

SAS® Efficiency Techniques

Steven First, Systems Seminar Consultants

Kim Kolbe, Systems Seminar Consultants

Introduction

SAS software certainly can consume large amounts of resources, however by understanding the unique nature of the system and by taking advantage of its characteristics, efficient and competitive programs can be written. This paper will present numerous topics and examples of how SAS users can produce such programs.

SAS Efficiencies

Efficiency can be measured in terms of:

- CPU time
- I/O time
- memory
- disk and tape space
- human resources.

Notes:

You may need to make trade-offs among the above.

Keys to Efficiency

Efficient programs:

- keep data sizes to a minimum
- use efficient data types
- read and write selectively
- make efficient use of SAS statements
- use the correct procedure for the task
- know defaults and override them when desirable
- avoid sorting
- take advantage of data characteristics
- comment and code clearly.

Measuring Efficiency

Invocation options can provide complete run time statistics.

STIMER compute CPU time consumed.
MEMRPT compute memory usage.
FULLSTIMER compute CPU, elapsed time, memory, I/O.
STATS reports all statistics to SAS log.

Sample invocations:

```
MVS // EXEC SAS,OPTIONS='STATS FULLSTIMER'
TSO SAS OPTIONS('STATS FULLSTIMER')
CMS SAS (STATS FULLSTIMER
```

Directory Based SAS/STATS/FULLSTIMER

Sample LOG:

```
397 PROC PRINT DATA=LOANAPPS;TITLE 'LOANAPPS';RUN;
NOTE:The PROCEDURE PRINT printed page 1.
NOTE:The PROCEDURE PRINT used the following resources:
CPU time - 00:00:00.17
Elapsed time - 00:00:00.35
EXCP count - 76
Task memory - 637K (25K data, 612K program)
Total memory - 2397K (1341K data, 1056K program)
```

Keeping Data Sizes to a Minimum

Make your datasets as small as possible as early as possible.

- output only the rows needed
- keep only the columns needed
- use shorter variables if possible
- use permanent SAS datasets when appropriate
- use summary files when appropriate
- use samples
- preprocess with a faster program.

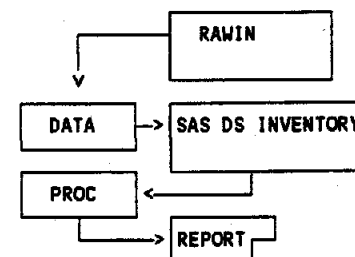
DATA Set Sizes

A standard DATA step is followed by various PROC steps.

RAW file RAWIN

101002	COMPRESSOR UNIT	350.69	10	66738
103020	TEMPERATURE REGULATOR	116.33	300	30291
106303	OUTER UNIT REPAIR KIT	33.23	500	10101
109283	THERMOSTAT	55.20	65	30309
110928	REPAIR NUTS & BOLTS	2.99	602	99383
111493	INSULATION KIT	4.34	228	83831
118457	COOLING COILS	37.83	33	81818

```
DATA INVENTORY;
  INFILE RAWIN;
  INPUT PARTNUM $1-6 DESCRIP $10-49
  COST 50-55 INVENTORY 60-62
  LOCATION $65-70 ; RUN;
PROC PRINT DATA=INVENTORY;
  TITLE 'INEFFICIENT INVENTORY';
  WHERE COST > 100;
  VAR PARTNUM COST; RUN;
Data Flow
```



Can we improve this program?

A More Efficient Program

Eliminate rows and columns as early as possible.

101002	COMPRESSOR UNIT	350.69	10	66738
103020	TEMPERATURE REGULATOR	116.33	300	30291
106303	OUTER UNIT REPAIR KIT	33.23	500	10101
109283	THERMOSTAT	55.20	65	30309
110928	REPAIR NUTS & BOLTS	2.99	602	99383
111493	INSULATION KIT	4.34	228	83831
118457	COOLING COILS	37.83	33	81818

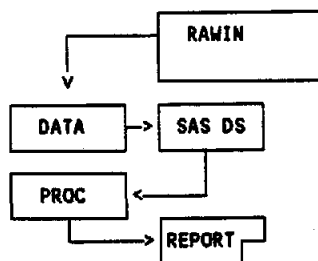
```
DATA INVENTORY;
  INFILE RAWIN;
  INPUTPARTNUM $1-6
  /* DESCRIP $10-49 */
  COST 50-55
  /* INVENTORY 60-62 */
  /* LOCATION $65-70 */
```

```

IF COST > 100; RUN;
PROC PRINT DATA=INVENTORY;
TITLE 'BETTER INVENTORY'; RUN;

```

Data Flow

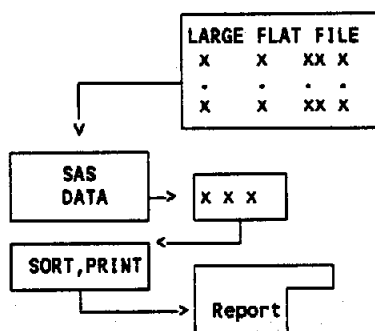


Notes:

A similar example reading permanent SAS datasets will be covered later.

