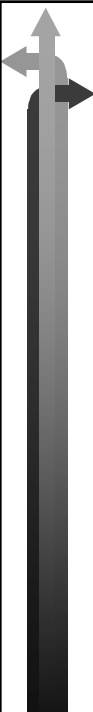


Tips & Techniques for Large Files



Andrew H. Karp
Sierra Information Services, Inc.
Sonoma, California
www.SierraInformation.com

Hawai'i SAS® Users Group
March 18, 2003



Tips & Techniques for Large Files

- Presents *options* you should *consider* when using SAS® Software for handling “large” files.
- Presents an overview of a variety of topics and approaches, instead of presenting only a few approaches in depth



Efficiency Elements

- Efficiency is composed of many elements
- The goal is to balance all elements
 - Priorities need to be set for each situation
 - Applying a standard series of approaches to every programming requirement involving a large file is usually *inefficient*
- Identify the best solution
 - by understanding the specific problems at hand
 - from many viewpoints



Large File Characteristics

- When does a file become “large”?
 - Depends on your computing environment
 - Mainframe systems can process files with millions of observations and thousands of variables
 - Batch processing using tape storage is still the standard method by which most programs process data in the mainframe environment
 - Computing environments using micro or mini-computers the point at which the file becomes “large” is heavily system-capacity dependent.



Large File Characteristics

- A large file may be:
 - “short and wide:”
 - relatively few observations
 - relatively large number of variables
 - “long and narrow”
 - relatively large number of observations
 - relatively few variables



Working With Large Files is Easier When....

- Working with large files is much easier when the programmer has a high level of knowledge about:
 - contents of the data set
 - How were the data collected?
 - How are values of key variables defined?
 - Could the definitions change over time?
 - SAS System capabilities
 - the operating environment(s) in which the programs and applications will run



Working With Large Files is Easier When....

- Without enough information about a large file, just knowing where to start is often difficult
 - introduces both human and machine *inefficiency* in to the programming process
- Deliberate planning for careful and efficient processing of data from large files on projects with high repetition values leads to improved utilization of both machine and human resources



One “Core” Strategy for Working With Large Files

- Test your code on an extract from the large file, not on the large file itself
 - Reduces costs to run many programs
 - Reduces time required to “see the results”
 - Eliminates the need for
 - repeated tape mounts
 - excessive I/O and CPU resource utilization
 - scheduling the jobs to run in an “overnight” or other “slow queue.”



SAS System Tools for Obtaining an Extract

- Two potential ways to obtain an extract from a large file are:
 - select the first n records from an existing data set
 - choose a random sample from an existing data set
 - depending on your needs, a stratified, or other more complex random sample, could be selected



SAS System Tools for Obtaining an Extract

- Using the OBS = option
 - INFILE Statement
`INFILE RAWDATA OBS=1000;`
 - SAS Data Set Option
`PROC MEANS
DATA=MY.ReallyBigDataSet(OBS=100);`
 - `OPTIONS OBS=MAX;`



SAS System Tools for Obtaining an Extract

- Selecting a random sample
 - Several SAS *functions* in the BASE SAS product are available to use in selecting a random sample.
 - Of these, the UNIFORM(*seed*) function is perhaps the most commonly used.
- Use PROC SURVEYSELECT
 - Version 8
 - SAS/STAT Module



Obtaining a Simple Random Sample

- The UNIFORM(*seed*) function is used in a DATA STEP to obtain a “pseudo random sample” of some desired proportion of observations in a SAS data set.
- The *seed* value is used to initialize the random number generator which the SAS System will use to pick cases.
 - The value of *seed* must be either zero or an odd number. If you use zero, you get a different random sample each time the program runs.



Obtaining a Simple Random Sample

To obtain a 10 per cent random sample from a tape data set:

```
//DATA DD DSN=ANDREW.MY.BIGTAPE.DATASET,DISP=SHR,UNIT=TAPE  
//DISK DD DSN=ANDREW.MY.DISK.FILES,DISP=OLD
```

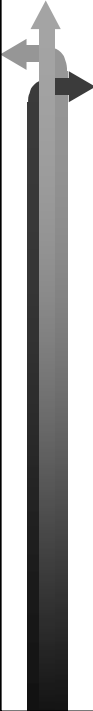
```
DATA DISK.SAMPLE;  
  SET DATA.TAPE;  
  IF UNIFORM(94115) LE .10;  
  RUN;
```

```
PROC SURVEYSELECT;
```



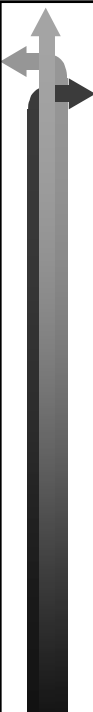
Using the Random Sample

- Once the appropriate random sample has been selected and “written off” to storage (most likely on a hard drive), write and test your code against it, rather than the entire data set
- What are some general strategies to follow now that a sample of the large file is available for use?



