**Guided Lab: Bulk Loading from a Local File System**

This tutorial describes how to load data from files in your local file system into a table.

**Introduction**

In this tutorial, you will learn how to:

Create named file format objects that describe your data files.

Create named stage objects.

Upload your data to the internal stages.

Load your data into tables.

Resolve errors in your data files.

### **Downloading the sample data files**

For this tutorial you need to download the sample data files provided by Snowflake.

To download and unzip the sample data files:

1. Right-click the name of the archive file, [data-load-internal.zip](https://docs.snowflake.com/en/_downloads/22c3a6290f5d1f4d97075282729f3859/data-load-internal.zip) and save the link/file to your local file system.
2. Unzip the sample files. The tutorial assumes you unpacked files in to the following directories:

* Linux/macOS: /tmp/load
* Windows: C:\tempload

C:\Users\Administrator\Downloads\data-load-internal

These data files include sample contact data in the following formats:

CSV files that contain a header row and five records. The field delimiter is the pipe (|) character. The following example shows a header row and one record:  
ID**|**lastname**|**firstname**|**company**|email|**workphone**|**cellphone**|**streetaddress**|**city**|**postalcode

6**|**Reed**|**Moses**|**Neque Corporation**|**eget**.**lacus**@**facilisis**.**com**|**1**-**449**-**871**-**0780**|**1**-**454**-**964**-**5318**|**Ap **#**225**-**4351 Dolor Ave**|**Titagarh**|**62631

A single file in JSON format that contains one array and three objects. The following is an example of an array that contains one of the objects:  
**[**

**{**

"customer"**:** **{**

"address"**:** "509 Kings Hwy, Comptche, Missouri, 4848"**,**

"phone"**:** "+1 (999) 407-2274"**,**

"email"**:** "blankenship.patrick@orbin.ca"**,**

"company"**:** "ORBIN"**,**

"name"**:** **{**

"last"**:** "Patrick"**,**

"first"**:** "Blankenship"

**},**

"\_id"**:** "5730864df388f1d653e37e6f"

**}**

**},**

**]**

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### **Step 1: Creating the database, tables, and warehouse**

Execute the following statements to create a database, two tables (for csv and json data), and a virtual warehouse needed for this tutorial. After you complete the tutorial, you can drop these objects.

-- Create a database. A database automatically includes a schema named 'public'.

**CREATE** **OR** **REPLACE** **DATABASE** mydatabase**;**

/\* Create target tables for CSV and JSON data. The tables are temporary, meaning they persist only for the duration of the user session and are not visible to other users. \*/

**CREATE** **OR** **REPLACE** **TEMPORARY** **TABLE** mycsvtable **(**

id **INTEGER,**

**last\_name** **STRING,**

**first\_name** **STRING,**

company **STRING,**

**email** **STRING,**

workphone **STRING,**

cellphone **STRING,**

streetaddress **STRING,**

city **STRING,**

postalcode **STRING);**

**CREATE** **OR** **REPLACE** **TEMPORARY** **TABLE** myjsontable **(**

json\_data **VARIANT);**

-- Create a warehouse

**CREATE** **OR** **REPLACE** **WAREHOUSE** mywarehouse **WITH**

**WAREHOUSE\_SIZE=**'X-SMALL'

**AUTO\_SUSPEND** **=** 120

**AUTO\_RESUME** **=** **TRUE**

**INITIALLY\_SUSPENDED=TRUE;**

The CREATE WAREHOUSE statement sets up the warehouse to be suspended initially. The statement also sets AUTO\_RESUME = true, which starts the warehouse automatically when you execute SQL statements that require compute resources.

## **Step 2: Create file format objects**

When you load data from a file into a table, you must describe the format of the file and specify how the data in the file should be interpreted and processed. For example, if you are loading pipe-delimited data from a CSV file, you must specify that the file uses the CSV format with pipe symbols as delimiters.

When you execute the [COPY INTO <table>](https://docs.snowflake.com/en/sql-reference/sql/copy-into-table) command, you specify this format information. You can either specify this information as options in the command (e.g. TYPE = CSV, FIELD\_DELIMITER = '|', etc.) or you can specify a file format object that contains this format information. You can create a named file format object using the [CREATE FILE FORMAT](https://docs.snowflake.com/en/sql-reference/sql/create-file-format) command.

In this step, you create file format objects describing the data format of the sample CSV and JSON data provided for this tutorial.

