

Introduction to efficiency techniques in SAS programming

Van Holle Lionel

Introduction

There are two aspects of efficiency:

- Efficient use of human resources
- Efficient use of computing resources:
elapsed real time (CPU, I/O,...), RAM and
disk storage

Unefficiency can have an impact on time
and/or « numeric resources ».

Efficiency techniques are more and more crucial for several reasons:

- Data grow larger in every pharmaceutical field: clinical trials, pharmacovigilance, pharmacoeconomy, epidemiology,...
- Development of platform activities with limited connection.
- Strong negative association between company's benefits and delays due to inefficiency (in every field).

Being efficient in terms of computing resources is a case-by-case situation depending on the limiting factor:

- time? (deadlines/routine procedures)
- RAM? (PC/server resources)
- disk space?

Focus on general techniques which will improve efficiency at every level.

Major efficiency problems

1. Misunderstanding of the requirements
2. Syntax/Logic errors
3. Carrying useless information along all the process
4. Using non optimal SAS code according to the situation
5. Storing the information in a non optimal way.
6. Performing repetitive tasks manually

1. Misunderstanding of the requirements

Major source of waste of time because it implies re-doing everything 😞

Solutions

- Pre discussion before starting the programming.
- In the exploratory field, always ask yourself « Let's suppose I have done it, would it really help? »

2. Syntax/logic errors

SAS does check the syntax at every data step but not globally.

Solutions:

- Use of the options `obs=0` in data steps & `inobs=0` in PROC SQL in order to debug the code globally. It takes then several seconds to detect syntax errors.
- Use of subsets in order to detect logic errors without having to process all data.

```
proc sql noprint;  
  create table subset as  
  select distinct member, min(min_dt) as min_dt  
  from inlib.diag  
  where substr(diag,1,3) eq "V30"  
  group by member  
  order by member, min_dt;  
quit;  
  
data fdata.perinatal_subset;  
  merge subset(in=in)  
          inlib.enrollment(keep=member der_yob der_sex mxce_fst);  
  by member;  
  if in;  
run;
```

```
90 proc sql noprint;
91   create table subset as
92   select distinct member, min(min_dt) as min_dt
93   from inlib.diag
94   where substr(diag,1,3) eq "V30"
95   group by member
96   order by member;
97 quit;
```

NOTE: Table WORK.SUBSET created, with **872709** rows and 2 columns.

NOTE: PROCEDURE SQL used (Total process time):

real time **15:20.53**
cpu time **2:11.79**

```
98 data fdata.perinatal_subset;
99   merge subset(in=in)
100     pharm.enrollment(keep=member der_yob der_sex mxce_fst mxce_lst);
101   by member;
102   if in;
103 run;
```

NOTE: There were **872709** observations read from the data set WORK.SUBSET.

NOTE: There were **32395035** observations read from the data set PHARM.ENROLLMENT.

NOTE: The data set FDATA.PERINATAL_SUBSET has 872709 observations and 8 variables.

NOTE: DATA statement used (Total process time):

real time **4:25.61**
cpu time **4:18.73**

```
%macro SAS_code(check=);
```

```
%let cdtdata=; %let cdtsql=;
```

```
%if &check eq 1 %then %do;
```

```
  %let cdtdata=obs=0;
```

```
  %let cdtsql=inobs=0;
```

```
%end;
```

```
proc sql noprint &cdtsql;
```

```
  create table subset as
```

```
  select distinct member, min(min_dt) as min_dt
```

```
  from inlib.diag
```

```
  where substr(diag,1,3) eq "V30"
```

```
  group by member
```

```
  order by member, min_dt;
```

```
quit;
```

```
data fdata.perinatal_subset;
```

```
  merge subset(in=in)
```

```
          inlib.enrollment(&cdtdata keep=member der_job der_sex mxce_fst);
```

```
  by member;
```

```
  if in;
```

```
run;
```

```
%mend;
```

```
%macro SAS_code(check=);  
  
%let cdtdata=; %let cdtsql=; %let ext=;  
  
%if %lowercase(&check) eq "syntax" %then %do;  
  %let cdtdata=obs=0; %let cdtsql=inobs=0;  
%end;  
%else %if %lowercase(&check) eq "logic" %then %do;  
  %let ext=_s;  
%end;
```

```
proc sql noprint &cdtsql;  
  create table subset as  
  select distinct member, min(min_dt) as min_dt  
  from inlib.diag&ext  
  where substr(diag,1,3) eq "V30"  
  group by member  
  order by member, min_dt;  
quit;
```

```
data fdata.perinatal_subset;  
  merge subset(in=in)  
         inlib.enrollment&ext(&cdtdata keep=member der_yob der_sex mxce_fst);  
  by member;  
  if in;  
run;  
  
%mend;
```

- Advantages:
 - Syntax errors are checked in virtually no time
 - Logic errors can be checked quickly after processing the code on a small sample.
 - Adjustments on the layout output can be performed on the results from the sample
 - Code on entire data can be run during night once syntax/logic/layout problems have been solved

- Small constraints

- Use of a template for the conditional use of the macros
- Small adaptation to the original code
- Creation of a consistent sample across all original DBs

The bigger the original data are the higher the ratio advantage/constraints is.

