

SESUG Paper SD-185-2017**Guide to ETL Best Practices in SAS® Data Integration Studio**

Sai S Potluri, Synectics for Management Decisions; Ananth Numburi, Synectics for Management Decisions;

ABSTRACT

This Paper converses about the ETL best practices, tips and techniques that can be followed by developers when using SAS® Data Integration Studio for building and maintaining data warehouses/data marts. This Paper discusses the following topics.

- Best ETL Design Practices.
- Helpful coding insights in SAS DI studio.
- Techniques and implementation using the Key transformations in SAS DI studio.

INTRODUCTION

ETL plays a very critical role in integrating the data from multiple applications and sources into one centralized location - the Data warehouse. The purpose of this paper is to serve as a simple guide to the ETL Best practices using SAS Data Integration Studio in implementing, deploying and maintaining a production end-to-end data management environment to support Business intelligence initiatives across an enterprise. In this paper, we shall talk about best practices, tool functionality, comparison between different transformations and capabilities offered by the SAS Data Integration Studio.

ETL DESIGN PRACTICES

The scope of the ETL varies from project to project and based on its complexity, due to which it is not possible to provide an ideal list of ETL best design practices that would suit every project. But, below are some ETL best design practices that might serve as a quick checklist when working on your ETL design.

Data Requirements:

It is extremely important to have a good understanding of the source data, their dependencies, their different data formats, data relations and the data volume. This is the first step in the data warehouse projects and is wiser to spend more time in this phase and fix any data issues that are seen.

Infrastructure:

Build the right infrastructure by forecasting the future data needs for the organization and ensure that it can handle the ETL Loads by stress testing the ETL platform with full volume to evaluate if any additional resources are needed to be added to the infrastructure.

Process minimum data:

As part of the ETL daily batch processing, process only the incremental or the delta data where ever possible. Bring only the data fields that are needed into the ETL.

DBMS Processing:

Make the Database do the processing where ever possible. This will reduce the data movement between the DBMS and SAS server and can help improve the performance of the ETL Jobs.

Bulk Load:

Make use of the Bulk Loading options that are available when large volume of data needs to be processed.

Parallel Processing of the ETL Jobs:

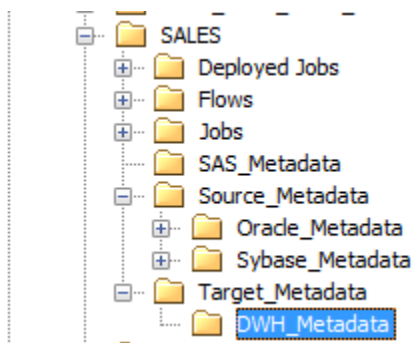
Enable parallel processing of the ETL Jobs either by using the Job Scheduler or inside the ETL tool itself. This will reduce the ETL run times.

ETL Code Modularization:

Having a modularized code helps in avoiding to create the same code repeatedly in several ETL Jobs. Creating custom transformations will reduce the effort and helps with easy maintenance of the code.

Create a ETL Folder Structure

ETL repository is a collection of metadata which includes Libraries, Source & Target Metadata, Jobs and Flows. All the ETL code in the repository needs to be organized as per the team's need. A repository can have multiple projects in it. In the below display 1, "SALES" is a project folder which has sub-folders underneath it to store the Source Metadata, Target Metadata, Libraries, Jobs, Deployed Jobs and Flows.



Display 1. Project Structure

You may also create a shared folder as well to store the code objects that need to be shared across multiple Projects.

ENVIRONMENT DETAILS:

Below are the environment details based on which the best practices in this paper were written.

- SAS Data Integration Studio 4.9
- SAS 9.4 (TS1M2)
- Red Hat Enterprise Linux Server release 6.9

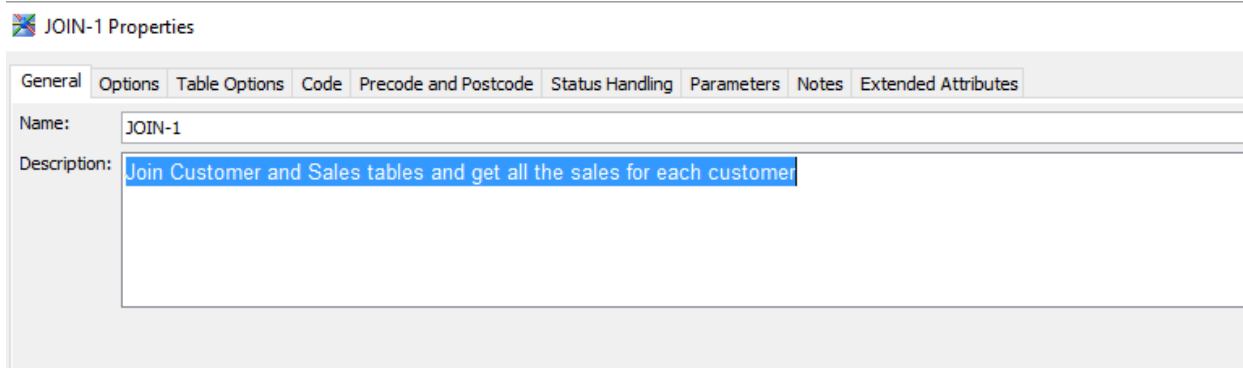
ETL Job Naming Standards

It is always a good practice to follow a certain job naming standard in any Project. It can be based on the Target table on which the load must be performed or based on the subject area etc. Below is one example of the Job naming standard.

