Automated JSON Data Handling with Snowflake

Craig Warman
Sr. Sales Engineer
Atlanta, Georgia USA

BACKGROUND

Snowflake Timeline and Statistics

Founded 2012 by industry veterans

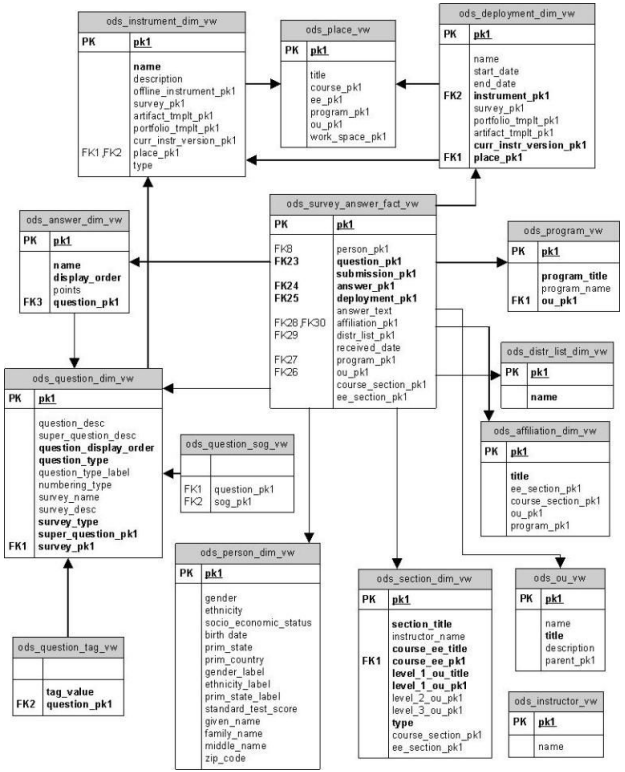
Over $950M in venture funding from leading investors - $4.5B valuation

First customers 2014, general availability 2015

1800+ employees
Over 2500 customers today

Queries processed in Snowflake per day:	**~ 300 Million**
Largest single table:	**> 68 Trillion Rows**
Largest number of tables single DB:	**> 200,000**
Single customer most data:	**> 55 PB**
Single customer most users:	**> 10,000**

The difference between Structured and Semi-Structured Data (and why it's so difficult to manage with traditional databases)




```
"web-app": {
 "servlet": [
   {
     "servlet-name": "cofaxCDS",
     "servlet-class": "org.cofax.cds.CDSServlet",
     "init-param": {
       "configGlossary:installationAt": "Philadelphia, PA",
       "configGlossary:adminEmail": "ksm@pobox.com",
       "configGlossary:poweredBy": "Cofax",
       "configGlossary:poweredByIcon": "/images/cofax.gif",
       "configGlossary:staticPath": "/content/static",
       "templateProcessorClass": "org.cofax.WysiwygTemplate",
       "templateLoaderClass": "org.cofax.FilesTemplateLoader",
       "templatePath": "templates",
       "templateOverridePath": "",
       "defaultListTemplate": "listTemplate.htm",
       "defaultFileTemplate": "articleTemplate.htm",
       "cachePagesTrack": 200,
       "cachePagesStore": 100,
       "cachePagesRefresh": 10,
       "cachePagesDirtyRead": 10,
       "searchEngineListTemplate": "forSearchEnginesList.htm",
       "maxUrlLength": 500}},
   {
     "servlet-name": "cofaxEmail",
     "servlet-class": "org.cofax.cds.EmailServlet",
     "init-param": {
     "mailHost": "mail1",
     "mailHostOverride": "mail2"}},
   {
     "servlet-name": "cofaxAdmin",
     "servlet-class": "org.cofax.cds.AdminServlet"},

   {
     "servlet-name": "fileServlet",
     "servlet-class": "org.cofax.cds.FileServlet"},
   {
     "servlet-name": "cofaxTools",
     "servlet-class": "org.cofax.cms.CofaxToolsServlet",
     "init-param": {
       "templatePath": "toolstemplates/",
       "log": 1,
       "logLocation": "/usr/local/tomcat/logs/CofaxTools.log",
       "logMaxSize": "",
       "dataLog": 1,
       "dataLogLocation": "/usr/local/tomcat/logs/dataLog.log",
       "dataLogMaxSize": "",
       "removePageCache": "/content/admin/remove?cache=pages&id=",
       "removeTemplateCache": "/content/admin/remove?cache=templates&id=",
       "fileTransferFolder": "/usr/local/tomcat/webapps/content/fileTransferFolder",
       "lookInContext": 1,
       "adminGroupID": 4,
       "betaServer": true}}],
 "servlet-mapping": {
   "cofaxCDS": "/",
   "cofaxEmail": "/cofaxutil/aemail/*",
   "cofaxAdmin": "/admin/*",
   "fileServlet": "/static/*",
   "cofaxTools": "/tools/*"},

 "taglib": {
   "taglib-uri": "cofax.tld",
   "taglib-location": "/WEB-INF/tlds/cofax.tld"}}}
```

