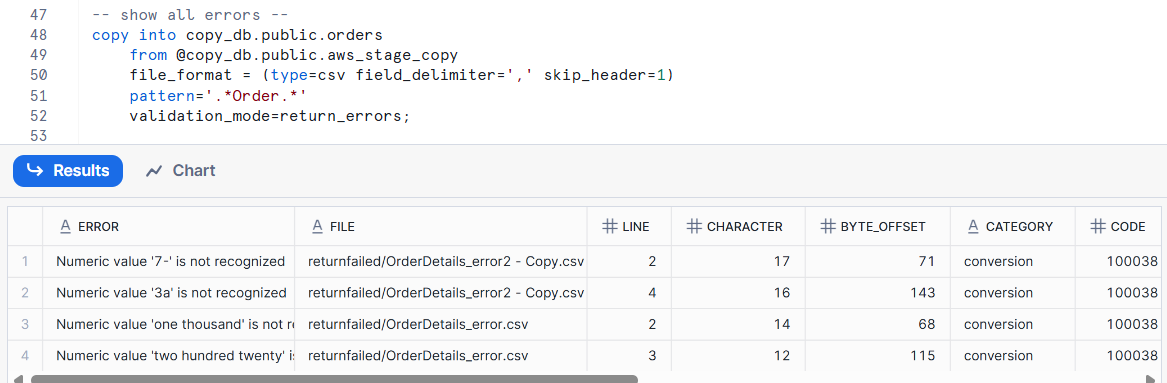
copy into copy\_db.public.orders

from @copy\_db.public.aws\_stage\_copy

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

pattern='.\*Order.\*'

validation\_mode=return\_errors;



-- validate first n rows --

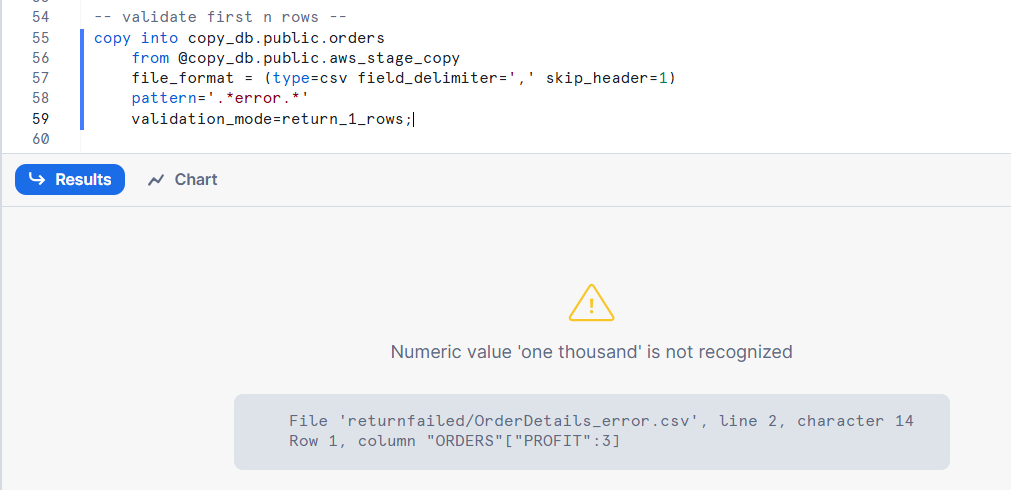
copy into copy\_db.public.orders

from @copy\_db.public.aws\_stage\_copy

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

pattern='.\*error.\*'

validation\_mode=return\_1\_rows;



**Working with rejected records:**

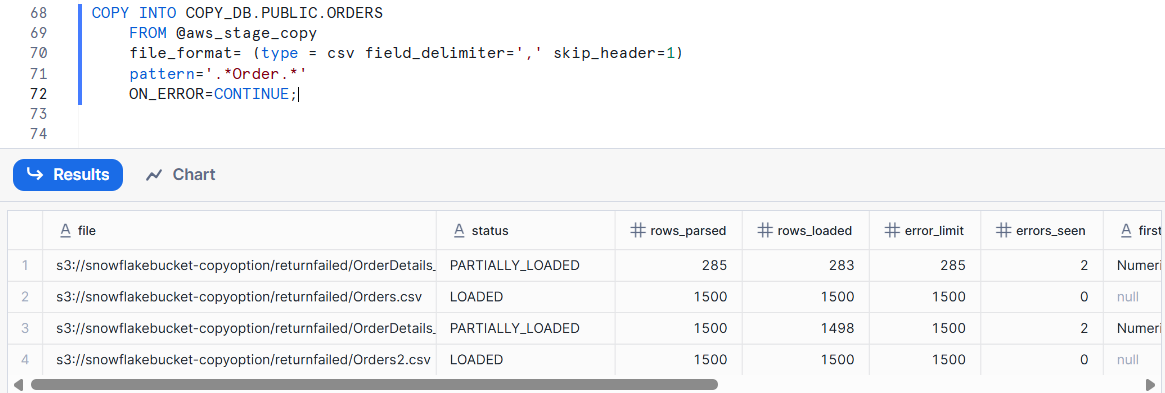
****

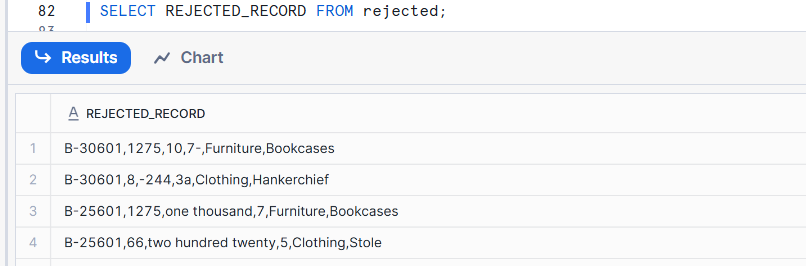
// Storing rejected /failed results in a table

CREATE OR REPLACE TABLE rejected AS

select rejected\_record from table(result\_scan(last\_query\_id()));

Load valid records into the actual table.





--creating another table with the rejected records

CREATE OR REPLACE TABLE rejected\_values as

SELECT

SPLIT\_PART(rejected\_record,',',1) as ORDER\_ID,

SPLIT\_PART(rejected\_record,',',2) as AMOUNT,

SPLIT\_PART(rejected\_record,',',3) as PROFIT,

SPLIT\_PART(rejected\_record,',',4) as QUATNTITY,

SPLIT\_PART(rejected\_record,',',5) as CATEGORY,

SPLIT\_PART(rejected\_record,',',6) as SUBCATEGORY

FROM rejected;



**SIZE\_LIMIT:**

SIZE\_LIMIT parameter in the COPY INTO command is used to restrict the maximum size (in bytes) of files that are processed during a load.

SIZE\_LIMIT = <bytes>

* The SIZE\_LIMIT value applies to the entire set of files processed by the COPY statement, not to each individual file.
* Skipped files appear in the COPY\_HISTORY and LOAD\_HISTORY views.
* A skipped file due to SIZE\_LIMIT is **not an error**, but Snowflake reports it in the output.
* Snowflake guarantees loading at least one file even if the total size exceeds SIZE\_LIMIT.
* The COPY operation continues processing the file that exceeds the threshold before stopping. For example, if the SIZE\_LIMIT is set to 15MB and there are three 10MB files, the process stops after successfully loading two files.

