Storing Semi-structured Data in Snowflake Cloud Data Warehouse

BA Basu, Abhi (CAI - Austin)
Principal Technical Architect

A modern data warehouse needs to support the storage of structured and unstructured formats of data. The support for semi-structured data formats like JSON and XML (which may contain n-level of hierarchical data) is essential, as these formats have become an efficient method of data exchange by web services, machines and applications. Earlier in my career, I have designed Data Platforms with amalgamation of multiple database technologies like distributed systems for structured big data (Impala, Drill, Presto), RDBMS for structured small data (MySQL, PostgresSQL, SQLServer, Oracle) and document-based NoSQL databases for semi-structured data like XML and JSON (MongoDB, DynamoDB, CouchBase). With the availability of a true cloud data platform like Snowflake (https://www.snowflake.com/) we have the ability to store diverse data formats like structured (CSV, TSV etc.) and semi-structured (JSON, XML, Avro, ORC, Parquet), together in one data warehouse that is scalable to handle the storage and retrieval requirements of big data.

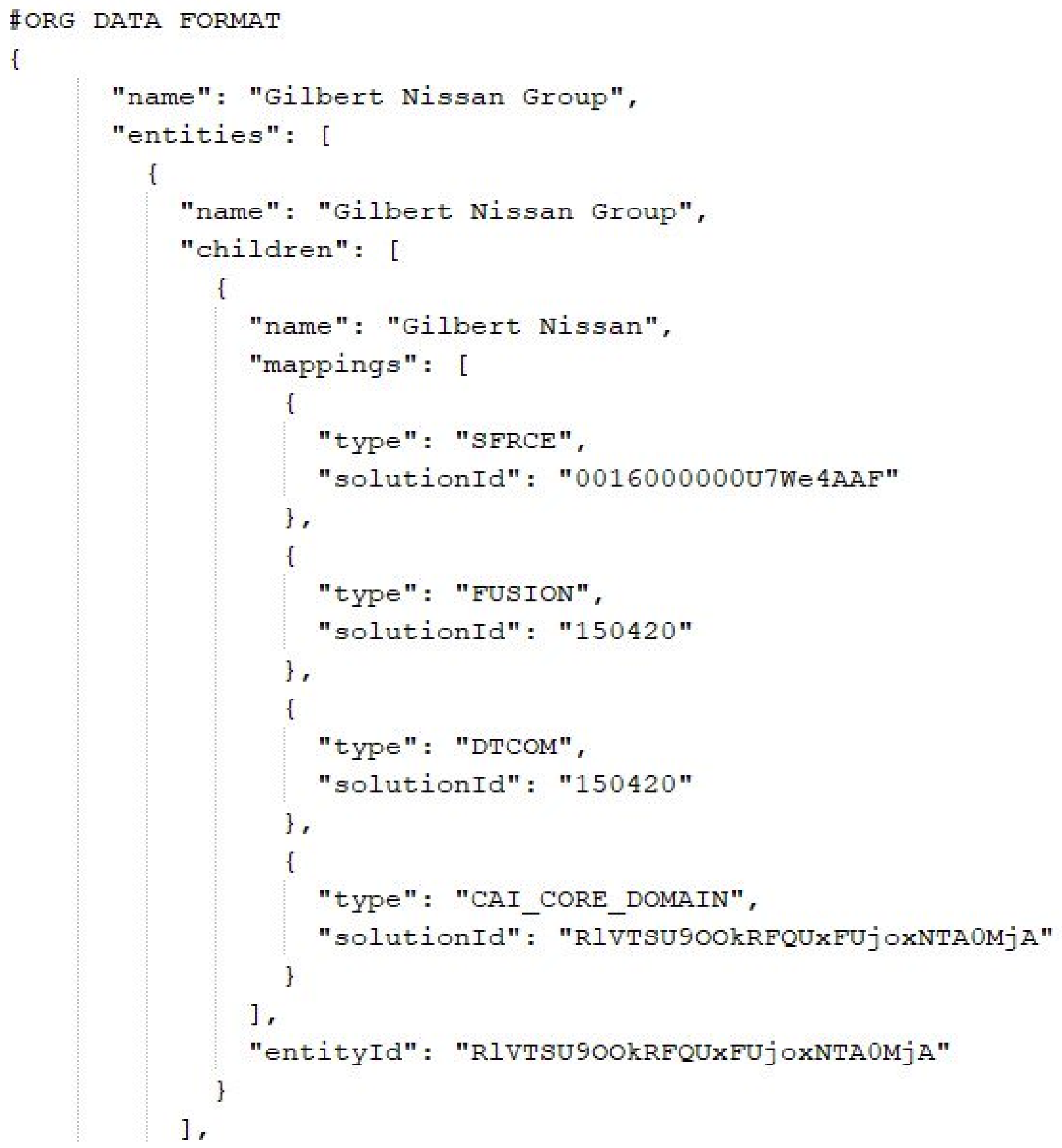
Snowflake offers a VARIANT datatype/column to handle various semi-structured data formats. Unstructured data can be loaded into relational tables without having to specify a schema up-front. Once the data is loaded into a VARIANT column, it can be directly queried using SQL. However, a better method is to extract the values and objects from the data arrays (from the VARIANT column) and build structured tables or views that can be queried efficiently.

