COPY INTO *<table>*

Loads data from staged files to an existing table. The files must already be staged in one of the following locations:

- Named internal stage (or table/user stage). Files can be staged using the PUT command.
- Named external stage that references an external location (Amazon S3, Google Cloud Storage, or Microsoft Azure).

  You cannot access data held in archival cloud storage classes that requires restoration before it can be retrieved. These archival storage classes include, for example, the Amazon S3 Glacier Flexible Retrieval or Glacier Deep Archive storage class, or Microsoft Azure Archive Storage.
- External location (Amazon S3, Google Cloud Storage, or Microsoft Azure).

See also:

COPY INTO <location>

Syntax

```
/* Standard data load */
COPY INTO [<namespace>.]<table_name>
     FROM { internalStage | externalStage | externalLocation }
[ FILES = ( '<file_name>' [ , '<file_name>' ] [ , ... ] ) ]
[ PATTERN = '<regex_pattern>' ]
[ FILE_FORMAT = ( { FORMAT_NAME = '[<namespace>.]<file_format_name>' |
                    TYPE = { CSV | JSON | AVRO | ORC | PARQUET | XML } [
formatTypeOptions ] } ) ]
[ copyOptions ]
[ VALIDATION_MODE = RETURN_<n>_ROWS | RETURN_ERRORS | RETURN_ALL_ERRORS ]

/* Data load with transformation */
COPY INTO [<namespace>.]<table_name> [ ( <col_name> [ , <col_name> ... ] ) ]
     FROM ( SELECT [<alias>.]$<file_col_num>[.<element>] [ ,
[<alias>.]$<file_col_num>[.<element>] ... ]
            FROM { internalStage | externalStage } )
[ FILES = ( '<file_name>' [ , '<file_name>' ] [ , ... ] ) ]
[ PATTERN = '<regex_pattern>' ]
[ FILE_FORMAT = ( { FORMAT_NAME = '[<namespace>.]<file_format_name>' |
                    TYPE = { CSV | JSON | AVRO | ORC | PARQUET | XML } [
formatTypeOptions ] } ) ]
[ copyOptions ]
```

Where:

```
internalStage ::=
    @[<namespace>.]<int_stage_name>[/<path>]
  | @[<namespace>.]%<table_name>[/<path>]
  | @~[/<path>]
```

```
externalStage ::=
  @[<namespace>.]<ext_stage_name>[/<path>]
```

```
externalLocation (for Amazon S3) ::=
  '<protocol>://<bucket>[/<path>]'
  [ { STORAGE_INTEGRATION = <integration_name> } | { CREDENTIALS = ( {  { 
AWS_KEY_ID = '<string>' AWS_SECRET_KEY = '<string>' [ AWS_TOKEN = 
'<string>' ] } } ) } ]
  [ ENCRYPTION = ( [ TYPE = 'AWS_CSE' ] [ MASTER_KEY = '<string>' ] |
                   [ TYPE = 'AWS_SSE_S3' ] |
                   [ TYPE = 'AWS_SSE_KMS' [ KMS_KEY_ID = '<string>' ] ] |
                   [ TYPE = 'NONE' ] ) ]
```

```
externalLocation (for Google Cloud Storage) ::=
  'gcs://<bucket>[/<path>]'
  [ STORAGE_INTEGRATION = <integration_name> ]
  [ ENCRYPTION = ( [ TYPE = 'GCS_SSE_KMS' ] [ KMS_KEY_ID = '<string>' ] | 
[ TYPE = 'NONE' ] ) ]
```

```
externalLocation (for Microsoft Azure) ::=
  'azure://<account>.blob.core.windows.net/<container>[/<path>]'
  [ { STORAGE_INTEGRATION = <integration_name> } | { CREDENTIALS = ( [ 
AZURE_SAS_TOKEN = '<string>' ] ) } ]
  [ ENCRYPTION = ( [ TYPE = { 'AZURE_CSE' | 'NONE' } ] [ MASTER_KEY = 
'<string>' ] ) ]
```

```
formatTypeOptions ::=
-- If FILE_FORMAT = ( TYPE = CSV ... )
     COMPRESSION = AUTO | GZIP | BZ2 | BROTLI | ZSTD | DEFLATE | 
RAW_DEFLATE | NONE
     RECORD_DELIMITER = '<string>' | NONE
     FIELD_DELIMITER = '<string>' | NONE
     PARSE_HEADER = TRUE | FALSE
     SKIP_HEADER = <integer>
     SKIP_BLANK_LINES = TRUE | FALSE
     DATE_FORMAT = '<string>' | AUTO
     TIME_FORMAT = '<string>' | AUTO
     TIMESTAMP_FORMAT = '<string>' | AUTO
     BINARY_FORMAT = HEX | BASE64 | UTF8
     ESCAPE = '<character>' | NONE
     ESCAPE_UNENCLOSED_FIELD = '<character>' | NONE
     TRIM_SPACE = TRUE | FALSE
     FIELD_OPTIONALLY_ENCLOSED_BY = '<character>' | NONE
     NULL_IF = ( '<string>' [ , '<string>' ... ] )
     ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE | FALSE
     REPLACE_INVALID_CHARACTERS = TRUE | FALSE
     EMPTY_FIELD_AS_NULL = TRUE | FALSE
     SKIP_BYTE_ORDER_MARK = TRUE | FALSE
     ENCODING = '<string>' | UTF8
-- If FILE_FORMAT = ( TYPE = JSON ... )
     COMPRESSION = AUTO | GZIP | BZ2 | BROTLI | ZSTD | DEFLATE |
RAW_DEFLATE | NONE
     DATE_FORMAT = '<string>' | AUTO
     TIME_FORMAT = '<string>' | AUTO
     TIMESTAMP_FORMAT = '<string>' | AUTO
     BINARY_FORMAT = HEX | BASE64 | UTF8
     TRIM_SPACE = TRUE | FALSE
     NULL_IF = ( '<string>' [ , '<string>' ... ] )
     ENABLE_OCTAL = TRUE | FALSE
     ALLOW_DUPLICATE = TRUE | FALSE
     STRIP_OUTER_ARRAY = TRUE | FALSE
     STRIP_NULL_VALUES = TRUE | FALSE
     REPLACE_INVALID_CHARACTERS = TRUE | FALSE
     IGNORE_UTF8_ERRORS = TRUE | FALSE
     SKIP_BYTE_ORDER_MARK = TRUE | FALSE
-- If FILE_FORMAT = ( TYPE = AVRO ... )
     COMPRESSION = AUTO | GZIP | BROTLI | ZSTD | DEFLATE | RAW_DEFLATE |
NONE
     TRIM_SPACE = TRUE | FALSE
     REPLACE_INVALID_CHARACTERS = TRUE | FALSE
     NULL_IF = ( '<string>' [ , '<string>' ... ] )
-- If FILE_FORMAT = ( TYPE = ORC ... )
     TRIM_SPACE = TRUE | FALSE
     REPLACE_INVALID_CHARACTERS = TRUE | FALSE
     NULL_IF = ( '<string>' [ , '<string>' ... ] )
-- If FILE_FORMAT = ( TYPE = PARQUET ... )
     COMPRESSION = AUTO | SNAPPY | NONE
     BINARY_AS_TEXT = TRUE | FALSE
     USE_LOGICAL_TYPE = TRUE | FALSE
     TRIM_SPACE = TRUE | FALSE
     REPLACE_INVALID_CHARACTERS = TRUE | FALSE
     NULL_IF = ( '<string>' [ , '<string>' ... ] )
-- If FILE_FORMAT = ( TYPE = XML ... )
     COMPRESSION = AUTO | GZIP | BZ2 | BROTLI | ZSTD | DEFLATE |
RAW_DEFLATE | NONE
     IGNORE_UTF8_ERRORS = TRUE | FALSE
     PRESERVE_SPACE = TRUE | FALSE
     STRIP_OUTER_ELEMENT = TRUE | FALSE
     DISABLE_SNOWFLAKE_DATA = TRUE | FALSE
     DISABLE_AUTO_CONVERT = TRUE | FALSE
     REPLACE_INVALID_CHARACTERS = TRUE | FALSE
     SKIP_BYTE_ORDER_MARK = TRUE | FALSE
```

```
copyOptions ::=
     ON_ERROR = { CONTINUE | SKIP_FILE | SKIP_FILE_<num> |
'SKIP_FILE_<num>%' | ABORT_STATEMENT }
     SIZE_LIMIT = <num>
     PURGE = TRUE | FALSE
     RETURN_FAILED_ONLY = TRUE | FALSE
     MATCH_BY_COLUMN_NAME = CASE_SENSITIVE | CASE_INSENSITIVE | NONE
     INCLUDE_METADATA = ( <column_name> = METADATA$<field> [ ,
<column_name> = METADATA${field} ... ] )
     ENFORCE_LENGTH = TRUE | FALSE
     TRUNCATECOLUMNS = TRUE | FALSE
     FORCE = TRUE | FALSE
     LOAD_UNCERTAIN_FILES = TRUE | FALSE
     FILE_PROCESSOR = (SCANNER = <custom_scanner_type> SCANNER_OPTIONS =
(<scanner_options>))
```

Required parameters

`[<namespace>.]<table_name>`

Specifies the name of the table into which data is loaded.

Namespace optionally specifies the database and/or schema for the table, in the form of `<database_name>.<schema_name>` or `<schema_name>`. It is ***optional*** if a database and schema are currently in use within the user session; otherwise, it is required.

`FROM ...`

Specifies the internal or external location where the files containing data to be loaded are staged:

`@[<namespace>.]<int_stage_name>[/<path>]`	Files are in the specified named internal stage.
`@[<namespace>.]<ext_stage_name>[/<path>]`	Files are in the specified named external stage.
`@[<namespace>.]%<table_name>[/<path>]`	Files are in the stage for the specified table.
`@~[/<path>]`	Files are in the stage for the current user.
`'<protocol>://<bucket>[/<path>]'`	Files are in the specified external location (S3

	bucket). Additional parameters might be required. For details, see Additional Cloud Provider Parameters (in this topic).
`'gcs://<bucket>[/<path>]'`	Files are in the specified external location (Google Cloud Storage bucket). Additional parameters could be required. For details, see Additional Cloud Provider Parameters (in this topic).
`'azure://<account>.blob.core.windows.net/<container>[/<path>]'`	Files are in the specified external location (Azure container). Additional parameters might be required. For details, see Additional Cloud Provider Parameters (in this topic).

Where:

- `<namespace>` is the database and/or schema in which the internal or external stage resides, in the form of `<database_name>.<schema_name>` or `<schema_name>`. It is ***optional*** if a database and schema are currently in use within the user session; otherwise, it is required.
- `<protocol>` is one of the following:
  - `s3` refers to S3 storage in public AWS regions outside of China.
  - `s3china` refers to S3 storage in public AWS regions in China.
  - `s3gov` refers to S3 storage in government regions.