The difference between Structured and Semi-Structured Data (and why it's so difficult to manage with traditional databases)

- Semi-Structured data that may be of (nearly) any type
- It can be variable in length (arrays)
- Its structure can rapidly and unpredictably change
- It's usually self-describing

Some Examples:

- XML
- AVRO
- JSON
- Parquet
- ORC

Example: JSON

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

Example: JSON

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
CREATE OR REPLACE TABLE colors
   (
       json_data  VARIANT
   );
```

DEMO #1:
LOADING JSON DATA

© 2019 Snowflake Computing Inc.

Example: JSON

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
CREATE OR REPLACE TABLE colors
   (
       json_data  VARIANT
   );
```

Example: JSON

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
CREATE OR REPLACE TABLE colors
   (
       json_data  VARIANT
   );
```

```
SELECT
    json_data:ID::INTEGER as ID,
    json_data:color::STRING as color,
    json_data:category::STRING as category,
    json_data:type::STRING as type,
    json_data:code.rgb::STRING as code_rgb,
    json_data:code.hex::STRING as code_hex
FROM
    colors;
```

DEMO #2:
QUERYING JSON DATA

© 2019 Snowflake Computing Inc.

Example: JSON

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
CREATE OR REPLACE TABLE colors
   (
       json_data  VARIANT
   );
```

```
SELECT
    json_data:ID::INTEGER as ID,
    json_data:color::STRING as color,
    json_data:category::STRING as category,
    json_data:type::STRING as type,
    json_data:code.rgb::STRING as code_rgb,
    json_data:code.hex::STRING as code_hex
FROM
    colors;
```

Example: JSON

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
CREATE OR REPLACE TABLE colors
   (
       json_data  VARIANT
   );
```

```
CREATE OR REPLACE VIEW
    colors_vw
AS SELECT
    json_data:ID::INTEGER as ID,
    json_data:color::STRING as color,
    json_data:category::STRING as category,
    json_data:type::STRING as type,
    json_data:code.rgb::STRING as code_rgb,
    json_data:code.hex::STRING as code_hex
FROM
    colors;
```

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT
   *
FROM
  colors;
```

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT
   f.seq, f.key, f.path, f.value, typeof(f.value)
FROM
  colors,
  LATERAL FLATTEN(json_data, RECURSIVE=>true) f
```

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT
   f.seq, f.key, f.path, f.value, typeof(f.value)
FROM
  colors,
  LATERAL FLATTEN(json_data, RECURSIVE=>true) f
WHERE
  TYPEOF(f.value) != 'OBJECT'
```

