Paper CT-20

Learning PROC SQL the DATA Step Way

Meghal Parikh, University of Central Florida, Orlando, FL
Elayne Reiss, University of Central Florida, Orlando, FL

ABSTRACT

As evidenced by some dissenting opinions within our own office, the use of PROC SQL for dataset manipulations may be considered dreadful by those who learned their way around Base SAS® with only the DATA step. For many tasks, there are few better routes to successful completion than a well-designed DATA step. However, PROC SQL can serve as an efficient replacement to many tasks. This paper illustrates where replacing DATA step code with PROC SQL-based code might be a smart decision for any Base SAS programmer. Additionally, it draws simple analogies to DATA step syntax, reducing the possible intimidation associated with learning the complexities of PROC SQL. With knowledge of both techniques, programmers will always be able to select the best technique for each data manipulation scenario.

INTRODUCTION

As many SAS users have previously suggested, PROC SQL can sometimes be rather intimidating for those who have learned data manipulation techniques solely through the use of DATA step programming. In our office, the primary analysts are relatively new users to SAS and enjoy exploring diverse SAS data management techniques. However, when working in an environment led by some seasoned SAS users who happen to be set in the ways of the DATA step, analysts who code in a diverse fashion must have very specific reasons for coding with PROC SQL or risk having to recode their work. With PROC SQL, as is the case with many SAS procedures, it is always possible to accomplish more tasks by increasing the complexity of the syntax.

Creating harmony between these two schools of thought is not an impossible feat. When PROC SQL syntax reads similarly to DATA step syntax as well as the code of other SAS procedures, it can be easy for other SAS programmers in your team to understand your PROC SQL code. Conversely, if you are a hybrid user of Base SAS and SAS Enterprise Guide, understanding complex PROC SQL code becomes gradually easier, since PROC SQL is the underlying data manipulation procedure used by SAS Enterprise Guide 4.3.

The main goal for this paper is to find ways of coding that can be similar in both DATA step and PROC SQL. The following list outlines some of the basic underlying terminology differences between the DATA step and PROC SQL code:

- observations in DATA step are equivalent to “rows” in a SQL table
- variables in the DATA step are equivalent to “columns” in the SQL table
- **set** or **merge** statements in DATA step are equivalent to the **from** statement in SQL code
- the **select** statement syntax in SQL code can be used as a substitute for the **keep**, **drop,** or **rename** statements in DATA step

This paper will explore the equivalencies of these statements between DATA step and PROC SQL, providing examples along the way to aid in understanding and potential adaptation into any type of coding application.

BASIC SYNTAX

Before addressing the comparisons associated with more complex code, it is important to first establish basic similarities and differences in the simplest code.

First, consider these few lines of code that almost any DATA step programmer likely has committed to memory:

```
/**DATA Step Syntax**/
data work.student_data_file;
    set project.student_data_file;
run;
```

After the comment line, you’ll notice an invocation statement that tells SAS that the DATA step is to be used to create a table, student_data_file in the temporary work library. The **set** statement provides SAS with the source dataset, student_data_file in the library called “project.” Finally, the statement is executed with the use of the **run** statement.

Now examine another few lines of code:

```
/**PROC SQL Syntax**/
proc sql;
    create table work.student_data_file as
    select *
    from project.student_data_file;
quit;
```

This PROC SQL statement achieves the identical result as the DATA step code previously displayed. Again, the first line after the comment provides an invocation of the procedure to SAS. **Create table** provides the destination table name and library location. The **from** statement takes the place of the DATA step's **set** statement. Finally, the invocation ends with **quit** rather than **run.**

Some functional differences exist, however. Unlike the DATA step's **set** statement, PROC SQL's **from** statement cannot append two similar data sets separated in code by a space. Additionally, the DATA step will bring all of the variables from the input data set into the output data set by default with no additional statements or options. PROC SQL, on the other hand, requires the use of **select** followed by an asterisk, as shown in the above PROC SQL code, in order to output all of the input dataset variables into the output dataset.

