SESUG Paper 144-2020

Joining SAS® and a Complex, Sparsely Populated Database for Machine Learning/Data Mining

Thomas E. Billings, MUFG Union Bank, N.A., San Francisco, California

ABSTRACT

The challenge: join a SAS dataset with as many tables as possible from a large, complex, sparsely-populated star schema database. The joined data are to be used for data mining/machine learning analyses. We start by analyzing the database for feasible joins using the supplied synthetic keys, and work with the client to reduce/clarify the scope. We develop a cross-reference table that links the SAS dataset ID variables with numerous database synthetic keys. We then do inner joins for the database dim + fact tables and pre-process other database tables, after which the database extracts are merged with the SAS dataset. The processing involves PROC SQL, DATA step MERGE, with some metadata-driven processing. Some of the joins are 1-many; additional processing is done to handle these cases. Data are run for a few months and the variable population rates are determined and used to identify which variables to keep. Finally the programs are productionized and history is run.

Keywords: MERGE, PROC SQL, star schema, missing, non-missing, sparse, macro
SAS products: Base SAS.
User level: intermediate+.

THE SITUATION

A SAS® dataset was previously created for use in modeling. There is a separate database that is closely related in terms of content, and the SAS dataset does contain some information from the database - but very little. The request is to add as much data as possible from the database to the SAS dataset, for use in a data mining/machine learning exercise. This will require joining the SAS dataset to database tables that were not previously joined. The underlying objective is to determine if the database contains additional variables that are useful in modeling applications, and thereby to develop better models.

The database is a very complex, sparsely populated, star schema Oracle system. The data it contains, and the SAS dataset, both primarily come from the same underlying source system (but from different tables). However, because the SAS dataset and database use different types of records, and the use of a star schema, the “natural” join is broken between the 2 systems, forcing the use of alternative joins.

Herein we describe the processing required to create a large SAS dataset - over 2K variables - by combining the SAS dataset and multiple Oracle database tables. Much of the discussion here is at a conceptual level. However we do show some potentially useful code as well. We begin with refining the requirements and clarifying the scope.

SCOPE AND REQUIREMENTS

The star schema database is very large, so the first task is to clarify/manage the scope. The tables were reviewed and classified as:

1. Utility/internal database admin tables
2. Staging tables
3. Lookup/support tables (i.e., for recodes)
4. Dimension tables
5. Fact tables.

The primary tables of interest here are the dim (dimension) and fact tables; the other tables (#1-#3 above) are not relevant to the exercise. The dim and fact tables were reviewed to identify associations. Most associations/joins were of (1 dim + 1 fact) table, but there were a few cases of (2 dim + 1 associated fact) table. The tables were then reviewed for content and a description of the content – just a few words – along with a table list were supplied to the client, to be reviewed for relevancy. A few tables were identified by the client as not of interest/not relevant, reducing the scope of the effort.

Granularity. The star schema contains many types of data, and some of these are in 1-many relationships with the SAS dataset rows. The customer was advised of this issue and that a simple unduplication would be used to avoid multiple rows produced by the 1-many in joins, and that if a parameter that is 1-many turns out to be important, it can be captured in arrays in a later phase of this effort. There are too many variables that are 1-many to capture them in arrays before determining if they are useful in models/populated.

ANALYIS OF POSSIBLE JOINS

Create cross-reference table with correct row set/cardinality. Compared to other datasets extracted from the same source, the "natural" or optimal join does not work for the SAS dataset and Oracle database. This is due to the use of different record types (SAS dataset vs. Oracle) and the (daily) snapshot/data fragmentation that is a structural feature of a star schema (which impacts the ID variables used for natural joins). There is one dim + fact combination in the Oracle database that supports the 2nd best join, i.e., the alternate join, and it has the same granularity as the SAS dataset. We start with this join, which we refer to as DF0, the base Oracle dataset join.

The DF0: dim + fact join is an inner join, for a target date (end of month, similar to the SAS dataset), and with other relevant filter criteria. The fact table in DF0 has synthetic key variables for over 20 other database tables. The next step is to perform the alternate join between DF0 and the SAS dataset to create a cross-reference table:

SAS dataset ID variables (only) + Oracle database keys from DF0.

This cross-reference table is the master/base row set that we use to join with additional database dim+fact (DF) tables. The steps so far are illustrated in figure 1.