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT
   f.seq, f.key, f.path, f.value, typeof(f.value)
FROM
  colors,
  LATERAL FLATTEN(json_data, RECURSIVE=>true) f
WHERE
  TYPEOF(f.value) != 'OBJECT'
```

SEQ	KEY	PATH	VALUE	TYPEOF(F.VALUE)
1	ID	ID	1	INTEGER
1	category	category	"hue"	VARCHAR
1	hex	code.hex	"#000"	VARCHAR
1	rgb	code.rgb	"255,255,255"	VARCHAR
1	color	color	"black"	VARCHAR
1	type	type	"primary"	VARCHAR
2	ID	ID	2	INTEGER
2	category	category	"value"	VARCHAR
2	hex	code.hex	"#FFF"	VARCHAR
2	rgb	code.rgb	"0,0,0"	VARCHAR
2	color	color	"white"	VARCHAR
3	ID	ID	3	INTEGER

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT
   f.seq, f.key, f.path, f.value, typeof(f.value)
FROM
  colors,
  LATERAL FLATTEN(json_data, RECURSIVE=>true) f
WHERE
  TYPEOF(f.value) != 'OBJECT'
```

SEQ	KEY	PATH	VALUE	TYPEOF(F.VALUE)
1	ID	ID	1	INTEGER
1	category	category	"hue"	VARCHAR
1	hex	code.hex	"#000"	VARCHAR
1	rgb	code.rgb	"255,255,255"	VARCHAR
1	color	color	"black"	VARCHAR
1	type	type	"primary"	VARCHAR
2	ID	ID	2	INTEGER
2	category	category	"value"	VARCHAR
2	hex	code.hex	"#FFF"	VARCHAR
2	rgb	code.rgb	"0,0,0"	VARCHAR
2	color	color	"white"	VARCHAR
3	ID	ID	3	INTEGER

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT
   f.seq, f.key, f.path, f.value, typeof(f.value)
FROM
  colors,
  LATERAL FLATTEN(json_data, RECURSIVE=>true) f
WHERE
  TYPEOF(f.value) != 'OBJECT'
```

SEQ	KEY	PATH	VALUE	TYPEOF(F.VALUE)
1	ID	ID	1	INTEGER
1	category	category	"hue"	VARCHAR
1	hex	code.hex	"#000"	VARCHAR
1	rgb	code.rgb	"255,255,255"	VARCHAR
1	color	color	"black"	VARCHAR
1	type	type	"primary"	VARCHAR
2	ID	ID	2	INTEGER
2	category	category	"value"	VARCHAR
2	hex	code.hex	"#FFF"	VARCHAR
2	rgb	code.rgb	"0,0,0"	VARCHAR
2	color	color	"white"	VARCHAR
3	ID	ID	3	INTEGER

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT
   f.seq, f.key, f.path, f.value, typeof(f.value)
FROM
  colors,
  LATERAL FLATTEN(json_data, RECURSIVE=>true) f
WHERE
  TYPEOF(f.value) != 'OBJECT'
```

SEQ	KEY	PATH	VALUE	TYPEOF(F.VALUE)
1	ID	ID	1	INTEGER
1	category	category	"hue"	VARCHAR
1	hex	code.hex	"#000"	VARCHAR
1	rgb	code.rgb	"255,255,255"	VARCHAR
1	color	color	"black"	VARCHAR
1	type	type	"primary"	VARCHAR
2	ID	ID	2	INTEGER
2	category	category	"value"	VARCHAR
2	hex	code.hex	"#FFF"	VARCHAR
2	rgb	code.rgb	"0,0,0"	VARCHAR
2	color	color	"white"	VARCHAR
3	ID	ID	3	INTEGER