LOAD_<DBNAME>_<TARGET_TBL_NAME>

Similarly, it is a good practice to follow a certain naming standards for the transformations in an ETL Job. This can be particularly helpful to navigate through logs and identify the transformation where the failure has occurred. In the "Description" section of the Transformation, provide additional descriptive details to help understand what is being accomplished in the transformation.

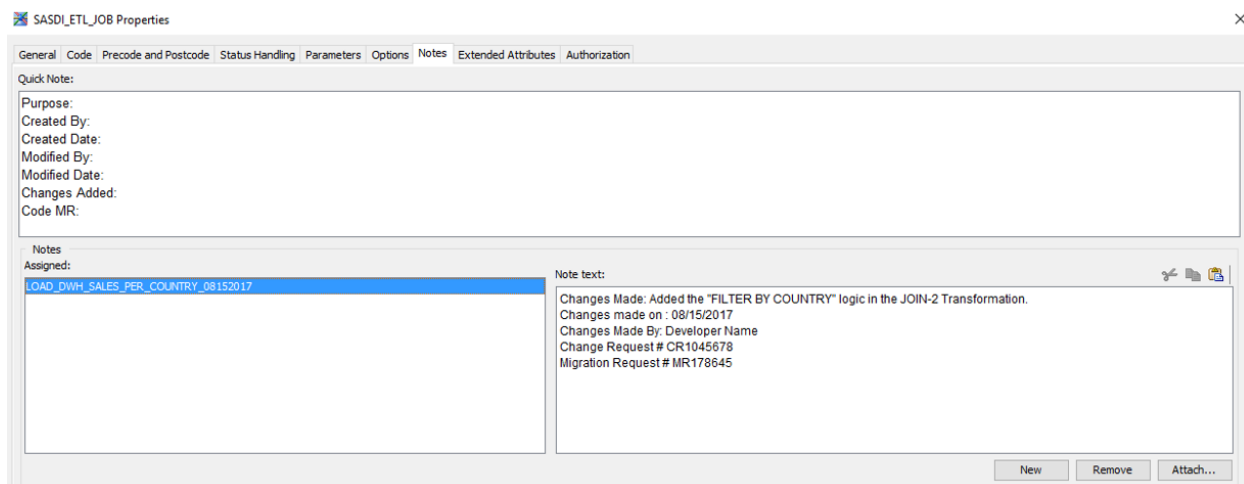
EX: JOIN-1, JOIN-2, MERGE-1, TBLLDR-1 etc



Display 2. Example of Standard for Naming the Transformations.

Maintain a Changelog for the ETL Jobs

Most organizations follow agile methodology software development, where the business requirements evolve over time and there will be frequent changes to the code. Maintaining a changelog in the ETL Job will be very helpful in tracking these changes. It also helps other members in the team to better understand the purpose of the job and support any additional changes needed in the future. Use the "Notes" tab in the Job Properties to list all the details. You may even save the notes with the JOB_NAME_<DATE> as metadata in the SAS folders. Additional documentation can also be attached if needed.



Display 3. Change Log

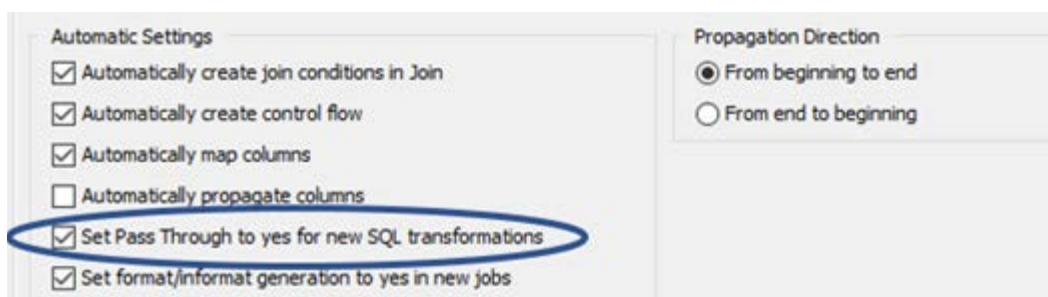
SAS DI STUDIO CODING INSIGHTS

In this section, we will go through some coding tips and insights while picking the right transformation when building the ETL Jobs in SAS Data Integration Studio.

ENABLE EXPLICIT DB PASS-THROUGH BY DEFAULT FOR ALL THE SQL TRANSFORMATIONS

Setting the SQL transformations to pass-through processing pushes all the DBMS specific statements to the DBMS and the processing takes place in the DBMS. This may help you with some performance improvement in some cases by minimizing the data movement between SAS and DBMS and performing in-database execution. Please Keep in mind that when this option is enabled, only the DBMS specific functions will work. Any SAS specific functions will not work. You can set the SQL Transformations to "Pass-through" as default by selecting the below option.

select Tools→Options→Job Editor Tab, and then select the pass-through check box in the Automatic Settings area as shown in the below display 4.



