Best Practices Using BASE SAS® Software

THE POWER TO KNOW®

Chapter 1: Best Practices

- 1.1 Introduction
- 1.2 Techniques for Conserving CPU and Memory
- 1.3 Techniques for Minimizing I/O Operations
- 1.4 Techniques for Conserving Disk Space
- 1.5 Creating and Using Indexes with SAS Data Sets
- 1.6 Techniques to Minimize Network Traffic (Self-Study)

2

What Are Best Practices?

Best practices can reduce usage of the following five critical system resources to improve performance:

- CPU
- I/O
- Disk Space
- Memory
- Data Storage Space

⚠ Reducing one resource often increases another.

3

Deciding What Is Important for Efficiency

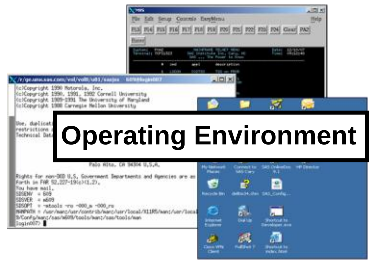
Your Programs

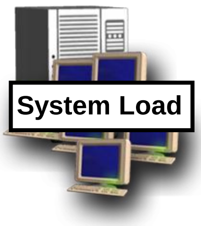
Your Site

Your Data

4

Understanding Efficiency at Your Site





5

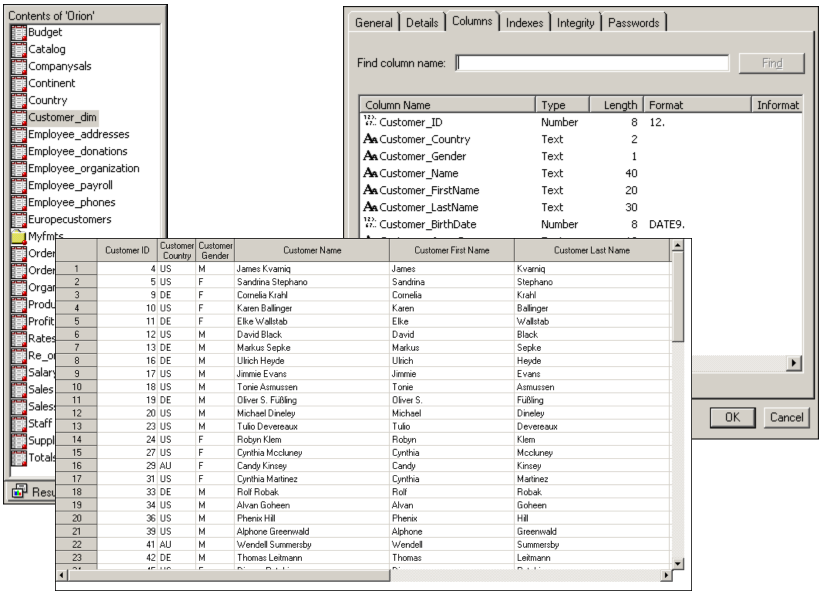
Knowing How Your Program Will Be Used

The importance of efficiency increases with the following:

- the complexity of the program or the size of the files being processed
- the number of times that the program will be executed

6

Knowing Your Data



Considering Trade-Offs

In this seminar, many tasks are performed using one or more techniques.

To decide which technique is most efficient for a given task, *benchmark* (measure and compare) the resource usage of each technique.

You should benchmark with the actual data to determine which technique is the most efficient.

⚠ The effectiveness of any efficiency technique depends greatly on the data with which you use the technique.

Running Benchmarks: Guidelines

To benchmark your programming techniques, do the following:

- Turn on the appropriate options to report resource usage.
- Test each technique in a separate SAS session.
- Test only one technique or change at a time, with as little additional code as possible.
- Run your tests under the conditions that your final program will use (for example, batch execution, large data sets, and so on).

9

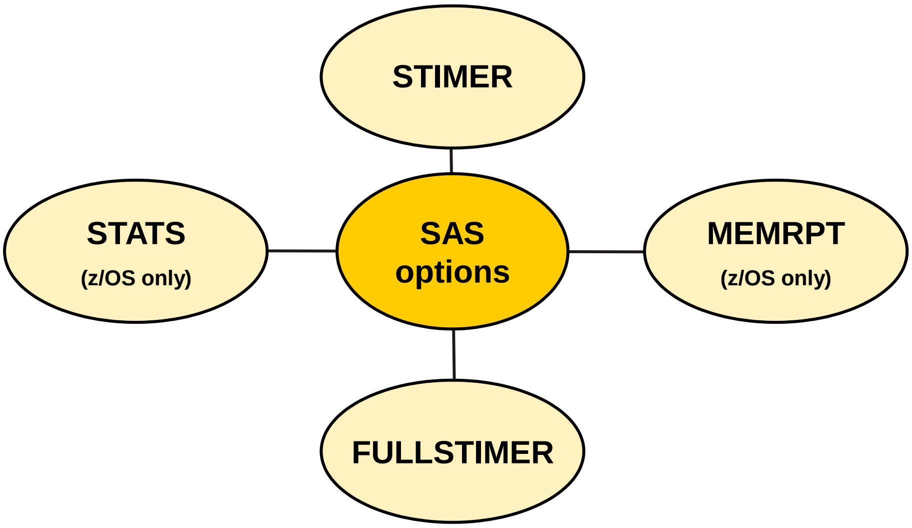
continued...

Running Benchmarks: Guidelines

- Run each program several times and base your conclusions on averages, not on a single execution. (This is even more critical when you are benchmarking elapsed time.)
- Exclude outliers from the analysis because that data might lead you to tune your program to run less efficiently than it should.
- Turn off the options that report resource usage after testing is finished, because they consume resources.

10

Tracking Resource Usage



11

Tracking Resources with SAS Options

Windows, UNIX, and z/OS

```
OPTIONS NOFULLSTIMER | FULLSTIMER;
```

```
OPTIONS STIMER | NOSTIMER;
```

Invocation or configuration file only on z/OS

z/OS only

```
OPTIONS STATS | NOSTATS;
```

```
OPTIONS MEMRPT | NOMEMRPT;
```

12

Sample Windows Log

Partial SAS Log

```
options fullstimer;
data _null_;
   length var $ 30;
   retain var2-var50 0 var51-var100 'ABC';
   do x=1 to 100000000;
      var1=10000000*ranuni(x);
      if var1>1000000 then var='Greater than 1,000,000';
      if 500000<=var1<=1000000 then var='Between 500,000 and 1,000,000';
      if 100000<=var1<500000 then var='Between 100,000 and 500,000';
      if 10000<=var1<100000 then var='Between 10,000 and 100,000';
      if 1000<=var1<10000 then var='Between 1,000 and 10,000';
      if var1<1000 then var='Less than 1,000';
   end;
run;

NOTE: DATA statement used (Total process time):
      real time           1.26 seconds
      user cpu time       0.98 seconds
      system cpu time     0.04 seconds
      Memory                             278k
      OS Memory                          4976k
      Timestamp            6/29/2010  12:39:21 PM

options nofullstimer;
```

13

Sample UNIX Log

Partial SAS Log

```
options fullstimer;
data _null_;
   length var $30;
   retain var2-var50 0 var51-var100 'ABC';
   do x=1 to 10000000;
      var1=10000000*ranuni(x);
      if var1>10000000 then var='Greater than 1,000,000';
      if 500000<=var1<=1000000 then var='Between 500,000 and 1,000,000';
      if 100000<=var1<500000 then var='Between 100,000 and 500,000';
      if 10000<=var1<100000 then var='Between 10,000 and 100,000';
      if 1000<=var1<10000 then var='Between 1,000 and 10,000';
      if var1<1000 then var='Less than 1,000';
   end;
run;

NOTE: DATA statement used (Total process time):
      real time           6.62 seconds
      user cpu time       5.14 seconds
      system cpu time     0.01 seconds
      Memory                             526k
      OS Memory                          5680k
      Timestamp            6/29/2010  11:55:32 AM
      Page Faults                        82
      Page Reclaims                      0
      Page Swaps                         0
      Voluntary Context Switches         91
      Involuntary Context Switches       48
      Block Input Operations             91
      Block Output Operations            0

options nofullstimer;
```

14

Sample z/OS Log

Partial SAS Log

```
+Log-----------------------------------------------------------------------+
 Command ===>

 1     options fullstimer;
 2     data _null_;
 3        length var $ 30;
 4        retain var2-var50 0 var51-var100 'ABC';
 5        do x=1 to 10000000;
 6           var1=10000000*ranuni(x);
 7           if var1>1000000 then var='Greater than 1,000,000';
 8           if 500000<=var1<=1000000 then var='Between 500,000 and 1,000,000'
 8  ! ;
 9           if 100000<=var1<500000 then var='Between 100,000 and 500,000';
 10          if 10000<=var1<100000 then var='Between 10,000 and 100,000';
 11          if 1000<=var1<10000 then var='Between 1,000 and 10,000';
 12          if var1<1000 then var='Less than 1,000';
 13       end;
 14    run;

 NOTE: The DATA statement used the following resources:
       CPU      time -         00:00:16.66
       Elapsed time -         00:00:17.98
       EXCP count   - 387
       Task  memory - 4558K (131K data, 4427K program)
       Total memory - 17853K (3744K data, 14109K program)
 NOTE: The address space has used a maximum of 660K below the line and
       20516K above the line.
```

15

SAS Data Set Pages

A SAS *data set page* has the following attributes:

- It is the unit of data transfer between the operating system buffers and SAS buffers in memory.
- It includes the number of bytes used by the descriptor portion, the data values, and any operating system overhead.
- It is fixed in size when the data set is **created**, either to a default value or to a value specified by the programmer.

16

Using PROC CONTENTS to Report Page Size

```
proc contents data=work.sales_history;
run;
```

16,384*20,585= 337,264,640 bytes

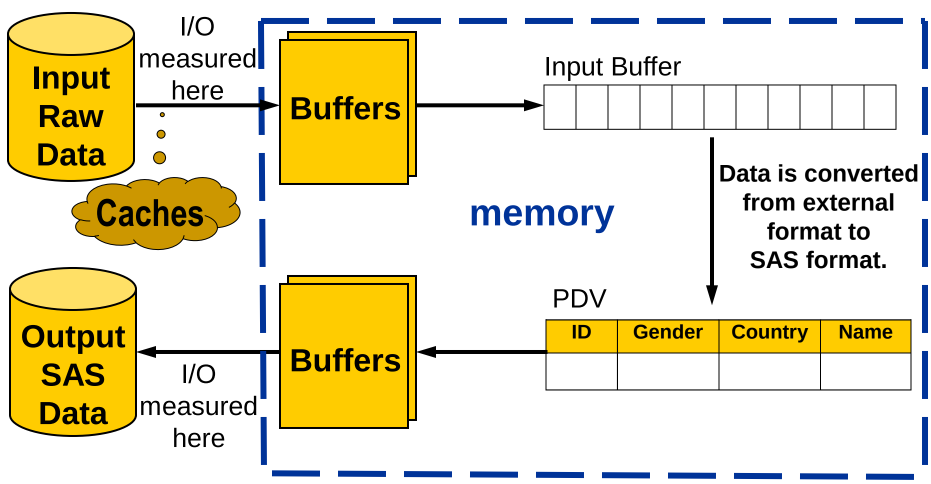
Partial PROC CONTENTS Output

```
                 Engine/Host Dependent Information

Data Set Page Size           16384
Number of Data Set Pages     20585
First Data Page              1
Max Obs per Page             92
Obs in First Data Page       72
Number of Data Set Repairs   0
File Name                    c:\sas\...\sales_history.sas7bdat
Release Created              9.0201M0
Host Created                 XP_PRO
```

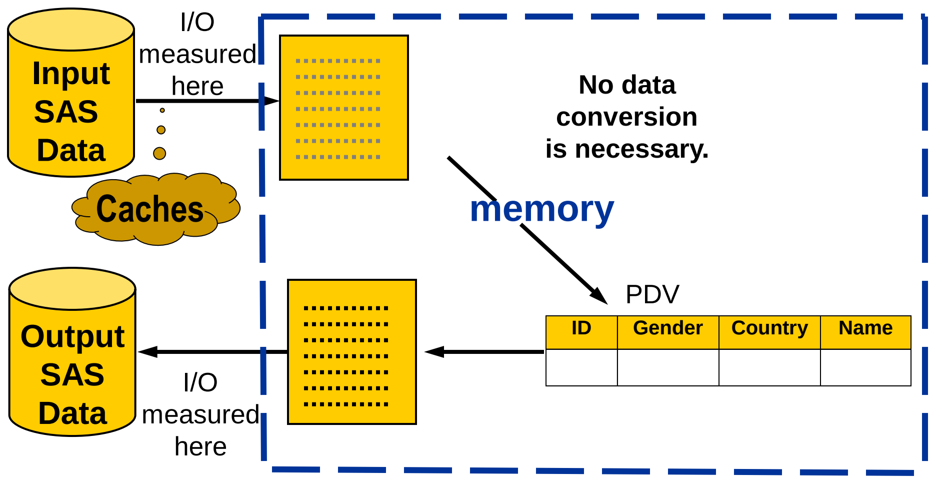
17

Reading External Files



18

Reading a SAS Data Set with a SET Statement



19 ...

Chapter 1: Best Practices

1.1 Introduction

1.2 Techniques for Conserving CPU and Memory

1.3 Techniques for Minimizing I/O Operations

1.4 Techniques for Conserving Disk Space

1.5 Creating and Using Indexes with SAS Data Sets

1.6 Techniques to Minimize Network Traffic (Self-Study)

20

Techniques for Conserving CPU and Memory

- Execute only the necessary statements.
- Eliminate unnecessary passes of the data.
- Read and write only the data that you require.
- Do not reduce the length of numeric variables.
- Do not compress SAS data sets.
- Use Indexes

21

Execute Only Necessary Statements

You minimize the CPU time that SAS uses when you execute the minimum number of statements in the most efficient order.

Techniques for executing only the statements that you require include the following:

- subsetting your data as soon as logically possible
- processing your data conditionally by using the most appropriate syntax for your data

22

Subsetting IF Statement at Bottom of Step

Create a new SAS data set from **work.sales**. The new SAS data set should contain four new variables and only those flights filled to less than 80% capacity.

```
data totals;
   set work.sales;
   PercentCap =
      sum(Num1st,NumEcon,NumBus)/CapPassTotal;
   NumNonEconomy = sum(Num1st,NumBus);
   CargoKG = CargoWeight*0.454;
   Month = month(FltDate);
   if PercentCap < 0.8;
run;
```

23

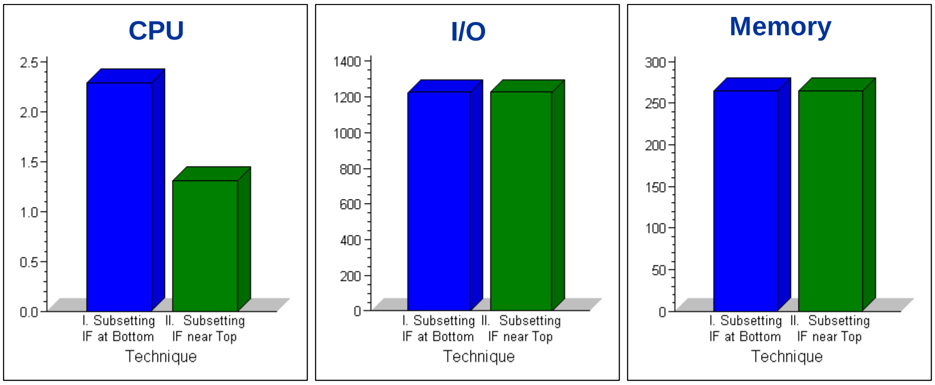
Subsetting IF Statement as High as Possible

```
data totals;
   set work.sales;
   PercentCap =
      sum(Num1st,NumEcon,NumBus)/CapPassTotal;
   if PercentCap < 0.8;
   NumNonEconomy = sum(Num1st,NumBus);
   CargoKG = CargoWeight*0.454;
   Month = month(FltDate);
run;
```

24

Comparing Techniques

Technique	CPU	I/O	Memory
I. Subsetting IF at Bottom	2.3	1226.0	265.0
II. Subsetting IF near Top	1.3	1226.0	265.0
Percent Difference	42.8	0.0	0.0



25

Using Conditional Logic

You can use *conditional logic* to alter the way that SAS processes specific observations.

IF-THEN/ELSE statement	executes a SAS statement for observations that meet specific conditions.
SELECT statement	executes one of several statements or groups of statements.

26

Using Parallel IF Statements

For the data in **work.sales**, create a variable named **Month**, based on the existing variable **FltDate**.

```
data month;
   set work.sales;
   if month(FltDate) = 1 then Month = 'Jan';
   if month(FltDate) = 2 then Month = 'Feb';
   if month(FltDate) = 3 then Month = 'Mar';
   if month(FltDate) = 4 then Month = 'Apr';
   if month(FltDate) = 5 then Month = 'May';
   if month(FltDate) = 6 then Month = 'Jun';
   if month(FltDate) = 7 then Month = 'Jul';
   if month(FltDate) = 8 then Month = 'Aug';
   if month(FltDate) = 9 then Month = 'Sep';
   if month(FltDate) = 10 then Month = 'Oct';
   if month(FltDate) = 11 then Month = 'Nov';
   if month(FltDate) = 12 then Month = 'Dec';
run;
```

27

Using ELSE-IF Statements

```
data month;
   set work.sales;
   if month(FltDate) = 1 then Month = 'Jan';
   else if month(FltDate) = 2 then Month = 'Feb';
   else if month(FltDate) = 3 then Month = 'Mar';
   else if month(FltDate) = 4 then Month = 'Apr';
   else if month(FltDate) = 5 then Month = 'May';
   else if month(FltDate) = 6 then Month = 'Jun';
   else if month(FltDate) = 7 then Month = 'Jul';
   else if month(FltDate) = 8 then Month = 'Aug';
   else if month(FltDate) = 9 then Month = 'Sep';
   else if month(FltDate) = 10 then Month = 'Oct';
   else if month(FltDate) = 11 then Month = 'Nov';
   else if month(FltDate) = 12 then Month = 'Dec';
run;
```

28

Using the Function Only Once

```
data month(drop=mon);
   set work.sales;
   mon = month(FltDate);
   if mon = 1 then Month = 'Jan';
   else if mon = 2 then Month = 'Feb';
   else if mon = 3 then Month = 'Mar';
   else if mon = 4 then Month = 'Apr';
   else if mon = 5 then Month = 'May';
   else if mon = 6 then Month = 'Jun';
   else if mon = 7 then Month = 'Jul';
   else if mon = 8 then Month = 'Aug';
   else if mon = 9 then Month = 'Sep';
   else if mon = 10 then Month = 'Oct';
   else if mon = 11 then Month = 'Nov';
   else if mon = 12 then Month = 'Dec';
run;
```

29

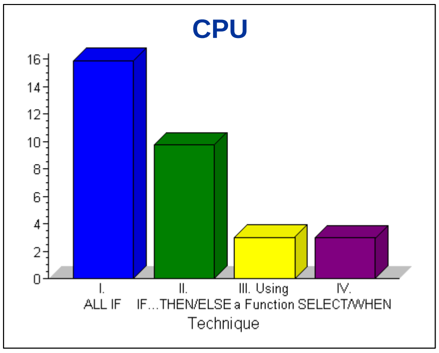
Using a SELECT Block

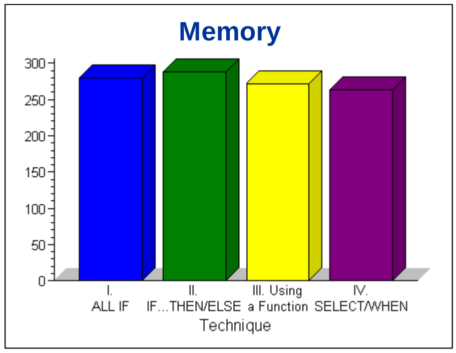
```
data month;
   set work.sales;
   select(month(FltDate));
      when(1) Month = 'Jan';  when(2) Month = 'Feb';
      when(3) Month = 'Mar';  when(4) Month = 'Apr';
      when(5) Month = 'May';  when(6) Month = 'Jun';
      when(7) Month = 'Jul';  when(8) Month = 'Aug';
      when(9) Month = 'Sep';  when(10) Month = 'Oct';
      when(11) Month = 'Nov'; when(12) Month = 'Dec';
      otherwise;
   end;
run;
```

30

Comparing Techniques

Technique	CPU	I/O	Memory
I. ALL IF Statements	15.9	6797.0	280.0
II. ELSE-IF Statements	9.7	6797.0	288.0
III. Using a Function Once	3.0	6797.0	272.0
IV. SELECT/WHEN Block	3.0	6795.0	263.0





31

The I/O for each technique is the same.