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT
   f.seq, f.key, f.path, f.value, typeof(f.value)
FROM
  colors,
  LATERAL FLATTEN(json_data, RECURSIVE=>true) f
WHERE
  TYPEOF(f.value) != 'OBJECT'
```

SEQ	KEY	PATH	VALUE	TYPEOF(F.VALUE)
1	ID	ID	1	INTEGER
1	category	category	"hue"	VARCHAR
1	hex	code.hex	"#000"	VARCHAR
1	rgb	code.rgb	"255,255,255"	VARCHAR
1	color	color	"black"	VARCHAR
1	type	type	"primary"	VARCHAR
2	ID	ID	2	INTEGER
2	category	category	"value"	VARCHAR
2	hex	code.hex	"#FFF"	VARCHAR
2	rgb	code.rgb	"0,0,0"	VARCHAR
2	color	color	"white"	VARCHAR
3	ID	ID	3	INTEGER

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT DISTINCT
   f.seq, f.key, f.path, f.value, typeof(f.value)
FROM
  colors,
  LATERAL FLATTEN(json_data, RECURSIVE=>true) f
WHERE
  TYPEOF(f.value) != 'OBJECT'
```

Tapping Into The Snowflake Metadata

```
{
   ID: 1,
   "color": "black",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,255,255",
     "hex": "#FFF"
   }
}
{
   ID: 2,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
{
   ID: 3,
   "color": "red",
   "category": "hue",
   "type": "primary",
   "code": {
     "rgb": "255,0,0",
     "hex": "#FF0"
   }
}
```

```
SELECT
   f.seq, f.key, f.path, f.value, typeof(f.value)
FROM
  colors,
  LATERAL FLATTEN(json_data, RECURSIVE=>true) f
WHERE
  TYPEOF(f.value) != 'OBJECT'
```

PATH	TYPEOF(F.VALUE)
ID	INTEGER
color	VARCHAR
category	VARCHAR
type	VARCHAR
code.rgb	VARCHAR
code.hex	VARCHAR

Tapping Into The Snowflake Metadata

```
CREATE OR REPLACE VIEW
    colors_vw
AS SELECT
    json_data:ID::INTEGER as ID,
    json_data:color::STRING as color,
    json_data:category::STRING as category,
    json_data:type::STRING as type,
    json_data:code.rgb::STRING as code_rgb,
    json_data:code.hex::STRING as code_hex
FROM
    colors;
```

PATH	TYPEOF(F.VALUE)
ID	INTEGER
color	VARCHAR
category	VARCHAR
type	VARCHAR
code.rgb	VARCHAR
code.hex	VARCHAR

Tapping Into The Snowflake Metadata

```
CREATE OR REPLACE VIEW
    colors_vw
AS SELECT
    json_data:ID::INTEGER as ID,
    json_data:color::STRING as color,
    json_data:category::STRING as category,
    json_data:type::STRING as type,
    json_data:code.rgb::STRING as code_rgb,
    json_data:code.hex::STRING as code_hex
FROM
    colors;
```

PATH	TYPEOF(F.VALUE)
ID	INTEGER
color	VARCHAR
category	VARCHAR
type	VARCHAR
code.rgb	VARCHAR
code.hex	VARCHAR

Tapping Into The Snowflake Metadata

```
CREATE OR REPLACE VIEW
    colors_vw
AS SELECT
    json_data:ID::INTEGER as ID,
    json_data:color::STRING as color,
    json_data:category::STRING as category,
    json_data:type::STRING as type,
    json_data:code.rgb::STRING as code_rgb,
    json_data:code.hex::STRING as code_hex
FROM
    colors;
```

PATH	TYPEOF(F.VALUE)
ID	INTEGER
color	VARCHAR
category	VARCHAR
type	VARCHAR
code.rgb	VARCHAR
code.hex	VARCHAR

DEMO #3: SNOWFLAKE METADATA

© 2019 Snowflake Computing Inc.

Logical Flow For Constructing a CREATE VIEW Statement

1. Build a query that returns the elements and their datatypes
2. Run the query
3. Loop through the returned elements
4. Build the view column list
5. Construct the view DDL

```
CREATE VIEW ... AS SELECT ...
```

6. Run the DDL to create the view

DEMO #4:
CREATE_VIEW_OVER_JSON STORED PROCEDURE

© 2019 Snowflake Computing Inc.