Some General Strategies for Maximum Efficiency

- To achieve maximum efficiency when working with large files:
 - plan, experiment and then revise the plan
 - work with sample data
 - stay flexible
 - document everything



Some General Strategies for Maximum Efficiency

- Think First
 - Allow time for planning, experimentation and more planning
 - The more you know about your
 - hardware environment
 - software capabilities
 - data
 - the better prepared you will be to design efficient programs
 - Experimenting with actual data is often the only way to determine the best approach



Some General Strategies for Maximum Efficiency

- Understand Your Data
 - Often overlooked until it is too late
 - Insist that those with whom you are working:
 - explain data collection processes and define key variables
 - share their understandings of key parts of the analytic process
 - indicate how values and variables might change later in the analytic process
 - Merely working from a file layout or PROC CONTENTS listing is *never enough!*



Some General Strategies for Maximum Efficiency

- Be flexible
 - The solution for the prior project may not be the right solution for *this* project
 - Keep an open mind about everything!
 - Examine your most fundamental assumptions
- Document ***everything!***
 - SAS data sets are "self-documenting"
 - LABELs
 - FORMATs



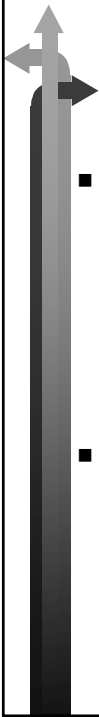
Some General Strategies for Maximum Efficiency

- Documentation (continued)
 - Use comments in programs
 - Label variables and format them, as appropriate
 - Use TITLES and FOOTNOTES in printed output.



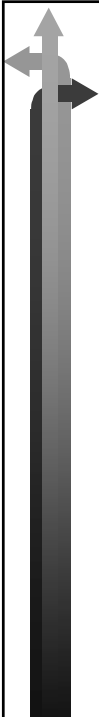
Some General Suggestions on Programming Strategies

- SAS System software changes...make sure you know what release(s) are available to you, as well as which modules are installed at your site
 - Review the appropriate "Changes and Enhancements" documents for the most current release(s) at your site.
- Become familiar with the SAS System "companion" for your operating systems(s)



Some General Suggestions on Programming Strategies

- Have a “code review” session with other programmers
 - at your site
 - at a local or regional SAS user group meeting
 - via SAS-L
- Become familiar with SAS resources on the Internet and World Wide Web:
 - SAS-L and the File Contribution Server
 - SAS Institute Web Site (www.sas.com)



Storage Strategies

- Data storage is another major aspect of developing efficient solutions to large file projects.
 - Where the data are stored can significantly impact costs and time required to complete the project.
- The SAS System offers several tools we will discuss in further detail:
 - compression
 - indexing
 - access to other files stored in other data base systems

Programming Strategies and SAS System Software Features



- Here are some
 - programming strategies
 - SAS System Features
- you may want to consider when working on large data files with SAS System software.

Programming Strategies

- Important core concepts to remember:
 - The SAS System is composed of Data Steps and Procedure Steps
 - Data Steps Create Data Sets
 - Procedures Operate on Data Sets
 - and sometimes create new Data Sets
 - Data Sets are composed of
 - descriptor portion
 - variables
 - observations



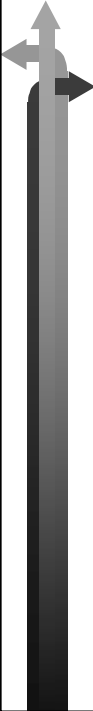
Programming Strategies

- Core concepts, continued:
 - If you are working on *observations*, use the Data Step
 - If you are working on *variables*, use a Procedure
 - In many situations using a Procedure will be easier than duplicating the functionality of the Procedure in the Data Step
 - example: PROC MEANS, PROC REPORT
 - Understand the descriptor portion of a SAS data set



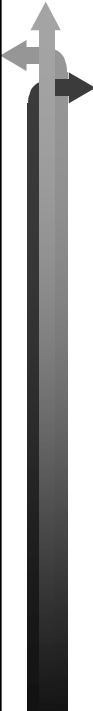
Programming Strategies

- The SAS System creates a Data Set, one observation at a time, from either raw data or from an existing SAS data set
- The SAS System executes statements in the *exact order* they are encountered in your program
- Maximize the number of *operations* performed on the minimum number of *observations* required to perform the desired tasks