**Example**

// Prepare database & table

CREATE OR REPLACE DATABASE COPY\_DB;

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

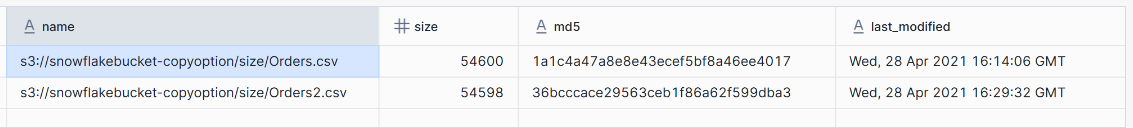
// Prepare stage object

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/size/';

// List files in stage

LIST @aws\_stage\_copy;



//Load data using copy command

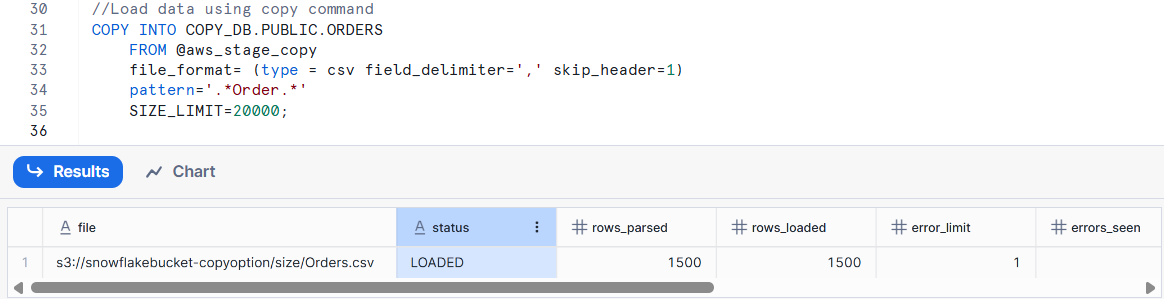
COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

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

pattern='.\*Order.\*'

SIZE\_LIMIT=20000;



Now increased the size limit to **54600**

****

In this example threshold is 54600 and first it loaded the file with 54598 and threshold is not exceeded so it loaded the second file.

**RETURN\_FAILED\_ONLY**

Specifies whether to return only files that have failed to load in the statement result.

//created a table

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

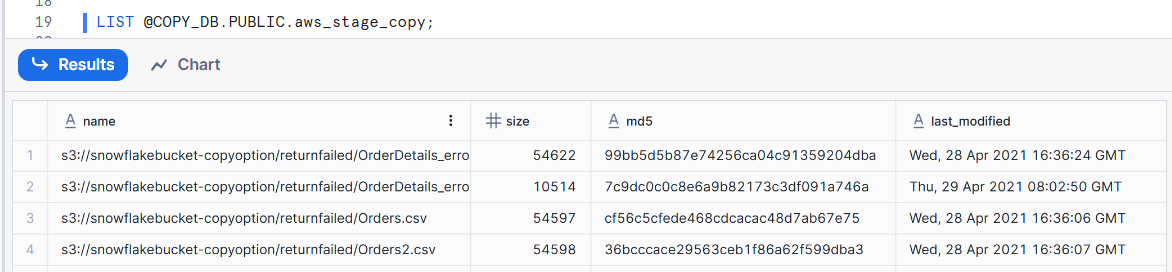
CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Prepare stage object

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/returnfailed/';



COPY INTO COPY\_DB.PUBLIC.ORDERS

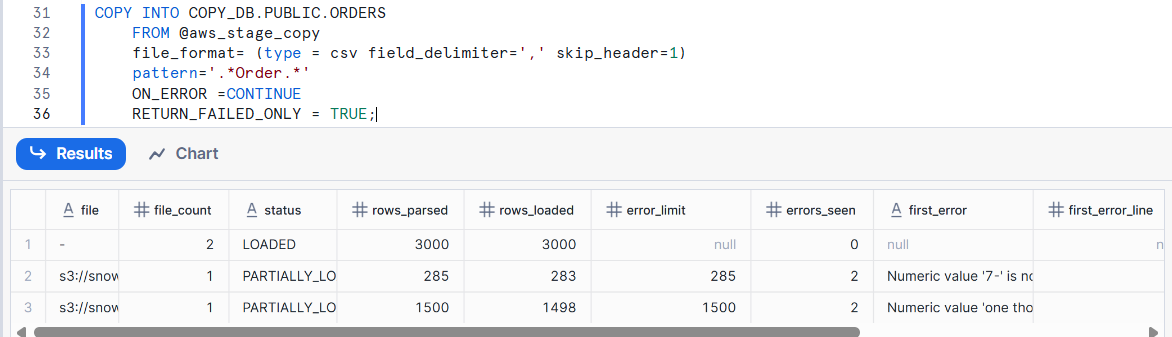
FROM @aws\_stage\_copy

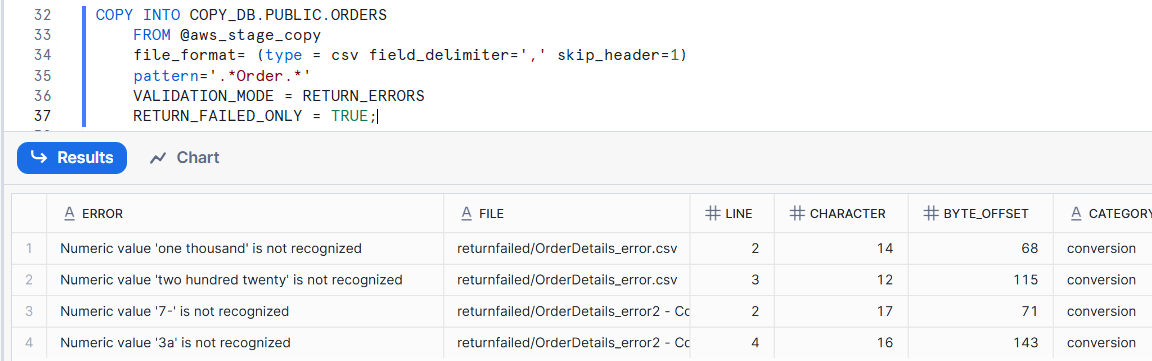
file\_format= (type = csv field\_delimiter=',' skip\_header=1)

pattern='.\*Order.\*'

ON\_ERROR =CONTINUE

RETURN\_FAILED\_ONLY = TRUE;





**RETURN\_FAILED\_ONLY must be used along with either VALIDATION\_MODE or ON\_ERROR parameters.**

**TRUNACATECOLUMNS:**

Specifies whether to truncate text strings that exceed the target column length.

* **TRUNCATECOLUMNS = TRUE** → The extra characters are **automatically truncated** to fit the column.
* **TRUNCATECOLUMNS = FALSE** (default) → The entire row **fails to load** and is recorded as an error.

**Example:**

//creating a table

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

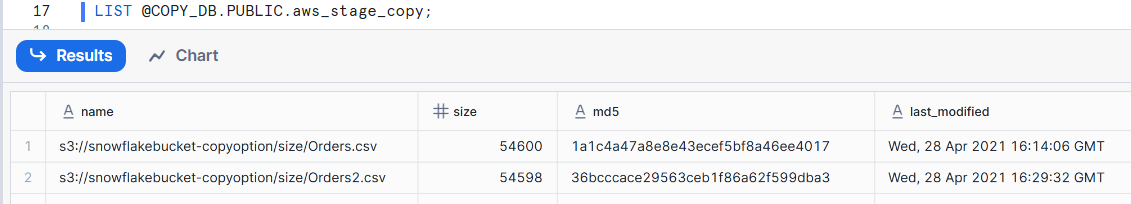
CATEGORY VARCHAR(10),

SUBCATEGORY VARCHAR(30));

// Prepare stage object

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/size/';