Automated JSON Data Handling with Snowflake

Craig Warman
Sr. Sales Engineer
Atlanta, Georgia USA

Automating Snowflake's Semi-Structured JSON Data Handling

Craig Warman
Sr. Sales Engineer
Atlanta, Georgia USA

HOW IT WORKS: CREATE_VIEW_OVER_JSON STORED PROCEDURE

© 2019 Snowflake Computing Inc.

Stored Procedure CREATE_VIEW_OVER_JSON

```
create or replace procedure create_view_over_json
   (TABLE_NAME varchar, COL_NAME varchar, VIEW_NAME varchar)
returns varchar
language javascript
as
$$

do something…

$$;
```

Stored Procedure CREATE_VIEW_OVER_JSON

1. Build a query that returns the elements and their datatypes:

```
var element_query = "SELECT DISTINCT \n" +
                    path_name + " AS path_name, \n" +
                    attribute_type + " AS attribute_type, \n" +
                    alias_name + " AS alias_name \n" +
                    "FROM \n" +
                    TABLE_NAME + ", \n" +
                    "LATERAL FLATTEN(" + COL_NAME + ", RECURSIVE=>true) f \n" +
                    "WHERE TYPEOF(f.value) != 'OBJECT' \n";
```

Stored Procedure CREATE_VIEW_OVER_JSON

1. Build a query that returns the elements and their datatypes:

```
var path_name = "regexp_replace(f.path,'(\\\\w+)','\"\\\\1\"')";


var element_query = "SELECT DISTINCT \n" +
                    path_name + " AS path_name, \n" +
                    attribute_type + " AS attribute_type, \n" +
                    alias_name + " AS alias_name \n" +
                    "FROM \n" +
                    TABLE_NAME + ", \n" +
                    "LATERAL FLATTEN(" + COL_NAME + ", RECURSIVE=>true) f \n" +
                    "WHERE TYPEOF(f.value) != 'OBJECT' \n";
```

This generates paths with levels enclosed by double quotes (ex: "path"."to"."element")

Stored Procedure CREATE_VIEW_OVER_JSON

1. Build a query that returns the elements and their datatypes:

```
var attribute_type =
 "DECODE (substr(typeof(f.value),1,1),'B','BOOLEAN','I','FLOAT','D','FLOAT','STRING')";

var element_query = "SELECT DISTINCT \n" +
                    path_name + " AS path_name, \n" +
                    attribute_type + " AS attribute_type, \n" +
                    alias_name + " AS alias_name \n" +
                    "FROM \n" +
                    TABLE_NAME + ", \n" +
                    "LATERAL FLATTEN(" + COL_NAME + ", RECURSIVE=>true) f \n" +
                    "WHERE TYPEOF(f.value) != 'OBJECT' \n";
```

This generates column datatypes of BOOLEAN, FLOAT, and STRING only

Stored Procedure CREATE_VIEW_OVER_JSON

1. Build a query that returns the elements and their datatypes:

```
var alias_name = "REGEXP_REPLACE(
  REGEXP_REPLACE(f.path, '\\\\[(.+)\\\\]'),'[^a-zA-Z0-9]','_')" ;

var element_query = "SELECT DISTINCT \n" +
                    path_name + " AS path_name, \n" +
                    attribute_type + " AS attribute_type, \n" +
                    alias_name + " AS alias_name \n" +
                    "FROM \n" +
                    TABLE_NAME + ", \n" +
                    "LATERAL FLATTEN(" + COL_NAME + ", RECURSIVE=>true) f \n" +
                    "WHERE TYPEOF(f.value) != 'OBJECT' \n";
```

This generates column aliases based on the path

Stored Procedure CREATE_VIEW_OVER_JSON

2. Run the query:

```
var element_stmt = snowflake.createStatement({sqlText:element_query});
```

Stored Procedure CREATE_VIEW_OVER_JSON

2. Run the query:

```
var element_stmt = snowflake.createStatement({sqlText:element_query});
var element_res = element_stmt.execute();
```