Dim+fact inner joins across the database. During the scope evaluation process, the various dim+fact joins in the database were listed, and the key variable(s) that defined the inner join were identified. Next, the dim+fact pairs are classified into 2 groups, based on the synthetic key used for the dim+fact join:

1. Synthetic join key is in the cross-reference table, i.e., is in the fact table DF0, or
2. Synthetic join key is <u>not</u> in the cross-reference table.

Regarding #1 above, it should be noted that while a specific synthetic key may appear in the cross-reference, a few of the synthetic keys are very poorly populated, i.e., are mostly missing across the xref table rows. This is caused by the sparse nature of the underlying raw data.

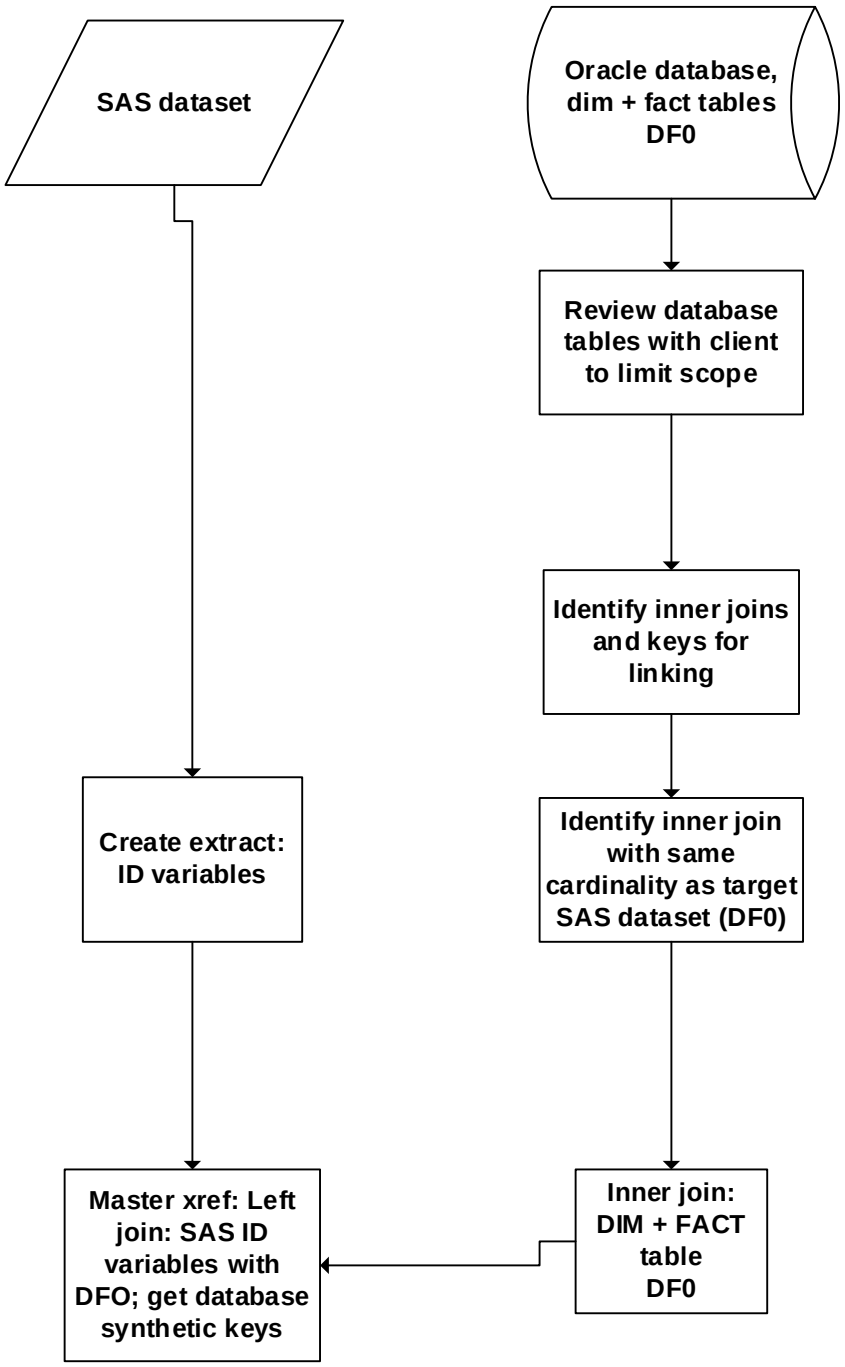

Figure 1: Refine scope, create xref table for database linkage

Regarding #2, we needed to find a key from the other keys available in the cross-reference, and use that for the join. The approach was to choose an initial key based on content, i.e., a key that is well-populated and (in theory) "should" work well. However this did not always succeed, i.e., the join rate for certain keys was very low – and when that happened, other alternate keys were tested to yield a better join rate (this was an iterative process).

