Paper 3306-2015

Extending the Scope of Custom Transformations

Emre G. SARICICEK, The University of North Carolina at Chapel Hill.

ABSTRACT

Building and maintaining a data warehouse can require complex series of jobs. Having an ETL flow that is reliable and well integrated is one big challenge. An ETL process might need some pre- and post-processing operations on the database to be well integrated and reliable. Some might handle this via maintenance windows. Others like us might generate custom transformations to be included in SAS® Data Integration Studio jobs. Custom transformations in SAS Data Integration Studio can be used to speed ETL process flows and reduce the database administrator's intervention after ETL flows are complete. In this paper, we demonstrate the use of custom transformations in SAS Data Integration Studio jobs to handle database-specific tasks for improving process efficiency and reliability in ETL flows.

INTRODUCTION

SAS Data Integration (DI) Studio is a powerful visual tool that can handle data integration projects. We will be focusing on using DI transformations that are used in conjunction with building enterprise data warehouse ETL jobs.

SAS DI Studio has metadata objects that are called transformation and they are there to perform a specific task such as extract data, transform data or loading data.

DI Studio provides a comprehensive set of transformations for performing the ETL processes necessary to build and maintain a data warehouse, however large amount of tasks required on the database for jobs to run reliably.

The purpose of this paper is to introduce a process of developing custom transformations to handle some processes out of ETL job itself but in scheduled job flows to keep it integrated.

We will be focusing on RDBMS (mainly ORACLE) operations via SAS DI studio to handle some maintenance and ETL preparation operations.

Creating a user written transformation helps us to stream line custom code and make code reusable. We can easily integrate custom transformations into data integration workflows.

CREATING CUSTOM DI TRANSFORMATION

You can create a new transformation using transformations tab navigation menu. As seen in Figure 1. From here you can select category for this transformation. Category is for organizing metadata objects there are no differences on transformation generator screens between different categories.

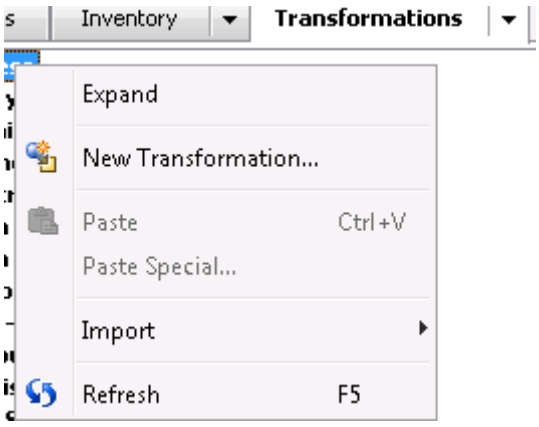

Figure 1. Start a New Transformation

DI Studio gives a good transformation generator wizard for us developers to create our custom transformation. Wizard has a step where we can put SAS code in. I think SAS Enterprise Guide is a great tool to write SAS code and do initial transformation code and just paste here and configure input and outputs of transformation.