Read Raw Data Selectively

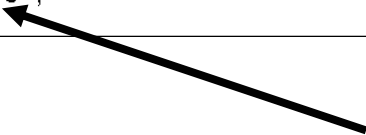
- When creating SAS data sets from a flat files, test the values of some of the variables before writing ALL variables in to the PDV
 - Useful when creating a SAS data set from a subset of records in a raw data file



Reading Raw Data Selectively

```
DATA PRIMEPAT(DROP=CN1-CNT4);  
  INFILE PRIMEPAT;  
  INPUT  @008 PPAU1  $7.  
         @050 HHTYPE $1.  
         @139 CNT1   PD4.  
         @150 CNT2   PD4.  
         @161 CNT3   PD4.  
         @172 CNT4   PD4.  
         @725 TOTDEP PD7.2  
         @723 MVCODE 2.;  
  
  NUMTRANS=SUM(OF CNT1-CNT4);  
  IF SUBSTR(PPAU1,3,1) IN ('A','T')  
    THEN PPAU1='WF0' || SUBSTR(PPAU1,4,4);  
  IF HHTYPE = 'S';  
  RUN;
```

Approach 1



Reading Raw Data Selectively

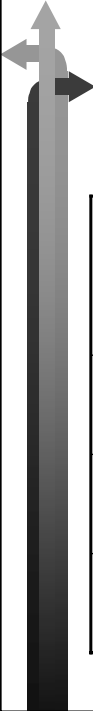
Approach 2

```
DATA PRIMEPAT(DROP=CNT1-CNT4);  
INFILE PRIMEPAT;  
INPUT @008 PPAU1 $7.  
      @050 HHTYPE $1.  
      @139 CNT1 PD4.  
      @150 CNT2 PD4.  
      @161 CNT3 PD4.  
      @172 CNT4 PD4.  
      @725 TOTDEP PD7.2  
      @723 MVCODE 2.;  
  
IF HHTYPE = 'S';  
NUMTRANS=SUM(OF CNT1-CNT4);  
IF SUBSTR(PPAU1,3,1) IN ('A','T')  
THEN PPAU1='WF0' || SUBSTR(PPAU1,4,4);  
RUN;
```

Reading Raw Data Selectively

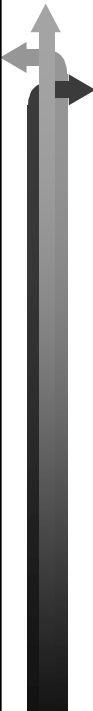
Approach 3

```
DATA PRIMEPAT(DROP=CNT1-CNT4);  
INFILE PRIMEPAT;  
INPUT @050 HHTYPE $1. @;  
IF HHTYPE NE 'S' THEN DELETE;  
ELSE  
INPUT @008 PPAU1 $7.  
      @139 CNT1 PD4.  
      @150 CNT2 PD4.  
      @161 CNT3 PD4.  
      @172 CNT4 PD4.  
      @725 TOTDEP PD7.2  
      @723 MVCODE 2.;  
NUMTRANS=SUM(OF CNT1-CNT4);  
IF SUBSTR(PPAU1,3,1) IN ('A','T')  
THEN PPAU1='WF0' || SUBSTR(PPAU1,4,4);  
DROP MVCODE CNT1-CNT4;  
RUN;
```



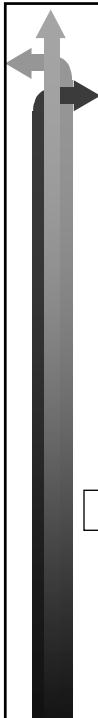
Reading Raw Data Selectives

Metric	Approach	Mean of Five Trials	Max	Min	Range
CPU	1	59.23	61.21	58.50	2.71
CPU	2	47.56	49.43	46.83	2.60
CPU	3	22.26	23.15	21.73	1.42



Programming Strategies

- Avoid the Data Step wherever possible.
 - Use Procedures
 - Use PROC DATASETS to manage the descriptor portion of an existing data set, or to manage libraries of SAS data sets.
 - Avoid unnecessary data steps
 - Explore PROC SQL as an alternative to the Data Step



Programming Strategies: Examples

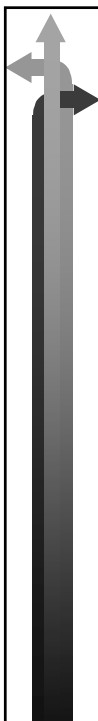
- Avoid unnecessary Data Steps

HARD WAY

```
DATA MALES;  
    SET BIGDATA;  
    IF GENDER = 'MALE';  
DATA FEMALES;  
    SET BIGDATA;  
    IF GENDER = 'FEMALE';
```