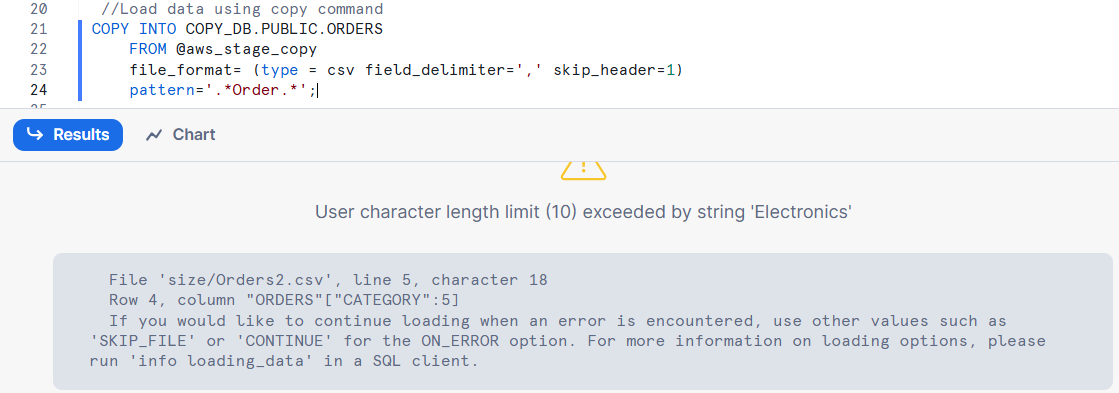
**//Copy command without TRUNCATECOLUMNS option**

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

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

pattern='.\*Order.\*';



**//Copy command with TRUNCATECOLUMNS option**

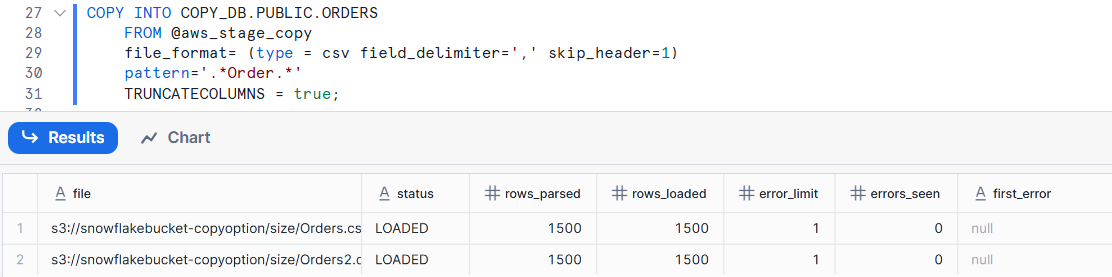
COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

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

pattern='.\*Order.\*'

TRUNCATECOLUMNS = true;



**FORCE:**

Specifies to load all files regardless of whether they have been loaded previously and have not changed since they were loaded.

This option reloads files potentially duplicating data in a table.

**Example:**

//creating a table

CREATE OR REPLACE TABLE COPY\_DB.PUBLIC.ORDERS (

ORDER\_ID VARCHAR(30),

AMOUNT VARCHAR(30),

PROFIT INT,

QUANTITY INT,

CATEGORY VARCHAR(30),

SUBCATEGORY VARCHAR(30));

// Prepare stage object

CREATE OR REPLACE STAGE COPY\_DB.PUBLIC.aws\_stage\_copy

url='s3://snowflakebucket-copyoption/size/';

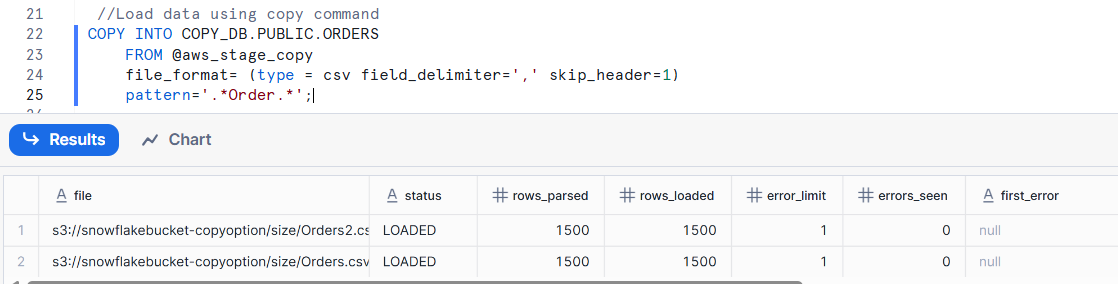
//Load data using copy command

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

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

pattern='.\*Order.\*';



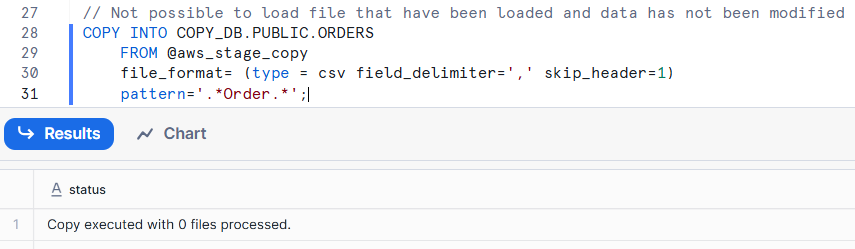
// Not possible to load file that have been loaded and data has not been modified

COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

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

pattern='.\*Order.\*';



// Using the FORCE option

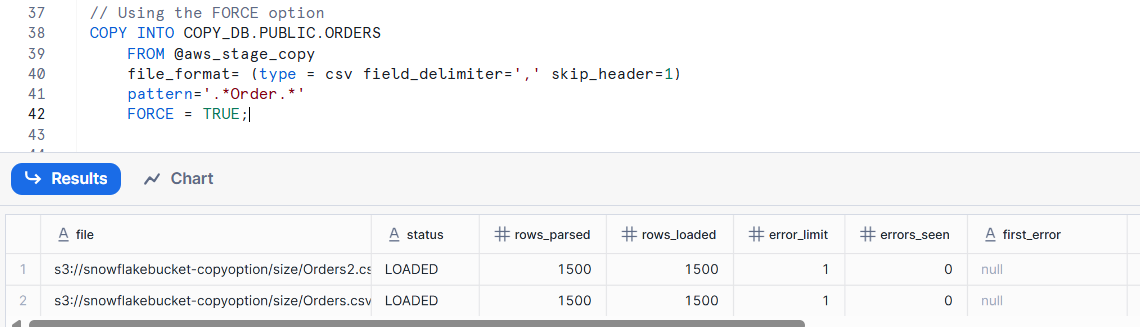
COPY INTO COPY\_DB.PUBLIC.ORDERS

FROM @aws\_stage\_copy

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

pattern='.\*Order.\*'

FORCE = TRUE;