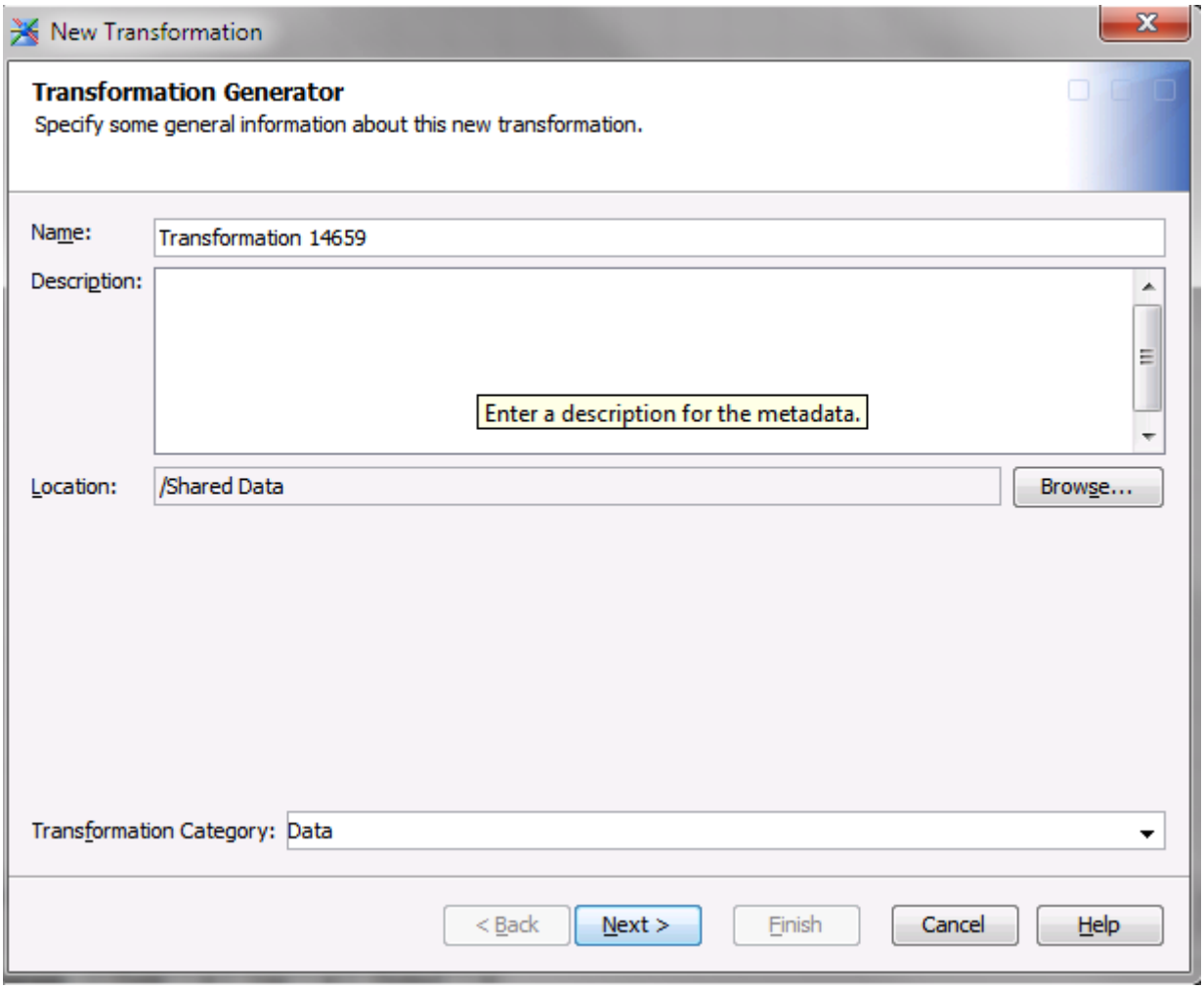

Figure 2. New Transformation Generator

We have created dedicated folders for our transformations. Keeping them organized in separate folders helped us during our migration process as well.

Since we are developing custom transformations it's a really good practice to write some brief description for the transformation. This actually will display on basic properties window on SAS DI studio.

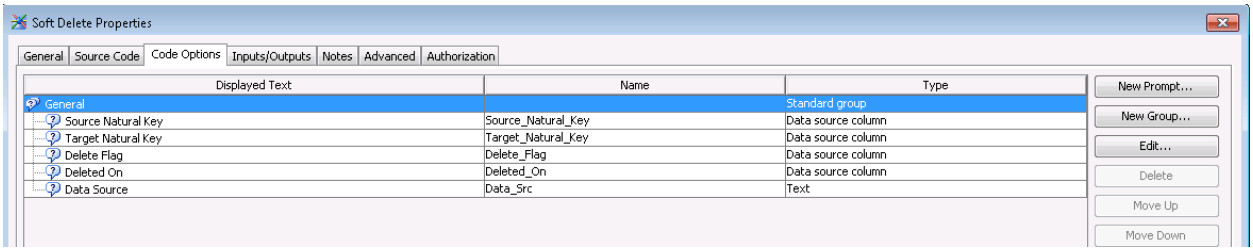

Figure 3. New Transformation Generator

In new transformation generator wizard screen after source code is where we can tie our data source columns to our transformation.

Some cases we used transformation which did not use any input or output. You can setup your tables as input tables on transformation properties screen of transformation generator wizard.

