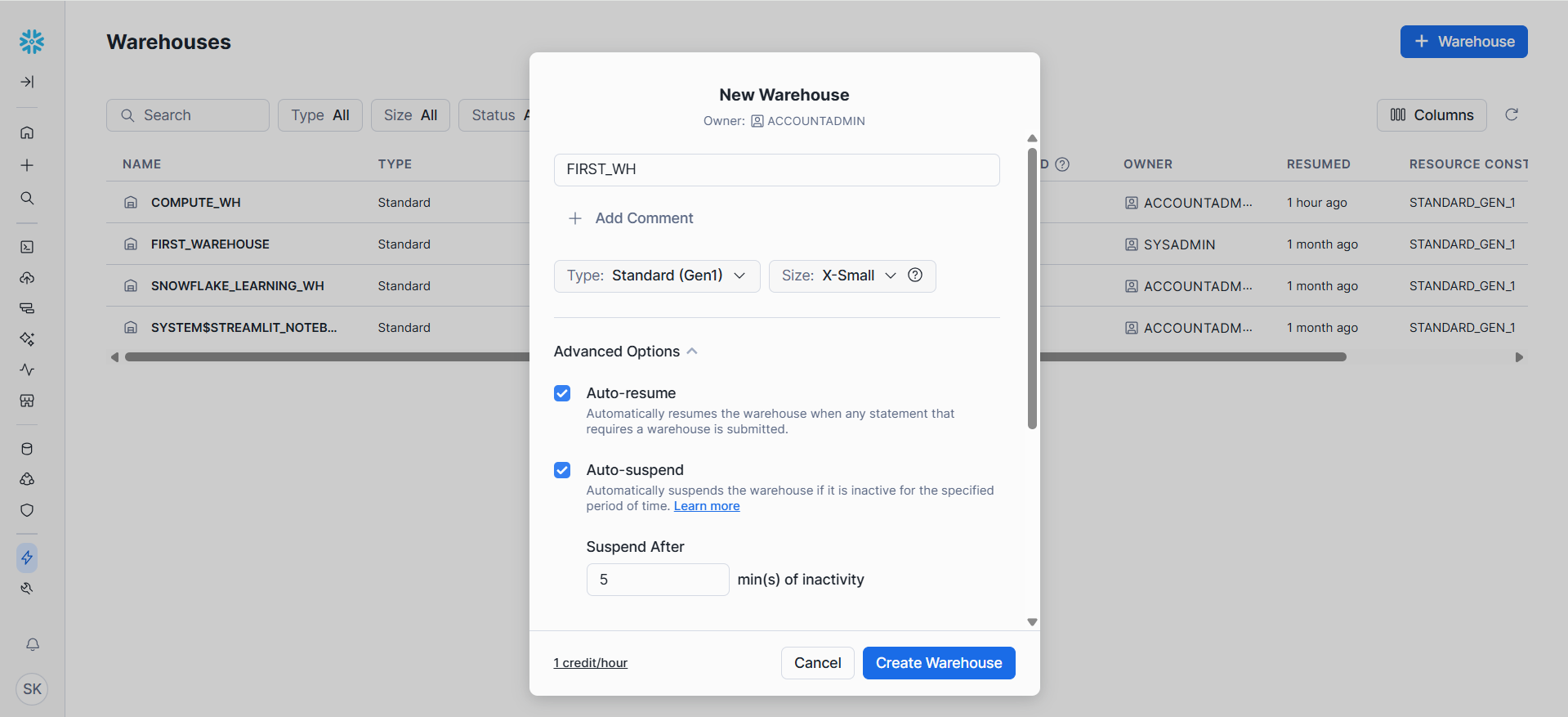
