Utilizing Best Practices with SAS and Macros made easy

Presented by: Josée Ranger-Lacroix
SAS Canada - Education

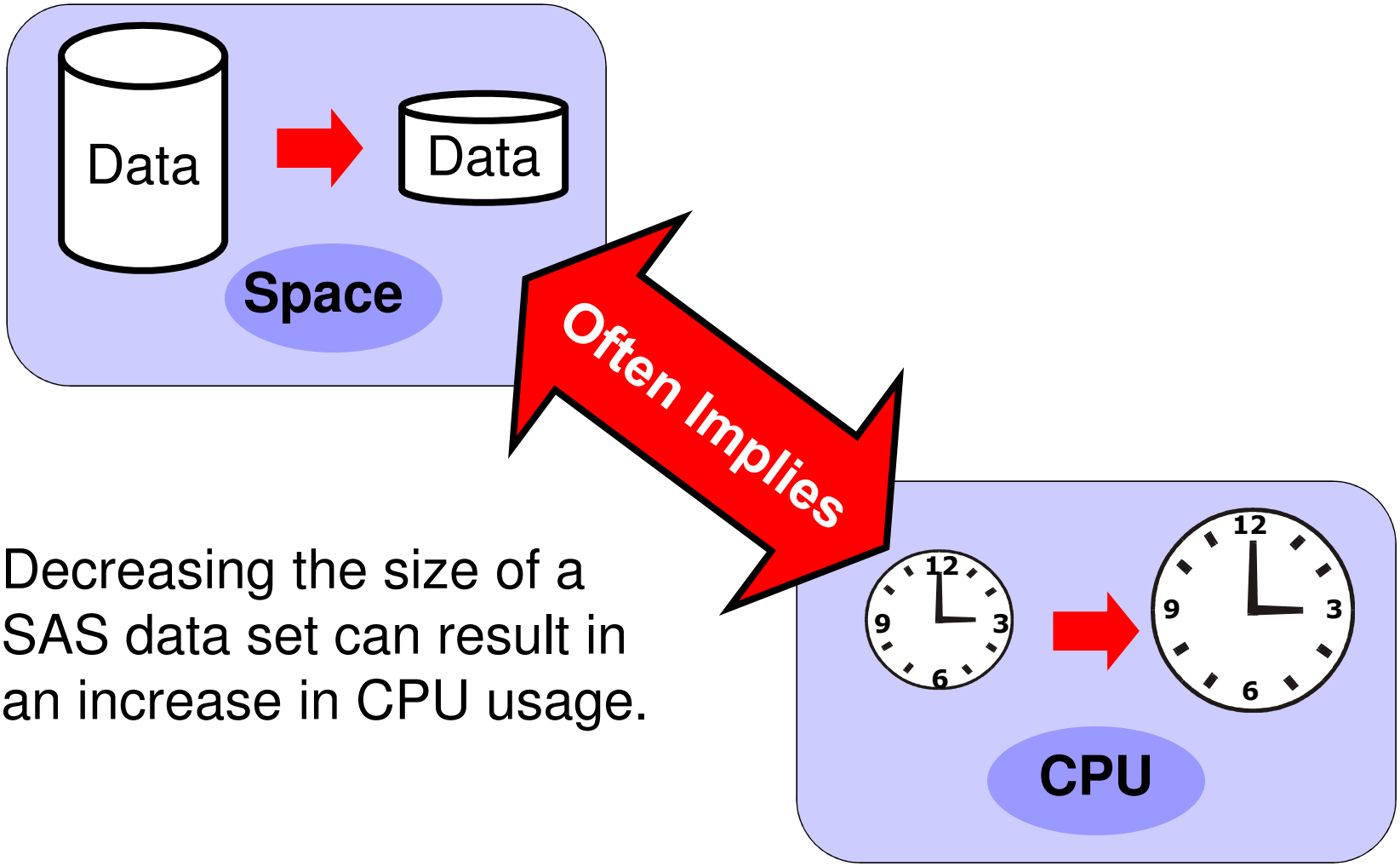
What Are Best Practices?

As programmers, you want to perform data driven tasks as efficiently as possible and optimize the use of the following resources:

- I/O
- CPU
- memory
- data storage space
- network bandwidth
- programmer time

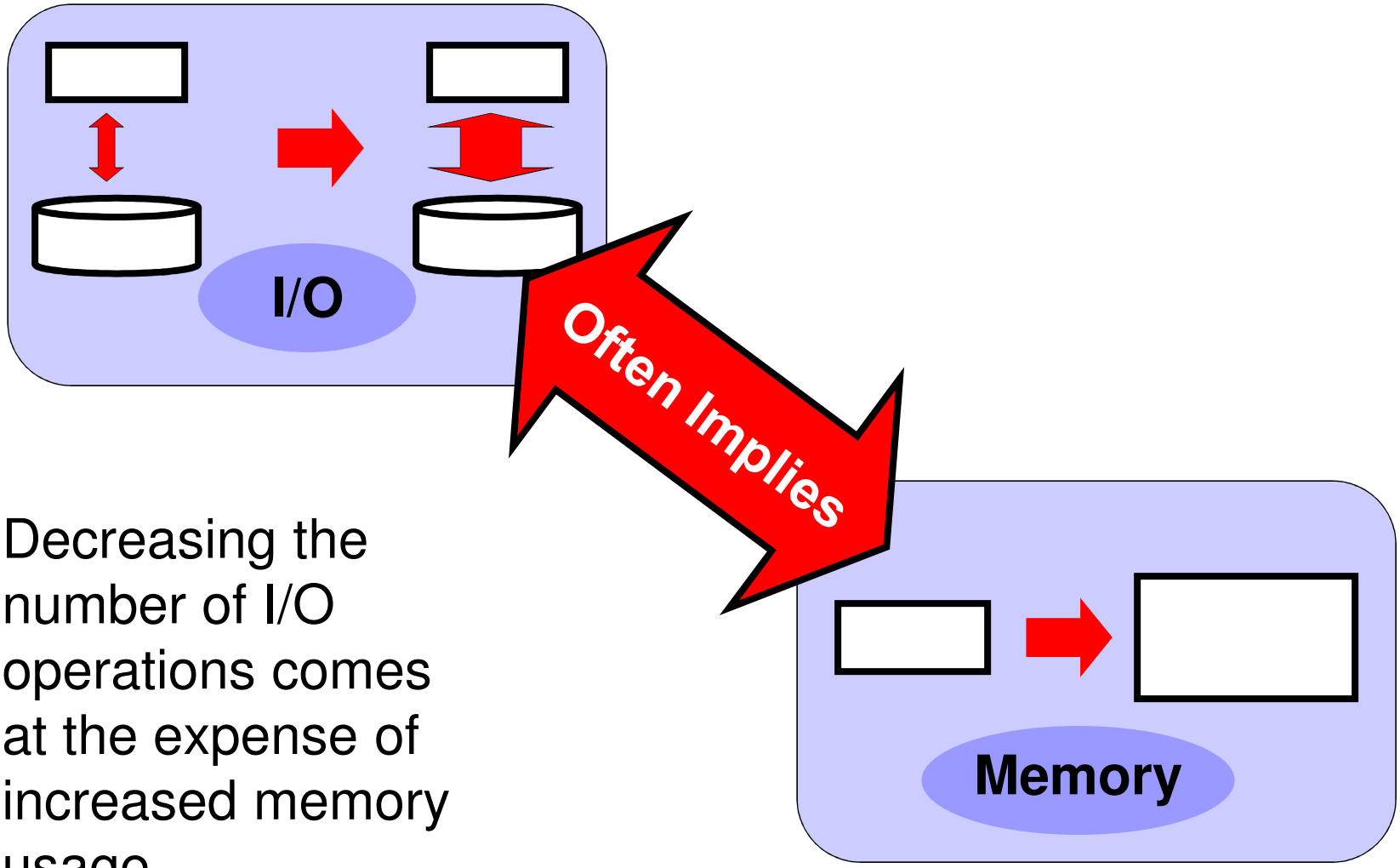
Reducing one resource often increases another.

Understanding Efficiency Trade-offs



Decreasing the size of a SAS data set can result in an increase in CPU usage.

Understanding Efficiency Trade-offs



Decreasing the number of I/O operations comes at the expense of increased memory usage.

Techniques to Reduce Network Traffic (I/O)

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

5 Techniques for Conserving CPU

- Execute only 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.