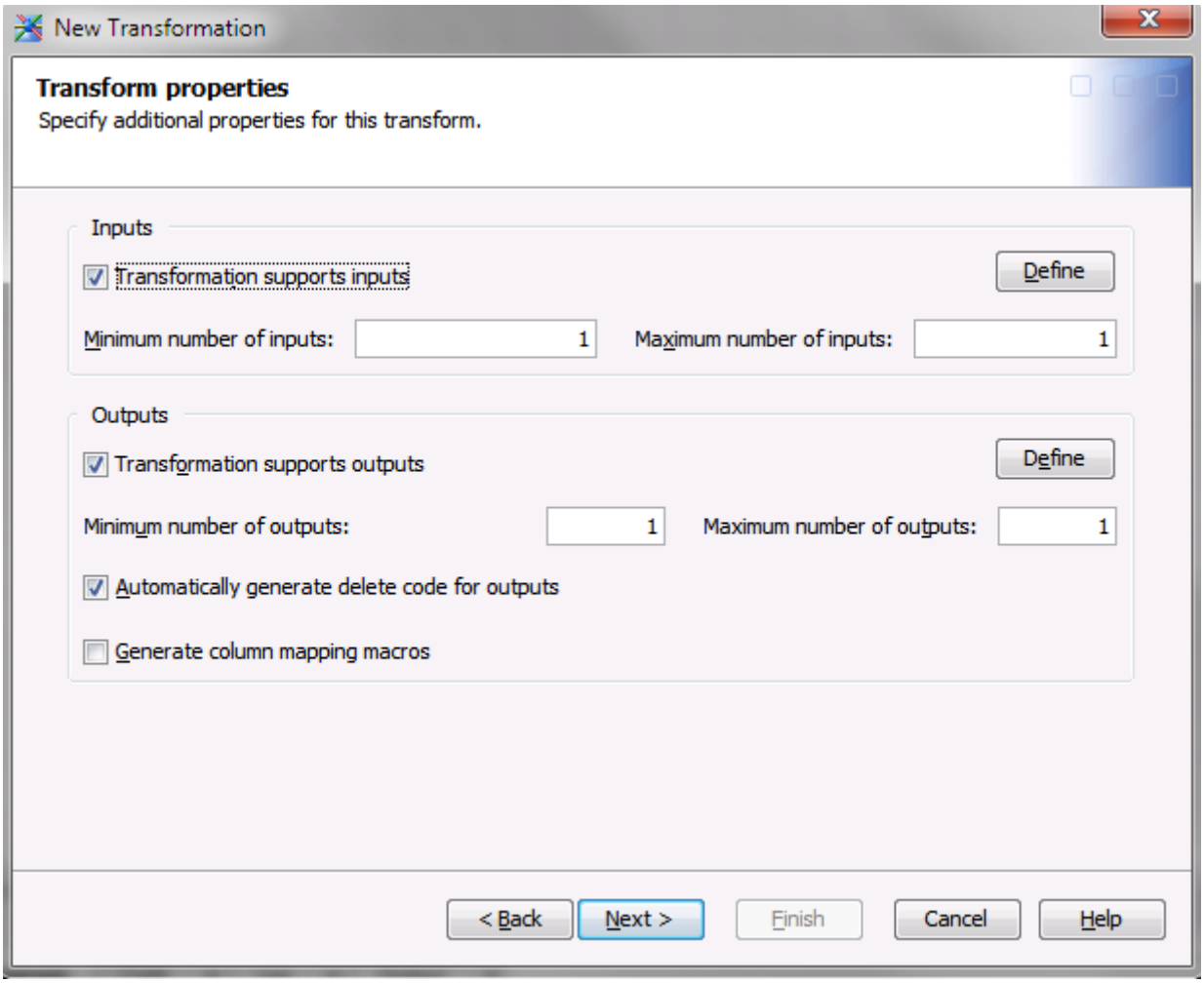

Figure 4. New Transformation Input/Output

As seen in Figure 6 We can setup our input table names.

One thing I have found really useful is to use table names of my choice.

Using my own table names helped me keep track of input and output tables. Also setting a proper prompt here will help you identify which port on the transformation to use when you are connecting your tables.

This order might be important based on how you wrote code for custom transformation. I have assigned and used macro variables for specific tables in code so knowing which port to connect source tables in order was important.