  Accessing cloud storage in a government region using a storage integration is limited to Snowflake accounts hosted in the same government region.

  Similarly, if you need to access cloud storage in a region in China, you can use a storage integration only from a Snowflake account hosted in the same region in China.

  In these cases, use the CREDENTIALS parameter in the CREATE STAGE command (rather than using a storage integration) to provide the credentials for authentication.
- `<bucket>` is the name of the bucket.
- `<account>` is the name of the Azure account (e.g. `myaccount`). Use the `blob.core.windows.net` endpoint for all supported types of Azure blob storage accounts, including Data Lake Storage Gen2.

  Note that currently, accessing Azure blob storage in government regions using a storage integration is limited to Snowflake accounts hosted on Azure in the same government region. Accessing your blob storage from an account hosted outside of the government region using direct credentials is supported.

- `<container>` is the name of the Azure container (e.g. `mycontainer`).
- `<path>` is an optional case-sensitive path for files in the cloud storage location (i.e. files have names that begin with a common string) that limits the set of files to load. Paths are alternatively called *prefixes* or *folders* by different cloud storage services.

  Relative path modifiers such as `/./` and `/../` are interpreted literally because “paths” are literal prefixes for a name. For example:

  ```
  -- S3 bucket
  COPY INTO mytable FROM 's3://mybucket/./../a.csv';

  -- Google Cloud Storage bucket
  COPY INTO mytable FROM 'gcs://mybucket/./../a.csv';

  -- Azure container
  COPY INTO mytable FROM
  'azure://myaccount.blob.core.windows.net/mycontainer/./../a.csv';
  ```

  In these COPY statements, Snowflake looks for a file literally named `./../a.csv` in the external location.

Note

- If the internal or external stage or path name includes special characters, including spaces, enclose the `FROM ...` string in single quotes.
- The `FROM ...` value must be a literal constant. The value cannot be a SQL variable.

Additional cloud provider parameters

`STORAGE_INTEGRATION = <integration_name>` *or*
`CREDENTIALS = ( <cloud_specific_credentials> )`

Supported when the FROM value in the COPY statement is an external storage URI rather than an external stage name.

Required only for loading from an external private/protected cloud storage location; not required for public buckets/containers

Specifies the security credentials for connecting to the cloud provider and accessing the private/protected storage container where the data files are staged.

Amazon S3

`STORAGE_INTEGRATION = <integration_name>`

Specifies the name of the storage integration used to delegate authentication responsibility for external cloud storage to a Snowflake identity and access management (IAM) entity. For more details, see CREATE STORAGE INTEGRATION.

> **Note**
> We highly recommend the use of storage integrations. This option avoids the need to supply cloud storage credentials using the CREDENTIALS parameter when creating stages or loading data.

`CREDENTIALS = ( AWS_KEY_ID = '<string>' AWS_SECRET_KEY = '<string>' [ AWS_TOKEN = '<string>' ] )`

Specifies the security credentials for connecting to AWS and accessing the private/protected S3 bucket where the files to load are staged. For more information, see Configuring secure access to Amazon S3.

The credentials you specify depend on whether you associated the Snowflake access permissions for the bucket with an AWS IAM (Identity & Access Management) user or role:

- **IAM user:** Temporary IAM credentials are required. Temporary (aka “scoped”) credentials are generated by AWS Security Token Service (STS) and consist of three components:
  - `AWS_KEY_ID`
  - `AWS_SECRET_KEY`
  - `AWS_TOKEN`

  All three are ***required*** to access a private/protected bucket. After a designated period of time, temporary credentials expire and can no longer be used. You must then generate a new set of valid temporary credentials.

  > **Important**
  > COPY commands contain complex syntax and sensitive information, such as credentials. In addition, they are executed frequently and are often stored in scripts or worksheets, which could lead to sensitive information being inadvertently exposed. The COPY command allows permanent (aka “long-term”) credentials to be used; however, for security reasons, do not use ***permanent*** credentials in COPY commands. Instead, use temporary credentials.

If you must use permanent credentials, use external stages, for which credentials are entered once and securely stored, minimizing the potential for exposure.

- **IAM role:** Omit the security credentials and access keys and, instead, identify the role using `AWS_ROLE` and specify the AWS role ARN (Amazon Resource Name).

Google Cloud Storage

`STORAGE_INTEGRATION = <integration_name>`

Specifies the name of the storage integration used to delegate authentication responsibility for external cloud storage to a Snowflake identity and access management (IAM) entity. For more details, see CREATE STORAGE INTEGRATION.

Microsoft Azure

`STORAGE_INTEGRATION = <integration_name>`

Specifies the name of the storage integration used to delegate authentication responsibility for external cloud storage to a Snowflake identity and access management (IAM) entity. For more details, see CREATE STORAGE INTEGRATION.

Note

We highly recommend the use of storage integrations. This option avoids the need to supply cloud storage credentials using the CREDENTIALS parameter when creating stages or loading data.

`CREDENTIALS = ( AZURE_SAS_TOKEN = '<string>' )`

Specifies the SAS (shared access signature) token for connecting to Azure and accessing the private/protected container where the files containing data are staged. Credentials are generated by Azure.

`ENCRYPTION = ( <cloud_specific_encryption> )`

For use in ad hoc COPY statements (statements that do not reference a named external stage). Required only for loading from encrypted files; not required if files are unencrypted. Specifies the encryption settings used to decrypt encrypted files in the storage location.

Amazon S3

```
ENCRYPTION = ( [ TYPE = 'AWS_CSE' ] [ MASTER_KEY = '<string>' ] | [ TYPE =
'AWS_SSE_S3' ] | [ TYPE = 'AWS_SSE_KMS' [ KMS_KEY_ID = '<string>' ] ] | [ TYPE
= 'NONE' ] )
```

`TYPE = ...`

Specifies the encryption type used. Possible values are:

- `AWS_CSE`: Client-side encryption (requires a `MASTER_KEY` value). Currently, the client-side master key you provide can only be a symmetric key. Note that, when a `MASTER_KEY` value is provided, Snowflake assumes `TYPE = AWS_CSE` (i.e. when a `MASTER_KEY` value is provided, `TYPE` is not required).
- `AWS_SSE_S3`: Server-side encryption that requires no additional encryption settings.
- `AWS_SSE_KMS`: Server-side encryption that accepts an optional `KMS_KEY_ID` value.
- `NONE`: No encryption.

For more information about the encryption types, see the AWS documentation for client-side encryption or server-side encryption.

`MASTER_KEY = '<string>'` **(applies to `AWS_CSE` encryption only)**

Specifies the client-side master key used to encrypt the files in the bucket. The master key must be a 128-bit or 256-bit key in Base64-encoded form.

`KMS_KEY_ID = '<string>'` **(applies to `AWS_SSE_KMS` encryption only)**

Optionally specifies the ID for the AWS KMS-managed key used to encrypt files ***unloaded*** into the bucket. If no value is provided, your default KMS key ID is used to encrypt files on unload.

Note that this value is ignored for data loading.

Google Cloud Storage

```
ENCRYPTION = ( [ TYPE = 'GCS_SSE_KMS' | 'NONE' ] [ KMS_KEY_ID = '<string>' ]
)
```

`TYPE = ...`

Specifies the encryption type used. Possible values are:

- `GCS_SSE_KMS`: Server-side encryption that accepts an optional `KMS_KEY_ID` value.

  For more information, see the Google Cloud Platform documentation:

        - https://cloud.google.com/storage/docs/encryption/customer-managed-keys
        - https://cloud.google.com/storage/docs/encryption/using-customer-managed-keys
    - `NONE`: No encryption.

`KMS_KEY_ID = '<string>'` **(applies to `GCS_SSE_KMS` encryption only)**

Optionally specifies the ID for the Cloud KMS-managed key that is used to encrypt files ***unloaded*** into the bucket. If no value is provided, your default KMS key ID set on the bucket is used to encrypt files on unload.

Note that this value is ignored for data loading. The load operation should succeed if the service account has sufficient permissions to decrypt data in the bucket.

Microsoft Azure

```
ENCRYPTION = ( [ TYPE = 'AZURE_CSE' | 'NONE' ] [ MASTER_KEY = '<string>' ] )
```

`TYPE = ...`

Specifies the encryption type used. Possible values are:

- `AZURE_CSE`: Client-side encryption (requires a MASTER_KEY value). For information, see the Client-side encryption information in the Microsoft Azure documentation.
- `NONE`: No encryption.

`MASTER_KEY = '<string>'` **(applies to AZURE_CSE encryption only)**

Specifies the client-side master key used to decrypt files. The master key must be a 128-bit or 256-bit key in Base64-encoded form.

Transformation parameters

```
( SELECT [<alias>.]$<file_col_num>[.<element>] [ , [<alias>.]$<file_col_num>[.<element>] ... ] FROM ... [ <alias> ] )
```

Required for transforming data during loading

Specifies an explicit set of fields/columns (separated by commas) to load from the staged data files. The fields/columns are selected from the files using a standard SQL query (i.e. SELECT list), where:

`<alias>`	Specifies an optional alias for the `FROM` value (e.g. `d` in `COPY INTO t1 (c1) FROM (SELECT d.$1 FROM @mystage/file1.csv.gz d);`).
`<file_col_num>`	Specifies the positional number of the field/column (in the file) that contains the data to be loaded (`1` for the first field, `2` for the second field, etc.)
`<element>`	Specifies the path and element name of a repeating value in the data file (applies only to ***semi-structured data files***).

The SELECT list defines a numbered set of field/columns in the data files you are loading from. The list ***must*** match the sequence of columns in the target table. You can use the optional `( <col_name> [ , <col_name> ... ] )` parameter to map the list to specific columns in the target table.

Note that the actual field/column order in the data files can be different from the column order in the target table. It is only important that the SELECT list maps fields/columns in the data files to the ***corresponding*** columns in the table.

Note
The SELECT statement used for transformations does not support all functions. For a complete list of the supported functions and more details about data loading transformations, including examples, see the usage notes in Transforming data during a load.

Also, data loading transformation ***only*** supports selecting data from user stages and named stages (internal or external).

`( <col_name> [ , <col_name> ... ] )`

Optionally specifies an explicit list of table columns (separated by commas) into which you want to insert data:

- The first column consumes the values produced from the first field/column extracted from the loaded files.
- The second column consumes the values produced from the second field/column extracted from the loaded files.
- And so on, in the order specified.

Columns cannot be repeated in this listing. Any columns excluded from this column list are populated by their default value (NULL, if not specified). However, excluded columns ***cannot*** have a sequence as their default value.

Optional parameters

`FILES = ( '<file_name>' [ , '<file_name>' ... ] )`

Specifies a list of one or more files names (separated by commas) to be loaded. The files must already have been staged in either the Snowflake internal location or external location specified in the command. If any of the specified files cannot be found, the default behavior `ON_ERROR = ABORT_STATEMENT` aborts the load operation unless a different `ON_ERROR` option is explicitly set in the COPY statement.