APPENDING DATASETS

The **set** statement is the simplest form of allocating and appending data sets in the DATA step. PROC SQL uses the **from** statement to bring records from an input dataset to the output dataset. However, if two or more datasets with overlapping or identical layouts need to be appended, **from** cannot be used in the same fashion as the DATA step's **set** statement. In this case, the **outer union** statement in PROC SQL needs to be used. The following example demonstrates how this statement looks in PROC SQL:

```
proc sql;
    create table work.student_data_all as
    select *
           from work.student_freshmen
    outer union
    select *
           from work.student_seniors;
quit;
```

Outer union is PROC SQL's method of simply concatenating two datasets (adding records, not merging on a variable). This statement does not necessarily have the same intuitive syntax for a longtime DATA step programmer, but using **outer union** does the same job. However, because it is combined with a **select** statement, using this approach may be more intuitive when you want to concatenate another dataset but only want to select a few variables from your dataset for the addition of extra observations. When using DATA step, you would need to utilize an extra **keep** statement within your **set** declaration, whereas in the case of PROC SQL the **select** statement is always ready and waiting for more specific variable selection.

THE SELECT STATEMENT SUBSTITUTES

The select statement syntax in PROC SQL can be one of the most intimidating features for new PROC SQL users. Unlike in DATA step, which by default assumes that the user would like to keep all variables in the dataset unless otherwise specified, PROC SQL requires the user to specify in the **select** statement the desired output variables (columns) to retain from the input datasets. Using the asterisk operator after **select** allows the coder to keep all the columns in the input dataset. An easier way for a DATA STEP traditionalist to learn PROC SQL is to use the data set options in the **from** statement. This method provides the flexibility of intuitive syntax and shorter, more organized code of PROC SQL. The following section will explore the use of these PROC SQL options.

KEEP, DROP AND RENAME VARIABLES (COLUMNS) IN PROC SQL

Most SAS coders have found themselves in the situation of having an input dataset with many variables and wanting to create an output dataset that features all of the same variables…except for one. In such scenarios, many coders would avoid using PROC SQL in fear of having to explicitly declare so many variable names in the **select** statement. However, by using the '*drop=*' option with the **from** statement, you only have to specify the variables that you don't want to retain. Similarly, the '***keep=***' option is a better choice over listing many individual variables in the **select** statement to specify which variables you want to bring into the output dataset.

Datasets for Examples

The datasets as displayed in Figure 1 and Figure 2 will serve as examples for data manipulations throughout the rest of this paper.

student_id	Term	Gender	Highest_Degree	Academic_Level	Acad_Plan
[redacted]049	201005	M	Masters	Doctoral Candidacy	Modeling & Simulation-ENGR PhD
[redacted]141	201005	F	Masters	Doctoral	Industrial Engineering PhD
[redacted]499	201005	M	Baccalaureate	Doctoral Candidacy	Computer Science PhD
[redacted]553	201005	M	Masters	Doctoral	Educational Leadership EdD
[redacted]641	201005	F	Masters	Doctoral Candidacy	Nursing PhD
[redacted]779	201005	M	Baccalaureate	Masters	Criminal Justice MS
[redacted]992	201005	F	Masters	Doctoral	Education EdD
[redacted]160	201005	M	Masters	Masters	Hlth Care Informatics-PSM MS
[redacted]680	201005	M	Masters	Doctoral	Sociology PhD
[redacted]782	201005	F	Masters	Masters	Business Admin MBA

Figure 1. Dataset, student_data

Term	student_id	Course_Prefix	Course_Number	Course_Section	Section_Credits
201005	[redacted]049	CAP	7980	C004	3.0
201005	[redacted]141	ESI	5219	CR01	3.0
201005	[redacted]499	COT	7980	C001	3.0
201005	[redacted]553	EDH	6105	D001	3.0
201005	[redacted]641	NGR	7980	C033	3.0
201005	[redacted]779	CCJ	5456	CW61	3.0
201005	[redacted]992	EDG	7221	C001	3.0
201005	[redacted]992	IDS	7938	CM02	3.0
201005	[redacted]160	HIM	6117	CW59	4.0
201005	[redacted]160	HIM	6938	CW59	2.0

Figure 2: Dataset, student_course_data

Example

As an example of how dataset options can be used with PROC SQL to make the syntax more intuitive for those who usually write code with DATA step, we will perform two coding tasks. First, we will rename the “student_id” column as “emplid” in the output dataset (table) student_data. Secondly, we will drop the “academic_level” column.