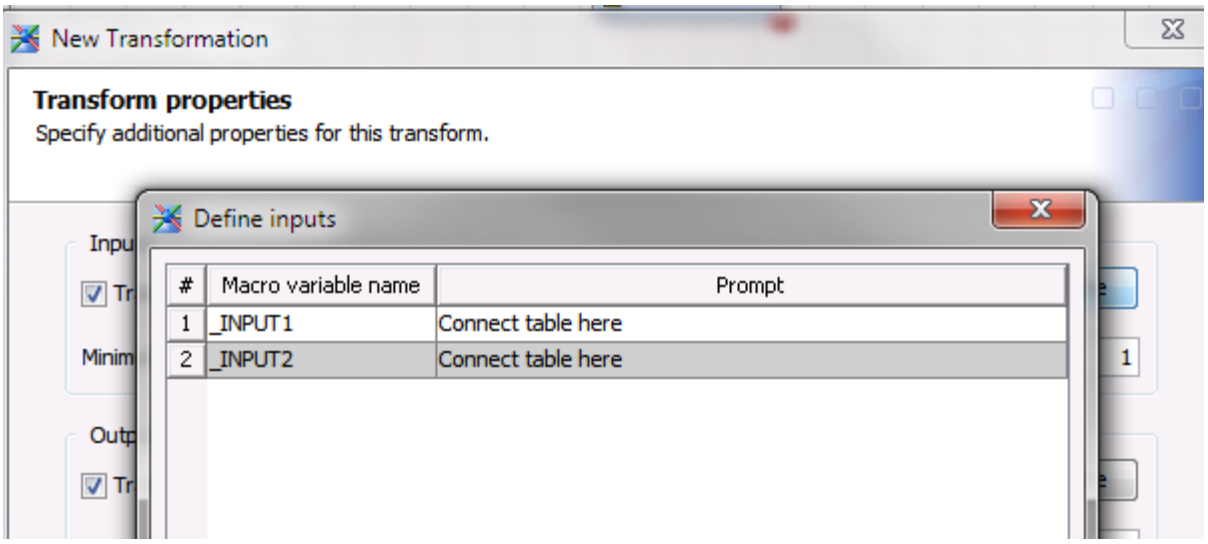

Figure 5. Define Input/Output

MODULARIZED TRANSFORMATIONS

GATHERING STATISTICS TRANSFORMATION

As we mentioned transformations are there to perform a specific task such as extract data, transform data or loading data. We can also have transformations modular to be used for some preparations before job.

Building a warehouse requires some maintenance tasks to be executed. Working on ORACLE database it became clear to us tables which had large volumes of data change did benefit from gathering statistics. By gathering statistics in ORACLE we basically quantify storage characteristics of tables and indexes.

We decided to make a custom transformation and use it in SAS DI Studio jobs. This way before job started we could gather statistics on needed tables and improve job performance.

Here is code we used for gathering table statistics:

```
%macro gather_statistics(source_tbl=,);

%let stmt = execute
DBMS_STATS.GATHER_TABLE_STATS('"%qscan(&source_tbl.,1,%str(.))"',
'"%qscan(&source_tbl.,2,%str(.))"');

proc sql;
   connect to oracle as db (&connect_str);
   execute ("&stmt.") by db;
quit;

%mend;

%gather_statistics(source_tbl=&_INPUT0.);
```

Also after ETL flow completion we ran schema version of same transformation with different execute statement.

```
%let stmt = execute
gather_schema_stats('"%qscan(&source_tbl.,1,%str(.))"');
```