The maximum number of files names that can be specified is 1000.

> **Note**
> For external stages only (Amazon S3, Google Cloud Storage, or Microsoft Azure), the file path is set by concatenating the URL in the stage definition and the list of resolved file names.
>
> However, Snowflake doesn’t insert a separator implicitly between the path and file names. You must explicitly include a separator (`/`) either at the end of the URL in the stage definition or at the beginning of each file name specified in this parameter.

`PATTERN = '<regex_pattern>'`

A regular expression pattern string, enclosed in single quotes, specifying the file names and/or paths to match.

> **Tip**
> For the best performance, try to avoid applying patterns that filter on a large number of files.

Note that the regular expression is applied differently to bulk data loads versus Snowpipe data loads.

- Snowpipe trims any path segments in the stage definition from the storage location and applies the regular expression to any remaining path segments and filenames. To view the stage definition, execute the DESCRIBE STAGE command for the stage. The URL property consists of the bucket or container name and zero or more path segments. For example, if the FROM location in a COPY INTO *<table>* statement is `@s/path1/path2/` and the URL value for stage `@s` is `s3://mybucket/path1/`, then Snowpipe trims `/path1/` from the storage location in the FROM clause and applies the regular expression to `path2/` plus the filenames in the path.
- Bulk data load operations apply the regular expression to the entire storage location in the FROM clause.

Note

When the `FILES` and `PATTERN` options are used together, only the specified paths in the `FILES` option are loaded. It is recommended to not use these two options together.

`FILE_FORMAT = ( FORMAT_NAME = '<file_format_name>' )` *or*
`FILE_FORMAT = ( TYPE = CSV | JSON | AVRO | ORC | PARQUET | XML [ ... ] )`

Specifies the format of the data files to load:

`FORMAT_NAME = '<file_format_name>'`

Specifies an existing named file format to use for loading data into the table. The named file format determines the format type (CSV, JSON, etc.), as well as any other format options, for the data files. For more information, see CREATE FILE FORMAT.

`TYPE = CSV | JSON | AVRO | ORC | PARQUET | XML [ ... ]`

Specifies the type of files to load into the table. If a format type is specified, then additional format-specific options can be specified. For more details, see Format Type Options (in this topic).

Note

`FORMAT_NAME` and `TYPE` are mutually exclusive; specifying both in the same COPY command might result in unexpected behavior.

`COPY_OPTIONS = ( ... )`

Specifies one or more copy options for the loaded data. For more details, see Copy Options (in this topic).

`VALIDATION_MODE = RETURN_<n>_ROWS | RETURN_ERRORS | RETURN_ALL_ERRORS`

String (constant) that instructs the COPY command to validate the data files ***instead*** of loading them into the specified table; i.e. the COPY command tests the files for errors but does not load them. The command validates the data to be loaded and returns results based on the validation option specified:

Supported Values	Notes
`RETURN_<n>_ROWS` (e.g. `RETURN_10_ROWS`)	Validates the specified number of rows, if no errors are encountered; otherwise, fails at the first error encountered in the rows.
`RETURN_ERRORS`	Returns all errors (parsing, conversion, etc.) across all files specified in the COPY statement.
`RETURN_ALL_ERRORS`	Returns all errors across all files specified in the COPY statement, including files with errors that were partially loaded during an earlier load because the `ON_ERROR` copy option was set to `CONTINUE` during the load.

Note

- `VALIDATION_MODE` does not support COPY statements that transform data during a load. If the parameter is specified, the COPY statement returns an error.
- Use the VALIDATE table function to view all errors encountered during a previous load. Note that this function also does not support COPY statements that transform data during a load.

Format type options (`formatTypeOptions`)

Depending on the file format type specified (`FILE_FORMAT = ( TYPE = ... )`), you can include one or more of the following format-specific options (separated by blank spaces, commas, or new lines):

TYPE = CSV

`COMPRESSION = AUTO | GZIP | BZ2 | BROTLI | ZSTD | DEFLATE | RAW_DEFLATE | NONE`

String (constant) that specifies the current compression algorithm for the data files to be loaded. Snowflake uses this option to detect how ***already-compressed*** data files were compressed so that the compressed data in the files can be extracted for loading.

Supported Values	Notes
`AUTO`	Compression algorithm detected automatically, except for Brotli-compressed files, which cannot currently be detected automatically. If loading Brotli-compressed files, explicitly use `BROTLI` instead of `AUTO`.

Supported Values	Notes
`GZIP`	
`BZ2`	
`BROTLI`	Must be specified when loading Brotli-compressed files.
`ZSTD`	Zstandard v0.8 (and higher) supported.
`DEFLATE`	Deflate-compressed files (with zlib header, RFC1950).
`RAW_DEFLATE`	Raw Deflate-compressed files (without header, RFC1951).
`NONE`	Data files to load have not been compressed.

`RECORD_DELIMITER = '<string>' | NONE`

One or more characters that separate records in an input file. Accepts common escape sequences or the following singlebyte or multibyte characters:

Singlebyte characters	Octal values (prefixed by `\\`) or hex values (prefixed by `0x` or `\x`). For example, for records delimited by the circumflex accent (`^`) character, specify the octal (`\\136`) or hex (`0x5e`) value.
Multibyte characters	Hex values (prefixed by `\x`). For example, for records delimited by the cent (`¢`) character, specify the hex (`\xC2\xA2`) value. The delimiter for RECORD_DELIMITER or FIELD_DELIMITER cannot be a substring of the delimiter for the other file format option (e.g. `FIELD_DELIMITER = 'aa' RECORD_DELIMITER = 'aabb'`).

The specified delimiter must be a valid UTF-8 character and not a random sequence of bytes. Also note that the delimiter is limited to a maximum of 20 characters.

Also accepts a value of `NONE`.

Default: New line character. Note that “new line” is logical such that `\r\n` is understood as a new line for files on a Windows platform.

`FIELD_DELIMITER = '<string>' | NONE`

One or more singlebyte or multibyte characters that separate fields in an input file. Accepts common escape sequences or the following singlebyte or multibyte characters:

Singlebyte characters	Octal values (prefixed by `\\`) or hex values (prefixed by `0x` or `\x`). For example, for records delimited by the circumflex accent (`^`) character, specify the octal (`\\136`) or hex (`0x5e`) value.
Multibyte characters	Hex values (prefixed by `\x`). For example, for records delimited by the cent (`¢`) character, specify the hex (`\xC2\xA2`) value.

The delimiter for RECORD_DELIMITER or FIELD_DELIMITER cannot be a substring of the delimiter for the other file format option (e.g. `FIELD_DELIMITER = 'aa' RECORD_DELIMITER = 'aabb'`).

> **Note**
> For non-ASCII characters, you must use the hex byte sequence value to get a deterministic behavior.

The specified delimiter must be a valid UTF-8 character and not a random sequence of bytes. Also note that the delimiter is limited to a maximum of 20 characters.

Also accepts a value of `NONE`.

Default: comma (`,`)

`PARSE_HEADER = TRUE | FALSE`

Boolean that specifies whether to use the first row headers in the data files to determine column names.

This file format option is applied to the following actions only:

- Automatically detecting column definitions by using the INFER_SCHEMA function.
- Loading CSV data into separate columns by using the INFER_SCHEMA function and MATCH_BY_COLUMN_NAME copy option.

If the option is set to TRUE, the first row headers will be used to determine column names. The default value FALSE will return column names as c*, where * is the position of the column.

Note that the SKIP_HEADER option is not supported with PARSE_HEADER = TRUE.

Default: `FALSE`

`SKIP_HEADER = <integer>`

Number of lines at the start of the file to skip.

Note that SKIP_HEADER does not use the RECORD_DELIMITER or FIELD_DELIMITER values to determine what a header line is; rather, it simply skips the specified number of CRLF (Carriage Return, Line Feed)-delimited lines in the file. RECORD_DELIMITER and FIELD_DELIMITER are then used to determine the rows of data to load.

Default: `0`

`SKIP_BLANK_LINES = TRUE | FALSE`

Use	Data loading only
Definition	Boolean that specifies to skip any blank lines encountered in the data files; otherwise, blank lines produce an end-of-record error (default behavior).

Default: `FALSE`

`DATE_FORMAT = '<string>' | AUTO`

String that defines the format of date values in the data files to be loaded. If a value is not specified or is `AUTO`, the value for the DATE_INPUT_FORMAT session parameter is used.

Default: `AUTO`

`TIME_FORMAT = '<string>' | AUTO`

String that defines the format of time values in the data files to be loaded. If a value is not specified or is `AUTO`, the value for the TIME_INPUT_FORMAT session parameter is used.

Default: `AUTO`

`TIMESTAMP_FORMAT = '<string>' | AUTO`

String that defines the format of timestamp values in the data files to be loaded. If a value is not specified or is `AUTO`, the value for the TIMESTAMP_INPUT_FORMAT session parameter is used.

Default: `AUTO`

`BINARY_FORMAT = HEX | BASE64 | UTF8`

String (constant) that defines the encoding format for binary input or output. This option only applies when loading data into binary columns in a table.

Default: `HEX`

`ESCAPE = '<character>' | NONE`

Use Data loading and unloading

Definition A singlebyte character used as the escape character for enclosed field values only. An escape character invokes an alternative interpretation on subsequent characters in a character sequence. You can use the ESCAPE character to interpret instances of the `FIELD_OPTIONALLY_ENCLOSED_BY` character in the data as literals.

Accepts common escape sequences (e.g. `\t` for tab, `\n` for newline, `\r` for carriage return, `\\` for backslash), octal values, or hex values.

> **Note**
> This file format option supports singlebyte characters only. Note that UTF-8 character encoding represents high-order ASCII characters as multibyte characters. If your data file is encoded with the UTF-8 character set, you cannot specify a high-order ASCII character as the option value.
>
> In addition, if you specify a high-order ASCII character, we recommend that you set the `ENCODING = '<string>'` file format option as the character encoding for your data files to ensure the character is interpreted correctly.

Default `NONE`

`ESCAPE_UNENCLOSED_FIELD = '<character>' | NONE`

Use Data loading and unloading

Definition A singlebyte character used as the escape character for unenclosed field values only. An escape character invokes an alternative interpretation on subsequent characters in a character sequence. You can use the ESCAPE character to interpret instances of the `FIELD_DELIMITER` or `RECORD_DELIMITER` characters in the data as literals. The escape character can also be used to escape instances of itself in the data.

Accepts common escape sequences (e.g. `\t` for tab, `\n` for newline, `\r` for carriage return, `\\` for backslash), octal values, or hex values.

