MWSUG 2016 - Paper RF-03

A Quick Macro to Replace Missing Values with NULL for Numeric Fields within a CSV File

John Schmitz, Luminare Data LLC, Omaha, NE

ABSTRACT

When large file uploads to RDBMS systems are required, csv file imports can provide a faster alternative than using SAS to insert the records. However, some RMDBS systems will convert null csv columns to 0, creating a potentially undesirable result. This macro using the original dataset and resulting csv file to identify and replace missing values with NULL for all numeric columns in the csv output.

INTRODUCTION

SAS® provides many great resources for data management making it a natural choice for the primary ETL system in many companies. Yet a world of big data sets and multiple disparate systems pushes all data managers to find better methods to improve data processing. One such option is to leverage the native RDBMS utilities to load data rather than utilizing SAS based processes. This can be especially beneficial when bulk load capabilities may not be supported for the target database system.

However, successfully implementing this process can require added steps to address differences in how SAS and these utilities may handle data. In this paper, we will address a quick solution to one such potential problem; the handling of missing values.

THE CHALLENGE

The general warehouse project involved reading data from multiple sources, performing various ETL processes in SAS, then loading the resulting data into a RDBMS system on a separate server. Due to the volume of change records involved, load times using PROC APPEND were having a detrimental impacting the overall process. As an alternative, a process was developed to leverage the database's csv file load capabilities rather than running the SAS-based append process. The core steps to support this design include:

- Execute a PROC EXPORT to output the new records as a csv file.
- Execute a PowerShell script to trigger the native file import of the csv file created above.
- Validate that all records were successfully inserted as required.

THE PROBLEM

The steps above did provide a notable speed improvement for the table load process. However, the file importer loaded 0 for NULL values found in numeric fields. The solution was to replace any null values in the csv file with NULL. In the csv, the occurrence of ",," would represent a null value and would need to be replaced by ",NULL,". However, this would only be done for numeric columns. A similar replace in a character column would result in "NULL" being loaded as the character field value.

THE SOLUTION

A macro was introduced into the process to address the null value issue. That macro (%csv_null_replacement) is:

```
%macro CSV_NULL_Replacement (filename=, source_table=, debug=0);

   ** VALIDATE FILE EXISTS -- IF NOT THEN FLAG AND EXIT **;
   filename csv "&filename.";
   %if %sysfunc(fileexist("&filename.")) = 0 %then %return;
```

```
** DETERMINE SIZE AND LAYOUT OF CSV FILE FROM SOURCE TABLE **;
proc contents data=&source_table. out=varlist order=varnum noprint;
run;

proc sql noprint;
      ** LIST FIELD TYPES IN ORDER OF APPERANCE IN TABLE **;
      select type
            into :Vartypes
            separated by ' '
      from varlist
      order by varnum;

      ** FORMULA TO COMPUTE LRECL FOR FILE **;
      select
            sum(max(formatl,length(name),5)) +
            count (formatl) +
            10
            into :lrecl
      from varlist;
quit;

** BUILD TEXT STRING WITH CORRECTED LAYOUT **;
data fix;
      length str str2  $&lrecl
            %if &debug=1 %then  str_o $&lrecl flags flag2 $20  ;;
      infile csv lrecl=&lrecl pad ;
      input str $ 1-&lrecl;

      ** LOGIC TO BALANCE ANY UNBALANCED QUOTES **;
      %if &debug=1 %then str_o = str;;
      str = tranwrd(str,"'","''");

      ** COPY FROM STR TO STR2 WITH REQUIRED ADJUSTMENTS **;
      str2 = '';
      %if &debug = 1 %then %do;
            flags = '';
            flag2 = '';
      %end;

      do i = 1 to countw("&vartypes");
            if scan("&vartypes",i) = '1' and scan(str,i,',','mq') = ''
            then do;
                  str2 = catx(',',str2,'NULL');
                  %if &debug=1 %then
                        flags = catx('~',flags,put(i,3.));;
            end;
            else if scan(str,i,',','mq') = '' then do;
                  str2 = trim(str2) || ',';
                  %if &debug=1 %then
                        flag2 = catx('~',flags,put(i,3.));;
            end;
            else
                  str2 = catx(',',str2,scan(str,i,',','mq'));
      end;

      ** LOGIC TO REMOVE THE UNBALANCED QUOTE ADJUSTMENT ABOVE **;
      str2 = tranwrd(str2,"''","'");
```

```
            %if &debug=0 %then keep str2;;
      run;

      ** DROP ORIGINAL CSV FILE **;
      %let rc = %sysfunc(fdelete(csv));

      ** WRITE REVISED CSV FILE **;
      data _null_;
            set fix;
            file csv lrecl=&lrecl pad;
            put str2 $ 1-&lrecl;
      run;

      ** CLEAN UP DATASETS **;
      %if &debug=0 %then %do;
            proc datasets nolist nowarn;
                  delete fix varlist;
            run;
            quit;
      %end;
  %mend CSV_NULL_Replacement;
```