Other database tables. The database had a number of tables that were not dim+fact. These were joined using keys present in DF0 whenever possible. Again, the choice of key to join by was problematic for some tables, requiring iterative testing of alternates.

THE JOINS: SQL INNER JOIN AND DATA STEP MERGE

Separate macros were developed to:

- Create data sets via inner join of specified dim+fact tables (PROC SQL)
- Create extracts from other tables (PROC SQL)

Another macro was written that performed sequential, cumulative merges (DATA step MERGE). MERGE was used instead of SQL as the datasets involved are large and while SELECT A.* could be used for one of the files, the other file would need to enumerate the selected variables, leaving out redundant variables (to avoid warnings/issues with duplicate variable names across the files), and DATA step MERGE handles that automatically.

The preceding point deserves clarification: in a DATA step MERGE, consider:

```
MERGE file_A file_B:
BY relevant_variables_here;
```

and file_A and file_B both have variable X. When rows match and are merged, the value of X will be populated from file_B, and not from file_A. (This assumes that variable X is the same type in both files; if not then an error or warning message will be generated).

The database files and the SAS dataset all contain the ID variables that are very important in the SAS dataset. However due to the star schema and record type mismatch, the ID variables can be different in each of the files. It is important to preserve the ID variables as-is in the SAS dataset, hence the use of MERGE.

FURTHER DEVELOPMENT AND TESTING

Once the 2 join macros were developed, they were tested using 1 month of data. As previously mentioned, iterative development was required for dim+fact joins whose key was not in the xref/DF0 table; and also for the supplementary joins. In those cases the associated fact tables usually had several keys that were in the xref/DF0 table. However the optimal key was not always clear, and multiple test runs were needed (in some cases) to get a good join. Once we had a good 1-month run, a macro was developed to encapsulate the 1-month code and to run it for multiple months. This produced a huge omnibus file containing over 2K variables.

Sparseness. The macro was then run for 7 months, producing 7 month-end files. These were concatenated and the variables analyzed to determine % non-missing over the 7-month period. A derivative SAS dataset was created that contained the variable names, plus the % non-missing rate for each variable.

There are many ways in SAS to determine the missing/non-missing rates of a variable. What we wanted here is a dataset with variables: variable_name and percent_missing, a single variable per row. For

numeric variables, PROC SUMMARY paired with PROC TRANSPOSE is an easy way to do this. For character variables, inspired by a SAS blog posting, we tried to use SAS/IML. We immediately ran into a memory issue: SAS/IML wants to store all of the data in memory. This was overcome by creating an analogue dataset with just the character variables, truncated to length 1 (after a LEFT operation). Because of the memory constraint, we recommend against using SAS/IML for this application, and since this effort have found a much easier way of getting the non-missing percentages for character variables (i.e., a method that is not memory bound). Another constraint on SAS/IML: very few sites have the product installed.

Code was developed to ingest the list of variables with their non-missing percentages, and create lists of the variable names for a KEEP statement. The code copies the variable name to the list if the % non-missing is greater than a threshold value (set via macro variable). This code was tested thoroughly and then inserted into the monthly macro, so that poorly-populated variables are excluded from the final file.

The final main macro was then rerun for the target time period (present date back to 2012), creating a file for each month-end. The modelling team used the file with machine learning processes, to investigate possible new models. Finally, the additional variables identified as useful for models are to be added to a new, expanded version of the SAS dataset.

This processing is illustrated in figure 2.

DISCUSSION

Stabilize the 1-many joins (for unduplication). Consider the situation: we have a SAS dataset that is the left side of a left join with a file created by an inner join of a dim + fact, and it is a 1-many relationship; some SAS rows will join with multiple database rows. Now recall that in databases, order is not guaranteed unless there is an ORDER BY that forces a unique ordering. We do not have that condition here and as a result, a naïve PROC SORT NODUPKEY may produce different results on multiple runs – by being fed rows in a different order by the database. The preceding would make the runs non-reproducible, and that is a problem.