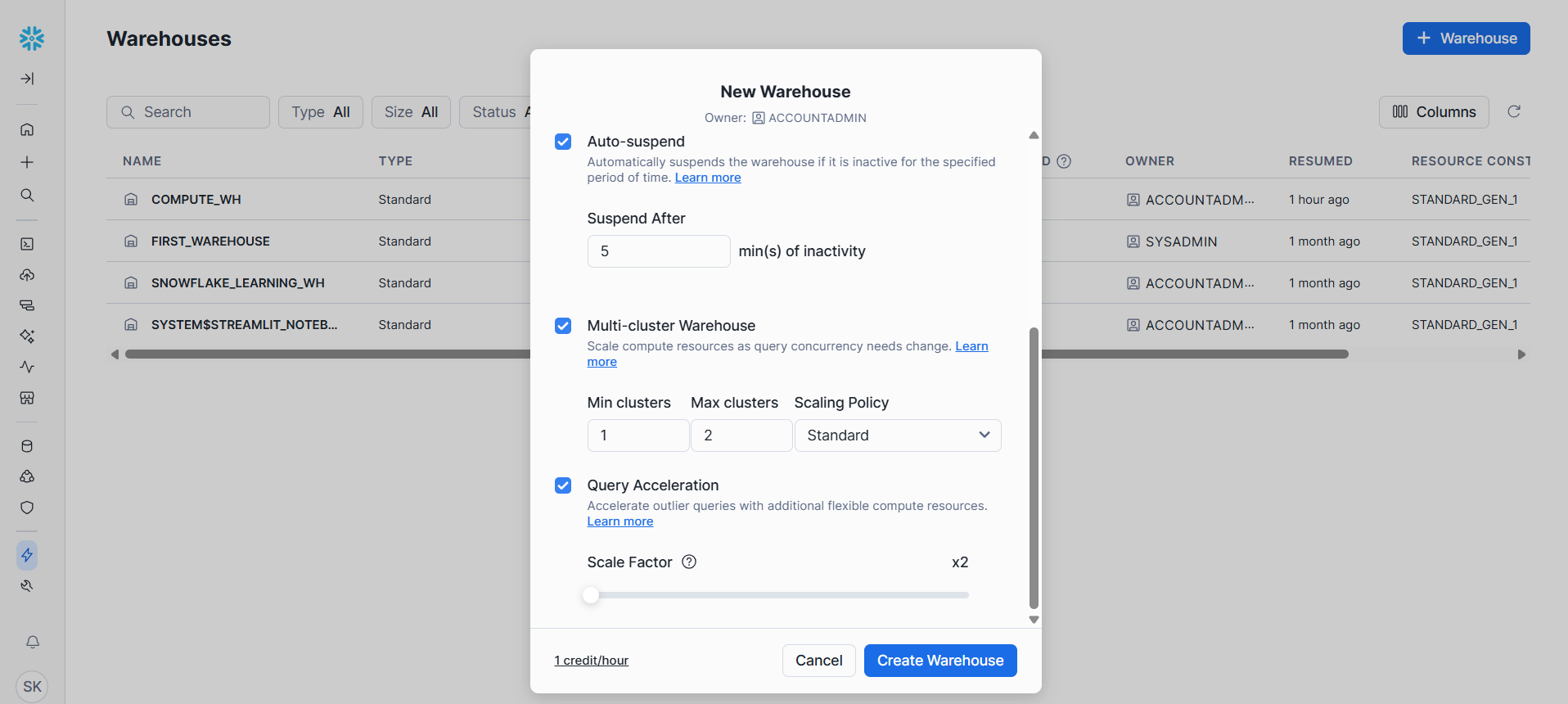
**Warehouse creation:**

