How to load dynamic column order using Talend Open Studio.

This tutorial was inspired by a problem we faced when collaborating with a client. The client environment is a Talend Job running in an Azure virtual machine. The nature of the job was to read the data from AWS S3 dumped by partners, load the data into Amazon RDS for MySQL, a managed MySQL instance. Once we had files from multiple partners, we found that, though all required data is present in source files, they are not in the same order.

In Talend Open Studio (TOS), schemas are defined to describe the purpose and data types for a source or target. TOS is strict about associating the column position in flat files or spreadsheets with a particular field definition. If the first field is appointed "Id" column, the input is expected to have "Id" in its first field despite what the column header says.

Our first solution was to read the header information and store it in the context variable. Then initiate a tMap transformation of each input row based on the header information. But processing millions of records this way doesn’t scale well.

We need an alternative strategy that can scale well to handle millions of records. t. The below solution uses a temporary table to reduce the overhead on the Talend server at the cost of storage overhead.

# Algorithm:

The input file header is processed separately. The input data is stored in a temporary table and then, based on header information, dynamic SQL is generated to load from temporary table and store into the final table.

The three steps are:

1. Record the header information in Context Variables (In this example context variable is created to map the output columns, but can be easily replaced with a HashMap context variable)
2. Load the data into a temporary table.
3. Write dynamic query using column alias to read data from temporary table in expected output order and write to final table.

Note: Step 1 & 2 can be run in parallel. Step 3 is dependent on both step 1 & 2

Below is an example of two input files.

Input File 1Table

Description automatically generated

Input File 2

Table

Description automatically generated

Based on this example, the output table is expected to have 5 columns.

One thing to note here is the column type. It is best practice to dump that data into a string type when reading data from unknown column types. You can convert from that type once you are sure of the actual data type held by that column.

The following screenshot shows all the job components.

Timeline

Description automatically generated

# Define Context Variables

Define one context variable for each output column. In our case we have 5 output columns so have defined 5 context variables. These context variables will be updated based on the input file header data.

A screenshot of a computer

Description automatically generated with medium confidence

Note: A single context variable of type HashMap can be used instead of 5 context variables.

# Read Header (tFileInputDelimited)

Using tFileInputDelimited, read the header record (Set Header = 0 and Limit=1 to read only header). The tFileInputDelimited is defined with 5 generic columns. All are defined to be String data types as column data type is unknown at this point. Use generic column names in the temporary table each source file has different order of data.

Table

Description automatically generated

# tInputFileDelimited Schema Definition

The iterate output is passed on to tJavaRow which updates the context variables with corresponding column headers.

For example, if the first header is “id”, then context.col\_id value will be “col1”. “col1” is the first column name defined in the input schema.

Text

Description automatically generated with low confidence

tJavaRow Code

# Read Data (tFileInputDelimited)

Using tFileInputDelimited, read the data and store it in a temporary table (Set Header = 1 to read data). **The Schema of this tFileInputDelimited should match with schema of tFileInputDelimited that reads header data.**

Now the temporary table named “input\_user\_data” will have the data stored in generic columns on the same order received in the input file.

# Read Using Dynamic Query (tDBInput)

Now use tDBInput to read from the temporary table “input\_user\_data” and define Query using the context variable. For example, the below query loads the final output table with column order as:

* ID.
* First name.
* Last name.
* Email.
* Flags.

Also, this is the step where the column data types are converted to actual data types.

Graphical user interface, text

Description automatically generated

As previously noted, this method has the advantage of ending the need to look-up each input row. It reduces the computational requirement of the ETL server at the cost of storing intermediate data in a temporary table.

With this change, we were able to complete this stage of ETL process five-times faster and at three-times lower computational cost.

Reach out to Xenovex Technologies at sales@xenovex.com for innovative solution to your needs.