First, for reference, the following code outlines these activities using DATA step. The “student_id” column is renamed as of the output data set, while “academic_level” is dropped as the dataset is entered.

```
/**DATA Step Syntax**/
data work.student_data_new (rename=(student_id=emplid));
    set work.student_data (drop=academic_level);
run;
```

Next, we provide an example of this same technique using PROC SQL syntax. Note how the identical **rename** and **drop** options are used on the output and input tables, respectively. These statements are not only more intuitive to understand for the DATA step programmer, but save coding time over alternative PROC SQL methods for achieving the same result.

```
/**PROC SQL Syntax**/
proc sql;
    create table work.student_data_newsql (rename=(student_id=emplid)) as
           select *
           from work.student_data (drop=academic_level);
quit;
```

As seen from these code snippets, PROC SQL manipulation can be made as simple as the corresponding DATA step version. These coding shortcuts are not limited simply to **rename** and **drop**. Other statements commonly found in DATA step can also be used with PROC SQL, including **keep**, **label**, **compress**, **rename**, and **where**, amongst many others. We recommend consulting Borowiak’s paper, “Using Data Set Options in PROC SQL” (SUGI 31) for more detailed information on this topic.

SUBSETTING IF, RECODING VARIABLES AND WHERE CONDITIONS

Both the subsetting **if** and **where** statements can be used in the DATA step to remove unwanted records from a dataset based on a specified condition. In PROC SQL, the **where** statement allows the use of functions, just as the subsetting **if** and **where** clause does in DATA step. Recoding in PROC SQL, on the other hand, may appear more confusing for those accustomed to the use of the DATA step.

For the purposes of explaining this concept, we will present an example from the student_data table (Figure 1). Our goal is to recode a new gender variable ("gender_new") that presents this value as a whole word instead of the original variable ("gender") that only presents a single-character indicator. Additionally, we only wish to select a subset of data containing students pursuing a PhD.

First, the DATA step version.

```
/**DATA Step Syntax**/
data work.student_data_recode
    set work.student_data
    /**Gender Recode**/
    if gender = 'F' then gender_new = 'Female';
    else if gender = 'M' then gender_new = 'Male';
    /**Subsetting if to keep PhD students only**/
    if substr(Academic_Level,1,8) = "Doctoral";
run;
```

The recode is handled by an **if-then-else** construct, while the data subsetting is addressed with a short **if** statement. Next, we present the PROC SQL version.

```
/**PROC SQL Syntax**/
proc sql;
    create table work.student_data_recode as
           select *,
                  case when gender = 'F' then 'Female'
                         else 'Male'
                  end as gender_new
           from work.student_data
           where substr(Academic_Level,1,8) = "Doctoral";
quit;
```

In the PROC SQL case, the subsetting **if**, covered by the **where** statement, is nearly equivalent. However, the recoding syntax is a little less familiar, necessitating the use of a **case** statement. This equivalency to the DATA step version, while not overly complicated, will require a little more familiarizing oneself than some of the other techniques.

MERGING TABLES (DATASETS) USING WHERE STATEMENT

One of the most intimidating tasks that a DATA step user can find in using PROC SQL involves the complexity of learning and understanding the use of **join** statements to merge datasets. On the other hand, someone who uses PROC SQL recognizes its benefits over DATA step in merging datasets; namely, in the areas of efficiency and speed. The major benefits a SAS programmer will find in using PROC SQL for merging data sets that we would like to highlight include the following:

- no necessity to sort datasets prior to merging;
- no necessity to ensure that the names of matching variables are identical prior to merging; and
- one dataset can be subsetted during the merge using a condition on a variable in the other dataset.

One of the more straightforward ways to handle merging is through the use of the **where** statement of PROC SQL. Understanding this technique can assist in the ease of learning the use of **join** statements at a later time. The **where** statement can handle most types of merges and joins and simply requires the programmer to maintain clear logic in coding. The following examples will explain a few scenarios of merges that can help avoid the complications of using **join** statements.

EXAMPLE: ONE-TO-MANY MERGE/LEFT OUTER JOIN

Our first example involves what is known as a one-to-many merge, or as it is called in PROC SQL, a left outer join. In this case, the goal is to take a dataset that contains a single record per individual in the population and merge it with another dataset that may have multiple entries per population member.