Guidelines for Efficient Conditional Logic

IF		SELECT
Character	**Type**	*Numeric*
Few	**Conditions**	*Many*
Not Uniform	**Distribution** *(check for most commonly occurring value first)*	*Uniform*

- For mutually exclusive conditions, use the ELSE-IF statement (or SELECT statement) rather than an IF statement for all conditions except the first.
- Check the most frequently occurring condition first.
- When you execute multiple statements based on a condition, put the statements into a DO group.

32

Most Frequently Occuring Condition

```
PROC FREQ  DATA=libraryname.datasetname
           ORDER=FREQ ;
   TABLES variablename;
RUN;
```

33

Do Group Processing

<u>No Do Groups - Not as Efficient</u>:

```
If Status = 1
   Then Bonus = Salary * 0.05;
Else If Status = 2
   Then Bonus = Salary * 0.06;
Else Bonus = Salary * 0.04;

If Status = 1
   Then Start_Month = month(Hire_Date);
Else If Status = 2
   Then Start_Month = 6;
Else Start_Month = 1;
```

34

Do Group Processing

Use of Do Groups – More Efficient:

```
If Status = 1  Then Do;
        Bonus = Salary * 0.05;
        Start_Month = month(Hire_Date);
  End;
Else If Status = 2  Then Do;
        Bonus = Salary * 0.06;
        Start_Month = 6;
  End;
Else Do;
        Bonus = Salary * 0.04;
        Start_Month = 1;
End;
```

35

Eliminate Unnecessary Passes of the Data

Avoid reading or writing data more than necessary in order to minimize I/O operations.

Techniques include the following:

- creating multiple output data sets from one pass of the input data, rather than processing the input data each time that you create an output data set
- creating sorted subsets with the Sort procedure
- using the SORTED BY data set option or the PRESORTED option in the PROC SORT statement to avoid sorting already ordered data

36

Multiple DATA Steps

Create six subsets from **work.sales**, one for each destination on the East Coast.

```
data rdu;
   set work.sales;
   if Dest = 'RDU';
run;
data bos;
   set work.sales;
   if Dest = 'BOS';
run;
```

continued...

37

Multiple DATA Steps

```
data iad;
   set work.sales;
   if Dest = 'IAD';
run;
data jfk;
   set work.sales;
   if Dest = 'JFK';
run;
data mia;
   set work.sales;
   if Dest = 'MIA';
run;
data pwm;
   set work.sales;
   if Dest = 'PWM';
run;
```

38

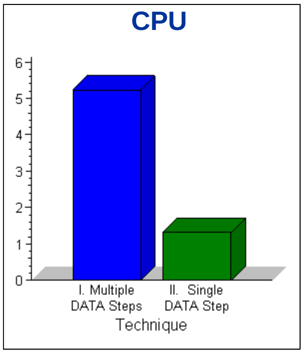
Single DATA Step

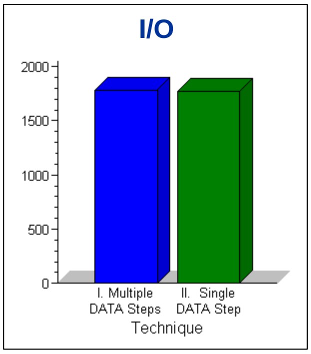
```
data rdu bos iad jfk mia pwm;
   set work.sales;
   if Dest = 'RDU' then output rdu;
   else if Dest = 'BOS' then output bos;
   else if Dest = 'IAD' then output iad;
   else if Dest = 'JFK' then output jfk;
   else if Dest = 'MIA' then output mia;
   else if Dest = 'PWM' then output pwm;
run;
```

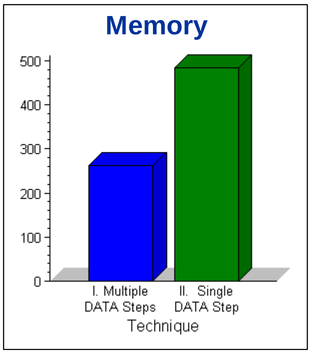
39

Comparing Techniques

Technique	CPU	I/O	Memory
I. Multiple DATA Steps	5.2	1781.0	262.0
II. Single DATA Step	1.3	1774.0	483.0
Percent Difference	74.8	0.4	-84.4







40

DATA Step / PROC SORT Step

Create a sorted subset of **work.sales** that contains the flights to the East Coast.

```
data east;
   set work.sales;
   where Dest in
        ('RDU','BOS','IAD','JFK','MIA','PWM');
run;
proc sort data = east;
   by Dest;
run;
```

41

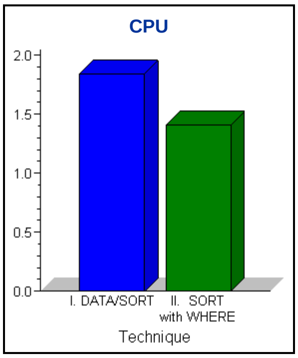
PROC SORT Step

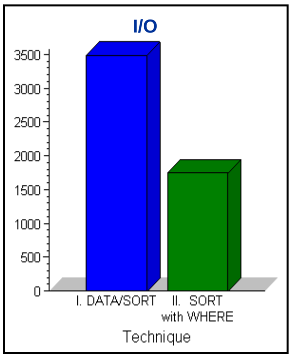
```
proc sort data = work.sales out = east;
   by Dest;
   where Dest in
        ('RDU','BOS','IAD','JFK','MIA','PWM');
run;
```

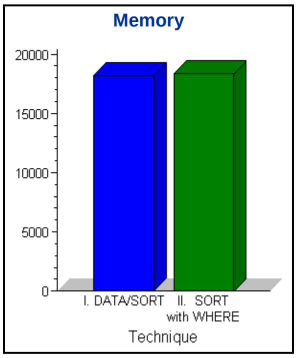
42

Comparing Techniques

Technique	CPU	I/O	Memory
I. DATA/SORT	1.8	3490.0	18199
II. SORT with WHERE	1.4	1745.0	18355
Percent Difference	23.4	50.0	-0.9







43

Using the SORTEDBY= Option

If the input data is in sorted order, you can specify the order by using the SORTEDBY= output data set option.

The SORTEDBY= option has the following attributes:

- sets the sort flag on the data set to YES
- defines the sort flag as an asserted data order
- requires that SAS check the order of the data as it processes it

General form of the SORTEDBY option:

> *data-set-name*(SORTEDBY=*by-clause* | _NULL_)

44

Using the SORTEDBY= Option

Create a SAS data set from an external file containing invoice information. The external file is in sorted order by order date.

```
filename M1 'mon1.dat';

data january(sortedby=Order_Date);
   infile M1 dlm=',';
   input Customer_ID Order_ID Order_Type
         Order_Date:date9.
         Delivery_Date:date9.;
run;
```

45

Using the SORTEDBY= Option

Partial SAS Log

```
                                The CONTENTS Procedure

Data Set Name          WORK.JANUARY                          Observations          4
Member Type            DATA                                  Variables             5
Engine                 V9                                    Indexes               0
Created                Sunday, January 27, 2008 05:36:23 PM  Observation Length    40
Last Modified          Sunday, January 27, 2008 05:36:23 PM  Deleted Observations  0
Protection                                                   Compressed            NO
Data Set Type                                                Sorted                YES
Label
Data Representation    WINDOWS_32
Encoding               wlatin1  Western (Windows)

                              <lines removed>

                              Sort Information

                           Sortedby       Order_Date
                           Validated      NO
                           Character Set  ANSI
```

46

Using the SORTEDBY= Option

Attempt to sort the data.

```
proc sort data=january;
   by Order_Date;
run;
```

Log

```
1197  proc sort data=january;
1198     by Order_Date;
1199  run;

NOTE: Input data set is already sorted, no sorting done.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds
```

47

Using the PRESORTED Option

Beginning in SAS 9.2, there is a PROC SORT statement option, PRESORTED, that checks within the input data set to determine whether the sequence of observations are in order before sorting. By specifying this option, you avoid the cost of sorting the data set.

```
proc sort data=january presorted;
   by Order_Date;
run;

proc contents data=january;
run;
```

✎ If the data set **january** is not in sorted order by **Order_Date**, PROC SORT with the PRESORTED option still sorts the data.

48

Using the PRESORTED Option

Partial Log

```
34    proc sort data=january presorted;
35       by Order_Date;
36    run;

NOTE: Sort order of input data set has been verified.
NOTE: There were 4 observations read from the data set WORK.JANUARY.
NOTE: Input data set is already sorted, no sorting done.
```

Partial PROC CONTENTS Output

```
                    Sort Information

                 Sortedby         Order_Date
                 Validated        YES
                 Character Set    ANSI
```

The SORTEDBY= option was not required when creating the data set **`january`** in order to use the PRESORTED option in PROC SORT.

49

Business Task

Change the variable attributes in **`work.salesc`** to be consistent with those in **`work.sales`**.

	Var Name	Var Format
`work.sales`	**`FlightID`**	$7.
	`FltDate`	DATE9.
`work.salesc`	**`FlightIDNumber`**	$7.
	`FltDate`	MMDDYYP10.

50

DATA Step / PROC DATASETS

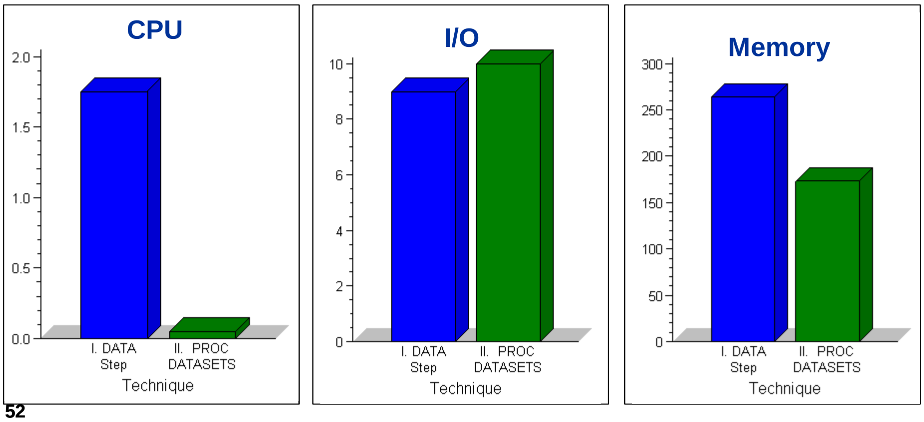
```
data work.salesc;
   set work.salesc;
   rename FlightIDNumber = FlightID;
   format FltDate date9.;
run;
```

```
proc datasets library=work nolist;
   modify salesc;
      rename FlightIDNumber=FlightID;
      format FltDate date9.;
quit;
```

51

Comparing Techniques

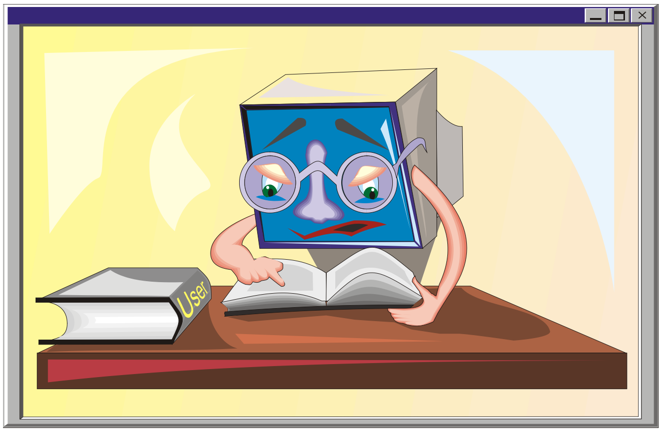
Technique	CPU	IO	Memory
I. DATA Step	1.8	9.0	264.0
II. PROC DATASETS	0.1	10.0	173.0
Percent Difference	97.1	-11.1	34.5



52

Read and Write Data Selectively

If you process fewer variables and observations,
CPU and/or I/O operations can be affected significantly.



53

Selecting Observations

WHERE **`Dest`** = "BWI"

Destination	Flight Number	Route Number
BWI	SE00007	0000206
ATL	SE0003	0000202
GSP	SE0001	0000200
BWI	SE0006	0000206

54 ...

Selecting Observations

IF **Dest** = "BWI"

Destination	Flight Number	Route Number
BWI	SE00007	0000206
ATL	SE0003	0000202
GSP	SE0001	0000200
BWI	SE0006	0000206

55 ...

Subsetting IF versus WHERE

Create a subset of the sales data that contains data for West Coast destinations.

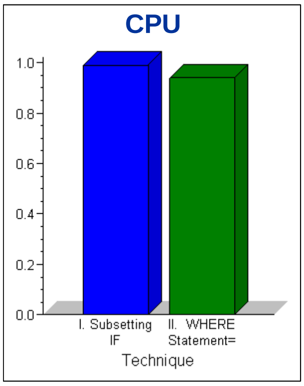
```
data west;
   set work.sales;
   if Dest in ('LAX','SEA','SFO');
run;
```

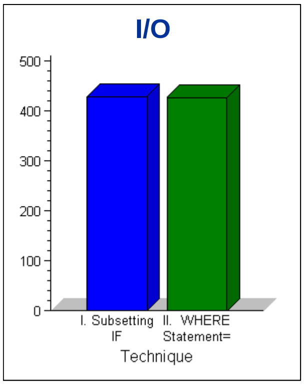
```
data west;
   set work.sales;
   where Dest in ('LAX','SEA','SFO');
run;
```

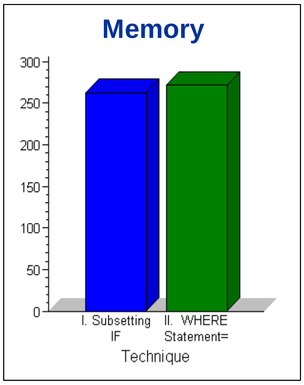
56

Comparing Techniques

Technique	CPU	I/O	Memory
I. Subsetting IF	1.0	429.0	263.0
II. WHERE Statement	0.9	427.0	272.0
Percent Difference	5.1	0.5	-3.4

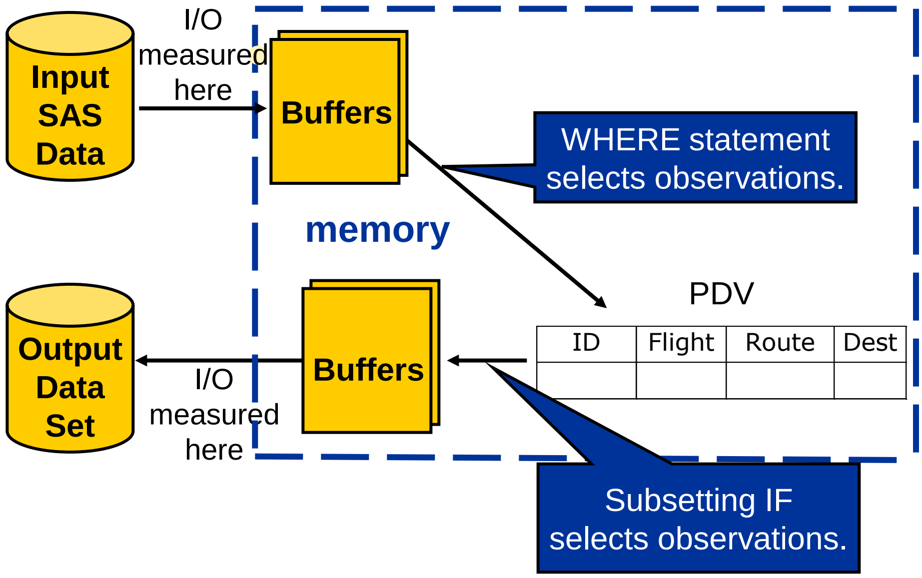






57

Subsetting IF versus WHERE Statements



58

Subsetting IF versus WHERE Statements

```
data work.ontime work.late;
  set work.custord;
  FullName = catx(' ',customer_firstname,customer_lastname);
  age = int((today()-customer_birthdate)/364.25);
  total = (total_retail_price * quantity)*discount;
  profit = total - (costprice_per_unit*quantity);
  if employee_id ne 99999999;
  if (delivery_date > order_date+4)
    then output work.late;
    else output work.ontime;
run;

NOTE: There were 973323 observations read from the data set WORK.CUSTORD.
NOTE: The data set WORK.ONTIME has 719208 observations and 26 variables.
NOTE: The data set WORK.LATE has 18416 observations and 26 variables.
NOTE: DATA statement used (Total process time):
      real time           2.26 seconds
      user cpu time       0.82 seconds
      system cpu time     0.62 seconds
      memory              439k
      OS Memory           10424k
      Timestamp           02/15/2012 09:45:02 AM
```