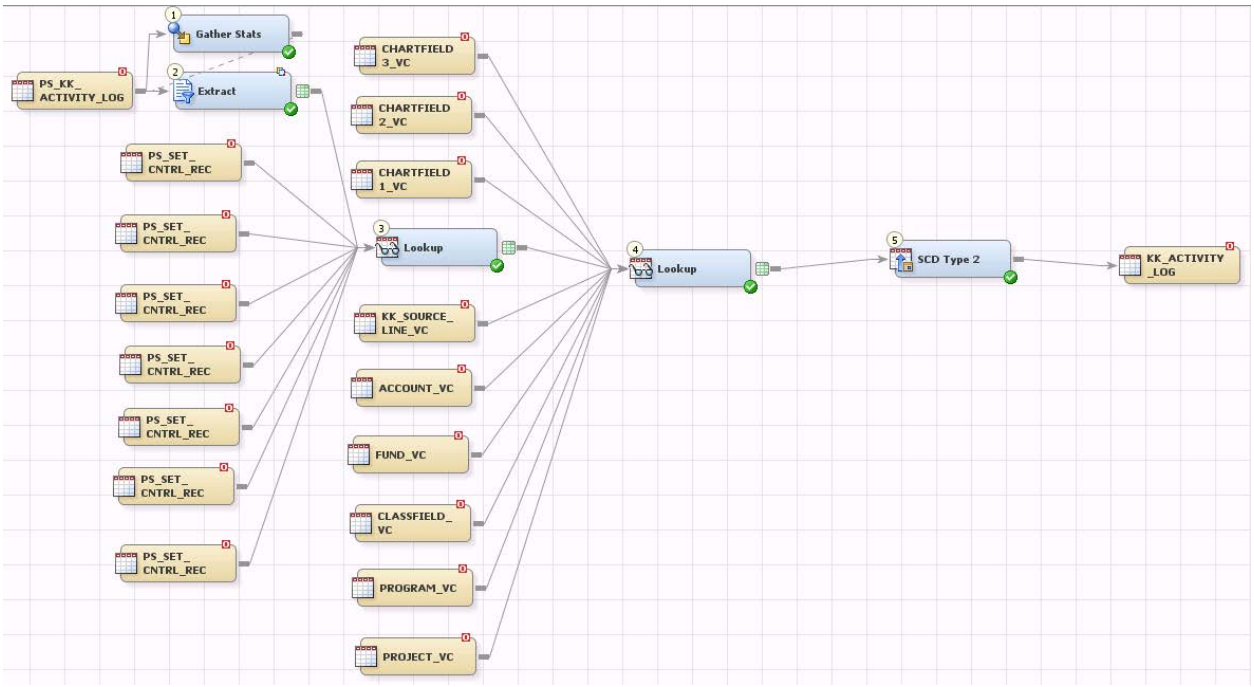

Figure 6. Example use in a job

As you can see from Figure 6 we have our gather statistics transformation to be run before extract happens. We could also add it to pre code section of the job too. But then for every job you would need to edit based on table name. This keeps it more in line with SAS DI Studio job construction on canvas.

We can write whole SQL procedures in SAS. But keeping those in ORACLE made it easy to maintain and troubleshoot. We have experienced keeping SQL procedure in a custom transformation is hard to debug.

REFRESH MVIEW TRANSFORMATION

Another transformation we worked on was for refreshing materialized views on ORACLE. We created another custom transformation for this operation. After every ETL cycle we needed to rebuild some materialized views in ORACLE.

Refresh Mview Properties (Read-Only)

General | Source Code | Code Options | Inputs/Outputs | Notes | Advanced | Authorization

```
1  %macro refresh_mview(mview_name=,connect_str=);
2
3  %let stmt = execute refresh_mview('&mview_name');
4
5  proc sql;
6  connect to oracle as db (&connect_str);
7  execute( "&stmt." ) by db;
8  quit;
9
10 %mend;
11
12 %let input_name = %upcase(%substr(&_INPUT0,%eval(%index(&_INPUT0,.)+1)));
13 %refresh_mview(mview_name=&input_name,connect_str=&_INPUT0_connect);
14
```

Figure 7. Example refresh transformation

Since our ETL jobs were in flows we wanted to have refresh jobs to be triggered after ETL jobs were complete. We simply made a transformation that refreshed materialized view that was connected to it. We then use this for every materialized view that needed to be refreshed in certain order.