EASY WAY

```
DATA MALES FEMALES;  
    SET BIGDATA;  
    IF GENDER = 'FEMALE' THEN OUTPUT FEMALES;  
    ELSE IF GENDER = 'MALE' THEN OUTPUT MALES;  
RUN;
```



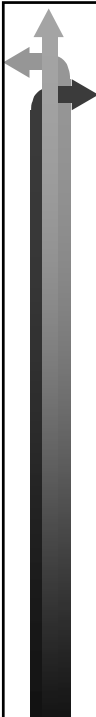
Programming Strategies: Examples

- The SAS System executes statements in the exact order they are encountered in the program

HARD WAY:

```
DATA NEWPRICE;  
    SET OLDPRICE;  
    NEWPRICE = PRICE * 1.15;  
    IF ITEM = 'X' THEN DELETE;  
    IF ITEM = 'Y' THEN DELETE;  
    IF STATE = 'DC' OR STATE = 'VA' OR STATE = 'MD';  
    IF PRICE > 10.50;  
RUN;
```

In this Data Step, all observations in Data Set OLDPRICE are operated upon, but only those which meet the testing conditions are output to NEWPRICE



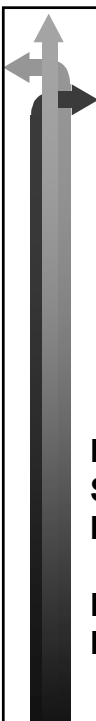
Programming Strategies: Examples

- The SAS System executes statements in the exact order they are encountered in the program

EASY WAY:

```
DATA NEWPRICE;  
    SET OLDPRICE;  
IF PRICE > 10.50 AND STATE IN('DC','MD','VA')  
    AND ITEM NOT IN('X','Y');  
NEWPRICE = PRICE * 1.15;  
RUN;
```

In this example, prices will be changed for only those observations which meet the "subsetting if" statement's tests.



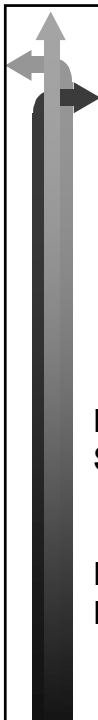
Programming Strategies: Examples

- Operate on the minimum number of observations required to accomplish the desired task

HARD WAY:

```
DATA NEWPRICES;  
SET OLDPRICE;  
IF PRICE > 10.50 AND STATE IN('DC','MD','VA')  
    AND ITEM NOT IN('X','Y');  
NEWPRICE = PRICE * 1.15;  
RUN;
```

Here, all observations in the "source" data set are read in to the PDV, but only some are output to the new data set



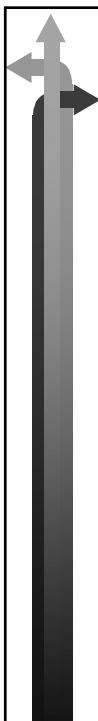
Programming Strategies: Examples

- Operate on the minimum number of observations required to accomplish the desired task

EASY WAY:

The WHERE clause restricts entry in to the PDV of only observations which meet the specified conditions.

```
DATA NEWPRICE;  
SET OLDPRICE (WHERE=(PRICE > 10.50 AND  
                     STATE IN('DC','MD','VA')  
                     AND ITEM NOT IN('X','Y')));  
NEWPRICE = PRICE * 1.15;  
RUN;
```

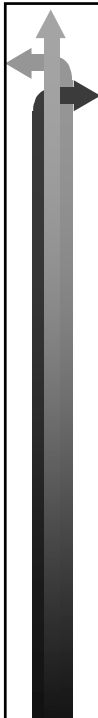


Programming Strategies: Examples

- Avoid the Data Step whenever possible

HARD WAY

```
DATA HOSPITAL.DISCHG;  
  SET HOSPITAL.DISCHG;  
  FORMAT DISDATE MMDDYY8.;  
  LABEL DISDATE = 'DISCHARGE DATE';  
RUN;
```

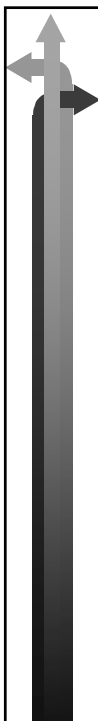


Programming Strategies: Examples

- Avoid the Data Step whenever possible

EASY WAY

```
PROC DATASETS LIBRARY = HOSPITAL;  
MODIFY DISCHG;  
FORMAT DISDATE MMDDYY8.;  
LABEL DISDATE = 'DISCHARGE DATE';  
RUN;  
QUIT;
```