59

Subsetting IF versus WHERE Statements

```
data work.ontime work.late;
  set work.custord;
  if employee_id ne 99999999;
  FullName = catx(' ',customer_firstname,customer_lastname);
  age = int((today()-customer_birthdate)/364.25);
  total = (total_retail_price * quantity)*discount;
  profit = total - (costprice_per_unit*quantity);
  if (delivery_date > order_date+4)
    then output work.late;
    else output work.ontime;
run;

NOTE: There were 973323 observations read from the data set WORK.CUSTORD.
NOTE: The data set WORK.ONTIME has 719208 observations and 26 variables.
NOTE: The data set WORK.LATE has 18416 observations and 26 variables.
NOTE: DATA statement used (Total process time):
      real time           2.07 seconds
      user cpu time       0.68 seconds
      system cpu time     0.70 seconds
      memory              431k
      OS Memory           10424k
      Timestamp           02/15/2012 09:45:04 AM
```

60

Subsetting IF versus WHERE Statements

```
data work.ontime work.late;
  set work.custord;
  where employee_id ne 99999999;
  FullName = catx(' ',customer_firstname,customer_lastname);
  age = int((today()-customer_birthdate)/364.25);
  total = (total_retail_price * quantity)*discount;
  profit = total - (costprice_per_unit*quantity);
  if (delivery_date > order_date+4)
    then output work.late;
    else output work.ontime;
run;

NOTE: There were 737624 observations read from the data set WORK.CUSTORD.
      WHERE employee_id not = 99999999;
NOTE: The data set WORK.ONTIME has 719208 observations and 26 variables.
NOTE: The data set WORK.LATE has 18416 observations and 26 variables.
NOTE: DATA statement used (Total process time):
      real time           1.96 seconds
      user cpu time       0.53 seconds
      system cpu time     0.88 seconds
      memory              438k
      OS Memory           14424k
      Timestamp           02/15/2012 09:45:06 AM
```

61

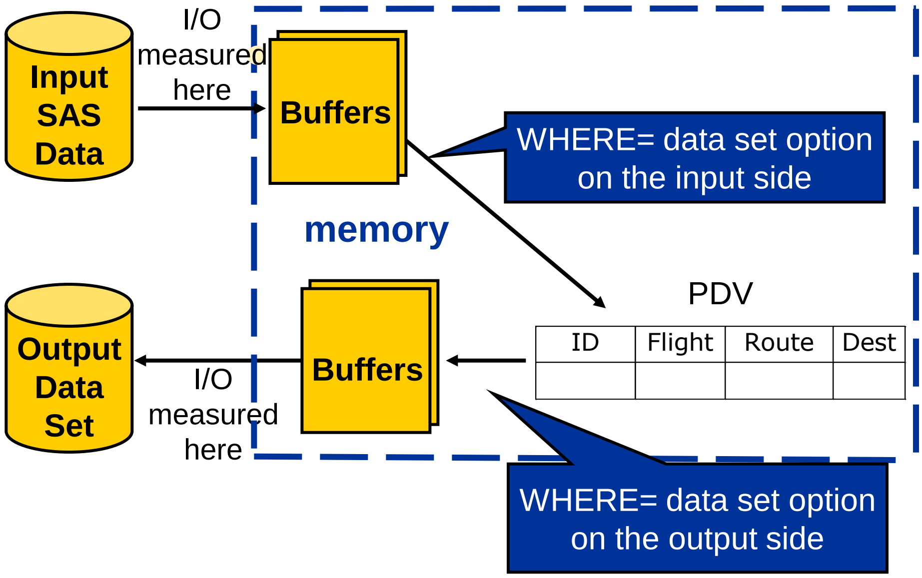
Subsetting IF versus WHERE Statements

```
data work.ontime work.late;
  set work.custord (where=(employee_id = 99999999));
  FullName = catx(' ',customer_firstname,customer_lastname);
  age = int((today()-customer_birthdate)/364.25);
  total = (total_retail_price * quantity)*discount;
  profit = total - (costprice_per_unit*quantity);
  if (delivery_date > order_date+4)
    then output work.late;
    else output work.ontime;
run;

NOTE: There were 737624 observations read from the data set WORK.CUSTORD.
      WHERE employee_id not = 99999999;
NOTE: The data set WORK.ONTIME has 719208 observations and 26 variables.
NOTE: The data set WORK.LATE has 18416 observations and 26 variables.
NOTE: DATA statement used (Total process time):
      real time           1.93 seconds
      user cpu time       0.51 seconds
      system cpu time     0.90 seconds
      memory              438k
      OS Memory           14424k
      Timestamp           02/15/2012 09:45:06 AM
```

62

The WHERE= Data Set Option



63

Subsetting an External File

Create a subset of data that contains only the flights to the West Coast.

The data is in an external file that contains information about all flights.

64

Reading All Variables and Subsetting

```
data west;
   infile rawdata ;
   input FlightID $7.  RouteID $7.
         Origin $3.  Dest $3.
         DestType $13.  FltDate date9.
         Cap1st 8.  CapBus 8.
         CapEcon 8. CapPassTotal 8.
         CapCargo 8.  Num1st 8.
         NumBus 8. NumEcon 8.
         NumPassTotal 8. Rev1st 8.
         RevBus 8.  RevEcon 8.
         CargoRev 8. RevTotal 8.
         CargoWeight 8.;
   if Dest in ('LAX','SEA','SFO');
run;
```

65

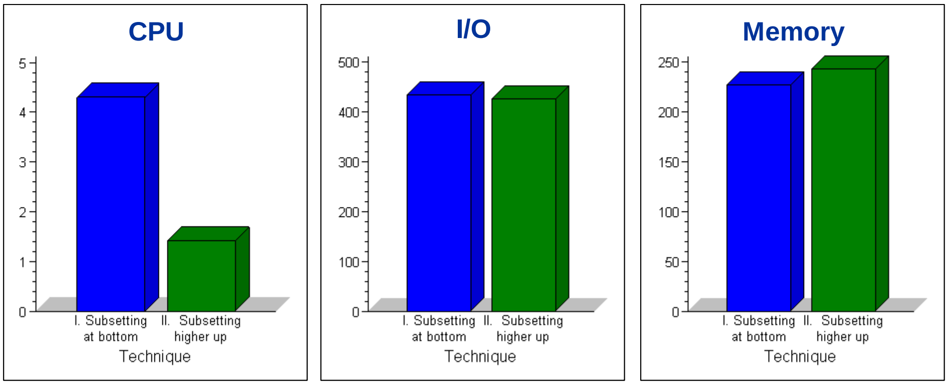
Reading Selected Variable(s) and Subsetting

```
data west;
   infile rawdata ;
   input @18 Dest $3. @;
   if Dest in ('LAX','SEA','SFO');
   input @1 FlightID $7.  RouteID $7.
         Origin $3.
         @21 DestType $13.  FltDate date9.
         Cap1st 8.  CapBus 8.
         CapEcon 8. CapPassTotal 8.
         CapCargo 8.  Num1st 8.
         NumBus 8. NumEcon 8.
         NumPassTotal 8. Rev1st 8.
         RevBus 8.  RevEcon 8.
         CargoRev 8. RevTotal 8.
         CargoWeight 8.;
run;
```

66

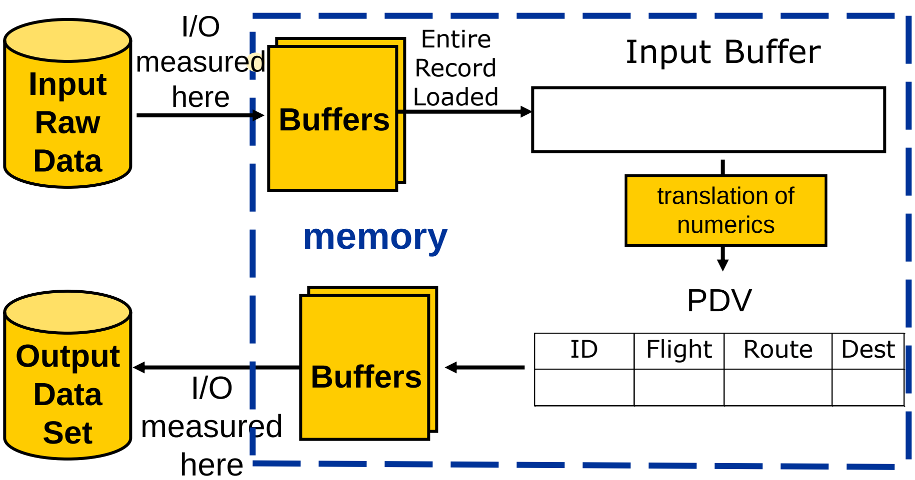
Comparing Techniques

Technique	CPU	I/O	Memory
I. Subsetting at bottom	4.3	433.0	227.0
II. Subsetting higher up	1.4	425.0	243.0
Percent Difference	67.2	1.8	-7.0



67

Reading External Files

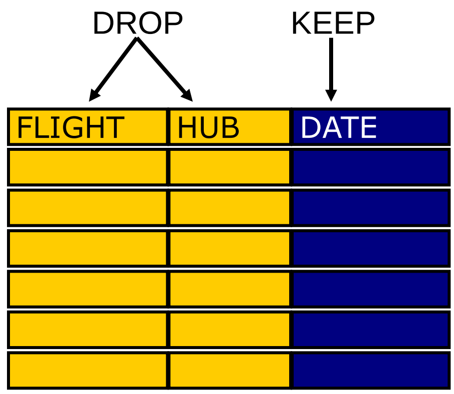


68

Subsetting Variables

To subset variables, you can use the following:

- DROP and KEEP statements
- DROP= and KEEP= data set options



69

Reading and Writing All Variables

Create a report that contains the average and median of the total number of passengers on the flights for each destination in **`work.sales`** that has 21 variables.

```
data totals;
   set work.sales;
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

70

Reading All Variables/Writing Two Variables

```
data totals(keep = Dest NonEconPass);
   set work.sales;
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

71

Reading Three Variables

```
data totals;
   set work.sales(keep = Dest Num1st
                         NumBus);
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

72

Reading Three Variables/Writing Two Variables

```
data totals(keep = Dest NonEconPass);
   set work.sales(keep = Dest Num1st
                         NumBus);
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

73

Reading Three Variables/Reading Two Variables

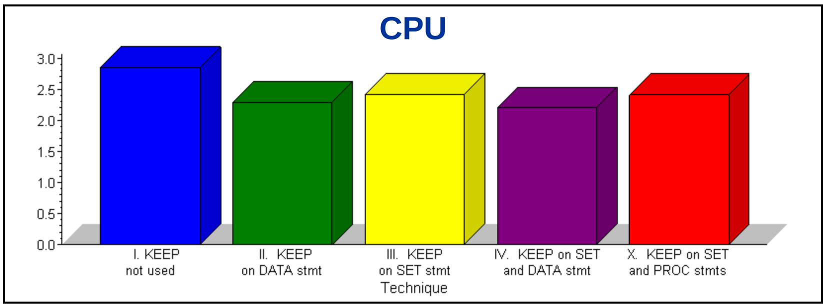
```
data totals;
   set work.sales(keep = Dest Num1st
                         NumBus);
   NonEconPass =
      sum(Num1st,NumBus);
run;

proc means data = totals
           (keep = Dest NonEconPass)
            mean median;
   title 'Non-Economy Passengers';
   class Dest;
   var NonEconPass;
run;
```

74

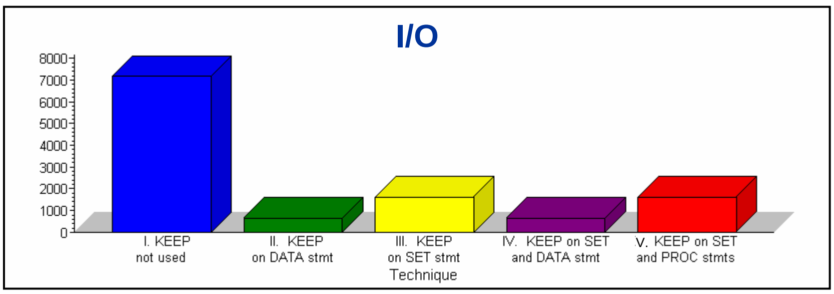
Comparing Techniques

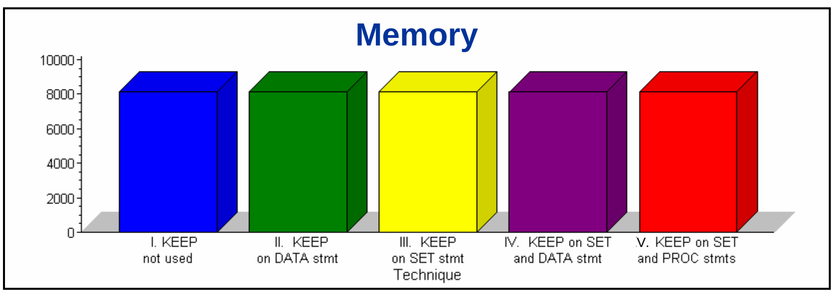
Technique	CPU	I/O	Memory
I. KEEP not used	2.9	7177	8140
II. KEEP on DATA statement	2.3	656	8138
III. KEEP on SET statement	2.4	1625	8138
IV. KEEP on SET and DATA statements	2.2	662	8138
V. KEEP on SET and PROC statements	2.4	1625	8139



75

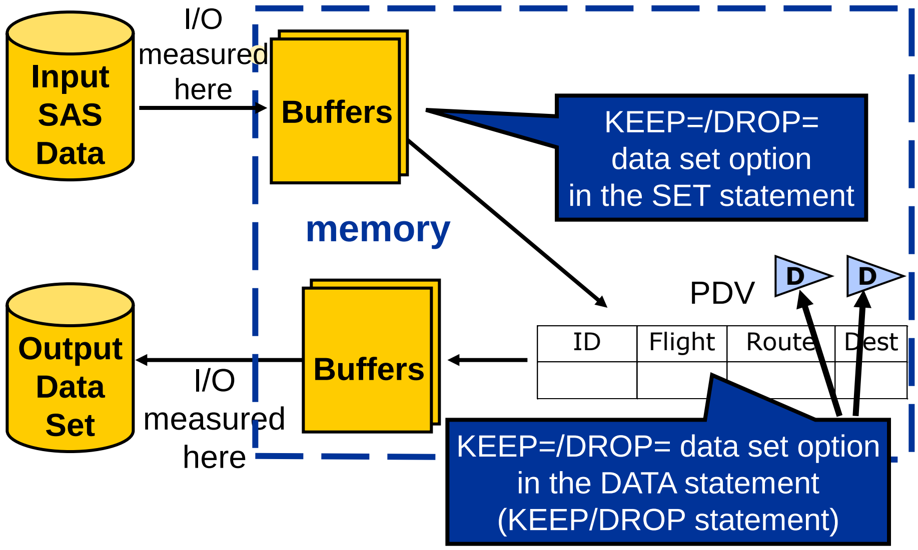
Comparing Techniques





76

Using the KEEP=/DROP= Options



77

Reading All Fields

```
data sales(keep = FlightID Num1st
                  NumBus NumEcon NumPassTotal);
   infile rawdata ;
   input FlightID $7.  RouteID $7.
         Origin $3.  Dest $3.
         DestType $13.  FltDate date9.
         Cap1st 8.  CapBus 8.
         CapEcon 8. CapPassTotal 8.
         CapCargo 8.  Num1st 8.
         NumBus 8. NumEcon 8.
         NumPassTotal 8. Rev1st 8.
         RevBus 8.  RevEcon 8.
         CargoRev 8. RevTotal 8.
         CargoWeight 8.;
run;
```

78

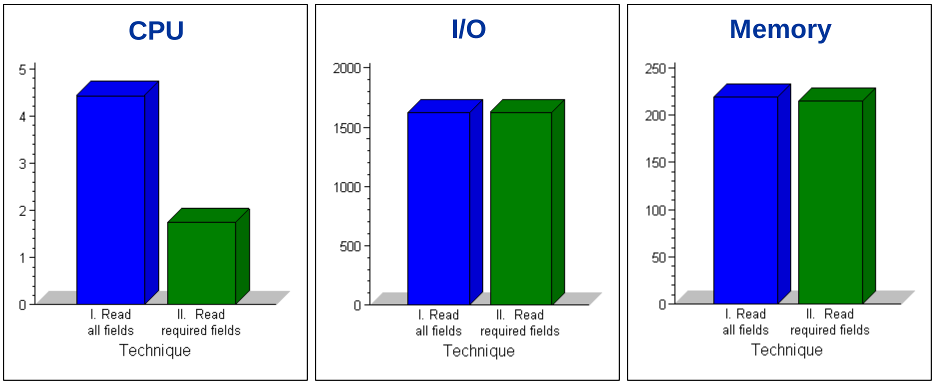
Reading Required Fields

```
data sales;
   infile rawdata ;
   input FlightID $7. @85 Num1st 8.
         NumBus 8. NumEcon 8.
         NumPassTotal 8. ;
run;
```

79

Comparing Techniques

Technique	CPU	I/O	Memory
I. Read all fields	4.4	1627.0	219.0
II. Read required fields	1.7	1625.0	215.0
Percent Difference	60.7	0.1	1.8



80

Conclusions

If the variable is already in a SAS data set, you can use the following to minimize the volume of data processed:

- WHERE statements in DATA and PROC steps
- KEEP and DROP statements in the DATA step
- WHERE=, KEEP=, and DROP= data set options in DATA and PROC steps

If the data is not in a SAS data set or the variable is a calculated variable, you can use the following to minimize the volume of data processed:

- subsetting IF statements
- selective INPUT statements

81

Question & Answer

82

Chapter 1: Best Practices

83

Techniques for Minimizing I/O Operations

- Process only the necessary variables and observations.
- Reduce the number of times that the data is processed.
- Reduce the number of data accesses using the appropriate BUFSIZE= and BUFNO= options for the way that the data is accessed.
- Create a SAS data set, if you process the same non-SAS data repeatedly. SAS can process SAS data sets more efficiently than it can process raw data files or database data.
- Create indexes on variables used for WHERE processing.