Executing Only Necessary Statements

Subsetting IF Statement at Bottom of Step

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

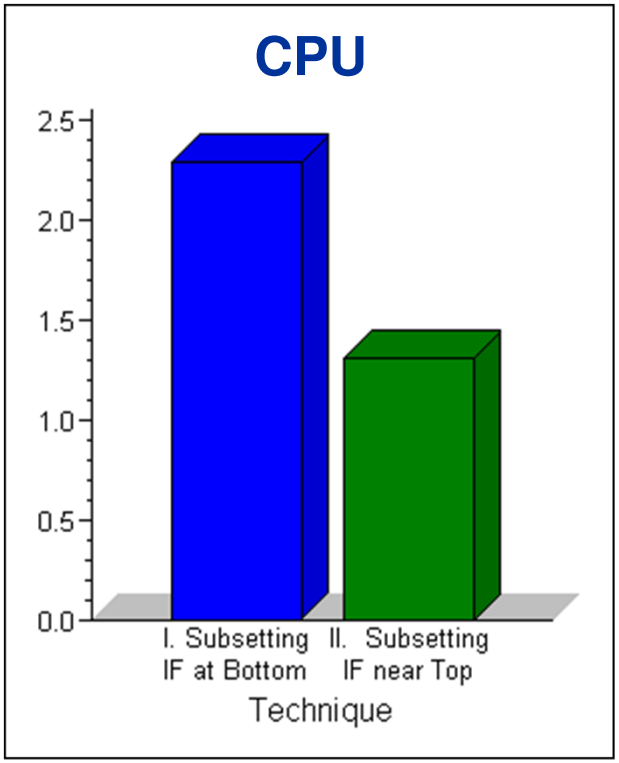
```
data totals;
   set ia.sales;
   PercentCap =
      sum(Num1st,NumEcon,NumBus)/CapPassTotal;
   NumNonEconomy = sum(Num1st,NumBus);
   CargoKG = CargoWeight*0.454;
   Month = month(FltDate);
   if PercentCap < 0.8;
run;
```

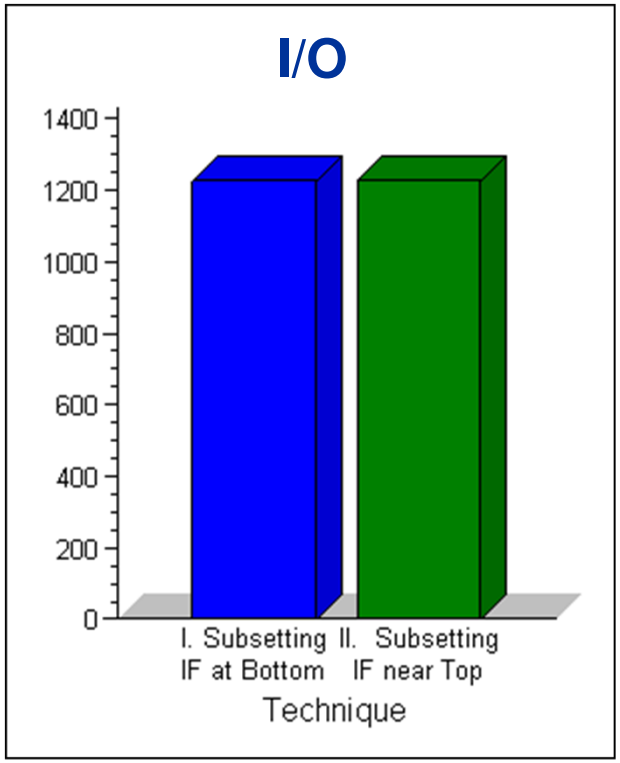
Subsetting IF Statement as High as Possible

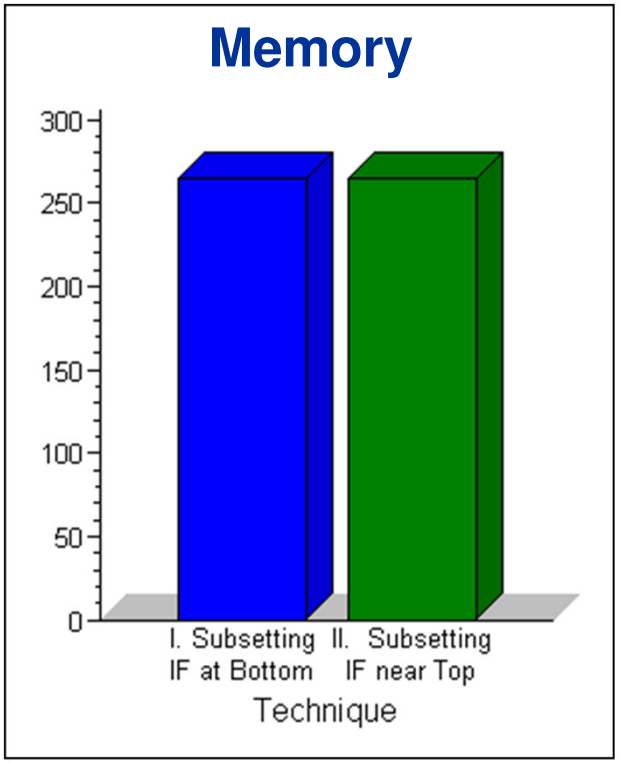
```
data totals;
   set ia.sales;
   PercentCap =
      sum(Num1st,NumEcon,NumBus)/CapPassTotal;
   if PercentCap < 0.8;
   NumNonEconomy = sum(Num1st,NumBus);
   CargoKG = CargoWeight*0.454;
   Month = month(FltDate);
run;
```

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







Using Parallel IF Statements

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

```
data month;
   set ia.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;
```

Using ELSE-IF Statements

```
data month;
   set ia.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;
```

Using the Function Only Once

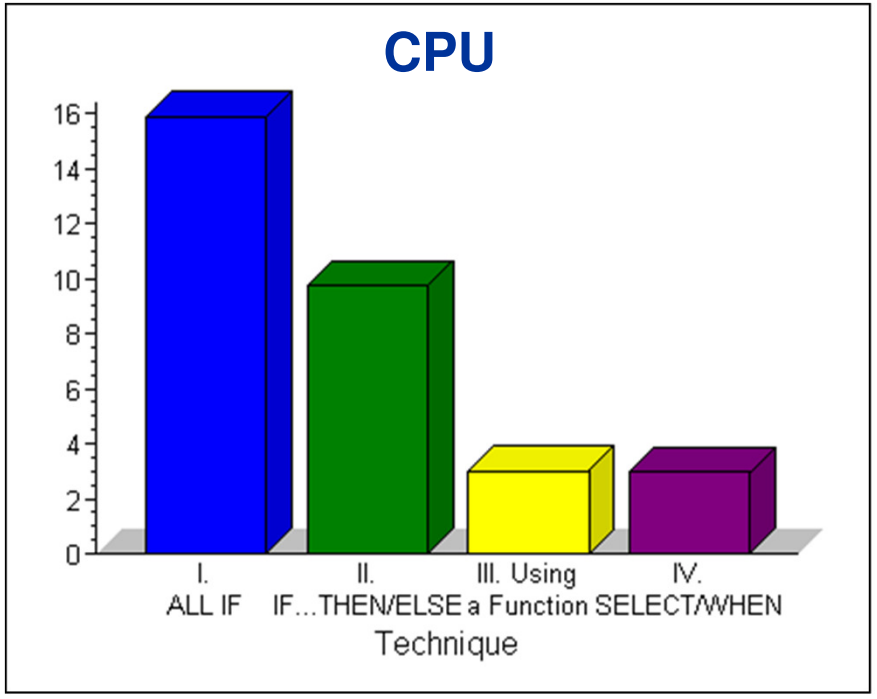
```
data month(drop=mon);
   set ia.sales;
   mon = month(FltDate);
   if mon = 1 then Month = 'Jan';
   else if mon = 2 then Month = 'Feb';
   else if mon = 3 then Month = 'Mar';
   else if mon = 4 then Month = 'Apr';
   else if mon = 5 then Month = 'May';
   else if mon = 6 then Month = 'Jun';
   else 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;
```

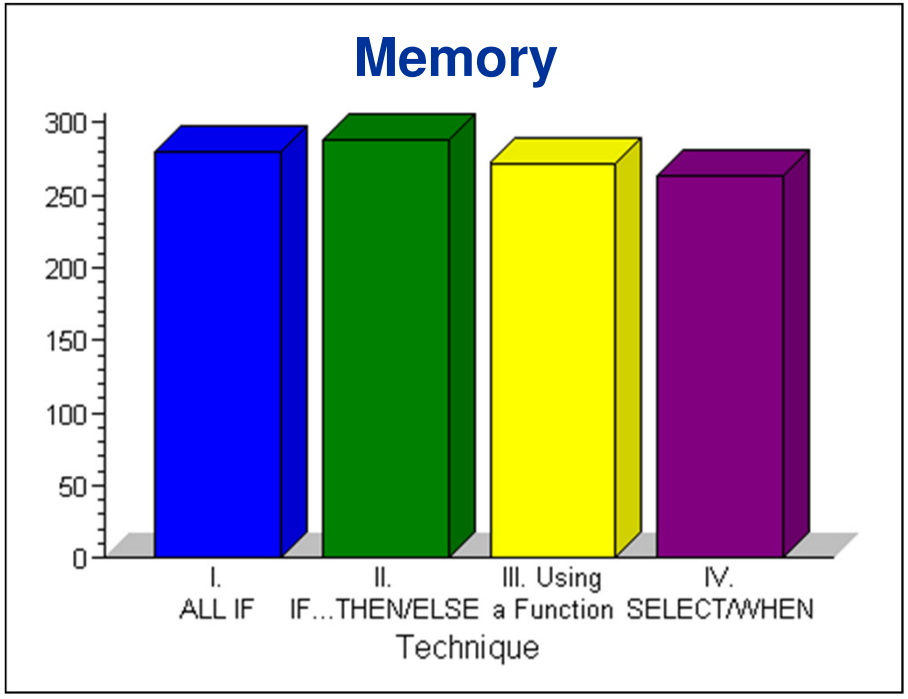
Using a SELECT Block

```
data month;
   set ia.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;
```

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





The I/O for each technique is the same.

Guidelines for Writing Efficient IF/THEN Logic

- Use IF-THEN/ELSE statements when the following circumstances exist:
  - There are few conditions to check.
  - The data values are not uniformly distributed.
  - The values are character or discrete numeric data.
- Check the most frequently occurring condition first.