**In Web UI**





**Using Snowsight**

****

**Example:**

**CREATE OR REPLACE WAREHOUSE SAI\_WH**

**WITH**

**WAREHOUSE\_SIZE = XSMALL**

**MIN\_CLUSTER\_COUNT = 1**

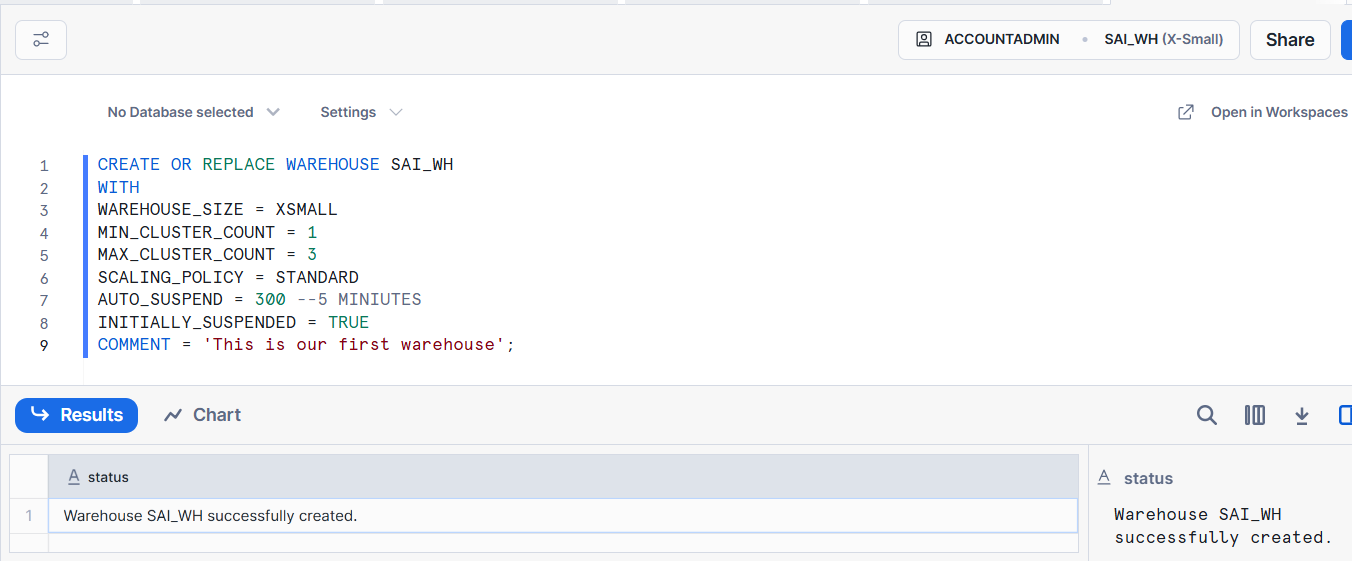
**MAX\_CLUSTER\_COUNT = 3**

**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;

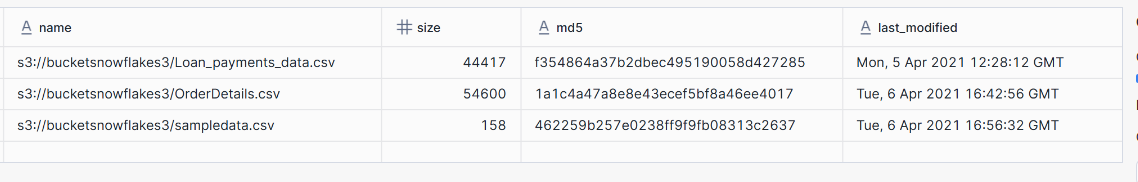
// creating stage object from publicly available s3 bucket

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

);

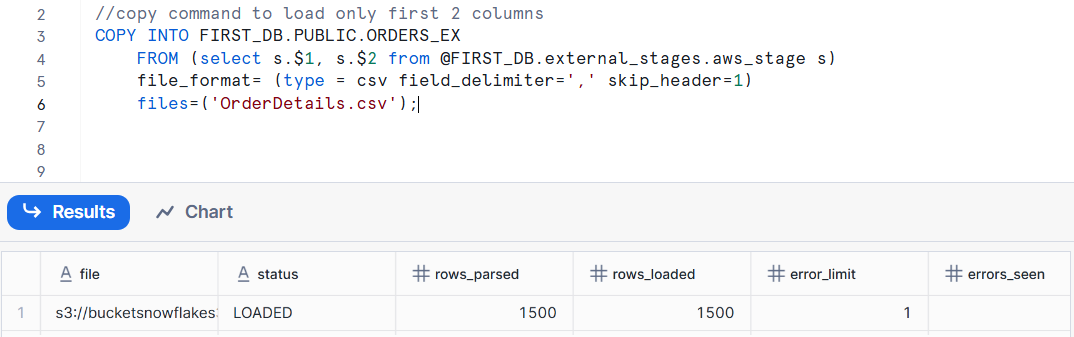
//copy command to load only first 2 columns

COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

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

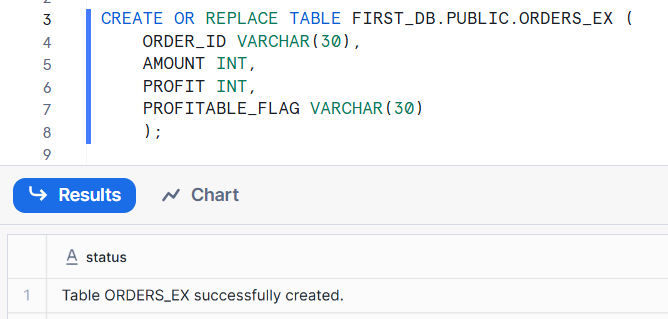
file\_format= (type = csv field\_delimiter=',' skip\_header=1)

files=('OrderDetails.csv');