Note

- The default value is `\\`. If a row in a data file ends in the backslash (`\`) character, this character escapes the newline or carriage return character specified for the `RECORD_DELIMITER` file format option. As a result, the load operation treats this row and the next row as a single row of data. To avoid this issue, set the value to `NONE`.
- This file format option supports singlebyte characters only. Note that UTF-8 character encoding represents high-order ASCII characters as multibyte characters. If your data file is encoded with the UTF-8 character set, you cannot specify a high-order ASCII character as the option value.

  In addition, if you specify a high-order ASCII character, we recommend that you set the `ENCODING = '<string>'` file format option as the character encoding for your data files to ensure the character is interpreted correctly.

Default backslash (`\\`)

`TRIM_SPACE = TRUE | FALSE`

Boolean that specifies whether to remove white space from fields.

For example, if your external database software encloses fields in quotes, but inserts a leading space, Snowflake reads the leading space rather than the opening quotation character as the beginning of the field (i.e. the quotation marks are interpreted as part of the string of field data). Use this option to remove undesirable spaces during the data load.

As another example, if leading or trailing space surrounds quotes that enclose strings, you can remove the surrounding space using the `TRIM_SPACE` option and the quote character using the `FIELD_OPTIONALLY_ENCLOSED_BY` option. Note that any space ***within*** the quotes is preserved.

For example, assuming the field delimiter is `|` and `FIELD_OPTIONALLY_ENCLOSED_BY = '"'`:

```
|"Hello world"|
|" Hello world "|
| "Hello world" |
```

becomes:

```
+---------------+
| C1            |
|----+----------|
| Hello world   |
|  Hello world  |
| Hello world   |
+---------------+
```

Default: `FALSE`

`FIELD_OPTIONALLY_ENCLOSED_BY = '<character>' | NONE`

Character used to enclose strings. Value can be `NONE`, single quote character (`'`), or double quote character (`"`). To use the single quote character, use the octal or hex representation (`0x27`) or the double single-quoted escape (`''`).

Default: `NONE`

`NULL_IF = ( '<string1>' [ , '<string2>' ... ] )`

String used to convert to and from SQL NULL. Snowflake replaces these strings in the data load source with SQL NULL. To specify more than one string, enclose the list of strings in parentheses and use commas to separate each value.

Note that Snowflake converts all instances of the value to NULL, regardless of the data type. For example, if `2` is specified as a value, all instances of `2` as either a string or number are converted.

For example:

```
NULL_IF = ('\N', 'NULL', 'NUL', '')
```

Note that this option can include empty strings.

Default: `\N` (i.e. NULL, which assumes the `ESCAPE_UNENCLOSED_FIELD` value is `\\` (default))

`ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE | FALSE`

Boolean that specifies whether to generate a parsing error if the number of delimited columns (i.e. fields) in an input data file does not match the number of columns in the corresponding table.

If set to `FALSE`, an error is not generated and the load continues. If the file is successfully loaded:

- If the input file contains records with more fields than columns in the table, the matching fields are loaded in order of occurrence in the file and the remaining fields are not loaded.

- If the input file contains records with fewer fields than columns in the table, the non-matching columns in the table are loaded with NULL values.

This option assumes all the records within the input file are the same length (i.e. a file containing records of varying length return an error regardless of the value specified for this option).

Default: `TRUE`

> **Note**
> When transforming data during loading (i.e. using a query as the source for the COPY INTO <table> command), this option is ignored. There is no requirement for your data files to have the same number and ordering of columns as your target table.

`REPLACE_INVALID_CHARACTERS = TRUE | FALSE`

Boolean that specifies whether to replace invalid UTF-8 characters with the Unicode replacement character (`�`). The copy option performs a one-to-one character replacement.

If set to `TRUE`, Snowflake replaces invalid UTF-8 characters with the Unicode replacement character.

If set to `FALSE`, the load operation produces an error when invalid UTF-8 character encoding is detected.

Default: `FALSE`

`EMPTY_FIELD_AS_NULL = TRUE | FALSE`

Boolean that specifies whether to insert SQL NULL for empty fields in an input file, which are represented by two successive delimiters (e.g. `,,`).

If set to `FALSE`, Snowflake attempts to cast an empty field to the corresponding column type. An empty string is inserted into columns of type STRING. For other column types, the COPY INTO *<table>* command produces an error.

Default: `TRUE`

`SKIP_BYTE_ORDER_MARK = TRUE | FALSE`

Boolean that specifies whether to skip the BOM (byte order mark), if present in a data file. A BOM is a character code at the beginning of a data file that defines the byte order and encoding form.

If set to `FALSE`, Snowflake recognizes any BOM in data files, which could result in the BOM either causing an error or being merged into the first column in the table.

Default: `TRUE`

`ENCODING = '<string>'`

String (constant) that specifies the character set of the source data.

Character Set	`ENCODING` Value	Supported Languages	Notes
Big5	`BIG5`	Traditional Chinese	
EUC-JP	`EUCJP`	Japanese	
EUC-KR	`EUCKR`	Korean	
GB18030	`GB18030`	Chinese	
IBM420	`IBM420`	Arabic	
IBM424	`IBM424`	Hebrew	
IBM949	`IBM949`	Korean	
ISO-2022-CN	`ISO2022CN`	Simplified Chinese	
ISO-2022-JP	`ISO2022JP`	Japanese	
ISO-2022-KR	`ISO2022KR`	Korean	
ISO-8859-1	`ISO88591`	Danish, Dutch, English, French, German, Italian, Norwegian, Portuguese, Swedish	
ISO-8859-2	`ISO88592`	Czech, Hungarian, Polish, Romanian	

Character Set	`ENCODING` Value	Supported Languages	Notes
ISO-8859-5	`ISO88595`	Russian	
ISO-8859-6	`ISO88596`	Arabic	
ISO-8859-7	`ISO88597`	Greek	
ISO-8859-8	`ISO88598`	Hebrew	
ISO-8859-9	`ISO88599`	Turkish	
ISO-8859-15	`ISO885915`	Danish, Dutch, English, French, German, Italian, Norwegian, Portuguese, Swedish	Identical to ISO-8859-1 except for 8 characters, including the Euro currency symbol.
KOI8-R	`KOI8R`	Russian	
Shift_JIS	`SHIFTJIS`	Japanese	
UTF-8	`UTF8`	All languages	For loading data from delimited files (CSV, TSV, etc.), UTF-8 is the default. For loading data from all other supported file formats (JSON, Avro, etc.), as well as unloading data, UTF-8 is the only supported character set.
UTF-16	`UTF16`	All languages	
UTF-16BE	`UTF16BE`	All languages	
UTF-16LE	`UTF16LE`	All languages	
UTF-32	`UTF32`	All languages	
UTF-32BE	`UTF32BE`	All languages	
UTF-32LE	`UTF32LE`	All languages	
windows-874	`WINDOWS874`	Thai	
windows-949	`WINDOWS949`	Korean	

Character Set	`ENCODING` Value	Supported Languages	Notes
windows-1250	`WINDOWS1250`	Czech, Hungarian, Polish, Romanian	
windows-1251	`WINDOWS1251`	Russian	
windows-1252	`WINDOWS1252`	Danish, Dutch, English, French, German, Italian, Norwegian, Portuguese, Swedish	
windows-1253	`WINDOWS1253`	Greek	
windows-1254	`WINDOWS1254`	Turkish	
windows-1255	`WINDOWS1255`	Hebrew	
windows-1256	`WINDOWS1256`	Arabic	

Default: `UTF8`

Note
Snowflake stores all data internally in the UTF-8 character set. The data is converted into UTF-8 before it is loaded into Snowflake.

TYPE = JSON

`COMPRESSION = AUTO | GZIP | BZ2 | BROTLI | ZSTD | DEFLATE | RAW_DEFLATE | NONE`

String (constant) that specifies the current compression algorithm for the data files to be loaded. Snowflake uses this option to detect how ***already-compressed*** data files were compressed so that the compressed data in the files can be extracted for loading.

Supported Values	Notes
`AUTO`	Compression algorithm detected automatically, except for Brotli-compressed files, which cannot currently be detected automatically. If loading Brotli-compressed files, explicitly use `BROTLI` instead of `AUTO`.
`GZIP`	
`BZ2`	
`BROTLI`	
`ZSTD`	
`DEFLATE`	Deflate-compressed files (with zlib header, RFC1950).
`RAW_DEFLATE`	Raw Deflate-compressed files (without header, RFC1951).
`NONE`	Indicates the files for loading data have not been compressed.

Default: `AUTO`

`DATE_FORMAT = '<string>' | AUTO`

Defines the format of date string values in the data files. If a value is not specified or is `AUTO`, the value for the DATE_INPUT_FORMAT parameter is used.

This file format option is applied to the following actions only:

- Loading JSON data into separate columns using the MATCH_BY_COLUMN_NAME copy option.
- Loading JSON data into separate columns by specifying a query in the COPY statement (i.e. COPY transformation).

Default: `AUTO`

`TIME_FORMAT = '<string>' | AUTO`

Defines the format of time string values in the data files. If a value is not specified or is `AUTO`, the value for the TIME_INPUT_FORMAT parameter is used.

This file format option is applied to the following actions only:

- Loading JSON data into separate columns using the MATCH_BY_COLUMN_NAME copy option.
- Loading JSON data into separate columns by specifying a query in the COPY statement (i.e. COPY transformation).

Default: `AUTO`

`TIMESTAMP_FORMAT = <string>' | AUTO`

Defines the format of timestamp string values in the data files. If a value is not specified or is `AUTO`, the value for the TIMESTAMP_INPUT_FORMAT parameter is used.

This file format option is applied to the following actions only:

- Loading JSON data into separate columns using the MATCH_BY_COLUMN_NAME copy option.
- Loading JSON data into separate columns by specifying a query in the COPY statement (i.e. COPY transformation).

Default: `AUTO`

`BINARY_FORMAT = HEX | BASE64 | UTF8`

Defines the encoding format for binary string values in the data files. The option can be used when loading data into binary columns in a table.

This file format option is applied to the following actions only:

- Loading JSON data into separate columns using the MATCH_BY_COLUMN_NAME copy option.
- Loading JSON data into separate columns by specifying a query in the COPY statement (i.e. COPY transformation).

Default: `HEX`

`TRIM_SPACE = TRUE | FALSE`

Boolean that specifies whether to remove leading and trailing white space from strings.

For example, if your external database software encloses fields in quotes, but inserts a leading space, Snowflake reads the leading space rather than the opening quotation character as the beginning of the field (i.e. the quotation marks are interpreted as part of the string of field data). Set this option to `TRUE` to remove undesirable spaces during the data load.

This file format option is applied to the following actions only when loading JSON data into separate columns using the MATCH_BY_COLUMN_NAME copy option.

Default: `FALSE`

`NULL_IF = ( '<string1>' [ , '<string2>' , ... ] )`

String used to convert to and from SQL NULL. Snowflake replaces these strings in the data load source with SQL NULL. To specify more than one string, enclose the list of strings in parentheses and use commas to separate each value.