84

Controlling Page Size and Memory Usage

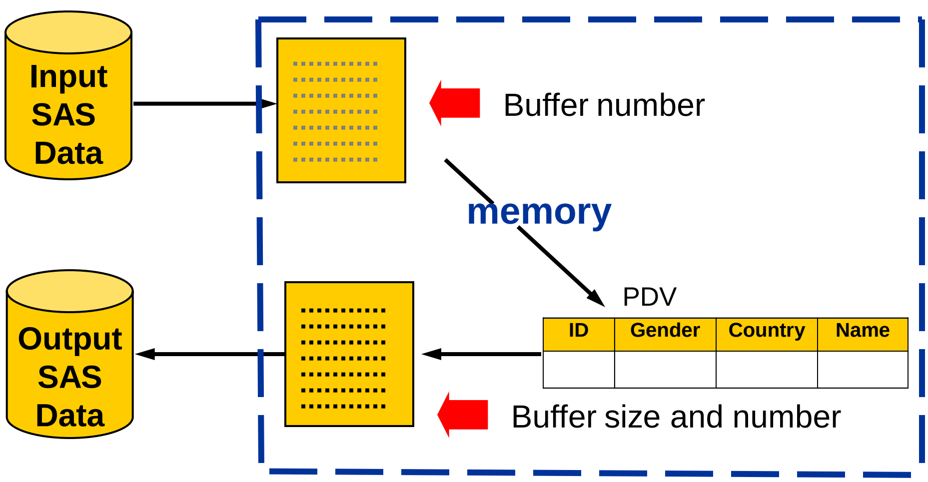
- You can use the BUFSIZE= system option or data set option to control the page size of an output SAS data set.
- You can use the BUFNO= system option or data set option to control the number of SAS buffers open simultaneously in memory.

```
BUFSIZE= n | nK | nM | nG | nT | hexX | MIN | MAX
```

```
BUFNO= n
```

85

Controlling Page Size and Memory Usage



86

Controlling Page Size and Memory Usage

The product of BUFNO= and BUFSIZE= determines how much data can be transferred in a read operation.

BUFSIZE	BUFNO	Bytes transferred in one I/O
16384	2	32,768

Increasing either BUFSIZE= or BUFNO= increases the amount of data that can be transferred in a read operation.

87 ...

Controlling Page Size

In order to select a default page size, SAS software uses an algorithm based on observation length, engine, and operating environment.

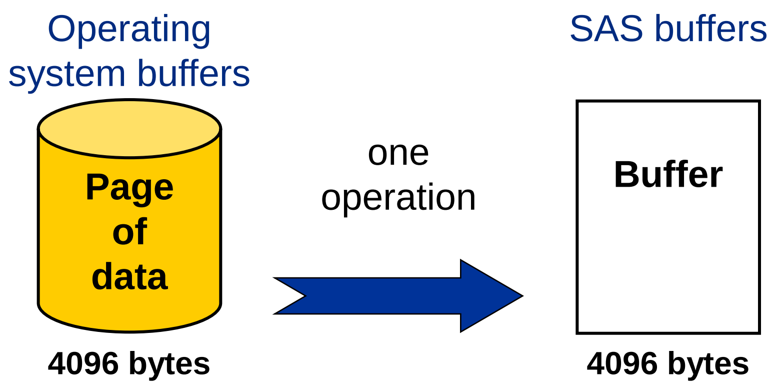
You can use the BUFSIZE= system or data set option to override the default page size.

BUFSIZE= specifies not only the page size (in bytes), but also the size of each buffer used to read or write the SAS data set.

```
data work.times(bufsize = 4096);
   infile rtetimes;
   input @1 RouteID $7.
         @8 Origin $3.
         @11 Dest $3.
         @14 Distance 8.
         @24 Depart time5.
         @32 Arrival time5.;
run;
```

88

Controlling Page Size



89

...

Controlling Page Size

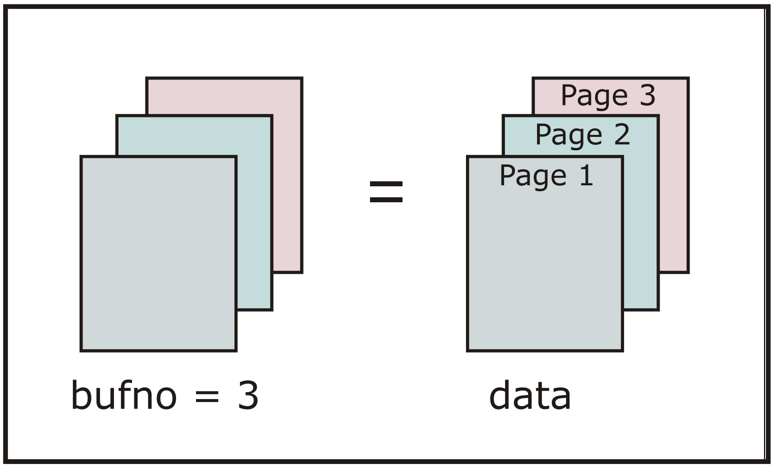
After it is specified, page size is a permanent attribute of the data set, and is used whenever the data set is processed.

Choosing a page size that is larger than the default can reduce execution time by reducing the number of times that SAS must read from or write to the operating system buffers.

The reduction in I/O comes at the cost of increased memory consumption.

90

Controlling Memory Usage



current SAS session

91

Controlling Memory Usage

The buffer number is not a permanent attribute of the data set and is valid only for the current step or SAS session. As more buffers are available, more pages can be transferred in a single move operation.

The reduction in number of moves comes at the cost of increased memory consumption.

```
data _null_;
   set work.times(bufno = 3);
run;
```

92

SASFILE Global Statement

- The SASFILE statement requests that a SAS data set be opened and loaded into SAS memory in its entirety instead of a few pages at a time.
- After it is read, data is held in memory for subsequent DATA and PROC steps to process.
- A second SASFILE statement closes the file and frees the SAS buffers.

93

SASFILE Global Statement

General form of the SASFILE statement:

SASFILE <*libref.*>*member-name*
<(*password-data-set-option*(*s*))>
OPEN | LOAD | CLOSE;

94

Buffer Allocation

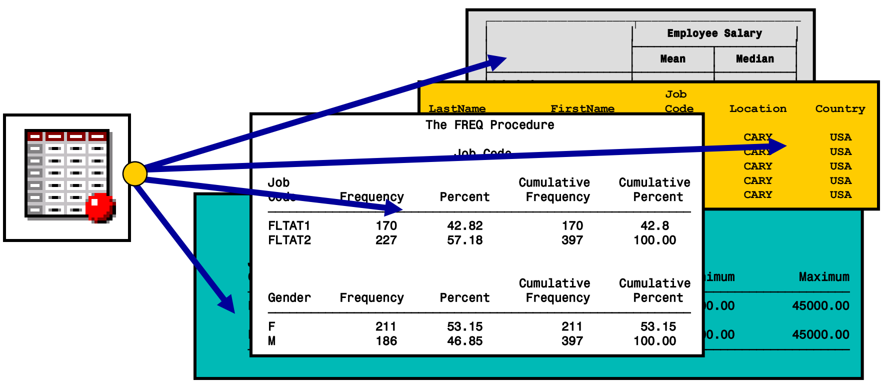
When the SASFILE statement executes, SAS allocates the number of buffers based on the number of pages of the SAS data set and index file.

If the file in memory increases in size during processing by editing or appending data, the number of buffers also increases.

95

Using the SASFILE Statement

Create reports using the PRINT, TABULATE, MEANS, and FREQUENCY procedures against a single SAS data set.



96

Using the SASFILE Statement

```
sasfile work.fltaten load;
proc print data = work.fltaten;
   var LastName FirstName JobCode
       Country Location;
   sum Salary;
run;
proc tabulate data = work.fltaten;
   class Gender;
   var Salary;
   table Gender, Salary*(mean median);
run;
proc means data = work.fltaten;
   var Salary;
   class Gender;
   output out = summary sum =;
run;
proc freq data = work.fltaten;
   tables JobCode Gender;
run;
sasfile work.fltaten close;
```

`work.fltaten` is read into memory only once instead of four times. This results in one-fourth as many I/O operations, increased memory usage, and probably reduced elapsed time.

97

Using the SGIO System Option in Windows

The SGIO system option performs the following functions:

- activates the Scatter-Read/Gather-Write I/O feature
- improves I/O performance for SAS I/O files when the PC has a large amount of RAM

General form of the SGIO system option:

```
NOSGIO | SGIO;
```

⚠ Prior to SAS 9.2 NOSGIO | SGIO is an invocation option only. Starting with SAS 9.2 SGIO also became available as a data set option.

98

Using the SGIO System Option in Windows

When SGIO is active, SAS does the following:

- uses the number of buffers that are specified by the BUFNO= system option to transfer data between disk and RAM
- bypasses Windows file cache when reading or writing data
- reads ahead the number of pages specified by the BUFNO= system option and places the data in memory before it is needed

When the data is needed, it is already in memory and is, in effect, a direct memory access.

Try different values of the BUFNO system option to tune each SAS job or DATA step.

99

Using Direct File I/O in UNIX

The ENABLEDIRECTIO and USEDIRECTIO LIBNAME statement and data set options perform the following functions:

- activates direct I/O access
- bypasses UNIX file caching
- improves I/O performance for SAS I/O files that require a single sequential pass

The LIBNAME statement option enables direct I/O to any file in the library. The data set option allows direct I/O to the data set for this SAS program step only.

⚠ You must use the LIBNAME statement option and the data set option together to enable direct I/O.

100

Using Direct File I/O in UNIX

General form of the USEDIRECTIO LIBNAME statement option:

LIBNAME *libref* '*directory*' USEDIRECTIO=NO|YES
ENABLEDIRECTIO**;**

General form of the USEDIRECTIO SAS data set option:

SAS-data-set-name (USEDIRECTIO=NO|YES)

101

Question & Answer

102

Chapter 1: Best Practices

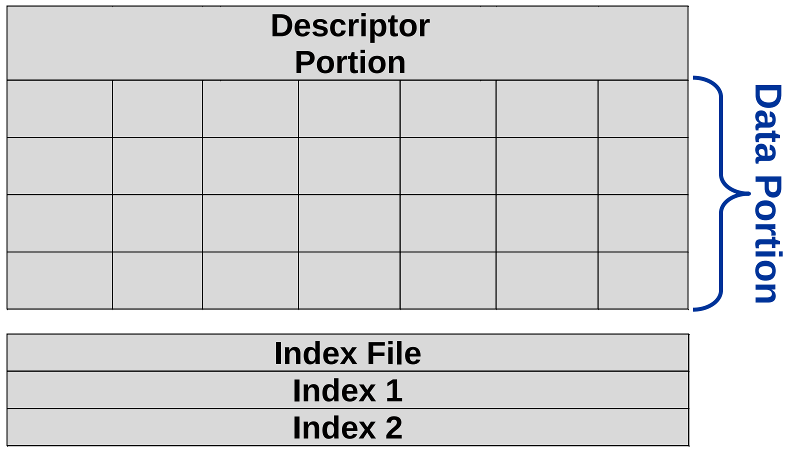
103

Techniques for Conserving Disk Space

- Process only the necessary variables.
- Create reduced length numerics.
- Compress SAS data files.

104

Storage Required for Data Files



105

Review of the Data Set Page

A data set page

- is the unit of data transfer between the SAS storage device and main memory
- includes the bytes used by the descriptor portion, the data values, and any overhead
- is fixed in size when the data set is created.

106

Determining Page Size with PROC CONTENTS

```
proc contents data = work.sales;
run;
```

Partial Output

```
 Engine/Host Dependent Information

Data Set Page Size           16384
Number of Data Set Pages     3396
First Data Page              1
Max Obs per Page             97
Obs in First Data Page       76
Index File Page Size         4096
Number of Index File Pages   2552
Number of Data Set Repairs   0
File Name                    C:\workshop\w
Release Created              9.0101M3
Host Created                 XP_PRO
```

work.sales contains 55,640,064 bytes of data in the data portion and 10,452,992 bytes for the index file. The total number of bytes is 66,093,056.

107

Characteristics of Numeric Variables

Numeric variables

- store multiple digits per byte
- take eight bytes of storage per variable, by default
- can be reduced in size
- always have a length of eight bytes in the PDV
- are stored as floating-point numbers in real-binary representation
- use a minimum of one byte to store the sign and exponent of the value (depending on the operating environment) and use the remaining bytes to store the mantissa of the value.

108

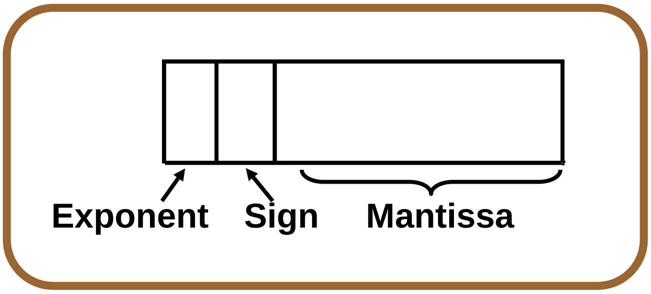
Default Length of Numeric Variables

The number 35298 can also be written as follows:

+0.35298*(105)**

Sign Mantissa Base Exponent

SAS stores numeric variables in floating point form:



109

Assigning the Length of Numeric Variables

- You can use a LENGTH statement to assign a length from two to eight bytes to numeric variables.
- The minimum length of numeric variables depends on the operating environment.

Example:

```
data reducedsales;
   length Cap1st CapBus CapEcon 3
          CapCargo Num1st NumBus
          NumEcon CargoWeight FltDate 4
          Rev1st RevBus
          RevEcon CargoRev 5;

   <more SAS code>
run;
```

110

Assigning the Length of Numeric Variables

Size of **work.sales** (without index)	Size of **reducedsales**	% Difference
55,640,064 bytes	37,134,336 bytes	33%

111

Comparing Data Sets

```
proc compare data = work.sales
             compare = work.reducedsales;
run;
```

Partial Output

```
                 Observation Summary

           Observation      Base  Compare

           First Obs           1        1
           Last  Obs      329264   329264

Number of Observations in Common: 329264.
Total Number of Observations Read from work.sales: 329264.
Total Number of Observations Read from work.reducedsales: 329264.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 329264.

NOTE: No unequal values were found. All values compared are exactly equal.
```

112

Possible Storage Lengths for Integer Values

Windows and UNIX

Length (bytes)	Largest Integer Represented Exactly
3	8,192
4	2,097,152
5	536,870,912
6	137,438,953,472
7	35,184,372,088,832
8	9,007,199,254,740,992

113

Possible Storage Lengths for Integer Values

z/OS

Length (bytes)	Largest Integer Represented Exactly
2	256
3	65,536
4	16,777,216
5	4,294,967,296
6	1,099,511,627,776
7	281,474,946,710,656
8	72,057,594,037,927,936

114

Assigning the Length of Numeric Variables

The use of a numeric length less than 8 bytes does the following:

- reduces the number of bytes available for the mantissa, and thus reduces the precision of the largest number that can be accurately stored
- does not affect how numbers are stored in the PDV; numbers are always eight bytes in length in the PDV
- causes the number to be truncated to the specified length when the value is written to the SAS data set
- causes the number to be expanded to eight bytes in the PDV when the data set is read by padding the mantissa with binary zeros.

115

Reading Reduced-Length Numeric Variables

Reading reduced-length numeric variables

- requires less I/O
- uses additional CPU
- can be dangerous for high precision values, including non-integer and large integer values.

116

Dangers of Reduced-Length Numeric Variables

It is **not** recommended that you change the length of non-integer numeric variables.

```
data test;
   length x 4;
   x = 1/10;
   y = 1/10;
run;

data _null_;
   set test;
   put x=;
   put y=;
run;
```

117

Dangers of Reduced-Length Numeric Variables

Partial Log

```
81   data test;
82      length x 4;
83      x = 1/10;
84      y = 1/10;
85   run;
NOTE: The data set WORK.TEST has 1 observations and 2 variables.

86
87   data _null_;
88      set test;
89      put x=;
90      put y=;
91   run;

x=0.0999999642
y=0.1
NOTE: There were 1 observations read from the data set WORK.TEST.
```

118

Dangers of Reduced-Length Numeric Variables

It is **not** recommended that you change the length of integer numeric variables inappropriately or that you change the length of large integer numeric variables.

```
data test;
   length x 3;
   x = 8193;
run;

data _null_;
   set test;
   put x=;
run;
```

119

Dangers of Reduced-Length Numeric Variables

Partial Log

```

data _null_;
   set test;
   put x=;
run;

x=8192
NOTE: There were 1 observations read from the
data set WORK.TEST.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

120

Simplified Uncompressed Data File Structure

Page 1	24 / 40 byte OH	Obs 1	Obs 2	Obs 3	Obs 4	Obs 5	*	1 bit / obs OH	Descriptor	
Page 2	24 / 40 byte OH	Obs 6	Obs 7	Obs 8	Obs 9	Obs 10	Obs 11	Obs 12	*	1 bit / obs OH
	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
Page n	24 / 40 byte OH	Obs x	Obs y	Obs z	* Unused space					1 bit / obs OH

121

Uncompressed SAS Data File

The features of uncompressed SAS data files include the following:

- All observations use the same number of bytes.
- Each variable occupies the same number of bytes in every observation.
- Character values are padded with blanks.
- Numeric values are padded with binary zeros.
- The descriptor portion of the data set uses part of the first data set page.

122

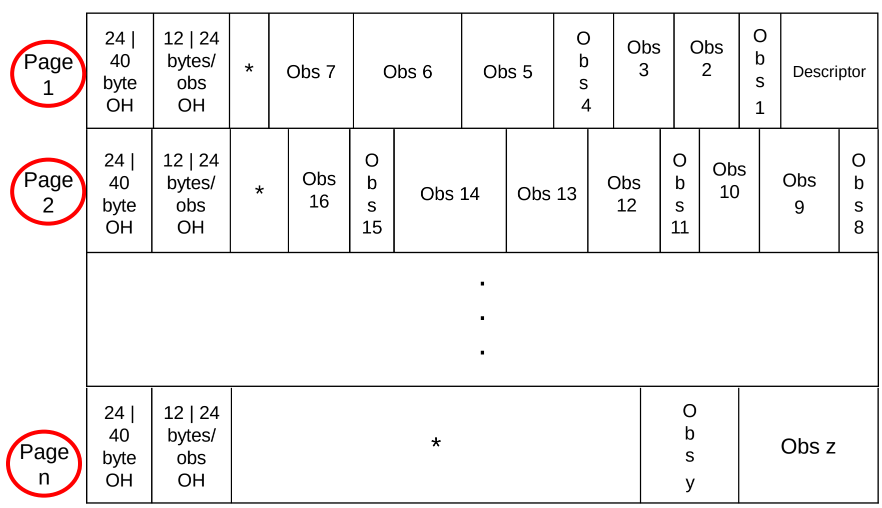
continued...

Uncompressed SAS Data File

- There is a 24-byte overhead at the beginning of each page on 32-bit systems.
- There is a 40-byte overhead at the beginning of each page on 64-bit systems.
- There is a 1-bit per observation overhead, rounded up to the nearest byte.
- New observations are added at the end of the file. If a new page is needed for a new observation, a whole data set page is added.
- Deleted observation space is never reused, unless the entire data file is rebuilt.

123

Simplified Structure of a Compressed Data Set



* Unused space

124

Compressed SAS Data File

Features of compressed SAS data files:

- Each observation is a single string of bytes. Variable types and boundaries are ignored.
- Each observation can have a different length.
- Consecutive repeating characters and numbers are collapsed into fewer bytes.
- If an updated observation is larger than its original size, it is stored on either the same data set page or on a different page with a pointer to the original page.
- The descriptor portion of the data set is stored at the end of the first data set page.

125

continued...

Compressed SAS Data File

- There is a 24-byte overhead at the beginning of each page on 32-bit systems.
- There is a 40-byte overhead at the beginning of each page on 64-bit systems.
- There is a 12-byte-per-observation overhead on 32-bit systems.
- There is a 24-byte-per-observation overhead on 64-bit systems.
- Deleted observation space can be reused if the REUSE=YES data set or system option was turned on when the SAS data file was compressed.

126

Compressing SAS Files

There are two different algorithms that can be used to compress files:

- the RLE (Run Length Encoding) compression algorithm (compress = YES | CHAR)
- the RDC (Ross Data Compression) algorithm (COMPRESS = BINARY)

⚠ The optimal algorithm depends on the characteristics of your data.

127

Creating an Uncompressed Data File