SECTION 1 – FILE VALIDATION

The first section of the code completes a quick file validation. This process verifies that the file supplied by the macro parameter &filename does exists and will trigger an exit from the macro if the file is not found.

SECTION 2 – IDENTIFICATION OF NUMERIC AND CHARACTER COLUMNS

The second section uses the original data set (&source_table) and PROC CONTENTS to determine column order and type. This will allow the macro to later classify cases of ",," as character or numeric. The process also captures length information to compute the required string length and LRECL for the DATA step process to follow, allowing the code to match string lengths with the requirements to complete the task. This section includes the PROC CONTENT and PROC SQL statement that follows.

SECTION 3 – DATA STEP TO ALTER THE CSV FILE STRING

This is the core process of the macro. Each record from the csv file is read as a long string into DATA FIX. The process uses the SCAN function to count by fields across the string. This count in conjunction with the &vartype macro variable are used to identify whether a column is numeric or character. If the word returned by scan is "" and the field type is numeric, then the word is replaced by "NULL". Otherwise the word is carried over as defined in the original csv file.

There are several exceptions to the simple process flow that must be addressed in the code for it to work correctly.

- Multiple delimiters with null/blank columns.
- Commas that may appear within quoted text (such as an address).
- Quotes in text, especially unbalanced quotes (such as O’Malley).
- Special case when last field is a blank character.

The SCAN function requires the option parameter ‘m’ to accurately count column position when multiple columns appear together having null or bank values and option ‘q’ to not count delimiters within quoted strings. This will correct most of the delimiter-based counting issues. It does, however, introduce the third exception – unbalanced quotes. Since the PROC EXPORT process uses double quotes to quote text strings when required, and most single quote exceptions occur as a single quote, the code replaces all cases of a single quote " ' " with two single quotes " '' " prior to running the scan logic then removes the second quote before writing out the final string. Finally, if the last field is a blank character, the string should end with a ",". The logic appends a comma with blank strings to insure the last column is passed correctly.

SECTION 4 CLEAN UP / DEBUG OPTION

The code completes by deleting the original csv file and replacing it by the altered version and deleting the work table it created. In several places throughout the code, there are optional steps that trigger when DEBUG is set to 1. These statements were included during design and testing for validation. In production, the code should run with DEBUG=0. However, the user is welcome to switch to DEBUG mode (debug=1) which will generate and retain additional information from the process. This information is helpful to decipher what the macro process does in more detail.

CONCLUSION

The macro above provides an easy process to address the error introduced by using the database table load utility. One macro call was introduced, following the PROC EXPORT step to modify the csv file so that NULL numeric fields were inserted as NULL values. The macro requires two parameter inputs, filename (the path used when creating the csv file) and source_table (the name of the dataset used to create the csv file). No additional changes to the process were necessary to add the feature and correct the error.

CONTACT INFORMATION

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

John Schmitz
Sr. Manager, Luminare Data LLC
john.schmitz@luminaredata.com