Using StreamSets to Write Postgres JSON Columns to Snowflake Variants

April 9, 2025 mark.brooks@ibm.com

This document describes one way to automatically write Postgres JSON Columns to Snowflake variant columns.

A solution like this is currently needed for the following reason:

- StreamSets' JDBC Query Origin does not currently support the Postgres JSON datatype (see the docs here).

- In order for StreamSets' JDBC Query Origin to capture JSON columns without errors, one needs to set the Advanced property `On Unknown Type` to the value `Convert to String` which will allow the JDON data to be read, but StreamSets will see the data as Strings, not JSON.

- However, if StreamSets attempts to insert String data into Snowflake Variant columns, Snowflake will throw errors, as shown below.

Consider a source Postgres Table with a schema like this:

```
create table t1 (
      col1 int,
      col2 varchar(255),
      col3 json,
      col4 json,
      col5 int,
      col6 json);
```

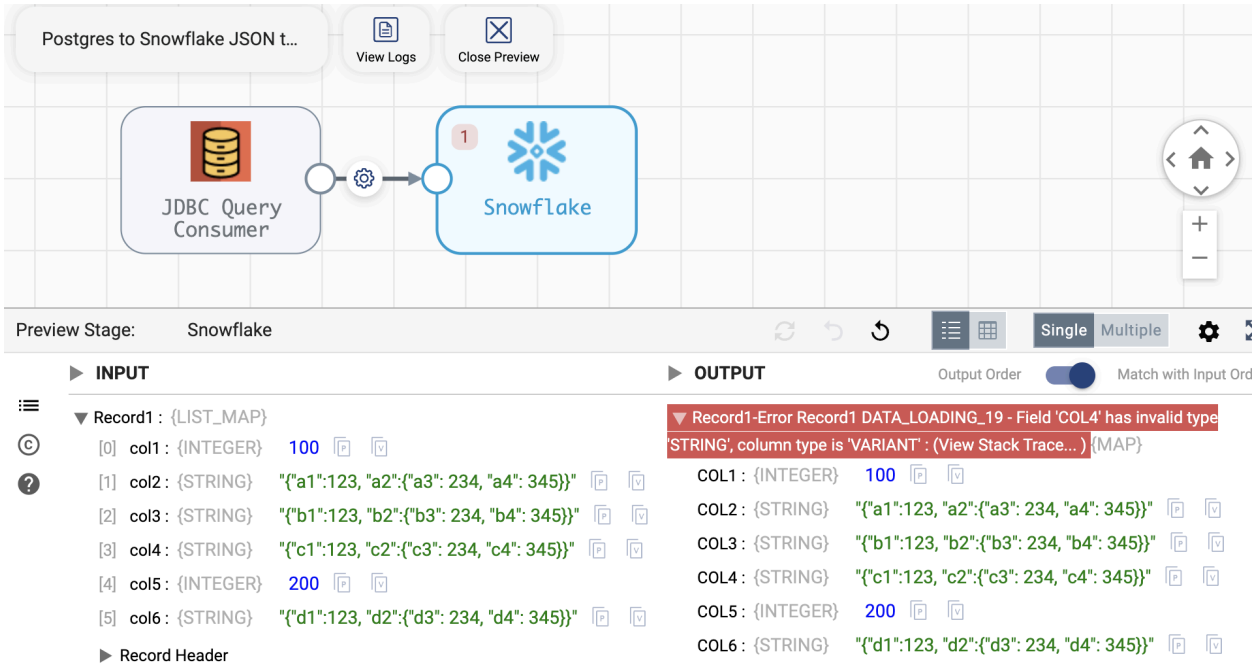
And consider a target Snowflake Table with a schema like this:

```
create TABLE JSON_TO_VARIANT (
      col1 NUMBER(38,0),
      col2 STRING,
      col3 STRING,
      col4 VARIANT,
      col5 INT,
      col6 VARIANT);
```

