PhUSE 2015

Paper CC06

SQL by Example: A Comparative Introduction to PROC SQL

Jeanina (Nina) Worden, Santen, Emeryville, USA

ABSTRACT

There once was a programmer who felt pretty comfortable using DATA steps to complete their dataset tasks. Then, at a conference, they learned how programs, very similar to the ones they had been writing (SORT, SORT, SORT, DATA step merge), could utilize PROC SQL to reduce the size of their programs and, in some cases, gain efficiencies. The goal of this paper is not to argue whether or not PROC SQL is superior to the DATA step. Instead it is to provide an introduction to PROC SQL using comparisons between common DATA step tasks with the hope of providing other programmers exposure to an additional tool that can be used for their data programming. This paper uses BASE SAS® and is for any level of programmer interesting in getting started with PROC SQL.

INTRODUCTION

In the world of SAS®, most programmers come quickly to the realization any given task can be accomplished by more than one method. This is never more apparent than when attending a programming conference and seeing all the ways other users are completing similar tasks. They sometimes learn how programs, very similar to the ones they had been writing (SORT, SORT, SORT, DATA step merge), could utilize a different tool to reduce the size of their programs and, in some cases, gain efficiencies. While opinions may very on the "right" or "best" way to do things, the goal of this paper is not to argue whether or not PROC SQL is superior to the DATA step, or another PROC. Instead it is to provide an introduction to PROC SQL using comparisons between common DATA step tasks with the hope of providing other programmers exposure to an additional technique that can be used for their data programming. But the understanding of the use of PROC SQL has to start with an understanding of the basic syntax of the statement in relation to that of the DATA step structure. A good example is using the syntax for setting a dataset:

```
data <new dsn>;
set <original dsn>
run;
```

The DATA line identifies the new dataset name; we then identify the source dataset with the SET statement before ending with a RUN. To do the same task with the basic syntax for PROC SQL is:

```
proc sql;
create table <new dsn> as
select *
from <original dsn>;
quit;
```

After the PROC SQL call, the new dataset is identified using CREATE TABLE followed by the SELECT line which allows selection of the variables to be included. After the SELECT statement, FROM is used in the basic statement to identify the original dataset before ending the statement with a QUIT. Note that in this example, to mimic the DATA step code, we want all variables and a quick way to do that is by using a star (*) on the SELECT line. If you only want a select number of variables, the specific variables can be identified using the following code on the SELECT line:

```
select <var1, var2…>
```

This acts the same as a using a KEEP statement when doing a basic SET. This paper will build on these basic code structures but the following datasets will be used to allow actual dateset and variable names to be used in the examples to provide more clarity of the concepts presented.

SITE

SITEID	NAME
001	John Smith
002	Sam Jones
002	Sam Jones

SUBJECT

SUBJID	SITE	VISIT
123	001	1
123	001	4
456	002	1
456	002	2

789	003	1

Examples for comparing DATA steps and PROC SQL tasks could fill an entire book; however, as this is simply an introduction to PROC SQL, this paper will focus on handling two specific programming tasks, dealing with data and merging. As the reader may note, there are some data issues present, like duplicate records and common variables with differing names, so the first examples will address dealing with these types of issues before moving on to some additional data manipulation examples. The final section will be concentrated on merging, both full and conditional merges. All the examples will utilize BASE SAS® and are intended for any level of programmer interesting in getting started with PROC SQL.

DEALING WITH DATA

Before combining data sources together we need to ensure the content of the source data is ready for use. This may require some data manipulation, or reduction, depending on how the data will be used. With that in mind the first examples will focus on dealing with the data by removing duplicates, renaming variables, and adding new variables where needed.

DE-DUP

Let's start by looking at the contents of the SUBJECTS dataset. Because some of the subjects have multiple visits there are multiple records with the same subject number. If we need only the unique subjects we must remove the duplicate records. This can be done by using a PROC SORT using the code:

```
proc sort data = site output = site2 NODUPKEY;
by siteid;
run;
```

The key created by the BY statement tells SAS® how to utilize the NODUPKEY option. In this example, records will be selected by keeping only the first SITEID record. The PROC SQL statement would add the DISTINCT qualifier to the SELECT line to identify variable used to determine uniqueness. The code would be:

```
proc sql;
create table site2 as
select DISTINCT siteid, name
from site
quit;
```

Only the unique, or DISTINCT, site IDs will be output to the new dataset SITE2.