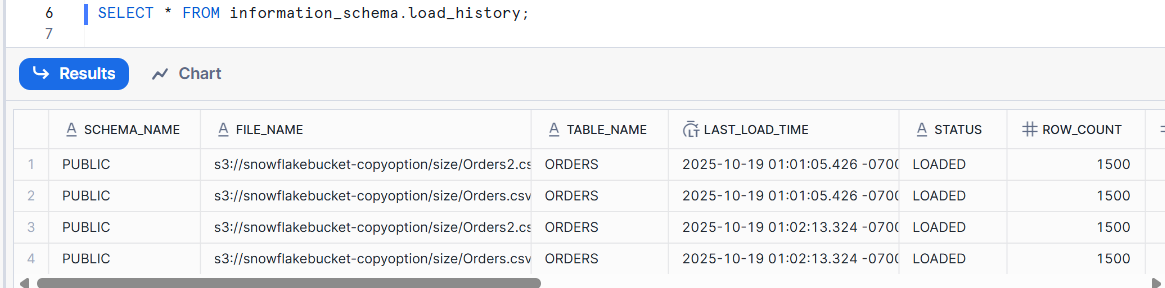


**Load history:**

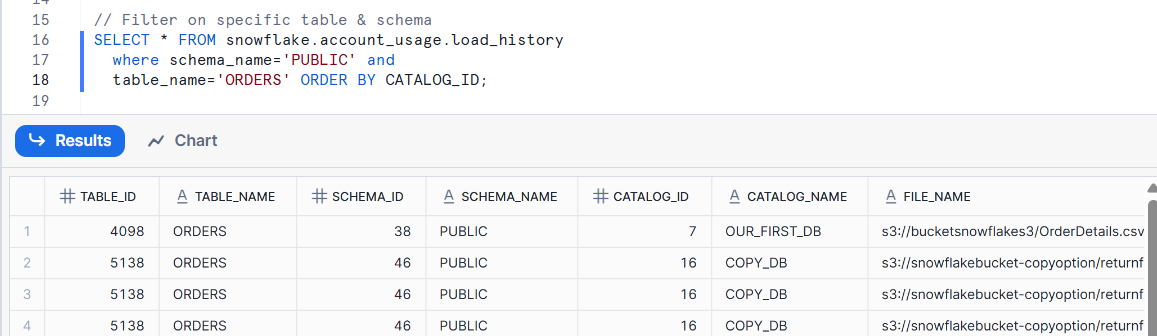
load history is a record Snowflake maintains to keep track of files that have been loaded using the COPY INTO command.

It’s an important part of how Snowflake prevents duplicate loading of the same file.

**To view load history in existing database**



**To view load history in global database**

****

**SEMI STRUCTURED DATA:**

semi-structured data refers to data that does not have a strict tabular format but still contains some organizational structure, such as key–value pairs, arrays, or nested elements.

👉 Common formats include:

* JSON (.json)
* Parquet (.parquet)
* Avro (.avro)
* ORC (.orc)
* XML (.xml)

**Example JSON:**

//creating stage

CREATE OR REPLACE stage FIRST\_DB.EXTERNAL\_STAGES.JSONSTAGE

url='s3://bucketsnowflake-jsondemo';

//creating JSON file format

CREATE OR REPLACE file format FIRST\_DB.FILE\_FORMATS.JSONFORMAT

TYPE = JSON;

****

//creating a table to store JSON data

CREATE OR REPLACE table FIRST\_DB.PUBLIC.JSON\_RAW (

raw\_file variant);

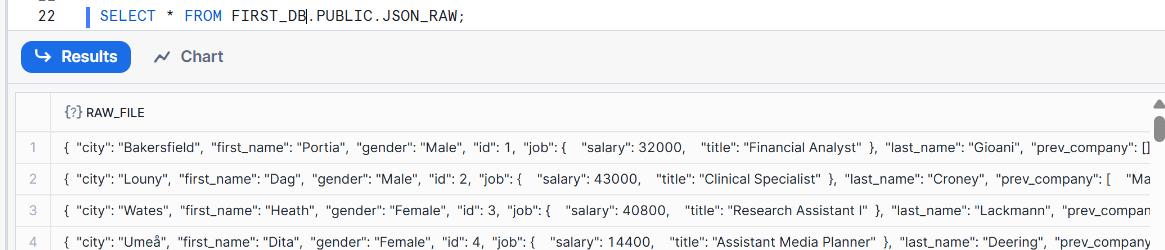
//copy command to load JSON data into table

COPY INTO FIRST\_DB.PUBLIC.JSON\_RAW

FROM @FIRST\_DB.EXTERNAL\_STAGES.JSONSTAGE

file\_format= FIRST\_DB.FILE\_FORMATS.JSONFORMAT

files = ('HR\_data.json');



**Record structure:**

{

"city": "Louny",

"first\_name": "Dag",

"gender": "Male",

"id": 2,

"job": {

"salary": 43000,

"title": "Clinical Specialist"

},

"last\_name": "Croney",

"prev\_company": [

"MacGyver, Kessler and Corwin",

"Gerlach, Russel and Moen"

],

"spoken\_languages": [

{

"language": "Assamese",

"level": "Basic"

},

{

"language": "Papiamento",

"level": "Expert"

},

{

"language": "Telugu",

"level": "Basic"

}

]

}

**Parse JSON data:**

SELECT

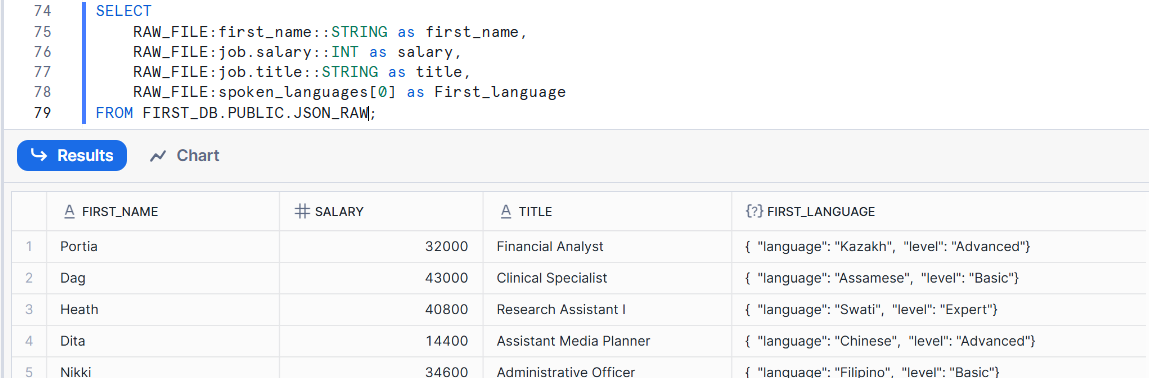
RAW\_FILE:first\_name::STRING as first\_name,

RAW\_FILE:job.salary::INT as salary,

RAW\_FILE:job.title::STRING as title,

RAW\_FILE:spoken\_languages[0] as First\_language

FROM FIRST\_DB.PUBLIC.JSON\_RAW;

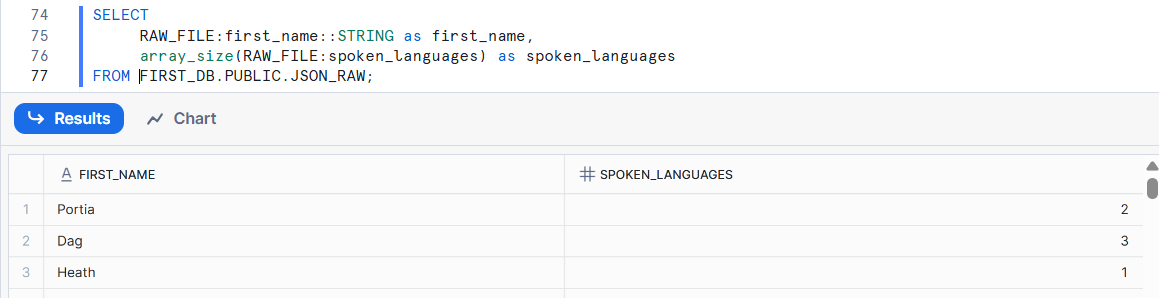


SELECT

RAW\_FILE:first\_name::STRING as first\_name,

array\_size(RAW\_FILE:spoken\_languages) as spoken\_languages

FROM FIRST\_DB.PUBLIC.JSON\_RAW;



Each person speaks multiple languages so to get all the languages record by record instead of using multiple union queries we can use flatten function.