```
data sales;
   infile 'Sales.dat';
   input @1 FlightID $7.      @8 RouteID $7.
         @15 Origin $3.       @18 Dest $3.
         @21 DestType $13.    @34 FltDate date9.
         @43 Cap1st 3.        @46 CapBus 3.
         @49 CapEcon 3.       @52 CapPassTotal 3.
         @55 CapCargo 6.      @61 Num1st 3.
         @64 NumBus 3.        @67 NumEcon 3.
         @70 NumPassTotal 3.  @73 Rev1st 7.
         @80 RevBus 7.        @87 RevEcon 7.
         @94 CargoRev 8.      @102 RevTotal 10.
         @112 CargoWeight 5.;
run;
```

128

Creating a Compressed Data File

```
data saleschar(compress = char);
   infile 'Sales.dat';
   input @1 FlightID $7.      @8 RouteID $7.
         @15 Origin $3.       @18 Dest $3.
         @21 DestType $13.    @34 FltDate date9.
         @43 Cap1st 3.        @46 CapBus 3.
         @49 CapEcon 3.       @52 CapPassTotal 3.
         @55 CapCargo 6.      @61 Num1st 3.
         @64 NumBus 3.        @67 NumEcon 3.
         @70 NumPassTotal 3.  @73 Rev1st 7.
         @80 RevBus 7.        @87 RevEcon 7.
         @94 CargoRev 8.      @102 RevTotal 10.
         @112 CargoWeight 5.;
run;
```

129

Partial Log

```
NOTE: The data set WORK.SALESCHAR has 329264 observations and 21
variables.
NOTE: Compressing data set WORK.SALESCHAR decreased size by 28.14
percent.
      Compressed is 4930 pages; un-compressed would require 6861 pages.
NOTE: DATA statement used (Total process time):
      real time           17.36 seconds
      cpu time            3.25 seconds
```

130

Creating a Compressed Data File

```
data salesbin(compress = binary);
   infile 'Sales.dat';
   input @1 FlightID $7.      @8 RouteID $7.
         @15 Origin $3.       @18 Dest $3.
         @21 DestType $13.    @34 FltDate date9.
         @43 Cap1st 3.        @46 CapBus 3.
         @49 CapEcon 3.       @52 CapPassTotal 3.
         @55 CapCargo 6.      @61 Num1st 3.
         @64 NumBus 3.        @67 NumEcon 3.
         @70 NumPassTotal 3.  @73 Rev1st 7.
         @80 RevBus 7.        @87 RevEcon 7.
         @94 CargoRev 8.      @102 RevTotal 10.
         @112 CargoWeight 5.;
run;
```

131

Partial Log

```
NOTE: The data set WORK.SALESBIN has 329264 observations and 21
variables.
NOTE: Compressing data set WORK.SALESBIN decreased size by 31.51
percent.
      Compressed is 4699 pages; un-compressed would require 6861 pages.
NOTE: DATA statement used (Total process time):
      real time           7.04 seconds
      cpu time            3.62 seconds
```

132

Summary of Compression Results

Data Set	Algorithm Used	Number of Bytes	Decreased size
sales	none	55,623,680	--
saleschar	CHAR	40,386,560	28.14%
salesbin	BINARY	38,494,208	31.51%

133

Creating a Compressed Data File

To create a compressed data file, use the COMPRESS= output data set option or system option.

General forms of the COMPRESS= options:

SAS-data-set(COMPRESS = NO | YES | CHAR | BINARY)

OPTIONS COMPRESS = NO | YES | CHAR | BINARY;

134

Comparing Compression Methods

COMPRESS = YES | CHAR

- is effective with character data that contains repeated characters (such as blanks)

COMPRESS = BINARY

- takes significantly more CPU time to uncompress than COMPRESS=YES | CHAR
- is more efficient with observations greater than a thousand bytes in length
- can be very effective with numeric data
- can be effective with character data that contains patterns, rather than simple repetitions

135

How SAS Compresses Data

A data file has these variables:

Name	Type	Length
LastName	Character	20
FirstName	Character	15

In uncompressed form, all observations use 35 bytes for these two variables.

LastName FirstName

01 ... 20 ...

A	D	A	M	S								B	I	L	L						

136

COMPRESS = BINARY

Ross Data Compression uses both run-length encoding and sliding window compression.

A data set has these variables:

Name	Type	Length
`Answer1`	Numeric	8
. . .		
`Answer200`	Numeric	8

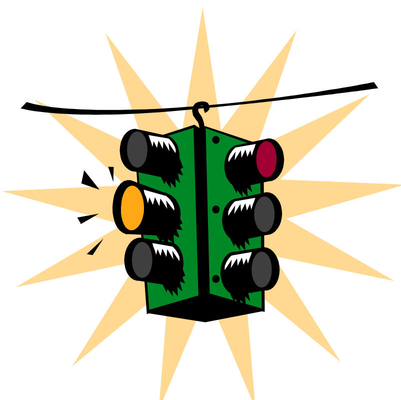
In uncompressed form, the data file resembles this:

Obs	answer1	answer2	answer3	answer4	answer5		answer200
1	1	2	1	2	1	. . .	2
2	1	1	1	1	1	. . .	1
3	2	2	2	2	2	. . .	2

137

...

Compression Guidelines



Some data sets do not compress well or at all.

138

Compression Dependencies

Because there is higher overhead for each observation, a data file can occupy more space in compressed form than in uncompressed form if the file has the following:

- few repeated characters
- small physical size
- few missing values
- short text strings

139

Compression Guidelines

```
data work.capacity_ch(compress = yes);
   set work.capacity;
run;
```

Partial Log

```
1175  data capacity_ch(compress = yes);
1176     set work.capacity;
1177  run;

NOTE: There were 108 observations read from the data set WORK.CAPACITY.
NOTE: The data set WORK.CAPACITY_CH has 108 observations and 7 variables.
NOTE: Compressing data set WORK.CAPACITY increased size by 50.00 percent.
      Compressed is 3 pages; un-compressed would require 2 pages.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds
```

140

Compression Dependencies

When you use the COMPRESS= data set option or the COMPRESS= system option, SAS knows the following:

- size of the overhead introduced by compression
- maximum size of an observation.

⚠ If the maximum size of the observation is less than the overhead introduced by compression, SAS disables compression, creates an uncompressed data set, and issues a note stating that the file was not compressed.

141

Compression Dependencies

```
1     data test(compress = yes);
2        x = 1;
3     run;