Stored Procedure CREATE_VIEW_OVER_JSON

2. Run the query:

```
var element_stmt = snowflake.createStatement({sqlText:element_query});
var element_res = element_stmt.execute();
```

3. Loop through the returned elements:

```
while (element_res.next()) {



    }
```

Stored Procedure CREATE_VIEW_OVER_JSON

2. Run the query:

```
var element_stmt = snowflake.createStatement({sqlText:element_query});
var element_res = element_stmt.execute();
```

3. Loop through the returned elements:

```
while (element_res.next()) {



    }
```

Goal is to generate column expressions that look like:

`col_name:"name"."first"::STRING as "name_first"`

Stored Procedure CREATE_VIEW_OVER_JSON

2. Run the query:

```
var element_stmt = snowflake.createStatement({sqlText:element_query});
var element_res = element_stmt.execute();
```

3. Loop through the returned elements:

```
while (element_res.next()) {
      col_list += COL_NAME + ":" + element_res.getColumnValue(1);


    }
```

Path name

Goal is to generate column expressions that look like:

```
col_name:"name"."first"::STRING as "name_first"
```

Stored Procedure CREATE_VIEW_OVER_JSON

2. Run the query:

```
var element_stmt = snowflake.createStatement({sqlText:element_query});
var element_res = element_stmt.execute();
```

3. Loop through the returned elements:

```
while (element_res.next()) {
      col_list += COL_NAME + ":" + element_res.getColumnValue(1);
      col_list += "::" + element_res.getColumnValue(2);

     }
```

Path name
Datatype

Goal is to generate column expressions that look like:

```
col_name:"name"."first"::STRING as "name_first"
```

Stored Procedure CREATE_VIEW_OVER_JSON

2. Run the query:

```
var element_stmt = snowflake.createStatement({sqlText:element_query});
var element_res = element_stmt.execute();
```

3. Loop through the returned elements:

```
while (element_res.next()) {
      col_list += COL_NAME + ":" + element_res.getColumnValue(1);
      col_list += "::" + element_res.getColumnValue(2);
      col_list += " as " + element_res.getColumnValue(3);
     }
```

Path name
Datatype
Alias

Goal is to generate column expressions that look like:

```
col_name:"name"."first"::STRING as "name_first"
```

Stored Procedure CREATE_VIEW_OVER_JSON

2. Run the query:

```
var element_stmt = snowflake.createStatement({sqlText:element_query});
var element_res = element_stmt.execute();
```

3. Loop through the returned elements:

```
while (element_res.next()) {
    col_list += COL_NAME + ":" + element_res.getColumnValue(1);
    col_list += " :" + element_res.getColumnValue(2);
    col_list += " as " + element_res.getColumnValue(3);
}
```

Path name
Datatype
Alias

4. Build the view column list

Stored Procedure CREATE_VIEW_OVER_JSON

5. Construct the view DDL:

```
var view_ddl = "CREATE OR REPLACE VIEW " + VIEW_NAME + " AS \n" +
               "SELECT \n" + col_list + "\n" +
               "FROM " + TABLE_NAME;
```

Stored Procedure CREATE_VIEW_OVER_JSON

5. Construct the view DDL:

```
var view_ddl = "CREATE OR REPLACE VIEW " + VIEW_NAME + " AS \n" +
               "SELECT \n" + col_list + "\n" +
               "FROM " + TABLE_NAME;
```

Stored Procedure CREATE_VIEW_OVER_JSON

5. Construct the view DDL:

```
var view_ddl = "CREATE OR REPLACE VIEW " + VIEW_NAME + " AS \n" +
               "SELECT \n" + col_list + "\n" +
               "FROM " + TABLE_NAME;
```

6. Run the DDL to create the view:

```
var view_stmt = snowflake.createStatement({sqlText:view_ddl});
```

Stored Procedure CREATE_VIEW_OVER_JSON

5. Construct the view DDL:

```
var view_ddl = "CREATE OR REPLACE VIEW " + VIEW_NAME + " AS \n" +
               "SELECT \n" + col_list + "\n" +
               "FROM " + TABLE_NAME;
```