Eliminating Unnecessary Passes through the Data

Multiple DATA Steps

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

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

continued...

Multiple DATA Steps

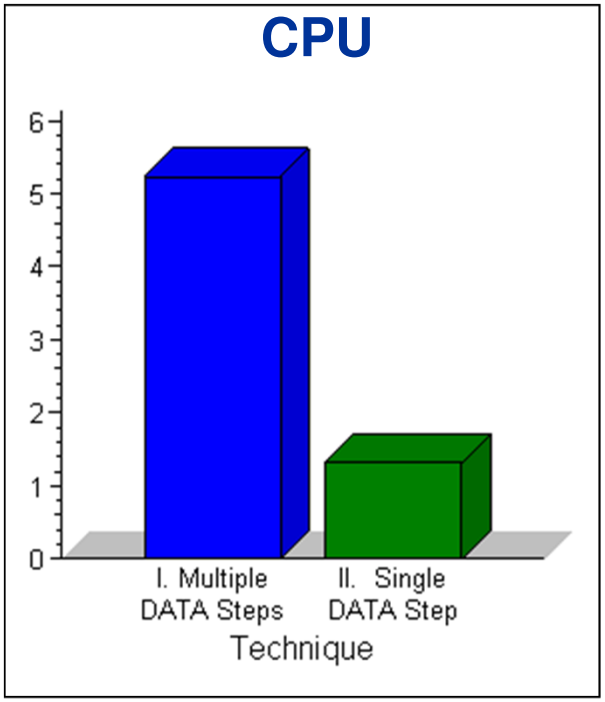
```
data iad;
    set ia.sales;
    if Dest = 'IAD';
run;
data jfk;
    set ia.sales;
    if Dest = 'JFK';
run;
data mia;
    set ia.sales;
    if Dest = 'MIA';
run;
data pwm;
    set ia.sales;
    if Dest = 'PWM';
run;
```

Single DATA Step

```
data rdu bos iad jfk mia pwm;
   set ia.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;
```

Comparing Techniques

Technique	CPU
I. Multiple DATA Steps	5.2
II. Single DATA Step	1.3
Percent Difference	74.8



DATA Step / PROC SORT Step

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

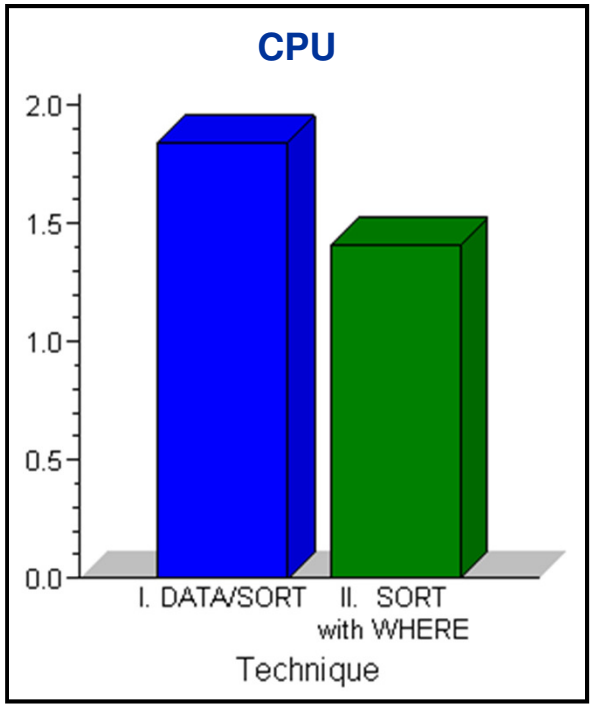
```
data east;
   set ia.sales;
   where Dest in
         ('RDU','BOS','IAD','JFK','MIA','PWM');
run;
proc sort data = east;
   by Dest;
run;
```

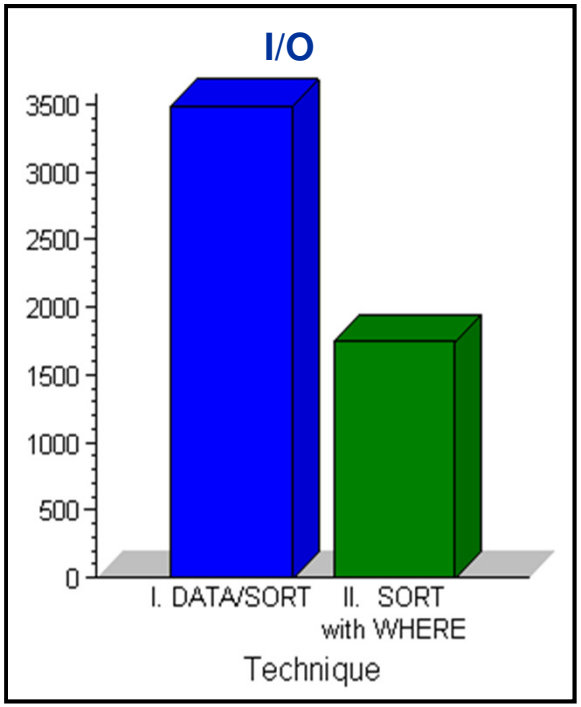
PROC SORT Step

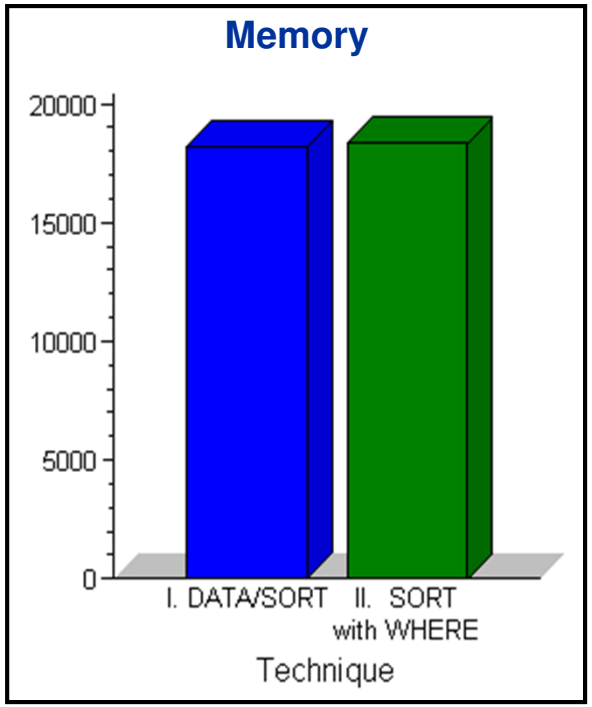
```
proc sort data = ia.sales out = east;
    by Dest;
    where Dest in
        ('RDU','BOS','IAD','JFK','MIA','PWM');
run;
```

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

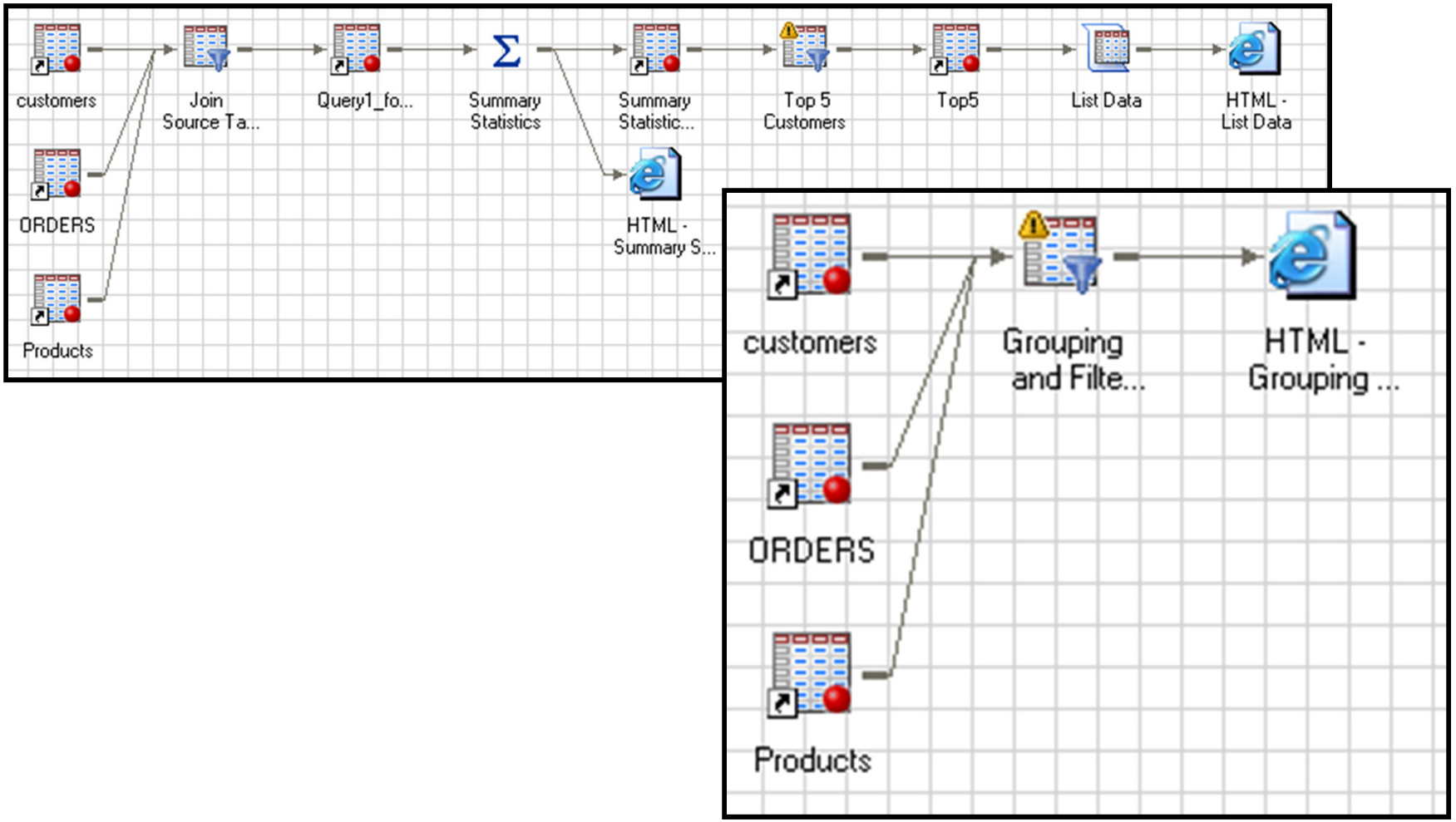






Eliminate steps – True for all applications

These 2 processes give you the same results in Enterprise Guide.