In the following sections, I will describe how to ingest a JSON file into Snowflake and query it using SQL.

Dataset

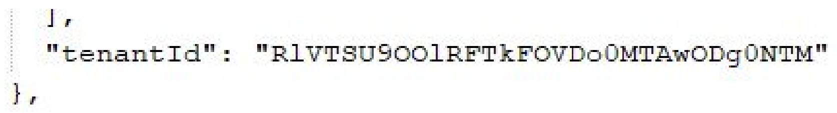
various products (in JSON format) and I wanted to investigate whether it made sense to store it in a Snowflake table and share with product teams as needed. This file is usually updated daily, so we could utilize a service like Snowpipe (https://www.snowflake.com/news/snowflake-introduces-snowpipe-continuous-automated-cost-effective-data-loading/#:~:text=Snowpipe%20is%20an%20automated%20service,loads%20that%20data%20into%20Snowflake.) to automate the update process.

The following is an example of the structure of this JSON file.



```
#ORG DATA FORMAT
{
    "name": "Gilbert Nissan Group",
    "entities": [
      {
        "name": "Gilbert Nissan Group",
        "children": [
          {
            "name": "Gilbert Nissan",
            "mappings": [
              {
                "type": "SFRCE",
                "solutionId": "0016000000U7We4AAF"
              },
              {
                "type": "FUSION",
                "solutionId": "150420"
              },
              {
                "type": "DTCOM",
                "solutionId": "150420"
              },
              {
                "type": "CAI_CORE_DOMAIN",
                "solutionId": "RlVTSU9OOkRFQUxFUjoxNTA0MjA"
              }
            ],
            "entityId": "RlVTSU9OOkRFQUxFUjoxNTA0MjA"
          }
        ],
```

Skip to main content



```
    ],
    "tenantId": "RlVTSU9OOlRFTkFOVDo0MTAwODg0NTM"
  },
```