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



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

FROM (select

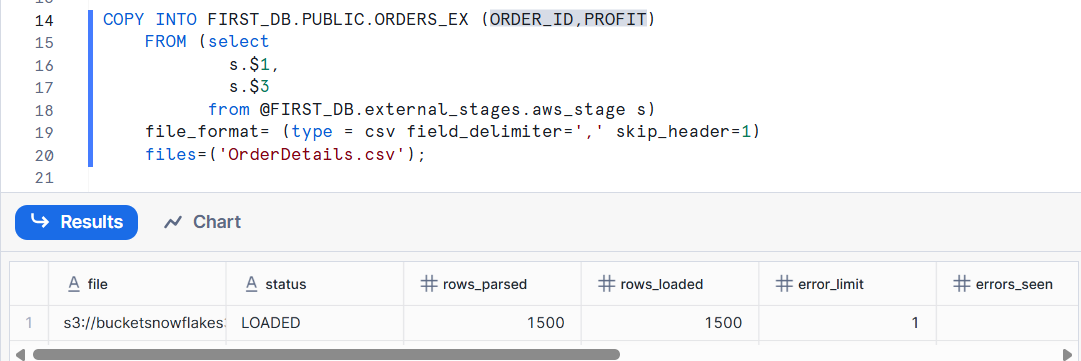
s.$1,

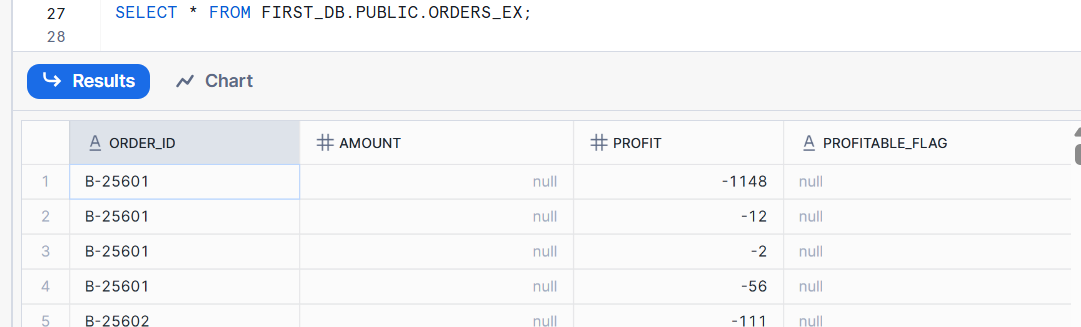
s.$3

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**

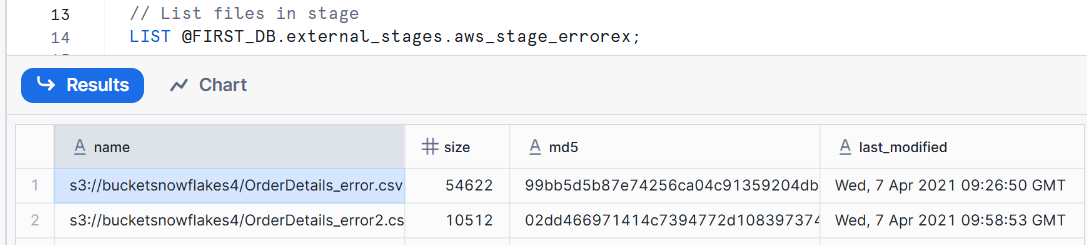
| Option | Behavior | When to Use |
| --- | --- | --- |
| ABORT\_STATEMENT *(default)* | Stops the entire load if any error occurs. | When you want strict data quality. |
| CONTINUE | Loads valid records and skips the bad ones. | When you can tolerate some bad records. |
| SKIP\_FILE | Skips the entire 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

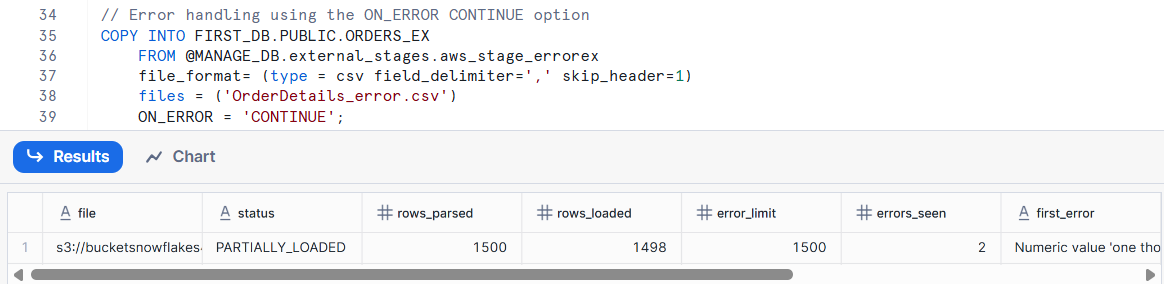
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

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**

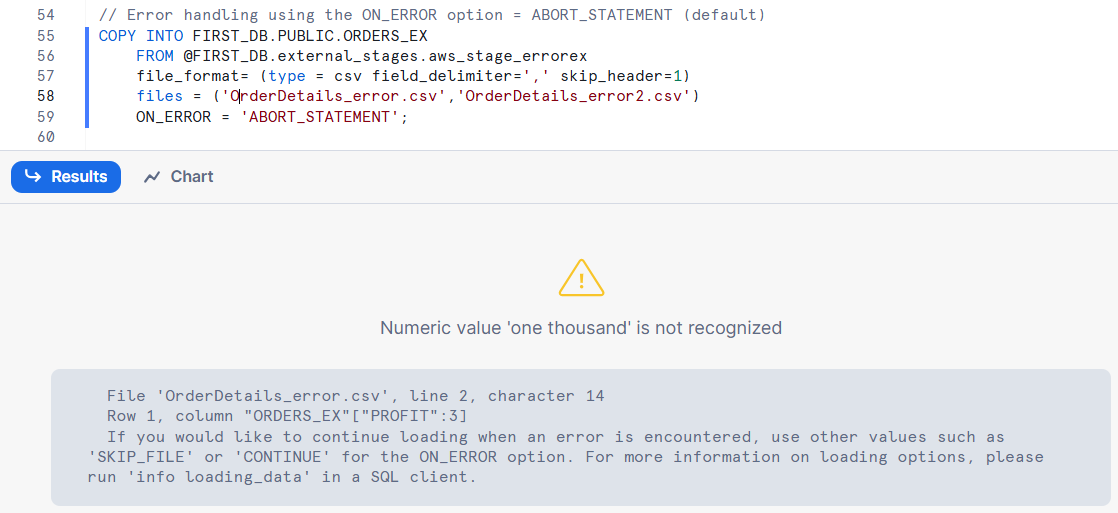
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

