

# DATA Step

## Are there any efficiency benefits to using the KEEP= dataset option?

One of the largest 'costs' in processing any code is the I/O (Input / Output), the physical transfer of data from and to disk storage. Significant savings can be made to I/O, and therefore to processing time, by reducing the amount of data being transferred.

The following macro program can be used to generate a dataset with the variables A1-A20 through to Z1-Z20 and 1,024,000 observations.

```
%macro makebig ;
  %local i j k m ;
  data largeds ;
  %do i = 1 %to 1000 ;
    %do j = 1 %to 20 ;
      %do k = 65 %to 90 ;
        %sysfunc(byte(&k))&j = %sysfunc(rannorm(0)) ;
      %end ;
    %end ;
  %end ;
  output ;
  %end ;
  run ;
  %do m = 1 %to 10 ;
    proc append base = largeds
      data = largeds
    ;
    run ;
  %end ;
%mend makebig ;
%makebig
```

### Efficiencies in DATA Step

Processing all of the variables and all of the variables in a DATA step:

```
data lds2 ;
  set largeds ;
run ;
```

generates the following processing times in the LOG:

NOTE: There were 1024000 observations read from the data set WORK.LARGE  
EDS.

NOTE: The data set WORK.LDS2 has 1024000 observations and 520 variable  
Page 1 / 5

# DATA Step

S.

NOTE: DATA statement used (Total process time):

real time	24.40 seconds
cpu time	11.57 seconds

Adding the KEEP= dataset option to the DATA Statement restricts the Output to select only the variables A1-A20 in the output dataset:

```
data lds3 (keep = a1-a20) ;  
    set largeds ;  
run ;
```

NOTE: There were 1024000 observations read from the data set WORK.LARGEDS.

NOTE: The data set WORK.LDS3 has 1024000 observations and 20 variables  
.

NOTE: DATA statement used (Total process time):

real time	5.80 seconds
cpu time	5.58 seconds

For a range of variables with a common prefix the colon modifier can also be used (keep = a:) to select all variables which 'begin with' A.

Using the KEEP= dataset option on the SET statement restricts the variables being processed at Input (and therefore also on Output), generating a more efficient process, as can be seen in the LOG:

```
data lds4 ;  
    set largeds (keep = a:) ;  
run ;
```

NOTE: There were 1024000 observations read from the data set WORK.LARGEDS.

NOTE: The data set WORK.LDS4 has 1024000 observations and 20 variables  
.

NOTE: DATA statement used (Total process time):

real time	2.13 seconds
-----------	--------------

# DATA Step

cpu time

2.13 seconds

## Efficiencies in PROC SQL

The same principles can also be applied in the SQL Procedure. Selecting all of the rows and columns using the \* operator and no WHERE clause:

```
proc sql noprint ;  
  create table lds2 as  
  select *  
  from largeds  
  ;  
quit ;
```

generates the following in the LOG:

NOTE: Table WORK.LDS2 created, with 1024000 rows and 520 columns.

NOTE: PROCEDURE SQL used (Total process time):

real time	19.67 seconds
cpu time	10.46 seconds

Attempting to restrict columns in the output dataset using the SELECT clause allows only for the \* wildcard, or an explicit list of variables; it is not possible to use the range expression select a1-a20 as this is seen simply as a subtraction, and the colon modifier select a: is not supported on the SELECT clause.

It is however possible to generate a macro variable containing the desired variables in a comma separated list:

```
proc sql noprint ;  
  select name into :a_list separated by ', '  
  from dictionary.columns  
  where libname = 'WORK'  
    and memname = 'LARGEDS'  
    and name like 'A%'  
  ;  
quit ;
```

# DATA Step

and resolving this in the desired PROC SQL step:

```
proc sql noprint ;  
  create table lds3 as  
  select &a_list  
  from largeds  
  ;  
quit ;
```

NOTE: Table WORK.LDS3 created, with 1024000 rows and 20 columns.

NOTE: PROCEDURE SQL used (Total process time):

real time	14.91 seconds
cpu time	8.84 seconds

However this still only restricts the Output. To generate maximum efficiency the restriction should be applied to the Input. This can be achieved by using the KEEP= option on a dataset listed on the FROM clause.

```
proc sql noprint ;  
  create table lds4 as  
  select *  
  from largeds (keep = a:)  
  ;  
quit ;
```

NOTE: Table WORK.LDS4 created, with 1024000 rows and 20 columns.

NOTE: PROCEDURE SQL used (Total process time):

real time	2.87 seconds
cpu time	2.87 seconds

It will be noted that the a: syntax 'is' supported in this context, removing the need to generate the macro variable list of variable names.

NOTE: These run-times are based on a single instance - to get a true measure of the savings

# DATA Step

the exercise should be run multiple times and an average obtained.

Unique solution ID: #1021

Author: Alan D Rudland

Last update: 2017-04-06 12:50