ThinkETL

MENU

Data Load using COPY INTO Command in Snowflake

February 23, 2025

Spread the love

Contents [hide]

1. What is COPY INTO Command in Snowflake?

COPY INTO command in Snowflake enables loading data from files in a stage location to an existing table or unloading data from a table into a stage. Snowflake Stages are locations that store files which helps in loading and unloading into tables.

There are two types of Snowflake stages

- **Internal Stage:** Stores the files internally within Snowflake (User, Table and Named).
- **External Stage:** Stores the files in an external location (AWS S3 bucket or Azure Containers or GCP Cloud storage).

You can learn more about the **types of Snowflake Stages** from our previous article here.

2. How to Load Data using COPY INTO Command in Snowflake?

The following is the basic syntax of the **COPY INTO** command in Snowflake for loading files from the stage location into a database table.

```
COPY INTO target_table
FROM @stage_name/file_path
FILE_FORMAT = (FORMAT_NAME = 'my_file_format' | TYPE = filetype [formatTypeOptions] )
;
```

Key Components:

- **target_table**: The table where data is to be loaded.
- **@stage_name/file_path**: The location of the source file in a Snowflake stage.
- **FILE_FORMAT**: Defines the format of the input file.
- **FORMAT_NAME**: Specifies an existing named file format for loading data into the table.
- **TYPE**: Specifies the type of files (CSV, JSON, AVRO, ORC, PARQUET or XML) to load into the table. When file type is specified, additional format-specific options can be specified.

> *Note that only the **FORMAT_NAME** or **TYPE** of the file should be specified.*

Examples:

The following example loads data from the **input.csv** file in the internal named stage **my_int_stage** into **my_table** using a named file format.

```
COPY INTO my_table
FROM @my_int_stage/input.csv
FILE_FORMAT = (FORMAT_NAME = 'my_csv_format');
```

The following example loads data from all files from the inbox directory in the external stage named **my_ext_stage** into **my_table** using a file type with format-specific options.

```
COPY INTO my_table
FROM @my_ext_stage/inbox/
FILE_FORMAT = (TYPE = CSV FIELD_DELIMITER = ',' SKIP_HEADER = 1);
```

3. How to Load Data using Pattern Matching with COPY INTO Command in Snowflake?

The **COPY INTO** command supports regular expressions to match and load specific file names using the **PATTERN** keyword.

The following example loads all CSV files containing "**employees**" in their filename using the COPY INTO command.

```
COPY INTO employees
FROM @my_ext_stage
PATTERN = '.*employees.*.csv'
FILE_FORMAT = (FORMAT_NAME = 'my_csv_format');
```

4. How to Load Specific List of Files using the COPY INTO Command in Snowflake?

The **COPY INTO** command supports loading a specific list of files by defining them with the **FILES** keyword.

The following example loads only the specified CSV files using the COPY INTO command.

```
COPY INTO my_table
FROM @my_ext_stage
FILES = ('file1.csv', 'file2.csv')
FILE_FORMAT = (FORMAT_NAME = 'my_csv_format');
```

> *It is recommended not to use **FILES** and **PATTERN** options together. When both are specified, only the files specified using the FILES option are loaded.*

5. How to Transform Data while loading using COPY INTO Command in Snowflake?

The **COPY INTO** command supports loading data of only the required columns from stage files and transforming them before loading into a table using a **SELECT** statement.

Example Scenario

Suppose we have an employee data file staged in an external location. Instead of loading all columns, we only need `EMPLOYEE_ID`, `FULL_NAME` (concatenated first and last name), `HIRE_DATE` (formatted as YYYY-MM-DD), and `EMAIL`.

```
-- Sample Staged File (@my_ext_stage/employee.csv)
DEPARTMENT_ID,EMPLOYEE_ID,FIRST_NAME,LAST_NAME,EMAIL,PHONE_NUMBER,HIRE_DATE
```

```
14,101,Peter,Parker,peter.parker@outlook.com,393436,10/13/2018 11:25:00
62,124,Bruce,Banner,bruce.banner@outlook.com,906468,07/02/2018 10:35:02
```

Transforming and Loading Data Using COPY INTO

The following example selects and loads only the required columns from the stage files while applying transformations.

```
COPY INTO employee
FROM (
  SELECT
    $2::NUMBER,
    CONCAT($3,' ',$4),
    TO_DATE(SUBSTR($7,1,10),'MM/DD/YYYY'),
    $5
  FROM @my_ext_stage)
FILE_FORMAT = (FORMAT_NAME = 'my_csv_format')
;
```

Expected Output in the Employee Table

After executing the COPY INTO command, the transformed data will be loaded into the employee table as shown below.

```
+------------+---------------+------------+-----------------------------+
| EMPLOYEE_ID | FULL_NAME     | HIRE_DATE  | EMAIL                       |
+------------+---------------+------------+-----------------------------+
| 101         | Peter Parker  | 2018-10-13 | peter.parker@outlook.com    |
| 124         | Bruce Banner  | 2018-07-02 | bruce.banner@outlook.com    |
+------------+---------------+------------+-----------------------------+
```

6. How to Handle Errors while using COPY INTO Command in Snowflake?

The **COPY INTO** command provides several options to handle errors encountered due to data inconsistencies or format mismatches while copying data using the **ON_ERROR** keyword.

