Paper 222-2007

The Power of the BY Statement

Paul Choate, California Department of Developmental Services, Sacramento, CA
Toby Dunn, Stat Works Inc., San Antonio, TX

INTRODUCTION

When used to its fullest potential, the BY statement can save time and effort, and add clarity and simplicity to SAS programs. It can make complex coding problems simple or accomplish in one pass what would otherwise require multiple passes of the data. Unfortunately the full power of the BY statement is often left unleashed, and instead is relegated to simple mundane everyday work. This paper surveys the basics of the BY statement: the DATA step observation read control statements SET, UPDATE, and MERGE, as well as several basic applications of the BY statement in SAS procedures. Also discussed is how proper data structure will help you unleash the power of the BY statement to gain overall efficiency in your code. The intended audience is beginner and intermediate level SAS programmers.

BY STATEMENT SYNTAX AND USAGE

The BY statement is used in SAS to instruct the DATA step or procedures to process dataset observations in groups, rather than singly. It can be used whenever SAS data is ordered, or can be accessed in order through a SAS dataset index. In the DATA step this allows observations to be summarized or reorganized according to a group structure. In PROC steps it allows SAS to process and present data in groups.

The basic syntax of the BY statement is the same throughout SAS, with the exception that the GROUPFORMAT option is only available in the DATA step.

```
BY <DESCENDING> var1 <...<DESCENDING> varn> <NOTSORTED> <GROUPFORMAT>;
```

In its simplest form the BY statement is followed by a list of variables on which the data is sorted. Data may be from a previously sorted source, or sorted during processing with PROC SORT or PROC SQL with a GROUP BY statement. Once sorted, the SAS DATA step and procedures can take advantage of the ordered structure through use of the BY statement. For example the following data could be accessed with:

```
BY Sex Age Name;
```

NAME	SEX	AGE	HEIGHT	WEIGHT
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Judy	F	14	64.3	90
Jeffrey	M	13	62.5	84
Alfred	M	14	69	112.5
Ronald	M	15	67	133
Philip	M	16	72	150

Sort order is platform dependent and is based on the internal ordering of the platform character set, called the collating sequence. Different platform character sets have different ordering depending on the native collating sequence of the platform. For example, most personal computers use the ASCII character set which is ordered 1, 2, 3, ... A, B, C, ... a, b, c ... while IBM mainframes use the EBCDIC character set with order a, b, c, ... A, B, C, ... 1, 2, 3, ... Character code sets have 256 characters based on permutations of eight binary bit values, forming what is called a byte or a character. The byte value of the characters determines the order of the characters in the collating sequence. The SAS RANK function returns the byte value of a character, which is the position of the character in the collating sequence.

In SAS, sorts are performed from the leftmost character to the rightmost character of a character variable's value, or on the full numeric value of a numeric variable. Missing numeric values are sorted before any non-missing numeric values. Character collating sequences include all the printable characters, as well as non-printable characters such as tabs and line breaks, so embedded blanks and special characters also effect the order of sorted data. This paper refers to sorts on an ASCII platform, but the concepts are applicable on all supported platforms.

To make use of the BY statement SAS datasets need not be in the sort order of the collating sequence; they may also be ordered as descending, in unsorted groups, or in unsorted groups based on a SAS format. For example, if the ages were ordered in descending order the corresponding BY statement would be:

```
BY Sex DESCENDING Age Name;

NAME      SEX    AGE    HEIGHT    WEIGHT
Janet      F      15     62.5      112.5
Carol      F      14     62.8      102.5
Alice      F      13     56.5      84
Barbara    F      13     65.3      98
Philip     M      16     72        150
Alfred     M      14     69        112.5
Henry      M      14     63.5      102.5
```

If the data are grouped on key variables but not sorted, then NOTSORTED can be used. That is to say that the data need only be grouped and the groupings need not be in any kind of sorted order. In this case the age value 13 is not in the proper order, but since all ages are grouped the BY statement still works:

```
BY Age NOTSORTED;

NAME      AGE    HEIGHT    WEIGHT
Carol      14     62.8      102.5
Judy       14     64.3      90
Janet      15     62.5      112.5
Ronald     15     67        133
Mary       15     66.5      112
Alice      13     56.5      84
Jeffrey    13     62.5      84
```

If a format can be used and the result from applying that format to the data will result in the data being grouped as above, then the GROUPFORMAT option may be used. In this example the FORMAT statement groups the Name variable based on the first initial, and so although the names starting with A's and J's are out of order, the formatted values are not:

```
PROC FORMAT;
  VALUE $Initials 'A'-<'B'='A'
                  'B'-<'C'='B'
                  'C'-<'D'='C'
                   ...
RUN;

DATA Class;
  SET Class;
  FORMAT Name $Initials.;
  BY Name GROUPFORMAT;
RUN;

NAME       AGE     HEIGHT    WEIGHT
Alice       13      56.5      84
Alfred      14      69        112.5
Judy        14      64.3      90
Janet       15      62.5      112.5
Jeffrey     13      62.5      84
William     15      66.5      112
```

Another example would be to format a date variable as a year value, where data are in order by year, but dates are mixed within the years. By using the NOTSORTED and GROUPFORMAT options, such grouped data may be processed without first sorting the dataset, potentially saving much time and cost in processing.

INTRODUCTION TO DATA STRUCTURE

As discussed, the BY statement allows observations to be processed in groups. Reducing programming problems by using groups of observations can make the problems simpler by taking advantage of business rules. The BY statement identifies and uses group processing based on the structure of your data. For the SAS programmer, a

good foundation in data structure is important, among other reasons by helping the programmer take full advantage of the power and efficiencies available with BY group processing.

To understand proper data structure, one needs to understand the dataset and its parts. This is a field of computer science known more broadly as *database theory*. A database is composed of associated *tables*, known as *datasets* in SAS parlance. Datasets are sets of records representing information collected from the real world for some business purpose. This collection of records and the structural relationships of the records are referred to as a *model*. The model describes the organization of the records, preferably in an efficient design. Efficient design of a database results from imposing organizational rules or logical constraints known as the *form* of the database. Although most databases do not have perfect form, datasets in more optimal forms will generally be more efficient in processing and storage.

In a dataset, variable values can be thought of as pieces of information which describe attributes of an observation or record. Within an observation, the variables may be divided into two classes: *primary key variables*, whose values may be combined uniquely to identify one observation or event, and *non-primary keys*, whose values cannot be combined to uniquely identify an observation. The primary and non-primary keys are all related to each other in some fashion known as *functional dependencies*. Primary keys must be unique or form unique combinations called c*omposite keys* in order to appropriately link datasets within a database.

The most fundamental rule in dataset design is that no two rows shall have the same unique values for all primary key variables. For example, the following dataset:

```
VEHICLETYPE   MODEL   MAKE    YEAR   COLOR
Truck         1500    Chevy   2008    Blue
Truck         1500    Chevy   2008    Blue
```

should be reduced to:

```
VEHICLETYPE   MODEL   MAKE    YEAR   COLOR  COUNT
Truck         1500    Chevy   2008    Blue   2
```

Furthermore, only one variable should describe a single fact regarding the observation. A common dataset design error occurs when more than one variable describes the same piece of information. This often happens when information is stored in the variable name rather than the variable value. For example, in this observation the color of vehicles sold is inappropriately stored within variable names, rather than appropriately within a variable named Color.

```
VEHICLETYPE   MODEL   MAKE    YEAR   RED  BLUE  GOLD
Truck         1500    Chevy   2008    1    2     3
```

To correct this, the color and number of vehicles sold should each be given its own variable:

```
VEHICLETYPE  MODEL    MAKE    YEAR    COLOR   NUMSOLD
Truck        1500     Chevy   2008    Red       1
Truck        1500     Chevy   2008    Blue      2
Truck        1500     Chevy   2008    Gold      3
```

One common example of this type of error is when date or time values are stored as variable names. Dates and times should be stored as values, and not as variable names. When stored as values, the data can then be sorted on date or time values and BY group processing applied on the reporting intervals of year, month, or day.

Also, each variable in the dataset should have atomic values, otherwise the data structure is said to have a relation within a relation. Having variables with atomic values ensures efficiency in the sense that information does not need to be additionally parsed from a value when it is read. For example the following dataset:

```
VEHICLETYPE  MODEL    MAKE    YEAR    COLOR   NUMSOLD  PACKAGE
Truck        1500     Chevy   2008    Red       1      Sports
Truck        1500     Chevy   2008    Blue      2      Sports, Standard
Truck        1500     Chevy   2008    Gold      3      Sports, Sports, Standard
```

should be restructured as:

```
VEHICLETYPE  MODEL    MAKE    YEAR    COLOR   NUMSOLD  PACKAGE
```

```
Truck        1500     Chevy   2008    Red       1      Sports
Truck        1500     Chevy   2008    Blue      1      Sports
Truck        1500     Chevy   2008    Blue      1      Standard
Truck        1500     Chevy   2008    Gold      2      Sports
Truck        1500     Chevy   2008    Gold      1      Standard
```

This final example demonstrates a dataset that is said to be in *first normal form with redundancies*. Higher normal forms of a dataset have to do with the uniqueness of relationships between non-key variables, and subsetting tables apart into linked tables that have the least amount of redundant information possible. While there are many higher levels of normal forms, data should be in at least first normal form to make proper use of BY group processing. References for further reading on database models and normal forms are provided in *Suggested Reading* below. Adhering to the first normal form creates a favorable data structure for BY group processing.

BY STATEMENT IN THE DATA STEP

The BY statement controls the operation of a SET, MERGE, MODIFY, or UPDATE statement in the DATA step, and provides two automatic temporary variables for each BY variable: the *FIRST.variable* and the *LAST.variable*. For example, if the data are sorted on Age, then when SAS processes the BY statement it creates the two automatic temporary variables: FIRST.Age and LAST.Age. SAS reads ahead by one observation as it passes through the data and sets the automatic variable values to 1 when the first or last values in a group are processed and 0 otherwise.

The FIRST.variable and LAST.variable values indicate whether an observation is:

- the first in a BY group
- the last in a BY group
- neither the first nor the last in a BY group
- both first and last, as is the case when there is only one observation in a BY group.

```
SET Class;
BY Sex Age;

SEX AGE  FIRST.SEX LAST.SEX FIRST.AGE LAST.AGE
 F   13       1         0         1         0
 F   13       0         0         0         1
 F   14       0         0         1         0
 F   14       0         0         0         1
 F   15       0         0         1         0
 F   15       0         1         0         1
 M   13       1         0         1         1
 M   14       0         0         1         0
 M   14       0         0         0         1
 M   15       0         0         1         0
 M   15       0         0         0         1
 M   16       0         1         1         1
```

The temporary FIRST.variable and LAST.variable can be used by the SAS programmer to identify the breaks between groups during DATA step processing. The sort ordering of the variables is hierarchical, or also called nested; in this case Age is sorted within Sex. This is useful for many things, such as unduplicating the data or counting members of lower sort groups within a higher sort group.

This example shows how the automatic FIRST. variables can be used to unduplicate the dataset by Sex and by Age within Sex:

```
DATA Undup_Sex(KEEP=Sex) Undup_Age_Within_Sex;
  SET class;
  BY Sex Age;

  IF FIRST.Sex THEN OUTPUT Undup_Sex;
  IF FIRST.Age THEN OUTPUT Undup_Age_Within_Sex;
RUN;

Undup_Sex:              SEX
                         F
```

```
                          M

Undup_Age_Within_Sex:  SEX     AGE
                        F      13
                        F      14
                        F      15
                        M      13
                        M      14
                        M      15
                        M      16
```

The following example uses both the FIRST. and LAST. variables to count observations by the Sex and by Age within Sex:

```
DATA Sex_Count(drop=Age Count_Age_Within_Sex)
     Age_Within_Sex_Count(drop=Count_Sex);
  SET class;
  BY Sex Age;

  IF FIRST.Sex THEN Count_Sex=0;
  IF FIRST.Age THEN Count_Age_Within_Sex=0;

  Count_Sex+1;
  Count_Age_Within_Sex+1;

  IF LAST.Sex THEN OUTPUT Sex_Count;
  IF LAST.Age THEN OUTPUT Age_Within_Sex_Count;
RUN;

Sex_Count:                SEX     Count_Sex
                           F       6
                           M       6

Age_Within_Sex_Count:     SEX  AGE   Count_Age_Within_Sex
                           F    13        2
                           F    14        2
                           F    15        2
                           M    13        1
                           M    14        2
                           M    15        2
                           M    16        1
```

Variables values in a dataset can either be unique or non-unique. For sorted variables with unique values, all FIRST.variable and LAST.variable temporary variables will be set to the value 1. In the result Age_Within_Sex_Count of the last example Sex is non-unique while Age is unique:

```
   SEX  AGE   FIRST.SEX LAST.SEX FIRST.AGE LAST.AGE
F     13       1         0         1         1
F     14       0         0         1         1
F     15       0         1         1         1
M     13       1         0         1         1
M     14       0         0         1         1
M     15       0         0         1         1
M     16       0         1         1         1
```

Primary key variables are typically used when sorting or indexing a SAS dataset. The primary keys may then be used to process data in groups, and to combine data across datasets, including interleaving, merging, and updating files.

COMBINING DATASETS

So far, the effects of the BY statement while reading a single dataset have been considered. In the DATA step the BY statement is often used for combining data, either by interleaving, match-merging, updating, or modifying. Modifying a dataset is an advanced form of updating and is beyond the scope of this paper.

INTERLEAVING DATASETS

When a BY statement is used with a SET statement that specifies two or more datasets, the DATA step reads the two or more files simultaneously, alternating sequentially between the files based on the lowest level groupings of the BY variables. This has the advantage of maintaining the sort order of the data from the datasets as they are jointly processed.

For example, suppose there are two datasets, one for males and one for females, and both are sorted on Age. They can be interleaved into a single dataset sorted on Age and Gender. Since the values of Sex are unique within each dataset, the variable may be included on the BY statement even though it is not explicitly sorted.

```
PROC SORT DATA=Sashelp.Class OUT=Boys;
  BY Age;
  WHERE Sex='M';
RUN;

PROC SORT DATA=Sashelp.Class OUT=Girls;
  BY Age;
  WHERE Sex='F';
RUN;

DATA Class;
  SET Boys Girls;
  BY Age Sex;
RUN;
```

Whereas the above example interleaved the data into sorted order on Sex within Age, the same two datasets could also be interleaved on Age within Sex:

```
DATA Class;
  SET Boys Girls;
  BY Sex Age;
RUN;
```

In this example, although the Girls dataset is specified after the Boys dataset in the SET statement, SAS reads all records from the Girls dataset first, according to the sort order of Sex. Without the BY statement the observations from the Boys dataset would be read first.

Using basic interleaving as a tool, the SAS programmer can perform some useful data processing techniques. Consider the case where the sum of a variable by a group is needed, attached back to the non-aggregated dataset. This will always require at least two passes of the data, but the processing efficiency and amount and complexity of code may vary considerably based on the programmer's approach. In 2003, Howard Shreier demonstrated how this interleaving of a dataset can produce summed values attached to the base dataset in his paper "Interleaving a Dataset with Itself: How and Why?" cited in the *References* below:

```
DATA Have;
  INFILE Cards;
  INPUT X Y $ @@;
CARDS;
1 A 2 A 3 A 4 A 5 B 6 B 7 B 8 B
;
RUN;

DATA Need;
  SET Have (IN=FirstPass)
      Have (IN=SecondPass);
  BY Y;
  IF FirstPass THEN DO;
    IF First.Y THEN SumX=0;
```

```
    SumX+X;
  END;
  ELSE IF SecondPass THEN OUTPUT;
RUN;
```

In this technique SAS reads each observation from the first BY group of Y, using the first reference to dataset Have (with the automatic IN variable FirstPass), and then reads the same BY group again from the second reference (with the automatic IN variable SecondPass). During the first pass of the BY group the variable X is summed into the variable SumX. Then, as the same BY group is read again in SecondPass, SAS is explicitly instructed to output the observation. Since the sum statement SumX+X implicitly retains the value of SumX, the values of X are accumulated in FirstPass and then retained and appended to each record for output during SecondPass. SumX is then explicitly reset to zero at the beginning of each BY group in FirstPass.

This demonstrates basic summation in conjunction with BY group processing in a very unique way, implementing code that is efficient, readable, and flexible. This example is a useful instance of a very important technique known as *look-ahead* processing, discussed fully in Toby Dunn and Chang Chung's paper "Retaining, Lagging, Leading, and Interleaving Data" cited in the *References* below.

MATCH-MERGING DATASETS

When a BY statement is used with a MERGE statement that specifies two or more datasets, the DATA step reads the two files simultaneously, merging observations from the two datasets based on the lowest level groupings of the BY variables. When merging multiple datasets, usually at least all but one of the datasets should be unique at the lowest level of the sort groupings. If this is case, the combined unique observations are merged with each matching observation in the non-unique dataset. The unique observations are duplicated across the non-unique observations.

The following Stock and Price datasets are sorted and match-merged on variable Item. The variable Item is unique in the Prices dataset but not unique in the Stock dataset, and so values of the unique variable Price from the Prices dataset is duplicated across matching values of Item in Stock.

```
TITLE                   ITEM
Castaways and Cutouts    CD
Picaresque               CD
Billy Liar               EP
The Tain                 Single

ITEM    PRICE
CD      $16.99
EP      $10.99
Single  $6.99

DATA Inventory;
  MERGE Stock Prices;
  BY Item;
RUN;

TITLE                   ITEM       PRICE
Castaways and Cutouts    CD        $16.99
Picaresque               CD        $16.99
Billy Liar               EP        $10.99
The Tain                 Single    $6.99
```

This technique is useful when a single variable changes over time, but the overall dataset does not. A major drawback of this method is that all of the values of Price for matching values of Item are overwritten. The UPDATE statement is a specialized form of MERGE for use when some values of the updating variable may be missing.

UPDATING DATASETS

The UPDATE statement is similar to the MERGE statement but only allows two datasets, a *master* dataset and a *transaction* dataset. UPDATE functions similarly to MERGE, but is used when missing transaction values are not wanted to overwrite existing values in the master dataset. In the UPDATE statement the master dataset is specified first and the transaction dataset second, and is followed by a BY statement with the sorted variables on which to update. As with MERGE, the DATA step reads the two datasets simultaneously, updating observations from the master dataset with observations from the transaction dataset based on the lowest level groupings of the BY

variables. When a transaction variable has a missing value, by default UPDATE does not overwrite the value in the master dataset, whereas the MERGE statement does.

The following master Inventory and transaction Price datasets are sorted and updated on variable Item. By default missing Price values do not overwrite Price values in matching Items of Inventory:

```
TITLE                    ITEM       PRICE
Castaways and Cutouts     CD        $16.99
Picaresque                CD        $16.99
Billy Liar                EP        $10.99
The Tain                  Single    $6.99

ITEM   PRICE
CD     $17.99
EP     .
Single $8.99

DATA Inventory;
  UPDATE Inventory Prices;
  BY Item;
RUN;

TITLE                    ITEM       PRICE
Castaways and Cutouts     CD        $17.99
Picaresque                CD        $17.99
Billy Liar                EP        $10.99
The Tain                  Single    $8.99
```

Following is a more advanced example where the UPDATE statement is used to “flatten” a dataset containing different variable data values for a key spread across several observations. The goal is to combine non-missing values into one record per unique key value. The master dataset structure is read, but the OBS=0 option stops the DATA step from reading any data from it. The same dataset is then updated as a transaction dataset, flattening non-missing observations for each unique key into one observation.

```
DATA Scores;
  INPUT ID Sat_Math Sat_Verbal Act_Comp;
CARDS;
8188 560 .    .
8188 .   540  .
8188 .   .    12
8189 660 .    .
8189 .   740  13
8189 .   .    .
;
RUN;

DATA Scores;
  UPDATE Scores(OBS=0) Scores;
  BY ID;
RUN;

ID    SAT_MATH  SAT_VERBAL  ACT_COMP
8188   560       540         12
8189   660       740         13
```

For a fuller explanation of SET, MERGE, and UPDATE see Andrew Kuligowski’s paper “How to Incorporate Old SAS Data into a New DATA Step, or “What is S-M-U?” cited in the *Suggested Reading* below.

DO-LOOP OF WHITLOCK (DoW)

Another more advanced DATA step technique that takes unique advantage of BY group processing is the DO loop of Whitlock, more commonly called the *DoW loop*. Several years ago, the respected SAS programmer Ian Whitlock

quietly answered a DATA step question on the SAS listserv. While the actual post was of little consequence, how he solved the question was not. Ian wrapped the SET statement inside a DO UNTIL loop and used the BY statement to control the loop. Later, the similarly renowned SAS programmer Paul Dorfman recognized the power that this construct brought to the SAS DATA step programming tool kit, and expounded upon its many uses until the construct became a commonly known SAS programming technique. Paul Dorfman's classic paper "The Magnificent Do" is cited in the *References* below.

There are many variations of the DoW, but this is the general form:

```
DATA ...;
 <Stuff done before break-event>;
  DO <Index Specs> UNTIL (Break-Event);
    SET A;
 <Stuff done for each record>;
  END;
 <Stuff done after break-event...>;
RUN;
```

As with Howard Shreier's interleaving technique, this construct is simple and easily coded, yet it has great power when combined with natural functionality of the DATA step.

In the usual DATA step, observations are read sequentially one at a time from the first observation to the last. In the DATA step, statements and functions operate on the variables and variable values and then observations are sequentially written out, either explicitly or implicitly. When BY group processing is used in this default manner, SAS implicitly keeps track of the data BY groups, and the programmer must explicitly instruct SAS using the FIRST. and LAST. automatic variables to execute statements and functions conditionally. Returning to our earlier look-ahead example, this DATA step sums the X variable values and outputs one observation per BY group.

```
DATA Have;
  INFILE Cards;
  INPUT X Y $ @@;
CARDS;
1 A 2 A 3 A 4 A 5 B 6 B 7 B 8 B
;
RUN;

DATA Need1(DROP=X);
  SET Have;
  BY Y;
  SumX+X;

  IF Last.Y THEN DO;
    OUTPUT;
    SumX=0;
  END;
RUN;

Y   SUMX
A    10
B    26
```

While this construct works, it is fairly complicated. It relies on the programmer coding statements before, during, and after each break event. FIRST. and LAST. BY variables must be used to explicitly control output statements and to explicitly set accumulation variables to zero between groups. In comparison, the DoW works with the natural execution of the DATA step by isolating what happens between two consecutive break events. Statements and functions acting on observation variables and values are placed plainly within the loop, and the implicit action of the DATA step resets calculated values to missing after each BY group. In the following example the break events are BY groups, but in other cases could be anything that triggers the DO loop to stop.

The following DoW example returns the same results as the more standard DATA step example above, but makes clearer use of the default actions of the datastep, and so is simpler to program and understand.

```
DATA Need2(DROP=X);
  DO UNTIL(LAST.Y);
    SET Have;
    BY Y;
    SumX = SUM(SumX, X);
  END;
RUN;

Y   SUMX
A    10
B    26
```

Here there are no statements before the DO loop, so the first action is the SET statement inside the loop. On each iteration of the Do loop the SET statement reads an observation into the DATA step. The values of X are then summed until the last observation of the BY group is reached. At the last observation of the BY group a break event is controlled by the automatic LAST. Variable and the program passes out of the loop. The program then reaches the end of the DATA step where the record is implicitly output to the dataset Need2. Finally, the control returns to the top of the DATA step and the DO loop instructs SAS to resume reading the next BY group. This process is repeated until there are no more observations in the input dataset.

Since there is no explicit output statement, SAS implicitly executes an output at the end of the DATA step. Also, by default SAS sets all variables created during the DATA step to missing after each implicit output, unless specified in a RETAIN statement, so there is no need to set the variable SumX back to zero for each BY group. In comparison, the first example required a RETAIN statement to bypass this default and then an assignment statement to reset the sum to zero for each new BY group. The DoW only passes the control to the end of the DATA step at each break of the BY groupings, and so uses the implicit output and resetting action of the DATA step to the programmer's advantage.

As an extension, multiple DoW loops can be used in tandem inside a DATA step. The following code appends an aggregated sum of the variable X within BY groups of Y just the same as in the previous look-ahead example.

```
DATA Need3;
DO UNTIL(Last.Y);
  SET Have;
  BY Y;
  SumX = SUM(SumX, X);
END;

DO UNTIL(Last.Y);
  SET Have;
  BY Y;
  OUTPUT;
END;
RUN;
```

Here the data is also read twice; the first time to get the value of X for each BY group, and a second time to output the observations with SumX attached. As with the previous DoW example, since end of the DATA step isn't reached until after a full BY group is processed by both DoW loops, the values of SumX are not reset before the second loop, and so the value of the summed variable is attached to each output record.