This file format option is applied to the following actions only when loading JSON data into separate columns using the MATCH_BY_COLUMN_NAME copy option.

Note that Snowflake converts all instances of the value to NULL, regardless of the data type. For example, if `2` is specified as a value, all instances of `2` as either a string or number are converted.

For example:

```
NULL_IF = ('\N', 'NULL', 'NUL', '')
```

Note that this option can include empty strings.

Default: `\\N` (i.e. NULL, which assumes the `ESCAPE_UNENCLOSED_FIELD` value is `\\`)

`ENABLE_OCTAL = TRUE | FALSE`

Boolean that enables parsing of octal numbers.

Default: `FALSE`

`ALLOW_DUPLICATE = TRUE | FALSE`

Boolean that allows duplicate object field names (only the last one will be preserved).

Default: `FALSE`

`STRIP_OUTER_ARRAY = TRUE | FALSE`

Boolean that instructs the JSON parser to remove outer brackets `[ ]`.

Default: `FALSE`

`STRIP_NULL_VALUES = TRUE | FALSE`

Boolean that instructs the JSON parser to remove object fields or array elements containing `null` values. For example, when set to `TRUE`:

Before	After
`[null]`	`[]`
`[null,null,3]`	`[,,3]`
`{"a":null,"b":null,"c":123}`	`{"c":123}`
`{"a":[1,null,2],"b":{"x":null,"y":88}}`	`{"a":[1,,2],"b":{"y":88}}`

Default: `FALSE`

`REPLACE_INVALID_CHARACTERS = TRUE | FALSE`

Boolean that specifies whether to replace invalid UTF-8 characters with the Unicode replacement character (`�`). The copy option performs a one-to-one character replacement.

If set to `TRUE`, Snowflake replaces invalid UTF-8 characters with the Unicode replacement character.

If set to `FALSE`, the load operation produces an error when invalid UTF-8 character encoding is detected.

Default: `FALSE`

`IGNORE_UTF8_ERRORS = TRUE | FALSE`

Boolean that specifies whether UTF-8 encoding errors produce error conditions. It is an alternative syntax for `REPLACE_INVALID_CHARACTERS`.

If set to `TRUE`, any invalid UTF-8 sequences are silently replaced with the Unicode character `U+FFFD` (i.e. “replacement character”).

If set to `FALSE`, the load operation produces an error when invalid UTF-8 character encoding is detected.

Default: `FALSE`

`SKIP_BYTE_ORDER_MARK = TRUE | FALSE`

Boolean that specifies whether to skip any BOM (byte order mark) present in an input file. A BOM is a character code at the beginning of a data file that defines the byte order and encoding form.

If set to `FALSE`, Snowflake recognizes any BOM in data files, which could result in the BOM either causing an error or being merged into the first column in the table.

Default: `TRUE`

TYPE = AVRO

`COMPRESSION = AUTO | GZIP | BROTLI | ZSTD | DEFLATE | RAW_DEFLATE | NONE`

String (constant) that specifies the current compression algorithm for the data files to be loaded. Snowflake uses this option to detect how ***already-compressed*** data files were compressed so that the compressed data in the files can be extracted for loading.

Supported Values	Notes
`AUTO`	Compression algorithm detected automatically, except for Brotli-compressed files, which cannot currently be detected automatically. If loading Brotli-compressed files, explicitly use `BROTLI` instead of `AUTO`.
`GZIP`	
`BROTLI`	
`ZSTD`	
`DEFLATE`	Deflate-compressed files (with zlib header, RFC1950).
`RAW_DEFLATE`	Raw Deflate-compressed files (without header, RFC1951).
`NONE`	Data files to load have not been compressed.

Default: `AUTO`.

Note

We recommend that you use the default `AUTO` option because it will determine both the file and codec compression. Specifying a compression option refers to the compression of files, not the compression of blocks (codecs).

`TRIM_SPACE = TRUE | FALSE`

Boolean that specifies whether to remove leading and trailing white space from strings.

For example, if your external database software encloses fields in quotes, but inserts a leading space, Snowflake reads the leading space rather than the opening quotation character as the beginning of the field (i.e. the quotation marks are interpreted as part of the string of field data). Set this option to `TRUE` to remove undesirable spaces during the data load.

This file format option is applied to the following actions only when loading Avro data into separate columns using the MATCH_BY_COLUMN_NAME copy option.

Default: `FALSE`

`REPLACE_INVALID_CHARACTERS = TRUE | FALSE`

Boolean that specifies whether to replace invalid UTF-8 characters with the Unicode replacement character (`�`). The copy option performs a one-to-one character replacement.

If set to `TRUE`, Snowflake replaces invalid UTF-8 characters with the Unicode replacement character.

If set to `FALSE`, the load operation produces an error when invalid UTF-8 character encoding is detected.

Default: `FALSE`

`NULL_IF = ( '<string1>' [ , '<string2>' , ... ] )`

String used to convert to and from SQL NULL. Snowflake replaces these strings in the data load source with SQL NULL. To specify more than one string, enclose the list of strings in parentheses and use commas to separate each value.

This file format option is applied to the following actions only when loading Avro data into separate columns using the MATCH_BY_COLUMN_NAME copy option.

Note that Snowflake converts all instances of the value to NULL, regardless of the data type. For example, if `2` is specified as a value, all instances of `2` as either a string or number are converted.

For example:

```
NULL_IF = ('\N', 'NULL', 'NUL', '')
```

Note that this option can include empty strings.

Default: `\\N` (i.e. NULL, which assumes the `ESCAPE_UNENCLOSED_FIELD` value is `\\`)

TYPE = ORC

`TRIM_SPACE = TRUE | FALSE`

Boolean that specifies whether to remove leading and trailing white space from strings.

For example, if your external database software encloses fields in quotes, but inserts a leading space, Snowflake reads the leading space rather than the opening quotation character as the beginning of the field (i.e. the quotation marks are interpreted as part of the string of field data). Set this option to `TRUE` to remove undesirable spaces during the data load.

This file format option is applied to the following actions only when loading Orc data into separate columns using the MATCH_BY_COLUMN_NAME copy option.

Default: `FALSE`

`REPLACE_INVALID_CHARACTERS = TRUE | FALSE`

Boolean that specifies whether to replace invalid UTF-8 characters with the Unicode replacement character (`�`). The copy option performs a one-to-one character replacement.

If set to `TRUE`, Snowflake replaces invalid UTF-8 characters with the Unicode replacement character.

If set to `FALSE`, the load operation produces an error when invalid UTF-8 character encoding is detected.

Default: `FALSE`

`NULL_IF = ( '<string1>' [ , '<string2>' , ... ] )`

String used to convert to and from SQL NULL. Snowflake replaces these strings in the data load source with SQL NULL. To specify more than one string, enclose the list of strings in parentheses and use commas to separate each value.

This file format option is applied to the following actions only when loading Orc data into separate columns using the MATCH_BY_COLUMN_NAME copy option.

Note that Snowflake converts all instances of the value to NULL, regardless of the data type. For example, if `2` is specified as a value, all instances of `2` as either a string or number are converted.

For example:

`NULL_IF = ('\N', 'NULL', 'NUL', '')`

Note that this option can include empty strings.

Default: `\\N` (i.e. NULL, which assumes the `ESCAPE_UNENCLOSED_FIELD` value is `\\`)

TYPE = PARQUET

`COMPRESSION = AUTO | SNAPPY | NONE`

String (constant) that specifies the current compression algorithm for the data files to be loaded. Snowflake uses this option to detect how ***already-compressed*** data files were compressed so that the compressed data in the files can be extracted for loading.

Supported Values	Notes
`AUTO`	Compression algorithm detected automatically. Supports the following compression algorithms: Brotli, gzip, Lempel-Ziv-Oberhumer (LZO), LZ4, Snappy, or Zstandard v0.8 (and higher).
`SNAPPY`	
`NONE`	Data files to load have not been compressed.

`BINARY_AS_TEXT = TRUE | FALSE`

Boolean that specifies whether to interpret columns with no defined logical data type as UTF-8 text. When set to `FALSE`, Snowflake interprets these columns as binary data.

Default: `TRUE`

Note
Snowflake recommends that you set BINARY_AS_TEXT to FALSE to avoid any potential conversion issues.

`TRIM_SPACE = TRUE | FALSE`

Boolean that specifies whether to remove leading and trailing white space from strings.

For example, if your external database software encloses fields in quotes, but inserts a leading space, Snowflake reads the leading space rather than the opening quotation character as the beginning of the field (i.e. the quotation marks are interpreted as part of the string of field data). Set this option to `TRUE` to remove undesirable spaces during the data load.

This file format option is applied to the following actions only when loading Parquet data into separate columns using the MATCH_BY_COLUMN_NAME copy option.

Default: `FALSE`

`USE_LOGICAL_TYPE = TRUE | FALSE`

Boolean that specifies whether to use Parquet logical types. With this file format option, Snowflake can interpret Parquet logical types during data loading. For more information, see Parquet Logical Type Definitions. To enable Parquet logical types, set USE_LOGICAL_TYPE as TRUE when you create a new file format option.

Default: `FALSE`

`USE_VECTORIZED_SCANNER = TRUE | FALSE`

Boolean that specifies whether to use a vectorized scanner for loading Parquet files.

The default value is `FALSE`. In a future BCR, the default value will be `TRUE`. We recommend that you set `USE_VECTORIZED_SCANNER = TRUE` for new workloads, and set it for existing workloads after testing.

Using the vectorized scanner can significantly reduce the latency for loading Parquet files, because this scanner is well suited for the columnar format of a Parquet file. The scanner only downloads relevant sections of the Parquet file into memory, such as the subset of selected columns.

You can only enable the vectorized scanner if the following conditions are met:

- The `ON_ERROR` option must be set to `ABORT_STATEMENT` or `SKIP_FILE`.

The other values, `CONTINUE`, `SKIP_FILE_<num>`, `'SKIP_FILE_<num>%'` are not supported.