Display 4. Pass through setting for SQL transformations

PRE-CODE AND POST-CODE:

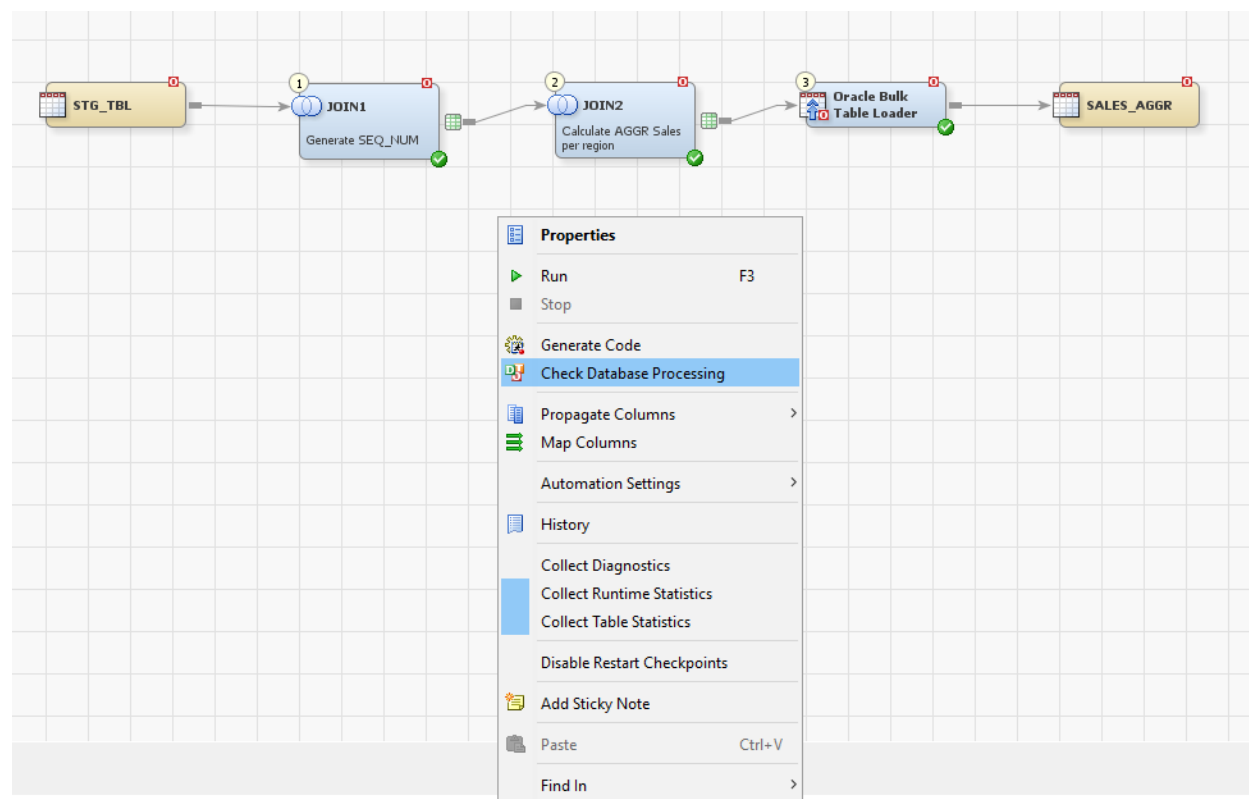
If there is a need to set any SAS options or to run any pre-requisite operation prior to the job execution, you may set that in the Job Pre-code. Any transformation specific operations can be set in the transformations

Pre-code. Similarly, any post-code operations after the job completion can be added to the Post-code of the ETL Job.

DATA PROCESSING IN DBMS:

When you have a requirement that needs tables which exists in the same DBMS, then you may build a job where all the processing is completely pushed to the DBMS. The advantage with this approach is that, it makes the DBMS perform all the processing, minimizes the data movement between the SAS Server and the DBMS, which in turn helps in improving the ETL performance. To ensure all the data processing happens in the DBMS itself, you need to set the Physical Storage of the output datasets of each transformation in the DBMS by selecting the needed DBMS library. You may either create them as a view or as a table.

Here is one example where all the processing is pushed to the DBMS. You can check if the process will be run in the DBMS by using the option “Check Database Processing” in the job properties.



Display 5. SAS DI Job with DBMS Processing

SURROGATE KEY GENERATOR VS MONOTONIC ()

Both surrogate key generator and monotonic function can be used to generate sequence numbers in SAS.

Monotonic

A source table with 16,260,857 rows was taken, surrogate keys were generated using monotonic function in the Table Loader transformation and this operation took 318 seconds. To generate the surrogate keys using a monotonic function, get the current max surrogate key from the target table by adding the below proc sql statement in the Pre-code section of the Table Loader and then use monotonic() + &TBL_MAXKEY to generate the surrogate keys.

General Load Technique Mappings Options Table Options Code Precode and Postcode Status Handling Parameters Notes Extended Attributes

☒ Precode:

```
LIBNAME DWHDWP ORACLE UPDATE_LOCK_TYPE=ROW INSERTBUFF=32767 PATH=DWH SCHEMA=DWP AUTHDOMAIN="DWH DWP OracleAuth" ;

proc sql noprint;
  SELECT COALESCE(MAX(DWH_DIM_ID),0) INTO :TBL_MAXKEY FROM DWHDWP.DWH_DIM;
quit;
```

Target table: DWH_DIM (LOI_DIM)

#	Column	Column Description	Expression	Type	Length	Informat
1	DWH_DIM_ID	LOI_DIM_ID	monotonic() + &tbl_maxKey.	Numeric	8 11.	