THE BY STATEMENT IN SAS PROCEDURES

So far the BY statement has been demonstrated in the SAS DATA step, but it is also of important use in SAS procedures. Nearly all SAS PROCs that process datasets allow for the BY statement with the same syntax as in the DATA step, except for the GROUPFORMAT option which is only available to the DATA step. Procedures that produce printed output, such as PROC PRINT, format printed output into groups depending on the BY groups specified. Procedures that summarize datasets, like PROC FREQ or PROC SUMMARY process the data in groups, sometimes as an alternative to other statements such as TABLES or CLASS.

THE PRINT PROCEDURE

The PRINT procedure is a straightforward, yet very useful SAS procedure. PROC PRINT writes a dataset's values in columnar table to an output destination, with the variable names or labels at the top of each column. Following are four examples of how the BY statement, and the related PAGEBY and SUMBY statements can be used within PROC PRINT. For a more full discussion of PROC PRINT see the *Suggested Reading* below.

In PRINT's most simple form the variables are selected with a VAR statement and printed to the output destination as columns in a single table.

```
PROC PRINT DATA=Class;
     VAR Sex Age Name Height Weight;
RUN;

OBS    SEX    AGE    NAME       HEIGHT    WEIGHT
  1     F      13    Alice       56.5       84.0
  2     F      13    Barbara     65.3       98.0
  3     F      14    Carol       62.8      102.5
etc.
```

If the values of certain variables are ordered, then a BY statement may be used create multiple tables, one for each of the lowest level of the BY groups. The BY group values are placed in a header row above each BY group table.

```
PROC PRINT DATA=Class;
     BY Sex Age;
     VAR Name Height Weight;
RUN;

Sex=F Age=13

OBS     NAME      HEIGHT    WEIGHT
  1    Alice       56.5       84
  2    Barbara     65.3       98

Sex=F Age=14

OBS    NAME     HEIGHT    WEIGHT
  3    Carol     62.8      102.5
  4    Judy      64.3       90.0
etc.
```

In conjunction with the BY statement, an ID statement may also be used to override the default header line. Instead of in the table header, the BY group values are prepended in columns to the left of the table in the first row of each BY group.

```
PROC PRINT DATA=Class;
     ID Sex Age;
     BY Sex Age;
     VAR Name Height Weight;
RUN;

SEX    AGE    NAME       HEIGHT    WEIGHT
 F      13    Alice       56.5       84.0
              Barbara     65.3       98.0

 F      14    Carol       62.8      102.5
              Judy        64.3       90.0
etc.
```

To additionally control paging of the output, a PAGEBY statement may be used in conjunction with the BY statement to force page breaks between each BY group table:

```
PROC PRINT DATA=Class;
     PAGEBY Sex;
     ID Age;
     BY Sex Age;
     VAR Name Height Weight;
```

```
RUN;

Sex=F Age=13

AGE     NAME      HEIGHT    WEIGHT
 13     Alice       56.5       84
 13     Barbara     65.3       98

<page break>

Sex=F Age=14

AGE     NAME      HEIGHT    WEIGHT
 14     Carol       62.8      102.5
 14     Judy        64.3       90.0
etc.
```

Finally, PROC PRINT allows summing of analysis variables with the SUMBY statement, in conjunction with the BY statement, to create sums for each BY group table:

```
PROC PRINT DATA=Class;
     BY Sex;
     ID Sex;
     SUMBY Sex;
     SUM Fees;
     VAR Name Age;
     FORMAT Fees DOLLAR12.2;
RUN;

SEX     NAME        AGE            FEES
 F      Alice        13            $4.62
        Barbara      13           $24.25
        Carol        14           $10.00
        Judy         14            $6.48
        Janet        15           $23.04
        Mary         15           $24.23
---                          ------------
 F                                $92.62
etc.
```

THE FREQ PROCEDURE

The FREQ procedure tabulates frequencies and categorical statistics on nominal or discrete variables, optionally weighted by a measure variable. Both printed tabulations and summary SAS datasets may be produced, as well as statistics either printed or output to a dataset. Cross-tabulations are requested with a TABLES statement, but levels of a tabulation for sorted variables may alternately be specified with a BY statement. One important difference is that if not all categories of a variable are present in a BY group then the PROC FREQ will not show rows or columns for that level of the variable in the table for that BY group, whereas if the variable is in the TABLE statement the row or column will be zero filled. Calculated statistics such as Chi-Square will be different based either on the missing or zero filled or missing columns. This example shows how missing levels of Age and Grade for females, not missing in the data for males, are zero filled by PROC FREQ:

```
PROC FREQ DATA=Class;
     TABLES Sex*Age*Grade/NOROW NOCOL NOPERCENT CHISQ;
RUN;

Table 1 of Age by Grade
Controlling for Sex=F

Frequency|       7|       8|       9|      10|      11|  Total
---------+--------+--------+--------+--------+--------+
      13 |      1 |      1 |      0 |      0 |      0 |      2
```

```
---------+--------+--------+--------+--------+--------+
      14 |      0 |      2 |      0 |      0 |      0 |      2
---------+--------+--------+--------+--------+--------+
      15 |      0 |      0 |      0 |      2 |      0 |      2
---------+--------+--------+--------+--------+--------+
      16 |      0 |      0 |      0 |      0 |      0 |      0
---------+--------+--------+--------+--------+--------+
Total           1        3        0        2        0        6

Statistics for Table 1 of Age by Grade
Controlling for Sex=F

Row or column sum zero. No statistics computed for this table.
Sample Size = 6
etc.
```