### **Creating a file format object for CSV data**

Execute the [CREATE FILE FORMAT](https://docs.snowflake.com/en/sql-reference/sql/create-file-format) command to create the mycsvformat file format.

**CREATE** **OR** **REPLACE** **FILE** **FORMAT** mycsvformat

**TYPE** **=** 'CSV'

**FIELD\_DELIMITER** **=** '|'

**SKIP\_HEADER** **=** 1**;**

Where:

* TYPE = 'CSV' indicates the source file format type. CSV is the default file format type.
* FIELD\_DELIMITER = '|' indicates the ‘|’ character is a field separator. The default value is ‘,’.
* SKIP\_HEADER = 1 indicates the source file includes one header line. The COPY command skips these header lines when loading data. The default value is 0.

### **Creating a file format object for JSON data**

Execute the [CREATE FILE FORMAT](https://docs.snowflake.com/en/sql-reference/sql/create-file-format) command to create the myjsoWhere:

CREATE OR REPLACE FILE FORMAT myjsonformat  
 TYPE = 'JSON'  
 STRIP\_OUTER\_ARRAY = TRUE;  
 Where:  
 ⦁  TYPE = 'JSON' indicates the source file format type.  
 ⦁  STRIP\_OUTER\_ARRAY = TRUE directs the COPY command to exclude the root brackets ([]) when loading data to the table.

* TYPE = 'JSON' indicates the source file format type.
* STRIP\_OUTER\_ARRAY = TRUE directs the COPY command to exclude the root brackets ([]) when loading data to the table.

## **Step 3: Create stage objects**

A stage specifies where data files are stored (i.e. “staged”) so that the data in the files can be loaded into a table. A named [internal stage](https://docs.snowflake.com/en/user-guide/data-load-overview.html#label-data-load-overview-internal-stages) is a cloud storage location managed by Snowflake.

Creating a named stage is useful if you want multiple users or processes to upload files. If you plan to stage data files to load only by you, or to load only into a single table, then you may prefer to use your user stage or the table stage. For information, see [Bulk loading from a local file system](https://docs.snowflake.com/en/user-guide/data-load-local-file-system).

In this step, you create named stages for the different types of sample data files.

### **Creating a stage for CSV data files**

Execute CREATE STAGE to create the my\_csv\_stage stage:

**CREATE** **OR** **REPLACE** **STAGE** my\_csv\_stage

**FILE\_FORMAT** **=** mycsvformat**;**

Note that if you specify the FILE\_FORMAT option when creating the stage, it is not necessary to specify the same FILE\_FORMAT option in the COPY command used to load data from the stage.

### **Creating a stage for JSON data files**

Execute CREATE STAGE to create the my\_json\_stage stage:

**CREATE** **OR** **REPLACE** **STAGE** my\_json\_stage

**FILE\_FORMAT** **=** myjsonformat**;**

## **Step 4: Stage the data files**

Execute [PUT](https://docs.snowflake.com/en/sql-reference/sql/put) to upload (stage) sample data files from your local file system to the stages.

### **Staging the CSV sample data files**

Execute the PUT command to upload the CSV files from your local file system.

Linux or macOS  
**PUT** **file:///**tmp**/load/contacts\*.csv** **@**my\_csv\_stage **AUTO\_COMPRESS=TRUE;**

Windows  
**PUT** **file://**C**:\tempload\contacts\*.csv** **@**my\_csv\_stage **AUTO\_COMPRESS=TRUE;**

Let us take a closer look at the command:

* file://<file-path>[/]contacts\*.csv specifies the full directory path and names of the files on your local machine to stage. Note that file system wildcards are allowed.
* @my\_csv\_stage is the stage name where to stage the data.
* auto\_compress=true; directs the command to compress the data when staging. This is also the default.

The command returns the following result, showing the staged files:

**+**---------------+------------------+-------------+-------------+--------------------+--------------------+----------+---------+

| source | target | source\_size | target\_size | source\_compression | target\_compression | status | message |

|---------------+------------------+-------------+-------------+--------------------+--------------------+----------+---------|

| contacts1.csv | contacts1.csv.gz | 694 | 506 | NONE | GZIP | UPLOADED | |

| contacts2.csv | contacts2.csv.gz | 763 | 565 | NONE | GZIP | UPLOADED | |

| contacts3.csv | contacts3.csv.gz | 771 | 567 | NONE | GZIP | UPLOADED | |

| contacts4.csv | contacts4.csv.gz | 750 | 561 | NONE | GZIP | UPLOADED | |

| contacts5.csv | contacts5.csv.gz | 887 | 621 | NONE | GZIP | UPLOADED | |

**+**---------------+------------------+-------------+-------------+--------------------+--------------------+----------+---------+

### **Staging the JSON sample data files**

Execute the PUT command to upload the JSON file from your local file system to the named stage.