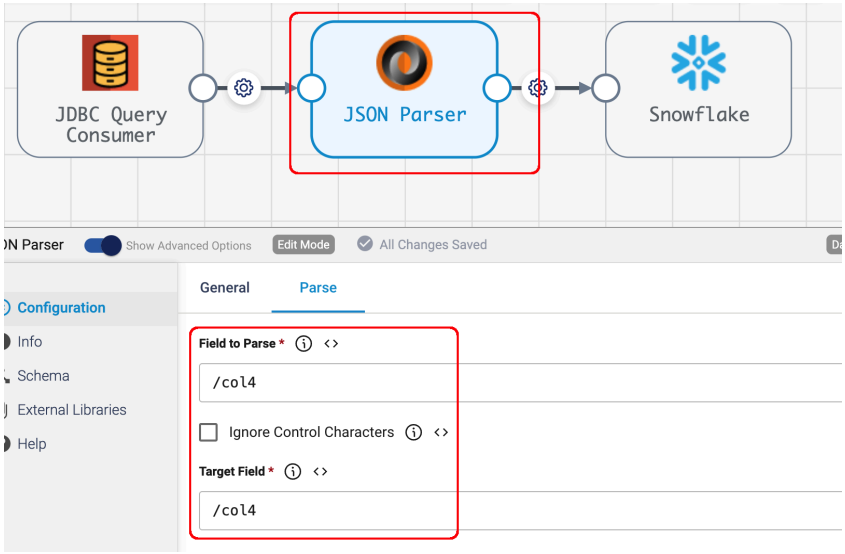
Note that `col3` is JSON in Postgres but is a String in Snowflake, while `col4` and `col6` are both JSON in Postgres and variants in Snowflake.

The user intends the JSON data from the Postgres `col3` to be left as a String to match the pre-existing String datatype of the Snowflake `col3`, but wants Postgres JSON columns `col4` and `col6` to be converted to map datatypes to be written to the respective Snowflake variant columns of the same names.

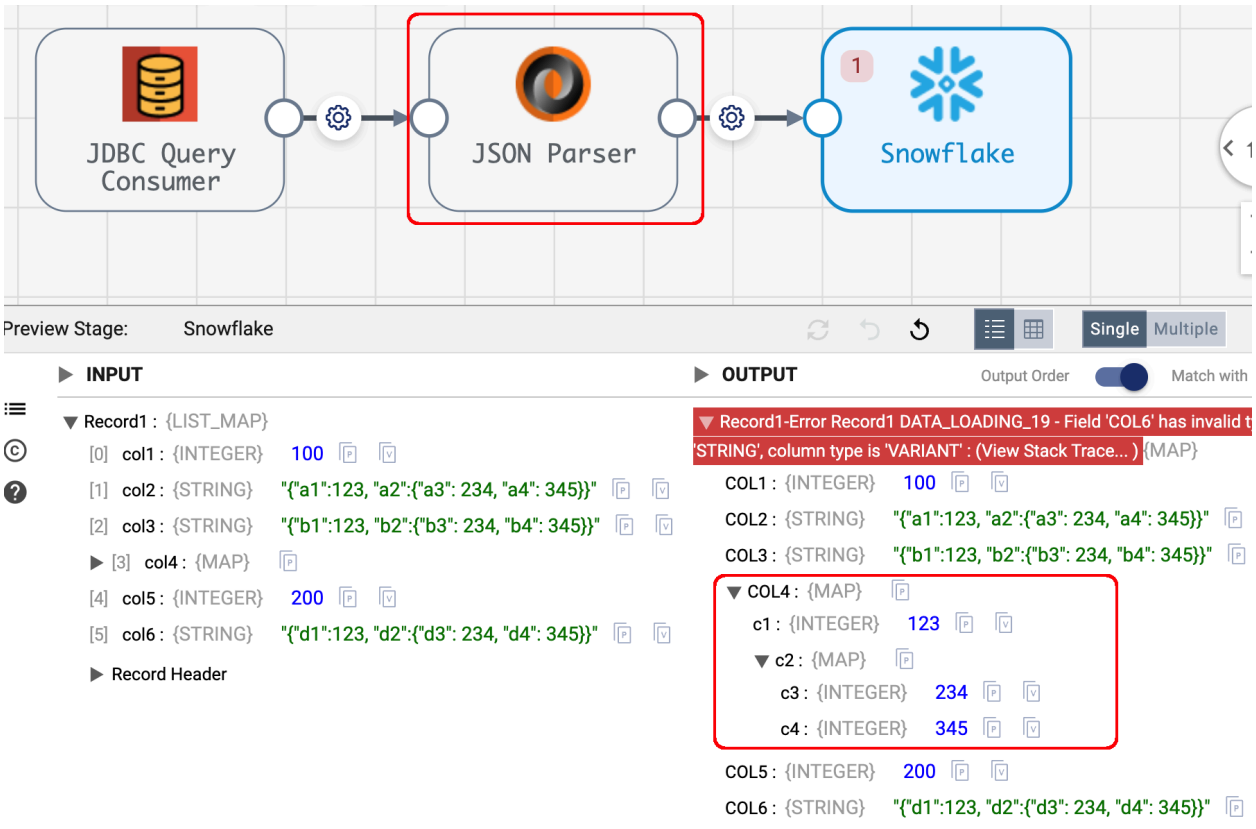
If no remediation is in place, the Postgres JSON columns will be converted to Strings, and the user will get the error `DATA_LOADING_19 - Field 'COL4' has invalid type 'STRING', column type is 'VARIANT'`:



It is easy to statically map individual JSON fields at design time to the appropriate datatype by adding a JSON Parser Processor like this for the field `col4`:



We can see this resolves the issue for `col4` (which is now a complex map) but we would need to add another JSON Parser Processor for `col6` as we have not yet remediated that column:

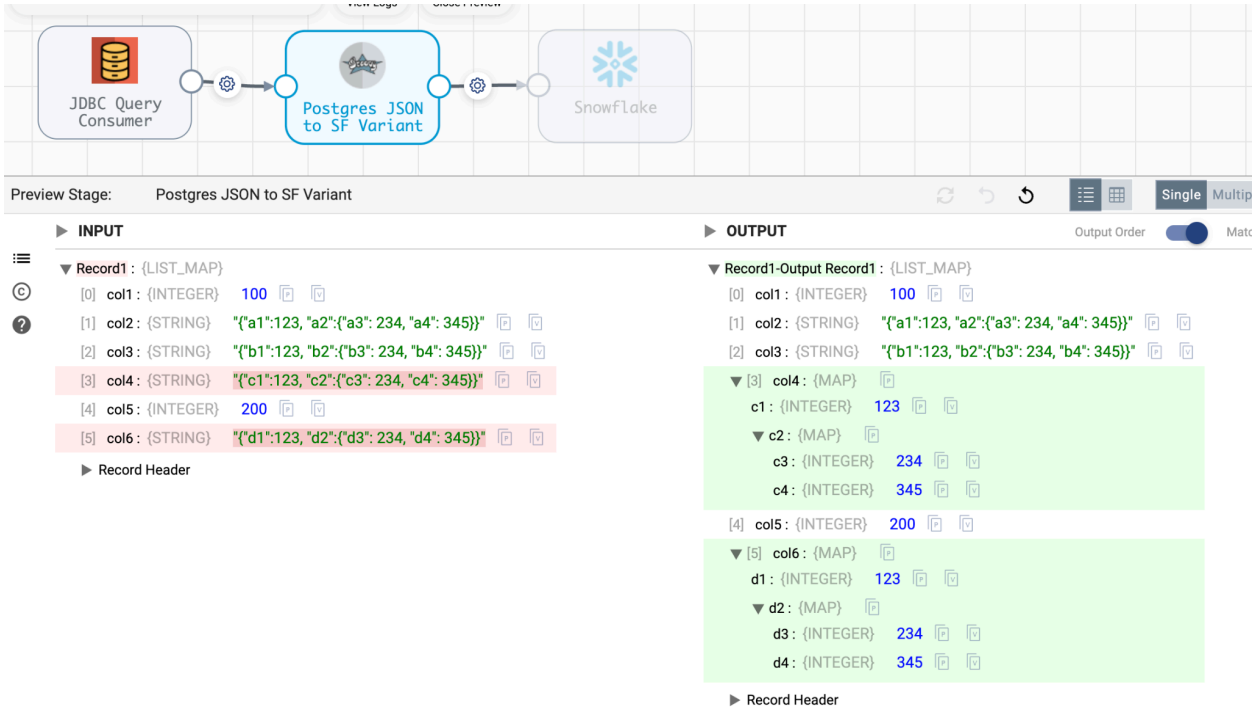


Moreover, the customer requires a generalized solution as they have hundreds of pipelines with all different sets of JSON columns, and one can't query the source schemas in advance because their pipeline origins execute arbitrary JDBC join queries with complex logic and type casting.

In the longer term, it would be great if StreamSets added JSON type support for Postgres (tracked as the internal IBM enhancement request https://ideas.ibm.com/ideas/SSETS-I-153), and if StreamSets could automatically convert JSON types (or map types) to either Snowflake variants or Strings depending on the target table's datatypes.

In the meantime, a workaround is possible using a single Groovy processor, as described below.

Here is an example pipeline that uses a Groovy stage to auto-detect that `col4` and `col6` need to be converted to map datatypes for Snowflake variants, while leaving `col3` as a String:

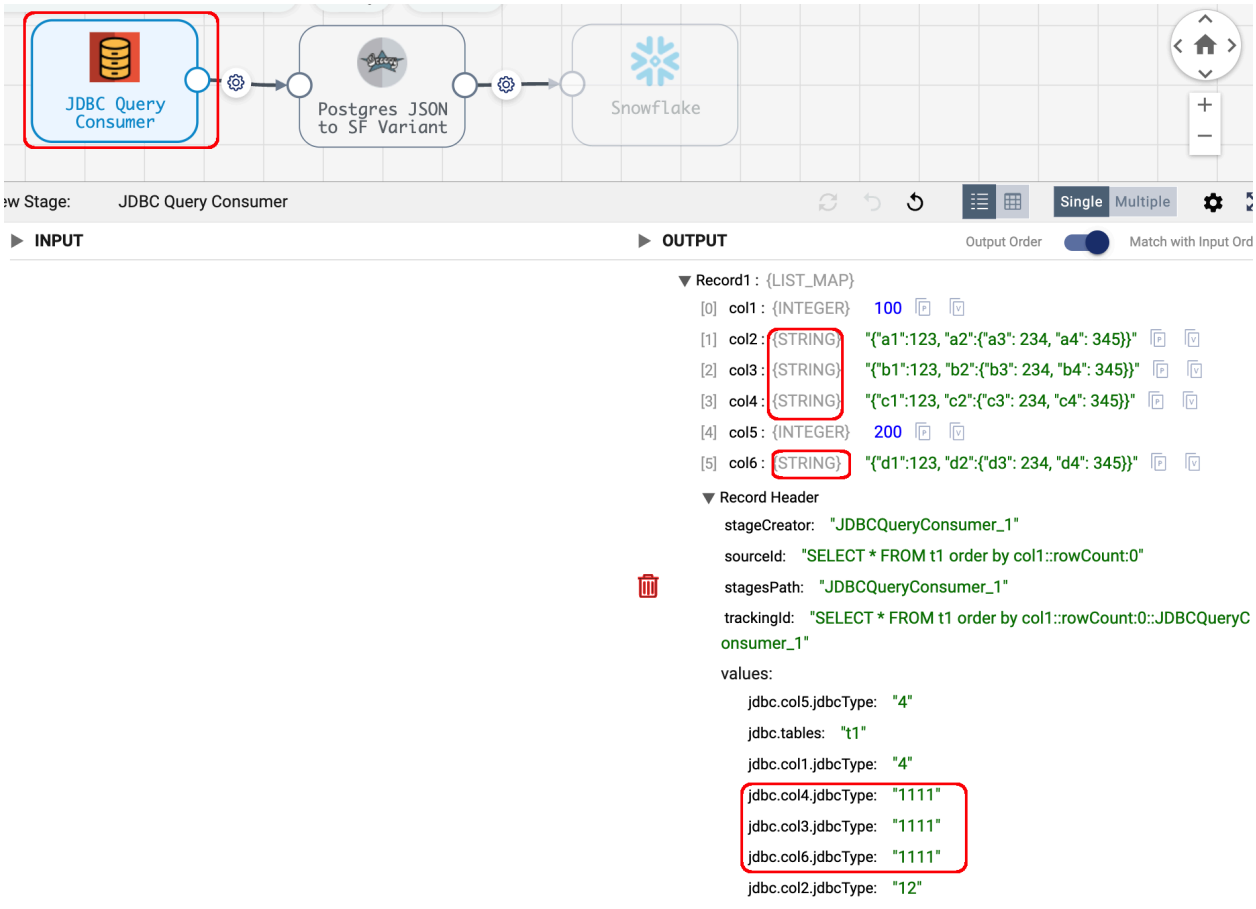


The pipeline and separate copies of the Groovy scripts that do the work within the pipeline are posted in the project here

Here are the key steps in the Groovy Processor's script:

- The init script, which runs only once, at pipeline startup, retrieves the Snowflake target table's column names and data types.

- The main script pays special attention to the pipeline's first record, and processes its JDBC metadata. Note that although `col2`, `col3`, `col4` and `col6` are seen as Strings, the JDBC metadata `jdbcType` lets us know that `col3`, `col4` and `col6` are actually JSON (the jdbcType value is "1111"):



When processing the first record, the Groovy script stores in global state the list of JSON source columns, so this analysis is only performed once.

- For all records, the Groovy script matches the names of JSON source columns to the names of Snowflake target columns, and if the matching target column is a variant, it converts the value to a JSON which StreamSets will see as a map . Here is the key conversion step in the Groovy code's main script:

```
// Convert JSON typed fields (which the pipeline sees as Stings)
// to actual JSON if the Snowflake column of the same name is a variant
for (column_name in record.value.keySet()){
    if (sdc.state['json_fields'].contains(column_name)){
        if (snowflake_schema[column_name]  == 'variant'){
            record.value[column_name] =
                jsonSlurper.parseText(record.value[column_name])
            }
        }
    }
}
```

Important note: the script performs its Postgres to Snowflake column mapping by lower-casing all column names, so if by chance you have case sensitive column names with multiple column names with the same characters in the same table with different cases (and I sure hope you don't!), the mapping may not be correct.

Set your Snowflake connection parameters in the Groovy stage's Advanced properties. Note that the Snowflake user, password,and URL are loaded from runtime resources, and the Snowflake table is set as a pipeline parameter. Also set the Snowflake role, warehouse, db, and schema: