

Programming with Efficiency, Techniques for Large Sequential Datasets on a Multi-User System

David L. Wilson, Regence Blue Cross Blue Shield of Oregon, Portland, Oregon

INTRODUCTION

Programmers who were trained with the classical mainframe programming languages (e.g. COBOL) do not always receive much training in using the SAS system to increase their productivity. Techniques that were developed for 3rd generation languages may not be the best way to code in the SAS language. This paper points out some common programming structures encountered while debugging SAS code, and demonstrates the efficiency gains that could be realized by using more efficient structures. Making your programs more efficient can increase your productivity, especially when working with large files with over 1 million records on batch processing systems. Efficiency will be measured in CPU seconds, since this is a common way of charging back computer time to departments. Since this paper is based upon coding techniques, a sample project will be described and the code to complete the project will be presented.

THE PROJECT

Select all of the active female members, ages 50-70 with birthdays during the month of June. Pull claims to determine how many of these women have had mammograms processed during the first half of the year. This is an example of a typical project at a medical insurance company that has quality improvement projects. The results may be used for an internal report or used to send reminder notices to the member. The project can be logically divided into three parts.

- **Part I** - Read the members file and select the members who meet the criteria:
- **Part II** - Read the year-to-date claims files to get the claims experience.
- **Part III** - Merge the two files together in preparation for reporting.

PART I - READING THE MEMBERS FILE

Standard Code

A typical dataset would be written as follows.

```
DATA ONE;
  INFILE MEMBERS;
  INPUT      L_NAME      $20.
```

```
F_NAME      $10.
CONTRACT    $9.
REL          $2.
MEMBERNO    $2.
DOB         YYMMDD6.
SEX         $1.
ADDR_1      $23.
ADDR_2      $23.
ADDR_3      $26.
COV_MED     $1.
TERM_DT     $6.
PRODUCT     $3.
PCP         $12. ;
```

```
IF SEX = 'F';
IF MONTH(DOB) = 6;
IF (1998-70) <= YEAR(DOB) <= (1998-50);
IF COV_MED = 'M';
IF TERM_DT > '980630';
```

2962760 records were read from the infile members.
The data set WORK.ONE has 3817 observations.
The DATA statement used 153.28 CPU seconds.

After moving all of the variables from the input buffer, five conditions are tested to determine if the record should be kept. Since only 3817 records are kept from the 2,962,760 records read, a lot of CPU time is spent in moving variables into records that are then discarded.

Efficient Code

A more efficient dataset using two input statements using a trailing @ sign would significantly reduce the amount of CPU time needed to process the membership file.

```
DATA ONE;
  INFILE MEMBERS;
  INPUT      DOB          YYMMDD6.
             SEX          $1.
             COV_MED      $1.
             TERM_DT      $6.    @ ;
```

```
IF SEX = 'F';
IF MONTH(DOB) = 6;
IF (1998-70) <= YEAR(DOB) <= (1998-50);
IF COV_MED = 'M';
```

Core Skill Development

```
IF TERM_DT > '980630';
```

```
INPUT      L_NAME      $20.  
           F_NAME      $10.  
           CONTRACT    $9.  
           REL          $2.  
           MEMBERNO    $2.  
           ADDR_1      $23.  
           ADDR_2      $23.  
           ADDR_3      $26.  
           PRODUCT     $3.  
           PCP          $12. ;
```

2962760 records were read from the infile members.
The data set WORK.ONE has 3817 observations.
The DATA statement used 69.85 CPU seconds.

In this example of efficient code we first input the variables that are needed to test the conditions for keeping the record. The trailing @ sign holds the record in the input buffer for the next INPUT statement. If the record meets all of the conditions of the subsetting IF statements, the program then inputs the remaining variables. The program reduces the number of CPU seconds from 153 down to 70. Further reductions in CPU time can be realized by using multiple input statements with the trailing @ sign.