A Preprocessing Example

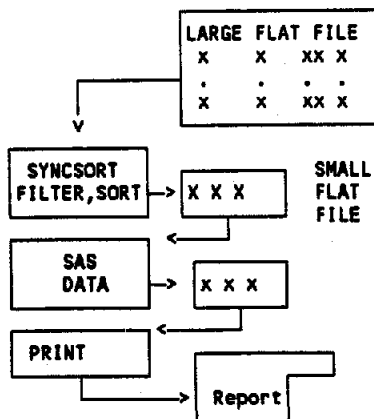
SAS reads a 6 million row dataset to select a few rows and columns. The job costs \$1200.



Can we improve?

Another Approach

Preprocess with a faster program like SYNCSORT® for \$200.



Advantages:

- SYNC SORT is much faster than SAS to SORT, FILTER raw data
- SYNC SORT can subset rows and columns, sort, even report in one pass
- works best with small subsets of very large raw files.

Disadvantages:

- much more complicated control codes

- two programs are involved.

Use Efficient Data Types

Data set structure can greatly affect performance.

Tips:

- avoid tape (disk is faster and more convenient)
- use SAS datasets if possible (avoids conversion)
- format numbers in floating point (minimizes conversion)
- minimize numerics
- watch default numeric length
- watch individual lengths
- use DATA _NULL_ if no SAS dataset is needed.

Minimize Numerics

The following program reads RAWIN, can we improve?

RAW file RAWIN

101002	COMPRESSOR UNIT	350.69	10	66738
103020	TEMPERATURE REGULATOR	116.33	300	30291
106303	OUTER UNIT REPAIR KIT	33.23	500	10101
109283	THERMOSTAT	55.20	65	30309
110928	REPAIR NUTS & BOLTS	2.99	602	99383
111493	INSULATION KIT	4.34	228	83831
118457	COOLING COILS	37.83	33	81818

```

DATA INVENTORY;
INFILE RAWIN;
INPUT PARTNUM DESCRIP & $40. COST
      INVENTORY LOCATION; RUN;
PROC CONTENTS DATA=INVENTORY;
TITLE 'INVENTORY'; RUN;

```

INVENTORY CONTENTS PROCEDURE					
Data Set Name:	WORK.INVENTORY	Type:			
Observations:	7	Record Len:	76		
Variables:	5				
Label:					
-----Alphabetic List of Variables and Attributes-----					
#	Variable	Type	Len	Pos	Label
3	COST	Num	8	52	
2	DESCRIP	Char	40	12	
4	INVENTORY	Num	8	60	
5	LOCATION	Num	8	68	
1	PARTNUM	Num	8	4	

A More Efficient Program

Use formatted input and carefully define variables.

```

DATA INVENTORY;
LENGTH INVENTORY 3 ;
INFILE RAWIN;
INPUT @1 PARTNUM $6. @10 DESCRIP $21.
      @50 COST 6.2
      @60 INVENTORY 3. @65 LOCATION $5. ; RUN;
PROC CONTENTS DATA=INVENTORY;
TITLE 'INVENTORY'; RUN;

```

Resulting Output:

INVENTORY CONTENTS PROCEDURE					
Data Set Name:	WORK.INVENTORY	Type:			
Observations:	7	Record Len:	47		
Variables:	5				
Label:					
-----Alphabetic List of Variables and Attributes-----					
#	Variable	Type	Len	Pos	Label
4	COST	Num	8	34	
3	DESCRIP	Char	21	13	
1	INVENTORY	Num	3	4	
5	LOCATION	Char	5	42	
2	PARTNUM	Char	6	7	

Notes:

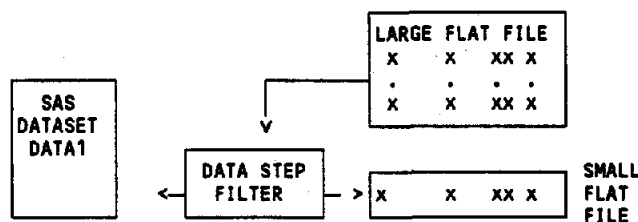
INPUT of numerics is one of the most expensive SAS operations.
Do not make numerics too short or shorten fractions.
SAS Dates can always fit in 4 bytes.

Use DATA _NULL_ When Possible

If you don't need a SAS dataset, DATA _NULL_ saves time and space.

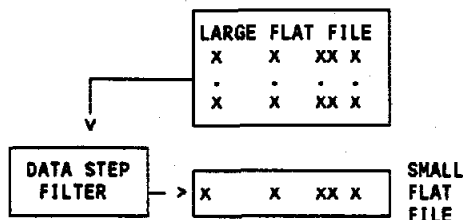
Example: Make a flat test file from a large one.

```
DATA;                                /* DON'T NEED DATASET */
  INFILE IN;                         /* RAW FILE IN          */
  FILE OUT;                          /* RAW FILE OUT         */
  INPUT @5 ID $CHAR1.;               /* INPUT FIELDS NEEDED*/
  IF ID='2';                         /* WANT THIS RECORD? */
  PUT _INFILE_;                     /* YES, WRITE IT OUT */
RUN;
```



A Better Program

```
DATA _NULL_;                        /* DON'T NEED DATASET */
  INFILE IN;                         /* RAW FILE IN          */
  FILE OUT;                          /* RAW FILE OUT         */
  INPUT @5 ID $CHAR1.;               /* INPUT FIELDS NEEDED*/
  IF ID='2';                         /* WANT THIS RECORD? */
  PUT _INFILE_;                     /* YES, WRITE IT OUT */
RUN;
```



Read and Write Data Selectively

Minimize reads, writes, and data movement.

