

Getting Subtotals and Percentages It's Never Been So Easy

Christopher Toppe, Ph.D.
Computer Sciences Corporation

Abstract

Getting subtotals and percentages onto a database has always been one of the curses of working with SAS data. The most common techniques are neither easy to conceptualize nor easy to program. Sorting your data and using FIRST. and LAST. variables is perhaps the standard brute-force programming methodology that we all have learned to use and come to hate. Using PROC MEANS (or PROC SUMMARY) with CLASS and OUTPUT statements is perhaps a more advanced approach to the same result, but is certainly no easier to take to a final product. After all, do you really like using SET with a POINT? Isn't it a monumental pain!? As the current 'phrase of the times' says, "Been there, done that." But no more! There is a much better way to get subtotals and percentages. A method so easy that even novice programmers will be able to use this technique without taking a trip to the stratosphere of SAS programming. What is it? PROC SQL!

Introduction

In my opinion, PROC SQL is the single most powerful data manipulation tool in Base SAS. It is both a SAS procedure and a programming language. As a SAS procedure, it manipulates SAS datasets, creating reports and/or new datasets in the process. But it is as a programming language that it is perhaps most powerful. As you get accustomed to working with PROC SQL, you will find that it makes easy some of the most difficult tasks in Base SAS Programming. Specifically, it makes the task of generating subtotals and percentages child's play when compared to other methods available through SAS.

For this paper, I will use as an example a dataset that has eight observations. This database contains

information on the salaries of four administrative and four management employees of a fictitious company, with each employee's gender also recorded. As you will see in the code used to create this dataset, females were assigned a salary less than that of men, and administrative positions were assigned salaries lower than those for management positions; these adjustments make the example reflect the real world. The code in Table 1 creates the data.

The Syntax of PROC SQL

To understand PROC SQL, you need to understand how the SQL syntax relates to normal SAS syntax. In SQL, you create a *table*, which is the exact same thing as a SAS *dataset*. A table has *rows* and *columns* while a dataset has *observations* and *variables*. Other than those naming differences, there is no difference between an SQL table and a SAS dataset. PROC SQL can read SAS datasets and SAS can read SQL tables.

To compare the basic syntax of PROC SQL and Base SAS, the following example is provided:

```
DATA NEW;  
  SET OLD;  
RUN;  
  
PROC SQL;  
  CREATE TABLE NEW AS  
    SELECT *  
    FROM OLD;  
QUIT;
```

Both these pieces of code do the same thing: create a dataset named NEW that is a copy of the dataset named MASTER from the MYLIB library. All the variables from MASTER are included in the dataset NEW. (This is not good code, it is just an example!)

```

DATA TEMP;
  LENGTH GENDER LEVEL $1 SALARY 3;
  DO I = 1 TO 2;
  DROP I;
    DO GENDER = 'F','M';
      DO LEVEL = 'ADMIN','MGMT';
        SALARY = ROUND(25000 + (5000*RANNOR(246246)),1000);
        IF LEVEL = 'MGMT' THEN SALARY + 10000;
        IF GENDER = 'F' THEN SALARY +(- 3000);
        FORMAT SALARY DOLLAR8.;
        OUTPUT;
      END;
    END;
  END;
RUN ;

PROC PRINT DATA=TEMP;
RUN;

```

INPUT DATASET		
GENDER	LEVEL	SALARY
F	A	\$14,000
F	M	\$29,000
M	A	\$24,000
M	M	\$31,000
F	A	\$22,000
F	M	\$24,000
M	A	\$27,000
M	M	\$29,000

Table 1
Sample Data Program and Print Out

Note several things about SQL. First, it is an interactive procedure. It runs until you QUIT it. Therefore, a RUN; statement is not needed and will be ignored if included. Second, there is only one main section, which in this example starts with the keyword CREATE and ends with a single semicolon. The SELECT statement uses an asterisk (*) to indicate all variables in MASTER are selected for inclusion in NEW. In English, you can read the SQL code as "Create a table named NEW, selecting ALL variables from the table MYLIB.MASTER.

Variable Lists

One critical difference between SAS and SQL is in the use of the comma in variable lists. In SQL, commas are required as separators in a variable list --

blanks do not work. Note the use of commas and blanks in the following:

```

PROC SQL;
  SELECT SALARY, GENDER, LEVEL
  FROM TEMP;
QUIT;

```

Note that the elements being selected, the *columns* in SQL terms, are separated by commas.

The GROUP BY Statement

A clear advantage of SQL over Base SAS can be seen when you want to work with data in sorted order. With SAS, you sort and process your data in separate programming steps, while in SQL, sorting and processing are accomplished in one programming

step. SQL uses the GROUP BY statement to indicate the sort order of the output table. Compare the following code extracts:

```
PROC SORT DATA=MYLIB.MASTER OUT=NEW;
  BY GENDER LEVEL;
RUN;
```

```
DATA NEW;
  SET NEW;
  BY GENDER LEVEL;
  ***MORE CODE***
RUN;
```

```
PROC SQL;
  CREATE TABLE NEW AS
  SELECT GENDER, LEVEL
  ***MORE CODE***
  FROM MYLIB.OLD
  GROUP BY GENDER, LEVEL;
QUIT;
```

While this may seem to be a trivial difference, it will be important when you see how this trivial difference makes it possible to generate subtotals later in this paper.