Business Task

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

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

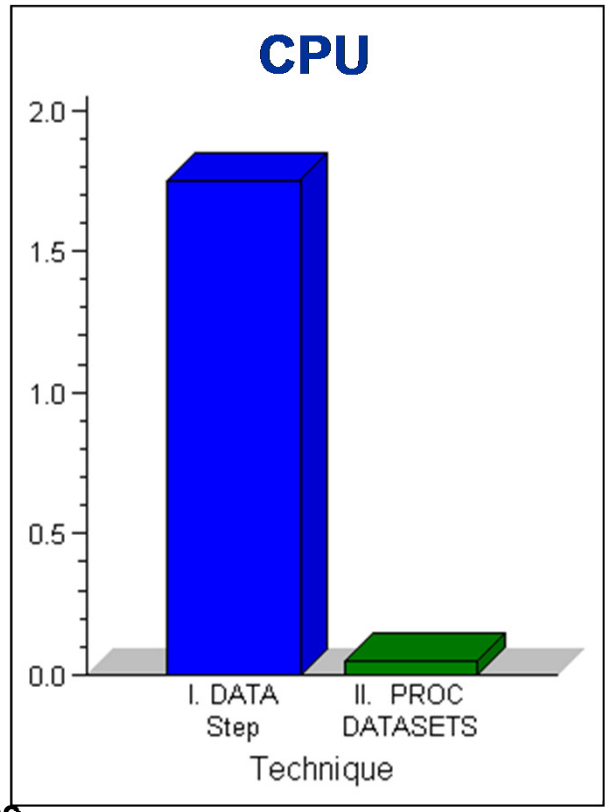
DATA Step / PROC DATASETS

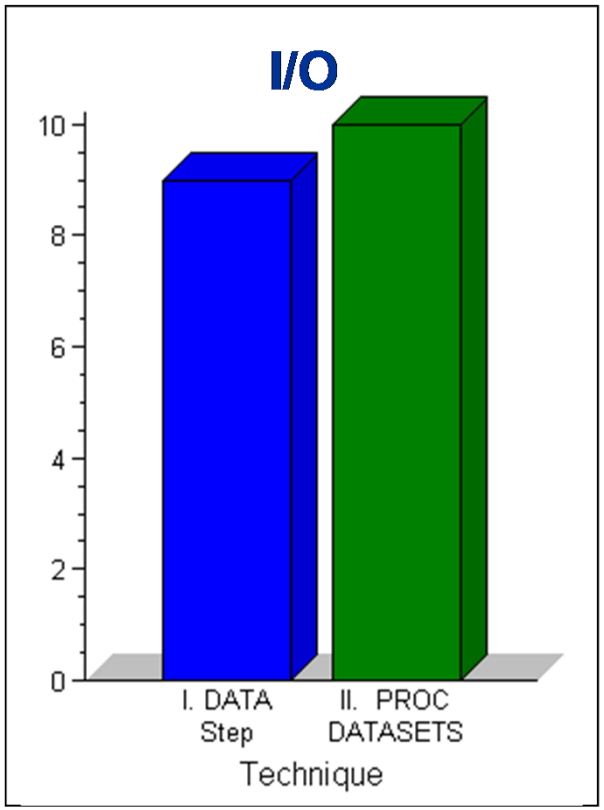
```
data ia.salesc;
   set ia.salesc;
   rename FlightIDNumber = FlightID;
   format FltDate date9.;
run;
```

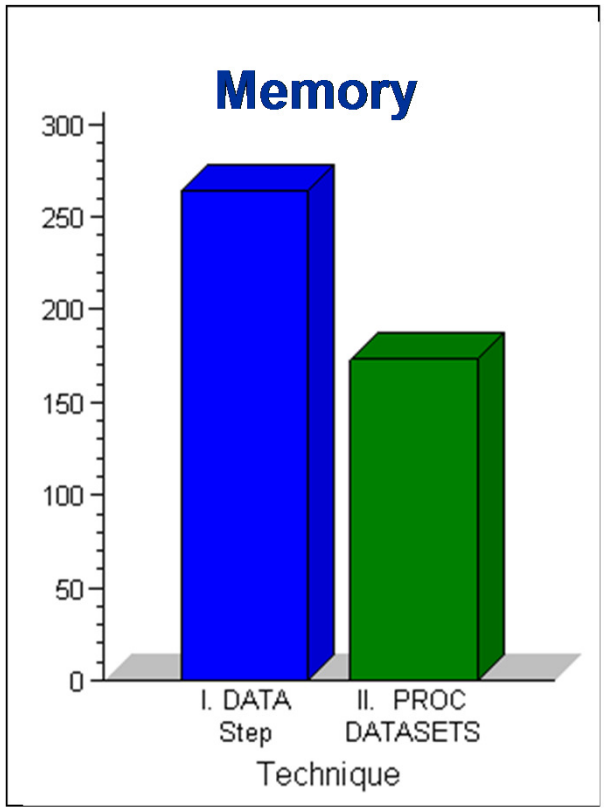
```
proc datasets library=ia nolist;
   modify salesc;
      rename FlightIDNumber=FlightID;
      format FltDate date9.;
quit;
```

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







Reading and Writing Only Essential Data

Subsetting IF versus WHERE

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

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

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

Subsetting Using IF

```
23         data year99;
24            set year8300;
25            if year = 1999;
26         run;
```

```
NOTE: There were 40250204 observations read from
the data set YEAR8300.
NOTE: The data set WORK.YEAR99 has 2413228
observations and 14 variables.
NOTE: DATA statement used (Total process time):
      real time          5:11.07
      cpu time           1:10.04
```

Subsetting Using WHERE

```
23         data year99;
24            set year8300;
25            where year = 1999;
26         run;
```

NOTE: There were 2413228 observations read from the data set YEAR8300.
NOTE: The data set WORK.YEAR99 has 2413228 observations and 14 variables.

NOTE: DATA statement used (Total process time):
real time 2:47.32
cpu time 45.80

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;
```

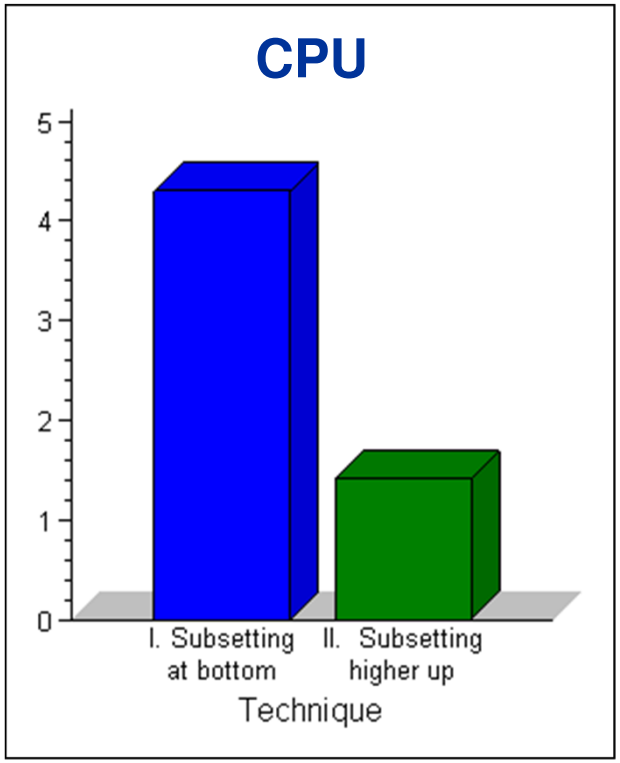
Reading Selected Variable(s) and Subsetting

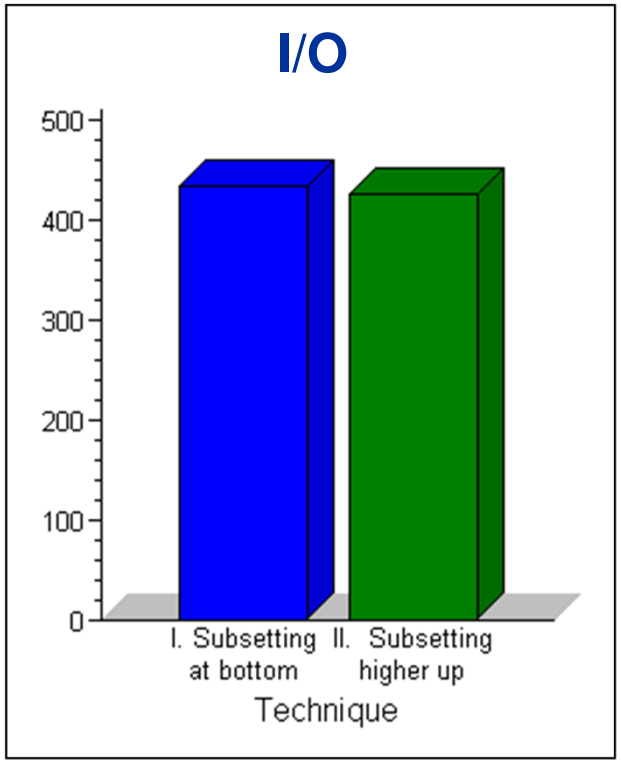
```
data west;
   infile rawdata ;
   input @18 Dest $3. @;
   if Dest in ('LAX','SEA','SFO');
   input @1 FlightID $7.  RouteID $7.
         Origin $3.
         @21 DestType $13.  FltDate date9.
         Cap1st 8.  CapBus 8.
         CapEcon 8. CapPassTotal 8.
         CapCargo 8.  Num1st 8.
      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