WHAT'S IN A NAME

Next, when comparing our two example datasets its worth noting one of the common variables for site identification is named differently. If a DATA step will be used for merging this will be problematic and therefore the variable will need to be renamed either as it comes in from the original dataset:

```
data subj;
set subject(rename=(site=siteid));
run;
```

…or as it is going out in the new dataset:

```
data subj(rename=(site=siteid));
set subject;
run;
```

Either example will result in SITE to be renamed to SITEID in the new dataset SUBJ. Merging common variables, though named differently, is not an issue when using PROC SQL (as will be discussed later in this paper), however renaming may still be needed for other reasons. Again the programmer has the flexibility to determine if the rename is done prior to the original dataset coming in or as part of the new dataset's structure. If changing the incoming dataset is preferred PROC SQL allows the same rename functionality as in the DATA step by modifying the FROM line to include the RENAME option, and the new variable name, SITEID, referenced on the SELECT line:

```
proc sql;
create table subj as
select subjid, SITEID, visit
from subject (rename=(site=siteid))
quit;
```

Alternatively the name of the variable can be changed when constructing the structure of the new dataset by simply modifying the basic SELECT statement for that variable. The old variable is specified with "as <new name>" added before the comma, as noted below:

```
proc sql;
create table subj as
select subjid, site as SITEID, visit
from subject
quit;
```

Again, either PROC SQL code results in the new dataset SUBJ containing a variable named SITEID instead of SITE.

NEW AND IMPROVED

This takes care of some data issues, but what if we need to add a new variable. For example, we could add a location depending on the site identifier. Within a DATA step IF/THEN/ELSE is used:

```
data subj;
set subject;
if site = "001" then loc = "MO";
       else if site = "002" then loc = "TX";
       else if site = "003" then loc = "CA";
       else loc = "99";
run;
```

The code identifies the condition the first variable has to meet then assigns the new variable the appropriate value. The PROC SQL statement uses the same concept but using a CASE statement:

```
proc sql;
create table subj as
select subjid, site, visit,
       case
         when site = "001" then "MO"
         when site = "002" then "TX"
         when site = "003" then "CA"
         else "99"
       end as loc
from subject
quit;
```

The CASE statement considers each WHEN line to determine if the condition is met and holds the assignment after the THEN until the END. In both cases, the final ELSE could replace the final ELSE/IF or WHEN statement as there are no other options than the ones listed. The inclusion of all options is dependent on programmer preference, and/or departmental guidelines, rather than a requirement. Either code would result in the following dataset:

SUBJ

SUBJID	SITE	VISIT	LOC
123	001	1	MO
123	001	4	MO
456	002	1	TX
456	002	2	TX
789	003	1	CA

FUN WITH FUNCTIONS

Similar to the RENAME, functions a programmer uses within a DATA step can also be utilized using PROC SQL, sometimes with less typing. For example, to create a dataset with the subjects and their maximum visit, a DATA step requires an initial sort to ensure the records are grouped appropriately. The DATA step can then be coded to create a new variable to hold the maximum value of the variable in question, but first you have to manipulate the structure:

```
proc sort data=subject;
by subjid site;
run;

proc transpose data=subject out=subj;
by subjid site;
var visit;
```

```
run;

data subj2;
set subj;
maxvisit=max(of col1-col2);
run;
```

The MAX function cannot be applied directly to the variable therefore the dataset must be sorted by the variables that will be maintained and then transposed for the VISIT variable. Once the individual columns are created the MAX function can be applied to create the new variable. With PROC SQL, the MAX function can be used on the variable itself to calculate the maximum value within the designated variable:

```
proc sql;
create table subj as
select distinct subjid, site, max(visit) as maxvisit
from subject
group by subjid;
quit;
```

Similar to the BY in the DATA step method, MAX is dependent on the use, or exclusion, of the GROUP BY statement. With the GROUP BY statement, as shown above, the maximum value is determined within the group, SUBJECT. The omission of this statement would still produce a variable named MAXVISIT, however would make the determination of its value from across all records in the dataset.