NOTE: Compression was disabled for data set
      WORK.TEST because compression overhead would
      increase the size of the data set.
NOTE: The data set WORK.TEST has 1 observations and
      1 variables.
NOTE: DATA statement used:
      real time             0.51 seconds
      cpu time              0.10 seconds
```

142

Compression Trade-Offs

Uncompressed	Compressed
Usually requires more disk storage.	Usually requires less disk storage.
Requires less CPU time to prepare observation for I/O.	Requires more CPU time to prepare observation for I/O.
Uses more I/O operations.	Uses fewer I/O operations.

The savings in I/O operations greatly outweighs the increase in CPU time.

continued...

143

Compression Trade-Offs

Uncompressed	Compressed
An updated observation fits in its original location.	An updated observation might be moved from its original location.
Deleted observation space is never reused.	Deleted observation space can be reused.
New observations are always inserted at the end of the data file.	When REUSE=YES, new observations might not be inserted at the end of the data file.

144

Question & Answer

145

Chapter 1: Best Practices

146

Using Indexes

An *index* is an optional file that you can create for a SAS data file that does the following:

- points to observations based on the values of one or more key index variables
- provides direct access to specific observations

An index locates an observation by value.

147

Simplified Index File

The index file consists of entries that are organized in a tree structure and connected by pointers.

Partial Listing of **work.sales**

Customer_ID	Employee_ID	. . .
14958	121031	. . .
14844	121042	. . .
14864	99999999	. . .
14909	120436	. . .
14862	120481	. . .
14853	120454	. . .
14838	121039	. . .
14842	121051	. . .
14815	99999999	. . .
14797	120604	. . .
. . .	. . .	. . .

Simplified Index

Customer_ID Key Value	Record Identifier (RID) Page(obs, obs, ...)
4006	17(85)
4021	17(89)
4059	17(90)
4063	17(80, 86)
. . .	
14958	1(1, 24)
14972	1(14)
. . .	

14

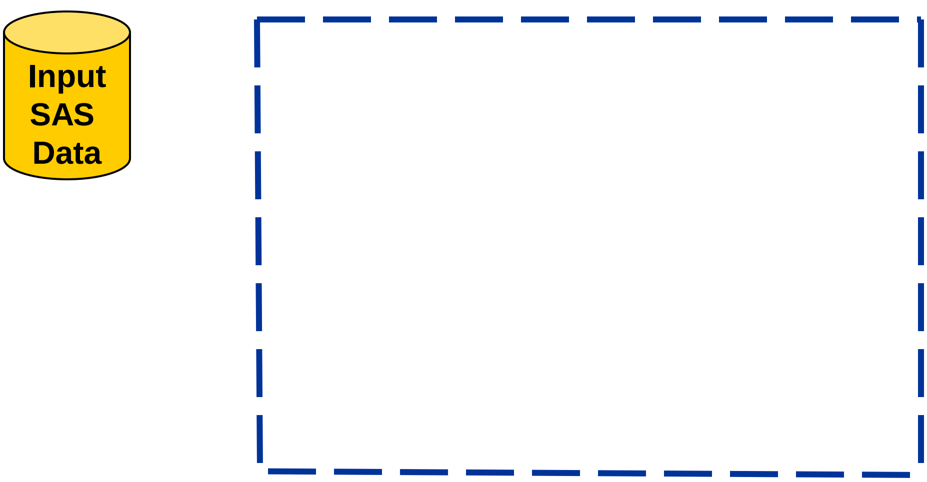
The Purpose of Indexes

Indexes can provide direct access to observations in SAS data sets to accomplish the following:

- yield faster access to small subsets (WHERE)
- return observations in sorted order (BY)
- perform table lookup operations (SET with KEY=)
- join observations (PROC SQL)
- modify observations (MODIFY with KEY=)

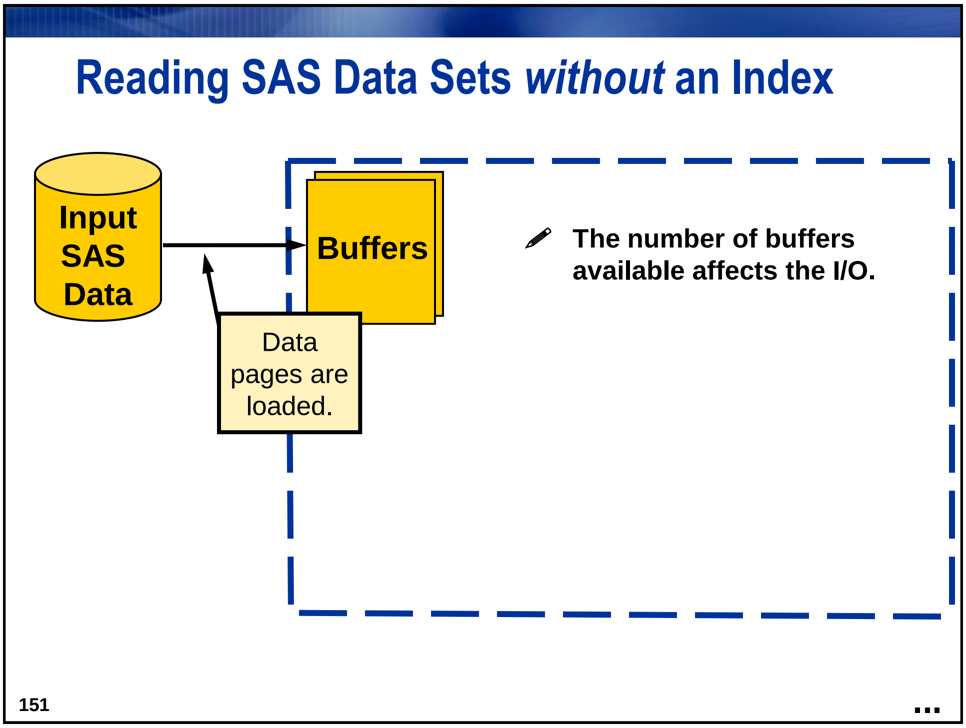
149

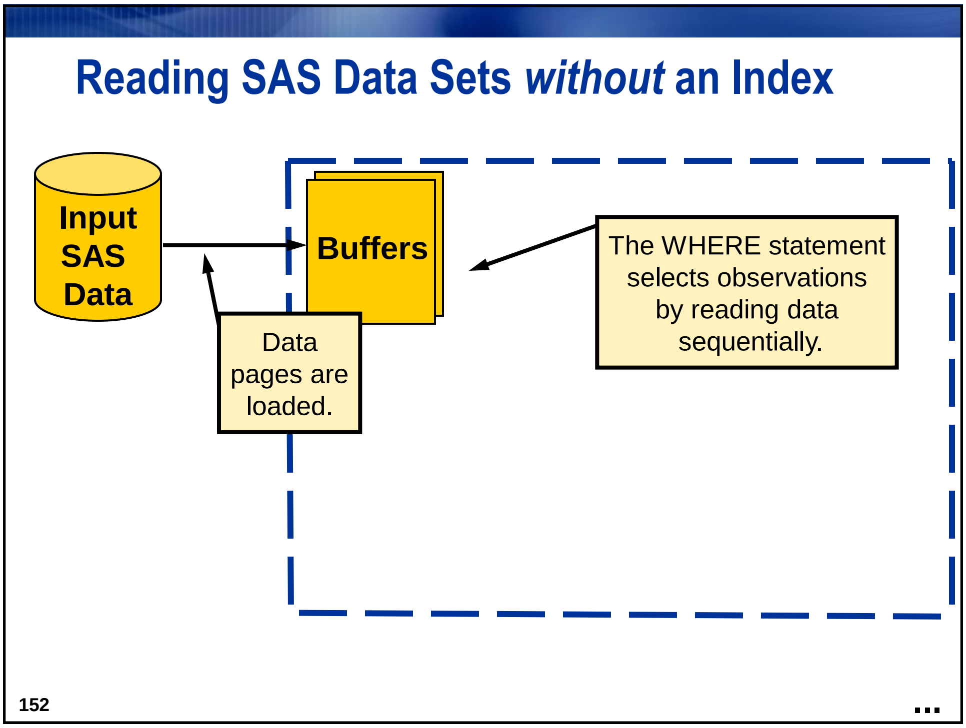
Reading SAS Data Sets *without* an Index



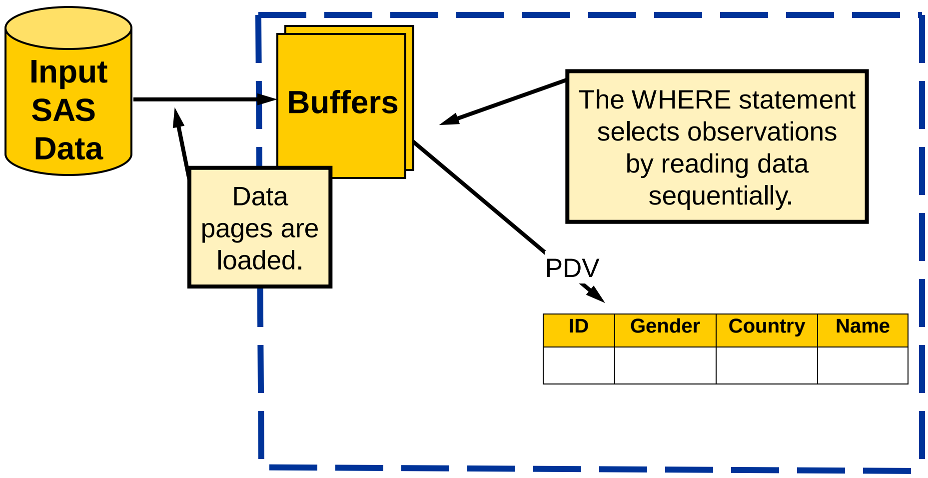
150

...


Reading SAS Data Sets *without* an Index
Input SAS Data
Buffers
Data pages are loaded.
The number of buffers available affects the I/O.
151
...

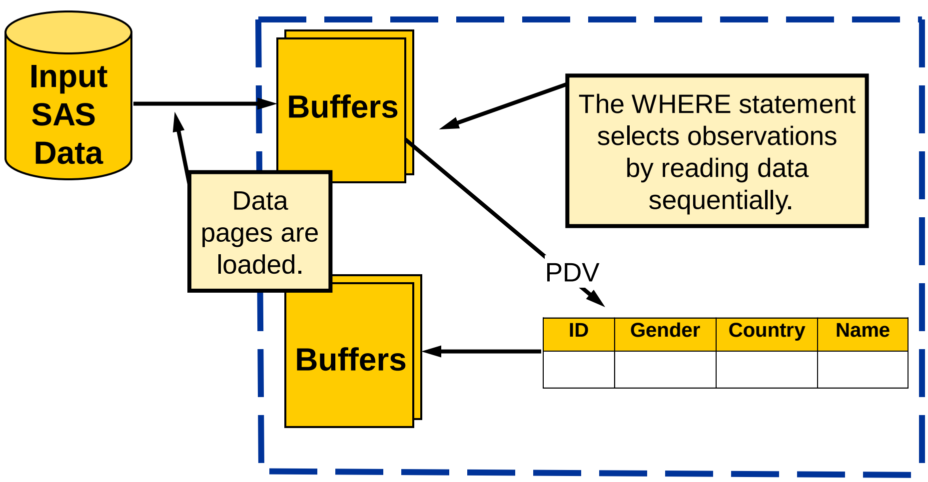

Reading SAS Data Sets *without* an Index
Input SAS Data
Buffers
Data pages are loaded.
The WHERE statement selects observations by reading data sequentially.
152
...

Reading SAS Data Sets *without* an Index

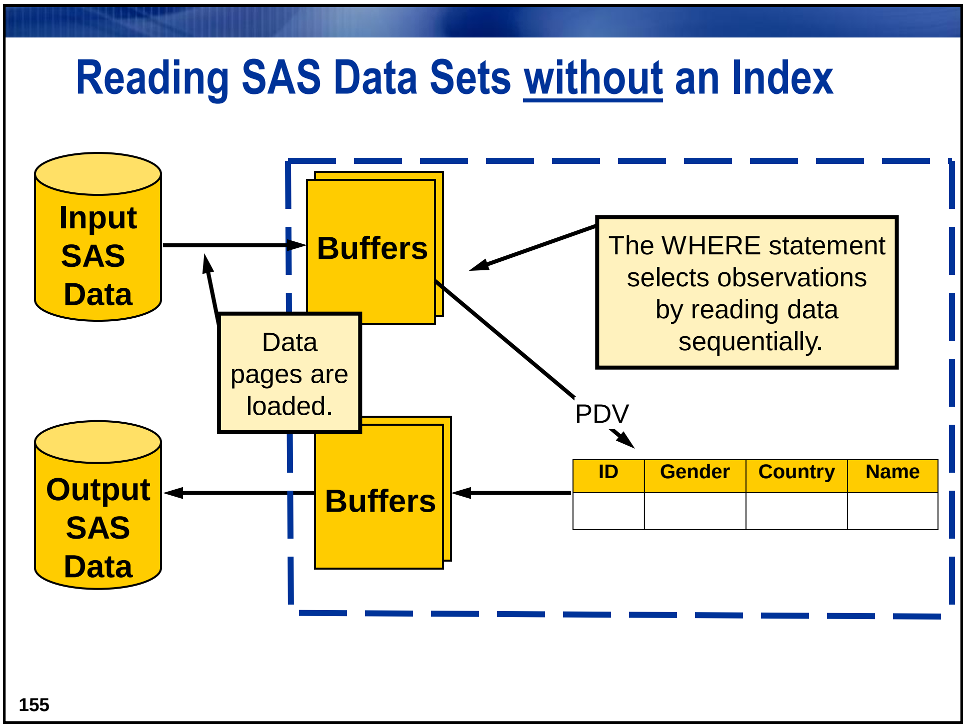


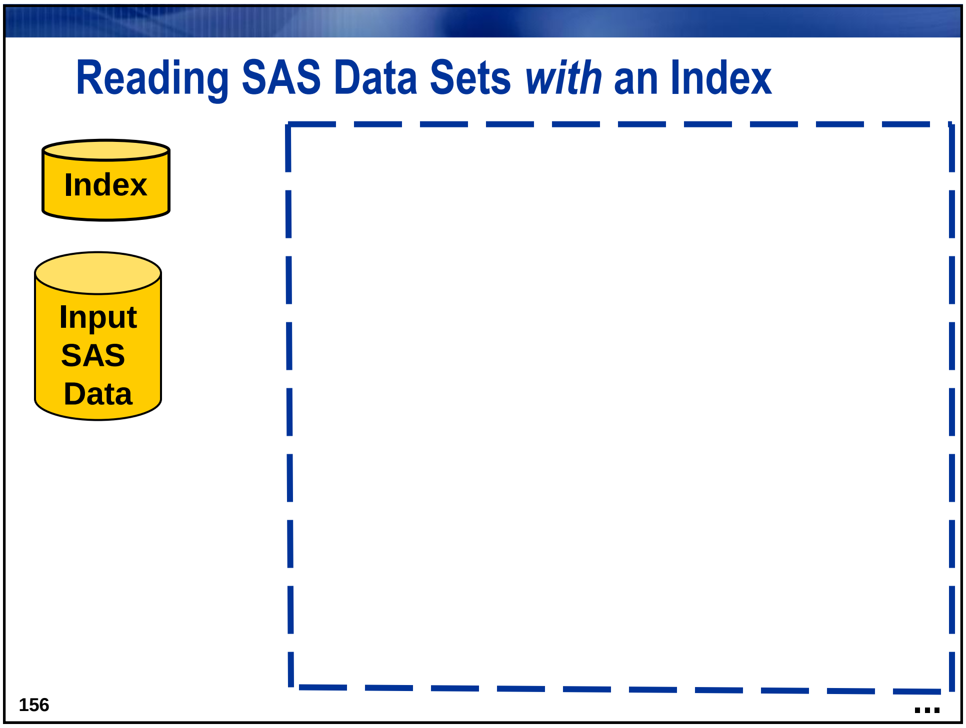
153 ...

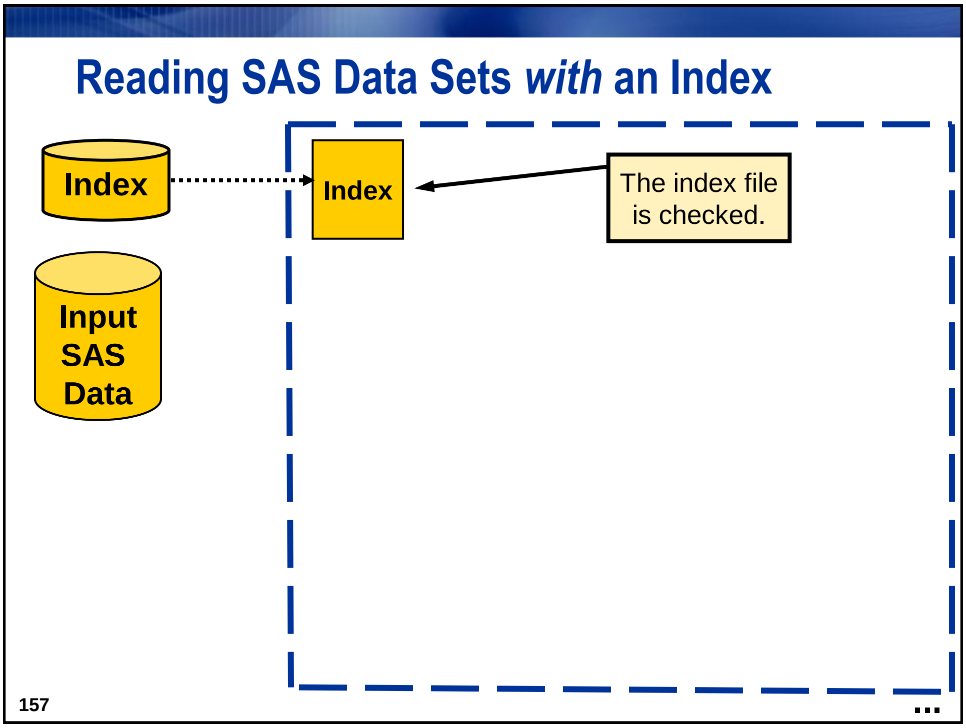
Reading SAS Data Sets <u>without</u> an Index

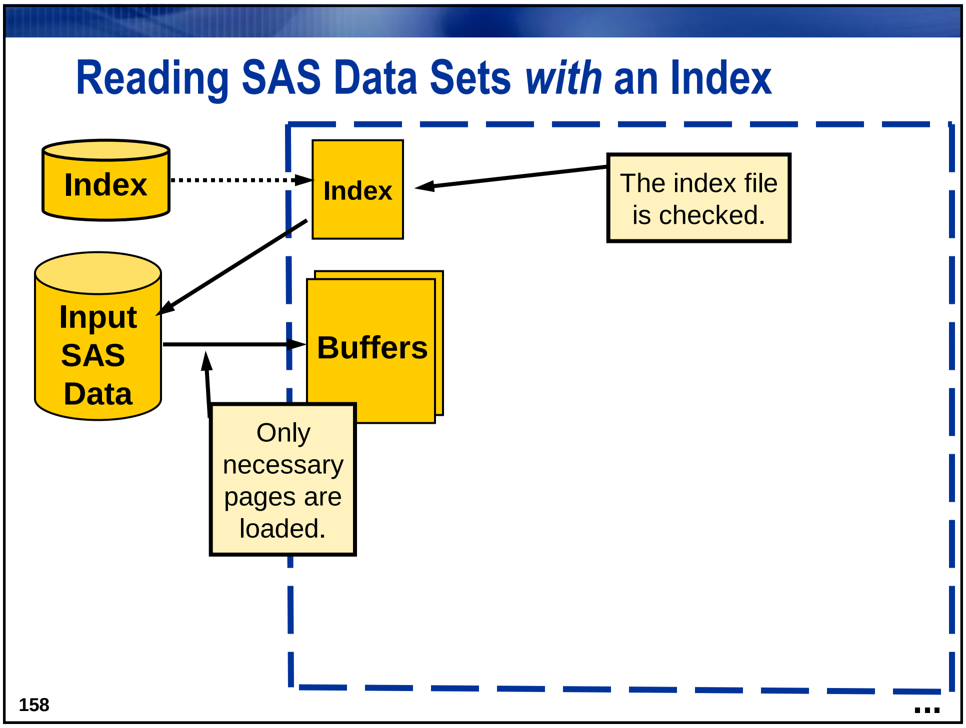


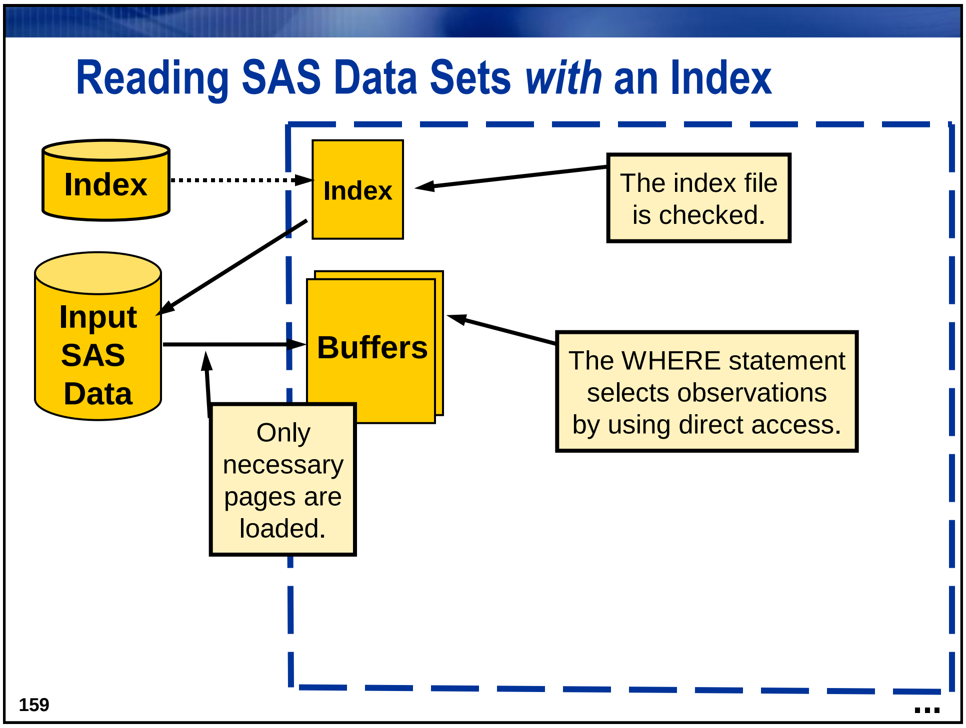
154 ...

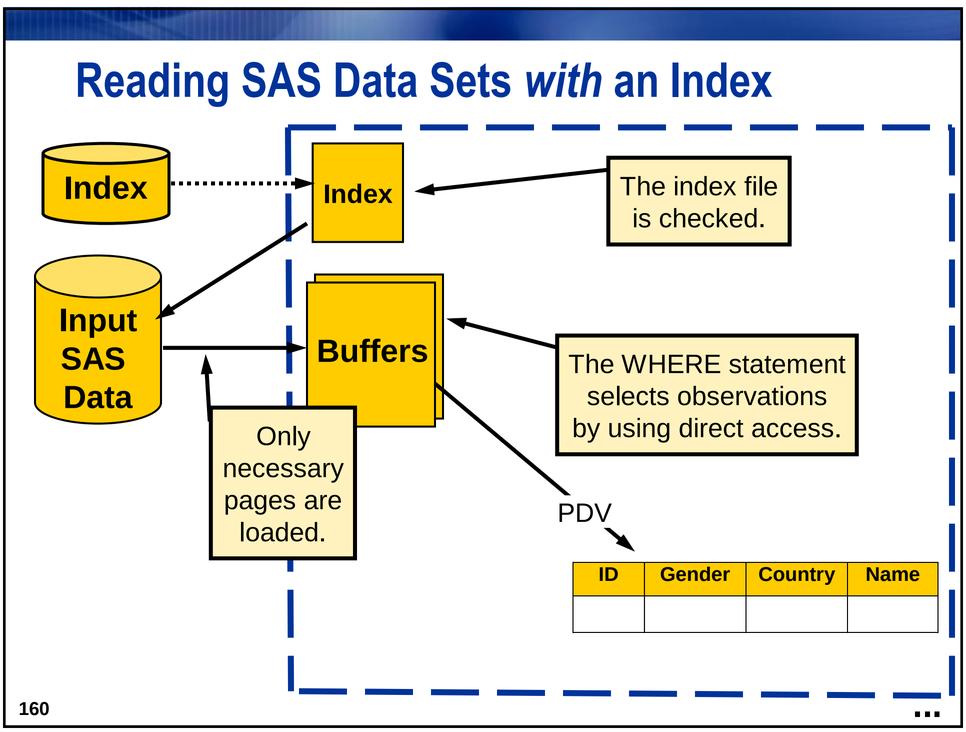

Reading SAS Data Sets without an Index
Input SAS Data
Buffers
The WHERE statement selects observations by reading data sequentially.
Data pages are loaded.
PDV
ID
Gender
Country
Name
Output SAS Data
Buffers
155

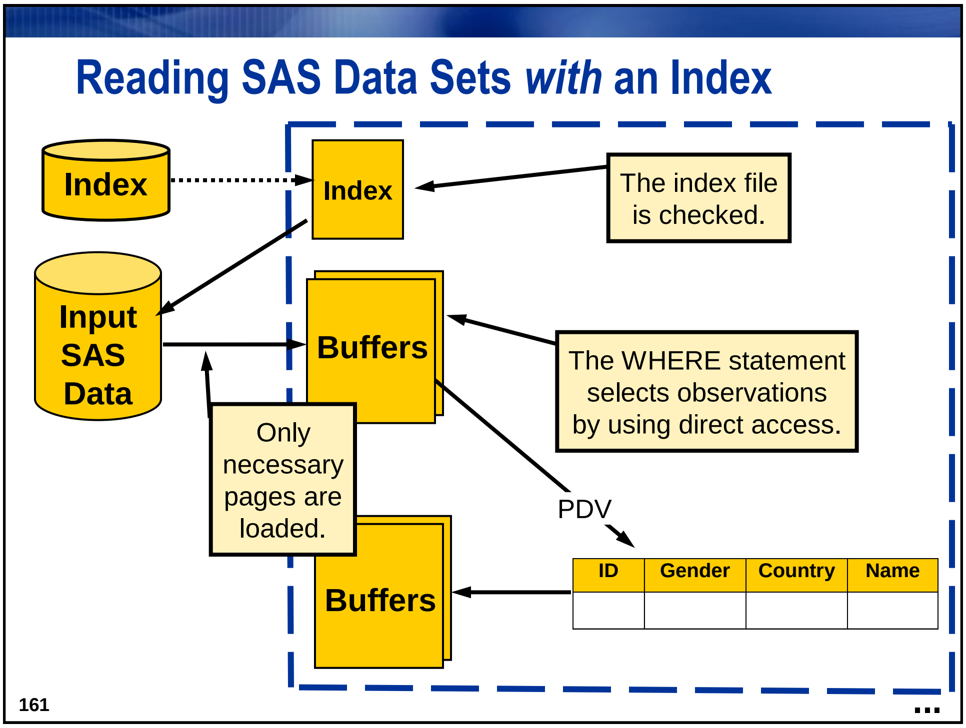

Reading SAS Data Sets with an Index
Index
Input SAS Data
156
...

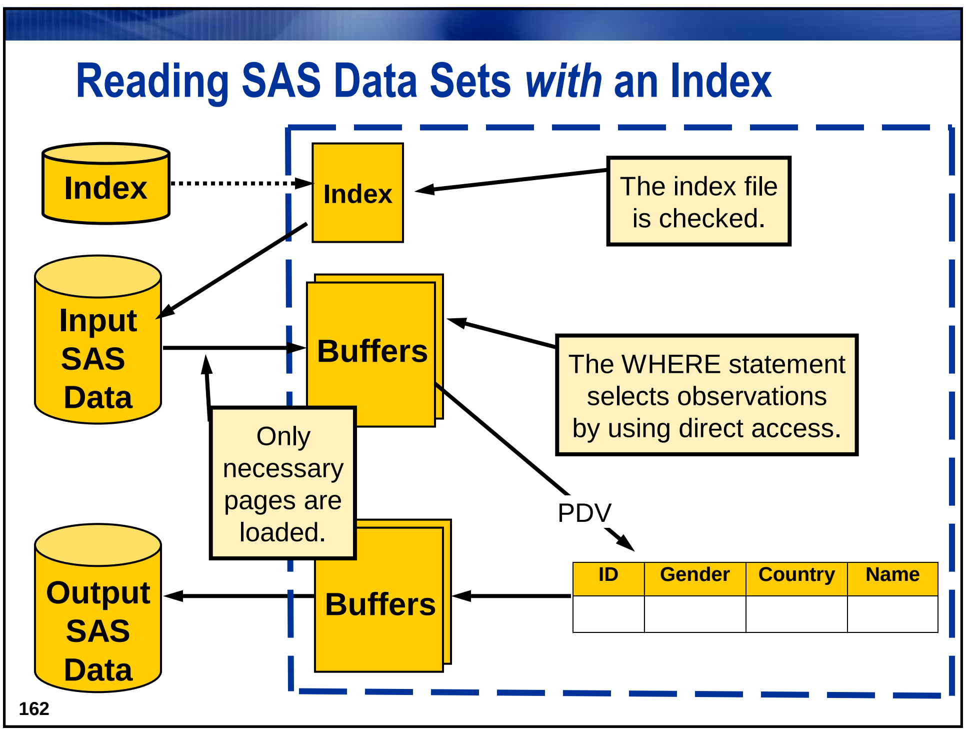

Reading SAS Data Sets *with* an Index
Index
Index
The index file is checked.
Input SAS Data
157
...


Reading SAS Data Sets *with* an Index
Index
Index
The index file is checked.
Input SAS Data
Buffers
Only necessary pages are loaded.
158
...


Reading SAS Data Sets *with* an Index
Index
Index
The index file is checked.
Input SAS Data
Buffers
The WHERE statement selects observations by using direct access.
Only necessary pages are loaded.
159
...


Reading SAS Data Sets *with* an Index
Index
Index
The index file is checked.
Input SAS Data
Buffers
The WHERE statement selects observations by using direct access.
Only necessary pages are loaded.
PDV
ID
Gender
Country
Name
160
...


Reading SAS Data Sets with an Index
Index
Index
The index file is checked.
Input SAS Data
Buffers
The WHERE statement selects observations by using direct access.
Only necessary pages are loaded.
PDV
ID
Gender
Country
Name
Buffers
161
...


Reading SAS Data Sets with an Index
Index
Index
The index file is checked.
Input SAS Data
Buffers
The WHERE statement selects observations by using direct access.
Only necessary pages are loaded.
PDV
ID
Gender
Country
Name
Output SAS Data
Buffers
162

Creating Indexes

To create indexes at the same time that you create a data set, use the INDEX= data set option on the output data set.

To create or delete indexes on existing data sets, use one of the following:

- DATASETS procedure
- SQL procedure

163

Creating Indexes

When you create the index, do the following:

- designate the key variable(s)
- specify the UNIQUE and/or the NOMISS option(s) index option if appropriate (SQL CREATE INDEX statement does not support the NOMISS option)
- select a valid SAS name for the index (composite index only)

A data set can have these index features:

- multiple simple and composite indexes
- character and numeric key variables

164

Viewing Information about Indexes

To display information in the log concerning index creation or index usage, change the value of the MSGLEVEL= system option from its default value of N to I.

General form of the MSGLEVEL= system option:

OPTIONS MSGLEVEL=N | I**;**

