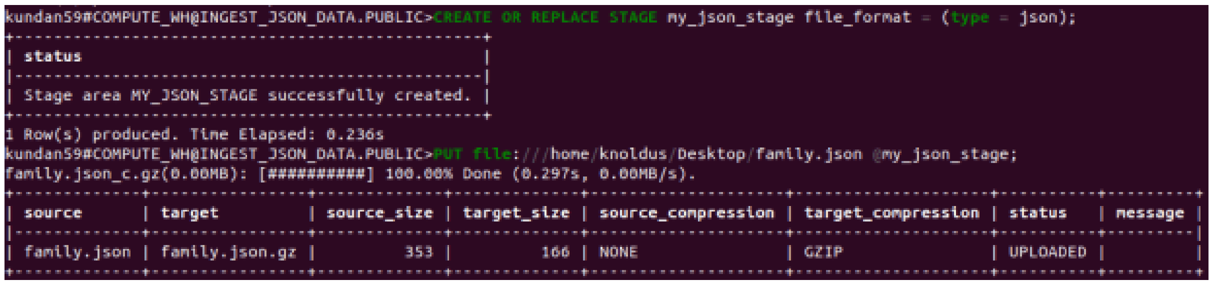
Snowflake: Loading JSON data into Snowflake

JSON is a relatively concise format. If we are implementing a database solution, it is very common that we will come across a system that provides data in JSON format. Snowflake has a very straight forward approach to load JSON data. In this blog, we will understand this approach in a step-wise manner.

1. Stage the JSON data

In snowflake Staging the data means, make the data available in Snowflake *stage*(intermediate storage) it can be *internal or external*. Staging JSON data in Snowflake is similar to staging any other files. Let's Staging JSON data file from a **local file system.**

```
CREATE OR REPLACE STAGE my_json_stage file_format = (type = json);

PUT file:///home/knoldus/Desktop/family.json @my_json_stage;
```



```
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>CREATE OR REPLACE STAGE my_json_stage file_format = (type = json);
+-----------------------------------------------+
| status                                        |
|-----------------------------------------------|
| Stage area MY_JSON_STAGE successfully created. |
+-----------------------------------------------+
1 Row(s) produced. Time Elapsed: 0.236s
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>PUT file:///home/knoldus/Desktop/family.json @my_json_stage;
family.json_c.gz(0.00MB): [##########] 100.00% Done (0.297s, 0.00MB/s).
+-------------+----------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source      | target         | source_size | target_size | source_compression | target_compression | status   | message |
|-------------+----------------+-------------+-------------+--------------------+--------------------+----------+---------|
| family.json | family.json.gz |         353 |         166 | NONE               | GZIP               | UPLOADED |         |
+-------------+----------------+-------------+-------------+--------------------+--------------------+----------+---------+
```

We can also create an external stage using the AWS S3 bucket, or Microsoft Azure blob storage that contains JSON data.

```
CREATE OR REPLACE STAGE my_json_stage url='url of s3 bucket or azure blob with credentials'
```

The JSON data looks like:

```
{

  "Name": "Aman Gupta",

  "family_detail": [
  {
   "Name": "Avinash Gupta",
   "Relationship": "Father",
  },
  {
   "Name": "Lata Gupta",
   "Relationship": "Mother",
  },
  {
   "Name": "Shrishti Gupta",
   "Relationship": "Sister",
  },
  {
   "Name": "Bobin Gupta",
   "Relationship": "Brother",
  }
 ]
}
```

2. Load JSON data as raw into temporary table

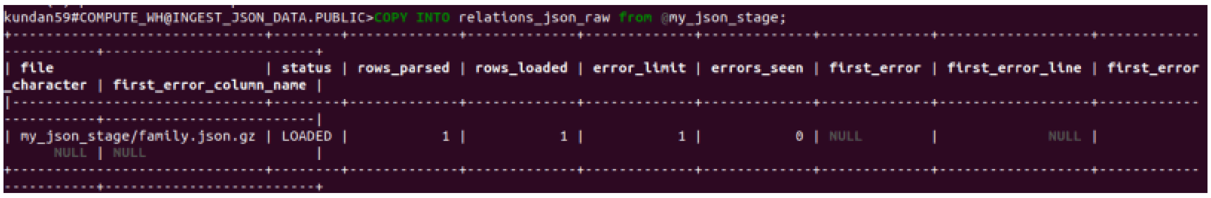
To load the JSON data as raw, first, create a table with a column of VARIANT type. VARIANT can contain any type of data so it is suitable for loading JSON data.

```
CREATE TABLE relations_json_raw (
 json_data_raw VARIANT
);
```

Now let’s **copy** the JSON file into **relations_json_raw** table.

```
COPY INTO relations_json_raw from @my_json_stage;
```

Note that a file format does not need to be specified because it is included in the stage definition.