Linux or macOS  
**PUT** **file:///**tmp**/load/contacts.json** **@**my\_json\_stage **AUTO\_COMPRESS=TRUE;**

Windows  
**PUT** **file://**C**:\tempload\contacts.json** **@**my\_json\_stage **AUTO\_COMPRESS=TRUE;**

The command returns the following result, showing the staged files:

**+**---------------+------------------+-------------+-------------+--------------------+--------------------+----------+---------+

| source | target | source\_size | target\_size | source\_compression | target\_compression | status | message |

|---------------+------------------+-------------+-------------+--------------------+--------------------+----------+---------|

| contacts.json | contacts.json.gz | 965 | 446 | NONE | GZIP | UPLOADED | |

**+**---------------+------------------+-------------+-------------+--------------------+--------------------+----------+---------+

### 

### **List the staged files**

### You can list the staged files by using the [LIST](https://docs.snowflake.com/en/sql-reference/sql/list) command.

#### **CSV**

**LIST** **@**my\_csv\_stage**;**

Snowflake returns a list of your staged files.

#### **JSON**

**LIST** **@**my\_json\_stage**;**

Snowflake returns a list of your staged files.

## **Step 5: Copy data into the target tables**

Execute [COPY INTO <table>](https://docs.snowflake.com/en/sql-reference/sql/copy-into-table) to load staged data into the target tables.

### **CSV**

To load the data from the sample CSV files:

Start by loading the data from one of the files (contacts1.csv.gz). Execute the following:  
**COPY** **INTO** mycsvtable

**FROM** **@**my\_csv\_stage**/**contacts1**.csv.**gz

**FILE\_FORMAT** **=** **(FORMAT\_NAME** **=** mycsvformat**)**

**ON\_ERROR** **=** 'skip\_file'**;**

Where:

* + The FROM clause specifies the location of the staged data file (stage name followed by the file name).
  + The ON\_ERROR clause specifies what to do when the COPY command encounters errors in the files. By default, the command stops loading data when the first error is encountered; however, we’ve instructed it to skip any file containing an error and move on to loading the next file. Note that this is just for illustration purposes; none of the files in this tutorial contain errors.

The COPY command returns a result showing the name of the file copied and related information:  
**+**-----------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------+

| file | status | rows\_parsed | rows\_loaded | error\_limit | errors\_seen | first\_error | first\_error\_line | first\_error\_character | first\_error\_column\_name |

|-----------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------|

| mycsvtable/contacts1.csv.gz | LOADED | 5 | 5 | 1 | 0 | NULL | NULL | NULL | NULL |

**+**-----------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------+

Load the rest of the staged files in the mycsvtable table.  
The following example uses pattern matching to load data from all files that match the regular expression .\*contacts[1-5].csv.gz into the mycsvtable table.  
**COPY** **INTO** mycsvtable

**FROM** **@**my\_csv\_stage

**FILE\_FORMAT** **=** **(FORMAT\_NAME** **=** mycsvformat**)**

**PATTERN=**'.\*contacts[1-5].csv.gz'

**ON\_ERROR** **=** 'skip\_file'**;**

Where the PATTERN clause specifies that the command should load data from the filenames matching this regular expression (.\*employees0[1-5].csv.gz).  
The COPY command returns a result showing the name of the file copied and related information:  
**+**-----------------------------+-------------+-------------+-------------+-------------+-------------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------+------------------+-----------------------+-------------------------+

| file | status | rows\_parsed | rows\_loaded | error\_limit | errors\_seen | first\_error | first\_error\_line | first\_error\_character | first\_error\_column\_name |

|-----------------------------+-------------+-------------+-------------+-------------+-------------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------+------------------+-----------------------+-------------------------|

| mycsvtable/contacts2.csv.gz | LOADED | 5 | 5 | 1 | 0 | NULL | NULL | NULL | NULL |

| mycsvtable/contacts3.csv.gz | LOAD\_FAILED | 5 | 0 | 1 | 2 | Number of columns in file (11) does not match that of the corresponding table (10), use file format option error\_on\_column\_count\_mismatch=false to ignore this error | 3 | 1 | "MYCSVTABLE"[11] |

| mycsvtable/contacts4.csv.gz | LOADED | 5 | 5 | 1 | 0 | NULL | NULL | NULL | NULL |

| mycsvtable/contacts5.csv.gz | LOADED | 6 | 6 | 1 | 0 | NULL | NULL | NULL | NULL |