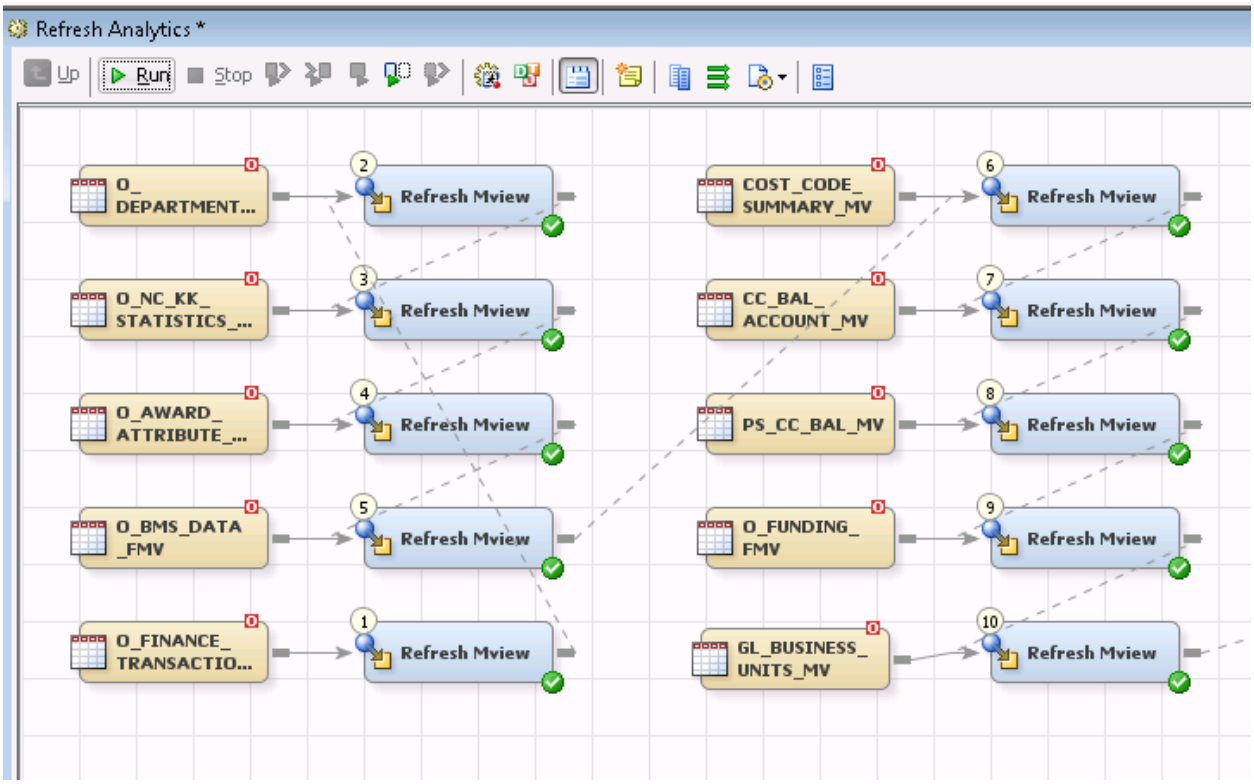

Figure 8. Use of refresh transformation in a job

DATA STATUS TRANSFORMATION

Every day after ETL cycle was complete we needed a way to check status of data. One step of this transformation is doing a basic row count on source and target.

For this purpose we created a new transaction to do row counts. Our approach was having a transformation which required two input tables and produced source and target row counts.

This transformation was used separately to create a data status table. This way after every ETL cycle, for the tables we had main focus we did create a report.

Another way to do a data quality check was checking some values for certain columns. Such as amount on transactions tables. We needed to run our job and create our report based on total dollar amounts matching. Since basic transformation is for row counts we could not use this transformation but also we did not wanted to create new transformation for every measure that needed quality check.

Our approach was to create mapping control table and eliminate complex logic to determine how comparison was supposed to be done. On mapping control table we created series of options such as row counts, amounts or values being populated. When transformation was used an option was needed to be picked. This way we were able to keep transformation modular and reusable in different jobs.

Code and usage can be seen in Figure 9 and Figure 10.

Row Count Compare Properties (Read-Only)

General | Source Code | Code Options | Inputs/Outputs | Notes | Advanced | Authorization

```
1   %global RowCount_SOURCE STBL_NAME;
2   %global RowCount_TARGET TTBL_NAME;
3   %global Difference;
4   %global Runtime;
5   %macro row_count_compare(source_tbl=, target_vc=, output_tbl=);
6
7   proc sql;
8   SELECT COUNT(*) into :RowCount_SOURCE FROM &source_tbl.;
9   run;
10  proc sql;
11  SELECT COUNT(*) into :RowCount_TARGET FROM &target_vc.;
12  run;
13
14  data &output_tbl.;
15        Format RUN_TIME datetime20.;
16        Format ETL_SOURCE_TABLE $30.;
17        Format ETL_TARGET_TABLE $30.;
18        ETL_SOURCE_TABLE = "$qscan(&source_tbl.,2,$str(.))";
19        ETL_TARGET_TABLE = "$qscan(&target_vc.,2,$str(.))";
20        SOURCE_COUNT = &RowCount_SOURCE.;
21        TARGET_COUNT = &RowCount_TARGET.;
22        RUN_TIME = %sysfunc(datetime());
23  run;
24  %mend;
25  %row_count_compare(source_tbl=&_INPUT0., target_vc=&_INPUT1., output_tbl=&_OUTPUT0.);
```

