



Quanticate

CONFIDENCE COMES FROM QUALITY

Effecting Efficiency Effortlessly

Daniel Carden, Quanticate



Quanticate

CONFIDENCE COMES FROM QUALITY

CONTENTS:

- **SAS VIEWS**
- **WHERE STATEMENTS**
- **EFFICIENT CODE STRUCTURING**
- **SKIP MACRO**
- **FORMAT LIBRARIES**

Efficiency Metrics

- *CPU time* = the time the Central Processing Unit spends performing the operations you assign.
- *I/O time* = the time the computer spends on two tasks, input and output. Input refers to moving the data from storage areas such as disks or tapes into memory. Output refers to moving the results out of memory to storage or to a display device.
- *Real time* = clock time.
- *Memory* = the size of the work area that the CPU must devote to the operations in the program.
- Another important resource is data storage - how much space on disk/tape.

A gain in efficiency is not usually absolute. A few programming techniques do improve performance in all areas.

SAS VIEWS

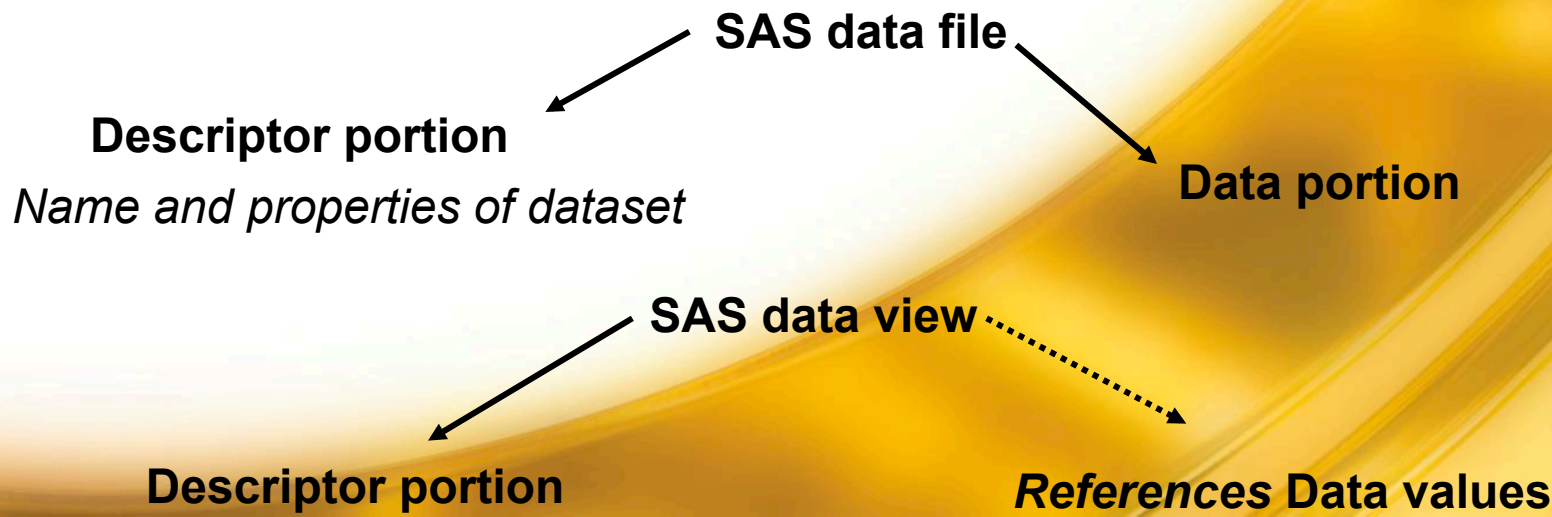
Three types of SAS data view:

- **DATA step views** are a type of data step program.
- **PROC SQL views** are stored query expressions that read data values from their underlying files, which can include SAS data files, SAS/ACCESS views, DATA step views, other PROC SQL views, or relational database data.
- **SAS/ACCESS views** (also called view descriptors) describe data that is stored in DBMS (Database Management System) tables.

SAS datasets: SAS views vs. SAS data files



- Descriptor portion: name and properties of the data set : e.g. when it was created, number of observations and variables.
- Data portion contains the data values.
- SAS data **file** stores descriptor information and data values together.
- A SAS data **view** defines a virtual data set. It has the information required to access data values and is stored separately from the data values.



SAS data views syntax:

- **data** labs / view = labs;
- set labsdata;
- gender = sex;
- label gender = 'Gender Type';
- mid = (lowrang + hirang)/2;
- **run;**

- **data** labs2;
- set labs;
- **run;**

SAS views and resources



- SAS views cut I/O time and hence real time.
- Negligible effect on CPU time or increase it slightly.
- Best used when real execution times greatly exceed CPU times.
- If a large dataset is used as an intermediate dataset more than once then use a SAS view in the code.

**Drawbacks of SAS views: fewer errors in log and cannot overwrite*

Method 1:

```
data labs;  
  set labsdata;  
  gender = sex;  
  label gender = 'Gender Type';  
  mid = (lowrang + hirang)/2;  
run;
```

NOTE: DATA statement used:

real time	17.39 seconds
cpu time	0.76 seconds

```
data labs2;  
  set labs;  
run;
```

NOTE: DATA statement used:

real time	28.75 seconds
cpu time	0.93 seconds

Total = 17.39s + 28.75s = 46.14s

Method 2:



```
data labs / view = labs;  
  set labsdata;  
  gender = sex;  
  label gender = 'Gender Type';  
  mid = (lowrang + hirang)/2;  
run;
```

NOTE: DATA STEP view saved on file WORK.LABS.

NOTE: A stored DATA STEP view cannot run under a different operating system.

NOTE: DATA statement used:

real time	0.01 seconds
cpu time	0.01 seconds

```
data labs2;  
  set labs;  
run;
```

NOTE: View WORK.LABS.VIEW used:

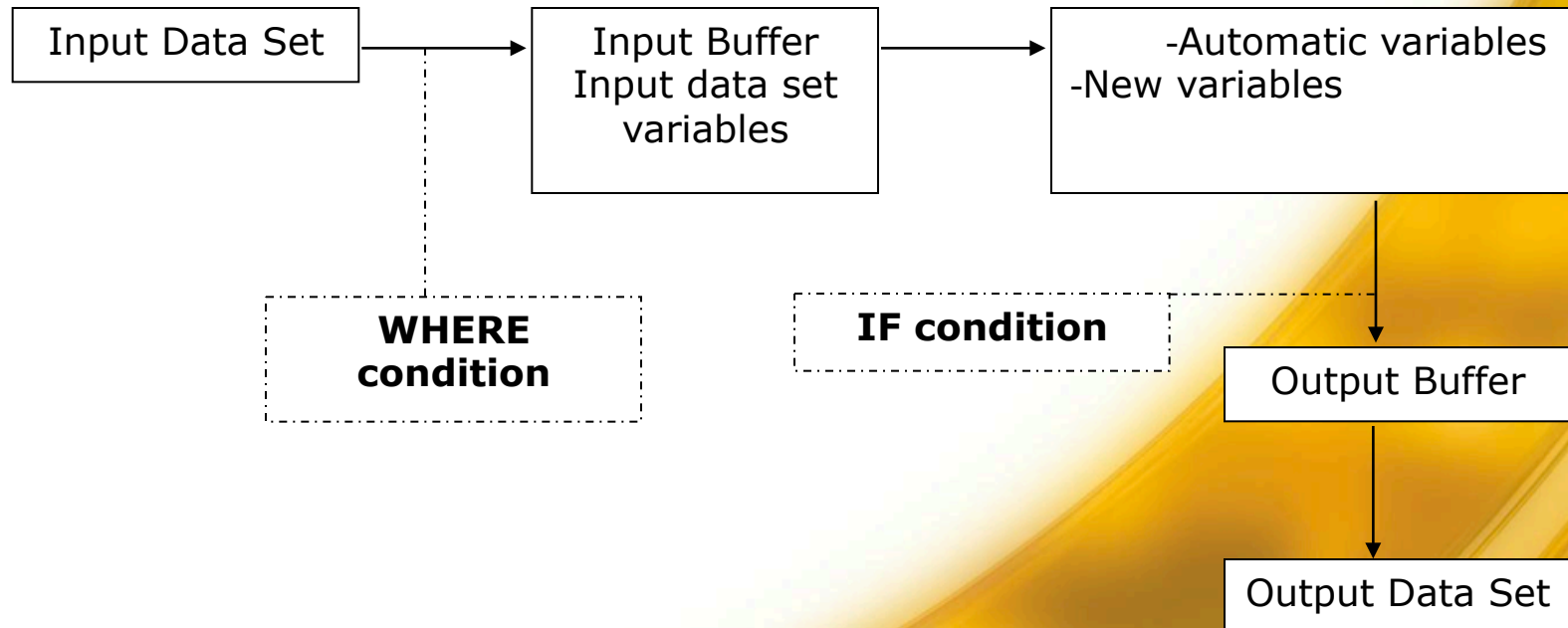
real time	19.32 seconds
cpu time	0.59 seconds

NOTE: DATA statement used:

real time	21.65 seconds
cpu time	1.10 seconds

Total = 0.01s + 21.65s = 21.66s

WHERE STATEMENTS



EFFICIENT CODE STRUCTURING



Two data step method	One data step method
<pre> 19 data labs; 20 set labsdata; 21 where obssd ^= 0; 22 run; </pre> NOTE: There were 319452 observations read from the data set WORK.LABSDATA. WHERE obssd not = 0; NOTE: The data set WORK.LABS has 319452 observations and 39 variables. NOTE: DATA statement used: <pre> real time 22.91 seconds cpu time 0.98 seconds 23 24 25 proc sort data = labs out = labs2; 26 by pt invsite; 27 run; </pre> NOTE: There were 319452 observations read from the data set WORK.LABS. NOTE: The data set WORK.LABS2 has 319452 observations and 39 variables. NOTE: PROCEDURE SORT used: <pre> real time 1:00.63 cpu time 2.78 seconds </pre>	<pre> 30 proc sort data = labsdata (where = (obssd ^= 0)) out = labs; 31 by pt invsite; 32 run; </pre> NOTE: There were 319452 observations read from the data set WORK.LABSDATA. WHERE obssd not = 0; NOTE: The data set WORK.LABS has 319452 observations and 39 variables. NOTE: PROCEDURE SORT used: <pre> real time 57.39 seconds cpu time 1.73 seconds </pre>
Total CPU run time = 0.98s + 2.78s = 3.76 s	Total CPU run time = 1.73 s
Total real run time = 1m0.6s + 22.9s = 1m23.5s	Total real run time = 57.4s

Invoke macros only when needed:



Method 1	Method 2
<pre>%macro labvital (n=, where=); proc sort data = rawdata.vitals out = vitals nodupkey; by pt; run; data labs&n; set labsdata; where &where; mid = (lowrang + hirang)/2; if hirang > 0 then percent = (lowrang /hirang) * 100; run; data vitlab&n; merge labs&n vitals; by pt; run; %mend; %labvital (n= 1, where= lvaluen>0.5); %labvital (n= 2, where= lvaluen>1); %labvital (n= 3, where= lvaluen>1.5); %labvital (n= 4, where= lvaluen>2); %labvital (n= 5, where= lvaluen>2.5); %labvital (n= 6, where= lvaluen>3);</pre>	<pre>proc sort data = rawdata.vitals out = vitals nodupkey; by pt; run; %macro labvital (n=, where=); data labs&n; set labsdata; where &where; mid = (lowrang + hirang)/2; if hirang > 0 then percent = (lowrang /hirang) * 100; run; data vitlab&n; merge labs&n vitals; by pt; run; %mend; %labvital (n= 1, where= lvaluen>0.5); %labvital (n= 2, where= lvaluen>1); %labvital (n= 3, where= lvaluen>1.5); %labvital (n= 4, where= lvaluen>2); %labvital (n= 5, where= lvaluen>2.5); %labvital (n= 6, where= lvaluen>3);</pre>
CPU run time = 2.31s	CPU run time = 1.46s
Total real run time = 72.51s	Total real run time = 68.41s



Sort first, then invoke macro!!

SKIP MACRO

Commenting out code by `/* */`:

Advantages = Quick & ideal for making small comments

Disadvantages =

Can cause errors if left accidentally in code

Can unintentionally comment out items if not closed

Will still show commented-out code in the log

Needs to be repeated if the code is already commented...

Skipping code with SKIP MACRO:

```

run;

/*Number of subjects calculated using proc freq*/

data number2;
    length col1 $54.;
    set number;
    order1 = 2;
    order2 = 0;
    col1 = 'Number of Subjects';
    col2 = trim(left(put(count,12.)));
    col3 = 0;

run;

/*Order variables as per SAP*/

data number3 (drop = count);
    retain order1 order2 trt col1 col2 col3;
    set number2;

run;

/*Data set for number of subjects created*/

data cyc (keep = usubjid trt cycle startcyc);
    set derived.endbycyc;
    if startcyc = '' then delete;

run;

/*Remove duplicate records*/

proc sort data = cyc out = cyc2 nodupkey;
    by usubjid trt cycle startcyc;

run;

/*Remove records which aren't at the start of the cycle*/

data cyc2b;
    set cyc2;
    if startcyc = 'N' then delete;

run;
    
```

EXAMPLE:
5 /* */ required

The more
comments, the
more /* */s!!

1

```

%macro skip;

/*Number of subjects calculated using proc freq*/

data number2;
    length col1 $54.;
    set number;
    order1 = 2;
    order2 = 0;
    col1 = 'Number of Subjects';
    col2 = trim(left(put(count,12.)));
    col3 = 0;

run;

/*Order variables as per SAP*/

data number3 (drop = count);
    retain order1 order2 trt col1 col2 col3;
    set number2;

run;

/*Data set for number of subjects created*/

data cyc (keep = usubjid trt cycle startcyc);
    set derived.endbycyc;
    if startcyc = '' then delete;

run;

/*Remove duplicate records*/

proc sort data = cyc out = cyc2 nodupkey;
    by usubjid trt cycle startcyc;

run;

/*Remove records which aren't at the start of the cycle*/

data cyc2b;
    set cyc2;
    if startcyc = 'N' then delete;

run;

%mend skip;
    
```

2

EASY!

SKIP MACRO Syntax

`%macro skip;`

`<CODE, which can include comments>`

`%mend skip;`

*NB: Don't leave an unclosed %macro, will treat all submitted as macro code.
Always close with %mend.*

FORMAT LIBRARIES

Efficient to restrict amount of data being read in by SAS.

- A SAS Index is similar to a search function, allowing access to a subset of records from a large data set
- Format libraries offer another way to subset the data

Scenario:

Situation:

D2
Lab test **#1** results for
Patient **1**,
Patient **2**,
Patient **3**,
Patient **4**.

Objective:

D1
Height, weight, ethnicity for
Patient **1** and Patient **2**.

D3
Lab test **#2** results for
Patient **1**,
Patient **2**,
Patient **3**,
Patient **4**.

Height, weight, ethnicity for Patient **1**
and Patient **2**.
Lab test **#1, #2, #3**
results for Patient **1**
and Patient **2**.

D4
Lab test **#3** results for
Patient **1**,
Patient **2**,
Patient **3**,
Patient **4**.

Create a Format Library:

- data D1;
 - set rawdata.D1;
 - start = subjid;
 - fmtname = '\$Fsubj';
 - label = 'Y';
 - type = 'C';
- run;
- proc format cntlin = D1;

PROC format is used with the CNTLIN option to create the dataset into a Format Library. Need the following variables to do this:

- *START: The value to format into a label (the KEY).
- FMTNAME: The name of the format being created, which can be anything except the name of a format which is already defined. When the KEY is character, FMTNAME must start with a \$ just like any PROC FORMAT value.
- TYPE: Either character ('C') or numeric ('N') format.
- LABEL: The label given to the KEY variable. This can be anything, but must not be the first byte in the KEY.
- *NB: There must not be any duplicates of the variable used as the KEY variable.

```
data D1;  
  set rawdata.D1;  
  start = subjid;  
  fmtname = '$Fsubj';  
  label = 'Y';  
  type = 'C';  
run;
```

```
proc format cntlin = D1;
```

```
data D234;  
  set D2 D3 D4;  
  by subjid;  
  if put (subjid,$Fsubj.)='Y';  
run;
```

```
data combine;  
  merge D1 (in = a) D234 (in=b);  
  by subjid;  
  if a and b;  
run;
```



BLUE code = Format library method
RED code = Standard method

```
data D234;  
  set D2 D3 D4;  
  by subjid;  
run;
```

```
data combine;  
  merge D1 (in = a) D234 (in=b);  
  by subjid;  
  if a and b;  
run;
```

CPU time: 11.24s. Real time: 2m37s CPU time: 12.25s. Real time: 5m53s

Effecting Efficiency Effortlessly



Thanks for listening!