SELECT

RAW\_FILE:id::int as id,

RAW\_FILE:first\_name::STRING as First\_name,

RAW\_FILE:spoken\_languages[0].language::STRING as First\_language,

RAW\_FILE:spoken\_languages[0].level::STRING as Level\_spoken

FROM FIRST\_DB.PUBLIC.JSON\_RAW

UNION ALL

SELECT

RAW\_FILE:id::int as id,

RAW\_FILE:first\_name::STRING as First\_name,

RAW\_FILE:spoken\_languages[1].language::STRING as First\_language,

RAW\_FILE:spoken\_languages[1].level::STRING as Level\_spoken

FROM FIRST\_DB.PUBLIC.JSON\_RAW

UNION ALL

SELECT

RAW\_FILE:id::int as id,

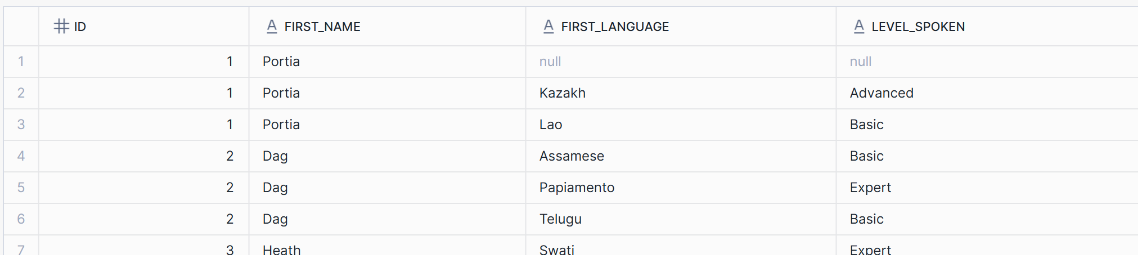
RAW\_FILE:first\_name::STRING as First\_name,

RAW\_FILE:spoken\_languages[2].language::STRING as First\_language,

RAW\_FILE:spoken\_languages[2].level::STRING as Level\_spoken

FROM FIRST\_DB.PUBLIC.JSON\_RAW

ORDER BY ID;



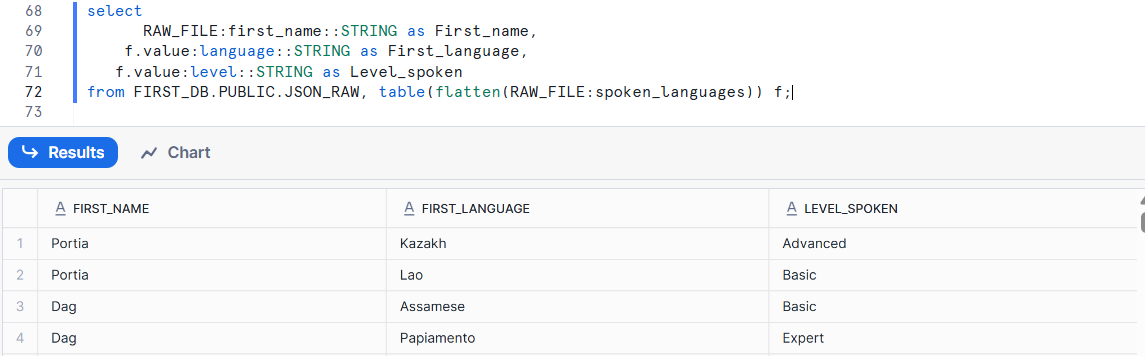
select

RAW\_FILE:first\_name::STRING as First\_name,

f.value:language::STRING as First\_language,

f.value:level::STRING as Level\_spoken

from FIRST\_DB.PUBLIC.JSON\_RAW, table(flatten(RAW\_FILE:spoken\_languages)) f;



**PARQUET file format:**

CREATE OR REPLACE FILE FORMAT FIRST\_DB.FILE\_FORMATS.PARQUET\_FORMAT

TYPE = 'parquet';

CREATE OR REPLACE STAGE FIRST\_DB.EXTERNAL\_STAGES.PARQUETSTAGE

url = 's3://snowflakeparquetdemo'

FILE\_FORMAT = FIRST\_DB.FILE\_FORMATS.PARQUET\_FORMAT;

// Create destination table

CREATE OR REPLACE TABLE FIRST\_DB.PUBLIC.PARQUET\_DATA (

ROW\_NUMBER int,

index\_level int,

cat\_id VARCHAR(50),

date date,

dept\_id VARCHAR(50),

id VARCHAR(50),

item\_id VARCHAR(50),

state\_id VARCHAR(50),

store\_id VARCHAR(50),

value int,

Load\_date timestamp default TO\_TIMESTAMP\_NTZ(current\_timestamp));

// Load the parquet data

COPY INTO FIRST\_DB.PUBLIC.PARQUET\_DATA

FROM (SELECT

METADATA$FILE\_ROW\_NUMBER,

$1:\_\_index\_level\_0\_\_::int,

$1:cat\_id::VARCHAR(50),

DATE($1:date::int ),

$1:"dept\_id"::VARCHAR(50),

$1:"id"::VARCHAR(50),

$1:"item\_id"::VARCHAR(50),

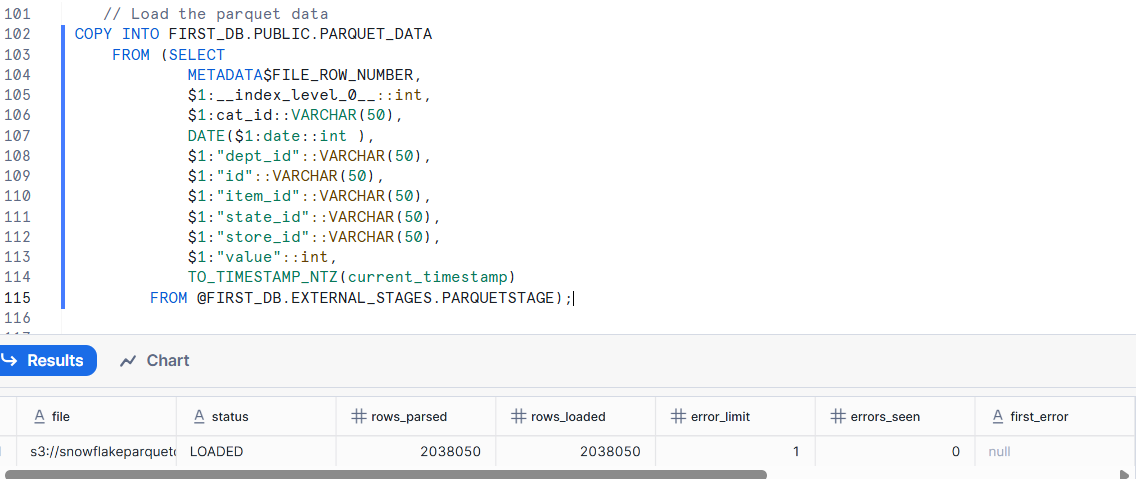
$1:"state\_id"::VARCHAR(50),

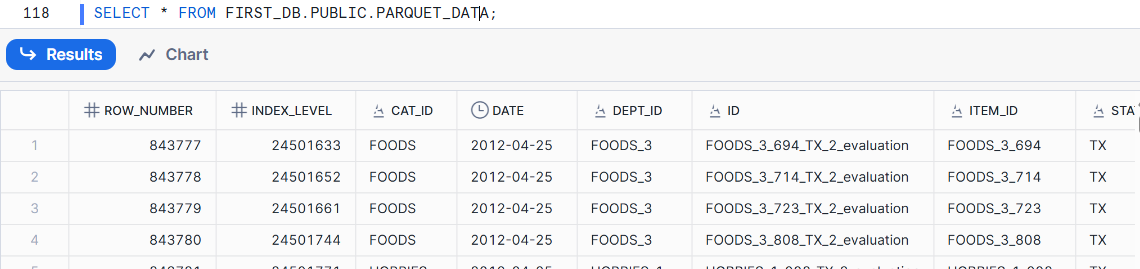
$1:"store\_id"::VARCHAR(50),

$1:"value"::int,

TO\_TIMESTAMP\_NTZ(current\_timestamp)

