

## Fun with PROC SQL

### Darryl Putnam, CACI Inc., Stevensville MD

#### ABSTRACT

PROC SQL® is a powerful yet still overlooked tool within our SAS® arsenal. PROC SQL can create tables, sort and summarize data, and join/merge data from multiple tables and in-line views. The SELECT statement with the CASE-WHEN clause can conditionally process the data like the IF-THEN-ELSE statement in the DATA step. An advantage specific to PROC SQL is that with careful coding, the SQL code can be ported to 3rd party Relational Database Management Systems (RBMS) such as Oracle® and SQL Server® with virtually no changes. This paper will show some techniques to QA and reshape data the way you want to see it, with a focus on in-line views.

#### INTRODUCTION

Structured Query Language (SQL) was originally designed to extract data out of a relational database but we can also use it to process SAS data files. SQL has many flavors depending on the relational database system and most flavors follow similar syntax so the same basic code can be used across the different flavors of SQL. This basic syntax is called ANSI SQL. The SAS implementation of SQL is in PROC SQL and enables the SAS programmer to leverage this powerful tool into code that is easier to follow and is more efficient.

This paper will have the reader follow the journey of an analyst working for a fictitious international shoe company that needs to create a series of adhoc management reports that will (unbeknown to our analyst) eventually be ported to the company's production database. Our journey will lead from simple listings to complex conditional processing.

#### EXAMPLE – SASHELP.SHOES

Since every installation of SAS comes with sample data in the SASHELP library, the data file SASHELP.SHOES will be used to demonstrate the PROC SQL code used throughout this paper. The SHOES data file represents the sales and inventory data for a fictitious shoe company with stores worldwide. Our job as an analyst is to analyze the data and generate summary reports to management. But first let us examine the contents of the data file. As with PROC DATASETS and PROC CONTENTS, PROC SQL we can get a layout of the data file with the DESCRIBE statement.

```
proc sql;
  describe table sashelp.shoes;
quit;
```

The output of the DESCRIBE statement occurs in the SAS log. Below is an excerpt of the result.

```
create table SASHELP.SHOES( label='Fictitious Shoe Company Data' bufsize=8192 )
(
  Region char(25),
  Product char(14),
  Subsidiary char(12),
  Stores num label='Number of Stores',
  Sales num format=DOLLAR12. informat=DOLLAR12. label='Total Sales',
  Inventory num format=DOLLAR12. informat=DOLLAR12. label='Total Inventory',
  Returns num format=DOLLAR12. informat=DOLLAR12. label='Total Returns'
);
```

In order to find the number of observations or rows in the data file, we will need to use the COUNT function in PROC SQL. COUNT(\*) counts all the rows of a table.

```
proc sql;
  select count(*) label='Number of Rows in SASHELP.SHOES'
  from sashelp.shoes;
quit;
```

**Figure 1: Number of Rows in SASHELP.SHOES**

Number of Rows in SASHELP.SHOES
395

We are going to quickly review the data by viewing the first 5 rows of data. The OUTOBS option in the PROC SQL statement below selects the top 5 rows out of 395 rows. Note: the OUTOBS= option restricts the displayed output not the number of rows processed.

```
proc sql outobs=5;
  select *
  from sashelp.shoes;
quit;
```

**Figure 2: Listing of Top 5 Rows of SASHELP.SHOES**

Region	Product	Subsidiary	Number of Stores	Total Sales	Total Inventory	Total Returns
Africa	Boot	Addis Ababa	12	\$29,761	\$191,821	\$769
Africa	Men's Casual	Addis Ababa	4	\$67,242	\$118,036	\$2,284
Africa	Men's Dress	Addis Ababa	7	\$76,793	\$136,273	\$2,433
Africa	Sandal	Addis Ababa	10	\$62,819	\$204,284	\$1,861
Africa	Slipper	Addis Ababa	14	\$68,641	\$279,795	\$1,771