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

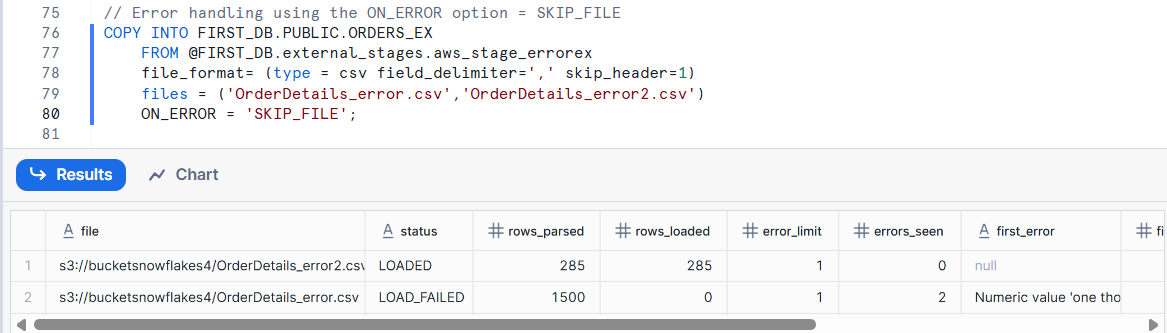
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

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>**

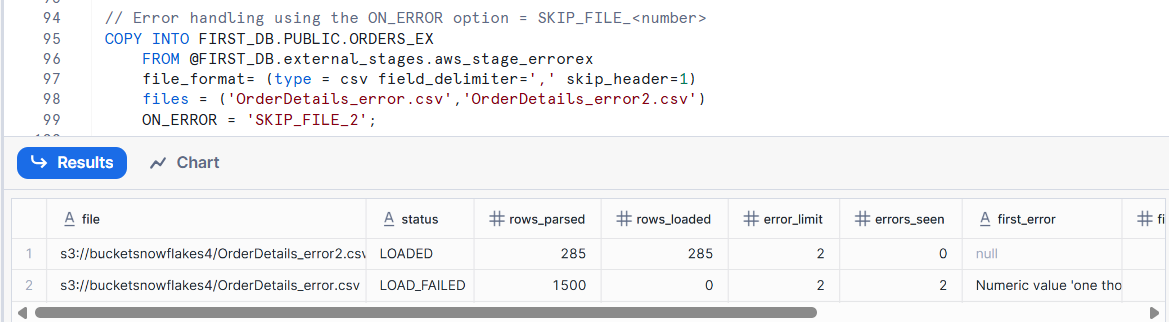
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

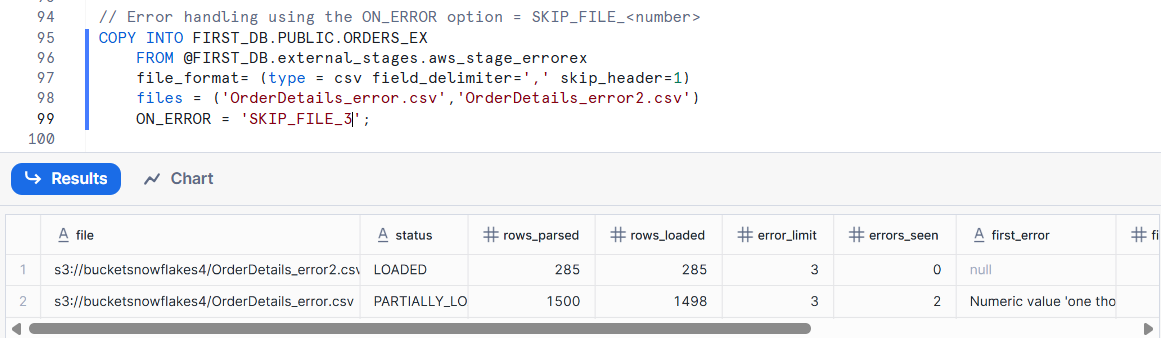
FROM @FIRST\_DB.external\_stages.aws\_stage\_errorex