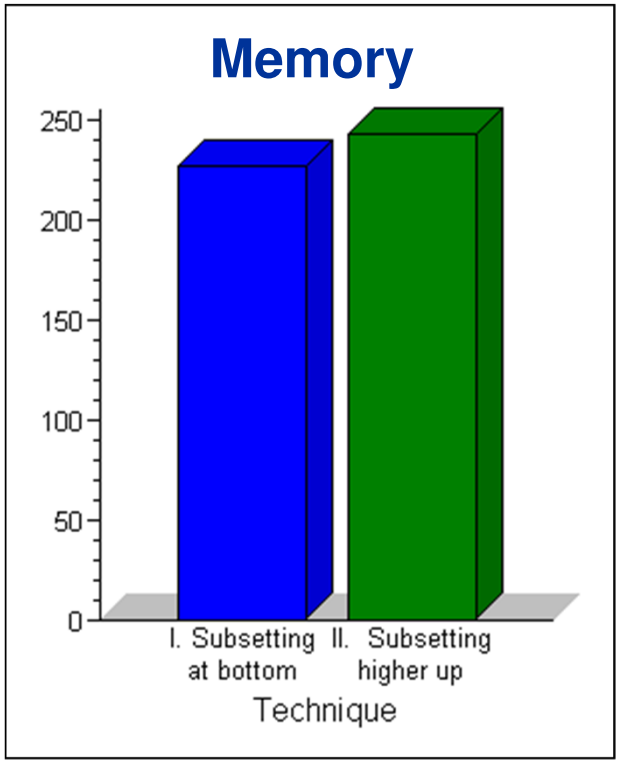
Add

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

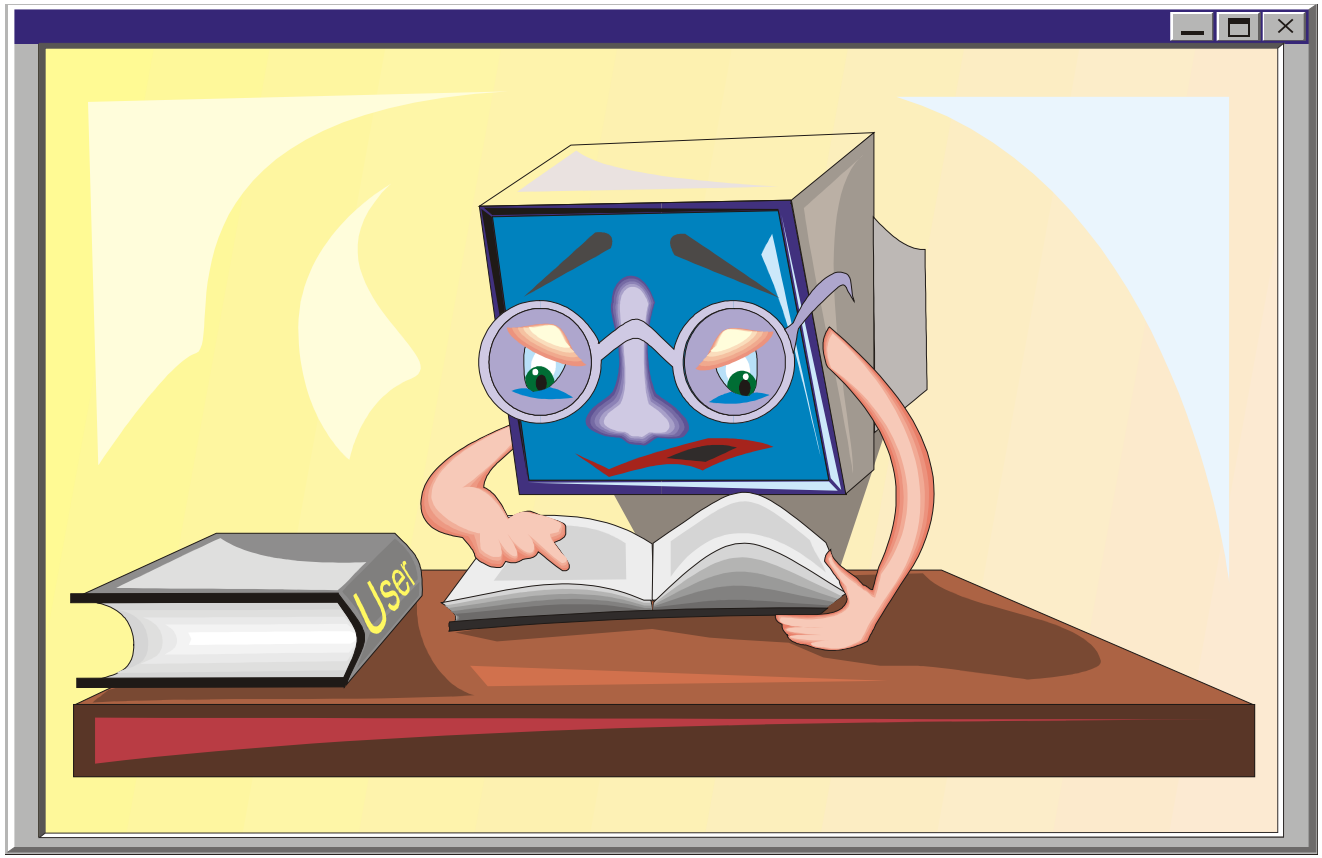






Read and Write Data Selectively

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



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 **ia.sales** that has 21 variables.

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

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

Reading All Variables/Writing Two Variables

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

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

Reading Three Variables

```
data totals;
   set ia.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;
```