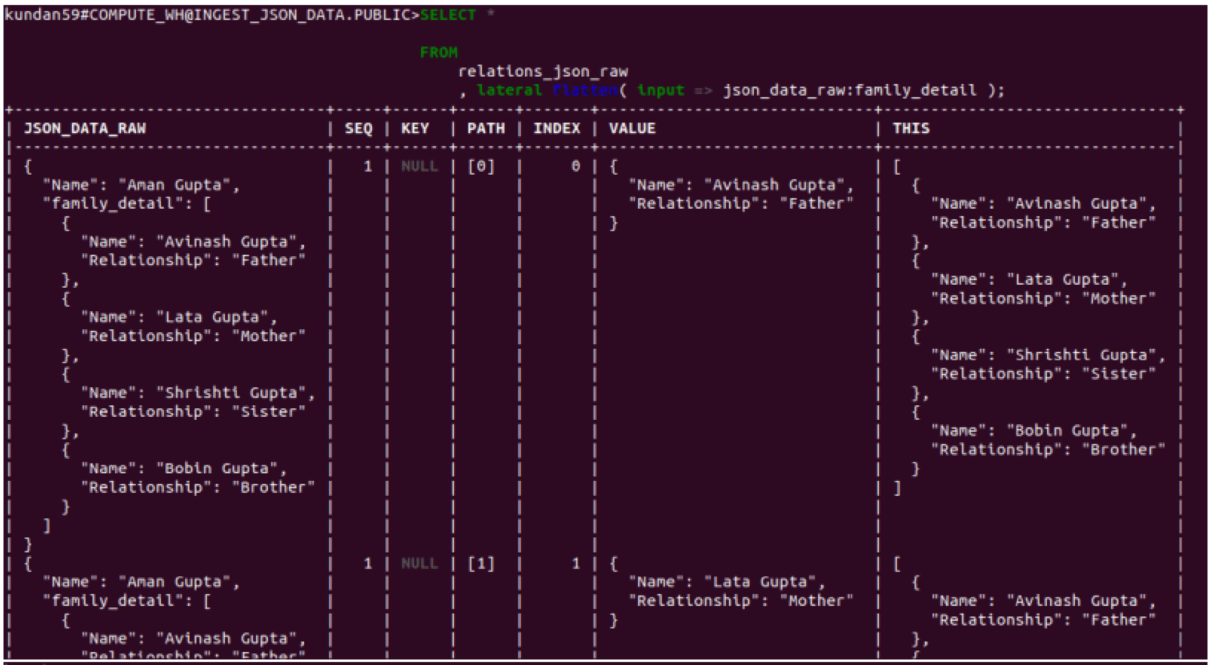
```
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>COPY INTO relations_json_raw from @my_json_stage;
+-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------
-----------+-------------------------+
| file                          | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error
_character | first_error_column_name |
|-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------
-----------+-------------------------|
| my_json_stage/family.json.gz | LOADED |           1 |           1 |           1 |           0 | NULL        |             NULL |
      NULL | NULL                    |
+-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+-----------
-----------+-------------------------+
```

3. Analyze and prepare raw JSON data

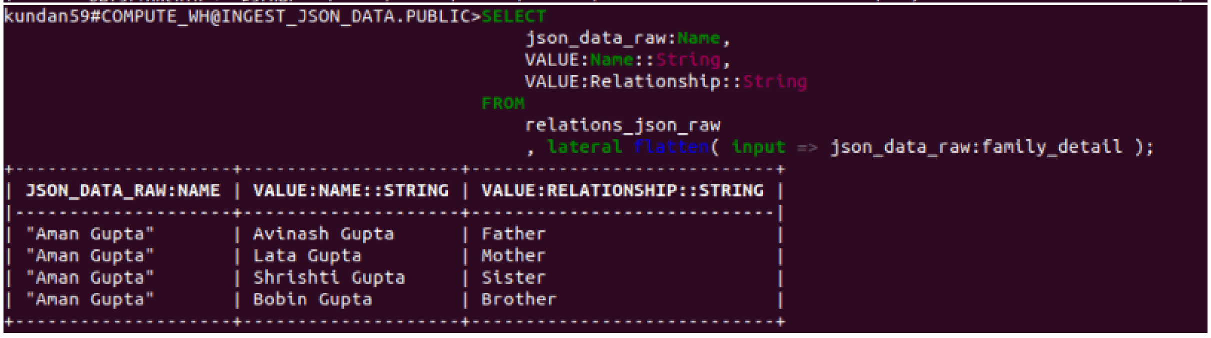
The next step would be **to analyze** the **loaded raw JSON data**. Determining what information needs to be extracted from JSON data. For example, in our case, we are interested to extract the name key and from the family_detail array object, we want to extract the name and relationship key from each JSON object. The below query will do that.

```
SELECT

   json_data_raw:Name,

   VALUE:Name::String,

   VALUE:Relationship::String

FROM

   relations_json_raw

   , lateral flatten( input => json_data_raw:family_detail );
```