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**

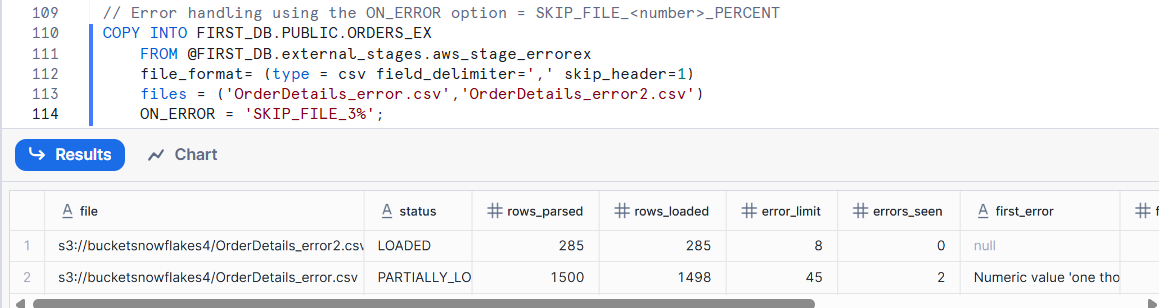
COPY INTO FIRST\_DB.PUBLIC.ORDERS\_EX

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

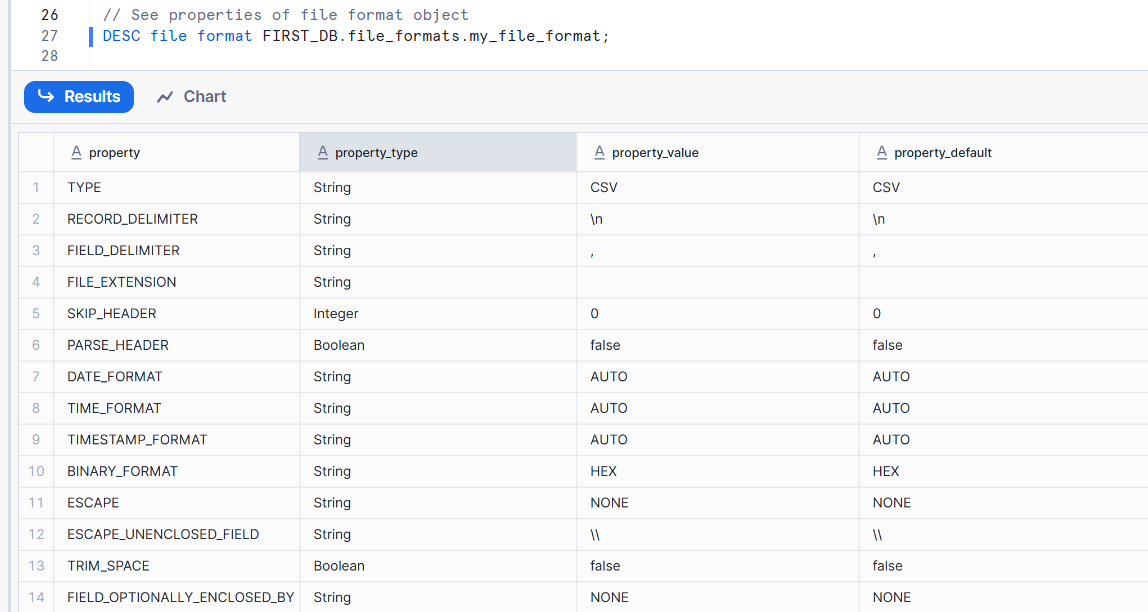
CREATE OR REPLACE SCHEMA FIRST\_DB.file\_formats;