If `USE_VECTORIZED_SCANNER` is set to `TRUE`, the vectorized scanner has the following behaviors:

- The `BINARY_AS_TEXT` option is always treated as `FALSE` and the `USE_LOGICAL_TYPE` option is always treated as `TRUE`, no matter what the actual value is being set to.
- The vectorized scanner supports Parquet map types. The output of scanning a map type is as follows:

```
"my_map":
  {
   "k1": "v1",
   "k2": "v2"
  }
```

- The vectorized scanner shows `NULL` values in the output, as the following example demonstrates:

```
"person":
 {
  "name": "Adam",
  "nickname": null,
  "age": 34,
  "phone_numbers":
  [
    "1234567890",
    "0987654321",
    null,
    "6781234590"
  ]
  }
```

- The vectorized scanner handles Time and Timestamp as follows:

Parquet	Snowflake vectorized scanner
TimeType(isAdjustedToUtc=True/False, unit=MILLIS/MICROS/NANOS)	TIME
TimestampType(isAdjustedToUtc=True, unit=MILLIS/MICROS/NANOS)	TIMESTAMP_LTZ
TimestampType(isAdjustedToUtc=False, unit=MILLIS/MICROS/NANOS)	TIMESTAMP_NTZ
INT96	TIMESTAMP_LTZ

If `USE_VECTORIZED_SCANNER` is set to `FALSE`, the scanner has the following behaviors:

- This option does not support Parquet maps. The output of scanning a map type is as follows:

  ```
  "my_map":
   {
    "key_value":
    [
     {
            "key": "k1",
            "value": "v1"
        },
        {
            "key": "k2",
            "value": "v2"
        }
      ]
    }
  ```

- This option does not explicitly show `NULL` values in the scan output, as the following example demonstrates:

  ```
  "person":
   {
    "name": "Adam",
    "age": 34
    "phone_numbers":
    [
     "1234567890",
     "0987654321",
     "6781234590"
    ]
   }
  ```

- This option handles Time and Timestamp as follows:

Parquet	When USE_LOGICAL_TYPE = TRUE	When USE_LOGICAL_TYPE = FALSE
TimeType(isAdjustedToUtc=True/False, unit=MILLIS/MICROS)	TIME	• TIME (If ConvertedType present) • INTEGER (If ConvertedType not present)

Parquet	When USE_LOGICAL_TYPE = TRUE	When USE_LOGICAL_TYPE = FALSE
TimeType(isAdjustedToUtc=True/False, unit=NANOS)	TIME	INTEGER
TimestampType(isAdjustedToUtc=True, unit=MILLIS/MICROS)	TIMESTAMP_LTZ	TIMESTAMP_NTZ
TimestampType(isAdjustedToUtc=True, unit=NANOS)	TIMESTAMP_LTZ	INTEGER
TimestampType(isAdjustedToUtc=False, unit=MILLIS/MICROS)	TIMESTAMP_NTZ	• TIMESTAMP_LTZ (If ConvertedType present) • INTEGER (If ConvertedType not present)
TimestampType(isAdjustedToUtc=False, unit=NANOS)	TIMESTAMP_NTZ	INTEGER
INT96	TIMESTAMP_NTZ	TIMESTAMP_NTZ

`REPLACE_INVALID_CHARACTERS = TRUE | FALSE`

Boolean that specifies whether to replace invalid UTF-8 characters with the Unicode replacement character (`�`). The copy option performs a one-to-one character replacement.

If set to `TRUE`, Snowflake replaces invalid UTF-8 characters with the Unicode replacement character.

If set to `FALSE`, the load operation produces an error when invalid UTF-8 character encoding is detected.

Default: `FALSE`

`NULL_IF = ( '<string1>' [ , '<string2>' , ... ] )`

String used to convert to and from SQL NULL. Snowflake replaces these strings in the data load source with SQL NULL. To specify more than one string, enclose the list of strings in parentheses and use commas to separate each value.

This file format option is applied to the following actions only when loading Parquet data into separate columns using the MATCH_BY_COLUMN_NAME copy option.

Note that Snowflake converts all instances of the value to NULL, regardless of the data type. For example, if `2` is specified as a value, all instances of `2` as either a string or number are converted.

For example:

```
NULL_IF = ('\N', 'NULL', 'NUL', '')
```

Note that this option can include empty strings.

Default: `\\N` (i.e. NULL, which assumes the `ESCAPE_UNENCLOSED_FIELD` value is `\\`)

TYPE = XML

PREVIEW FEATURE — OPEN

Available to all accounts.

`COMPRESSION = AUTO | GZIP | BZ2 | BROTLI | ZSTD | DEFLATE | RAW_DEFLATE | NONE`

String (constant) that specifies the current compression algorithm for the data files to be loaded. Snowflake uses this option to detect how ***already-compressed*** data files were compressed so that the compressed data in the files can be extracted for loading.

Supported Values	Notes
`AUTO`	Compression algorithm detected automatically, except for Brotli-compressed files, which cannot currently be detected automatically. If loading Brotli-compressed files, explicitly use `BROTLI` instead of `AUTO`.
`GZIP`	
`BZ2`	
`BROTLI`	
`ZSTD`	
`DEFLATE`	Deflate-compressed files (with zlib header, RFC1950).

Supported Values	Notes
`RAW_DEFLATE`	Raw Deflate-compressed files (without header, RFC1951).
`NONE`	Data files to load have not been compressed.

Default: `AUTO`

`IGNORE_UTF8_ERRORS = TRUE | FALSE`

Boolean that specifies whether UTF-8 encoding errors produce error conditions. It is an alternative syntax for `REPLACE_INVALID_CHARACTERS`.

If set to `TRUE`, any invalid UTF-8 sequences are silently replaced with the Unicode character `U+FFFD` (i.e. “replacement character”).

If set to `FALSE`, the load operation produces an error when invalid UTF-8 character encoding is detected.

Default: `FALSE`

`PRESERVE_SPACE = TRUE | FALSE`

Boolean that specifies whether the XML parser preserves leading and trailing spaces in element content.

Default: `FALSE`

`STRIP_OUTER_ELEMENT = TRUE | FALSE`

Boolean that specifies whether the XML parser strips out the outer XML element, exposing 2nd level elements as separate documents.

Default: `FALSE`

`DISABLE_SNOWFLAKE_DATA = TRUE | FALSE`

Boolean that specifies whether the XML parser disables recognition of Snowflake semi-structured data tags.

Default: `FALSE`

`DISABLE_AUTO_CONVERT = TRUE | FALSE`

Boolean that specifies whether the XML parser disables automatic conversion of numeric and Boolean values from text to native representation.

Default: `FALSE`

`REPLACE_INVALID_CHARACTERS = TRUE | FALSE`

Boolean that specifies whether to replace invalid UTF-8 characters with the Unicode replacement character (`�`). The copy option performs a one-to-one character replacement.

If set to `TRUE`, Snowflake replaces invalid UTF-8 characters with the Unicode replacement character.

If set to `FALSE`, the load operation produces an error when invalid UTF-8 character encoding is detected.

Default: `FALSE`

`SKIP_BYTE_ORDER_MARK = TRUE | FALSE`

Boolean that specifies whether to skip any BOM (byte order mark) present in an input file. A BOM is a character code at the beginning of a data file that defines the byte order and encoding form.

If set to `FALSE`, Snowflake recognizes any BOM in data files, which could result in the BOM either causing an error or being merged into the first column in the table.

Default: `TRUE`

Copy options (`copyOptions`)

You can specify one or more of the following copy options (separated by blank spaces, commas, or new lines):

`ON_ERROR = CONTINUE | SKIP_FILE | SKIP_FILE_<num> | 'SKIP_FILE_<num>%' | ABORT_STATEMENT`

Use Data loading only

Definition String (constant) that specifies the error handling for the load operation.

> **Important**
> Carefully consider the ON_ERROR copy option value. The default value is appropriate in common scenarios, but is not always the best option.

Values

- `CONTINUE`

  Continue to load the file if errors are found. The COPY statement returns an error message for a maximum of one error found per data file.

  Note that the difference between the ROWS_PARSED and ROWS_LOADED column values represents the number of rows that include detected errors. However, each of these rows could include multiple errors. To view all errors in the data files, use the VALIDATION_MODE parameter or query the VALIDATE function.

- `SKIP_FILE`

  Skip a file when an error is found.

  Note that the `SKIP_FILE` action buffers an entire file whether errors are found or not. For this reason, `SKIP_FILE` is slower than either `CONTINUE` or `ABORT_STATEMENT`. Skipping large files due to a small number of errors could result in delays and wasted credits. When loading large numbers of records from files that have no logical delineation (e.g. the files were generated automatically at rough intervals), consider specifying `CONTINUE` instead.

  Additional patterns:

  `SKIP_FILE_<num>` (e.g. `SKIP_FILE_10`)
  Skip a file when the number of error rows found in the file is equal to or exceeds the specified number.

  `'SKIP_FILE_<num>%'` (e.g. `'SKIP_FILE_10%'`)
  Skip a file when the percentage of error rows found in the file exceeds the specified percentage.

- `ABORT_STATEMENT`

  Abort the load operation if any error is found in a data file.

  The load operation is aborted only when the data files that were explicitly specified in the `FILES` parameter cannot be found. Otherwise, the load operation is ***not*** aborted if the data file cannot be found (for example, because it does not exist or cannot be accessed).

Note that the aborted operations do not show up in COPY_HISTORY as the data files were not ingested. We recommend that you search for the failures in QUERY_HISTORY.

Default	**Bulk loading using COPY**	`ABORT_STATEMENT`
	Snowpipe	`SKIP_FILE`

`SIZE_LIMIT = <num>`

Definition Number (> 0) that specifies the maximum size (in bytes) of data to be loaded for a given COPY statement. When the threshold is exceeded, the COPY operation discontinues loading files. This option is commonly used to load a common group of files using multiple COPY statements. For each statement, the data load continues until the specified `SIZE_LIMIT` is exceeded, before moving on to the next statement.

For example, suppose a set of files in a stage path were each 10 MB in size. If multiple COPY statements set SIZE_LIMIT to `25000000` (25 MB), each would load 3 files. That is, each COPY operation would discontinue after the `SIZE_LIMIT` threshold was exceeded.

Note that at least one file is loaded regardless of the value specified for `SIZE_LIMIT` unless there is no file to be loaded.

Default null (no size limit)

`PURGE = TRUE | FALSE`

Definition Boolean that specifies whether to remove the data files from the stage automatically after the data is loaded successfully.

If this option is set to `TRUE`, note that a best effort is made to remove successfully loaded data files. If the purge operation fails for any reason, no error is returned currently. We recommend that you list staged files periodically (using LIST) and manually remove successfully loaded files, if any exist.

Default `FALSE`

`RETURN_FAILED_ONLY = TRUE | FALSE`

Definition Boolean that specifies whether to return only files that have failed to load in the statement result.

Default `FALSE`

`MATCH_BY_COLUMN_NAME = CASE_SENSITIVE | CASE_INSENSITIVE | NONE`

Definition String that specifies whether to load semi-structured data into columns in the target table that match corresponding columns represented in the data.

Important
Do not use the MATCH_BY_COLUMN_NAME copy option with a SELECT statement for transforming data during a load in all cases. These two options can still be used separately, but cannot be used together. Any attempt to do so will result in the following error: `SQL compilation error: match_by_column_name is not supported with copy transform.`.