**+**-----------------------------+-------------+-------------+-------------+-------------+-------------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------+------------------+-----------------------+-------------------------+

Note the following highlights in the result:

* + The data in contacts1.csv.gz is ignored because you already loaded the data successfully.
  + The data in these files was loaded successfully: contacts2.csv.gz, contacts4.csv.gz, and contacts5.csv.gz.
  + The data in contacts3.csv.gz was skipped due to 2 data errors. The next step in this tutorial addresses how to validate and fix the errors.

### **JSON**

Load the contacts.json.gz staged data file into the myjsontable table.

**COPY** **INTO** myjsontable

**FROM** **@**my\_json\_stage**/contacts.json.**gz

**FILE\_FORMAT** **=** **(FORMAT\_NAME** **=** myjsonformat**)**

**ON\_ERROR** **=** 'skip\_file'**;**

The COPY command returns a result showing the name of the file copied and related information:

**+**------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------+

| file | status | rows\_parsed | rows\_loaded | error\_limit | errors\_seen | first\_error | first\_error\_line | first\_error\_character | first\_error\_column\_name |

|------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------|

| myjsontable/contacts.json.gz | LOADED | 3 | 3 | 1 | 0 | NULL | NULL | NULL | NULL |

**+**------------------------------+--------+-------------+-------------+-------------+--------

## **Step 6: Resolve data load errors**

In the preceding step, the COPY INTO command skipped loading one of the files when it encountered the first error. You need to find all the errors and fix them. In this step, you use the [VALIDATE](https://docs.snowflake.com/en/sql-reference/functions/validate) function to validate the previous execution of the COPY INTO command and returns all errors.

### **Validate the sample data files and retrieve any errors**

You first need the query ID associated with the COPY INTO command that you previously executed. You then call the VALIDATE function, specifying the query ID.

1. Retrieve the query ID.
   1. Sign in to Snowsight.
   2. Make sure the role in Snowsight is the same as the role you are using in SnowSQL to run SQL statements for this tutorial.
   3. Select **Monitoring** » **Query History**.
   4. Select the row for the specific COPY INTO command to open the query information pane.
   5. Copy the **Query ID** value.
2. Validate the COPY INTO command execution, represented by the query ID, and save errors to a new table named save\_copy\_errors.

In SnowSQL, execute the following command. Replace *query\_id* with the **Query ID** value.  
**CREATE** **OR** **REPLACE** **TABLE** save\_copy\_errors **AS** **SELECT** **\*** **FROM** **TABLE(**VALIDATE**(**mycsvtable**,** **JOB\_ID=>**'01bd6a43-0000-2372-0000-00259f8b6895'**));**

Query the save\_copy\_errors table.  
**SELECT** **\*** **FROM** SAVE\_COPY\_ERRORS**;**

The query returns the following results:  
**+**----------------------------------------------------------------------------------------------------------------------------------------------------------------------+-------------------------------------+------+-----------+-------------+----------+--------+-----------+-------------------------------+------------+----------------+-----------------------------------------------------------------------------------------------------------------------------------------------------+

| ERROR | FILE | LINE | CHARACTER | BYTE\_OFFSET | CATEGORY | CODE | SQL\_STATE | COLUMN\_NAME | ROW\_NUMBER | ROW\_START\_LINE | REJECTED\_RECORD |

|----------------------------------------------------------------------------------------------------------------------------------------------------------------------+-------------------------------------+------+-----------+-------------+----------+--------+-----------+-------------------------------+------------+----------------+-----------------------------------------------------------------------------------------------------------------------------------------------------|