FROM @FIRST\_DB.EXTERNAL\_STAGES.PARQUETSTAGE);





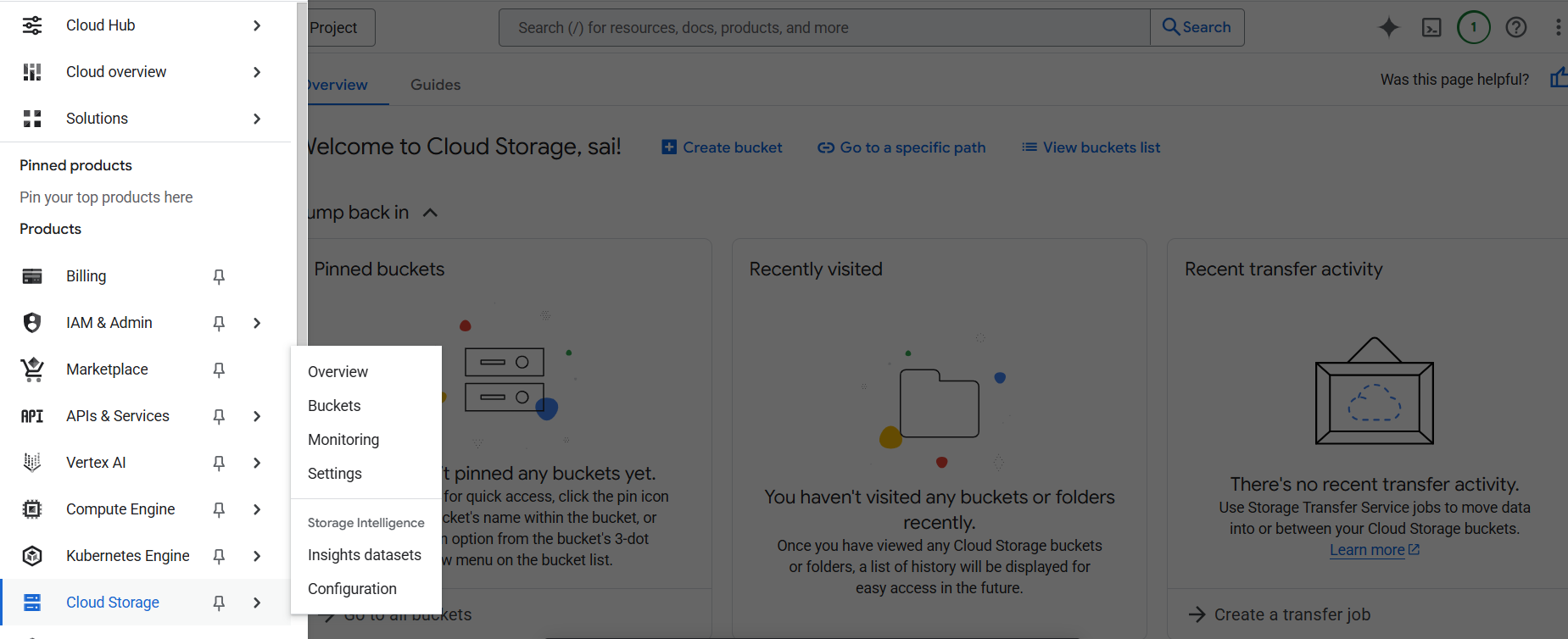
**Storage Integration**

Storage Integration is a secure way to connect Snowflake with external cloud storage such as:

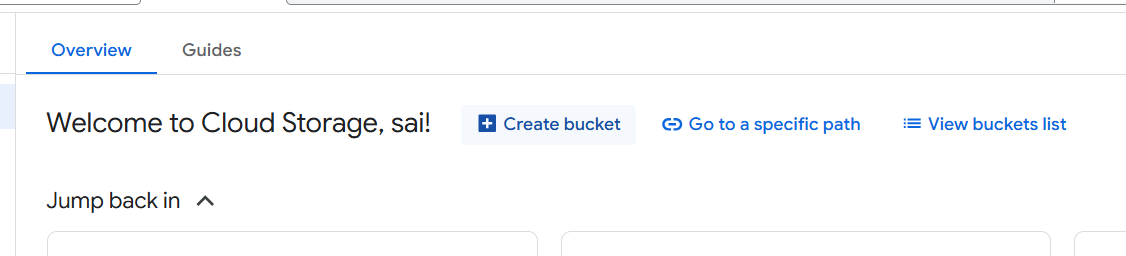
* ☁️ Amazon S3
* ☁️ Microsoft Azure Blob Storage
* ☁️ Google Cloud Storage

It allows Snowflake to access external cloud storage without storing credentials (like access keys or secrets) inside Snowflake.  
This is achieved by setting up a trusted relationship between Snowflake and your cloud provider.

