Performance Tips for Using the SAS® System on Personal Computers

Brett A. Knox
National Demographics & Lifestyles

ABSTRACT

Reducing the amount of time and disk space required to execute programs on the SAS® System for Personal Computers is a concern for all involved. Various methods for reducing processing time, programming time, and hard disk usage by using some of the new enhancements in Version 6.03 will be discussed.

This paper begins with the introduction of some of the new operators, statements, and Procedure step options available in Version 6.03 of the SAS System. The discussion of these new enhancements will hopefully allow SAS users to replace some of their traditional programming methods with these new enhancements.

The benchmark tests that follow were completed on an IBM PS/2 Model 60. This is a 286-class machine with a 70mb hard-drive, 1 mb of memory, and operates at 10mhz. The data sets that were used range from 10 observations to 20,000 observations, each containing 25 variables.

IN operator

The IN operator is a substitute for the logical operator - OR.

The syntax is as follows:

expression IN (value1,value2,...);

The IN operator compares the expression on the left to the values listed on the right. For example:

if zip in ('80200','80202','80204')
then aurora+1;

The above code is equivalent to:

if zip='80200' or zip='80202' or zip='80204'
then aurora+1;

WHERE statement

The WHERE statement allows you to select observations from an existing SAS data set based on one or more conditions specified in the whereexpression.

The syntax is as follows:

WHERE whereexpression;

The whereexpression can be an arithmetic or logical expression. As you will notice below, the WHERE statement has incorporated two new "English-like" operators that can be used as substitutes for traditional SAS operators. These include the Between-And and the Is Missing or Is Null operators. The IN operator can also be used. The following are valid and equivalent:

where '80000' <=zip<= '80099';
where zip between '80000' and '80099';
where zip is not missing;
where zip='80001' or zip='80003' or zip='80005';
where zip in ('80001','80003','80005');
The WHERE statement can be used as either a statement within the DATA and PROC step or as a data set option in a SET, MERGE, or UPDATE statement.

**** WHERE statement within the DATA step ****

data adams;
set us_data;
where county='ADAMS';

**** WHERE= data set option ****
data adams;
set us_data (where=(county='ADAMS'))
us_sales (where=(sales > 50000));

Within the PROC step, you cannot use the WHERE= data set option. However, you can use the WHERE statement.

**** Valid use of the WHERE statement ****
proc print data=us_data;
where county='ADAMS';

**** Invalid use of the WHERE statement ****
proc print data=us_data (where=(county='ADAMS'));

When you use a WHERE statement in the PROC step, the extraction of data that results, is the same as if you used a subsetting IF statement as your last statement in the DATA step that created the input data set.

You have probably concluded that the WHERE statement and the subsetting IF statement look very similar; however, these two statements operate in two different ways.

WHERE statement Operation:
* It is the first operation performed during execution of a SET, MERGE, or UPDATE statement.
* Observations are not brought into the Program Data Vector until the selection criteria is met.

IF statement Operation:
* Observations are first brought into the Program Data Vector, then kept or deleted based on the selection criteria.

For example, let’s say you want to read in a data set and only keep those observations in the western region. With the WHERE statement, only those observations in the western region will be present in the Program Data Vector (PDV). Whereas, with the subsetting IF statement, all observations are initially brought into the PDV, then only those observations in the western region are kept.

Because of the operational differences explained above, these two statements assign FIRST. and LAST. byvariable values differently. To wit:

WHERE statement:
* If both a WHERE statement and a BY statement are present in a DATA step, the FIRST. and LAST. variables are assigned values after the observations are selected.

IF statement:
* Whereas, if both a subsetting IF statement and a BY statement are present in a DATA step, the FIRST. and LAST. variables are assigned values before the desired observations are selected.

The following sets of code are various data extraction methods. Immediately following the code is a benchmark table that shows the processing times associated with each method.

**** Technique 1 ****
data west;
set us_data (keep = state zip pop);
if state in ('CO', 'UT', 'AZ', 'NM');

**** Technique 2 ****
data west;
do until (eof);
set us_data (keep = state zip pop) end = eof;
if state in ('CO', 'UT', 'AZ', 'NM') then output;
end;
stop;

**** Technique 3 ****
data west;
set us_data (keep = state zip pop);
where state in ('CO', 'UT', 'AZ', 'NM');

As you can see, the processing times associated with Technique 1 and Technique 2 are identical. The reason both techniques were shown is because on the mainframe, Technique 2 has been shown to be more efficient than Technique 1 when doing simple data extraction. However, on the P.C. there is no improvement. The WHERE statement begins to show substantial improvement after 1,000 observations. At 10,000 observation we can see a 23.8% improvement. As the number of observations increase we will see an even greater improvement.