## AGGREGATING DATA USING PROC SQL

PROC SQL can of course do more than just list data, PROC SQL can also summarize or aggregate data. Suppose management wants a sales summary for each region. Without PROC SQL, we could accomplish this with PROC MEANS.

```
proc means data=sashelp.shoes sum;
  class region;
  var sales;
run;
```

**Figure 3: Summary Sales by Region – PROC MEANS Output**

Analysis Variable : Sales Total Sales		
Region	N Obs	Sum
Africa	56	2342588.00
Asia	14	460231.00
Canada	37	4255712.00
Central America/Caribbean	32	3657753.00
Eastern Europe	31	2394940.00
Middle East	24	5631779.00
Pacific	45	2296794.00
South America	54	2434783.00
United States	40	5503986.00
Western Europe	62	4873000.00

PROC MEANS is powerful but it cannot be ported to our production database and has a few limitations. Fortunately for us, a similar output can be accomplished with PROC SQL. In the example below, the GROUP BY clause tells PROC SQL to calculate the sum of the sales for each region. In addition to calculating the group level statistics, PROC SQL can also format the display by using any SAS format, something PROC MEANS cannot do. To make the output more readable the format DOLLARw. was used for the summarized sales figure.

```
proc sql;
  select region
         ,count(sales)   as n_obs
         ,sum(sales)     as sales   format=dollar16.
  from sashelp.shoes
  group by region;
quit;
```

**Figure 4: Summarized Sales by Region – PROC SQL Output**

Region	n_obs	sales
Africa	56	\$2,342,588
Asia	14	\$460,231
Canada	37	\$4,255,712
Central America/Caribbean	32	\$3,657,753
Eastern Europe	31	\$2,394,940
Middle East	24	\$5,631,779
Pacific	45	\$2,296,794
South America	54	\$2,434,783
United States	40	\$5,503,986
Western Europe	62	\$4,873,000

Besides having similar summarizing functionality as PROC MEANS, PROC SQL can also summarize data based on 2 or more fields. Suppose management wants a report that shows net sales (Sales – Returns) and the ratio of net sales to sales for each region (See figure 5). To do that without PROC SQL, we would have to make another data set with a new variable such as Net\_Sales=Sales>Returns, then run PROC MEANS on that new variable. The PROC SQL solution is elegant and will open up a new world of coding to our analyst.

```
proc sql;
  select region
         ,sum(sales)           as Sales           format=dollar16.
         ,sum(sales>Returns)   as net_sales      format=dollar16. label='Net Sales'
         ,sum(sales>Returns)/sum(sales) as net_sales_ratio format=percent10.2
                                         label='Net Sales Ratio'
  from sashelp.shoes
  group by region;
quit;
```

**Figure 5: Summarized Region Sales and Net Sales**

Region	Sales	Net Sales	Net Sales Ratio
Africa	\$2,342,588	\$2,268,501	96.84%
Asia	\$460,231	\$449,336	97.63%
Canada	\$4,255,712	\$4,126,318	96.96%
Central America/Caribbean	\$3,657,753	\$3,530,855	96.53%
Eastern Europe	\$2,394,940	\$2,308,239	96.38%
Middle East	\$5,631,779	\$5,424,899	96.33%
Pacific	\$2,296,794	\$2,219,665	96.64%
South America	\$2,434,783	\$2,331,932	95.78%
United States	\$5,503,986	\$5,316,484	96.59%
Western Europe	\$4,873,000	\$4,703,245	96.52%

## IN-LINE VIEWS

Our journey as an analyst continues, our manager is asking for a copy of the original SHOES data with the summary

of sales by region added to it (See figure 6).