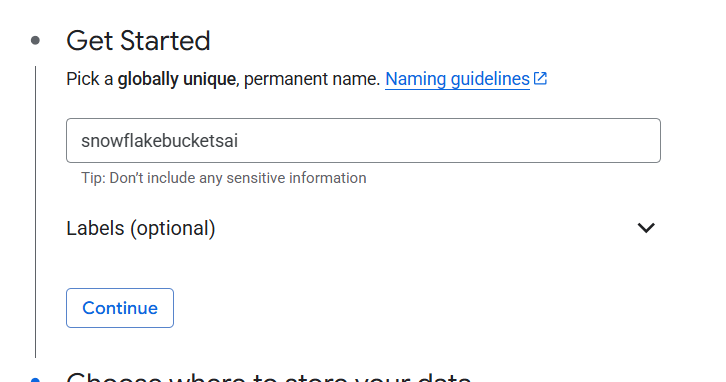
**Step 1: Creating a bucket in GCP.**

****

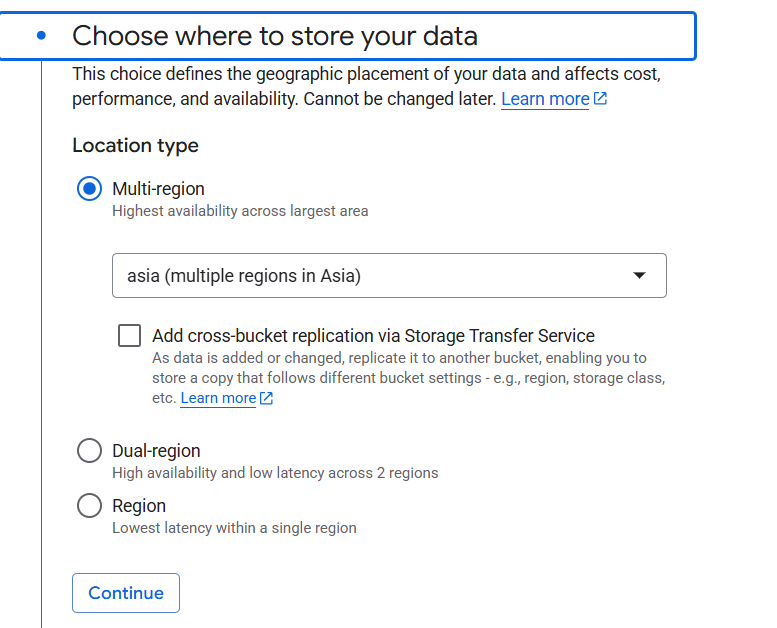
**Click on create bucket**

****

**Give unique bucketname and click continue.**

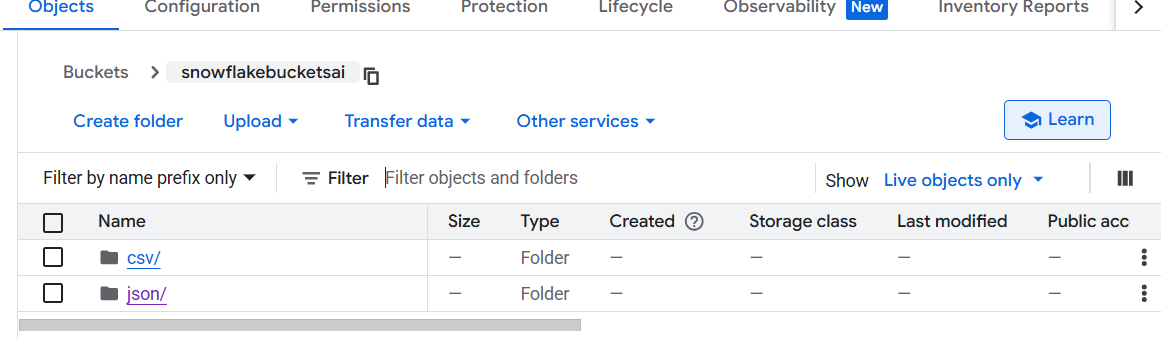
****

**Select region and click continue.**

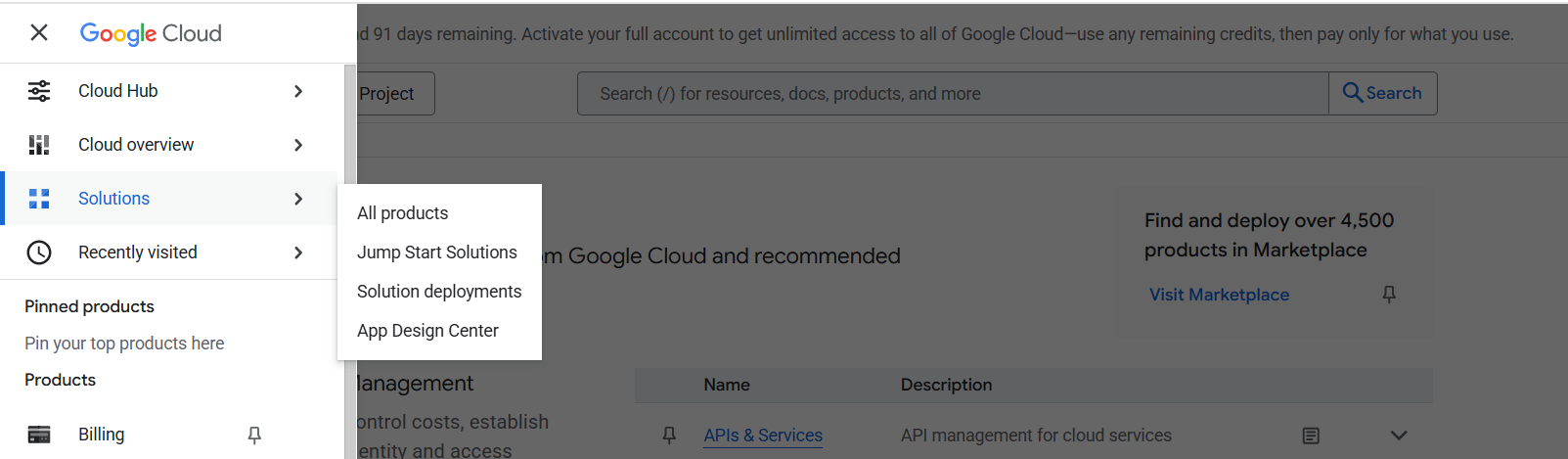
****

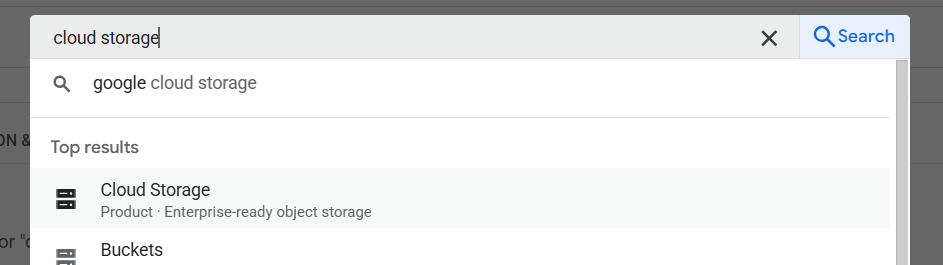
Choose all other options default and click on create.

Created 2 folders in bucket

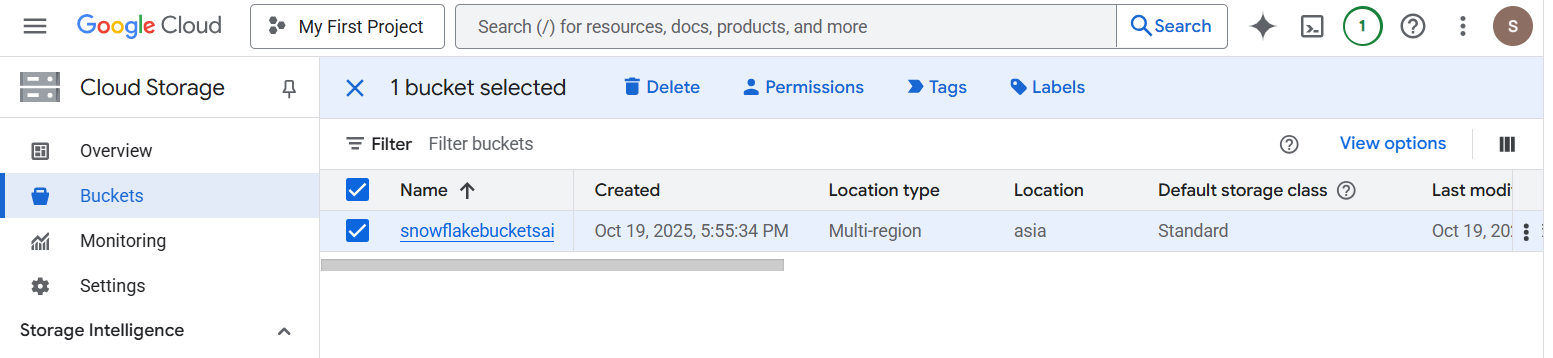


Go to Solutions and search for cloud storage.





Find your bucket name and give that name in storage integration object in snowflake.



-- create integration object that contains the access information

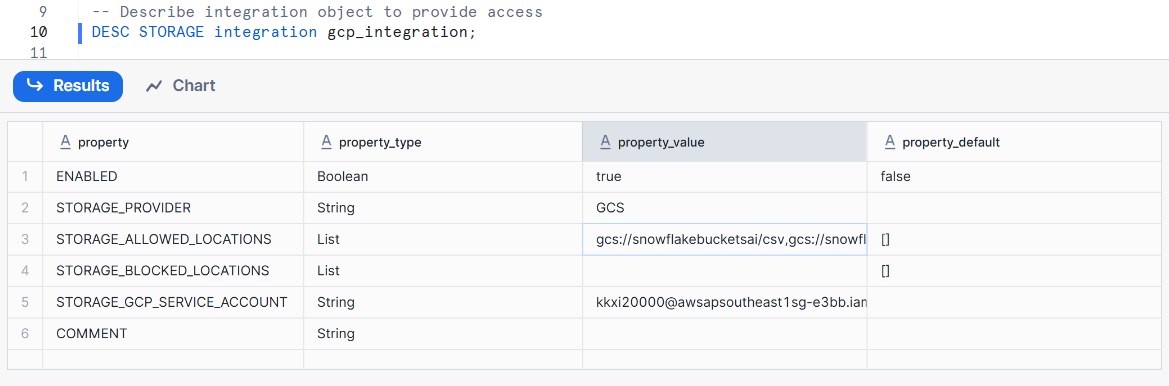
CREATE STORAGE INTEGRATION gcp\_integration