Tips:

- ☐ input only fields needed (covered earlier)
- ☐ use formatted input instead of list input
- ☐ use absolute pointer (@) instead of relative (+)
- ☐ output only fields needed (covered earlier)

- ☐ use selection fields
- ☐ use WHERE instead of IF when possible
- ☐ use KEEP, DROP early
- ☐ build multiple datasets in a single pass.

Use Selection Fields

Read the Inventory data again.

RAW file RAWIN

101002	COMPRESSOR UNIT	350.69	10	66738
103020	TEMPERATURE REGULATOR	116.33	20	30291
106303	OUTER UNIT REPAIR KIT	33.23	500	10101
109285	THERMOSTAT	55.20	65	30309
110923	NUTS & BOLTS KIT	2.99	602	99383
111493	INSULATION KIT	4.34	228	83831
118457	COOLING COILS	37.83	33	81818

We are interested in only:

Those parts which cost more than \$50 and have more 100 in inventory.

One possible solution:

```
DATA INVENTORY;
  LENGTH INVENTORY 3 ;
  INFILE RAWIN;
  INPUT @1 PARTNUM $6. @10 DESCRIP $21. @50 COST 6.2
        @60 INVENTORY 3. @65 LOCATION $5. ;
  IF COST > 50 AND INVENTORY > 100; RUN;
PROC PRINT DATA=INVENTORY;
  VAR PARTNUM COST INVENTORY; RUN;
```

Can we improve on the above program?

Reading and Writing Selectively

Read only what you need conditionally.

```
DATA INVENTORY;
  LENGTH INVENTORY 3 ;
  INFILE RAWIN;
  INPUT @50 COST 6.2 @; /* INPUT ONLY COST */
  IF COST > 50;         /* OVER 50 PASS THIS */
  INPUT @60 INVENTORY 3. @; /* REREAD SAME REC */
  IF INVENTORY > 100;    /* OVER 100 PASS THIS */
  INPUT @1 PARTNUM $6.; /* FINALLY GET PARTNUM */
RUN;
PROC PRINT DATA=INVENTORY; RUN;
```

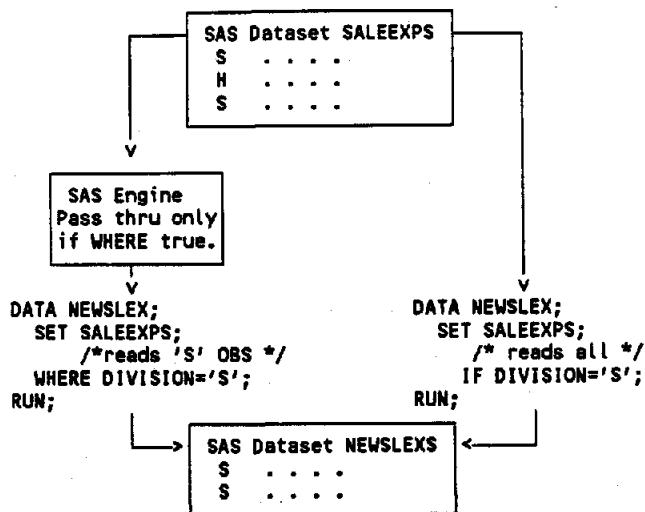
Notes:

Since the line is already in a buffer, the second and third INPUTS are very cheap (no I/O).

This minimizes a very expensive operation (INPUT).

Use WHERE if Possible.

If your input is a SAS dataset, WHERE is a usually faster than IF.



City	var1	var5	var7

SAS DATASET SMALL

var1	var5	var7
------	------	------

Build Multiple Datasets in a Single Pass

Example:

We need to pass through a file and produce three datasets:

INVENTORY contains all observations
PARTS contains observations with '3' in column 6.
KITS contains all observations that are not PARTS.

```

DATA KITS PARTS INVENTORY;
LENGTH INVENTORY 3 ;
INFILE RAWIN;
INPUT @1 PARTNUM $6. @10 DESCRIP $21. @50 COST
      6.2 @60 INVENTORY 3. @65 LOCATION $5. ;
OUTPUT INVENTORY;
IF SUBSTR(PARTNUM,6,1)='3' THEN
  OUTPUT KITS;
ELSE
  OUTPUT PARTS; RUN;

```

Notes:

The INVENTORY SAS data set will contain all observations.

Use SAS Statements Efficiently

Judicious use of statements can improve performance.

Tips:

- ☐ assign constants once
- ☐ create macro variables only when needed (Ex. CALL SYMPUT on last observation)
- ☐ use mutually exclusive IF or SELECT when possible
- ☐ use IN operator instead of OR
- ☐ order conditions in descending probability of occurring
- ☐ keep as much as possible outside of loops
- ☐ write loop with fewest iterations outermost.

A Mapping Program

Select midwestern states for a regional map and put total obs selected into a macro variable.