Display 6. Table Loader with monotonic function

Surrogate Key Generator

The same source table with 16,260,857 records was taken to generate sequence number using the Surrogate Key Generator and it took 415 seconds to complete this operation. Same source and target tables were used in both the jobs, but the time taken by the surrogate key generator is more compared to monotonic function. The reason for the Surrogate key generator to take longer is because it sorts both the input and output tables by the business key column before merging the data in to the target table. If both the source and target tables have large volume of data, then sorting both the datasets will consume lot of time and before performing the merge, the surrogate key generator replaces the data in the target table with the source data due to which you might lose all the historical data. If the need is to just insert the new records into the target table without replacing the data, then select the “keep just the input records in the output table” option in the Surrogate Key options and use a table loader to insert the new records.

Note: Please keep in mind that monotonic is an undocumented SAS function and has its limitations when used in complex SQL procedures to generate surrogate keys.

SCD Type 1 vs Table Loader with Inserts / Updates

Most of the folks who work in the Data warehouse environments are familiar with Slowly Changing Dimensions and their different types. Below is a short explanation of SCD Type 1.

SCD Type 1: If an attribute value changes in a dimension table, then the current row is overwritten with the latest values, without retaining a history.

SAS DI studio provides a SCD Type 1 Loader transformation that performs loads in to a slowly changing type 1 dimension table. This transformation compares the source dataset to the target dataset, detects any changes, generates the surrogate keys (if generate surrogate key option is selected), appends any new rows using SAS PROC APPEND and performs updates on the changed rows in the Target Dataset. While majority of these operations run in few seconds, usually the bottleneck is with performing the updates. To perform the updates, it generates an update statement for each field which looks like this...

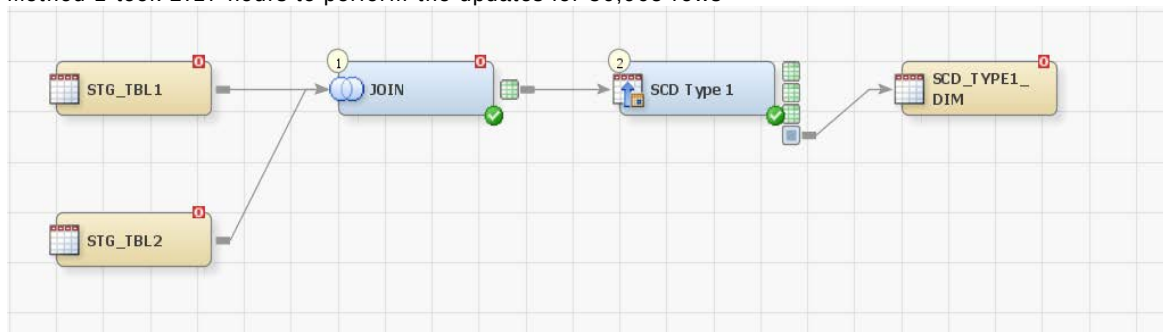
```
update DWP.SCDTYPE1_DIM as m
set STTS_CODE = (select STTS_CODE from STG.CUSTOMER as t
where m.STTS_DIM_ID = t.STTS_DIM_ID)
```

If the target table has 100 columns, it generates 100 similar update statements like above for each column and runs them for all the rows that are marked for updates. This operation can take a very long time if there are more updates to be performed on the target dataset. To run a performance comparison, a stage table with a modest 42,531 rows has been taken and a string field of varchar2(255) field has been modified in the target dataset for 36,965 rows.

Two ETL jobs with different design have been created to perform the updates.

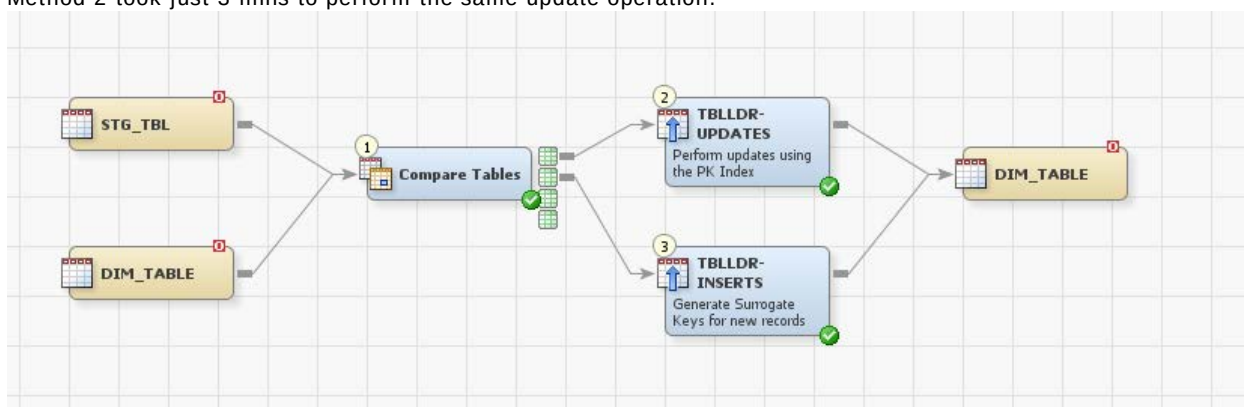
- Method 1: SCD Type 1 Loader to perform the updates
- Method 2: Compare tables and Table loader transformations to perform the updates.