For example, the following syntax is not allowed:

```
COPY INTO [<namespace>.]<table_name> [ ( <col_name> [ ,
<col_name> ... ] ) ]
FROM ( SELECT [<alias>.]$<file_col_num>[.<element>] [ ,
[<alias>.]$<file_col_num>[.<element>] ... ]
    FROM { internalStage | externalStage } )
[ FILES = ( '<file_name>' [ , '<file_name>' ] [ , ... ]
) ]
[ PATTERN = '<regex_pattern>' ]
[ FILE_FORMAT = ( { FORMAT_NAME = '[<namespace>.]
<file_format_name>' |
            TYPE = { CSV | JSON | AVRO | ORC | PARQUET |
XML } [ formatTypeOptions ] } ) ]
MATCH_BY_COLUMN_NAME = CASE_SENSITIVE | CASE_INSENSITIVE
| NONE
[ other copyOptions ]
```

For more information, see Transforming Data During a Load.

This copy option is supported for the following data formats:

- JSON
- Avro
- ORC
- Parquet
- CSV

For a column to match, the following criteria must be true:

- The column represented in the data must have the ***exact same name*** as the column in the table. The copy option supports case sensitivity for column names. Column order does not matter.
- The column in the table must have a data type that is compatible with the values in the column represented in the data. For example, string, number, and Boolean values can all be loaded into a variant column.

Values **`CASE_SENSITIVE` | `CASE_INSENSITIVE`**

Load semi-structured data into columns in the target table that match corresponding columns represented in the data. Column names are either case-sensitive (`CASE_SENSITIVE`) or case-insensitive (`CASE_INSENSITIVE`).

The COPY operation verifies that at least one column in the target table matches a column represented in the data files. If a match is found, the values in the data files are loaded into the column or columns. If no match is found, a set of NULL values for each record in the files is loaded into the table.

Note

- If additional non-matching columns are present in the data files, the values in these columns are not loaded.
- If additional non-matching columns are present in the target table, the COPY operation inserts NULL values into these columns. These columns must support NULL values.

`NONE`

The COPY operation loads the semi-structured data into a variant column or, if a query is included in the COPY statement, transforms the data.

Default `NONE`

Note

The following limitations currently apply:

- MATCH_BY_COLUMN_NAME cannot be used with the `VALIDATION_MODE` parameter in a COPY statement to validate the staged data rather than load it into the target table.
- ***Parquet data only.*** When MATCH_BY_COLUMN_NAME is set to `CASE_SENSITIVE` or `CASE_INSENSITIVE`, an empty column value (e.g. `"col1": ""`) produces an error.

`INCLUDE_METADATA = ( <column_name> = METADATA$<field> [ , <column_name> = METADATA$<field> ... ] )`

Definition A user-defined mapping between a target table’s existing columns to its METADATA$ columns. This copy option can only be used with the MATCH_BY_COLUMN_NAME copy option. The valid input for `METADATA$<field>` includes the following:

- METADATA$FILENAME
- METADATA$FILE_ROW_NUMBER
- METADATA$FILE_CONTENT_KEY
- METADATA$FILE_LAST_MODIFIED
- METADATA$START_SCAN_TIME

For more information about metadata columns, see Querying Metadata for Staged Files.

When a mapping is defined with this copy option, the column `<column_name>` is populated with the specified metadata value, as the following example demonstrates:

```
COPY INTO table1 FROM @stage1
MATCH_BY_COLUMN_NAME = CASE_INSENSITIVE
INCLUDE_METADATA = (
    ingestdate = METADATA$START_SCAN_TIME, filename =
METADATA$FILENAME);
```

```
+-----+----------------------+-----------------------------
----+-----+
| ... | FILENAME             | INGESTDATE
| ... |
|-------------------------------------------------------------
----+-----|
| ... | example_file.json.gz  | Thu, 22 Feb 2024 19:14:55
+0000 | ... |
+-----+----------------------+-----------------------------
----+-----+
```

Default NULL

Note

- The `INCLUDE_METADATA` target column name must first exist in the table. The target column name is not automatically added if it does not exist.

- Use a unique column name for the `INCLUDE_METADATA` columns. If the `INCLUDE_METADATA` target column has a name conflict with a column in the data file, the `METADATA$` value defined by `INCLUDE_METADATA` takes precedence.
- When loading CSV with `INCLUDE_METADATA`, set the file format option `ERROR_ON_COLUMN_COUNT_MISMATCH` to `FALSE`.

`ENFORCE_LENGTH = TRUE | FALSE`

Definition ***Alternative syntax for*** `TRUNCATECOLUMNS` ***with reverse logic (for compatibility with other systems)***

Boolean that specifies whether to truncate text strings that exceed the target column length:

- If `TRUE`, the COPY statement produces an error if a loaded string exceeds the target column length.
- If `FALSE`, strings are automatically truncated to the target column length.

This copy option supports CSV data, as well as string values in semi-structured data when loaded into separate columns in relational tables.

Default `TRUE`

Note

- If the length of the target string column is set to the maximum (e.g. `VARCHAR (16777216)`), an incoming string cannot exceed this length; otherwise, the COPY command produces an error.
- This parameter is functionally equivalent to `TRUNCATECOLUMNS`, but has the opposite behavior. It is provided for compatibility with other databases. It is only necessary to include one of these two parameters in a COPY statement to produce the desired output.

`TRUNCATECOLUMNS = TRUE | FALSE`

Definition ***Alternative syntax for*** `ENFORCE_LENGTH` ***with reverse logic (for compatibility with other systems)***

Boolean that specifies whether to truncate text strings that exceed the target column length:

- If `TRUE`, strings are automatically truncated to the target column length.

- If `FALSE`, the COPY statement produces an error if a loaded string exceeds the target column length.

This copy option supports CSV data, as well as string values in semi-structured data when loaded into separate columns in relational tables.

Default `FALSE`

Note

- If the length of the target string column is set to the maximum (e.g. `VARCHAR(16777216)`), an incoming string cannot exceed this length; otherwise, the COPY command produces an error.
- This parameter is functionally equivalent to `ENFORCE_LENGTH`, but has the opposite behavior. It is provided for compatibility with other databases. It is only necessary to include one of these two parameters in a COPY statement to produce the desired output.

`FORCE = TRUE | FALSE`

Definition Boolean that specifies to load all files, regardless of whether they've been loaded previously and have not changed since they were loaded. Note that this option reloads files, potentially duplicating data in a table.

Default `FALSE`

`LOAD_UNCERTAIN_FILES = TRUE | FALSE`

Definition Boolean that specifies to load files for which the load status is unknown. The COPY command skips these files by default.

The load status is unknown if ***all*** of the following conditions are true:

- The file's LAST_MODIFIED date (i.e. date when the file was staged) is older than 64 days.
- The initial set of data was loaded into the table more than 64 days earlier.
- If the file was already loaded successfully into the table, this event occurred more than 64 days earlier.

To force the COPY command to load all files regardless of whether the load status is known, use the `FORCE` option instead.

For more information about load status uncertainty, see Loading older files.

Default `FALSE`

`FILE_PROCESSOR = (SCANNER = *<custom_scanner_type>* SCANNER_OPTIONS = (*<scanner_options>*))`

Definition Specifies the scanner and the scanner options used for processing unstructured data.

PREVIEW FEATURE — OPEN

Available to all accounts.

- `SCANNER` (Required): specifies the type of custom scanner used to process unstructured data. Currently, only the `document_ai` custom scanner type is supported.
- `SCANNER_OPTIONS`: specifies the properties to the custom scanner type. For example, if you specify `document_ai` as the type of `SCANNER`, you must specify the properties of `document_ai`. The predefined set of properties for `document_ai` are:
  - `project_name`: the name of the project where you create the Document AI model.
  - `model_name` (Required for `document_ai`): the name of the Document AI model.
  - `model_version` (Required for `document_ai`): the version of the Document AI model.

For more information, see Loading unstructured data with Document AI.

Note
This copy option does not work with `MATCH_BY_COLUMN_NAME`.

Usage notes

- Some use cases are not fully supported and can lead to inconsistent or unexpected ON_ERROR behavior, including the following use cases:
  - Specifying the DISTINCT keyword in SELECT statements.
  - Using COPY with clustered tables.

- When you load CSV data, if a stream is on the target table, the ON_ERROR copy option might not work as expected.
- ***Loading from Google Cloud Storage only:*** The list of objects returned for an external stage might include one or more "directory blobs"; essentially, paths that end in a forward slash character (`/`), e.g.:

  ```
  LIST @my_gcs_stage;

  +---------------------------------------+------+-----------------------------------+-------------------------------+
  | name                                  | size | md5                               | last_modified                 |
  |---------------------------------------+------+-----------------------------------+-------------------------------|
  | my_gcs_stage/load/                    |  12  | 12348f18bcb35e7b6b628ca12345678c | Mon, 11 Sep 2019 16:57:43 GMT |
  | my_gcs_stage/load/data_0_0_0.csv.gz   |  147 | 9765daba007a643bdff4eae10d43218y | Mon, 11 Sep 2019 18:13:07 GMT |
  +---------------------------------------+------+-----------------------------------+-------------------------------+
  ```

  These blobs are listed when directories are created in the Google Cloud Platform Console rather than using any other tool provided by Google.

  COPY statements that reference a stage can fail when the object list includes directory blobs. To avoid errors, we recommend using file pattern matching to identify the files for inclusion (i.e. the PATTERN clause) when the file list for a stage includes directory blobs. For an example, see Loading Using Pattern Matching (in this topic). Alternatively, set ON_ERROR = SKIP_FILE in the COPY statement.
- `STORAGE_INTEGRATION`, `CREDENTIALS`, and `ENCRYPTION` only apply if you are loading directly from a private/protected storage location:
  - If you are loading from a public bucket, secure access is not required.
  - If you are loading from a named external stage, the stage provides all the credential information required for accessing the bucket.
- If you encounter errors while running the COPY command, after the command completes, you can validate the files that produced the errors using the VALIDATE table function.

  Note
  The VALIDATE function only returns output for COPY commands used to perform standard data loading; it does not support COPY commands that perform transformations during data loading (e.g. loading a subset of data columns or reordering data columns).

- Unless you explicitly specify `FORCE = TRUE` as one of the copy options, the command ignores staged data files that were already loaded into the table. To reload the data, you must either specify `FORCE = TRUE` or modify the file and stage it again, which generates a new checksum.

- The COPY command does not validate data type conversions for Parquet files.

Output

The command returns the following columns:

Column Name	Data Type	Description
FILE	TEXT	Name of source file and relative path to the file
STATUS	TEXT	Status: loaded, load failed or partially loaded
ROWS_PARSED	NUMBER	Number of rows parsed from the source file
ROWS_LOADED	NUMBER	Number of rows loaded from the source file
ERROR_LIMIT	NUMBER	If the number of errors reaches this limit, then abort
ERRORS_SEEN	NUMBER	Number of error rows in the source file
FIRST_ERROR	TEXT	First error of the source file
FIRST_ERROR_LINE	NUMBER	Line number of the first error
FIRST_ERROR_CHARACTER	NUMBER	Position of the first error character
FIRST_ERROR_COLUMN_NAME	TEXT	Column name of the first error

Examples

For examples of data loading transformations, see Transforming data during a load.