A comparable result is produced with a BY statement, except columns and rows with no counts are excerpted. This produces different statistical results:

```
PROC FREQ DATA=Class;
     BY Sex;
     TABLES Age*Grade/NOROW NOCOL NOPERCENT CHISQ;
RUN;

Sex=F
Table of Age by Grade

Frequency|       7|       8|      10|  Total
---------+--------+--------+--------+
      13 |      1 |      1 |      0 |      2
---------+--------+--------+--------+
      14 |      0 |      2 |      0 |      2
---------+--------+--------+--------+
      15 |      0 |      0 |      2 |      2
---------+--------+--------+--------+
Total           1        3        2        6

Statistics for Table of Age by Grade

Statistic                     DF       Value      Prob
------------------------------------------------------
Chi-Square                     4      8.0000    0.0916
Likelihood Ratio Chi-Square    4      9.3643    0.0526
Mantel-Haenszel Chi-Square     1      4.1667    0.0412
Phi Coefficient                       1.1547
Contingency Coefficient               0.7559
Cramer's V                            0.8165
etc.
```

THE SUMMARY OR MEANS PROCEDURE

As with the FREQ procedure, in PROC SUMMARY and the twin procedure MEANS the BY statement may be used as an alternate means to group variables, in this case as an alternative to the CLASS statement. In PROC SUMMARY the CLASS statement is used to build an n-dimensional summary of all permutations of the CLASS variable. For the three class variables A, B, and C, PROC SUMMARY calculates requested statistics for the overall data and all levels of A, B, C, A*B, A*C, B*C, and A*B*C.

```
PROC SUMMARY DATA=Class;
     CLASS Sex Age Grade;
     VAR Weight;
     OUTPUT MEAN=MeanWeight OUT=ClassMeans;
RUN;
```

OBS	SEX	AGE	GRADE	MEAN WEIGHT	_FREQ_
1		.	.	107.750	12
2		.	9	103.909	11
3		.	12	150.000	1
4		14	.	96.214	7
5		16	.	123.900	5
6		14	9	96.214	7
7		16	9	117.375	4
8		16	12	150.000	1
9	F	.	.	99.833	6
10	M	.	.	115.667	6
11	F	.	9	99.833	6
12	M	.	9	108.800	5
13	M	.	12	150.000	1
14	F	14	.	93.625	4
15	F	16	.	112.250	2
16	M	14	.	99.667	3
17	M	16	.	131.667	3
18	F	14	9	93.625	4
19	F	16	9	112.250	2
20	M	14	9	99.667	3
21	M	16	9	122.500	2
22	M	16	12	150.000	1

If the data are sorted, the sorted variable may be moved to a BY statement, but only permutations including that variable will be summarized. For example, if the data are sorted on A and A is specified in the BY statement rather than the CLASS statement then only statistics for A, A*B, A*C, and A*B*C are produced.

```
PROC SUMMARY DATA=Class;
     BY Sex;
     CLASS Age Grade;
     VAR Weight;
     OUTPUT MEAN=MeanWeight OUT=ClassMeans;
RUN;
```

OBS	SEX	AGE	GRADE	MEAN WEIGHT	_FREQ_
1	F	.	.	99.833	6
2	F	.	9	99.833	6
3	F	14	.	93.625	4
4	F	16	.	112.250	2
5	F	14	9	93.625	4
6	F	16	9	112.250	2
7	M	.	.	115.667	6

etc.

The PRINT, FREQ, and SUMMARY procedures were presented above as general examples for the use of the BY statement in SAS procedures. While the many other procedures in SAS are different in application, the function and syntax of the BY statement in them is generally similar. In this regard there is one procedure that stands apart from all other base SAS procedures, PROC SQL. The SQL procedure has its roots outside the SAS system and so is distinctive in its use of sorted data syntax and processing.

THE SQL PROCEDURE

PROC SQL is unique among SAS procedures in that it performs actions both similar to the DATA step and similar to summarizing procedures such as SUMMARY, TABULATE, and UNIVARIATE. PROC SQL also has an unique syntax that conforms to the SQL programming language common to most database applications. For an interesting discussion of the use of PROC SQL in SAS see Ian Whitlock's "PROC SQL - Is it a Required Tool for Good SAS Programming?" in the *Suggested Reading* below.