Programming Strategies: Examples

- Avoid the Data Step whenever possible

HARD WAY

*** append today's file to an historical data set;**

```
DATA HOSPITAL.DISCHG;  
    SET HOSPITAL.DISCHG HOSPITAL.TODAY;  
RUN;
```



Programming Strategies: Examples

- Avoid the Data Step whenever possible

EASY WAY

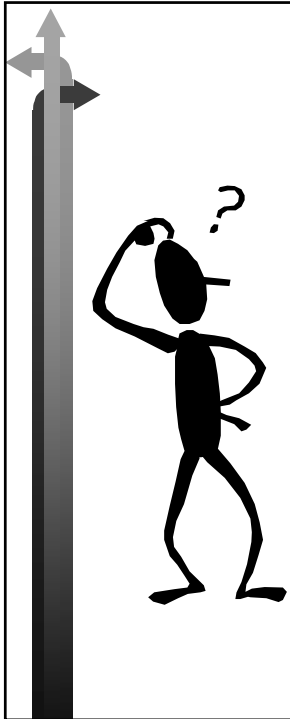
- * **append today's file to an historical data set;**

```
PROC DATASETS LIBRARY = HOSPITAL;  
APPEND BASE = DISCHG DATA=Today;  
RUN;  
QUIT;
```



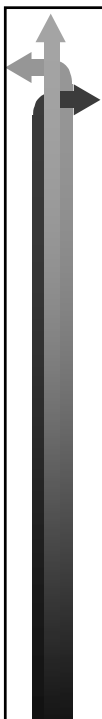
Documenting Your Data Sets

- SAS Data Sets are "self documenting"
 - the Descriptor Portion contains key information about your data set:
 - when created
 - n of observations
 - names and locations of variables
 - labels and formats
 - whether or not it is:
 - compressed
 - sorted
 - Put comments in your programs!



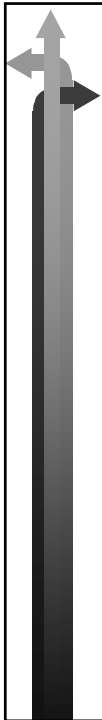
Avoid Confusion !

- Use consistent naming conventions
- Adopt common prefixes for groups of variables
- Sequentially number related groups of variables

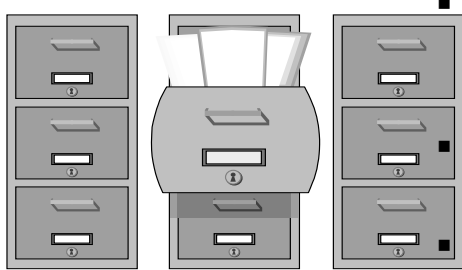


Avoid Confusion !

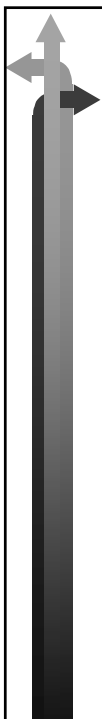
- Example: orders placed at a mail order catalog firm: up to four orders stored on a single customer record.
- HARD WAY:
FI_ORDER, S_ORDER, T_ORDER, FO_ORDER
- EASY WAY:
ORDER1, ORDER2, ORDER3, ORDER4



Some Storage Hints

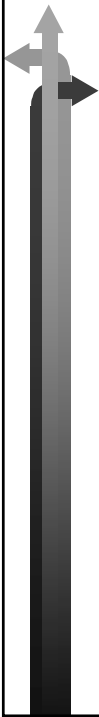


- Limit the number of variables in the Data Set
- Compress observations
- Reduce the byte length of variables



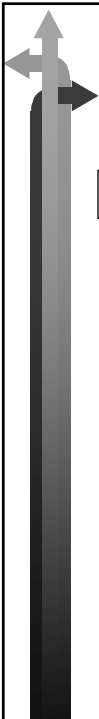
Some Storage Hints

- Limit the number of variables in the Data Set to those which you really need in subsequent steps in the process
 - Use KEEP and/or DROP Data Set *options* on the "source" SAS Data Set to reduce the number of variables in the Program Data Vector
 - Use KEEP and/or DROP statements in the Data Step to prevent unnecessary variables from being created in the new data set you are creating.



Example:Using KEEP and DROP Statements

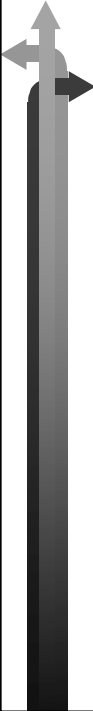
- User needs to sum the Kilowatt Hour (KwH) consumption for 12 months by region, for only the Eastern and Southern Regions of an electrical utility. The only variables required at the end of the analysis are: REGION and SUM of the 12 monthly values. Over 100 variables exist for each observation in the data set.