Method 1 took 2:17 hours to perform the updates for 36,965 rows



Display 7. Method 1 using SCD Type 1 loader to perform Insert/Updates

Method 2 took just 3 mins to perform the same update operation.



Display 8. Method 2 to perform Insert/Updates

JOIN VS EXTRACT

In most of the ETL jobs, either the DBMS tables or SAS Datasets are read and some transformation logic is applied as per the business requirements and is finally loaded into a DBMS table/SAS Dataset. To read the data from these tables & Datasets, you can use either an Extract transformation or a SQL Join Transformation in SAS DI Studio. While both transformations can perform the below listed operations, you need to pick the transformation based on the logic that needs to be performed.

- Reading the data from a DBMS Table or a SAS Dataset.
- Perform Aggregation calculations.
- Sorting the Data.

If the requirement is to read a single table and perform some transformation on it and load it in to a DBMS table/SAS Dataset, then Extract offers better performance compared to a Join transformation, but when you need to read multiple tables, perform joins between them and some data transformations- in that case a join might be a better option over Extract. In case of a Join transformation (Pass Through option should be set to Yes), if all the tables exist in the same DBMS, then all the joins and the data processing takes place in the DBMS itself. If the same operation is performed using an extract transformation, then all tables need to be read and the data needs to be landed in SAS Datasets and use a join transformation to perform a join between them. If the tables have large volume of data, then the performance of the job which uses an extract performs slower than the one using a join transformation.

OPTIMIZE READBUFF AND UPDATEBUFF PARAMETERS

READBUFF or UPDATEBUFF value is a positive integer which determines the number of rows that can be read in to the memory at a time for processing. Determining this value is critical to improve the performance and to avoid the Memory errors. Setting this value too high may result in out of memory error and setting it too low might impact the

performance as the number of rows processed at a time will be very less. So, calculating the optimal value for the READBUFF is needed.

Set the value for READBUFF based on the following parameters

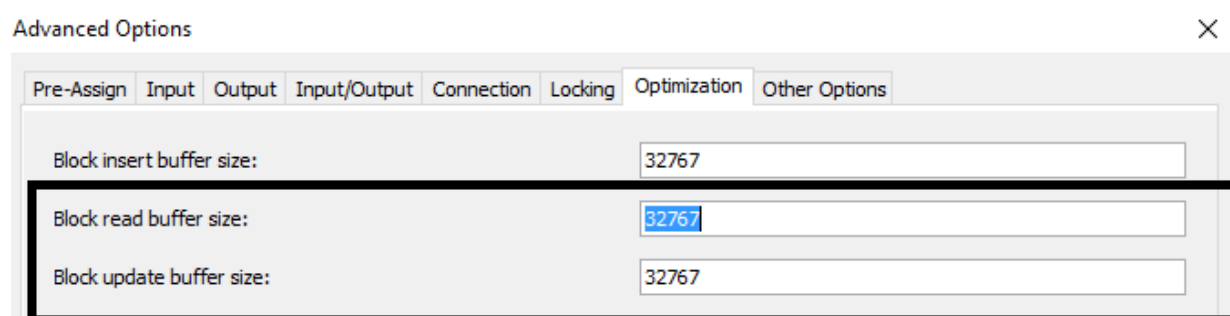
- ROW SIZE: Total length of all the columns in the Table. All the numeric columns are considered 8 bytes in length.
- MEMSIZE: The MEMSIZE system option specifies the total amount of memory available to each SAS session. This value is set in the sasv9_local.cfg file underneath the SASFoundation directory.

Calculate the Optimal READBUFF value using the below formula.

$READBUFF * ROW_SIZE < MEMSIZE$

The value of $READBUFF * ROW_SIZE$ should be less than the MEMSIZE value set.

This value can be changed in a SAS Library in the optimization tab of the Advanced Options section as shown below. READBUFF and UPDATEBUFF values need to be tested in the supervision of the SYSADMIN to monitor the use of memory on the SAS server.



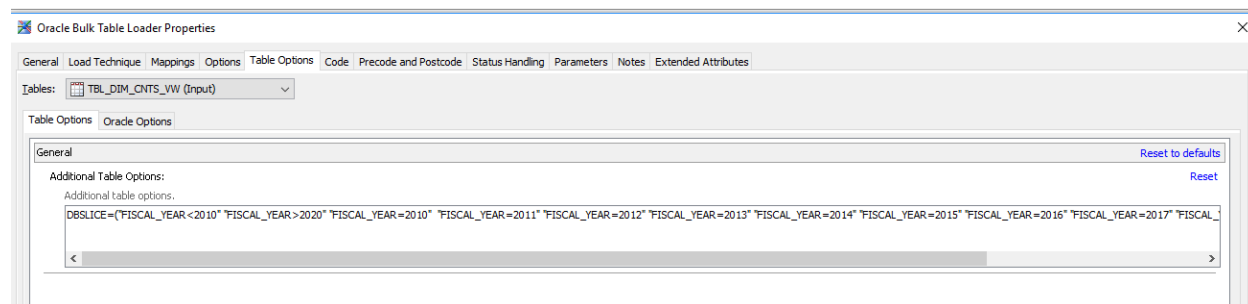
Display 9. READBUFF and UPDATEBUFF setting in the Library