6. Run the DDL to create the view:

```
var view_stmt = snowflake.createStatement({sqlText:view_ddl});
var view_res = view_stmt.execute();
```

Stored Procedure CREATE_VIEW_OVER_JSON

5. Construct the view DDL:

```
var view_ddl = "CREATE OR REPLACE VIEW " + VIEW_NAME + " AS \n" +
               "SELECT \n" + col_list + "\n" +
               "FROM " + TABLE_NAME;
```

6. Run the DDL to create the view:

```
var view_stmt = snowflake.createStatement({sqlText:view_ddl});
var view_res = view_stmt.execute();
return view_res.next();
```

What's Next?

- Column Case - Should it match JSON, or use all caps?

What's Next?

- Column Case - Should it match JSON, or use all caps?
  - If matched, then queries columns enclosed by "":

```
SELECT
   "ID",
   "color"
   "category",
   "code_hex",
   "code_rgb"
FROM
   colors_vw;
```

What's Next?

- Column Case - Should it match JSON, or use all caps?
  - If matched, then queries columns enclosed by "":

```
SELECT
   "ID",
   "color"
   "category",
   "code_hex",
   "code_rgb"
FROM
   colors_vw;
```

  - However, some JSON might contain reserved words, such as *type* or *number* - Those wouldn't work without being enclosed by double quotes.

What's Next?

- Column Case - Should it match JSON, or use all caps?
- Column Types - Is it better to just cast all as STRING?

What's Next?

- Column Case - Should it match JSON, or use all caps?
- Column Types - Is it better to just cast all as STRING?
  - Sometimes there cases where multiple JSON documents have attributes with the same name but actually contain data with different datatypes.
  - For example, one document might have an attribute that contains the value "10" while another document has the same attribute with a value of "ten".
  - This may lead to problems.

What's Next?

- Column Case - Should it match JSON, or use all caps?
- Column Types - Is it better to just cast all as STRING?
- Arrays - These should be "exploded" into columns.

What's Next?

- Column Case - Should it match JSON, or use all caps?
- Column Types - Is it better to just cast all as STRING?
- Arrays - These should be "exploded" into columns.

```
{
   ID: 1,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
}
```

What's Next?

- Column Case - Should it match JSON, or use all caps?
- Column Types - Is it better to just cast all as STRING?
- Arrays - These should be "exploded" into columns.

```
{
   ID: 1,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": "0,0,0",
     "hex": "#FFF"
   }
}
```

```
{
   ID: 1,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": [0,0,0],
     "hex": "#FFF"
   }
}
```

What's Next?

- Column Case - Should it match JSON, or use all caps?
- Column Types - Is it better to just cast all as STRING?
- Arrays - These should be "exploded" into columns.
  - Also: Object Array handling

What's Next?

- Column Case - Should it match JSON, or use all caps?
- Column Types - Is it better to just cast all as STRING?
- Arrays - These should be "exploded" into columns.
  - Also: Object Array handling

```
{
   ID: 1,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": [0,0,0],
     "hex": "#FFF"
   }
}
```

What's Next?

- Column Case - Should it match JSON, or use all caps?
- Column Types - Is it better to just cast all as STRING?
- Arrays - These should be "exploded" into columns.
  - Also: Object Array handling

```
{
   ID: 1,
   "color": "white",
   "category": "value",
   "code": {
     "rgb": [0,0,0],
     "hex": "#FFF"
   }
}
```

```
{
   ID: 1,
   name:  {first: "John", last: "Doe"},
   phone: [
           { type: "home", number:"678-555-5678" },
           { type: "cell", number:"770-555-5678" },
           { type: "work", number:"404-555-5678" }
          ] ,
}
```

Automating Snowflake's Semi-Structured JSON Data Handling

Craig Warman
Sr. Sales Engineer
Atlanta, Georgia USA