In our example, we have a dataset containing members of a population (student_data_recode, modified from Figure 1) and another dataset that contains the classes that each student took in a particular semester (student_course_data, Figure 2). Both datasets have common key variables, "student_id" and "term," suitable for merging. Our goal is to merge the two datasets to determine the courses that these PhD students took in the academic year 2010-11.

First, we present the activity as completed with a DATA step. Notice how we need to sort both datasets using PROC SORT and then perform the merge with the DATA step. The **in** statement specifies that we want to keep all entries from the student_data_recode dataset, as it is the key dataset that defines our population. The **if** statement ensures that only records that have an entry in student_data_recode are retained. Therefore, if there are any entries that are present only in student_course_data and not in student_data_recode, they will not appear in the final output dataset.

```
/**DATA Step Syntax**/

proc sort data = work.student_data_recode;
    by student_id term;
run;

proc sort data = work.student_course_data;
    by student_id term;
run;

data work.classes_by_phd_students;
    merge work.student_data_recode (in=a) work.student_course_data;
    by student_id term;
    if a;
run;
```

Next, we present the same operation as performed in PROC SQL. It is quite clear that the entire operation is now performed in one procedure, not three, as no sorting is required. No special keywords need to be specified regarding the join type, as the **where** statement indicates the only equivalencies that need to be present.

```
/**PROC SQL Syntax**/
proc sql;
    create table work.classes_by_phd_students_sql as
           select *
           from work.student_data_recode, work.student_course_data
           where student_data_recode.student_id = student_course_data.student_id
                  and student_data_recode.term = student_course_data.term;
quit;
```

Most types of the merges can be handled by manipulating with the where conditions shown above. However, many-to-many merges are difficult to handle with such logical conditions on **where**, so joins might need to be used.

SUMMARIZING DATA USING PROC SQL

One of the most commonly used data manipulation techniques involves sorting and summarizing data to obtain one row per unique ID. DATA step users can use either a very complicated DATA step for this purpose or utilize PROC SORT and PROC SUMMARY. PROC SQL, however, can act as a substitute for all of these methods. As in the case of merging, the benefits of using PROC SQL over other procedures come to light when a task that otherwise requires multiple steps can be performed with a single PROC SQL code snippet.

In the previous example, we gathered the course listing for all doctoral students. Now we would like to find all the students who took more than 9 credit hours of classes in Fall 2010 (term = '201008') and classify them by their academic level. Using PROC SUMMARY and DATA step, we must summarize the student_course_data table by the "section_credits" variable, merge that output table with the student_data table, and again sort and summarize the new data table to obtain the number of student_ids with more than 9 credits. The output dataset appears in Figure 3.

Academic_Level	_TYPE_	_FREQ_
Doctoral	0	183
Doctoral Candidacy	0	15
Freshman	0	6237
GRAD Teacher Certification	0	1
Junior	0	11116
Masters	0	871
Non-Degree	0	79
Professional Certification	0	1
Second Degree	0	215
Senior	0	14260
Sophomore	0	6371
Specialist	0	28

Figure 3. Output of Summarized Table

First, we present the DATA step and PROC SUMMARY way to obtain these results. This three-step process involves 1) sorting the student dataset and summarizing the course dataset, 2) merging the summarized results back to the original student data file that contains academic level, and 3) obtaining a summary by academic level, yielding the output as seen in Figure 3. The job gets done, but not in the most efficient fashion.

```
/**Step 1: Sorting and Summarizing Credits by student by each term**/
proc sort data = work.student_data;
    by student_id term;
run;
proc summary data=work.student_course_data;
    class student_id term;
    output out = work.credits_by_term_summary sum=;
    var section_credits;
run;

/**Step 2: Merging the Credits Summary to Academic Level from student_data table**/
data work.student_data_credits_summary;
    merge work.student_data (keep=student_id term academic_level)
          work.credits_by_term_summary (keep =student_id term section_credits
                where=(student_id ^= ' ' & term = '201008'));
    by student_id term;
    if term = '201008'; /**Subset the data to only include Fall 2010 records **/
run;

/**Step 3: Sorting and Summarizing the above data by Academic Level for Students
    who took more than 9 credits in Fall 2010**/
proc sort data = work.student_data_credits_summary;
    by academic_level;
run;
proc summary data=work.student_data_credits_summary (where=(section_credits>9));
    by academic_level;
     output out = work.acad_level_CreditSummary;
run;
```