UTILIZE DBSLICE TO IMPROVE THE LOAD PERFORMANCE

Reading data from RDBMS source can be time consuming when reading large tables. Single threaded processing can take up anywhere from several minutes to hours, just to read the data. As most ETL's run during the off-peak hours, multithreaded processing can be effectively used to the maximum server capacity to improve the runtime for ETL Load. SAS provides a way to perform multi-threaded operations to read the table in parallel using DBSLICE features. DBSLICE performs the multi-threaded reads by splitting the data into multiple groups. Below is a scenario explained using DBSLICE. In the below example, the table has the data grouped on FISCAL_YEAR. Based on the number of groups, DBSLICE creates the several threads to read the data which helps significantly in improving the performance.

```
DBSLICE=("FISCAL_YEAR=2011" "FISCAL_YEAR=2012" "FISCAL_YEAR=2013" "FISCAL_YEAR=2014"
"FISCAL_YEAR=2015" "FISCAL_YEAR=2016" "FISCAL_YEAR=2017" )
```

In this example, DBSLICE option has been set in the "Table Options" of an Oracle Bulk Loader transformation.

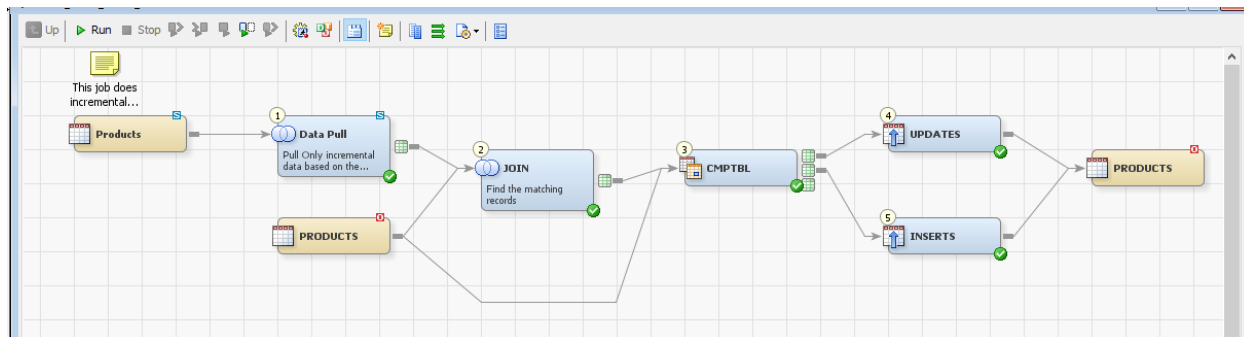


Display 10. Setting DBSLICE in Table Options

When the ETL Job is run, you may monitor the sessions on the database and will see multiple sessions with different SQLID's with DB Slice Option. Please be cautious when you try to implement this option and try it in the supervision of your DBA.

COMPARE TABLES TRANSFORMATION VS JOIN TO DETECT INSERTS/UPDATES

As part of the daily ETL batch processing, you may need to process the incremental data. The incremental data is compared with the target table to detect the changed records (Updates) and new records (Inserts). A Join or a Compare tables transformation can be used to perform this operation. The advantage of Compare Records over the Join transformation is that it detects the changes based on the primary key comparison and the columns selected in the compare columns section. This way the inserts and real updates are identified. If a Join is used to detect the changes, then all the matching records are marked as updates and will be updating them in the target dataset even if there is no real update to that row. In the below display11, an incremental job was built using compare tables.



Display 11. ETL Design to process Incremental data

Below is a design approach to build an incremental job

1. Pull the incremental data based on the audit columns in the source table.
2. Join the incremental data with the target table on Primary Key to find the matching records.
3. Use Compare Tables to compare the matching records with the delta records. This way you are not comparing every record in the target table to the source incremental data and compare is done on a small dataset.
4. Connect the Changed records and New records datasets to table loaders and complete the load.

Compare Tables transformation also outputs the Unchanged and Missing records. You may use them if needed as per your business requirements.

ISSUE WITH SET OPERATORS TRANSFORMATION:

There is a limitation using Set Operators transformation in SAS DI studio. When a union operation needs to be performed between two tables using Set Operators transformation in SAS DI Studio, the underlying deployed code changes back and forth between SAS PROC SQL and DB SQL Pass-through. If you use any Oracle or SAS specific functions in this transformation, then there is a possibility of a job failure based on the way the code was deployed. We noticed this behavior in SAS Data Integration Studio 4.9. The alternate way of performing this operation is use a Join transformation performing a full outer join or a User Written transformation with the union logic.