Reading Three Variables/Writing Two Variables

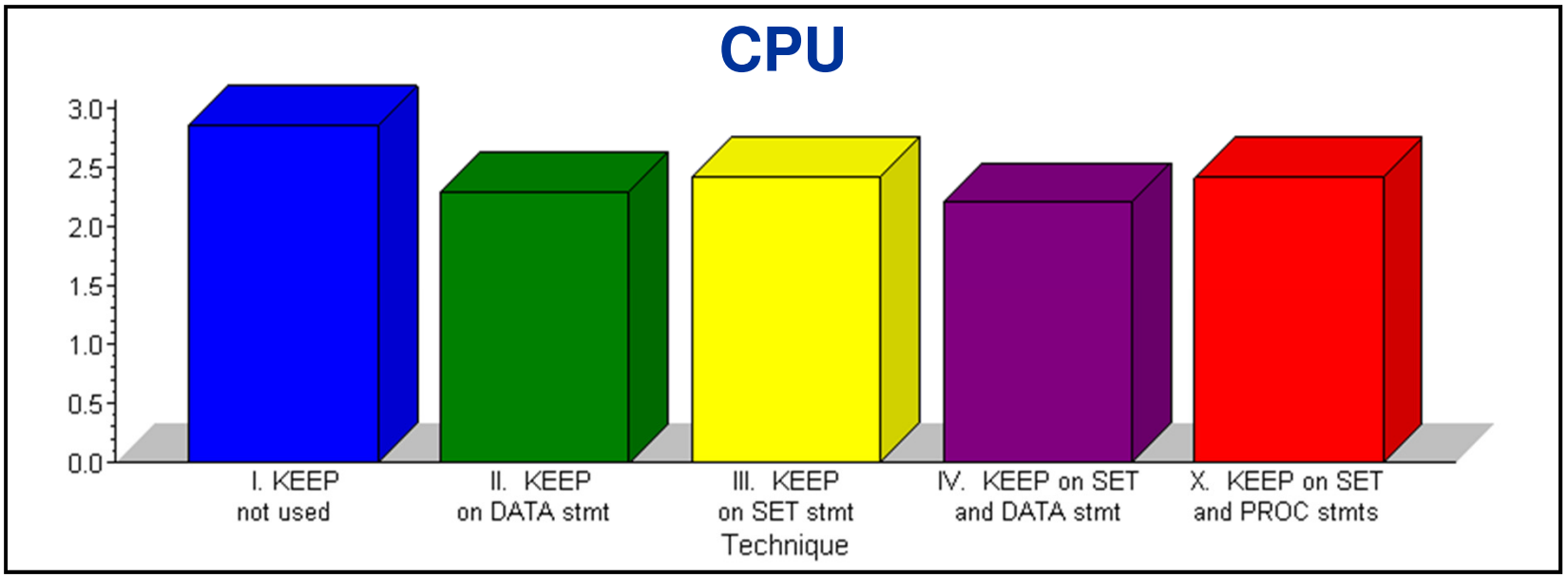
```
data totals(keep = Dest NonEconPass);
   set ia.sales(keep = Dest Num1st
                       NumBus);
   NonEconPass =
      sum(Num1st,NumBus);
run;

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

Reading Three Variables/Reading Two Variables

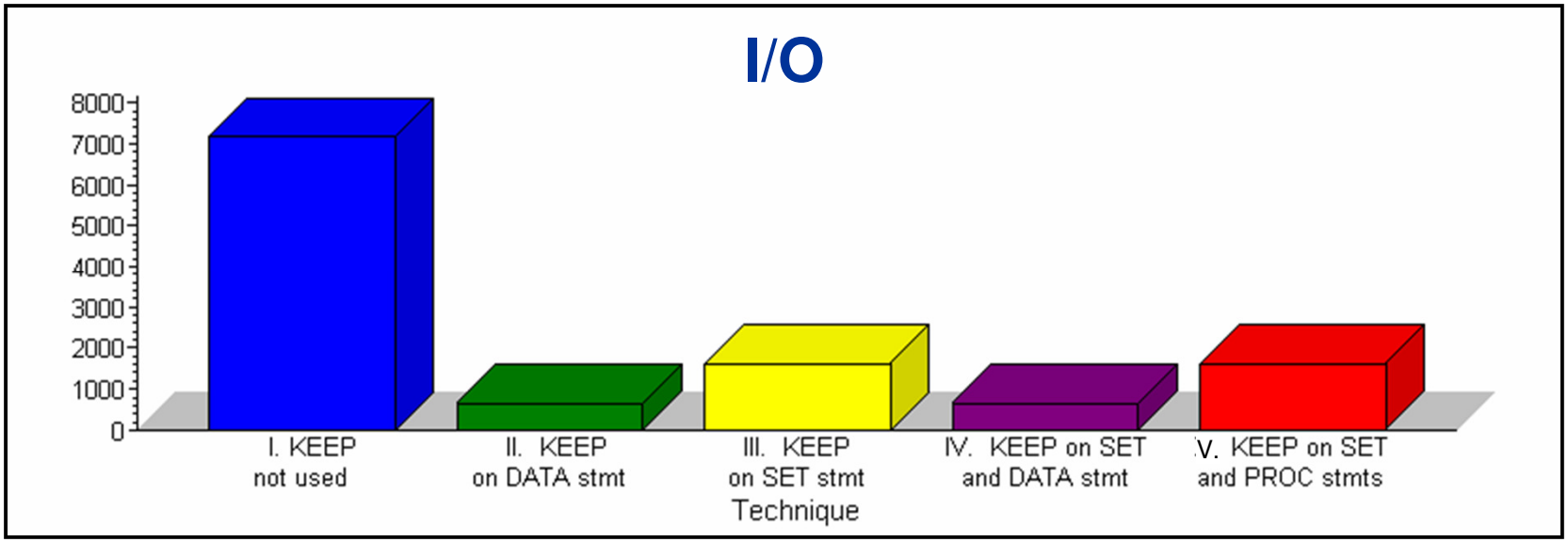
```
data totals;
   set ia.sales(keep = Dest Num1st
                       NumBus);
   NonEconPass =
      sum(Num1st,NumBus);
run;

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

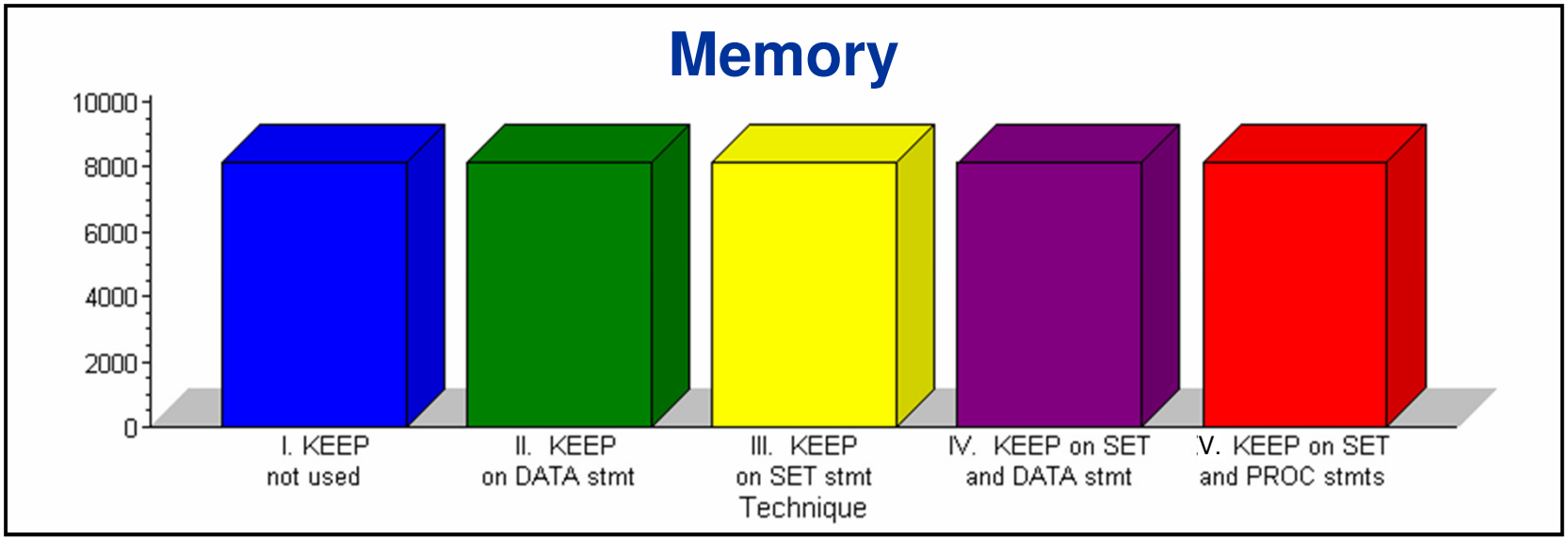
Comparing Techniques

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



Comparing Techniques





Use the right tools

Using Procedures

Example of selecting appropriate procedures for data processing:

Use the SUMMARY procedure…

```
proc summary data=orion.shoe_vendors nway;
   var Mfg_Suggested_Retail_Price;
   class Line_Name;
   output out=summary(keep=Line_Name Avg_MSP)
          mean=Avg_MSP;
run;
```

Using Procedures

…instead of the DATA step.

```
proc sort data=orion.shoe_vendors(keep=Line_Name
                                   Mfg_Suggested_Retail_Price
                    out=shoe_vendors;
   by Line_Name;
run;

data sum;
   keep Line_Name Avg_MSP;
   set shoe_vendors;
   by Line_Name;
   if first.Line_Name then do;
      Tot_MSP=0;
      Count=0;
   end;
   Tot_MSP + Mfg_Suggested_Retail_Price;
   if Mfg_Suggested_Retail_Price ne . then Count+1;
   if last.Line_Name then do;
      Avg_MSP=Tot_MSP/Count;
      output;
   end;
run;
```

Using Procedures

Use PROC SQL…

```
proc sql;
create table CustOrdProd as
select Customer_Name, Quantity ,Total_Retail_Price,
       Product_Name, Supplier
 from    CustOrd          as co,
         product_dim      as p,
         customer         as c,
         order_fact       as o
   where co.product_id=p.product_id and
         c.customer_id=o.customer_id
   order by product_id;
 quit;
```

…instead of several DATA and PROC steps.

Using PROC SQL

```
proc sort data=orion.order_fact out=work.order_fact;
   by Customer_ID;
run;
proc sort data=orion.customer out=work.customer_id;
   by Customer_ID;
run;
data CustOrd;
   merge work.customer(in=cust)
         work.order_fact(in=order);
   by Customer_ID;
   if cust=1 and order=1;
   keep Customer_Name Quantity Total_Retail_Price Product_ID;
run;
proc sort data=CustOrd;
   by Product_ID;
run;
proc sort data=orion.product_dim out=work.product_dim;
   by Product_ID;
run;
data CustOrdProd;
   merge CustOrd(in=ord)
         product_dim(in=prod);
   by Product_ID;
   if ord=1 and prod=1;
   keep Customer_Name Quantity Total_Retail_Price Product_Name Supplier;
run;
```

Advantages of the SQL over the DATA Step

SQL	DATA Step
Is very flexible when joining multiple tables that do not have key variables in common	Can require several steps to join multiple tables with different key variables
Can, in some cases, replace multiple SAS steps	Can require several steps
Is the native language of databases	Might need to generate SQL to get to data that is not SAS data

Advantages of the DATA Step over SQL

DATA Step	SQL
Can read data from many different sources	Can only read from SAS database tables
Can create multiple tables in a single pass of the data	Can only output one table at a time
Has comprehensive conditional processing	Only has the CASE clause
Can deal with repetitive programming using loops and arrays	Does not support loops or arrays

Choose the right tool for the task to be completed.