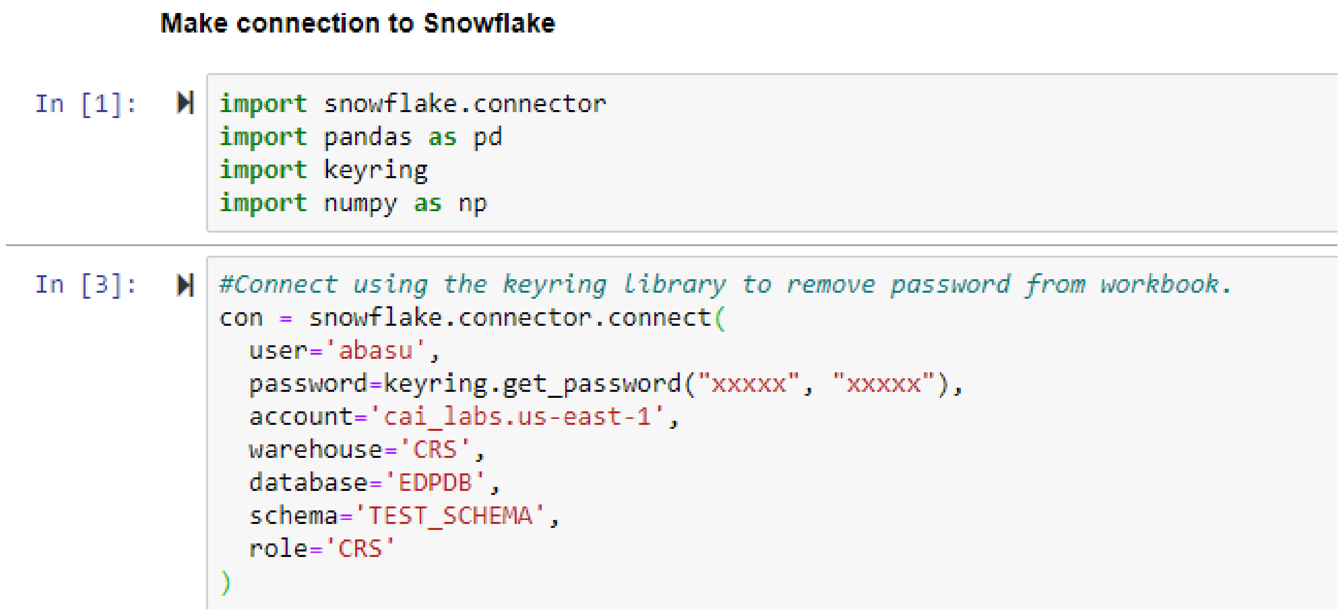
The Steps

Note: The examples use Jupyter Notebook and Python to connect to my Snowflake lab account (using the Snowflake Python connector). You can certainly use the Snowflake web client to run the commands.

1. We need to copy the source file to a S3 bucket and folder.

2. Go to Cox ALKS tool to get AWS_KEY_ID, AWS_SECRET_KEY and AWS_TOKEN - https://beta.alks.coxautoinc.com/overview?account=245928119370 , to be used in step

3. Build a Stage (https://docs.snowflake.com/en/sql-reference/sql/create-stage.html#:~:text=Creates%20a%20new%20named%20internal,a%20Stage%20for%20Local%20Files.) pointing to the source file on S3.



Make connection to Snowflake

```
In [1]:  import snowflake.connector
         import pandas as pd
         import keyring
         import numpy as np
```

```
In [3]:  #Connect using the keyring library to remove password from workbook.
         con = snowflake.connector.connect(
           user='abasu',
           password=keyring.get_password("xxxxx", "xxxxx"),
           account='cai_labs.us-east-1',
           warehouse='CRS',
           database='EDPDB',
           schema='TEST_SCHEMA',
           role='CRS'
         )
```

Note: The Keyring python library is used above as an safe way to store passwords in the local machine (https://pypi.org/project/keyring/).

Skip to main content

```
In [7]:  #Get only data where sale_price was part of the Transaction
         sql1 = """use schema CRS.ABHI_CRS;
         create or replace stage my_s3_stage
         URL='s3://awscoxautolabs199-abasu-dev/orgdata/'
         CREDENTIALS=(
           AWS_KEY_ID='xxxxxxx'
           AWS_SECRET_KEY='xxxxxx'
           AWS_TOKEN='xxxxxx'
         );"""

         con.execute_string(sql1)

Out[7]:  [<snowflake.connector.cursor.SnowflakeCursor at 0x227a1c98cc0>,
          <snowflake.connector.cursor.SnowflakeCursor at 0x227a1c981d0>]
```

4. Create a table with one column to store the JSON data in variant type column.

```
In [8]:  sql2 = """create or replace table CRS.ABHI_CRS.ORG_DATA (JSON_DATA VARIANT);"""
         con.execute_string(sql2)

Out[8]:  [<snowflake.connector.cursor.SnowflakeCursor at 0x227a1c98f28>]
```

5. Copy JSON data from Stage into table.

```
In [10]:  sql3 = """COPY INTO CRS.ABHI_CRS.ORG_DATA from '@my_s3_stage/orgdata.json' file_format = (type = 'JSON' strip_outer_array = false);"""
          con.execute_string(sql3)

Out[10]:  [<snowflake.connector.cursor.SnowflakeCursor at 0x227a1c71c88>]
```

6. Examine the contents of the table to ensure data has been copied correctly. The VARIANT column should have each JSON object separated into rows in the table.

Skip to main content

```
In [25]:  import pandas as pd
           sql4 = """select * from ABHI_CRS.ORG_DATA limit 10;"""
           # Create empty dataframe
           org_data_df = pd.DataFrame()
           cur = con.cursor().execute(sql4)
           org_data_df = pd.DataFrame.from_records(iter(cur), columns=[x[0] for x in cur.description])
           pd.set_option('max_colwidth', 400)
           org_data_df.head(10)
```

Out[25]:

	JSON_DATA
0	{\n "entities": [\n {\n "children": [\n {\n "entityId": "RlVTSU9OOkRFQUxFUjoxNDE0MzQ",\n "mappings": [\n {\n "solutionId": "00130000002c20hAAA",\n "type": "SFRCE"\n },\n {\n "solutionId": "141434",\n "type": "FUSION"\n },\n {\n "solutionId": ...
1	{\n "entities": [\n {\n "children": [\n {\n "entityId": "RlVTSU9OOkRFQUxFUjo0MTAxMjU0MzU",\n "mappings": [\n {\n "solutionId": "0010e00001KTPlHAAX",\n "type": "SFRCE"\n },\n {\n "solutionId": "9050492",\n "type": "HME"\n },\n {\n "solutionId"...
2	{\n "entities": [\n {\n "children": [\n {\n "entityId": "RlVTSU9OOkRFQUxFUjo0MTAwODM3NDk",\n "mappings": [\n {\n "solutionId": "0013200001DmPclAAF",\n "type": "SFRCE"\n },\n {\n "solutionId": "42859878",\n "type": "AT2"\n },\n {\n "solutionId...
3	{\n "entities": [\n {\n "children": [\n {\n "entityId": "RlVTSU9OOkRFQUxFUjo0MTAwMjQyODA",\n "mappings": [\n {\n "solutionId": "00160000012uMMjAAM",\n "type": "SFRCE"\n },\n {\n "solutionId": "410024280",\n "type": "FUSION"\n },\n {\n "soluti...
4	{\n "entities": [\n {\n "entityId": "RlVTSU9OOkdST1VQOjQxMDA3NjYwNg",\n "name": "Autonation Honda Columbus Group"\n }\n],\n "name": "Autonation Honda Columbus Group",\n "tenantId": "RlVTSU9OOlRFTkFOVDo0MTAwNzY2MDY"\n}
5	{\n "entities": [\n {\n "entityId": "RlVTSU9OOkdST1VQOjExNjg4NA",\n "name": "Thrifty Car Sales Group 882"\n }\n],\n "name": "Thrifty Car Sales Group 882",\n "tenantId": "RlVTSU9OOlRFTkFOVDoxMTY4ODQ"\n}
6	{\n "entities": [\n {\n "children": [\n {\n "entityId": "RlVTSU9OOkRFQUxFUjo0MTAwMTAyOTY",\n "mappings": [\n {\n "solutionId": "0016000000zJeVwAAK",\n "type": "SFRCE"\n },\n {\n "solutionId": "411890",\n "type": "AHFC"\n },\n {\n "solutionId"...
7	{\n "entities": [\n {\n "children": [\n {\n "entityId": "RlVTSU9OOkRFQUxFUjoyMzQwOTQ",\n "mappings": [\n {\n "solutionId": "0016000000hb0JfAAI",\n "type": "SFRCE"\n },\n {\n "solutionId": "234094",\n "type": "FUSION"\n },\n {\n "solutionId": ...

7. Create a view with all columns extracted out from JSON data stored in table. I am using the FLATTEN function here (https://docs.snowflake.com/en/sql-reference/functions/flatten.html).

Skip to main content

```
In [26]: sql5 = """CREATE OR REPLACE VIEW CRS.ABHI_CRS.ORG_DATA_MAPPING as
         select JSON_DATA:name::string as tenant_name, JSON_DATA:tenantId::string as tenant_id,
         e.value:name::string as entity_name, e.value:entityId::string as entity_id,
         c.value:name::string as child_name, c.value:entityId::string as child_entity_id,
         m.value:type::string as mapping_type, m.value:solutionId::string as mapping_soln_id
         from ABHI_CRS.ORG_DATA,
         lateral flatten(input => JSON_DATA:entities) e,
         lateral flatten(input => e.value:children) c,
         lateral flatten(input => c.value:mappings) m;
         """
         con.execute_string(sql5)

Out[26]: [<snowflake.connector.cursor.SnowflakeCursor at 0x227a3bf7be0>]
```

8. Examine data content and format of the newly created view. If some fields are not broken up in the correct columns, this would be the time to adjust the DDL statement above to rectify that.

```
In [27]: sql6 = """select * from CRS.ABHI_CRS.ORG_DATA_MAPPING where tenant_id = 'RlVTSU9OOlRFTkFOVDo0MTAxMjU0MzY';"""
         org_data_df = pd.DataFrame()
         cur = con.cursor().execute(sql6)
         org_data_df = pd.DataFrame.from_records(iter(cur), columns=[x[0] for x in cur.description])
         org_data_df.head(10)
```

Out[27]:

	TENANT_NAME	TENANT_ID	ENTITY_NAME	ENTITY_ID	CHILD_NAME	CHILD_ENTITY_ID	MA
0	Green Mount Rd Harley-Davidson Group	RIVTSU9OOIRFTkFOVDo0MTAxMjU0MzY	Green Mount Rd Harley-Davidson Group	RIVTSU9OOkdST1VQOjQxMDEyNTQzNg	Green Mount Rd Harley-Davidson	RIVTSU9OOkRFQUxFUjo0MTAxMjU0MzU	
1	Green Mount Rd Harley-Davidson Group	RIVTSU9OOIRFTkFOVDo0MTAxMjU0MzY	Green Mount Rd Harley-Davidson Group	RIVTSU9OOkdST1VQOjQxMDEyNTQzNg	Green Mount Rd Harley-Davidson	RIVTSU9OOkRFQUxFUjo0MTAxMjU0MzU	
2	Green Mount Rd Harley-Davidson Group	RIVTSU9OOIRFTkFOVDo0MTAxMjU0MzY	Green Mount Rd Harley-Davidson Group	RIVTSU9OOkdST1VQOjQxMDEyNTQzNg	Green Mount Rd Harley-Davidson	RIVTSU9OOkRFQUxFUjo0MTAxMjU0MzU	
	Group		Davidson Group		Davidson		

Skip to main content

9. Now we can issue a simple SQL query and compare the data representation in the view vs. the source JSON file. Here we are trying to compare a particular org hierarchy to validate the data.

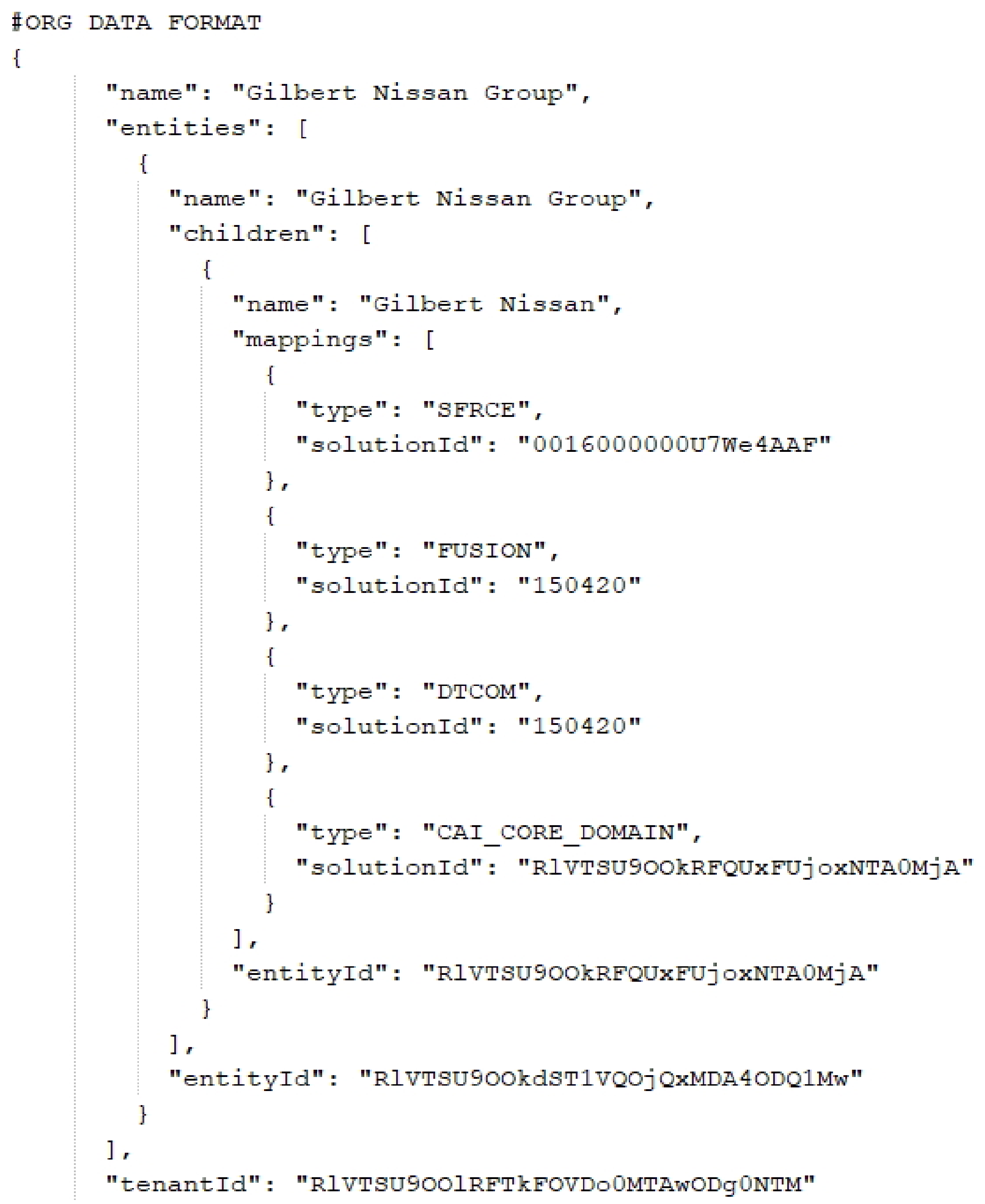
```
In [28]:  sql7 = """
          select * from CRS.ABHI_CRS.ORG_DATA_MAPPING where tenant_name = 'Gilbert Nissan Group';"""
          # Create empty dataframe
          org_data_df = pd.DataFrame()
          cur = con.cursor().execute(sql7)
          org_data_df = pd.DataFrame.from_records(iter(cur), columns=[x[0] for x in cur.description])
          org_data_df.head(10)
```