The WHERE statement should not be looked upon as a single statement improvement. You need to look at your overall program design to see where you can eliminate DATA steps by using a WHERE statement inside your PROC steps. This is where the real benefits will be derived from the use of the WHERE statement.
PROC SORT - TAGSORT option

TAGSORT operation:

- Only the variables listed in the BY statement (called Sort keys) and the observation number of each record are actually sorted.
- When the sort has been completed, the Sort keys and the observation numbers (together, they are referred to as Tags) are used to link to the input data set.
- The input data set is then retrieved in sorted order according to the order in which the Tags were positioned.

The syntax is simply:

```sas
proc sort tagson;
by region state;
```

The following are a number of methods for sorting a data set:

- **Technique 1**:  proc sort data = us_data out = us_sales; by region name;
- **Technique 2**:  proc sort data = us_data (keep = region name sales) out = us_sales; by region name;
- **Technique 3**:  proc sort data = us_data lagsort out = us_sales; by region name;
- **Technique 4**:  proc son data = us_data (keep = region name sales) tagsort out = us_sales; by region name;

<table>
<thead>
<tr>
<th>Obs</th>
<th>Read (Tech. 1)</th>
<th>Read (Tech. 2)</th>
<th>Read (Tech. 3)</th>
<th>Read (Tech. 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2 sec.</td>
<td>2 sec.</td>
<td>3 sec.</td>
<td>2 sec.</td>
</tr>
<tr>
<td>100</td>
<td>4 sec.</td>
<td>2 sec.</td>
<td>3 sec.</td>
<td>2 sec.</td>
</tr>
<tr>
<td>1,000</td>
<td>39 sec.</td>
<td>10 sec.</td>
<td>27 sec.</td>
<td>9 sec.</td>
</tr>
<tr>
<td>10,000</td>
<td>8.4 min.</td>
<td>2.2 min.</td>
<td>7.1 min.</td>
<td>6.0 min.</td>
</tr>
<tr>
<td>20,000</td>
<td>21.8 min.</td>
<td>4.5 min.</td>
<td>25.9 min.</td>
<td>22.3 min.</td>
</tr>
</tbody>
</table>

As shown above, Technique 3 (Tagsort option) is actually faster than Technique 1 until you reach 20,000 observations. When using the KEEP= data set option, Technique 4 (Tagsort option) is significantly slower than Technique 2. However, there is a performance trade-off that must be addressed between available hard-disk space and processing time.

The reason I showed you the PROC SORT statements with the KEEP= data set option is to make you aware that this option is available in the PROC step. For example, if you only want to print out a few variables from a large data set in a particular order, do not bring every variable from the data set into the sort. Use the KEEP= data set option and bring into the sort only those variables that you want printed. The savings in processing time is remarkable (see the above benchmark table). However, when using this option, be sure to specify the OUT= option of PROC SORT, otherwise, after the sort has completed, the original data set will only contain those variables specified in the KEEP= data set option.

PROC FORMAT - INVALUE statement

The INVALUE statement allows you to define an informat that converts an incoming variable's value into a different value.

**Character informat:**

- When the first character of the name is a dollar sign and the value is enclosed in quotes, it returns a character value.

**Numeric informat:**

- Specifying a numeric informat produces a numeric value.

**Keywords:**

- **SAME**: This tells the SAS system that the values specified in the informat range should be stored as they appear.
- **ERROR**: This causes the SAS system to treat values in the informat range as invalid data. The data line is printed along with a warning message.

To illustrate the use of this statement:

```sas
proc format;
invalue $model 'D1101' = 'D1100'
'D1102' = 'D1100'
'D1105' = 'D1100'
'D1200'- 'D1299' = SAME
OTHER = ERROR;
data file_1;
infile in1;
input mod_num $model.;
run;
```

When mod_num's value is read from IN1, its value may be changed depending on, whether or not its value matches those values stated in the $model informat. For example, if mod_num's value is D1102, its value will be changed to D1100. If mod_num's value is D1250, it will not be changed.