Example:Using KEEP and DROP Statements

HARD WAY:

```
DATA A;  
  SET BIGFILE;  
  TOTKWH = MTH1 + MTH2 + MTH3 + MTH4 + MTH5+ MTH6 +  
    MTH7 + MTH8 + MTH9 + MTH10  
    + MTH11 + MTH12;  
  IF REGION IN('SOUTHERN','EASTERN');  
  RUN;  
  
PROC MEANS DATA=A SUM;  
  CLASS REGION;  
  VAR TOTKWH;  
  TITLE 'Electricity Consumption Report';  
  RUN;
```



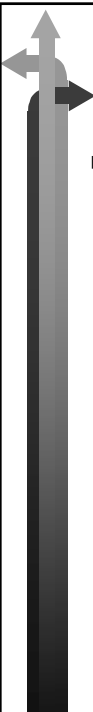
Example: Using KEEP and DROP Statements

EASY WAY:

```
DATA A(KEEP= REGION TOTKWH);  
SET BIGFILE(WHERE=(REGION IN('EASTERN','SOUTHERN')  
KEEP = REGION MTH1-MTH12);  
TOTKWH = SUM(OF MTH1-MTH12);  
RUN;
```

```
PROC MEANS DATA=A SUM;  
CLASS REGION;  
VAR TOTKWH;  
TITLE 'Electricity Consumption';  
RUN;
```

*Why is the VAR
Statement not
strictly required?*



Compress Observations

- The COMPRESS Data Set option can be used to reduce the size of SAS data sets.
 - Removes blanks spaces and repeating values, resulting in reduced storage space requirements
 - Best if used with data sets containing many, long character variables
 - Can increase the data set size if used on data sets which are comprised of many numeric variables and no/few character variables (Revised in V8)



Compress Observations

- Compress Data Set option, continued
 - additional CPU resources required for the SAS System to “uncompress” the data set before it is read by a Data Step or used by a PROC
 - Access by observation number is not possible with a compressed data set
 - Suggestion: best applied on infrequently used data sets which have many long character variables



Using an INDEX with a SAS Data Set

- Creating an INDEX for a SAS Data Set is suitable when:
 - the data set does not change often
 - is not updated frequently
 - does not have records added/removed regularly
 - the values of the variable(s) for which the INDEX will be created are uniformly distributed
 - the data set is frequently analyzed using selection or sort variables



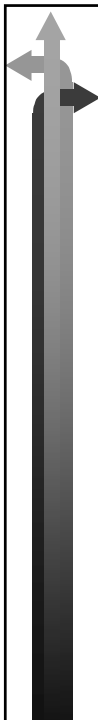
Using an INDEX with a SAS Data Set

- An INDEX will:
 - reduce the time required to find observations meeting conditions in a WHERE clause
 - eliminate the need to sort a Data Set upon which BY group processing will be applied
- Drawbacks:
 - Additional CPU utilization and storage are required
 - The INDEX will need to be repaired (using PROC DATASETS) if the values of the indexed variables change



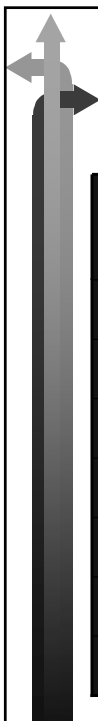
Reduce the Byte Length of Observations

- Probably the most common way to reduce the storage requirements of a SAS Data Set
 - accomplished by using the LENGTH statement in a Data Step
 - Placed immediately after the DATA statement
- Very useful for numeric variables that tend to have small values and for character variables with just a few letters or symbols.



Reduce the Byte Length of Observations

- Very effective way to reduce the size of your data sets IF.....
 - You know the System-dependent minimum byte length for numeric variables
 - You know the largest feasible value of the variables for which the LENGTHs will be changed



Integers and Byte Lengths in V8

Largest Integer Represented Exactly in a Given Byte Length

Byte Length	Windows And UNIX	OS/390 (Mainframe)
2	N/A	256
3	8,192	65,536
4	2,097,152	16,777,216
5	536,870,912	4,294,967,296
6	137,438,953,472	1,099,511,627,776
7	35,184,372,088,832	281,474,946,710,656
8	9,007,199,254,740,992	72,057,594,037,927,936



Be Efficient With PROCS

- Use PROCs instead of the Data Step
 - Example 1: Use PROC RANK to select observations from a data set
 - Example 2: Create multiple data sets in a single use of PROC MEANS
 - Use the new CHARTYPE and DESCENDTYPES options now available in PROC MEANS