The above query using **lateral join** and a **flatten functio**n. The flatten function returns a row for each JSON object from the family_detail array. and the lateral modifier joins the data with any information outside of the object, in our example candidate name that we are extracting with **json_data_raw: Name.**



```
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>SELECT *

                                            FROM
                                                relations_json_raw
                                                , lateral flatten( input => json_data_raw:family_detail );
+-------------------------------+-----+------+------+-------+---------------------------+-----------------------------+
| JSON_DATA_RAW                 | SEQ | KEY  | PATH | INDEX | VALUE                     | THIS                        |
|-------------------------------+-----+------+------+-------+---------------------------+-----------------------------|
| {                             |   1 | NULL | [0]  |     0 | {                         | [                           |
|   "Name": "Aman Gupta",       |     |      |      |       |   "Name": "Avinash Gupta",|   {                         |
|   "family_detail": [          |     |      |      |       |   "Relationship": "Father"|     "Name": "Avinash Gupta",|
|     {                         |     |      |      |       | }                         |     "Relationship": "Father"|
|       "Name": "Avinash Gupta",|     |      |      |       |                           |   },                        |
|       "Relationship": "Father"|     |      |      |       |                           |   {                         |
|     },                        |     |      |      |       |                           |     "Name": "Lata Gupta",   |
|     {                         |     |      |      |       |                           |     "Relationship": "Mother"|
|       "Name": "Lata Gupta",   |     |      |      |       |                           |   },                        |
|       "Relationship": "Mother"|     |      |      |       |                           |   {                         |
|     },                        |     |      |      |       |                           |     "Name": "Shrishti Gupta",|
|     {                         |     |      |      |       |                           |     "Relationship": "Sister"|
|       "Name": "Shrishti Gupta",|    |      |      |       |                           |   },                        |
|       "Relationship": "Sister"|     |      |      |       |                           |   {                         |
|     },                        |     |      |      |       |                           |     "Name": "Bobin Gupta",  |
|     {                         |     |      |      |       |                           |     "Relationship": "Brother"|
|       "Name": "Bobin Gupta",  |     |      |      |       |                           |   }                         |
|       "Relationship": "Brother"|    |      |      |       |                           | ]                           |
|     }                         |     |      |      |       |                           |                             |
|   ]                           |     |      |      |       |                           |                             |
| }                             |     |      |      |       |                           |                             |
| {                             |   1 | NULL | [1]  |     1 | {                         | [                           |
|   "Name": "Aman Gupta",       |     |      |      |       |   "Name": "Lata Gupta",   |   {                         |
|   "family_detail": [          |     |      |      |       |   "Relationship": "Mother"|     "Name": "Avinash Gupta",|
|     {                         |     |      |      |       | }                         |     "Relationship": "Father"|
|       "Name": "Avinash Gupta",|     |      |      |       |                           |   },                        |
```