More Efficient Code

A more efficient dataset using three input statements was determined by empirical methodology.

```
DATA ONE;  
  INFILE MEMBERS;  
  INPUT      SEX        $1.  
           COV_MED      $1.    @;  
  
  IF SEX = 'F';  
  IF COV_MED = 'M';  
  
  INPUT      TERM_DT    $6.    @ ;  
  IF TERM_DT > '980630';  
  
  INPUT      DOB         YYMMDD6. @;  
  IF MONTH(DOB) = 6;  
  IF (1998-70) <= YEAR(DOB) <= (1998-50);  
  
  INPUT      L_NAME      $20.  
           F_NAME      $10.  
           CONTRACT    $9.  
           REL          $2.  
           MEMBERNO    $2.  
           ADDR_1      $23.  
           ADDR_2      $23.
```

```
ADDR_3      $26.  
PRODUCT     $3.  
PCP          $12. ;
```

2962760 records were read from the infile members.
The data set WORK.ONE has 3817 observations.
The DATA statement used 29.81 CPU seconds.

This version of the dataset takes approximately 20% of the CPU seconds of the original version (30 vs. 153). This technique is especially important when a lot of variables are being input into a SAS dataset. If the job is run as part of a regular process, the time savings can be very important during the year. The CPU seconds that you save on your job are now available for other jobs running on the system. Depending upon the CPU second restrictions placed on the batch queues, this may enable the program to be submitted to run in a queue with a shorter line.

PART II - READING THE CLAIMS FILE

Standard Code

The following code uses techniques that are very common among programmers trained in legacy languages.

```
DATA TWO;  
  INFILE CLAIMS;  
  INPUT      CONTRACT    $9.  
           MEMBERNO     $2.  
           TYPE_CD       $5.  
           INCURDT       ZD8.  
  
           @285 DIAG1_3    $3.  
           @465 DIAG2_3    $3.  
           @470 DIAG3_3    $3.  
           @475 DIAG4_3    $3.  
           @285 DIAG1_4    $4.  
           @465 DIAG2_4    $4.  
           @470 DIAG3_4    $4.  
           @475 DIAG4_4    $4.  
           @285 DIAG1_5    $5.  
           @465 DIAG2_5    $5.  
           @470 DIAG3_5    $5.  
           @475 DIAG4_5    $5. ;
```

```
*** TEST FOR MAMMOGRAPHY ***;  
IF TYPE_CD = '76090' OR  
TYPE_CD = '76091' OR  
TYPE_CD = '76092' OR  
DIAG1_3 = '174' OR  
DIAG2_3 = '174' OR  
DIAG3_3 = '174' OR
```

```

DIAG4_3 = '174'      OR
DIAG1_3 = '217'      OR
DIAG2_3 = '217'      OR
DIAG3_3 = '217'      OR
DIAG4_3 = '217'      OR
DIAG1_4 = '2330'     OR
DIAG2_4 = '2330'     OR
DIAG3_4 = '2330'     OR
DIAG4_4 = '2330'     OR
DIAG1_4 = '7938'     OR
DIAG2_4 = '7938'     OR
DIAG3_4 = '7938'     OR
DIAG4_4 = '7938'     OR
DIAG1_4 = 'V103'     OR
DIAG2_4 = 'V103'     OR
DIAG3_4 = 'V103'     OR
DIAG4_4 = 'V103'     OR
DIAG1_4 = 'V761'     OR
DIAG2_4 = 'V761'     OR
DIAG3_4 = 'V761'     OR
DIAG4_4 = 'V761'     OR
DIAG1_5 = '19881'    OR
DIAG2_5 = '19881'    OR
DIAG3_5 = '19881'    OR
DIAG4_5 = '19881'    OR
DIAG1_5 = '61172'    OR
DIAG2_5 = '61172'    OR
DIAG3_5 = '61172'    OR
DIAG4_5 = '61172'    ;

```

7627150 records were read from infile CLAIMS.
The data set WORK.TWO has 134434 observations.
The DATA statement used 378.54 CPU seconds.