While there is no BY statement in PROC SQL, there is a GROUP BY statement that performs a similar function of ordered group processing. This GROUP BY statement affects any and all summary statistic functions regardless of

whether that summary function is in a subsequent select clause or having clause. SQL is also different than other procedures because if data are not sorted then the procedure will sort the data internally as needed by the GROUP BY statement.

One final return to our example where an aggregated value for a variable is wanted within BY groups of a dataset:

```
DATA Have;
  INFILE CARDS;
  INPUT X Y $ @@;
CARDS;
1 A 2 A 3 A 4 A 5 B 6 B 7 B 8 B
;
RUN;

PROC SQL;
CREATE TABLE Need AS
 SELECT *, SUM(X) AS SumX
   FROM Have
     GROUP BY Y;
QUIT;
```

Once again, the BY group summation variable SumX is appended to each observation of the base data.

CONCLUSION

The BY statement adds much power the programmer's toolkit. It has been shown how to use the BY statement to manipulate and summarize data within groups, to merge and interleave data on key variables, and many other tasks. Several methods were demonstrated to creatively apply the BY statement to control the SAS DATA step and procedures. Prudence was suggested when placing sorted variables in SAS procedure BY statements in lieu of TABLES or CLASS statements. When used in conjunction with proper data structure, some creative DATA step programming, and appropriate procedural usage, the BY statement is an indispensable processing tool that can both reduce the complexity and improve the efficiency of SAS programs.

REFERENCES

SAS 9.1.3 XP Platform
SAS Institute Inc., Cary, NC

SAS OnlineDoc 9.1.3 for the Web
SAS Institute Inc., Cary, NC

Dorfman, Paul, (2002), "The Magnificent Do," Proceedings of the Southeast SAS Users Group 2002 Conference (SESUG). (http://www.devenezia.com/papers/other-authors/sesug-2002/TheMagnificentDO.pdf)

Dunn, Toby and Chung, Chang Y., (2005), "Retaining, Lagging, Leading, and Interleaving Data," Proceedings of the Pharmaceutical Industry SAS Users Group 2005 (PharmaSUG2005). (http://changchung.com/download/retainLagLeadInterleave_draft.pdf)

Schreier, Howard, (2003), "Interleaving a Data Set with Itself: How and Why?" Proceedings of Northeast SAS Users Group Conference 2003 (NESUG). (http://www.nesug.org/html/Proceedings/nesug03/cc/cc002.pdf)

RECOMMENDED READING

For questions on BY group processing please search and post questions to:
SAS-L@LISTSERV.UGA.EDU or http://listserv.uga.edu/archives/sas-l.html, also available at http://groups.google.com/group/comp.soft-sys.sas/topics

There are many books on introductory database design and levels of normal forms to choose from, one popular title is: Stephens, Ryan K., and Ronald R. Plew, (2001), "Database Design," SAMS Publishing, Indianapolis, IN.

For more examples and a discussion of normal form in SAS:
Rhodes, Dianne Louise (2001), "Migrate to ORACLE? I need my SAS!" in the Proceedings of the Twenty-Sixth Annual SAS Users Group International (SUGI) Conference. (http://www2.sas.com/proceedings/sugi26/p126-26.pdf)

For a fuller explanation of SET, MERGE, and UPDATE see:

Kuligowski , Andrew T. (2004), “How to Incorporate Old SAS Data into a New DATA Step, or “What is S-M-U?” in the Proceedings of the Twenty-Ninth Annual SAS Users Group International (SUGI) Conference. (http://www2.sas.com/proceedings/sugi29/254-29.pdf)

For an excellent discussion of the PRINT procedure:
Delwiche, Lora D., and Slaughter, Susan J. (2005), “Turning Raw Data into Polished Reports” in the Proceedings of the Thirtieth Annual SAS Users Group International (SUGI) Conference. (http://www2.sas.com/proceedings/sugi30/265-30.pdf)

For a comprehensive explanation of PROC MEANS and PROC SUMMARY:
Karp, Andrew H. (2002), "Advanced Tips and Techniques with PROC MEANS" in the Proceedings of the Twenty-Seventh Annual SAS Users Group International (SUGI) Conference. (http://www2.sas.com/proceedings/sugi27/p018-27.pdf)

For a very illuminating look at the use of SQL in SAS:
Whitlock, Ian (2001), "PROC SQL - Is it a Required Tool for Good SAS Programming?" in the Proceedings of the Twenty-Sixth Annual SAS Users Group International (SUGI) Conference. (http://www2.sas.com/proceedings/sugi26/p060-26.pdf)

CONTACT INFORMATION

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

Paul Choate, Senior Programmer Analyst
California Department of Developmental Services
1600 9th Street, Sacramento, CA 95818
Work Phone: (916) 654-2160
E-mail: pchoate@dds.ca.gov

Toby Dunn, SAS Programming Specialist
Stat Works Inc.
San Antonio. TX
Work Phone: (512) 786-4982
E-mail: tobydunn@hotmail.com

Join the SAS-L! @ listserv.uga.edu or Google Groups