Alternatively, the same conclusion can be reached through the use of PROC SQL. We present the code in two steps. First, the **group by** statement is used to create a table featuring one entry for each student and a new variable, “summ_section_credits” containing the total number of credits earned. In the second step, a table is created to bring in student data and summarize credits earned by class standing. While this code is still not particularly brief, it certainly finishes the desired task in fewer steps than with DATA step.

```
proc sql;
    /*Step 1: Create table with section credits by student*/
    create table temp as
          select distinct student_course_data.student_id, student_course_data.term,
                sum(student_course_data.section_credits) as Summ_Section_Credits
          from student_course_data
    where student_course_data.term = '201008'
```

```
    group by student_course_data.student_id
    ;
    /*Step 2: Create table by class standing for those with greater than 9 hours*/
    create table temp_2 as
           select distinct student_data.academic_level, count(temp.student_id) as
           GT9credits
           from student_data, temp
    where student_data.student_id = temp.student_id AND student_data.term =
           temp.term AND temp.summ_section_credits > 9
    group by student_data.academic_level
    ;
quit;
```

CONCLUSION

Changing one's coding technique after many years of completing the task in a certain way may at first feel like forcing oneself to become left-handed after a life of being a right-hander. However, we have presented a series of what we have found to be the most important equivalencies to understand when either translating between DATA step and PROC SQL or figuring out how to write code more effectively. While those who favor DATA step may not save much coding time nor increase programmatic efficiency for simpler tasks, such as in the case of appending datasets, such programmers may want to give PROC SQL a second look for merging and summarizing. With the availability of many familiar SAS options that have long been available, DATA step programmers have plenty of opportunities to ease into PROC SQL, leaving programming biases behind and, perhaps, creating a more harmonious environment among colleagues of differing SAS backgrounds.

REFERENCES

Borowiak, K. W. (2006, March). *Using data set options in PROC SQL*. Paper presented at the 31st Annual SAS Users Group International Conference, San Francisco, CA.

Conway, T. (2008, March). *It's a bird, it's a plane, it's SQL transpose!* Paper presented at the 2008 SAS Global Forum, San Antonio, TX.

Dickstein, C., & Pass, R. (2004, May). *DATA step vs. PROC SQL: What's a neophyte to do?* Paper presented at the 29th Annual SAS Users Group International Conference, Montreal, QC.

Foley, M. J. (2005, April). *MERGING vs. JOINING: Comparing the DATA step with SQL*. Paper presented at the 30th Annual SAS Users Group International Conference, Philadelphia, PA.

Harrington, T. J. (2002, April). *An introduction to SAS PROC SQL*. Paper presented at the 27th Annual SAS Users Group International Conference, Orlando, FL.

Kahane, D. C. (2009, September). *Using DATA step MERGE and PROC SQL JOIN to combine SAS datasets*. Paper presented at the 2009 Northeast SAS Users Group Conference, Burlington, VT.

Lafler, K. P. (2006, March). A hands-on tour inside the world of PROC SQL. Paper presented at the 31st Annual SAS Users Group International Conference, San Francisco, CA.

Lafler, K. P. (2009). *DATA step versus PROC SQL programming techniques*. Paper presented at the Sacramento Valley SAS Users Group Meeting.

Lauderdale, K. (2007, April). *PROC SQL—The dark side of SAS?* Paper presented at the 2007 SAS Global Forum, Orlando, FL.

Marcella, S., & Jorgensen, G. (2010, November) *PROC SQL: Tips and translations for DATA step users*. Paper presented at the 2010 Northeast SAS Users Group Conference, Baltimore, MD.

Matthews, J. (2006, September). *PROC SQL versus the DATA step*. Paper presented at the 2006 Northeast SAS Users Group Conference, Philadelphia, PA.

Williams, C. S. (2002, September/October). *PROC SQL for DATA step die-hards*. Paper presented at the 2002 Northeast SAS Users Group Conference, Buffalo, NY.

CONTACT INFORMATION

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

Meghal Parikh and Elayne Reiss
University of Central Florida
12424 Research Parkway, Suite 215
Orlando, FL 32826
Phone: (407) 882-0285
Fax: (407) 882-0288
uaps@ucf.edu
http://uaps.ucf.edu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.