To stabilize the join results, 2 sorts are used: a regular PROC SORT with the target key variable plus an additional variable that induces uniqueness for the row set, then a PROC SORT NODUPKEY by the target variable. This makes the results reproducible when the joined row set (with multiples) has not changed.

Fix non-standard missing values for character strings. The database used '-'as a missing value for character variables. This is easy to fix with the SAS code:

```
%* recode - signs to missing (character variables only);
data WORK.clean_join;
       set WORK.target_inner_join;
       array ckm{*} _character_;

       do i = 1 to dim(ckm);
              if ckm{i} = "-" then
                     call missing(ckm{i});
       end;

       drop i;
run;
```

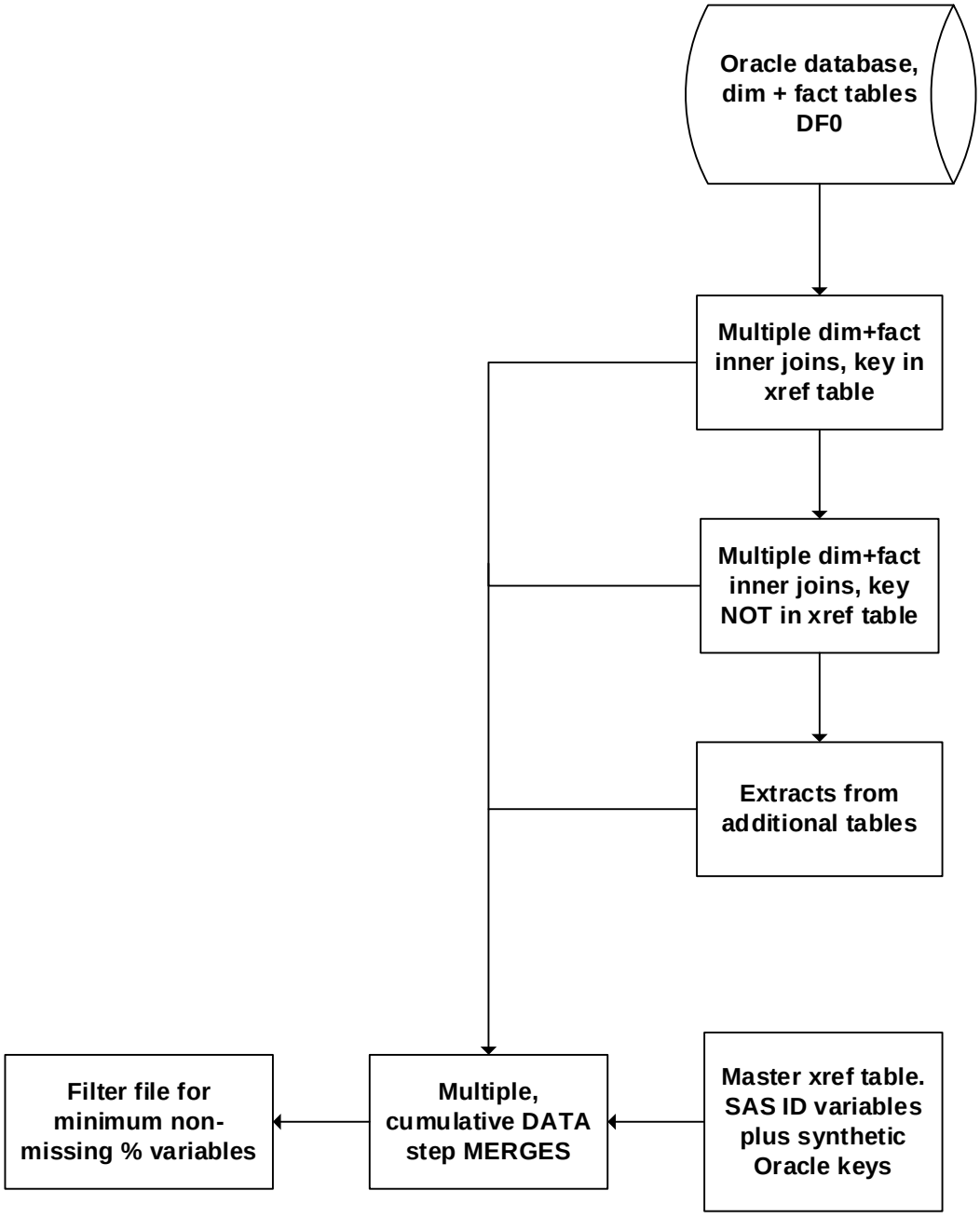

Figure 2. Monthly processing for omnibus file

Filter inner joins, limit to rows that join with the SAS dataset. Inner joins of a dim + fact table can produce many more rows than will join with the xref table. Technically, it is not necessary to limit the rows in the inner join because rows that will not join will be ignored in the left join with the xref table. However, as some initial joins were 1-many, it was helpful (as a first step) to see just the rows that joined with the xref table, in mitigating the problem.