Out[28]:

	TENANT_NAME	TENANT_ID	ENTITY_NAME	ENTITY_ID	CHILD_NAME	CHILD_ENTITY_ID	MAPPING_TYPE	MAPPING_SOLN_ID
0	Gilbert Nissan Group	RIVTSU9OOIRFTkFOVDo0MTAwODg0NTM	Gilbert Nissan Group	RIVTSU9OOkdST1VQOjQxMDA4ODQ1Mw	Gilbert Nissan	RIVTSU9OOkRFQUxFUjoxNTA0MjA	SFRCE	0016000000U7We4AAF
1	Gilbert Nissan Group	RIVTSU9OOIRFTkFOVDo0MTAwODg0NTM	Gilbert Nissan Group	RIVTSU9OOkdST1VQOjQxMDA4ODQ1Mw	Gilbert Nissan	RIVTSU9OOkRFQUxFUjoxNTA0MjA	FUSION	150420
2	Gilbert Nissan Group	RIVTSU9OOIRFTkFOVDo0MTAwODg0NTM	Gilbert Nissan Group	RIVTSU9OOkdST1VQOjQxMDA4ODQ1Mw	Gilbert Nissan	RIVTSU9OOkRFQUxFUjoxNTA0MjA	DTCOM	150420
3	Gilbert Nissan Group	RIVTSU9OOIRFTkFOVDo0MTAwODg0NTM	Gilbert Nissan Group	RIVTSU9OOkdST1VQOjQxMDA4ODQ1Mw	Gilbert Nissan	RIVTSU9OOkRFQUxFUjoxNTA0MjA	CAI_CORE_DOMAIN	RIVTSU9OOkRFQUxFUjoxNTA0MjA

You will notice above how the data arrays have been flattened into the respective columns and rows. The following JSON object was translated into the table section shown above.

Skip to main content



```
#ORG DATA FORMAT
{
      "name": "Gilbert Nissan Group",
      "entities": [
        {
          "name": "Gilbert Nissan Group",
          "children": [
            {
              "name": "Gilbert Nissan",
              "mappings": [
                {
                  "type": "SFRCE",
                  "solutionId": "0016000000U7We4AAF"
                },
                {
                  "type": "FUSION",
                  "solutionId": "150420"
                },
                {
                  "type": "DTCOM",
                  "solutionId": "150420"
                },
                {
                  "type": "CAI_CORE_DOMAIN",
                  "solutionId": "RlVTSU9OOkRFQUxFUjoxNTA0MjA"
                }
              ],
              "entityId": "RlVTSU9OOkRFQUxFUjoxNTA0MjA"
            }
          ],
          "entityId": "RlVTSU9OOkdST1VQOjQxMDA4ODQ1Mw"
        }
      ],
      "tenantId": "RlVTSU9OOlRFTkFOVDo0MTAwODg0NTM"
```

The purpose of this exercise was to do a quick proof-of-concept on how Snowflake allows the ingestion, storing and querying of semi-structured data. At Cox Automotive, there are various data science teams working on solving interesting challenges like prediction algorithms and computer vision classifications. Most of this work involves storing and accessing semi-structured file formats like JSON and XML. With Snowflake there is a possibility of simplifying the data pipeline by creating a unified data warehouse to store both structured and semi-structured data. Thanks for reading and feel free to reach out to me for additional details or provide your comments.