An Acceptable Program:

```

DATA MIDWEST;
SET MAPS.COUNTY;
DESC='MIDWEST';
IF STATE = 17
OR STATE = 18
OR STATE = 19
OR STATE = 55;
IF STATE=17 THEN STATENM='ILLINOIS';
IF STATE=18 THEN STATENM='INDIANA';
IF STATE=19 THEN STATENM='IOWA';
IF STATE=55 THEN STATENM='WISCONSIN';
KTR+1;
CALL SYMPUT('MKTR',PUT(KTR,5.)); RUN;

```

Use KEEP and DROP Early

A large SAS dataset used for extracts.

Example:

A state DOT keeps a SAS DATASET for statewide accidents. It contains 400 variables and hundreds of thousands of observations. It frequently subsets the file for various city inquiries.

PERM.ALLACCID

County	City	var1	var2	var3	var4	var400
ADAMS						
ADAMS						
ADAMS						
WASH						
WASH						

```

DATA SMALL;
SET PERM.ALLACCID;
IF CITY='MADISON';
KEEP VAR1 VAR5 VAR7; RUN;

```

County	City	var1	var2	var3	var4	var400

SAS DATASET SMALL

var1	var5	var7
------	------	------

Can we improve on the above?

Use KEEP and DROP Early

KEEP and DROP early minimizes movement and storage.

PERM.ALLACCID

County	City	var1	var2	var3	var4	var400
ADAMS						
ADAMS						
ADAMS						
WASH						
WASH						

```

DATA SMALL;
SET PERM.ALLACCID(KEEP=CITY VAR1 VAR5 VAR7);
WHERE CITY='MADISON';
KEEP VAR1 VAR5 VAR7; RUN;

```

A Faster Program (50% elapsed time of above)

```
DATA MIDWEST;
  SET MAPS.COUNTY END=EOF;
  WHERE STATE IN (17, 18, 19, 55);
  BY STATE;
  RETAIN DESC 'MIDWEST' STATENM ' ';
  IF FIRST.STATE THEN
    IF STATE=17 THEN STATENM='ILLINOIS';
    ELSE IF STATE=18 THEN STATENM='INDIANA';
    ELSE IF STATE=19 THEN STATENM='IOWA';
    ELSE IF STATE=55 THEN STATENM='WISCONSIN';
  KTR+1;
  IF EOF THEN
    CALL SYMPUT('MKTR',PUT(KTR,5.)); RUN;
```

Another Example

Order conditions in descending probability of occurring.

Problem: We have a census file with one obs for each city in the US (approximately 50000). The variables are STATE, COUNTY, CITY, and POPULATE. 2000 of the observations are in Wisconsin, 50 in DANE county, 1 for city of Madison.

We want to select the one observation for Madison in DANE county Wisconsin.

Raw Data
CENSUS

AZ MOHAVE	KINGMAN	132,303
IN DUBOIS	JASPER	38,303
WI DANE	MADISON	660,393
WY NATRONA	CASPER	73,202

```
DATA FINDCITY;
  INFILE CENSUS;
  INPUT STATE $ COUNTY $ CITY $ POPULATE;
  IF STATE='WI';          <==== Executed
    approx. 50,000
  IF COUNTY='DANE';       <==== Executed
    approx. 2,000
  IF CITY='MADISON';      <==== Executed
    approx. 50 RUN;
```

A more efficient program:

```
DATA FINDCITY;
  INFILE CENSUS;
  INPUT STATE $ COUNTY $ CITY $ POPULATE;
  IF CITY='MADISON';      <==== Executed
    < 50,000
  IF COUNTY='DANE';       <==== Executed
    approx. 19
  IF STATE='WI';          <==== Executed
    approx. 1
  OUTPUT;
  STOP; RUN;              <==== Stop when found
```

Use SAS Statements Efficiently

More Data step suggestions.

Tips:

- avoid using missing values in calculations
- avoid implicit conversions
- use functions instead of your own formulas
- use temporary arrays instead of creating and dropping variables
- use informats and formats for data transformation
- use compiled DATA steps.

Using Statements Efficiently

An example of CPU intensive code.

Raw Data
QTRSALES

30	43	10	9	1
			9	3
10	50	30	16	2
30	20	39	2	8

Example: Define an array of PROD1 PROD5; do various calculations.

```
DATA SALES;
  INFILE QTRSALES;
  INPUT JANSALES FEBSALES MARSALES PRODCODE $ COST;
  ARRAY VALID {5} $ PROD1-PROD5
    ('5', '8', '9', '12', '16');
  TOTSALES=SUM(OF JANSALES, FEBSALES, MARSALES);
  DO I=1 TO 5;
    IF PRODCODE=VALID{I} THEN
      DO;
        PROFIT=TOTSALES-COST;
        TAX=TOTSALES * .06;
        AVGSALE=TOTSALES/3;
      END;
    END;
  END;
  DROP I; RUN;
  PROC PRINT DATA=SALES;
  TITLE 'SALES DATA'; RUN;
```

Partial Log:

NOTE: Missing values were generated as a result of performing an operation on missing values.
Each place is given by: (Number of times) at (Line):(Column).

1 at 28:39
1 at 32:17
1 at 33:14
1 at 34:18

NOTE: The data set WORK.SALES has 4 observations and 14 variables.

NOTE: The DATA statement used 2.00 seconds.