| Number of columns in file (11) does not match that of the corresponding table (10), use file format option error\_on\_column\_count\_mismatch=false to ignore this error | mycsvtable/contacts3.csv.gz | 3 | 1 | 234 | parsing | 100080 | 22000 | "MYCSVTABLE"[11] | 1 | 2 | 11|Ishmael|Burnett|Dolor Elit Pellentesque Ltd|vitae.erat@necmollisvitae.ca|1-872|600-7301|1-513-592-6779|P.O. Box 975, 553 Odio, Road|Hulste|63345 |

| Field delimiter '|' found while expecting record delimiter '\n' | mycsvtable/contacts3.csv.gz | 5 | 125 | 625 | parsing | 100016 | 22000 | "MYCSVTABLE"["POSTALCODE":10] | 4 | 5 | 14|Sophia|Christian|Turpis Ltd|lectus.pede@non.ca|1-962-503-3253|1-157-|850-3602|P.O. Box 824, 7971 Sagittis Rd.|Chattanooga|56188 |

**+**----------------------------------------------------------------------------------------------------------------------------------------------------------------------+-------------------------------------+------+-----------+-------------+----------+--------+-----------+-------------------------------+------------+----------------+-----------------------------------------------------------------------------------------------------------------------------------------------------+

The result shows two data errors in mycsvtable/contacts3.csv.gz:

* Number of columns in file (11) does not match that of the corresponding table (10)  
  In Row 1, a hyphen was mistakenly replaced with the pipe (|) character, the data file delimiter, effectively creating an additional column in the record.
* Example 1 data error in record
* Field delimiter '|' found while expecting record delimiter 'n'  
  In Row 5, an additional pipe (|) character was introduced after a hyphen, breaking the record.
* Example 1 data error in record

### **Step 7: Fix the errors and load the data files again**

1. Fix the errors in the records manually in the contacts3.csv file in your local environment.
2. Use the [PUT](https://docs.snowflake.com/en/sql-reference/sql/put) command to upload the modified data file to the stage. The modified file overwrites the existing staged file.

Linux or macOS:  
**PUT** **file:///**tmp**/load/**contacts3**.csv** **@**my\_csv\_stage **AUTO\_COMPRESS=TRUE** **OVERWRITE=TRUE;**

Windows:  
**PUT** **file://**C**:\tempload\**contacts3**.csv** **@**my\_csv\_stage **AUTO\_COMPRESS=TRUE** **OVERWRITE=TRUE;**

Copy the data from the staged files into the tables.  
**COPY** **INTO** mycsvtable

**FROM** **@**my\_csv\_stage**/**contacts3**.csv.**gz

**FILE\_FORMAT** **=** **(FORMAT\_NAME** **=** mycsvformat**)**

**ON\_ERROR** **=** 'skip\_file'**;**

Snowflake returns the following results, indicating the data in contacts3.csv.gz was loaded successfully.

**+**-----------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------+

| file | status | rows\_parsed | rows\_loaded | error\_limit | errors\_seen | first\_error | first\_error\_line | first\_error\_character | first\_error\_column\_name |

|-----------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------|

| mycsvtable/contacts3.csv.gz | LOADED | 5 | 5 | 1 | 0 | NULL | NULL | NULL | NULL |

**+**-----------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------+

### **Step 8: Verify the loaded data**

Execute a [SELECT](https://docs.snowflake.com/en/sql-reference/sql/select) query to verify that the data was loaded successfully.

#### **CSV**

**SELECT** **\*** **FROM** mycsvtable**;**

The query returns the following results:

**+**----+-----------+------------+----------------------------------+----------------------------------------+----------------+----------------+---------------------------------+------------------+------------+

| ID | LAST\_NAME | FIRST\_NAME | COMPANY | EMAIL | WORKPHONE | CELLPHONE | STREETADDRESS | CITY | POSTALCODE |

|----+-----------+------------+----------------------------------+----------------------------------------+----------------+----------------+---------------------------------+------------------+------------|

| 6 | Reed | Moses | Neque Corporation | eget.lacus@facilisis.com | 1-449-871-0780 | 1-454-964-5318 | Ap #225-4351 Dolor Ave | Titagarh | 62631 |

| 7 | Audrey | Franks | Arcu Eu Limited | eu.dui@aceleifendvitae.org | 1-527-945-8935 | 1-263-127-1173 | Ap #786-9241 Mauris Road | Bergen | 81958 |

| 8 | Jakeem | Erickson | A Ltd | Pellentesque.habitant@liberoProinmi.ca | 1-381-591-9386 | 1-379-391-9490 | 319-1703 Dis Rd. | Pangnirtung | 62399 |