Creating New Columns (Variables)

In Base SAS, you create new variables by using an assignment statement, such as "NEWVAR = Y+Z;". In SQL, things work just a bit backwards: you define the column (variable) first then give it a name (and, optionally, other attributes such as a format). In the following code, you create a variable named NEWVAR to be the sum of Y and Z, giving it the comma7.2 format.

```
PROC SQL;
  SELECT VAR1, SUM(Y,Z) AS NEWVAR
  FORMAT = COMMA7.2
  FROM MYLIB.OLD;
QUIT;
```

This code will result in a new column being created for each row (a new variable for each observation) which is the sum of columns Y and Z, with the new column being named NEWVAR formatted with the COMMA7.2 format.

Using Summary Functions

Summary functions create summary statistics for a table, just as you could obtain from, for example,

PROC MEANS. The statistic keywords are just what you would expect, including standards such as MEAN, N, NMISS, MIN, MAX, and SUM. The following code results in *one* row (record) being output, giving the summary across the entire table (dataset). Note how you can give a calculated column a name (MEAN1), a label (MEAN) and/or a format (COMMA6.).

```
PROC SQL;
  SELECT
    MEAN(VAR1) AS MEAN1 LABEL="MEAN",
    SUM(VAR1) AS SUM1 FORMAT=COMMA6.,
    MIN(VAR1) AS MIN1
  FROM MYLIB.MASTER;
QUIT;
```

Creating Subtotals with a GROUP BY Statement

Adding a GROUP BY statement changes how SQL handles summary functions, a change that makes getting subtotals and percentages so very easy. The GROUP BY statement causes SQL to generate a function's results for each value of the GROUP BY variable(s). In the following example, note that we are SELECTING only the GROUP BY variable and the summarized values for the variable SALARY. In this case, SQL returns only summarized data, shown in Table 2.

```
PROC SQL;
  SELECT GENDER,
    MEAN(SALARY) AS MEAN,
    SUM(SALARY) AS SUM
  FROM TEMP
  GROUP BY GENDER;
QUIT;
```

GENDER	MEAN	SUM
F	22250	89000
M	27750	111000

Table 2
Summarized Data

This may be interesting and easy, but it is not quite what we are looking for. We want to compute percentages and ratios, and to have those summary statistics remerged with the original data. That is, we want to get the SUM and MEAN statistics added back to the original dataset. Since this is the main topic of this paper, let's look at this method more closely.

Remerging Summary Statistics with SQL

To show you how to get subtotals using PROC SQL, I will walk you through the steps one at a time, and show coding shortcuts along the way. What I am going to show you is how to get summary statistics merged back onto the original data file to that all the data you need to create a report is in one place.

In the SQL code that follows, it is important to note that we select columns to be included in the output table beyond just those required to calculate the summary statistics. That is, in the previous example, we selected just the column GENDER, which is the

GROUP BY variable, and the calculated statistics. In that example, the resulting table contained only the summary statistics for each value of the GROUP BY column (GENDER.) In the next example we are requesting that all columns be included in the new table (TEMP1) by using the "*" as an alias for the Base SAS keyword _ALL_. This syntax causes the SQL procedure to perform two steps. First, it calculates the requested statistics, then it remerges those values back onto the original data. The result is a new table (or dataset) that contains the summarized data along with the detailed data (Table 3).

```
PROC SQL;
TITLE 'STEP1: OVERALL SUMMARY';
CREATE TABLE TEMP1 AS
  SELECT *,
    SUM(SALARY) AS TOTAL FORMAT=DOLLAR8. LABEL='TOTAL SALARY',
    MEAN(SALARY) AS AVERAGE FORMAT=DOLLAR7. LABEL='AVERAGE SALARY'
  FROM TEMP;

SELECT *
FROM TEMP1;
QUIT;
```

STEP1: OVERALL SUMMARY

GENDER	LEVEL	SALARY	TOTAL SALARY	AVERAGE SALARY
F	A	\$14,000	\$200,000	\$25,000
F	M	\$29,000	\$200,000	\$25,000
M	A	\$24,000	\$200,000	\$25,000
M	M	\$31,000	\$200,000	\$25,000
F	A	\$22,000	\$200,000	\$25,000
F	M	\$24,000	\$200,000	\$25,000
M	A	\$27,000	\$200,000	\$25,000
M	M	\$29,000	\$200,000	\$25,000

Table 3
Overall Summary

Note that in the above, SQL has remerged the summary statistics (SUM and MEAN) back onto the original data, and uses the formats and labels in creating its report.

We now use this new table (dataset) as input into the next step, where we will get summary statistics for the values of LEVEL (using the GROUP BY statement).