SALES DATA													
J	F	M	P							T			A
A	E	A	R							O			V
N	B	R	O							T	P		
S	S	S	D		P	P	P	P	P	S	R		G
A	A	A	C	C	R	R	R	R	R	A	O		S
O	L	L	L	O	O	O	O	O	O	L	F	T	A
B	E	E	E	D	S	D	D	D	D	E	I	A	L
S	S	S	S	E	T	1	2	3	4	5	S	T	X
1	30	43	10	8	1	5	8	9	12	16	83	82	4.98 27.6667
2	.	.	.	9	3	5	8	9	12	16	.	.	.
3	10	50	30	16	2	5	8	9	12	16	90	88	5.40 30.0000
4	30	20	39	2	8	5	8	9	12	16	89	.	.

Notes:

PROD1-PROD5 are unneeded.

A More Efficient Solution

Avoiding calculation on missing values, using functions and _TEMPORARY_ arrays saves CPU resources.

```
DATA SALES;
  INFILE QTRSALES;
  INPUT JANSALES FEBSALES MARSALES PRODCODE $ COST;
  ARRAY VALID {5} $ _TEMPORARY_ ('5', '8', '9', '12', '16');
  TOTSALES=SUM(OF JANSALES, FEBSALES, MARSALES);
  DO I=1 TO 5;
    IF PRODCODE=VALID{I} AND TOTSALES NE . THEN
      DO;
        PROFIT=TOTSALES-COST;
        TAX=TOTSALES * .06;
        AVGSALE=MEAN(JANSALES, FEBSALES, MARSALES);
      END;
  END;
```

```
END;
DROP I; RUN;
PROC PRINT DATA=SALES; TITLE 'SALES DATA'; RUN;
```

SALES DATA										
OBS	JSALEF	SALEMS	SALEPROD	CODECOST	TOTSALE	PROFIT	TAXAVG	SALE		
1	30	43	10	8	1	83	82	4.98	27.6667	
2	.	.	.	9	3	.	.	.	.	
3	10	50	30	16	2	90	88	5.40	30.0000	
4	30	20	39	2	8	89	.	.	.	

Use Formats to Translate Data

Formats and PUT function perform a binary search.

RAW file RAWIN

101002	COMPRESSOR UNIT	350.69	10	66738
103020	TEMPERATURE REGULATOR	116.33	20	30291
106303	OUTER UNIT REPAIR KIT	33.23	500	10101
109285	THERMOSTAT	55.20	65	30309
110923	NUTS & BOLTS KIT	2.99	602	99383
111493	INSULATION KIT	4.34	228	83831
118457	COOLING COILS	37.83	33	81818

```
DATA BUILDING;
  INFILE RAWIN;
  INPUT LOCATION;
  IF LOCATION=109285 OR LOCATION=110923; RUN;
```

Another program

```
PROC FORMAT;
  VALUE BLDGFMT
    101002, 103020, 106303 = 'BLDG 1'
    109285, 110923      = 'BLDG 2'
    111493, 118457      = 'BLDG 3'
; RUN;
DATA BLDG2_REP;
  INFILE RAWIN;
  INPUT LOCATION;
  BLDG_LOC=PUT(LOCATION,BLDGFMT.);
  IF BLDG_LOC='BLDG 2'; RUN;
```

Notes:

FORMATS used for table lookups are especially useful for large tables.

Compiled DATA step code

6.06 can store and execute compiled DATA step source code.

Features:

- global statements such as LIBNAME, FILENAME, TITLE, FOOTNOTE, OPTIONS and MACRO facilities are NOT available
- SAS does NOT save the source code
- You CANNOT restore source code from the compiled version
- You CANNOT run a compiled DATA step under a different release of the SAS system or under a different operating system.

Storing a Compiled Data Step

The following steps are required:

1. Write, Test and Debug Program
2. Reference external files with a fileref rather than the actual file name
3. Submit program using PGM= option in the RUN, CARDS or CARDS4 statement.

```
LIBNAME IN 'OLDSAS.DATA';
```

```
LIBNAME OUT 'MYSAS.DATA';
LIBNAME STORED 'MYSAS.PGMS';
DATA OUT.TESTDATA;
  SET IN.SALEEXPS;
  NET=SALES-EXPENSE;
  RUN PGM=STORED.SALEEXPS;
```

Executing a Compiled Data Step

The following steps are required:

1. write an abbreviated SAS data step using PGM=
2. use REDIRECT statement to use different input or output SAS data sets if necessary
3. use FILENAME to assign filerefs to raw data files.

```
LIBNAME STORED 'MYSAS.PGMS';
LIBNAME TEST1 'MYSAS.DATA';
LIBNAME TEST2 'OLDSAS.DATA';
DATA PGM=STORED.SALEEXPS;
  REDIRECT INPUT IN.SALEEXPS=TEST1.SALEEXPS;
  REDIRECT OUTPUT OUT.TESTDATA=TEST2.TESTDATA; RUN;
```

Cautions of Using Compiled Data Steps

Be aware of the following when using compiled data steps.

- attributes of the variables used in the REDIRECT statement must match attributes of the variables used in the MERGE, SET or UPDATE statements of the source code
- the length of the source code variable determines the length of the variable in the redirected data set if the variable length differs between the two
- extra variables in the redirected data sets do not cause the stored program to stop processing, but may produce undesirable results.

Using the Right Procedure For the Right Task

SAS PROCs can save considerable resources.