3. Useless information/steps

- Keeping useless variables/rows
- Using longer character variables than needed
 - > higher running time & storage space

Solutions

- Use of the DROP=, KEEP= WHERE= options and LENGTH statement
- Use of formats
- Create multiple outputs from one pass of the input.

4. Non optimal code

According to the situation time, space or computer resources have to be spared

Solutions:

- Ability to reach objective by various methods
- Know the advantages associated to the various methods.

Subsetting Observations from Large SAS® Data Sets

```
proc sql noprint;  
  create table subset as  
  select distinct member, min(min_dt) as min_dt  
  from inlib.diag  
  where substr(diag,1,3) eq "V30"  
  group by member  
  order by member, min_dt;  
quit;
```

NOTE: There were **872,709** observations read from the data set WORK.SUB SET.

```
data fdata.perinatal_subset(drop=estring);  
  merge subset(in=in)  
         inlib.enrollment(keep=member der_yob der_sex mxce_fst);  
  by member;  
  if in;  
run;
```

NOTE: There were **32,395,035** observations read from the data set PHARM.EN ROLLMENT.

Available methods for subsetting observations

Christopher J. Bost, MDRC, New York, NY

WITH A MERGE

- Pros: easy to use; well-known technique
- Cons: Data set LARGE must be in sort order, sorted with PROC SORT, or indexed

SORT: 2 hr, 17 min

MERGE: 0 hr, 20 min

WITH A FORMAT

- Pros: speed; data set LARGE does not have to be sorted
- Cons: requires multiple steps; not an intuitive or obvious technique

0 hr, 17 min

WITH PROC SQL

- Pros: easy to use; well-known technique; data set LARGE does not have to be sort
- Cons: SAS may run PROC SORT in the background

0 hr, 16 min

WITH DATA STEP HASH OBJECT

- Pros: speed; efficiency; data set LARGE does not have to be sorted.
- Cons: hash table must fit in memory; keys must be unique

0 hr, 17 min

– APPENDING

- If variables in both datasets have same length and type then use PROC APPEND for appending instead of the SET statement.

– DATA SELECTION

- WHERE clause is more efficient than an IF condition (no reading in PDV) – original variables
- When constructing a chain of AND-ed conditions specify the most restrictive values first.
- Do use the ELSE if you have to use the IF statement

– Use of the SAS function & automatic variables

– Avoid unnecessary procedure invocation (multiple reports can be created with SQL, FREQ, TABULATE,... procedures)

- For more information about specific efficiency questions, please look at « Advanced SAS Programming and Efficiency techniques »:
- Simulations to compare CPU, I/O, memory between different methods
- Results for different OS: OS/390, Windows, UNIX

5. Storing the information

- Store large SAS datasets in the compressed format (COMPRESS=)
- Store the SAS dataset sorted in the order which will be the most needed in the future
- Store intermediate/summary information to facilitate identification & retrieval of subset populations
- Envisage index creation for most used variables

6. Repetitive tasks

Running same program every morning,
Running same initialisation program for a
given project, ...

Solutions:

- Practical tips to customize a SAS session
- Performing delayed run

1. Creation of a SAS shortcut by project.
2. This shortcut can be customized to run automatically an initialization program
 1. Right Click on the shortcut
 2. Properties
 3. Go to « Target » put at the end:
 1. -autoexec "C:\Myproject\program\init.sas"
 2. -sasinitialfolder "C:\Myproject\program"

Advantages: Afterwards, one single click to initialize everything and have a quick access to your project programs.

Practical Tips to Customize a SAS® Session

Xiaohui Wang, Beilei Xu, Merck & Co.

Delay the execution of a program until tomorrow 6 a.m. in order to have the results at 8 a.m. right after your hot chocolate:

```
data _null_;  
    slept=wakeup("24FEB2010:06:00:00"dt);  
run;
```

* SAS code which has to be run in the morning *;

No Task Before Its Time: Schedule Your Jobs With Robot Code

Charles R. vanWynbergen, SunTrust Banks Inc., USA

QUESTIONS?