Database inner joins: a metadata approach. The dim and fact tables include many variables with the same name, as well as variables that are irrelevant to the modeling process. When a variable has the same name in both tables, here we want to SELECT the variables from the fact table (vs. dim table) as the fact table (daily snapshot) data are "more current" than the dim table.

A metadata approach was used here: start with PROC CONTENTS on both tables, with OUT= output of a metadata file, plus the NOPRINT option. The metadata files were then merged, so there was 1 row per variable. These metadata files were then combined via MERGE and processed to produce a SAS dataset with variable names and table indicator. Here is the basic code, with annotations in comments:

```
proc contents data=database.&dim1 out=dcont1 noprint;
run;   * dim table metadata;

proc sort data=dcont1;
       by name;
run;

proc contents data= database.&fact1 out=fcont1 noprint;
run;   * fact table metadata;

proc sort data=fcont1;
       by name;
run;

** merge dim, fact metadata, split into dim-only, fact-only, and common sets;
data dimvars commonvars factvars;
       merge dcont1 (in=indim1) fcont1 (in=infact1);
       by name;

       if (upcase(name) IN (list vars to be deleted here)) then
              delete;

       if (indim1) and (infact1) then
              output commonvars;
       else if (indim1)  and (not infact1) then
              output dimvars;
       else if (not indim1) and (infact1) then
              output factvars;
       rename length=clength;
run;

** set indicator for preferred variable table source;
data varlist0;
       set dimvars (in=in1) commonvars factvars;

       if (in1) then
              t1=1;
       else t1=2;
run;

** sort the variable list;
proc sort data=varlist0 out=varlist (keep=memname name varnum clength type t1);
       by memname varnum;
run;
```

The SAS file varlist was then used in a DATA step to construct macro variables that contain the list of variables for the inner join SQL SELECT statement.

Spiders. It may be helpful to note that the overall processing here is similar to a spider process across the database tables. The path is:

SAS dataset → xref table → database synthetic keys → inner joins and other database tables

SUMMARY

A large, complex, sparsely populated star schema database can be joined with a SAS dataset for use in data mining/machine learning. The basic steps are:

1. Analyze database; figure out how to join with SAS dataset:
   - Identify optimal dim + fact inner join for linking with SAS dataset (i.e., same granularity/row set).
   - Review other dim + fact inner joins with client for availability of link keys, also relevancy; reduce scope of effort if possible.
   - Review other relevant, needed tables for likely linkages.

2. Create a master cross-reference table, (SAS dataset X optimal dim + fact inner join). This table maps the SAS dataset ID variables to many synthetic database keys.

3. Create database extracts and join the data:
   - Do multiple dim + fact inner joins across the database. Filter to rows that will join with the master xref table.
   - Pre-process other database tables for join with SAS dataset.
   - Multiple sequential, cumulative DATA step MERGEs (or SQL if you prefer) to join the database files with the master xref table.

4. Check variable population rates:
   - Run n months, concatenate files and analyze variables for % population rates.
   - Create DROP or KEEP statements to limit files to variables with acceptable population rates

5. Finalize the results:
   - Productionize code/programs
   - Client review of sample files (UAT: user acceptance testing)
   - Change management documentation (if required)
   - Run all of history

Note: As of the date of publication, the author is an employee of MUFG Union Bank, N.A. The content of this paper does not reflect the views or opinions or work product of MUFG Union Bank.

APPENDIX 1:
BSD 2-CLAUSE COPYRIGHT LICENSE (OPEN SOURCE)

```
* All program code in this paper is released under a Berkeley Systems
  Distribution BSD-2-Clause license, an open-source license that permits
  free reuse and republication under conditions;
```

CONTACT INFORMATION

A list of the author's SAS-related papers, including URLs for free access, is available at the URL (hosted by Google Drive): https://goo.gl/uCUHoa

Note: Your enterprise web filter might prevent access to this URL from work, in which case you will need to access via a personal device.

Thomas E. Billings
Email: tebillings@gmail.com