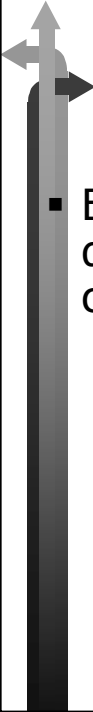
Use PROC RANK to Obtain Subsets of SAS Data Sets

- PROC RANK
 - Collapses, or categorizes the values of numeric variables in a SAS data set
 - Creates new SAS data sets, not printed output
 - new variables created by PROC RANK indicate observation "membership" in the ranking, or grouping variable
 - GROUPS= N , where N = number of groups to create



Use PROC RANK to Obtain Subsets of SAS Data Sets

- PROC RANK, continued
 - By default, the procedure ranks all numeric variables in the source data set
 - The same grouping/ranking is applied to all numeric variables on which the procedure is applied.
 - GROUPS = 2 *above/below median*
 - GROUPS = 4 *quartiles*
 - GROUPS = 5 *quintiles*
 - GROUPS = 100 *percentiles*



Use PROC RANK to Obtain Subsets of SAS Data Sets

- Example: From an electrical utility revenue data set, obtain the “top five percent” of customers based on the annual revenue.
 - Approach 1: Use PROC UNIVARIATE or PROC MEANS to obtain the value of annual revenue at the 95th percentile, then use that value in a Data Step.
 - Approach 2: Use **PROC RANK**
 - How can we be sure we obtain identical results?
 - **PROC COMPARE**

Use PROC RANK to Obtain Subsets of SAS Data Sets

```

705 * calculate the 95th percentile and ;
706 * then output it to the macro symbol table;
707
708 proc univariate noprint data=c.annual;
709 var sumrev;
710 output out=new p95=p95;
711
NOTE: The data set WORK.NEW has 1 observations and 1 variables.
NOTE: The PROCEDURE UNIVARIATE used 0.33 seconds.

```

```

712 proc print data=new;
713 title 'Data Set Created by PROC UNIVARIATE';
714 run;

```

```

NOTE: The PROCEDURE PRINT used 0.05 seconds.
715
716 data _null_;
717 set new;
718 call symput('P95',trim(left(compress(round(p95)))));
719 run;

```

NOTE: Numeric values have been converted to character values at the places given by: (Line):(Column).

718:38

NOTE: The DATA statement used 0.05 seconds.

Use PROC RANK to Obtain Subsets of SAS Data Sets

Data Set Created by PROC
UNIVARIATE

OBS	P95
1	1668

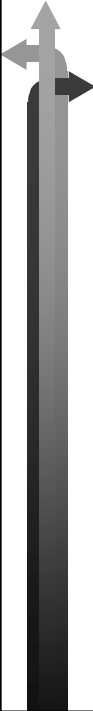
SAS Output
from PROC
UNIVARIATE

```

722 options symbolgen;
723
724 * create data set with top 5% ;
725 * of annual electric revenue customers;
726 data top5;
727 set c.annual(where=(sumrev ge &p95));
SYMBOLGEN: Macro variable P95 resolves to 1668
728 run;

```

NOTE: The data set WORK.TOP5 has 810 observations and 2 variables
NOTE: The DATA statement used 0.11 seconds.



Use PROC RANK to Obtain Subsets of SAS Data Sets

```
802  
803 * now do it using PROC RANK;  
804  
805 proc rank data=c.annual groups = 100  
      out=top5a(where=(revpct GE 95));  
806 var sumrev;  
807 ranks revpct;  
808 run;
```

NOTE: The data set WORK.TOP5A has 810 observations and 3 variables.

NOTE: The PROCEDURE RANK used 0.16 seconds.



Create Multiple Output SAS Data Sets in One Use of PROC MEANS

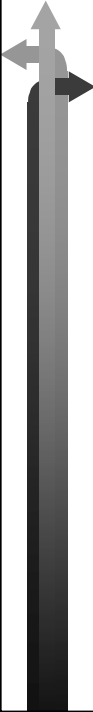
- New to Version 8 the **CHARTYPE** option simplifies creation of multiple output SAS data sets from a single use of PROC MEANS.
- This option converts the numeric values of `_TYPE_` to a character variable indicating the combination of CLASS Statement variables
 - Although this variable is composed only of zeros and ones, it is a ***character variable***