Selecting Appropriate Functions

Example of selecting appropriate functions for data processing:

Use one of the CAT functions…

```
data description;
   set orion.organization_dim;
   Employment_Description=catx(' - ', of Company -- Job_Title);
run;
```

…instead of the concatenation operator and the TRIM function.

```
data description;
   set orion.organization_dim;
   Employment_Description=trim(Company)||' - '||
                          trim(Department)||' - '||
                          trim(Section)||' - '||
                          trim(Org_Group)||' - '||
                          trim(Job_Title);
run;
```

Keeping up to date

Every releases new language elements are added:

Functions:

- PROPCASE, CATX, PERL Regular Expressions…

Formats/Informats:

- ANYDT…, NL …

New/enhanced procedures:

- POWER, GAREABAR, IMPORT …

Macros:

- %SYSMACDELETE, %SYSMACEXEC

Objects & Modules:

- ODS, XMLMAP engine, HASH…

Hash Objects: Merging 2 tables

```
data both(drop=rc);
declare Hash Plan ();
rc = plan.DefineKey ('Plan_id');
rc = plan.DefineData ('Plan_desc');
rc = plan.DefineDone ();
do until (eof1) ; /* loop to read records from Plan */
       set plans end = eof1;
       rc = plan.add (); /* add each record to the hash table */
end;
do until (eof2) ; /* loop to read records from Members */
       set members end = eof2;
       call missing(Plan_desc);
       rc = plan.find (); /* lookup each plan_id in hash Plan */
       output; /* write record to Both */
end;
stop;
run;
```

Hash Objects

In the following paper, **I cut my processing time by 90% using hash tables – You can do it too!** ,Jennifer K. WarnerFreeman looked at different ways to merge tables.

“In my own experience I took a process … that was taking between 2 and 4 hours (depending on network traffic) to run using a PROC SQL join, and using hash tables cut the execution time to a consistent 11 minutes.”

http://www.nesug.info/Proceedings/nesug07/bb/bb16.pdf

Conclusion

- **BENCHMARK all approaches on realistic data and hardware**

Other Techniques to Explore

- BUFNO= and BUFSIZE=
- SGIO option
- SASFILE statement
- HASH tables, Arrays, MERGE, PROC SQL
- Indexes
- SORTSIZE=
- THREADS=
- CLASS statement instead of BY statement
- GROUPFORMAT option
- PERL expressions
- Data step views

SAS Macro made “easy”

What Are Best Practices?

As programmers, you want to perform these tasks as efficiently as possible and optimize the use of the following resources:

- I/O
- CPU
- memory
- data storage space
- network bandwidth
- **programmer time**

Purpose of the Macro Facility

The *macro facility* is a text processing facility for automating and customizing flexible SAS code.

The macro facility supports

- symbolic substitution within SAS code
- automated production of SAS code
- dynamic generation of SAS code
- conditional construction of SAS code.

Macro Terminology

2 components

- macro processor
- macro language

2 delimiters

- macro variable reference (&name)
- macro call (%name)

2 types of macro variables

- automatic
- user defined

Scope of variables

- global
- local

Substitution within a SAS Literal

```
footnote1 "Created 10:24 Wednesday, 25AUG2008";
footnote2 "on the WIN system using Release 9.1";
title "REVENUES FOR DALLAS TRAINING CENTER";
proc tabulate data=perm.all;
   where upcase(location)="DALLAS";
   class course_title;
   var fee;
   table course_title=" " all="TOTALS",
      fee=" "*(n*f=3. sum*f=dollar10.)
      / rts=30 box="COURSE";
run;
```

Substitution within a SAS Literal

Example: Substitute system information in footnotes.

```
footnote1 "Created &systime &sysday, &sysdate9";
footnote2 "on the &sysscp system using Release
&sysver";
title "REVENUES FOR DALLAS TRAINING CENTER";
proc tabulate data=perm.all;
   where upcase(location)="DALLAS";
   class course_title;
   var fee;
   table course_title=" " all="TOTALS",
      fee=" "*(n*f=3. sum*f=dollar10.)
      / rts=30 box="COURSE";
run;
```

Automatic macro variables, which store system information, can be used to avoid hardcoding these values.

Substitution within a SAS Literal

REVENUES FOR DALLAS TRAINING CENTER

COURSE	N	Sum
Artificial Intelligence	25	$10,000
Basic Telecommunications	18	$14,310
Computer Aided Design	19	$30,400
Database Design	23	$8,625
Local Area Networks	24	$15,600
Structured Query Language	24	$27,600
TOTALS	133	$106,535

Created 14:56 Friday, 29AUG2008
on the WIN system using Release 9.1

Substituting User-Defined Information

Example: Include the same value repeatedly throughout a program.

```
proc print data=perm.schedule;
   where year(begin_date)=2004;
   title "Scheduled Classes for 2004";
run;
proc means data=perm.all sum;
   where year(begin_date)=2004;
   class location;
   var fee;
   title "Total Fees for 2004 Classes";
   title2 "by Training Center";
run;
```

What if you have 50 lines of code you need to update?

Substituting User-Defined Information

Example: Include the same value repeatedly throughout a program.

```
%Let yr=2008;
proc print data=perm.schedule;
   where year(begin_date)=&YR;
   title "Scheduled Classes for &YR";
run;
proc means data=perm.all sum;
   where year(begin_date)=&YR;
   class location;
   var fee;
   title "Total Fees for &YR Classes";
   title2 "by Training Center";
run;
```

User-defined macro variables enable you to define a value once, then substitute that value as often as necessary within a program.

Defining a Macro

A *macro* or *macro definition* enables you to write macro programs.

```
%MACRO macro-name;
        macro-text
%MEND <macro-name>;
```

macro-name (follows SAS naming conventions)

macro-text

Can include the following:

- any text
- SAS statements or steps
- macro variables, functions, statements, or calls
- any combination of the above

Generating Data-Dependent Steps

Example: Create a separate data set for each value of a selected variable in a selected data set. Use the variable **location** in **perm.schedule**.

Listing of PERM.SCHEDULE

Obs	Course_ Number	Course_ Code	Location	Begin_ Date	Teacher
1	1	C001	Seattle	26OCT2004	Hallis, Dr. George
2	2	C002	Dallas	07DEC2004	Wickam, Dr. Alice
3	3	C003	Boston	11JAN2005	Forest, Mr. Peter
4	4	C004	Seattle	25JAN2005	Tally, Ms. Julia
5	5	C005	Dallas	01MAR2005	Hallis, Dr. George
6	6	C006	Boston	05APR2005	Berthan, Ms. Judy
7	7	C001	Dallas	24MAY2005	Hallis, Dr. George
8	8	C002	Boston	14JUN2005	Wickam, Dr. Alice
9	9	C003	Seattle	19JUL2005	Forest, Mr. Peter
10	10	C004	Dallas	16AUG2005	Tally, Ms. Julia
11	11	C005	Boston	20SEP2005	Tally, Ms. Julia
12	12	C006	Seattle	04OCT2005	Berthan, Ms. Judy
13	13	C001	Boston	15NOV2005	Hallis, Dr. George
14	14	C002	Seattle	06DEC2005	Wickam, Dr. Alice
15	15	C003	Dallas	10JAN2006	Forest, Mr. Peter
16	16	C004	Boston	24JAN2006	Tally, Ms. Julia
17	17	C005	Seattle	28FEB2006	Hallis, Dr. George
18	18	C006	Dallas	28MAR2006	Berthan, Ms. Judy

Generating Data-Dependent Steps

SAS Program and Log

```
MPRINT(SITES):   data Boston Dallas Seattle ;
MPRINT(SITES):   set perm.schedule;
MPRINT(SITES):   select(location);
MPRINT(SITES):   when("Boston") output Boston;
MPRINT(SITES):   when("Dallas") output Dallas;
MPRINT(SITES):   when("Seattle") output Seattle;
MPRINT(SITES):   otherwise;
MPRINT(SITES):   end;
MPRINT(SITES):   run;

NOTE: There were 18 observations read from the data set PERM.SCHEDULE.
NOTE: The data set WORK.BOSTON has 6 observations and 5 variables.
NOTE: The data set WORK.DALLAS has 6 observations and 5 variables.
NOTE: The data set WORK.SEATTLE has 6 observations and 5 variables.
```

Generating Data-Dependent Steps

SAS Program and Log

```
MPRINT(SITES):   data Boston Dallas Seattle ;
MPRINT(SITES):   set perm.schedule;
MPRINT(SITES):   select(location);
MPRINT(SITES):   when("Boston") output Boston;
MPRINT(SITES):   when("Dallas") output Dallas;
MPRINT(SITES):   when("Seattle") output Seattle;
MPRINT(SITES):   otherwise;
MPRINT(SITES):   end;
MPRINT(SITES):   run;

NOTE: There were 18 observations read from the data set PERM.SCHEDULE.
NOTE: The data set WORK.BOSTON has 6 observations and 5 variables.
NOTE: The data set WORK.DALLAS has 6 observations and 5 variables.
NOTE: The data set WORK.SEATTLE has 6 observations and 5 variables.
```

Generating Data-Dependent Steps

Step1: Store data values in macro variables.

You can copy the current value of a DATA step variable into a macro variable by using the name of a DATA step variable as the second argument to the SYMPUTX routine.