**Figure 6: SHOES with Region Sales**

Region	Product	Subsidiary	Number of Stores	Total Sales	Total Inventory	Total Returns	Region Sales
Africa	Boot	Addis Ababa	12	\$29,761	\$191,821	\$769	\$2,342,588
Africa	Men's Casual	Addis Ababa	4	\$67,242	\$118,036	\$2,284	\$2,342,588
Africa	Men's Dress	Addis Ababa	7	\$76,793	\$136,273	\$2,433	\$2,342,588
Africa	Sandal	Addis Ababa	10	\$62,819	\$204,284	\$1,861	\$2,342,588
Africa	Slipper	Addis Ababa	14	\$68,641	\$279,795	\$1,771	\$2,342,588

In order to get the above desired results, we first need to create table in which we will call REGION\_SALES, then join REGION\_SALES to SASHELP.SHOES by region. This is a 2 step process and is demonstrated in the following PROC SQL code.

```
proc sql;
  create table region_sales as
    select region
      ,sum(sales) as region_sales
    from sashelp.shoes
  group by region;

proc sql outobs=5;
  select a.*
      ,b.region_sales format=dollar16. label='Region Sales'
    from sashelp.shoes a join region_sales b on a.region=b.region;
quit;
```

Instead of using a table as the source of data for your PROC SQL query, you can also use a data structure called an in-line view. An in-line view is a nested query that is specified in the FROM clause. An in-line view selects data from one or more tables to produce a temporary in-memory table. This virtual table exists only during the query. The main advantage of using an in-line view is to reduce the complexity of the code. (SAS Certification Prep Guide).

An in-line view is a SELECT statement within a SELECT statement, which we call a nested statement. Nested SELECT statements sound complex but they are not. In our example we are going to join the output of figure 4 to the SHOES data to get the results in figure 6.

```
proc sql outobs=5;
  select a.*
      ,b.region_sales format=dollar16. Label='Region Sales'
    from sashelp.shoes a join (select region
      ,sum(sales) as region_sales
    from sashelp.shoes
    group by region) b on a.region=b.region;
quit;
```

Let us examine this code in some detail. The code snippet below is the in-line view. Notice how the in-line view is enclosed in parentheses. A table like the table in figure 4 is created in memory and then is joined with SHOES to get our final result set which is identical to the output of the 2 step process. In essence we joined the SHOES data with a summarization of itself.

```
(select region
  ,sum(sales) as region_sales
 from sashelp.shoes
 group by region)
```

## FINDING DUPLICATES

We can also uncover duplicate rows in the data by using our newly found arsenal of SQL programming tricks. Our example data SASHELP.SHOES is supposed to be unique by REGION, PRODUCT, and SUBSIDIARY. We could test for and output any offending duplicate rows by using PROC SORT, but we can also do our duplicate testing with

PROC SQL by using our newly gained knowledge of in-line views and group by aggregation along with the HAVING clause. With PROC SQL, we can filter data before and after aggregation. The HAVING clause enables us to subset the in-line view after the aggregation is complete.

The below PROC SQL code outputs all rows that have duplicate values of REGION, PRODUCT, and SUBSIDIARY in figure 7.

```
proc sql;
  select a.*
  from sashelp.shoes a join (select region, product, subsidiary
                           ,count(sales) as count
                           from sashelp.shoes
                           group by region, product, subsidiary
                           having count(sales)>1) b on
  a.region=b.region
  and a.product=b.product and a.subsidiary=b.subsidiary;
quit;
```

**Figure 7: Duplicate Rows**

Region	Product	Subsidiary	Number of Stores	Total Sales	Total Inventory	Total Returns
Western Europe	Sport Shoe	Copenhagen	1	\$1,927	\$17,683	\$31
Western Europe	Sport Shoe	Copenhagen	13	\$101,922	\$327,742	\$4,204

The in-line view piece of the PROC SQL code, first generates a count for each REGION, PRODUCT, and SUBSIDIARY. Then the HAVING clause only keeps those rows where the count is more than 1. The HAVING clause must appear after the GROUP BY clause, and is processed after the GROUP BY is completed.