Because the programmer needed to check the diagnosis codes against strings that were 3, 4 and 5 characters in length, they read in each diagnosis code three times and used one long statement to check all possible combinations. All of the mammograms for all members are stored in the data step (N=134,434).

Efficient Code

Using a format built for doing a table look up , and rewriting the selection statements will produce the following code that is both more efficient and more elegant.

```

DATA TBL_LOOK;
  SET ONE;
  KEEP START LABEL FMTNAME;
  START = CONTRACT !! MEMBERNO;
  LABEL = 'FOUND MBR';
  FMTNAME = '$MBRFMT';

```

```
PROC FORMAT CNTLIN=TBL_LOOK;
```

```

DATA TWO;
  INFILE CLAIMS;
  INPUT              CONTRACT    $9.
                    MEMBERNO    $2. @;

```

```

  IF PUT ((CONTRACT !! MEMBERNO),
$MBRFMT.) = 'FOUND MBR';

```

```

INPUT              TYPE_CD    $5.
                    INCURDT    ZD8.
                    DIAG1_5    $5.
                    DIAG2_5    $5.
                    DIAG3_5    $5.
                    DIAG4_5    $5. ;

```

```

*** TEST FOR MAMMOGRAPHY***;
IF TYPE_CD IN ('76092','76091','76090') OR
DIAG1_5 IN: ('V761','61172','7938','V103','174',
             '217','2330','19881') OR
DIAG2_5 IN: ('V761','61172','7938','V103','174',
             '217','2330','19881') OR
DIAG3_5 IN: ('V761','61172','7938','V103','174',
             '217','2330','19881') OR
DIAG4_5 IN: ('V761','61172','7938','V103','174',
             '217','2330','19881') ;

```

7627150 records were read from infile CLAIMS.
The data set WORK.ONE has 1817 observations.
The DATA statement used 108.29 CPU seconds.

This section of code has several features that increase the efficiency of the code and also make the code easier to read and maintain. The data step TBL_LOOK takes the member records from Part I, and builds a file to produce a format that is used as a table-look-up in datastep TWO. The CONTRACT and MEMBERNO variables are used to identify an individual in this system. Only records (n=1817) for the members identified in DATA ONE are kept, reducing the subsequent processing requirements.

Using the IN and : operators (see SAS documentation) reduces the number of variables and lines of code in the program. Arranging the tested values in descending frequency of occurrence in the file will also reduce the number of comparisons made before the Boolean statement evaluates to TRUE. This data step requires only 28% of the CPU time of the standard code. Fewer records are passed to the next step, thus reducing processing time downstream.

PART III - MERGING THE FILES TOGETHER

The code is the same for the project from this point forward. The efficient code has an advantage in that only claims for the members identified in step ONE are being passed forward. If a more common procedure was being pulled (e.g. Office Visits), then the programmer might be faced with the prospect of finding space to sort several million records, most of which would be discarded during the merge. In this example there is not much savings between sorting 1,817 records or 134,434 records. The code to sort and merge is straightforward and not presented here. The total time for processing all of the steps of the standard code was 535 seconds versus 138 seconds for the efficient code which is a significant reduction in CPU time. Note that the total elapsed time may not be different due to the slow nature of tape input.

CONCLUSIONS

Programmers can save significant CPU processing time in INPUT data steps by using the trailing @ sign to input the variables used for selecting records before they input the remaining variables. Using the IN and : operators can reduce the complexity of Boolean statements, which make the code easier to read and maintain.

CONTACT INFORMATION

David L. Wilson
Regence Blue Cross Blue Shield of Oregon
200 W. Market Street, E8A
Portland, OR 97201
Phone: 503-225-6887
E-mail: dwilson851@aol.com