Below are the screen prints of how the code changes when the Set Operators transformation is re-opened multiple times.

```

proc sql;
  create view work.W2MOFFLV as
  /* Returns all unique rows from the two query results. */
  select
    CMTU.XIP_CODE as LOC length = 32
    format = $32.
    informat = $32.
    label = 'LOC'
  from
    HSDFLFP.CMTU
  union all
  select
    DDC.LOC length = 32
    format = $32.
    informat = $32.
    label = 'LOC'
  from
    HSDFLFP.DDC
;
quit;

```

Display 12. Code in the Set Operators deployed as SAS PROC SQL

```

@proc sql;
  connect to ORACLE
  (
    DSNMAX_TEXT=32767 PATH=STG AUTHIDMAIN="W2STG_FLFP_OracleAuth"
  )
  create view work.W2MOFFLV as
  select
    LOC length = 32
    format = $32.
    informat = $32.
    label = 'LOC'
  from connection to ORACLE
  (
    /* Returns all unique rows from the two query results. */
    select
      CMTU.XIP_CODE as LOC
    from
      CMTU CMTU
    union all
    select
      DDC.LOC
    from
      DDC DDC
  )
;

```

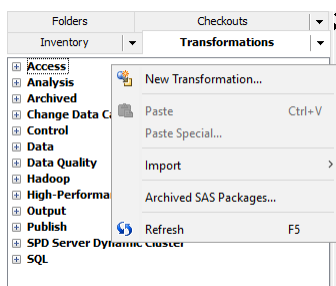
Display 13. Code in the Set Operators transformation deployed as SQL Pass-through

WORK AND UTILLOC:

When sort operation is performed in SAS, SORT creates intermediate files in the WORK and UTILLOC locations. They are set in the sasv9_local.cfg file underneath the SAS Foundation directory. If possible, create these directories on two separate file systems with adequate space allocation. When UTILLOC option is not set, all the files get created in the WORK directory. Segregating these two location may help in improving the sort performance. It is also a good practice to schedule a cleanup job at the end of the ETL Batch Processing to cleanup of the intermediate files in the WORK and UTILLOC directories.

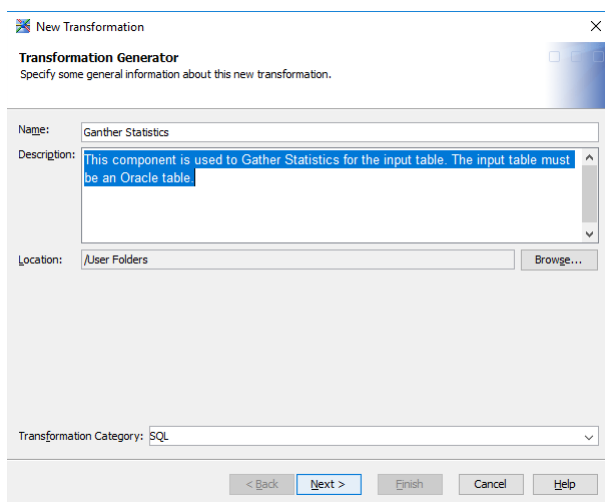
CUSTOM TRANSFORMATIONS:

The need for custom transformation arises when the same ETL logic is being implemented in multiple Jobs. To reduce the redundancy, you may create some custom transformations as per your business needs. Below is an example on how to create a custom transformation for automating the code to gather statistics on an Oracle table at the end of the load. To create custom transformation, navigate to the transformations tab and right-click anywhere and select New Transformations from the options as shown below in display 12.



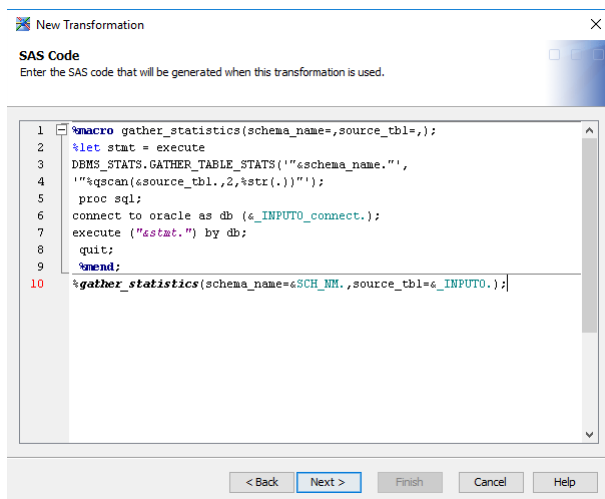
Display 14: Create New Transformation

New transformation wizard will open, where you can provide a name, location where the transformation must reside and the category that it belongs to. In the below example, it is named as "Gather Statistics".



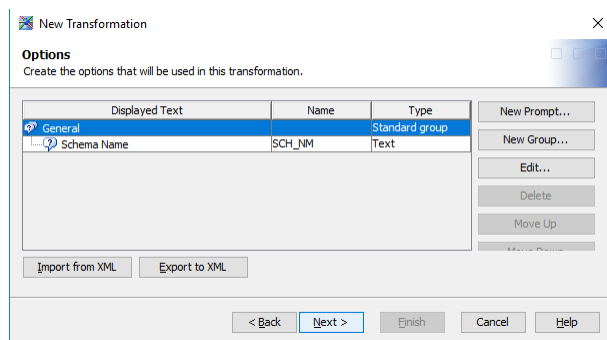
Display15: Transformation Wizard

Click Next. Now SAS will provide a window to add the user-defined code that will perform the gather statistics operation.