## CONDITIONAL PROCESSING WITH CASE-WHEN

Once more our manager comes in and has another data call. Management wants to know the percent of boot sales to total sales for each region. After some thought, we wondered if we could embed a conditional CASE-WHEN within our summary function. The summary function would create a new column which would contain only the boot sales for each region. Next we could divide that new column by the total sales to get our percentage of boot sales. Below is the PROC SQL query that was used to generate the data in figure 8.

```
proc sql outobs=5;
  select region
         ,sum(sales) as sales format=dollar16.
         ,sum(case when Product='Boot' then sales else 0 end) as boot_sales
           format=dollar16.
         ,sum(case when Product='Boot' then sales else 0 end) / sum(sales) as
           boot_sales_ratio format=percent10.2
  from sashelp.shoes
  group by region;
quit;
```

**Figure 8: CASE-WHEN with GROUP BY**

Region	sales	boot_sales	boot_sales_ratio
Africa	\$2,342,588	\$119,835	5.12%
Asia	\$460,231	\$62,708	13.63%
Canada	\$4,255,712	\$385,613	9.06%
Central America/Caribbean	\$3,657,753	\$190,743	5.21%
Eastern Europe	\$2,394,940	\$306,785	12.81%

## SQL IN DATABASE MANAGEMENT SYSTEMS

Much of this paper showed the reader how to use PROC SQL instead of other SAS procedures, to generate new tables and reports. One neglected benefit for using PROC SQL, is the ability to port the SQL code to 3rd party databases. As an analyst in our fictitious shoe company, our adhoc reports gathered visibility, and the management team wants the database system to automatically create out our reports. We can accomplish this by stripping out all the SAS only SQL syntax. We remove the format and label options and now we have ANSI SQL which can be sent to the database team for implementation. ANSI SQL is the industry standard for SQL syntax and is generally accepted by most database vendors.

Also, most of the PROC SQL queries ran in a single step without creating temporary tables. Some database systems will not allow the user to create temporary tables without permission from the database administrator. Structuring a query that needs a temporary table get the desired output can put a hindrance to our analysis. By using in-line views instead of temporary tables for our queries, we can solve that potential problem before it arises.

## CONCLUSION

The PROC SQL syntax is myriad and complex but with some simple techniques, we can leverage this powerful procedure to bend the data to our will. In this paper a few examples were shown to enable the reader to start using PROC SQL for common data processing tasks: summarizing data, finding duplicates, and merging data. The benefits of in-line views were shown not just to make our code elegant but also solving real world problem of porting our code to a third party database. By stripping out the SAS specific pieces of SQL, we can use the basic and advanced SQL queries to query any data from any ANSI SQL system, thus making us more well rounded analysts.

## REFERENCES

- Base SAS® Certification Prep Guide Advance Programming for SAS® v9, 2007, SAS Institute Inc., Cary, NC.
- DeFoor, Jimmy, "Proc SQL – A Primer for SAS Programmers", 2006, SUGI 31 Proceedings, San Francisco, CA, <http://www2.sas.com/proceedings/sugi31/250-31.pdf>
- Lafler, Kirk Paul; Shipp, Charles Edwin "A Visual Introduction to PROC SQL Joins", Proceedings of PharmSug 2004, San Diego, CA, <http://www.lexjansen.com/pharmasug/2004/tutorials/tu06.pdf>
- Ronk, Katie Minten; First, Steve; Beam, David, "An Introduction to PROC SQL", 2002, SUGI 27, Orlando, FL, [http://www.sys-seminar.com/presentations/pres\\_intro\\_to\\_SQL.htm](http://www.sys-seminar.com/presentations/pres_intro_to_SQL.htm)
- Hu, Weiming, "Top Ten Reasons to Use PROC SQL", 2004, SUGI 29, Montréal, Canada , <http://www2.sas.com/proceedings/sugi29/042-29.pdf>
- Williams, Christianna S, "PROC SQL for DATA Step Die-Hards", 2002, NESUG 15, Buffalo, NY, <http://www.ats.ucla.edu/stat/sas/library/nesug9>

## CONTACT INFORMATION

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

Darryl Putnam  
CACI Inc.  
6835 Deerpath Road  
Elkridge, MD 21075  
Work Phone: 410-762-6535  
E-mail: [dputnam@caci.com](mailto:dputnam@caci.com)

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.