TYPE = EXTERNAL\_STAGE

STORAGE\_PROVIDER = GCS

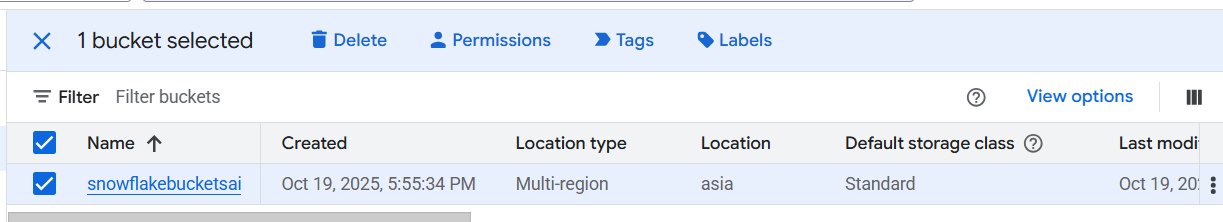
ENABLED = TRUE

STORAGE\_ALLOWED\_LOCATIONS = ('gcs://snowflakebucketsai/csv', 'gcs://snowflakebucketsai/csv');

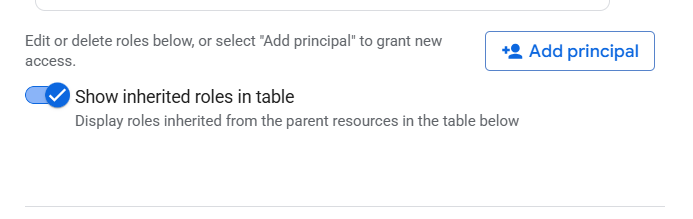


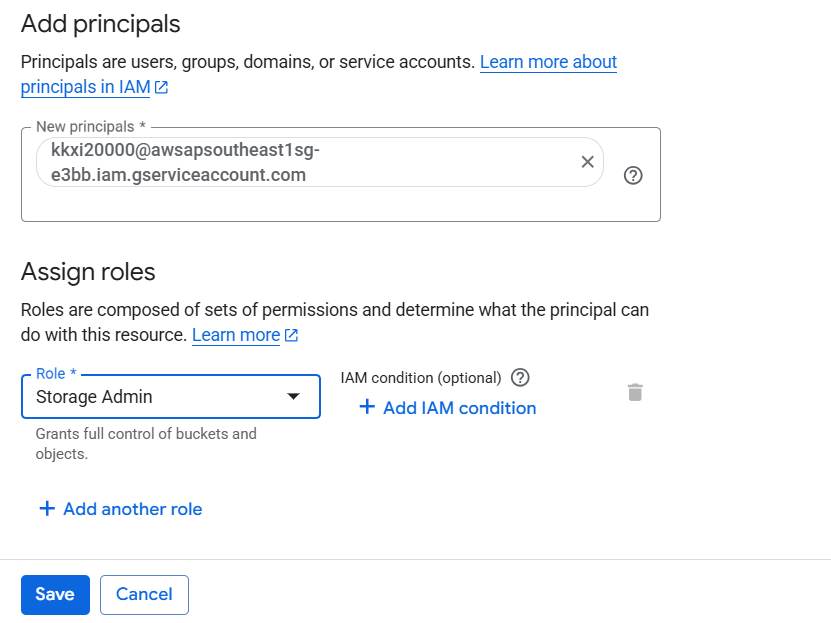
Now copy the STORAGE\_GCP\_SERVICE\_ACCOUNT value.

Select your bucket in GCP and click on permissions.



Click on Add principal.





-- create file format

create or replace file format copy\_db.public.fileformat\_gcp

TYPE = CSV

FIELD\_DELIMITER = ','

SKIP\_HEADER = 1;

-- create stage object

create or replace stage copy\_db.public.stage\_gcp

STORAGE\_INTEGRATION = gcp\_integration

URL = 'gcs://snowflakebucketsai/csv'

FILE\_FORMAT = fileformat\_gcp;



//create a table to load data

create or replace table happiness (

country\_name varchar,

regional\_indicator varchar,

ladder\_score number(4,3),

standard\_error number(4,3),

upperwhisker number(4,3),

lowerwhisker number(4,3),

logged\_gdp number(5,3),

social\_support number(4,3),

healthy\_life\_expectancy number(5,3),

freedom\_to\_make\_life\_choices number(4,3),

generosity number(4,3),

perceptions\_of\_corruption number(4,3),

ladder\_score\_in\_dystopia number(4,3),

explained\_by\_log\_gpd\_per\_capita number(4,3),

explained\_by\_social\_support number(4,3),

explained\_by\_healthy\_life\_expectancy number(4,3),

explained\_by\_freedom\_to\_make\_life\_choices number(4,3),

explained\_by\_generosity number(4,3),

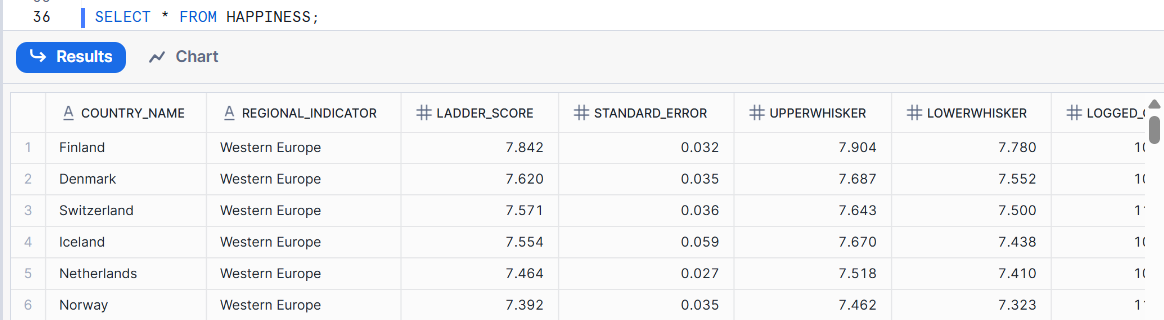
explained\_by\_perceptions\_of\_corruption number(4,3),

dystopia\_residual number (4,3));

//copy command to load data

COPY INTO HAPPINESS

FROM @copy\_db.public.stage\_gcp;



**Unloadind data**

-- create stage object

create or replace stage copy\_db.public.stage\_gcp

STORAGE\_INTEGRATION = gcp\_integration

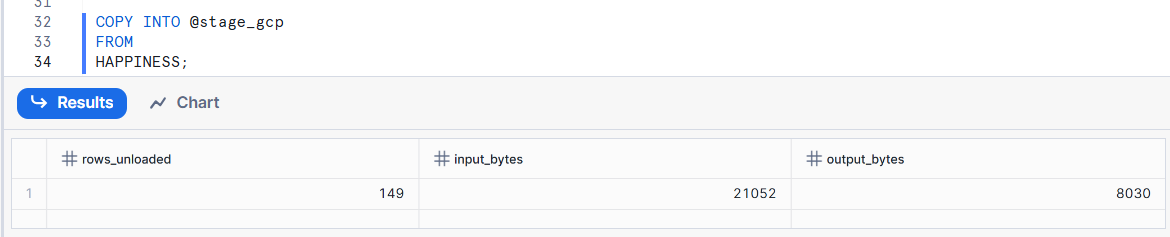
URL = 'gcs://snowflakebucketsai/csv/unload'

FILE\_FORMAT = fileformat\_gcp;

COPY INTO @stage\_gcp

FROM

HAPPINESS;



Now one zip file is available in unload path in GCP.