Examples:

- PROC FREQ to count and categorize
- PROC FREQ for statistics and tables
- PROC SUMMARY, MEANS for simple statistics
- PROC RANK for rankings
- Use PROC COPY rather than SETing a dataset with indexes.
- Use procedures rather than code you write yourself:
 - Use PROC TRANSPOSE rather than trying to transpose the data yourself.
 - Use PROC PRINT whenever possible instead of using the DATA step report writing.
- use procedures to perform as many different tasks as possible once you have invoked it
- use the output of the procedure for other tasks
- know the full capacity of each procedure
- use CLASS instead of BY if PROC supports it (CLASS takes more memory, but avoids sorting)

- use RUN group processing (avoids reloading PROC).

Using SQL To Simplify Code

A very powerful new procedure.

Features:

- especially useful when merging more than two data sets
- can match on variables that are not exactly the same
- can calculate intermediate results, as in percentage distributions
- can operate on specific variables without having to specify DROP and KEEP statements and prevents the entire read of an observation into the program data vector
- match observation by the values of any variable without pre-sorting any of the SAS data sets
- WHERE clauses limit the number of observations you process.

Understand SAS Defaults

Know and override SAS defaults when desirable.

- familiarize and set numeric length defaults
- use NOMACRO system option at SAS invocation or in a configuration file if macros are not needed:
(Reduces compile time of DATA and PROC steps because the SAS system does not keep the macro processor ready to check for special macro characters)
- compress large datasets (saves space, more CPU)

Compression Of Observations

Observations can be uncompressed or compressed.

Uncompressed (default) observations

- all fields are fixed length
- numerics are floating point
- characters are fixed and padded with blanks.

Obs 1	Field 1	Field 2	Field 3
Obs 2	Field 1	Field 2	Field 3
Obs 3	Field 1	Field 2	Field 3

Compressed observations:

- fields are differing lengths
- consecutive repeating characters are collapsed
- characters are expanded when read
- reductions of 50% or more in disk space are common.

Obs 1	Field 1	Field 2	Field 3
Obs 2	Field 1	Field 2	Field 3
Obs 3	Field 1	Field 2	Field 3

The SAS compression algorithm:

- 3 - 129 blanks compresses into 2 bytes
- 3 - 66 real binary zeros compresses into 2 bytes
- 3 - 63 occurrences of any other character compresses into 3 bytes.

Compressed Data Sets

Compression works best if:

- data sets are short and wide
- contiguous variables have the same value, such as a series of 0s, blanks, or missing values
- variables are grouped that are likely to have the same value together to optimize compression

Notes: Compression cannot be used on tape datasets.

Most forms of direct access cannot be used on a compressed data set, such as the POINT=, FIRSTOBS= and OBS= options.

How Do You Compress SAS Datasets?

Compression can be accomplished two ways:

- COMPRESS= data set option can be used to selectively compress a data file
- COMPRESS system option causes all subsequent data files created in the session to be compressed.

Notes:

Compression can be verified with PROC CONTENTS.

Compression Tradeoffs

Compression can reduce storage requirements.

Compressed Files

More CPU intensive in preparing observation for I/O

Utilizes less disk space

Deleted observation space is reused

Updated observation may not fit in its original location

Observations can not be addressed by number

Uncompressed Files

Less CPU intensive in preparing observation for I/O

Utilizes more disk space

Deleted observation space is not reused

Updated observation will fit in its original location

Observation can be addressed by number

Notes:

Use the REUSE= YES | NO system option to control if deleted observation space can be reused.

Avoid Sorting

SORTING is very common and resource consuming.

Tips:

- Plan your job. If several procedures require data sorted in the same order, group those PROC steps together rather than re-sorting the data each time.
- sort data only when necessary
- use CLASS instead

- remove unnecessary variables and observations
- specify NOEQUALS options if it is not necessary for observations within groups to have the same relative order they had in the original SAS data set
- use indexing when appropriate.

A Sorting Example

Do all procedures that need a sort sequence at the same time.

Adequate

```
PROC SORT DATA=MIDWEST;
  BY STATE;
RUN;
PROC PRINT DATA=MIDWEST;
  BY STATE;
RUN;
PROC SORT DATA=MIDWEST;
  BY COUNTY;
RUN;
PROC PRINT DATA=MIDWEST;
  BY COUNTY;
RUN;
PROC SORT DATA=MIDWEST;
  BY STATE;
RUN;
PROC CHART DATA=MIDWEST;
  BY STATE;
  VBAR X Y;
RUN;
PROC SORT DATA=MIDWEST;
  BY COUNTY;
RUN;
PROC CHART DATA=MIDWEST;
  BY COUNTY;
  VBAR X Y;
RUN;
```

Better

```
PROC SORT DATA=MIDWEST;
  BY STATE;
RUN;
PROC PRINT DATA=MIDWEST;
  BY STATE;
RUN;
PROC CHART DATA=MIDWEST;
  BY STATE;
  VBAR X Y;
RUN;
PROC SORT DATA=MIDWEST;
  BY COUNTY;
RUN;
PROC PRINT DATA=MIDWEST;
  BY COUNTY;
RUN;
PROC CHART DATA=MIDWEST;
  BY COUNTY;
  VBAR X Y;
RUN;
```

Notes: Sorting always requires at least three data passes.

Sorting Datasets

Many procedures support CLASS in place of sorting.