Create Multiple Output SAS Data Sets in One Use of PROC MEANS

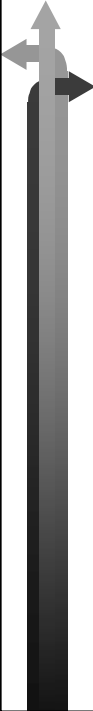
- With the CHARTYPE Option, making subset data sets with PROC MEANS in Version 8 is very simple!

```
proc means noprint data=order.orderfile2
chartype;
class mailcode dept_nbr segment status;
var itmprice itm_qty;
output out=one(where=(_type_ = '1001')) sum=;
output out=two(where=(_type_ = '1011')) sum=;
output out=three(where=(_type_ = '0110')) sum=;
run;
```



Create Multiple Output SAS Data Sets in One Use of PROC MEANS

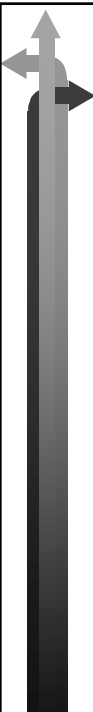
- The same value of _TYPE_ can be “re-used” when specifying the creation of multiple SAS data sets.
- Example: Add the “grand total” (observation where _TYPE_ = '0000' to each output SAS data set created in the previous example



Create Multiple Output SAS Data Sets in One Use of PROC MEANS

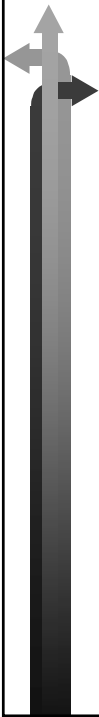
```
output out=one(where=
  (_type_ IN('0000','1001')) sum=;
output out=two(where=
  (_type_ IN('0000','1011')) sum=;
output out=three(where=
  (_type_ IN('0000','0110')) sum = ;
```

In this example, the “grand sum”
(_TYPE_ = '0000') is output to all three data sets



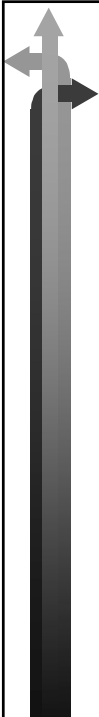
Use the DESCENDTYPES Option

- By default, observations in output data sets created by PROC MEANS are stored in ascending (lowest to highest) value of _TYPE_
- In many analytic/reporting situations, however, it would “save a PROC SORT step” if PROC MEANS would put the observation whose value of _TYPE_ is zero at the bottom of the data set.
- Using the DESCENDTYPES option, now available in SAS Version 8, does this for you.



Use the DESCENDTYPES Option

- The DESCENDTYPES option
 - Placed in the PROC MEANS Statement
 - Applies to ALL output data sets created in the OUTPUT Statements included in the task
 - Observations in the output data set are stored in descending order (highest to lowest) of the values of _TYPE_
 - Effect: puts the “grand total” at the “bottom” of the output data set



Use the DESCENDTYPES Option

```
proc print data=one;  
title1 'Tips for PROC MEANS';  
title2 'Data Set ONE w/DESCENDING  
Option';  
run;
```

Use the DESCENDTYPES Option

Tips for PROC MEANS

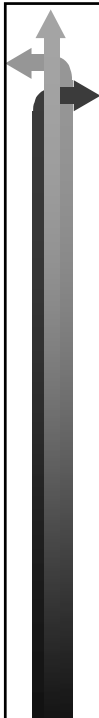
Data Set ONE w/DESCENDING Option

Obs	mailcode	DEPT_NBR	SEGMENT	status	TYPE	FREQ	ITMPRICE	ITM_QTY
1	A			1	1001	38373	1276468	48243
2	A			2	1001	10419	352074.8	13142
3	B			1	1001	18073	599346.1	22430
4	B			2	1001	4990	163952.0	6181
5	C			1	1001	44059	1464840	55035
6	C			2	1001	11828	393405.6	14858
7	D			1	1001	23154	773338.5	28719
8	D			2	1001	6270	208887.5	7943
9	E			1	1001	17039	563429.0	21305
10	E			2	1001	4404	148922.4	5491
11	F			1	1001	36504	1219881	45501
12	F			2	1001	9953	338758.9	12427
13	G			1	1001	16644	551935.7	20723
14	G			2	1001	4407	146974.1	5555
15	H			1	1001	21831	729721.2	27223
16	H			2	1001	5754	187638.1	7238
17	I			1	1001	49078	1637716	61109
18	I			2	1001	13212	438955.3	16382
19	.			.	0000	335992	11196244	419505

Summary and Conclusions



- We've presented a lot of information about approaches to and tools you can use to apply the SAS System to large data sets
- Remember:
- **Plan your Work, and Work your Plan !**



- comp.soft-sys.sas
- www.sconsig.com
- www.sashelp.com
- www.laurenhaworth.com
- www.sierrainformation.com

- www.restechonline.com/msug