Loading files from an internal stage

Note
These examples assume the files were copied to the stage earlier using the PUT command.

Load files from a named internal stage into a table:

```
COPY INTO mytable
FROM @my_int_stage;
```

Load files from a table's stage into the table:

```
COPY INTO mytable
FILE_FORMAT = (TYPE = CSV);
```

Note
When copying data from files in a table location, the FROM clause can be omitted because Snowflake automatically checks for files in the table's location.

Load files from the user's personal stage into a table:

```
COPY INTO mytable from @~/staged
FILE_FORMAT = (FORMAT_NAME = 'mycsv');
```

Loading files from a named external stage

Load files from a named external stage that you created previously using the CREATE STAGE command. The named external stage references an external location (Amazon S3, Google Cloud Storage, or Microsoft Azure) and includes all the credentials and other details required for accessing the location:

```
COPY INTO mycsvtable
  FROM @my_ext_stage/tutorials/dataloading/contacts1.csv;
```

Loading files using column matching

Load files from a named external stage into the table with the `MATCH_BY_COLUMN_NAME` copy option, by case-insensitive matching the column names in the files to the column names defined in the table. With this option, the column ordering of the file does not need to match the column ordering of the table.

```
COPY INTO mytable
  FROM @my_ext_stage/tutorials/dataloading/sales.json.gz
  FILE_FORMAT = (TYPE = 'JSON')
  MATCH_BY_COLUMN_NAME='CASE_INSENSITIVE';
```

Loading files directly from an external location

The following example loads all files prefixed with `data/files` from a storage location (Amazon S3, Google Cloud Storage, or Microsoft Azure) using a named `my_csv_format` file format:

Amazon S3

Access the referenced S3 bucket using a referenced storage integration named `myint`. Note that both examples truncate the `MASTER_KEY` value:

```
COPY INTO mytable
  FROM s3://mybucket/data/files
  STORAGE_INTEGRATION = myint
  ENCRYPTION=(MASTER_KEY = 'eSx...')
  FILE_FORMAT = (FORMAT_NAME = my_csv_format);
```

Access the referenced S3 bucket using supplied credentials:

```
COPY INTO mytable
  FROM s3://mybucket/data/files
  CREDENTIALS=(AWS_KEY_ID='$AWS_ACCESS_KEY_ID'
AWS_SECRET_KEY='$AWS_SECRET_ACCESS_KEY')
  ENCRYPTION=(MASTER_KEY = 'eSx...')
  FILE_FORMAT = (FORMAT_NAME = my_csv_format);
```

Google Cloud Storage

Access the referenced GCS bucket using a referenced storage integration named `myint`:

```
COPY INTO mytable
  FROM 'gcs://mybucket/data/files'
  STORAGE_INTEGRATION = myint
  FILE_FORMAT = (FORMAT_NAME = my_csv_format);
```

Microsoft Azure

Access the referenced container using a referenced storage integration named `myint`. Note that both examples truncate the `MASTER_KEY` value:

```
COPY INTO mytable
  FROM 'azure://myaccount.blob.core.windows.net/data/files'
  STORAGE_INTEGRATION = myint
  ENCRYPTION=(TYPE='AZURE_CSE' MASTER_KEY = 'kPx...')
  FILE_FORMAT = (FORMAT_NAME = my_csv_format);
```

Access the referenced container using supplied credentials:

```
COPY INTO mytable
  FROM 'azure://myaccount.blob.core.windows.net/mycontainer/data/files'
  CREDENTIALS=(AZURE_SAS_TOKEN='?sv=2016-05-
31&ss=b&srt=sco&sp=rwdl&se=2018-06-27T10:05:50Z&st=2017-06-
27T02:05:50Z&spr=https,http&sig=bgqQwoXwxzuD2GJfagRg7VOS8hzNr3QLT7rhS8OFRL
Q%3D')
  ENCRYPTION=(TYPE='AZURE_CSE' MASTER_KEY = 'kPx...')
  FILE_FORMAT = (FORMAT_NAME = my_csv_format);
```

Loading using pattern matching

Load files from a table's stage into the table, using pattern matching to only load data from compressed CSV files in any path:

```
COPY INTO mytable
  FILE_FORMAT = (TYPE = 'CSV')
  PATTERN='.*/.*/.*[.]csv[.]gz';
```

Where `.*` is interpreted as "zero or more occurrences of any character." The square brackets escape the period character (`.`) that precedes a file extension.

Load files from a table stage into the table using pattern matching to only load uncompressed CSV files whose names include the string `sales`:

```
COPY INTO mytable
  FILE_FORMAT = (FORMAT_NAME = myformat)
  PATTERN='.*sales.*[.]csv';
```

Loading JSON data into a VARIANT column

The following example loads JSON data into a table with a single column of type VARIANT.

The staged JSON array comprises three objects separated by new lines:

```
[{
    "location": {
      "city": "Lexington",
      "zip": "40503",
      },
      "sq__ft": "1000",
      "sale_date": "4-25-16",
      "price": "75836"
},
{
    "location": {
      "city": "Belmont",
      "zip": "02478",
      },
      "sq__ft": "1103",
      "sale_date": "6-18-16",
      "price": "92567"
}
{
    "location": {
      "city": "Winchester",
      "zip": "01890",
      },
      "sq__ft": "1122",
      "sale_date": "1-31-16",
      "price": "89921"
}]
```

```
/* Create a JSON file format that strips the outer array. */

CREATE OR REPLACE FILE FORMAT json_format
  TYPE = 'JSON'
  STRIP_OUTER_ARRAY = TRUE;

/* Create an internal stage that references the JSON file format. */
```

```
CREATE OR REPLACE STAGE mystage
  FILE_FORMAT = json_format;

/* Stage the JSON file. */

PUT file:///tmp/sales.json @mystage AUTO_COMPRESS=TRUE;

/* Create a target table for the JSON data. */

CREATE OR REPLACE TABLE house_sales (src VARIANT);

/* Copy the JSON data into the target table. */

COPY INTO house_sales
   FROM @mystage/sales.json.gz;

SELECT * FROM house_sales;

+----------------------------+
| SRC                        |
|----------------------------|
| {                          |
|   "location": {            |
|     "city": "Lexington",   |
|     "zip": "40503"         |
|   },                       |
|   "price": "75836",        |
|   "sale_date": "4-25-16",  |
|   "sq__ft": "1000",        |
|   "type": "Residential"    |
| }                          |
| {                          |
|   "location": {            |
|     "city": "Belmont",     |
|     "zip": "02478"         |
|   },                       |
|   "price": "92567",        |
|   "sale_date": "6-18-16",  |
|   "sq__ft": "1103",        |
|   "type": "Residential"    |
| }                          |
| {                          |
|   "location": {            |
|     "city": "Winchester",  |
|     "zip": "01890"         |
|   },                       |
|   "price": "89921",        |
|   "sale_date": "1-31-16",  |
|   "sq__ft": "1122",        |
|   "type": "Condo"          |
| }                          |
+----------------------------+
```

Reloading files

Add `FORCE = TRUE` to a COPY command to reload (duplicate) data from a set of staged data files that have not changed (i.e. have the same checksum as when they were first loaded).

In the following example, the first command loads the specified files and the second command forces the same files to be loaded again (producing duplicate rows), even though the contents of the files have not changed:

```
COPY INTO load1 FROM @%load1/data1/
    FILES=('test1.csv', 'test2.csv');

COPY INTO load1 FROM @%load1/data1/
    FILES=('test1.csv', 'test2.csv')
    FORCE=TRUE;
```

Purging files after loading

Load files from a table's stage into the table and purge files after loading. By default, COPY does not purge loaded files from the location. To purge the files after loading:

- Set `PURGE=TRUE` for the table to specify that all files successfully loaded into the table are purged after loading:

  ```
  ALTER TABLE mytable SET STAGE_COPY_OPTIONS = (PURGE = TRUE);

  COPY INTO mytable;
  ```

- You can also override any of the copy options directly in the COPY command:

  ```
  COPY INTO mytable PURGE = TRUE;
  ```

Validating staged files

Validate files in a stage without loading:

- Run the COPY command in validation mode and see all errors:

```
COPY INTO mytable VALIDATION_MODE = 'RETURN_ERRORS';

+---------------------------------------------------------------------
----------------------------------------------------------+-----------
-------------+------+-----------+-------------+----------+--------+---
--------+----------------------+------------+---------------+
|                                                         ERROR
|            FILE        | LINE | CHARACTER | BYTE_OFFSET | CATEGORY |
CODE  | SQL_STATE |   COLUMN_NAME        | ROW_NUMBER | ROW_START_LINE
|
+---------------------------------------------------------------------
----------------------------------------------------------+-----------
-------------+------+-----------+-------------+----------+--------+---
--------+----------------------+------------+---------------+
| Field delimiter ',' found while expecting record delimiter '\n'
| @MYTABLE/data1.csv.gz  | 3    | 21         | 76          | parsing  |
100016 | 22000      | "MYTABLE"["QUOTA":3] | 3          | 3
|
| NULL result in a non-nullable column. Use quotes if an empty field
should be interpreted as an empty string instead of a null |
@MYTABLE/data3.csv.gz  | 3    | 2          | 62          | parsing  |
100088 | 22000      | "MYTABLE"["NAME":1]  | 3          | 3
|
| End of record reached while expected to parse column '"MYTABLE"
["QUOTA":3]'                                                   |
@MYTABLE/data3.csv.gz  | 4    | 20         | 96          | parsing  |
100068 | 22000      | "MYTABLE"["QUOTA":3] | 4          | 4
|
+---------------------------------------------------------------------
----------------------------------------------------------+-----------
-------------+------+-----------+-------------+----------+--------+---
--------+----------------------+------------+---------------+
```

- Run the COPY command in validation mode for a specified number of rows. In this example, the first run encounters no errors in the specified number of rows and completes successfully, displaying the information as it will appear when loaded into the table. The second run encounters an error in the specified number of rows and fails with the error encountered:

```
COPY INTO mytable VALIDATION_MODE = 'RETURN_2_ROWS';

+--------------------+----------+-------+
|        NAME        |    ID    | QUOTA |
+--------------------+----------+-------+
| Joe Smith          |  456111  | 0     |
| Tom Jones          |  111111  | 3400  |
+--------------------+----------+-------+

COPY INTO mytable VALIDATION_MODE = 'RETURN_3_ROWS';

FAILURE: NULL result in a non-nullable column. Use quotes if an empty
```

```
field should be interpreted as an empty string instead of a null
  File '@MYTABLE/data3.csv.gz', line 3, character 2
  Row 3, column "MYTABLE"["NAME":1]
```