Adequate:

```
PROC SORT DATA=MIDWEST;
  BY STATE;
RUN;
PROC MEANS DATA=MIDWEST;
  BY STATE;
RUN;
```

Better:

```
PROC MEANS DATA=MIDWEST;
  CLASS STATE;
RUN;
```

Indexing of Datasets

SAS datasets can be indexed by one or more variables

What is an index?

An auxiliary data structure that assists in the location (selection) of observations specified by the value of one or more key variables.

EMP SAS Data Set				Index File	
Obs	EMPNUM	DEPT	SALES	Key Value	Location Page(obs,obs...)
1	1294	SOFT	55.99	SOFT	1(1,...) 2(...)
2	1011	HARD	22.34	HARD	1(2,4,...) 2(...)
3	3333	DOCU	12.22	DOCU	1(3,...) 2(...)
4	0938	HARD	88.01		

```
PROC PRINT DATA=EMP;
  WHERE DEPT='HARD'; RUN;
```

Obs	EMPNUM	DEPT	SALES	
1	1011	HARD	22.34	<----From OBS 2
2	0938	HARD	88.01	<----From OBS 4

Notes:

SAS Data Views cannot be indexed.

Indexing Tradeoffs

Indexing stores pointers that can locate observations more quickly and efficiently.

Advantages of Indexing:

- quick access to a small subset of observations in the index
- values returned from the index are returned in sorted order
- SAS automatically decides whether or not to use and index.

Disadvantages of Indexing:

- CPU and I/O intensive in order to create and maintain the index
- extra memory is required to load the index table
- extra disk space on a separate file is required to store the index's data structure
- extra resources are required to rebuild indexes for a new version of a data file with indexes
- any manipulation of files without the use of the SAS System must consider the auxiliary index file
- the DATA step SET statement cannot access keyed observations until Release 6.08.

How Do You Create an Index?

There are at least four ways to index SAS datasets.

- PROC DATASETS
- ACCESS window
- PROC SQL
- PROC IML.

An Indexing Example

```
PROC DATASETS LIBRARY=SASDATA;
  MODIFY EMPDATA;
  INDEX CREATE EMPNUM / UNIQUE NOMISS;
  INDEX CREATE NAME=(FIRSTNAM LASTNAM);
  INDEX DELETE HIREDAT; RUN;
```

Indexing Guidelines

Miscellaneous rules of thumb

- minimize the number of indexes to reduce disk storage and update costs
- create indexes only if the data file physical size is large
- when using WHERE processing to select missing values do not use the NOMISS attribute
- indexing works best when retrieving a small number of observations (<50%).
- works best on values not frequently updated and that are uniformly distributed
- works well when data set is subsetted by values of index variable.
- SAS System uses indexes when performing BY processing (except when the NOMISS attribute is specified and there are missing values for the index variable in the data file)
- in WHERE processing, SAS determines which is more efficient: the use of the index or a sequential pass of the entire data set
- should the indexed file be accidentally deleted or damaged, the data file can usually be repaired by using PROC DATASETS.

Know Your Data

Being familiar with your raw and final data can minimize resource consumption.

Tips:

- use your SAS log
- use test files and samples
- browse or list raw input before INPUT
- use partial prints
- use PROC Contents
- test in pieces
- stop when finished or errors are encountered.

A Testing Method

Testing incrementally breaks the problem into smaller pieces.

Problem: Need to know the phone charges for Cowley Hall.

Example:

PHONELOC	BLDGLOC	PHONECST
100 MAIN HALL	46519 102	26241 33.98
102 COWLEY HALL	56412 103	31454 45.21
103 WING LIBRARY	26241 102	56412 76.12
	78398 102	

```

PROC SORT DATA=PHONELOC;
  BY BLDGNO; RUN;
DATA BLDGLOC;
  MERGE BLDGCODE PHONELOC;
  BY BLDGNO;
  IF BLDGNO='102'; RUN;
PROC SORT DATA=BLDGLOC;
  BY PHONE; RUN;
PROC PRINT DATA=BLDGLOC;
RUN; /* =====Stop program and test up to here
PROC SORT DATA=PHONECST; if results look ok, remove
  BY PHONE;              comment and test next portion
RUN;                      until program is debugged.
PROC PRINT DATA=PHONECST;
RUN;
```

```

DATA FINAL;
  MERGE BLDGLOC(IN=ONBLDLC)
        PHONECST(IN=ONCOST);
  BY PHONE;
  IF ONBLDLC;
  IF ONCOST; RUN;
PROC PRINT DATA=FINAL;
  TITLE 'FINAL DATASET'; RUN;
*/
```

Stop When Appropriate

If you get what you are searching for, or if severe errors occur, give up.

Example:

A master file is sorted in ACCOUNT order. We are searching for account number '45121'.

Raw Data	00001	12	49
ACCTHIST	45121	0	65
	55155	0	0
	99388	100	500

```

DATA ACCOUNT;
  INFILE ACCTHIST;
  INPUT ACCTNUM $ PASTDUE BALANCE;
  IF ACCOUNT='45121'; RUN;
```

A better program: Since there is a sort sequence, quit when past the desired account.

```

DATA ACCOUNT;
  INFILE ACCTHIST;
  INPUT ACCTNUM $ PASTDUE BALANCE;
  IF ACCOUNT >= '45121' THEN STOP; /* PAST REC, STOP */
  IF ACCOUNT='45121'; RUN;
```

Comment and Code Clearly

Comments and clear code are welcome in any language.

Commenting tips:

- use comments
- write a header box and change log
- code flower boxes before separate routines
- comment as you develop code, not after
- add comments to each line if necessary
- comment difficult code
- code blank lines between steps.

A Commenting Example

An Adequate Program

```

DATA SJFSAS.PARTDESC;
  INFILE TAPEIN;
  INPUT PARTNO $1-9
        DESC $38-67 ;
PROC SORT DATA=SJFSAS.PARTDESC; BY PARTNO;
PROC CONTENTS DATA=SJFSAS.PARTDESC;
  TITLE 'SJFSAS.PARTDESC';
```

A Better Program

```

/*****
/* SAS WARRANTY SYSTEM    PROGRAM    WTPGM10    */
/* DESCRIPTION:            */
/* THIS CREATES A SAS DATASET SJFSAS.PARTDESC FROM AN */
/* INPUT TAPE.            */
/* THIS DATASET WILL BE MERGED WITH OTHER WARRANTY */
/* DATA LATER.          */
/* INPUT:    TAPE FROM SJF SYSTEMS.    */
/* OUTPUT:    SAS DATASET SJFSAS.PARTDESC    */
/* SCHEDULE:    ANNUALLY    */
/* DATE WRITTEN: 27 JANUARY 1988    */
/* AUTHOR:    STEVEN FIRST, MADISON WI, 608-222-7081 */
/* REVISIONS:    */
*****/

DATA SJFSAS.PARTDESC;          /* BUILD SAS DS    */
  INFILE TAPEIN;              /* RAW INPUT FILEDEF */
  INPUT PARTNO $1-9          /* PART NUMBER    */
        DESC $38-67         /* DESCRIPTION    */
        ;                  /* END OF INPUT    */
RUN;                          /* END OF STEP    */
PROC SORT DATA=SJFSAS.PARTDESC; /* RUN PROC CONTENTS */
  BY PARTNO;                /* PARTNUMBER ORDER */
RUN;                        /* END OF STEP    */
PROC CONTENTS DATA=
  SJFSAS.PARTDESC;          /* RUN PROC CONTENTS */
  TITLE 'SJFSAS.PARTDESC'; /* TITLE    */
RUN;                        /* END OF STEP    */
/*****END OF PROGRAM WTPGM10*****/

```

Choose Meaningful Names

Good names are more self documenting.

Tips:

- choose meaningful names for datasets and variables
(Ex. HOURRATE instead of X23)
- assign variable labels
(Ex. LABEL X23='Hourly Payroll Rate');
- assign good default formats for numeric fields
- use PROC Contents to display attributes.

```

DATA X;
  INFILE DD1;
  INPUT X $ Y Z ; RUN;
DATA PAYROLL;
  INFILE DISKIN;
  INPUT NAME $ HOURS HOURRATE;
  LABEL NAME='EMPLOYEE NAME';
  LABEL HOURS='HOURS WORKED';
  LABEL HOURS='HOURLY PAYROLL RATE'; RUN;

```

Program Clearly

Simple Clear Programs are the best.

- use parentheses to group difficult formulas
- break complex formulas into two statements
- group declaratives together
- avoid NOT logic
- explicitly name datasets
- code RUN
- expect and code for unknowns
- heed and eliminate SAS warnings.

A Clarity Example

Another fairly complex DATA step.

An Adequate Program

```

DATA FORMULA;
  SET PERM.DATIN;
  A=LOG(Y)**2*113/Y+Z;
  DROP Y;
  RETAIN A; LENGTH A 4;

```

```

IF A NOT > 10 THEN DELETE;
B=LOG(X)**2*113/X+Z;
DROP X;
RETAIN B; LENGTH B 4;
IF B NOT > 10 THEN DELETE; RUN;

```

A better Program

```

DATA FORMULA;
  SET PERM.DATIN;
  A=(LOG(Y)**2)*((13/(Y+Z)));
  B=(LOG(X)**2)*((13/(X+Z)));
  IF A > 10;
  IF B > 10;
  RETAIN A B; LENGTH A B 4;
  DROP X Y;
RUN;

```

Another Clarity Example

A program to subset a map.

An adequate program

```

DATA WISCON;
  SET MAPS.USCOUNTY;
  IF STATE=55;
  NAME='STATE '||55;
PROC PRINT;

```

Partial SAS Log:

Numeric values were converted to numeric 3245 times at 55:22

A better program

```

DATA WISCON;
  SET MAPS.USCOUNTY;
  IF STATE=55;
  NAME='STATE '||'55'; RUN;
PROC PRINT DATA=WISCON;
  TITLE 'WISCON'; RUN;

```

Notes:

You may get by with warnings, but they often cause real problems.

Know When NOT to Use SAS

Other software may be more appropriate for some applications.

- one page budget, use a spreadsheet
- need 1 quick graph, use a PC Graphics program
- extremely fast online access, use another language
- Database applications may require another system.

The author will be glad to answer questions and accept suggestions at the following address:

Steven First
Systems Seminar Consultants
6014 Gateway Green
Madison, WI 53716
(608) 222-7081

SAS is a registered trademark of SAS Institute Inc.