MERGING

Now programming tasks usually require data from more than one source, so now that the data in the individual datasets are in good shape the focus can shift to merging. There are several types of merges and all can be accomplished using either technique discussed here. In this section we will explore merges that combine all the data, only the records from each dataset that match, and finally to conditional merges. For each example we will be merging the two sample datasets using the site identifier as the linking variable. As discussed previously the variable holding this information is named differently within the two datasets. Before any type of merge can be programmed, if using a DATA step for this task, the variables must be renamed to be equal. The following SORT statements will be used throughout this section.

```
proc sort data = subject output=subj(rename=(site=siteid));
by site;
run;

proc sort data= site out=site1 nodupkey;
by siteid;
run;
```

In the above example, the NODUPKEY is also used on the SITE dataset to remove the duplicate records to create the SITE1 dataset. This dataset will also be used below in the PROC SQL statements to avoid merging with both records. Now that we have a consistently named variable, the new dataset can be used in the merging process. The next step is to determine the type of merge that will be performed.

EVERYONE WELCOME

In some cases all the data from both datasets are needed. For DATA step processing, after the variable naming is consistant, the two datasets can be merged to create a new, combined dataset containing all records from both dataset.

```
proc sort…
proc sort…

data ds_all;
merge subj site1;
by siteid;
run;
```

The resulting dataset, NEWSUBJ, will merge the record where the SITEID matches, but will also contain records from either dataset that do not meet the criteria. The order of the datasets on the MERGE line determine the layout of the combined dataset, with the variables from the first as the first set of variables loaded into the new dataset.

As noted above, merging the data sources using PROC SQL does not require renaming of the merging variables.

As long as the variables are the same type, the merge will use the original names from the two datasets for the merging criteria, it also does not require the datasets to be presorted prior to the merge code. Additionally, the datasets do not require the presorting as this is handled in the background before the merge. For SQL merging all records from each dataset is called a FULL join:

```
proc sql;
create table sql_all as
select subjid, site, visit, name
from subject as a FULL JOIN site1 as b
ON a.site = b.siteid;
quit;
```

This example uses the SITE1 dataset created above so the duplicate records in SITE have been resolved. If not, the merge would merge the SUBJECT records to each of the duplicated SITE=002. Unlike the DATA step example above, the original datasets are distinguished using “a” and “b” to identify which dataset the merging variable is coming from. This is also important if the datasets contain common variables that are not used in the merge. For instance, if both datasets contained a variable LOC, the incoming variable would need to be acknowledged on the SELECT line to identify which dataset to use to populate the content of the new dataset. One warning, when common variables are not used for the merge, PROC SQL will continue with the merge creating a Cartesian product based on the criteria given. In some cases this is the desired result while in others it is not. It is advisable to reference the PROC SQL documentation regarding Cartesian products to fully understand this. If this occurs, a SAS® warning will appear in the log and will need to be assessed by the programmer. Lastly, besides the addition of the dataset identifiers, another change from the basic syntax is that ON replaces WHERE when merging, or joining, datasets using PROC SQL.

MATCH MERGE

Though full joins can be useful for tasks like highlighting missing data, more often than not the resulting dataset needs to meet some conditional criteria, one type would be only where the records match. Assuming the data has already been sorted in both datasets, the DATA step code would be similar to the code above but changing the “or” to “=” on the IF line:

```
proc sort…
proc sort…

data ds_match;
merge subj(in=a) site1(in=b);
by siteid;
if a=b;
run;
```

Here, as with the PROC SQL statement above, each dataset is designated as “a” or “b” to control the merge by denoting the source dataset of the records. An IF statement is added to specify the records will be merged only if they match the merging criteria, the BY statement. SQL merging based on only matching records is done by replacing the FULL JOIN with a comma between the datasets on the FROM line and WHERE replaces the ON statement on the line designating the merging variables:

```
proc sql;
create table sql_match as
select subjid, site, visit, name
from subject as a, site1 as b
WHERE a.site = b.siteid;
order by site;
quit;
```

Additionally, this example also includes the use of the ORDER BY statement which has the same effect as a sort on the final dataset. This is not required but added for additional information.