| 9 | Xaviera | Brennan | Bibendum Ullamcorper Limited | facilisi.Sed.neque@dictum.edu | 1-260-757-1919 | 1-211-651-0925 | P.O. Box 146, 8385 Vel Road | Béziers | 13082 |

| 10 | Francis | Ortega | Vitae Velit Egestas Associates | egestas.rhoncus.Proin@faucibus.com | 1-257-584-6487 | 1-211-870-2111 | 733-7191 Neque Rd. | Chatillon | 33081 |

| 16 | Aretha | Sykes | Lobortis Tellus Justo Foundation | eget@Naminterdumenim.net | 1-670-849-1866 | 1-283-783-3710 | Ap #979-2481 Dui. Av. | Thurso | 66851 |

| 17 | Akeem | Casey | Pharetra Quisque Ac Institute | dictum.eu@magna.edu | 1-277-657-0361 | 1-623-630-8848 | Ap #363-6074 Ullamcorper, Rd. | Idar-Oberstei | 30848 |

| 18 | Keelie | Mendez | Purus In Foundation | Nulla.eu.neque@Aeneanegetmetus.co.uk | 1-330-370-8231 | 1-301-568-0413 | 3511 Tincidunt Street | Lanklaar | 73942 |

| 19 | Lane | Bishop | Libero At PC | non@dapibusligula.ca | 1-340-862-4623 | 1-513-820-9039 | 7459 Pede. Street | Linkebeek | 89252 |

| 20 | Michelle | Dickson | Ut Limited | Duis.dignissim.tempor@cursuset.org | 1-202-490-0151 | 1-129-553-7398 | 6752 Eros. St. | Stornaway | 61290 |

| 20 | Michelle | Dickson | Ut Limited | Duis.dignissim.tempor@cursuset.org | 1-202-490-0151 | 1-129-553-7398 | 6752 Eros. St. | Stornaway | 61290 |

| 21 | Lance | Harper | Rutrum Lorem Limited | Sed.neque@risus.com | 1-685-778-6726 | 1-494-188-6168 | 663-7682 Et St. | Gisborne | 73449 |

| 22 | Keely | Pace | Eleifend Limited | ante.bibendum.ullamcorper@necenim.edu | 1-312-381-5244 | 1-432-225-9226 | P.O. Box 506, 5233 Aliquam Av. | Woodlands County | 61213 |

| 23 | Sage | Leblanc | Egestas A Consulting | dapibus@elementum.org | 1-630-981-0327 | 1-301-287-0495 | 4463 Lorem Road | Woodlands County | 33951 |

| 24 | Marny | Holt | Urna Nec Luctus Associates | ornare@vitaeorci.ca | 1-522-364-3947 | 1-460-971-8360 | P.O. Box 311, 4839 Nulla Av. | Port Coquitlam | 36733 |

| 25 | Holly | Park | Mauris PC | Vestibulum.ante@Maecenasliberoest.org | 1-370-197-9316 | 1-411-413-4602 | P.O. Box 732, 8967 Eu Avenue | Provost | 45507 |

| 1 | Imani | Davidson | At Ltd | nec@sem.net | 1-243-889-8106 | 1-730-771-0412 | 369-6531 Molestie St. | Russell | 74398 |

| 2 | Kelsie | Abbott | Neque Sed Institute | lacus@pede.net | 1-467-506-9933 | 1-441-508-7753 | P.O. Box 548, 1930 Pede. Road | Campbellton | 27022 |

| 3 | Hilel | Durham | Pede Incorporated | eu@Craspellentesque.net | 1-752-108-4210 | 1-391-449-8733 | Ap #180-2360 Nisl. Street | Etalle | 84025 |

| 4 | Graiden | Molina | Sapien Institute | sit@fermentum.net | 1-130-156-6666 | 1-269-605-7776 | 8890 A, Rd. | Dundee | 70504 |

| 5 | Karyn | Howard | Pede Ac Industries | sed.hendrerit@ornaretortorat.edu | 1-109-166-5492 | 1-506-782-5089 | P.O. Box 902, 5398 Et, St. | Saint-Hilarion | 26232 |

| 11 | Ishmael | Burnett | Dolor Elit Pellentesque Ltd | vitae.erat@necmollisvitae.ca | 1-872-600-7301 | 1-513-592-6779 | P.O. Box 975, 553 Odio, Road | Hulste | 63345 |