Figure 9. Code for row count comparison

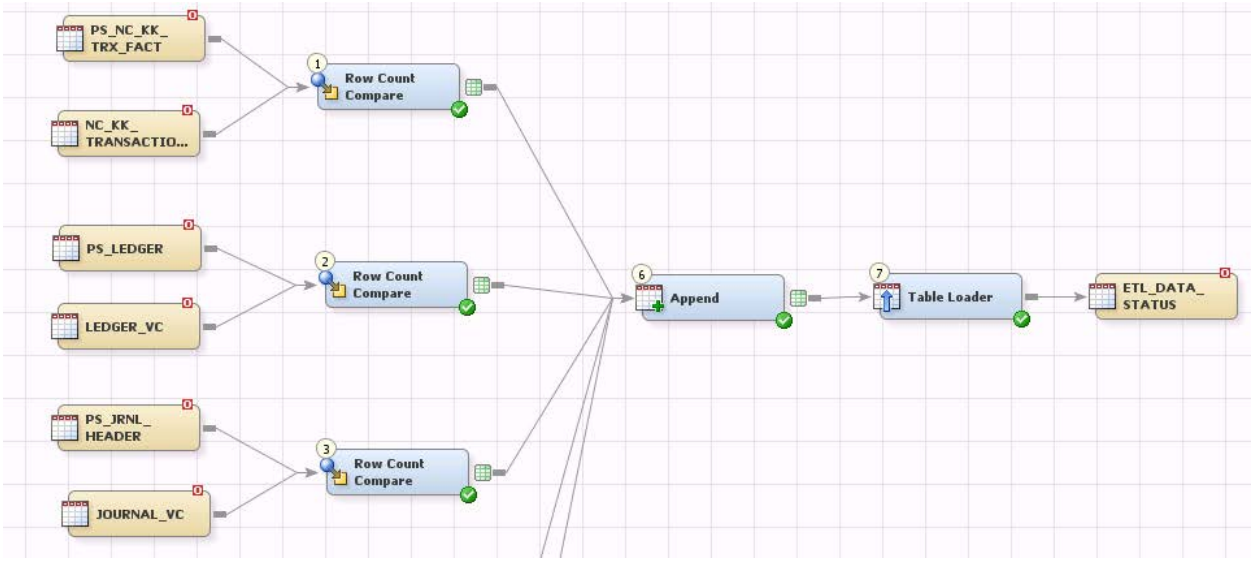

Figure 10. Use of row count transformation in a job

ENABLE/DISABLE INDEXES TRANSFORMATION

For some operations on database we needed to have a transformation to mass disable and then enable indexes on a table. SAS DI Studio has these options for some load type transformations but we needed to use it without being in context of a load action. Also our refresh transformation benefited from this transformation because ORACLE refreshes were being completed much faster when indexes were disabled. For some ETL jobs before job, disabling index on that table also did increase the performance. Also having flexibility on how we build indexes was a plus since we did rebuild them parallel.

We simply gathered indexes on a table by:

```
proc sql;
   connect to oracle as db (&connect_str);
   create table perm_indexes as select * from connection to db (select *
   from user_indexes where table_name = '&table_name');

quit;
data _NULL_;

   set perm_indexes nobs=nobs;

   stmt = "Alter index "|| strip(INDEX_NAME) ||" unusable;";
   call execute('%exec_stmt('||stmt||')');
run;
```

After disabling similarly we did enable them by:

```
   stmt = "Alter index "|| strip(INDEX_NAME) ||" rebuild parallel 16;";
```

As you can see from example using parallel keyword we were able to get ORACLE rebuild indexes using 16 cores.

CONCLUSION

In this paper we did present multiple different examples of using custom transformations other than extract data, transform data or loading data. There are many other ways users can write SAS code and make them transformations to be used in a modular and flexible way in SAS DI jobs.

Modularizing our transformations furthers the goal of code reusability and flexible job flows. Generally interacting with SAS DI using transformations makes it easier to use and maintain rather than keeping code in pre-code and post-code sections of jobs. Working this way is also more in line with SAS DI Studio job construction on canvas.

As SAS DI Studio keeps progressing I think custom transformations will be more important and traded among DI Studio developers.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

EMRE G. SARICICEK
The University of North Carolina at Chapel Hill
emre@unc.edu