Syntax:

```
COPY INTO my_table
FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_file_format')
ON_ERROR = CONTINUE | SKIP_FILE | SKIP_FILE_num | 'SKIP_FILE_num%' | ABORT_STATEMENT
;
```

Error Handling Options:

- **CONTINUE**
  The COPY statement continues to load the files even if errors are encountered.
- **SKIP_FILE**
  The COPY statement skips the file when an error is encountered.
- **SKIP_FILE_num (e.g. SKIP_FILE_5)**
  The COPY statement skips a file when the number of error rows encountered in the file is equal to more than the number specified.
- **‘SKIP_FILE_num%’ (e.g. ‘SKIP_FILE_10%’)**
  The COPY statement skips a file when the percentage of error rows encountered in the file is equal to more than the percentage specified.
- **ABORT_STATEMENT**
  The COPY statement aborts the load operation if any error is encountered in a file.

7. How to Validate Files before loading using COPY INTO Command in Snowflake?

The **COPY INTO** command allows validating files without loading them into a table using the **VALIDATION_MODE** keyword. It tests the files for errors and returns results based on with validation mode specified.

Syntax:

```
COPY INTO my_table
FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_file_format')
VALIDATION_MODE = RETURN_n_ROWS | RETURN_ERRORS | RETURN_ALL_ERRORS
;
```

Validation Mode Options:

- **RETURN_n_ROWS (e.g. RETURN_10_ROWS)**
  The COPY statement validates and returns the specified number of rows if no errors are encountered.
- **RETURN_ERRORS**
  The COPY statement returns all errors across all the specified files.
- **RETURN_ALL_ERRORS**
  The COPY statement returns all errors across all the specified files, including those from files that were partially loaded in a previous run where the ON_ERROR option was set to CONTINUE.

> *Note that **VALIDATION_MODE** does not support **COPY** statements that transform data during a load.*

8. What are Copy Options in COPY INTO Command in Snowflake?

The **COPY INTO** command supports several "**Copy Options**" that help in controlling the load behaviour and performance of the copy statement.

8.1. SIZE_LIMIT

The **SIZE_LIMIT** option in the **COPY INTO** statement specifies the maximum size (in bytes) of data to be loaded for a given COPY statement. The COPY operation discontinues when the specified threshold is reached.

- The **SIZE_LIMIT** value applies to the entire set of files processed by the COPY statement, not to each individual file.
- The default value is **NULL** i.e. all files that meet the COPY criteria are loaded regardless of their size.
- 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.

Syntax:

```
COPY INTO my_table
FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_file_format')
SIZE_LIMIT = 10000000 --(10MB)
```

8.2. PURGE

The **PURGE** option in the **COPY INTO** statement specifies whether to remove the data files from the stage automatically after the data is loaded successfully.

- The **PURGE** option can be **TRUE** or **FALSE**. Default is **FALSE**.
- Note that if the purge operation fails for any reason, no error is returned currently.

Syntax:

```
COPY INTO my_table
FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_file_format')
PURGE = TRUE
```

8.3. FORCE

The **FORCE** option in the **COPY INTO** statement forces the loading of files, even if they were previously loaded and remain unchanged.

- The **FORCE** option can be **TRUE** or **FALSE**. Default is **FALSE**.
- Note that enabling this option may result in duplicate data being loaded into the table.

Syntax:

```
COPY INTO my_table
FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_file_format')
FORCE = TRUE
```

8.4. TRUNCATECOLUMNS

The **TRUNCATECOLUMNS** option in the **COPY INTO** statement controls whether to truncate text strings that exceed the target column length.

- The **TRUNCATECOLUMNS** option can be **TRUE** or **FALSE**. Default is **FALSE**.
- If **TRUE**, strings exceeding the target column length are automatically truncated.
- If **FALSE**, the COPY statement returns an error if a string exceeds the target column length.

Syntax:

```
COPY INTO my_table
FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_file_format')
TRUNCATECOLUMNS = TRUE
```

8.5. ENFORCE_LENGTH

The **ENFORCE_LENGTH** option in the **COPY INTO** statement controls whether to truncate text strings that exceed the target column length.