| 12 | Ian | Fields | Nulla Magna Malesuada PC | rutrum.non@condimentumDonec.co.uk | 1-138-621-8354 | 1-369-126-7068 | P.O. Box 994, 7053 Quisque Ave | Ostra Vetere | 90433 |

| 13 | Xanthus | Acosta | Tortor Company | Nunc.lectus@a.org | 1-834-909-8838 | 1-693-411-2633 | 282-7994 Nunc Av. | Belcarra | 28890 |

| 14 | Sophia | Christian | Turpis Ltd | lectus.pede@non.ca | 1-962-503-3253 | 1-157-850-3602 | P.O. Box 824, 7971 Sagittis Rd. | Chattanooga | 56188 |

| 15 | Dorothy | Watson | A Sollicitudin Orci Company | diam.dictum@fermentum.co.uk | 1-158-596-8622 | 1-402-884-3438 | 3348 Nec Street | Qu�bec City | 63320 |

**+**----+-----------+------------+----------------------------------+----------------------------------------+----------------+----------------+---------------------------------+------------------+------------+

#### **JSON**

**SELECT** **\*** **FROM** myjsontable**;**

The query returns the following results:

**+**-----------------------------------------------------------------+

| JSON\_DATA |

|-----------------------------------------------------------------|

| { |

| "customer": { |

| "\_id": "5730864df388f1d653e37e6f", |

| "address": "509 Kings Hwy, Comptche, Missouri, 4848", |

| "company": "ORBIN", |

| "email": "blankenship.patrick@orbin.ca", |

| "name": { |

| "first": "Blankenship", |

| "last": "Patrick" |

| }, |

| "phone": "+1 (999) 407-2274" |

| } |

| } |

| { |

| "customer": { |

| "\_id": "5730864d4d8523c8baa8baf6", |

| "address": "290 Lefferts Avenue, Malott, Delaware, 1575", |

| "company": "SNIPS", |

| "email": "anna.glass@snips.name", |

| "name": { |

| "first": "Anna", |

| "last": "Glass" |

| }, |

| "phone": "+1 (958) 411-2876" |

| } |

| } |

| { |

| "customer": { |

| "\_id": "5730864e375e08523150fc04", |

| "address": "756 Randolph Street, Omar, Rhode Island, 3310", |

| "company": "ESCHOIR", |

| "email": "sparks.ramos@eschoir.co.uk", |

| "name": { |

| "first": "Sparks", |

| "last": "Ramos" |

| }, |

| "phone": "+1 (962) 436-2519" |

| } |

| } |

**+**-----------------------------------------------------------------+

## **Step 9: Remove the successfully copied data files**

After you verify that you successfully copied data from your stage into the tables, you can remove data files from the internal stage using the [REMOVE](https://docs.snowflake.com/en/sql-reference/sql/remove) command to save on [data storage](https://docs.snowflake.com/en/user-guide/cost-understanding-compute).

**REMOVE** **@**my\_csv\_stage **PATTERN=**'.\*.csv.gz'**;**

Snowflake returns the following results:

**+**-------------------------------+---------+

| name | result |

|-------------------------------+---------|

| my\_csv\_stage/contacts1.csv.gz | removed |

| my\_csv\_stage/contacts4.csv.gz | removed |

| my\_csv\_stage/contacts2.csv.gz | removed |

| my\_csv\_stage/contacts3.csv.gz | removed |

| my\_csv\_stage/contacts5.csv.gz | removed |

**+**-------------------------------+---------+

**REMOVE** **@**my\_json\_stage **PATTERN=**'.\*.json.gz'**;**

Snowflake returns the following results:

**+**--------------------------------+---------+

| name | result |

|--------------------------------+---------|

| my\_json\_stage/contacts.json.gz | removed |

**+**--------------------------------+---------+

## **Step 10: Clean up**

Congratulations, you have successfully completed the tutorial.

### **Tutorial clean up (optional)**

Execute the following [DROP <object>](https://docs.snowflake.com/en/sql-reference/sql/drop) commands to return your system to its state before you began the tutorial:

**DROP** **DATABASE** **IF** **EXISTS** mydatabase**;**

**DROP** **WAREHOUSE** **IF** **EXISTS** mywarehouse**;**

Dropping the database automatically removes all child database objects such as tables.