```
options msglevel=i;
data work.sales_history(index=
      (Customer_ID Product_Group
       SaleID=(Order_ID
               Product_ID)/unique));
set work.sales_history;
run;

NOTE: There were 1500 observations read from the data set WORK.SALES_HISTORY.
NOTE: The data set WORK.SALES_HISTORY has 1500 observations and 22 variables.
NOTE: Composite index SaleID has been defined.
NOTE: Simple index Product_Group has been defined.
NOTE: Simple index Customer_ID has been defined.
```

165

Creating an Index with the INDEX= Data Set Option

General form of the INDEX= data set option:

SAS-data-file-name **(INDEX =**
(*index-specification-1</option> </option>*
*…<index-specification-n</option> </option> >***));**

For increased efficiency, use the INDEX= option to create indexes when you initially create a SAS data set.

166

Creating an Index with the INDEX= Data Set Option

```
options msglevel=i;
data work.sales(index=Customer_ID Product_Group
                SaleID=(Order_ID Product_ID)/unique));
   set work.history;
   Value_Cost=CostPrice_Per_Unit*Quantity;
   Year_Month=mdy(Month_Num, 15, input(Year_ID,4.));
   format Value_Cost dollar12.
          Year_Month monyy7.;
   label Value_Cost="Value Cost"
         Year_Month="Month/Year";
run;
```

The following code would delete the indexes:

```
data work.sales;
   set work.history;
run;
```

167

Creating / Deleting Indexes with PROC DATASETS

You can use the DATASETS procedure on existing data sets to create or delete indexes.

General form of the PROC DATASETS step to delete or create indexes:

> **PROC DATASETS** LIBRARY=*libref* NOLIST**;**
> **MODIFY** *SAS-data-set-name***;**
> **INDEX DELETE** *index-name***;**
> **INDEX CREATE** *index-specification* *< / options>***;**
> **QUIT;**

168

Managing Indexes with PROC DATASETS

```
options msglevel=n;
proc datasets library=work nolist;
   modify sales;
      index create Customer_ID;
      index create Product_Group;
      index create SaleID=(Order_ID
                   Product_ID)/unique;
quit;
```

The following code would delete the indexes:

```
proc datasets library=work nolist;
   modify sales;
      index delete Customer_ID
            Product_Group SaleID;
quit;
```

169

Creating / Deleting Indexes with PROC SQL

You can use PROC SQL on existing data sets to create or delete indexes.

General form of the PROC SQL step to create or delete indexes:

> **PROC SQL;**
> **DROP INDEX** *index-name*
> **FROM** *table-name***;**
> **CREATE** *<option>* **INDEX** *index-name*
> **ON** *table-name*(*column-name-1,...*
> *column-name-n*)**;**
> **QUIT;**

170

Managing Indexes with PROC SQL

```
options msglevel=i;
proc sql;
   create index Customer_ID
      on work.sales(Customer_ID);
   create index Product_Group
      on work.sales(Product_Group);
   create unique index SaleID
      on work.sales(Order_ID, Product_ID);
quit;
```

Name of Index

Variable Name

The following code would delete the indexes:

```
proc sql;
   drop index Customer_ID, Product_Group, SaleID
      from work.sales;
quit;
```

171

Comparing Techniques for Index Creation

INDEX= Data Set Option	PROC DATASETS	PROC SQL
You can create the SAS data set at the same time that the index is created.	You can only create indexes on existing SAS data sets and existing variables.	You can only create indexes on existing SAS data sets and existing variables.
To create an additional index, you must re-create the existing indexes.	Additional indexes can be created without re-creating the original indexes.	Additional indexes can be created without re-creating the original indexes.
The DATA step can perform data manipulation at the same time that the index is created.	PROC DATASETS cannot perform data manipulation.	The CREATE INDEX statement cannot perform data manipulation.
To delete one or more indexes, you must re-create the other required indexes.	One or more indexes can be deleted without deleting all of the indexes on the data set.	One or more indexes can be deleted without deleting all of the indexes on the data set.
An existing index can be re-created without first deleting it.	If an index exists, it must be deleted before it can be re-created.	If an index exists, it must be deleted before it can be re-created.

172

Index Usage Possible

A WHERE condition might possibly use an index, provided the condition contains any one of the following:

- a comparison operator or the IN operator
- the NOT operator
- the special WHERE operators (CONTAINS, LIKE, IS NULL|IS MISSING, and BETWEEN…AND)
- the TRIM or SUBSTR functions (***if*** the second argument of the SUBSTR function is 1)

173

When Is an Index Not Used?

An index is ***not*** used in the following circumstances:

- with a subsetting IF statement in a DATA step
- No single index can supply all required observations.
- Any function other than TRIM or SUBSTR appears in the WHERE expression.
- The SUBSTR function does not search a string beginning at the first position.
- The SOUNDS-LIKE operator (=*) is used.
- if SAS determines that all observations will satisfy the WHERE expression
- if SAS determines that it is more efficient to read the data sequentially

174

Using a Subsetting IF

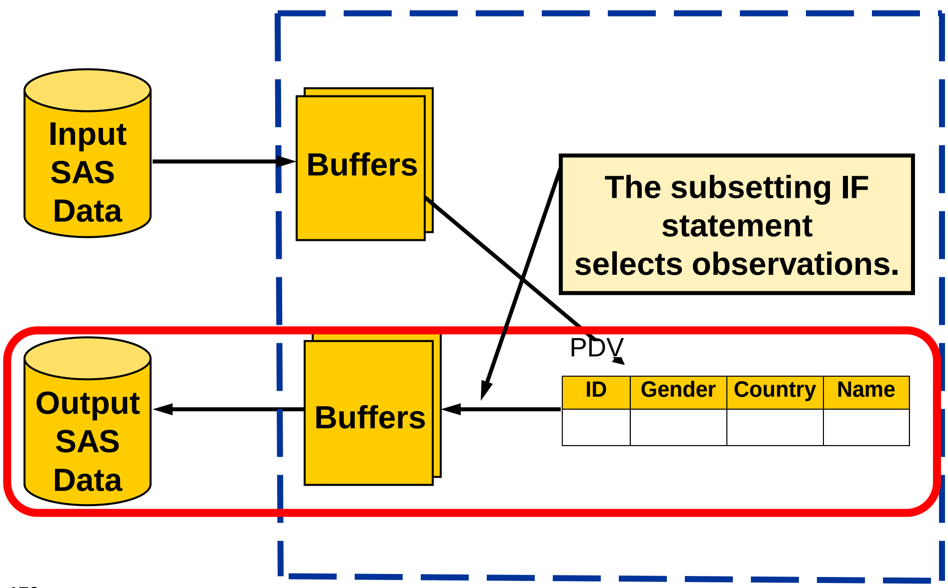
When does the subsetting IF statement select observations?

a. before the observation is copied into the PDV

b. after the observation is in the PDV

175

Using a Subsetting IF



176

WHERE Expression Index Usage

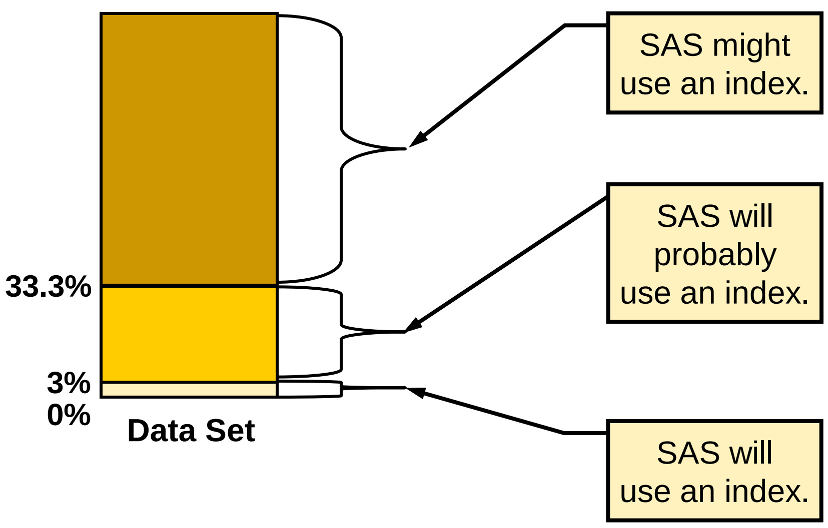
SAS uses the following steps to decide whether to evaluate a WHERE expression using a sequential read or using an index:

- Determine whether the WHERE expression can be satisfied by an existing index.
- Select the best index, if several indexes are available.
- Estimate the number of observations that qualify.
- Compare the probable resource usage for both methods.

✎ SAS estimates the I/O operations for indexed access based on the subset size and sort order.

177

Subset Size



178

Data Order

Unsorted data

Obs	Customer_ID
⋮	⋮
8939	56487
8940	70175
8941	74667
⋮	⋮
32548	89619
32549	70187
32550	76278
⋮	⋮
45775	84989
45776	70201
45777	20209
⋮	⋮

For data that is sorted and indexed on the same variable(s), retrieval time through the index is much faster than either sorted or indexed data alone.

```
where Customer_ID in
   (70201, 70187, 70175);
```

Fewer pages are copied into memory if the data is sorted.

Sorted data

Obs	Customer_ID
⋮	⋮
51050	70175
51051	70177
51052	70180
51053	70181
51054	70183
51055	70184
51056	70186
51057	70187
51058	70188
51059	70191
51060	70192
51061	70193
51062	70194
51063	70195
51064	70197
51065	70199
51066	70200
51067	70201
⋮	⋮

179

Maintaining Indexes

Data Management Tasks	Index Action Taken
Copy the data set with the COPY procedure or the DATASETS procedure	Index file constructed for new data file
Move the data set with the MOVE option in the COPY procedure	Index file deleted from IN= library; rebuilt in OUT= library
Copy the data set with a drag-and-drop action in SAS Explorer	Index file constructed for new file

180

continued...

Maintaining Indexes

Data Management Tasks	Index Action Taken
Rename the data set	Index file renamed
Rename the variable	Variable renamed to new name in index file
Add observations	Value/Identifier pairs added
Delete observations	Value/Identifier pairs deleted; space recovered for re-use
Update observations	Value/Identifier pairs updated if values change

The APPEND procedure and the INSERT INTO statement in the SQL procedure update the index file after all the data is appended or inserted.

181

continued...

Maintaining Indexes

Data Management Tasks	Index Action Taken

Delete a data set.
`proc datasets lib=work;`
`   delete a;`
`run;`
	Index file deleted

Rebuild a data set with a DATA step or the SQL procedure.
`data a;` `set a;` `run;`
`proc sql;` `create table a as` `select * from a;` `quit;`
	Index file deleted

Sort the data set in place with the FORCE option in the SORT procedure.
`proc sort data=a force;`
`   by var;`
`run;`
	Index file deleted

182

Guidelines for Indexing

Suggested guidelines for creating indexes:

- Create an index when you intend to retrieve a small subset of observations from a large data file.
- Do not create an index if the data file page count is less than three pages. It is faster to access the data sequentially.
- Create indexes on variables that are discriminating. These variables precisely identify observations that satisfy WHERE expressions.
- When you create a composite index, make the first key variable the most discriminating.
- Consider the cost of maintaining an index for a data file that is frequently changed.

183

continued...

Guidelines for Indexing

- To minimize I/O for indexed access, sort the data by the key variable(s) before creating the index. Maintain the data file in sorted order by the key variable to improve performance.
- Minimize the number of indexes to reduce disk storage and update costs. Create indexes only on variables that are often used in queries or BY-group processing (when the data cannot be sorted).
- Consider how often your applications use an index. An index must be used often in order to compensate for the resources used in creating and maintaining it.
- When you create an index to process a WHERE expression, do not try to create one index that might be used to satisfy every conceivable query.

184

Index Trade-offs

Advantages	Disadvantages
fast access to a small subset of observations	extra CPU cycles and I/O operations to create and maintain an index
values returned in sorted order	increased CPU to read the data
can enforce uniqueness	extra disk space to store the index file
	extra memory to load the index pages and the compiled SAS C code to use the index

185

Using an Index
Combining a Large Data Set with a Small One

You can use multiple SET statements to combine the two data sets.

```
data catalog_customers(keep=Customer_ID Order_ID Quantity
                        Total_Retail_Price Customer_Country
                        Customer_Gender Customer_Name
                        Customer_Age)
     errors(keep=Customer_ID);
① set work.catalog(keep=Customer_ID Order_ID Quantity
                     Total_Retail_Price);
② set work.all_customers key=Customer_ID;
   if _IORC_=0 then output catalog_customers;
     else do;
       output errors;
       message = iorcmsg();
       _ERROR_=0;
       putlog _n_ = 'The Problem is: ' message;
     end;
run;
```

186

Using the KEY= Option

An index is always used when a SET or MODIFY statement contains the KEY= option.

Specify the KEY= option in the SET statement to use an index to retrieve an observation that has key values equal to the current value of the key variable(s).

General form of the KEY= option:

SET *SAS-data-file-name* KEY=*index-name***;**

187

Using the _IORC_ Automatic Variable

When you use the KEY= option, SAS creates an automatic variable named _IORC_, which is an acronym for input/output return code.

You can use the value of _IORC_ to determine whether the search of the index was successful.

IORC=0	indicates that SAS found a matching observation.
IORC ne 0	indicates that the SET statement did not successfully execute. One possible cause is that SAS did ***not*** find a matching observation.

188

Be Careful When Outputting Data

- Data from the previous observation is retained in the PDV as data coming into the Data Step from a SAS data set does not reinitialize at the start of a new iteration of the Data Step. Thus, if an error occurs in reading a record via index processing, then the previous record's data remains in the PDV.

- If an index read error does occur, you can use the IORCMSG() Data Step function to see a more descriptive message why the error occurred.

189

Using an Index
Combining a Large Data Set with a Small One

You can use multiple SET statements to combine the two data sets.

```
data catalog_customers(keep=Customer_ID Order_ID Quantity
                            Total_Retail_Price Customer_Country
                            Customer_Gender Customer_Name
                            Customer_Age)
     errors(keep=Customer_ID);
① set work.catalog(keep=Customer_ID Order_ID Quantity
                        Total_Retail_Price);
② set work.all_customers key=Customer_ID;
   if _IORC_=0 then output catalog_customers;
     else do;
       output errors;
       message = iorcmsg();
       _ERROR_=0;
       putlog _n_ = 'The Problem is: ' message;
     end;
run;
```

190

Question & Answer

191

Chapter 1: Best Practices

192

Objectives

Examine available efficiency techniques to do the following tasks:

- access database data
- perform remote SAS processing

193

Techniques to Minimize Network Traffic

- Manipulate the data as close to the source of the data as possible.
- Transfer subsets of data or summarized data.

194

Accessing Database Data

When you access database (DBMS) data, the performance of your SAS job can be influenced by the following:

- technique chosen to access the data
- number of columns and rows returned
- ordering of the rows
- choice of PROC SQL or DATA step

195

Choosing a DBMS Access Technique

Access your DBMS data with the following primary techniques:

- SAS/ACCESS LIBNAME engine
- SQL Pass-Through Facility

196

LIBNAME Engine Advantages

DATA and PROC step features:

- You can take advantage of threaded reads.
- The WHERE clause can be passed to DBMS.
- Sort requests can be passed to DBMS.
- Transparent access to DBMS data occurs.
- DATA and PROC step syntax is unchanged.
- Knowledge of DBMS-specific SQL is unnecessary.
- Data retrieval results can be saved as a SAS table or a view.

197

LIBNAME Engine Advantages

When you use the SQL procedure the following are additional features:

- Joins can be passed to DBMS.
- GROUP BY criteria can be passed to DBMS.
- Selected aggregate functions are passed to DBMS.

198

Using SASTRACE and SASTRACELOC

Behind the scenes, when SAS sees that the code references a DBMS table, SAS sends an SQL query directly to the DBMS.

To display this query in the log, you can use the SASTRACE= and the SASTRACELOC= options.

⚠ The SASTRACE= and SASTRACELOC= system options are typically turned on for debugging and off for production jobs.

199

Using SASTRACE and SASTRACELOC

General form of the SASTRACE= option:

```
SASTRACE=',,,d'
```

General form of the SASTRACELOC= option:

```
SASTRACELOC = stdout | SASLOG
```

Example:

```
options sastrace= ',,,d' sastraceloc = saslog;
```

STDOUT is the file reference that can be assigned at invocation for the standard output files.

200

Threaded Reads

A threaded read retrieves the result set from the database on multiple connections between SAS and the DBMS.

Threaded reads are accomplished by doing the following:

- using the LIBNAME engine
- establishing a read connection between the DBMS and each SAS thread
- partitioning the result set across the connections
- passing the rows to SAS simultaneously (in parallel) across the connections

201

Scope of Threaded Reads

SAS steps, named *threaded applications*, are automatically eligible for a threaded read.

- Base SAS procedures
  - MEANS, REPORT, SORT, SQL, SUMMARY, TABULATE
- SAS/STAT procedures
  - GLM, LOESS, REG, ROBUSTREG
- SAS/SHARE procedure
  - SERVER (with the experimental THREADEDTCP option)
- SAS Enterprise Miner procedures
  - DMINE, DMREG

202

Performance Impact of Threaded Reads

Optimal performance of threaded reads requires the following:

- SAS running on a fast uniprocessor or a multiprocessor machine
- the database running on a high-end symmetric multiprocessor (SMP) machine
- partitioned database table(s)
- similar size partitions
- large DBMS result set

203

Reading Columns

Techniques for limiting the number of columns returned from the DBMS include the following:

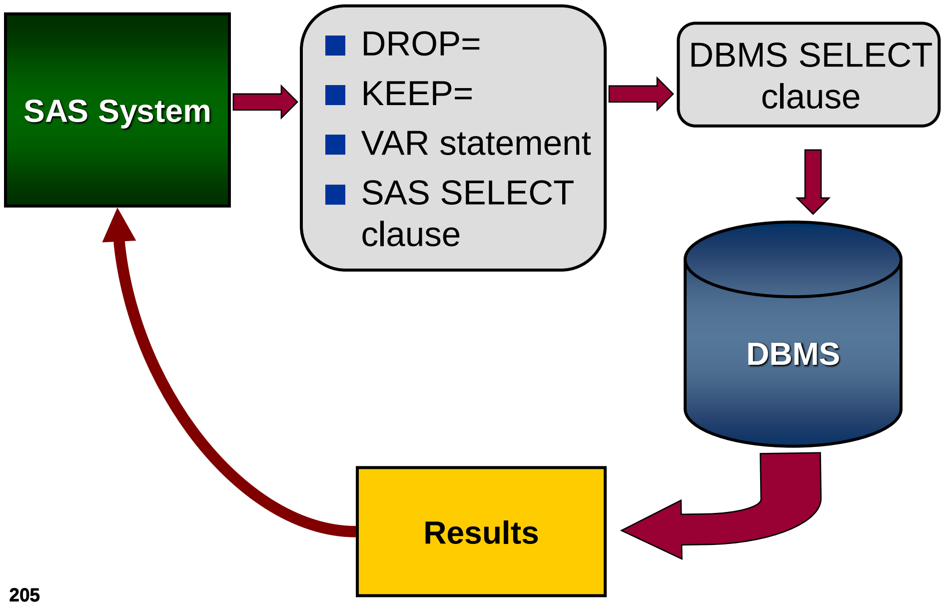
- DROP= SAS data set option
- KEEP= SAS data set option
- VAR statement in the PRINT procedure
- SELECT clause in the SQL procedure

Examples:

```
data temp;
   set mylib.table(keep = name age state);
run;
proc sql;
   select name, age, state
      from mylib.table;
quit;
```

204

Reading Columns



205

Subsetting Using WHERE Criteria

Subset the rows returned from a query to potentially reduce the following:

- processing time
- network traffic
- memory requirements

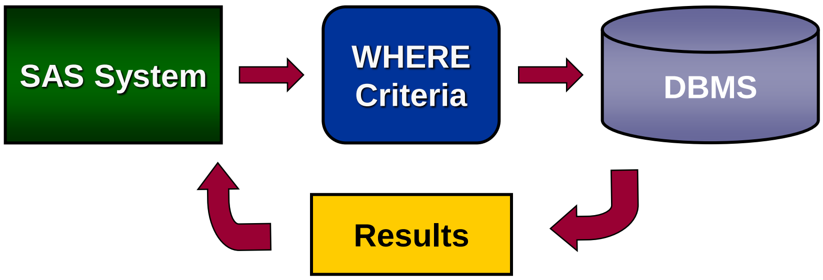
Examples:

```
data temp;
   set mylib.table;
   where state in ('NC', 'SC');
run;
proc sql;
   select *
      from mylib.table
         where state in ('NC', 'SC');
quit;
```

206

Subsetting Using WHERE Criteria

If the SAS/ACCESS engine can do so, the WHERE criteria is passed directly to the database to gain efficiency in processing.



207

Splitting the WHERE Criteria

If the WHERE clause or statement contains SAS enhancements not known to the database, the following events occur:

- The WHERE clause or statement is split up, which enables the DBMS to process as much of the WHERE criteria as possible.
- Rows that satisfy those criteria are sent back to SAS, and then checked to see if they meet the remaining WHERE clause or statement conditions.

208

Sorting the Rows Returned

If sorting is required, you can perform it by doing the following:

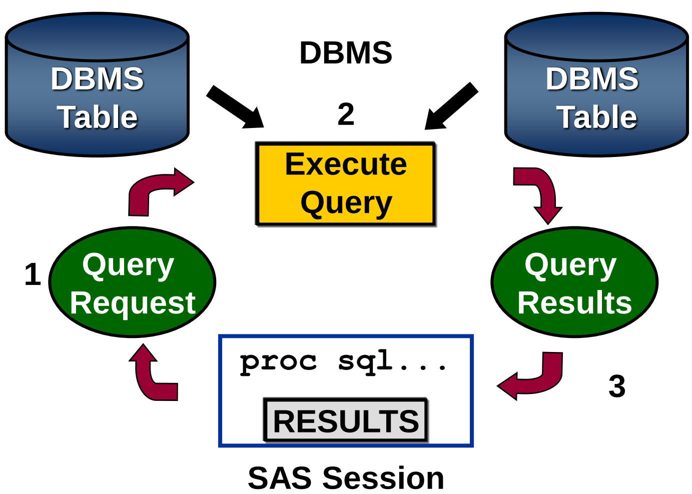
- Using a BY statement in a DATA or PROC step forces the DBMS to sort the data in the order specified by the BY variable(s) before returning the results to SAS.
- Using an ORDER BY clause in PROC SQL which is passed to the DBMS.

```
data temp;
   set mylib.table;
   by state;
run;

proc sql;
   select * from mylib.table
      order by state;
quit;
```

209

SQL Procedure Pass-Through Facility



210

SQL Pass-Through Advantages

- DBMS can optimize all table joins.
- Results of a query can be saved as a SAS data file.
- A SAS SQL view can contain a pass-through query.

211

SQL Pass-Through Example

```
proc sql;
   connect to DBMS (DBMS-specific connection
                     options);
   select *
      from connection to DBMS
         (select flightnumber, flightdate,
                 dayofweek, delay
             from DBMS-table-name
             where substr(destination, 1, 1)
               = 'C');
   disconnect from DBMS;
quit;
```

212

The Embedded LIBNAME Statement

An alternative to coding the LIBNAME statement or using the SQL Pass-Through Facility when you create a PROC SQL view is the embedded LIBNAME statement. The embedded LIBNAME statement has these characteristics:

- is defined in a USING clause within the PROC SQL view
- is assigned when the view begins to execute
- can contain connection information
- uses the LIBNAME engine to access the DBMS
- can store label, format, and alias information
- is de-assigned when the view completes executing

213

The Embedded LIBNAME Statement

Example:

```
proc sql;
   create view sasuser.joinview as
   select m.FlightNumber, m.FlightDate,
          Deplaned, DayOfWeek, Delay
      from oralib.marchflights as m,
           oralib.flightdelays as f
      where m.flightnumber = f.flightnumber
           and m.flightdate = f.flightdate
           and delay > 0
      using libname dbmslib engine
            engine-connection-options;
   select * from sasuser.joinview;
quit;
```

214

SAS/ACCESS Summary

The SAS/ACCESS LIBNAME engine enables transparent access to your DBMS tables. As much code as possible is passed behind the scenes by SAS to the DBMS for processing in order to optimize performance.

The SQL Pass-Through Facility enables the programmer to control the native DBMS SQL queries that are passed to the database to execute.

215

Distributed Processing

Distributed processing can be defined as any one of the following:

- one process (a client or local host) requesting services or data from another process (a server or remote host) executing on a different machine
- the distribution of computing resources to enable utilization of data files, hardware resources, and software resources between different computers
- the division of applications into tasks to be performed on the most appropriate machine, thereby maximizing all computing resources

216

Parallel Processing

Parallel processing is the dividing of an application into subunits of work that can be executed simultaneously.

This parallel processing can occur on the same machine or different machines.

The purposes of parallel processing (also known as multiprocessing or asynchronous processing) are to do the following:

- execute independent tasks in parallel (SAS Version 8)
- execute select dependent tasks in parallel (SAS®9)
- take advantage of multiple processors on a *symmetric multiprocessing* (*SMP*) single machine

continued...

217

Parallel Processing

- take advantage of each processor on a network of machines
- complete a job in less total **elapsed** time than it would take to execute the same job serially
- increase usage of underutilized CPUs
  - exploit current investment
  - prevent further monetary outlay for hardware

218

Grid Computing

A *computing grid* is a collection of multiple computers that solve one application problem.

The concept of grid computing is to tap into the unused processor cycles of computers hooked up to a network to solve problems that require a massive amount of processing power and deal with vast amounts of data.

The idea of grid computing is that any device or computer could hook into a network and make use of the collective unused power of every device on the network or grid.

continued...

219

Grid Computing

The goal is to use the processing cycles of all computers in a network for solving problems too intensive for any stand-alone machine.

Grid computing is not a new concept, but one that has gained renewed interest recently for at least two reasons:

- Grid computing offers a less expensive alternative to purchasing new, larger server platforms.
- Computing problems in several industries involve processing large volumes of data and/or performing repetitive computations to the extent that the workload requirements exceed existing server platform capabilities.

220

Distributed Processing Solutions

A distributed processing solution is implemented when an application requires a service from another computer or itself.

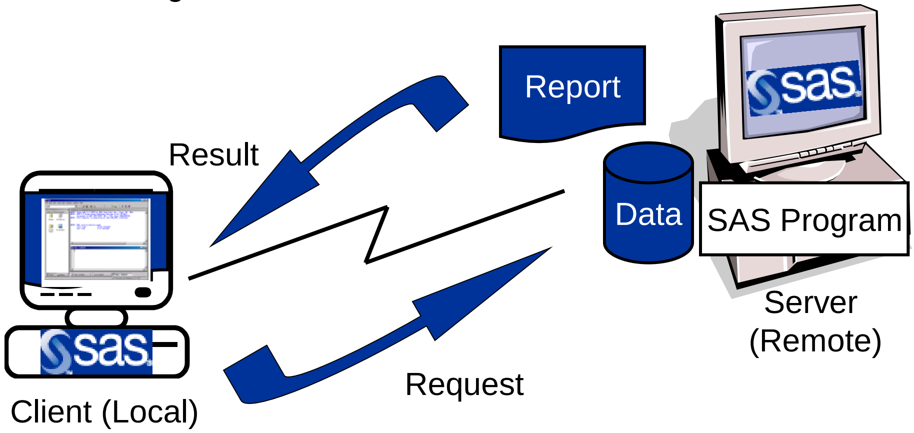
Services include the following:

- compute services
- data transfer services
- remote library services (RLS)

221

Compute Services

Compute services enable you to move any or all segments of an application to other processors to take advantage of hardware, software, and data resources.



222

...

Compute Services Benefits

Compute services are useful when the following conditions exist:

- Processing remote data files that have these attributes:
  - are too large to transfer
  - are frequently updated
  - must remain on the remote platform for security reasons
- The remote machine has necessary hardware or software resources that the local machine does not have.
- A remote CPU is underutilized.

223

Compute Services Considerations

Compute services are less appropriate when these circumstances occur:

- Data files are small.
- A remote CPU is near 100% utilization.
- The remote computer's I/O subsystem is heavily loaded.
- The remote computer has little memory available.

224

Requirements for Compute Services

To use compute services, you need to do the following:

- have SAS/CONNECT on both machines
- sign on to the remote machine to begin a remote SAS session
- submit an RSUBMIT block

225

Using Compute Services

Before you use compute services, a connection to the remote machine must be established. You can do either of the following:

- Sign on directly with a SIGNON statement.
- Use the AUTOSIGNON=YES option to specify to sign on when compute services needs to start a task on the remote machine.

226

Using Compute Services

The AUTOSIGNON option enables the local SAS session to automatically invoke a new SAS session when a request is made.

General form of the AUTOSIGNON option:

> **OPTIONS** AUTOSIGNON = NO|YES;

The default is NO.

Example:

```
options autosignon = yes;
```

227

Using Compute Services

After a connection to a remote machine is established, you can send code to execute on that machine by enclosing the code in an RSUBMIT block.

General form of the RSUBMIT block:

> **RSUBMIT** *<remote-machine-name>*;
> *code to be processed on the remote machine*
> **ENDRSUBMIT;**

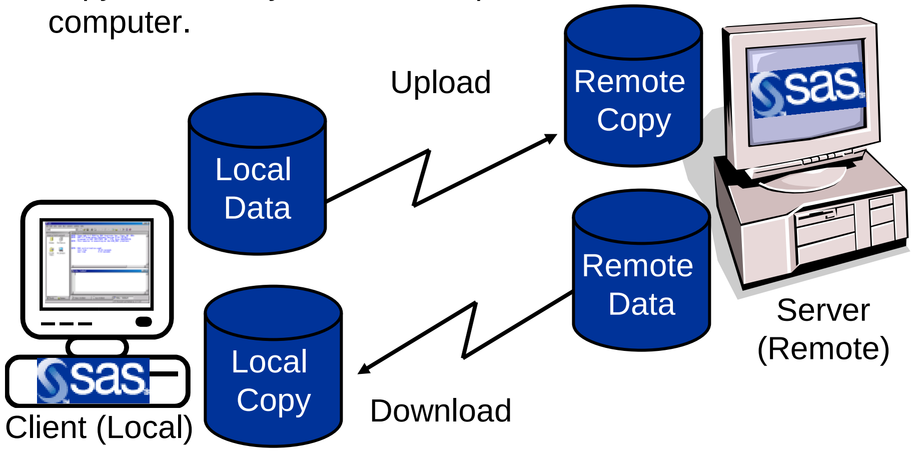
Example:

```
local SAS session
rsubmit bcom1;
  SAS code to run on remote machine
endrsubmit;
```

228

Data Transfer Services

Using data transfer services, you can transfer a copy of a remote data file to your local computer for processing, or copy data from your local computer to the remote computer.



229 ...

The UPLOAD and DOWNLOAD Procedures

To perform data transfer, use the UPLOAD and DOWNLOAD procedures. The UPLOAD and DOWNLOAD procedures enable you to do the following:

- transfer an entire SAS library or selected members of a SAS library in a single step
- transfer an entire SAS catalog or selected entries in a catalog in a single step
- transfer external files

continued...

230

The UPLOAD and DOWNLOAD Procedures

- enable WHERE processing to subset the data before the transfer occurs
- enable data set options (for example, DROP= or KEEP=) when transferring individual SAS data sets
- replicate certain data set attributes, including indexes and integrity constraints

231

UPLOAD and DOWNLOAD Procedure Benefits

Benefits of using the UPLOAD and DOWNLOAD procedures over other data transfer applications are as follows:

- control over variables and observations transferred
- transparent translation of SAS files across operating system types (for example, EBCDIC to ASCII)
- transparent translation of SAS files across differing releases of SAS

232

Transferring a SAS Data Library

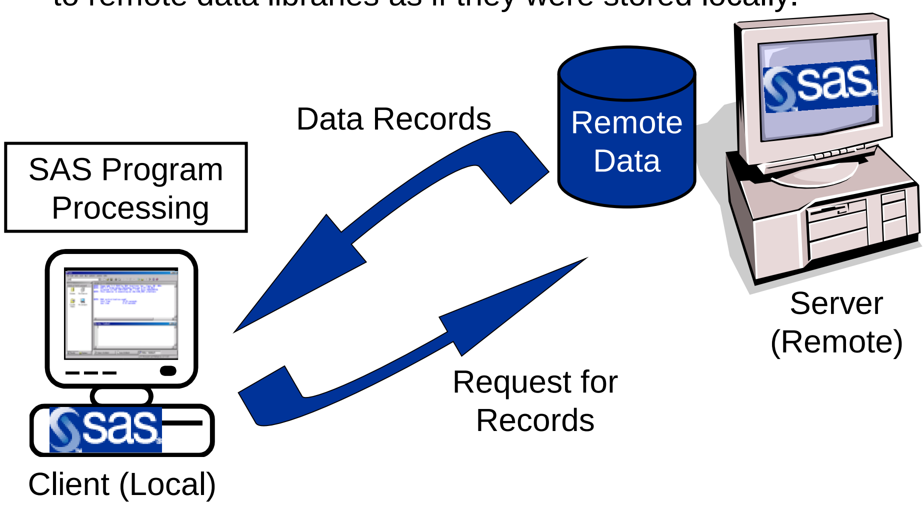
Example: Transfer the entire SAS data library on the remote machine to the local machine.

```
libname orionwin 'directory-on-Windows';
rsubmit bcom1;
libname orionunx 'directory-on-UNIX';
proc download inlib = orionunx
              outlib = orionwin;
run;
endrsubmit;
```

233

Remote Library Services

Remote library services (RLS) provide transparent access to remote data libraries as if they were stored locally.



234

...

Benefits of RLS

- A single copy of the data can be maintained while processing is performed on the local machine.
- The data appears to be local.
- RLS enables updates to remote data as a result of local processing.
- RLS permits a user interface to reside on the local system while the data is on a remote system.

235

Considerations for RLS

- Multiple passes of the data require the same data to go across the network multiple times. Examples include the following:
  - statistical procedures
  - multiple PROC steps on the same data
- Network traffic might increase significantly.

236

Requirements for RLS

To use RLS, you need to do one of the following:

- have SAS/CONNECT on both machines or SAS/CONNECT on the local machine and SAS/SHARE on the remote machine
- sign on to the remote machine to begin a remote SAS session, if SAS/CONNECT is used on the remote machine
- issue a LIBNAME statement in your local session with the SERVER= option

237

SERVER= Option

General form of the SERVER= option in the LIBNAME statement:

LIBNAME *libref* '*SAS-data-library*' | SLIBREF=*server-libref*
SERVER=*remote-host*;

Examples:

Access a library stored on your user ID on UNIX:

```
libname rmtunx '/orion/sasdata' server = sdcunx;
```

Access the Work library on z/OS:

```
libname rmtwork slibref = work server = sdcmvs;
```

238

Decisions, Decisions, Decisions

When deciding which strategy is most appropriate for your application, you must determine the following:

- computing needs of your application
- computing capacity and load of each computer
- charge-backs for use of mainframe or UNIX time and data storage
- amount of data to be processed
- load on your network
- output needs
  - printers
  - tape drives
  - GUI display

continued...

239

Decisions, Decisions, Decisions

- appropriateness of the data location
  - the frequency of data updates
  - available disk space
  - the increased speed of the application if the data is on the same computer
  - problems related to storing multiple copies of the data

240