Note that we are SELECTING all columns from table TEMP1, as well as a column for the calculated mean of SALARY. We are also using the mean of salary as part of a formula to create a new column for the ratio of the mean salary for each value of LEVEL to the overall mean (calculated in the prior step). The program and results are shown in Table 4 on the next page.

```

PROC SQL;
  TITLE 'STEP2: SUMMARY BY LEVEL';
  CREATE TABLE TEMP2 AS
  SELECT *,
    MEAN(SALARY) AS LMEAN FORMAT=DOLLAR7. LABEL='LEVEL AVERAGE',
    (100*MEAN(SALARY)/AVERAGE) AS LRATIO FORMAT=COMMA6. LABEL='LEVEL RATIO'
  FROM TEMP1
  GROUP BY LEVEL;

SELECT *
FROM TEMP2;
QUIT;

```

STEP2: SUMMARY BY LEVEL

GENDER	LEVEL	SALARY	TOTAL SALARY	AVERAGE SALARY	LEVEL AVERAGE	LEVEL RATIO
M	A	\$27,000	\$200,000	\$25,000	\$21,750	87
M	A	\$24,000	\$200,000	\$25,000	\$21,750	87
F	A	\$14,000	\$200,000	\$25,000	\$21,750	87
F	A	\$22,000	\$200,000	\$25,000	\$21,750	87
F	M	\$24,000	\$200,000	\$25,000	\$28,250	113
M	M	\$29,000	\$200,000	\$25,000	\$28,250	113
M	M	\$31,000	\$200,000	\$25,000	\$28,250	113
F	M	\$29,000	\$200,000	\$25,000	\$28,250	113

Table 4
Summary Statistics by Level

Notice two things above. One, the new summary statistics have been added to the original data. Two, the data have been reordered. In order to process the GROUP BY statement, PROC SQL sorted the data for us. It 'knew' what to do to process the SELECT statement. Also notice that we had SQL calculating MEAN(SALARY) twice: once for the column in the output dataset and again as part of the formula for the ratio.

This is eliminated in the next step when we use the CALCULATED keyword. The CALCULATED keyword allows you to use in a formula a column that is being calculated in the current SQL step. This is seen where we calculate GRATIO, the ratio of the mean SALARY by GENDER to the AVERAGE. Table 5 shows the SQL program and results.

```

PROC SQL;
  TITLE 'STEP3: SUMMARY BY GENDER';
  CREATE TABLE TEMP3 AS
  SELECT *,
    MEAN(SALARY) AS GMEAN FORMAT=DOLLAR7. LABEL='GENDER MEAN',
    (100*CALCULATED GMEAN/AVERAGE) AS GRATIO FORMAT=6. LABEL='GENDER RATIO'
  FROM TEMP2
  GROUP BY GENDER;

SELECT GENDER, LEVEL, AVERAGE, LMEAN, LRATIO, GMEAN, GRATIO
FROM TEMP3;
QUIT;

```

STEP3: SUMMARY BY GENDER

GENDER	LEVEL	AVERAGE SALARY	LEVEL AVERAGE	LEVEL RATIO	GENDER MEAN	GENDER RATIO
F	M	\$25,000	\$28,250	113	\$22,250	89
F	A	\$25,000	\$21,750	87	\$22,250	89
F	A	\$25,000	\$21,750	87	\$22,250	89
F	M	\$25,000	\$28,250	113	\$22,250	89
M	M	\$25,000	\$28,250	113	\$27,750	111
M	M	\$25,000	\$28,250	113	\$27,750	111
M	A	\$25,000	\$21,750	87	\$27,750	111
M	A	\$25,000	\$21,750	87	\$27,750	111

Table 5
Summary by Gender

Finally, I want to point out that storing the intermediate steps, the tables (datasets) named TEMP1 - TEMP2 are simply holding spots for the data in progress. All we really use them for is input to the next step. Therefore, it is possible to actually write SQL code that reads these results directly. This is known as a subquery. It is exactly the same process as the individual steps above except that the intermediate tables are not being written to your WORK library. This subquery technique is covered in the presentation, copies of which are available from me at the address below.

Once you become familiar with this technique, creating subtotals, percentages, and other summary statistics will become so easy you'll wonder why you ever used any other method. This code is easy to validate, easy to maintain, easy to explain, and easy to understand. I suggest that with no more than a few minutes of effort, you will add this technique to your repertoire of memorized techniques.

This is just one area in which SQL provides the perfect solution. I encourage you to explore it for other tricks and shortcuts.

About the Author...

Chris Toppe is a decision support systems specialist and Director of Educational Services at Computer Science Corporation's Commercial and Civil Business Center in Rockville, MD. In the DSS area, Dr. Toppe advises on decision strategies in data warehouse and EIS projects. In the education area, he manages the delivery of SAS and other application development training to CSC's commercial and government clients. He teaches base SAS, Intermediate SAS Programming, PROC TABULATE, PROC SQL, and Report Writing. He has been using SAS for nineteen years.

Chris Toppe
1301 Piccard Drive
Rockville, MD 20850
(301) 258 - 5308 (Voice)
(301) 258 - 6878 (Fax)
Christopher_Toppe/SED/CSC.CSC@CSCMAIL.CSC.COM (e-mail)
CTOPPE@CSC.COM (SAS-L)