ONE SIDED ARGUMENTS

The final type of conditional merge that will be discussed is a “one-sided” merge, where all records are included from one dataset but only those meeting the criteria are merged from the other. Again assuming both datasets have been sorted appropriately, a DATA step can be written to achieve the one sided merge by modifying the IF line with the designation of the dataset where all records should be included:

```
proc sort…
proc sort…
```

```
data ds_one;
merge subj(in=a) site1(in=b);
by siteid;
if a;
run;
```

In this example all records from SUBJ (designated as “a”) will be included and only records from SITE2 that match on the SITEID variable. As with the order of the datasets, the first dataset in the MERGE statement does not have to be the one being merged onto, it is based on the desired structure of the new dataset and designation preferences of the programmer.

The same result can be achieved by modifying the PROC SQL’s FROM line with a directional join statement. However, unlike the DATA step, in this type of merge the order on the FROM line does have significance. The “a” dataset is on the left therefore we specify the merge as a LEFT JOIN and replace the WHERE with the ON statement:

```
proc sql;
create table sql_one as
select subjid, siteid, visit, name
from subject as a LEFT join site1 as b
ON a.site = b.siteid
order by siteid;
quit;
```

ON is once again used in place of the WHERE statement in the previous example. This is a directional merge, so the same code can be used if the second dataset, SITE, was to be used as the primary dataset by using RIGHT on the FROM line or reversing the order.

ONE LAST NOTE

The datasets being used are admittedly simple with only a single merge variable; however the examples given throughout the paper can be modified to include multiple variables as needed. Just as with the DATA step, when using PROC SQL the additional variables are simply added to the WHERE/ON line:

```
proc sql;
create table <NEWDSN> as
select a.*, b.*
from <DSN1> as a, <DSN2> as b
where a.<VAR1> = b.<VAR1>
      a.<VAR2> = b.<VAR2>;
quit;
```

The WHERE line can further be expanded to include as many variables as needed to achieve the merging criteria desired.

CONCLUSION

The intent of this paper was to present an introductory look at a comparison of common data modification and merging taks performed by programmers. Either method can be used to achieve the same result so the information herein was to allow programmers a potential alternative for completing their tasks. Building off of the basic syntax, the discussion started with dealing with data issues within the original datasets. Examples were given to remove duplicate records, how to rename variables, creating conditionally based assignments for new variables and working with functions. Once the data was ready examples were given to compare merging techniques between the two methods. Different merging scenarios were demonstrated starting with merges where all records from each dataset were retained, though merging where possible. Then examples of “match merging” were given to demonstrate when only matching records are kept in the new dataset. Finally, “one sided” merges were presented so all records from one dataset were merged with only the matching records from the other. The examples given can be expanded much further but are beyond the scope of this introductory paper. If further information is desired, a good reference is the PROC SQL guidance/documentation provided by SAS®, other conference papers or any of the referenced papers below.

REFERENCES

SAS® 9.3 SQL Procedure User's SAS® 9.3 SQL® Procedure User’s Guide. Available at URL: https://support.sas.com/documentation/cdl/en/sqlproc/63043/PDF/default/sqlproc.pdf

Timothy J Harrington, 2001. “An Introduction to SAS® PROC SQL” Proceedings of the SUGI 26 Conference.

Gajanan Bhat, 2002. “Merging Tables in DATA Step vs. PROC SQL: Convenience and Efficiency Issues”. Proceedings of the SUGI 27 Conference.

Timothy J Harrington, 2000. “Merges and Joins”. Proceedings of the SUGI 25 Conference.

CONTACT INFORMATION

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

Jeanina (Nina) Worden
Santen Inc.
2100 Powell Street, 15th Floor
Emeryville / 94608
Work Phone: (425) 268-9095
Email: jworden@santeninc.com