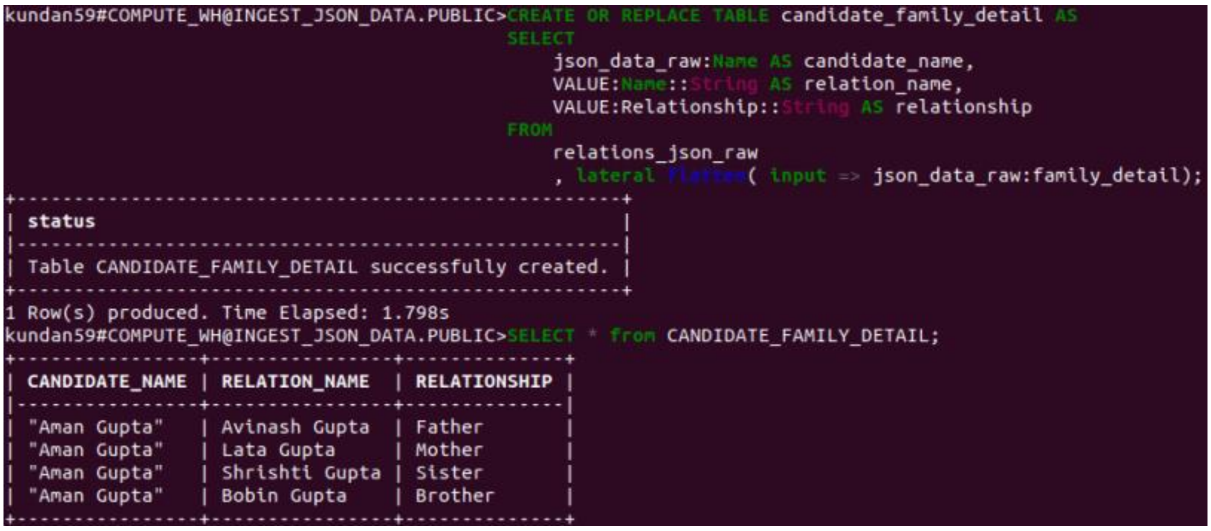


```
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>SELECT
                                                json_data_raw:Name,
                                                VALUE:Name::String,
                                                VALUE:Relationship::String
                                            FROM
                                                relations_json_raw
                                                , lateral flatten( input => json_data_raw:family_detail );
+--------------------+---------------------+-----------------------------+
| JSON_DATA_RAW:NAME | VALUE:NAME::STRING  | VALUE:RELATIONSHIP::STRING |
|--------------------+---------------------+-----------------------------|
| "Aman Gupta"       | Avinash Gupta       | Father                      |
| "Aman Gupta"       | Lata Gupta          | Mother                      |
| "Aman Gupta"       | Shrishti Gupta      | Sister                      |
| "Aman Gupta"       | Bobin Gupta         | Brother                     |
+--------------------+---------------------+-----------------------------+
```

Load Data into target table

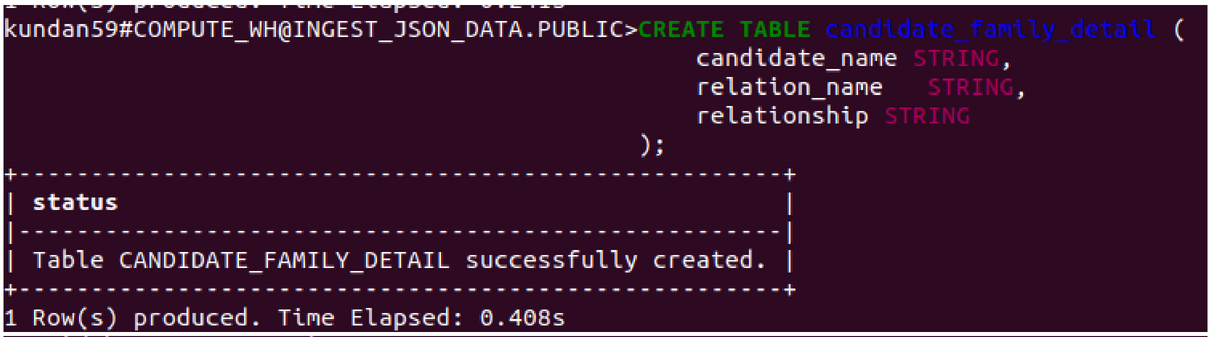
Now we have analyzed and extracted information. We can load the extracted data into the target table.

```
CREATE OR REPLACE TABLE candidate_family_detail AS

SELECT

   json_data_raw:Name AS candidate_name,

   VALUE:Name::String AS relation_name,

   VALUE:Relationship::String AS relationship



FROM

   relations_json_raw

   , lateral flatten( input => json_data_raw:family_detail );
```