// 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,

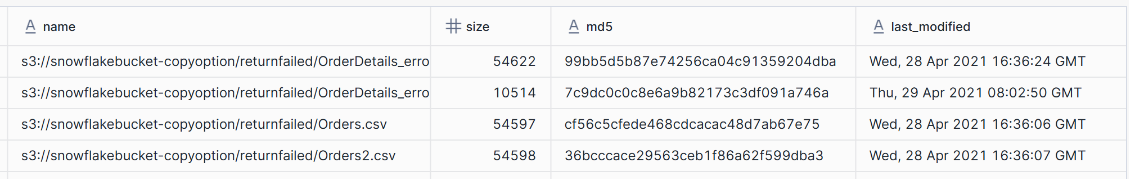
CATEGORY VARCHAR(30),

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 --

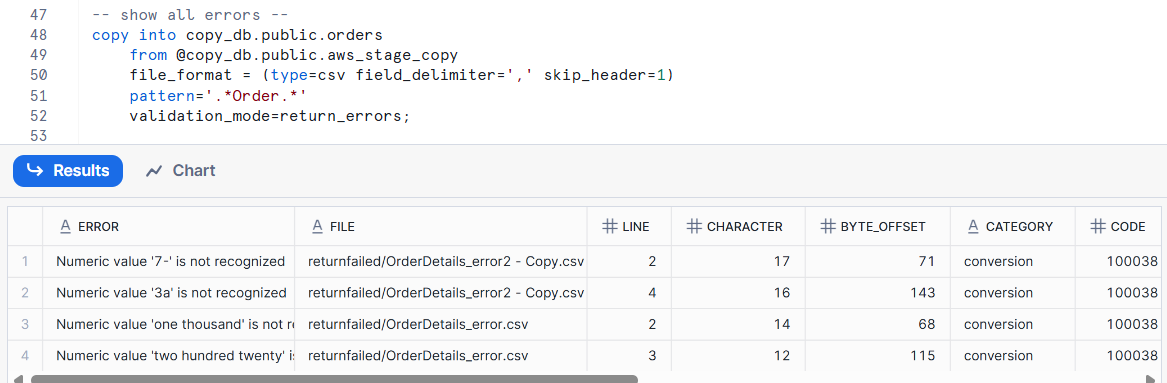
copy into copy\_db.public.orders

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 --

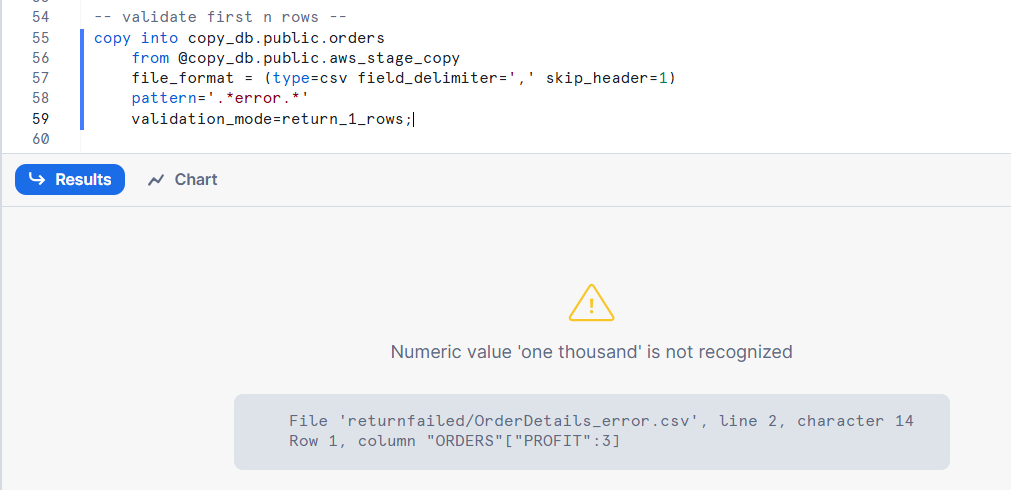
copy into copy\_db.public.orders

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:**

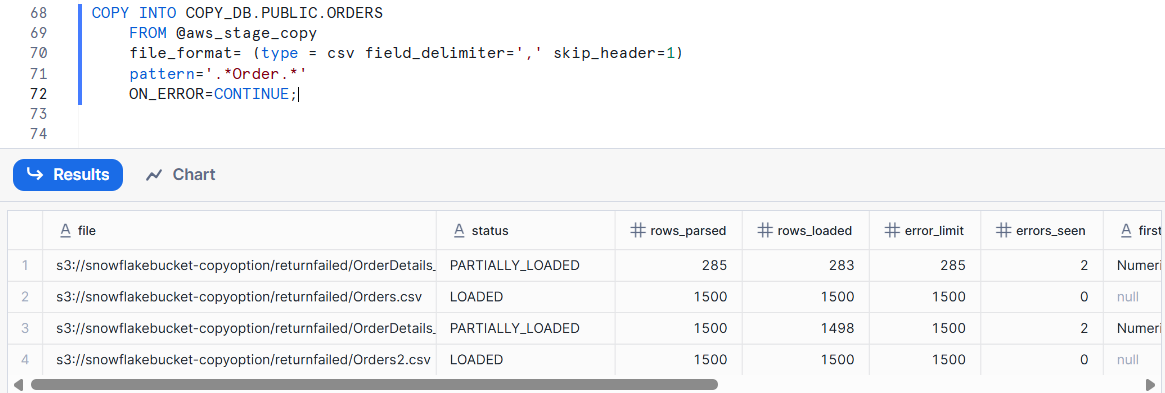
****

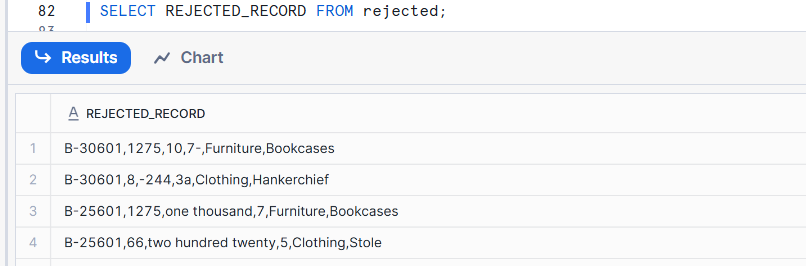
// Storing rejected /failed results in a table