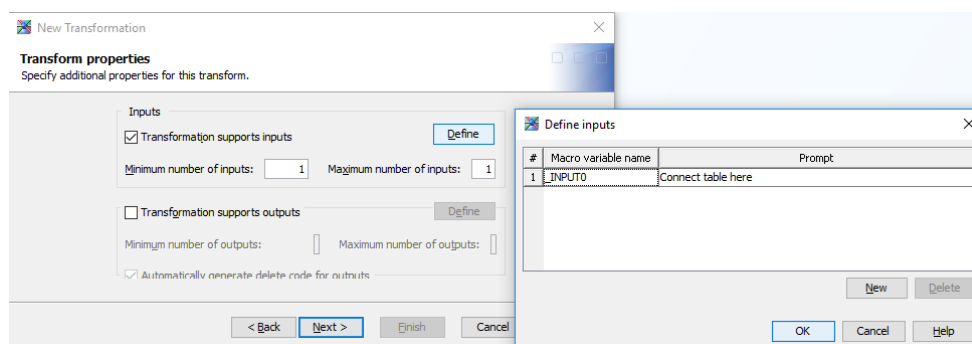
Display16: Source Code

Click Next once the code is completed. Now you will see Options tab where you can define the variables that are used in the source code as shown below



Display 17: Variables

Once the variables are defined click next and you will see a window where you can define the transformation properties such as how many inputs/outputs can the transformation allow and define naming for the inputs and outputs as shown below.



Display 18: Transformation Properties

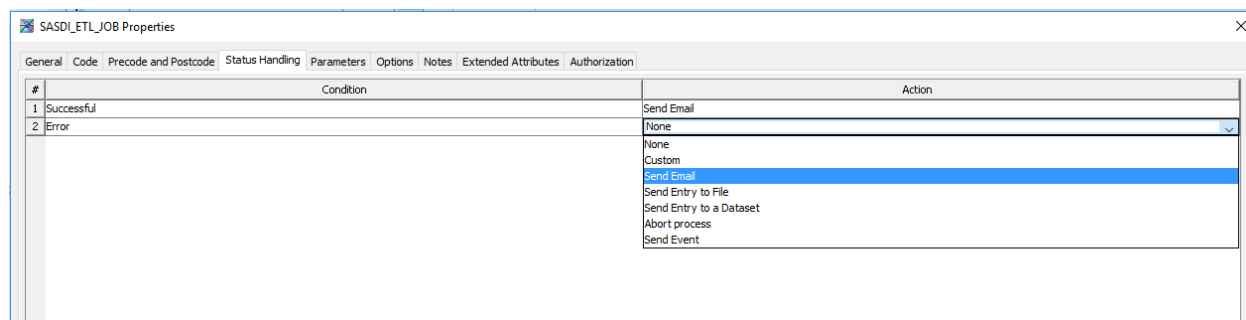
Click Next then click Finish. The custom transformation is now created and can be used in the SAS DI jobs.

ERROR HANDLING

A good error handling design will have measures for preventing and addressing job failures at the right granularity. This should be part of the initial design of the ETL architecture. You may need an error handling framework based on the needs of your organization. Consider the following:

- Failure Prevention: Can the ETL Load by-pass any errors for the load process to complete? Or fix the failure and re-start the jobs from the point of failure?
- Job Failure Notifications: Create an Error notification alert mechanism.
- Check for dependency: If process B is dependent on process A, In case of failure of Process A, Process B should not be executed.

SAS DI studio provides some Error Handling options in the "Status Handling" tab in the Job properties. You may pick the option that suits you best for Errors or Warnings.



Display 19: Error Handling in SAS DI Studio

If the action "Send entry to a Dataset" is selected, SAS creates a table with the Dataset Name in the specified library. This may help you in creating some reports for analysis on the data that is captured in the table.

LOGGING

ETL Logging is a key item which is essential to the success of any ETL architecture. ETL Logging is an activity log of the events that occur during an ETL Load process. When designing the ETL Logging strategy, consider the following:

- Does the logging need to be very detailed or a standard one?
- Who will be accessing these logs
- Log Retention Policy

Retention Policy needs to be again decided as part of the ETL Logging strategy based on the ETL design. Having a longer retention policy helps you analyze ETL load performance overtime and understand any performance bottlenecks. Setting up an archive logs strategy can also be helpful if storing the logs for longer duration is not an option. If it is a standard log, then you need to add a proc print to statement as follows in the pre-code of the ETL Job. Append the sysdate and systime to the logname. Adding "new" at the end will ensure that a new log is created every time the job is re-run from SAS DI Studio.

```
proc printto log="/saswrk/&Project./log/&etls_jobname._&sysdate._&systime..log" new;
```

CONCLUSION

There is a lot of good information available online for SAS BASE programming, but there is only limited information available for SAS DI Studio. The main purpose of this paper is to specifically help ETL developers with some of the best practices using SAS Data Integration Studio who have limited or no prior experience with SAS Base Programming and can still use SAS DI studio to implement their ETL solutions.

REFERENCES

Svein Erik Vrålstad, Knowit Decision Oslo (2013) "Need for Speed - Boost Performance in Data Processing with SAS/Access® Interface to Oracle" *Proceedings of the SAS Global Forum 2013 Conference, San Francisco*. Available at <http://support.sas.com/resources/papers/proceedings13/081-2013.pdf>

CONTACT INFORMATION

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

Sai S Potluri, ETL Architect

Synectics for Management decisions Inc

E-mail: sai.s.potluri@gmail.com

Ananth Numburi, Lead ETL Programmer

Synectics for Management decisions Inc

E-mail: ananth.numburi@gmail.com