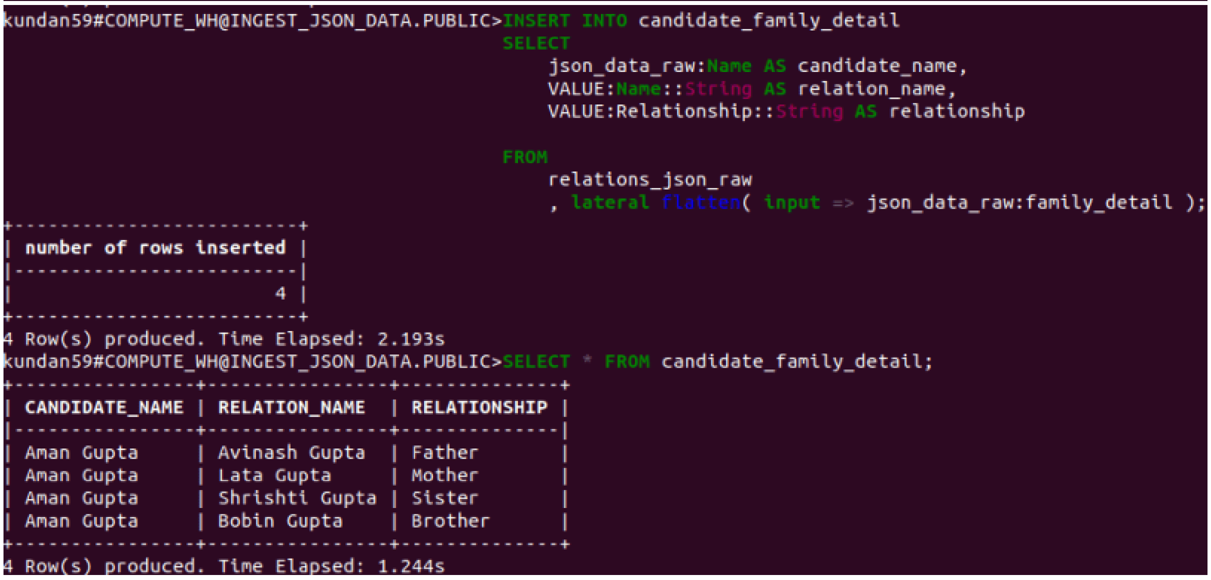


```
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>CREATE OR REPLACE TABLE candidate_family_detail AS
                                            SELECT
                                                json_data_raw:Name AS candidate_name,
                                                VALUE:Name::String AS relation_name,
                                                VALUE:Relationship::String AS relationship
                                            FROM
                                                relations_json_raw
                                                , lateral flatten( input => json_data_raw:family_detail);
+-------------------------------------------------+
| status                                          |
|-------------------------------------------------|
| Table CANDIDATE_FAMILY_DETAIL successfully created. |
+-------------------------------------------------+
1 Row(s) produced. Time Elapsed: 1.798s
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>SELECT * from CANDIDATE_FAMILY_DETAIL;
+----------------+----------------+--------------+
| CANDIDATE_NAME | RELATION_NAME  | RELATIONSHIP |
|----------------+----------------+--------------|
| "Aman Gupta"   | Avinash Gupta  | Father       |
| "Aman Gupta"   | Lata Gupta     | Mother       |
| "Aman Gupta"   | Shrishti Gupta | Sister       |
| "Aman Gupta"   | Bobin Gupta    | Brother      |
+----------------+----------------+--------------+
```

If you don’t want to do a “create table as”, you can pre-create a table and then insert the JSON data into the table.



```
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>CREATE TABLE candidate_family_detail (
                                                candidate_name STRING,
                                                relation_name   STRING,
                                                relationship STRING
                                            );
+-----------------------------------------------------+
| status                                              |
|-----------------------------------------------------|
| Table CANDIDATE_FAMILY_DETAIL successfully created. |
+-----------------------------------------------------+
1 Row(s) produced. Time Elapsed: 0.408s
```



```
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>INSERT INTO candidate_family_detail
                                            SELECT
                                                json_data_raw:Name AS candidate_name,
                                                VALUE:Name::String AS relation_name,
                                                VALUE:Relationship::String AS relationship

                                            FROM
                                                relations_json_raw
                                                , lateral flatten( input => json_data_raw:family_detail );
+-------------------------+
| number of rows inserted |
|-------------------------|
|                       4 |
+-------------------------+
4 Row(s) produced. Time Elapsed: 2.193s
kundan59#COMPUTE_WH@INGEST_JSON_DATA.PUBLIC>SELECT * FROM candidate_family_detail;
+----------------+----------------+--------------+
| CANDIDATE_NAME | RELATION_NAME  | RELATIONSHIP |
|----------------+----------------+--------------|
| Aman Gupta     | Avinash Gupta  | Father       |
| Aman Gupta     | Lata Gupta     | Mother       |
| Aman Gupta     | Shrishti Gupta | Sister       |
| Aman Gupta     | Bobin Gupta    | Brother      |
+----------------+----------------+--------------+
4 Row(s) produced. Time Elapsed: 1.244s
```

Note: If you are loading JSON data recursively, the process needs to be setup in such a way that you can identify which row’s already exists in the target table or which row’s are new.