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,

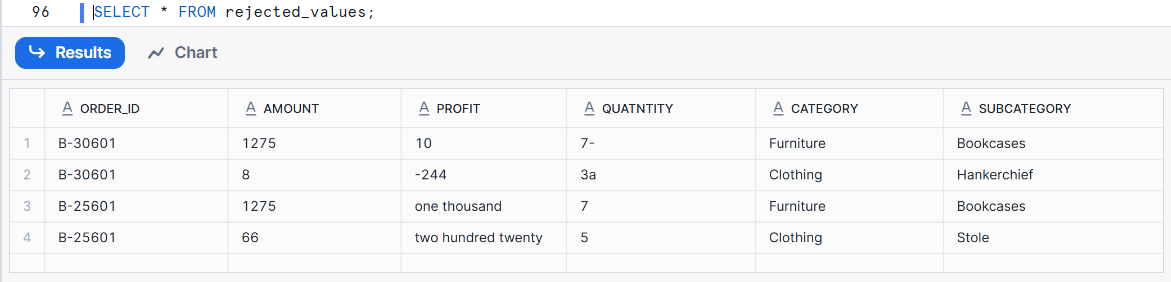
SPLIT\_PART(rejected\_record,',',3) as PROFIT,

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));

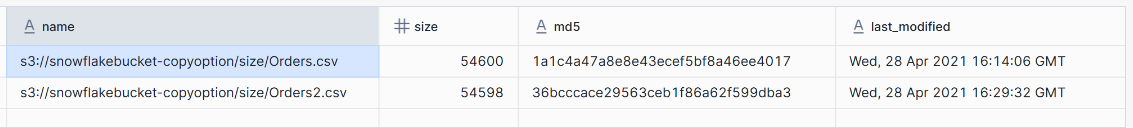
// Prepare stage object

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

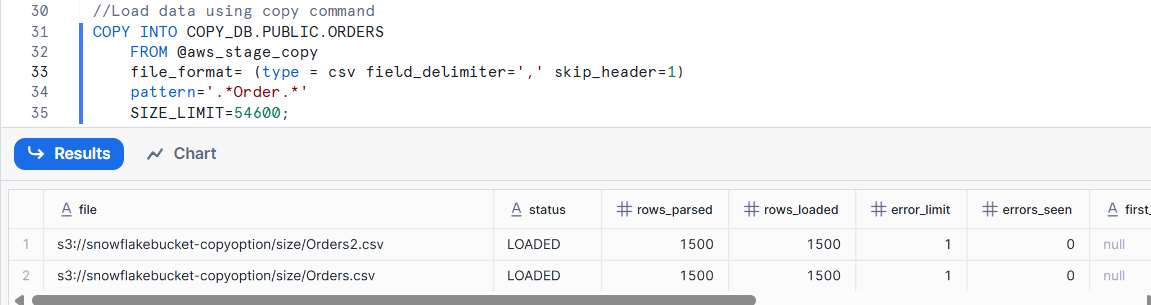
file\_format= (type = csv field\_delimiter=',' skip\_header=1)

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,

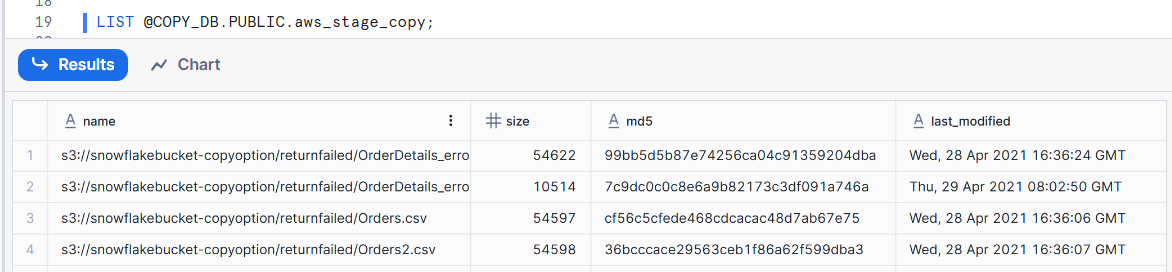
CATEGORY VARCHAR(30),

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

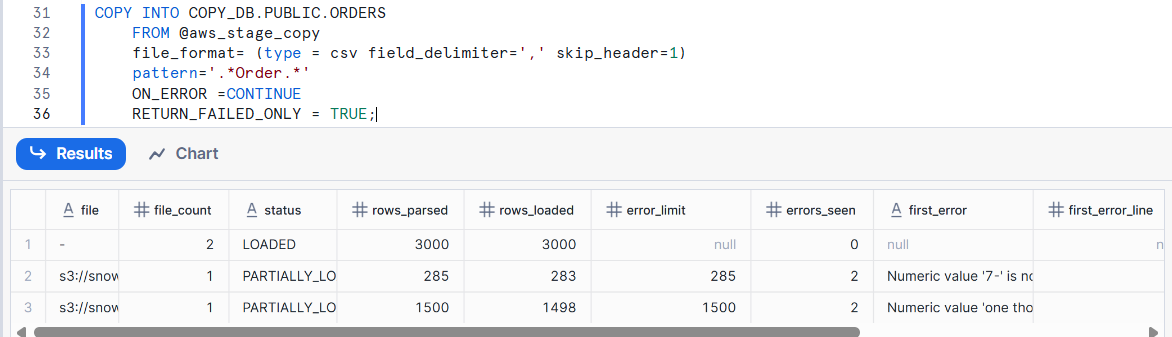
FROM @aws\_stage\_copy

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.**