- The **ENFORCE_LENGTH** option can be **TRUE** or **FALSE**. Default is **FALSE**.
- If **TRUE**, the COPY statement returns an error if a string exceeds the target column length.
- If **FALSE**, strings exceeding the target column length are automatically truncated.

Syntax:

```
COPY INTO my_table
FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_file_format')
ENFORCE_LENGTH = TRUE
```

> *Note that **TRUNCATECOLUMNS** and **ENFORCE_LENGTH** have the same functionality but "opposite" behaviour. Only one of these parameters needs to be included in a COPY statement to achieve the desire*

> *outcome.*

8.6. MATCH_BY_COLUMN_NAME

The **MATCH_BY_COLUMN_NAME** option in the **COPY INTO** statement ensures that data is loaded into the columns in the target table that match the corresponding column names in the staged data.

Syntax:

```
COPY INTO my_table
FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_file_format')
MATCH_BY_COLUMN_NAME = CASE_SENSITIVE | CASE_INSENSITIVE | NONE
```

8.6.1. Column Match Criteria

- The column name in the data file must exactly match a column name in the target table.
- The target table column must have a compatible data type.
- Column order does not matter.
- Case sensitivity for column name matching is determined by **CASE_SENSITIVE** or **CASE_INSENSITIVE** values.

8.6.2. Load Criteria

The **COPY** operation verifies that at least one column in the target table matches a column in the data files.

- If a match is found, values from the data files are loaded into the respective columns.
- If no match is found, NULL values are inserted into the table for each record in the data files.

8.6.3. Handling Additional Non-Matching Columns

- Extra columns in the data files that do not match table columns are ignored.
- Extra columns in the target table that are not in the data files are populated with NULL values. These columns must allow NULL values.

8.6.4. Limitations

- The **MATCH_BY_COLUMN_NAME** option is not supported when using a **SELECT** statement for data transformation.
- It cannot be used with the **VALIDATION_MODE** parameter for staged data validation.
- **XML** data format is not supported with this option.

8.7. INCLUDE_METADATA

The **INCLUDE_METADATA** option in the **COPY INTO** statement maps **METADATA$** columns to existing target table column names. This option must only be used with the **MATCH_BY_COLUMN_NAME** option.

The following are the supported **METADATA$** Columns.

- **METADATA$FILENAME**
  Name of the staged data file the current row belongs to, including the full file path.
- **METADATA$FILE_ROW_NUMBER**
  Row number for each record in the staged data file.
- **METADATA$FILE_CONTENT_KEY**
  Checksum of the staged data file the current row belongs to.
- **METADATA$FILE_LAST_MODIFIED**
  Last modified timestamp of the staged data file.
- **METADATA$START_SCAN_TIME**
  Start timestamp of the operation for each record in the staged data file.

Syntax:

For each row loaded into the target table, the **INCLUDE_METADATA** option maps **METADATA$** columns to the target columns `file_name` and `load_date_time`, as shown in the example below.

```
COPY INTO my_table FROM @my_stage
FILE_FORMAT = (FORMAT_NAME = 'my_file_format')
MATCH_BY_COLUMN_NAME = CASE_INSENSITIVE
INCLUDE_METADATA = (
  file_name = METADATA$FILENAME,
  Load_date_time = METADATA$START_SCAN_TIME
);
```

9. Summary

The **COPY INTO** command in Snowflake is a powerful tool for efficiently loading data from staged files into tables. Key features include:

- **Flexible File Handling:** Supports loading from internal and external stages.
- **File Format Support:** Works with CSV, JSON, AVRO, ORC, PARQUET, and XML formats.
- **Data Selection & Transformation:** Allows selecting specific columns and applying transformations during load.
- **Pattern Matching & File Selection:** Enables loading specific files using **PATTERN** or **FILES** options.
- **Error Handling:** Provides multiple strategies like **CONTINUE**, **SKIP_FILE**, and **ABORT_STATEMENT**.
- **Validation Mode:** Allows checking file integrity before loading using **VALIDATION_MODE**.
- **Metadata Mapping:** Maps metadata columns (e.g., filename, row number) using **INCLUDE_METADATA**.
- **Performance Optimization:** Includes options like **SIZE_LIMIT**, **FORCE**, and **MATCH_BY_COLUMN_NAME**.
- **File Management:** Supports automatic file removal from stages using **PURGE**.

By leveraging these features, users can optimize data ingestion workflows, ensuring accuracy and efficiency in Snowflake.

Related Articles:

- Snowflake Micro-partitions & Data Clustering
- Snowflake File Formats