The next thing I want to do is show you how to incorporate some of these new statements so that more efficient code can be produced. Let's say you want to produce a report that shows the total Sales dollars for each of your sales people in every region of the US, but you only want to see those Sales people that grossed between $50,000 & $250,000. The traditional way would be to:
**Least efficient method of solving the problem:**

```sas
data us_sales;
set us_data;
if 50000 <= sales <= 250000;
proc sort data = us_sales;
by region name;
proc print;
by region;
title1 'Regional Salesperson Report';
title2 'For Salespeople Grossing Between';
title3 '$50,000 and $250,000';
```

**** The following code results in a substantial reduction in processing time: ****

```sas
proc sort data = us_data (keep = region name sales)
out = us_sales;
where sales between 50000 and 250000;
by region name;
```

PROC PRINT;
by region;
title1 'Regional Salesperson Report';
title2 'For Salespeople Grossing Between';
title3 '$50,000 and $250,000';

The above code has the following advantages over the traditional method of coding:

1) the DATA step is eliminated.

2) the WHERE statement is more efficient than a subsetting IF statement.

3) the KEEP= data set option was incorporated into the PROC SORT.

4) Depending upon the number of observations, the elimination of the DATA STEP reduced the overall processing time by as much as 65%.

**SAS Functions**

Since hard-disk space is a limited resource on Personal Computers, you need to be aware that SAS functions return a pre-specified variable length. A variable with a length that is larger than necessary is a waste of valuable space. Use a LENGTH statement within the DATA step when you are creating a variable via a SAS function. This will reduce both your processing time and your hard-disk usage. For example, if you are using the ZIPSTATE function to translate zip code values to state abbreviations, place a LENGTH statement with a value of $2 inside the DATA step. Otherwise, the SAS system will return a two character state abbreviation with a length of $200. If your data set has 10,000 observations, this will save you 1.8 Mb of hard-disk space! Listed below are a number of common SAS functions along with the lengths that they return:

<table>
<thead>
<tr>
<th>SAS Function</th>
<th>Variable Type</th>
<th>Length Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAN</td>
<td>CHAR</td>
<td>200</td>
</tr>
<tr>
<td>ZIPSTATE</td>
<td>CHAR</td>
<td>200</td>
</tr>
<tr>
<td>ZIPSTATE</td>
<td>CHAR</td>
<td>200</td>
</tr>
<tr>
<td>REPEAT</td>
<td>CHAR</td>
<td>200</td>
</tr>
<tr>
<td>COMPRESS</td>
<td>CHAR</td>
<td>Length of the</td>
</tr>
<tr>
<td>SUBSTR</td>
<td>CHAR</td>
<td>Source</td>
</tr>
<tr>
<td>TRANSLATE</td>
<td>CHAR</td>
<td>Variable</td>
</tr>
<tr>
<td>REVERSE</td>
<td>CHAR</td>
<td>&quot;</td>
</tr>
<tr>
<td>UPCASE</td>
<td>CHAR</td>
<td>&quot;</td>
</tr>
<tr>
<td>TRIM</td>
<td>CHAR</td>
<td>&quot;</td>
</tr>
<tr>
<td>INDEXC</td>
<td>NUM</td>
<td>8</td>
</tr>
<tr>
<td>LENGTH</td>
<td>NUM</td>
<td>8</td>
</tr>
</tbody>
</table>

**General DATE Functions**

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPRESS</td>
<td></td>
</tr>
<tr>
<td>SUBSTR</td>
<td></td>
</tr>
<tr>
<td>TRANSLATE</td>
<td></td>
</tr>
<tr>
<td>REVERSE</td>
<td></td>
</tr>
<tr>
<td>UPCASE</td>
<td></td>
</tr>
<tr>
<td>TRIM</td>
<td></td>
</tr>
<tr>
<td>INDEXC</td>
<td></td>
</tr>
<tr>
<td>LENGTH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CONCLUSION**

In conclusion, by implementing some of the new operators, statements, and Procedure step options available in Version 6.03, your programming time and processing time should be reduced. However, you must look at your overall program design in the early stages of development, before you can really see the benefits derived from these new enhancements.

**ACKNOWLEDGEMENTS**

The author would like to thank Amy Vicioso and Joe McCormick for their valuable assistance and advice.

**REFERENCES**


SAS is a registered trademark of the SAS Institute, Inc.: Cary, NC, USA.

IBM is a registered trademark of International Business Machines Corporation, Armonk, NY, USA.