CALL SYMPUTX('*macro-variable*', *DATA-step-variable*);

```
%let month=1;
%let year=2007;
data orders;
   keep order_date order_type quantity total_retail_price;
   set orion.order_fact end=final;
   where year(order_date)=&year and month(order_date)=&month;
   if order_type=3 then Number+1;
   if final then call symputx('num', Number);
run;

proc print data=orders;
   title "Orders for &month-&year";
   footnote "&num Internet Orders";
run;
```

Generating Data-Dependent Steps

Step1: Store data values in macro variables.

```
%macro sites (data=, var=);
  proc sort data=&data(keep=&var)
       out=values nodupkey;
    by &var;
  run;
  data _null_;
    set values end=last;
    call symputx('site'||left(_n_),location);
    if last then call symputx('count',_n_);
  run;
  %put _local_;
```

LOOP 4

continued...

Generating Data-Dependent Steps

Partial SAS log with result of %put _local_;

```
SITES DATA perm.schedule
SITES I
SITES COUNT 3
SITES VAR location
SITES SITE3 Seattle
SITES SITE2 Dallas
SITES SITE1 Boston
```

The **_local_** argument of the **%PUT** statement lists the name and value of macro variables local to the currently executing macro.

Generating Data-Dependent Steps

Step 2: Generate the DATA step, using macro loops for iterative substitution

```
proc print data=orion.year2008;
run;
```

```
proc print data=orion.year2009;
run;
```

```
proc print data=orion.year2010;
run;
```

Iterative Processing

Example : generate the same summary report for each year between 2008 and 2012.

```
%Macro Loop;
      %do i=2008 %to 2012;
        proc print data=orion.year&i;
        run;
      %end;
%Mend;
```

A sas macro program can generate **iterative SAS code** by substituting different values in each iterations**.**

Generating Data-Dependent Steps

Step 2: Generate the DATA step, using macro loops for iterative substitution. Call the macro.

```
data
  %do i=1 %to &count;
    &&site&i
  %end;
  ;
  set &data;
  select(&var);
    %do i=1 %to &count;
      when("&&site&i") output &&site&i;
    %end;
    otherwise;
  end;
run;
%mend sites;
%sites(data=perm.schedule, var=location)
```

Generating Data-Dependent Steps

Partial SAS Log

```
MPRINT(SITES):   data Boston Dallas Seattle ;
MPRINT(SITES):   set perm.schedule;
MPRINT(SITES):   select(location);
MPRINT(SITES):   when("Boston") output Boston;
MPRINT(SITES):   when("Dallas") output Dallas;
MPRINT(SITES):   when("Seattle") output Seattle;
MPRINT(SITES):   otherwise;
MPRINT(SITES):   end;
MPRINT(SITES):   run;

NOTE: There were 18 observations read from the data set PERM.SCHEDULE.
NOTE: The data set WORK.BOSTON has 6 observations and 5 variables.
NOTE: The data set WORK.DALLAS has 6 observations and 5 variables.
NOTE: The data set WORK.SEATTLE has 6 observations and 5 variables.
```

Generating Data-Dependent Code

Use a macro loop to create excel sheets by regions.

```
proc freq data=regions noprint;
table region/ out=tabreg;
run;
Data _null_;
        set tabreg end=eof;
        call symputx('reg'!!left(_n_),region);
        if eof=1 then do;
     call symputx('end',_n_);
    end;
run;
%macro loop;
        %do i=1 %to &end;
           libname test&i excel "c:\temp\&&reg&i...xls";
           Data test&i..&&reg&i;
              set regions;
              where region="&&reg&i";
           run;
           libname test&i clear;
        %end;
%mend;
```

Generating Data-Dependent Code

Use a macro loop to print every data set in the library.

```
%macro printlib(lib=WORK,obs=5);
  %let lib=%upcase(&lib);
  data _null_;
    set sashelp.vstabvw end=final;
    where libname="&lib";
    call symputx('dsn'||left(_n_),memname);
    if final then call symputx('totaldsn',_n_);
  run;
  %do i=1 %to &totaldsn;
    proc print data=&lib..&&dsn&i(obs=&obs);
       title "&lib..&&dsn&i Data Set";
    run;
  %end;
%mend printlib;
%printlib(lib=orion)
```

What's new in SAS Macro

Creating and Deleting Directory Using Macros

```
filename testdir   'c:\saspaper';
%let newdir = %sysfunc(DCREATE(New_SASpaper, c:\));
NOTE: directory needs to be empty for this to work
%let deldir = %sysfunc(FDELETE(testdir));
```

New automatic variables:

&SYSENCODING
&SYSERRORTEXT, &SYSWARNINGTEXT
&SYSHOSTNAME
&SYSLOGAPPLNAME
&SYSTCPIPHOSTNAME

From Macros to the Prompting Framework

What Is the SAS Prompting Framework?

The *SAS Prompting Framework* provides a standard way for passing user selections to the various SAS platform applications.

The prompting framework has the following characteristics:

- provides a consistent user interface across applications
- is customizable to meet the needs of various user input requirements
- creates an interactive mechanism for requesting user input

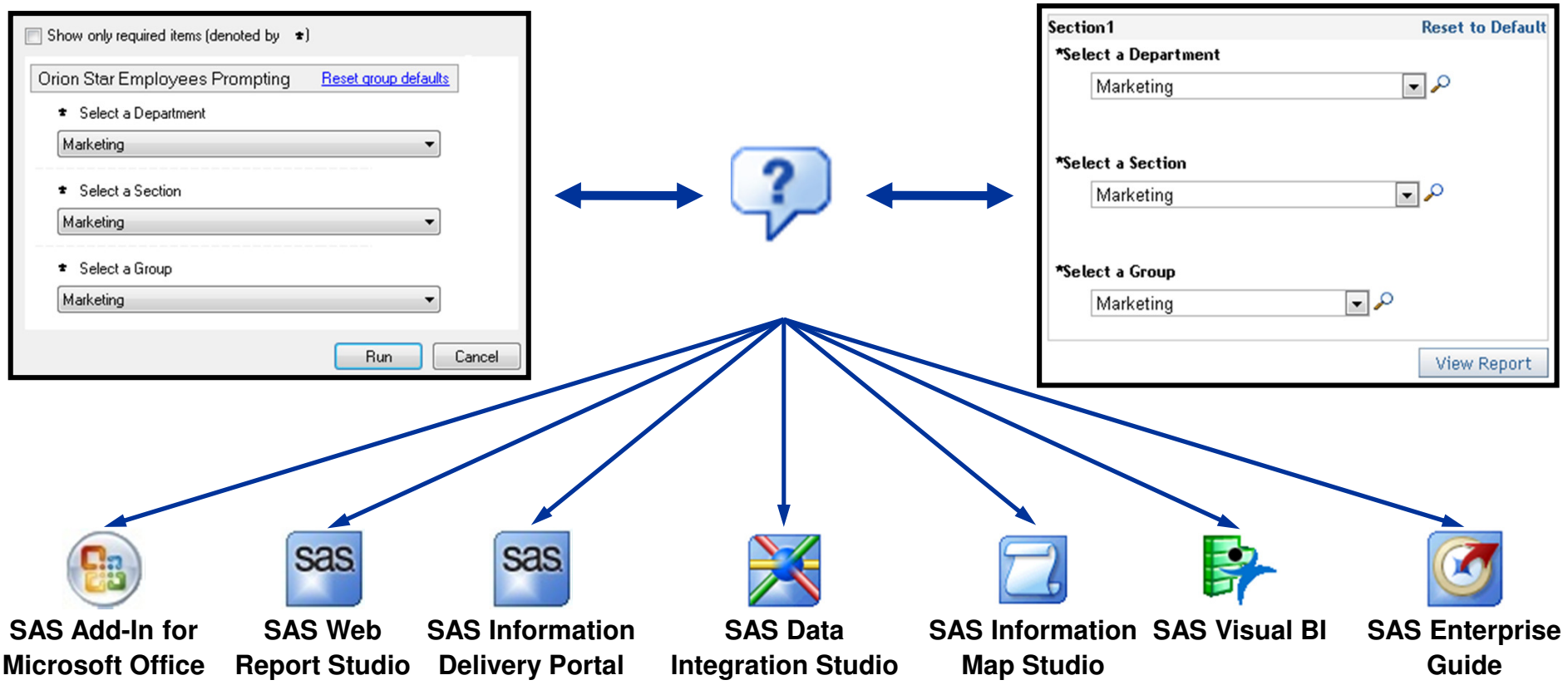
Two "Flavors" of SAS

With SAS®9 there are two different types of SAS installations.

SAS Foundation	The traditional SAS installation, which enables you to write SAS programs or use a point-and-click application such as SAS Enterprise Guide to assist with program creation
Platform for SAS Business Analytics	Enterprise software that utilizes multiple machines throughout the organization and consists of applications that help you accomplish the various tasks for accessing and creating information, as well as creating analysis and reporting

Dynamic Subsetting of Data

The SAS Prompting Framework, which is available from many of the SAS platform applications, provides a common interface for requesting user input.



Prompt Types

The SAS Prompting Framework enables you to create many different types of prompts, including the following:

- color
- data source
- data source item
- date, time, timestamp
- hyperlink
- library
- numeric
- text
- variable

Because there are different types of SAS platform applications, the way that the prompting framework displays information is slightly different between desktop applications and Web applications.

Prompt Categories

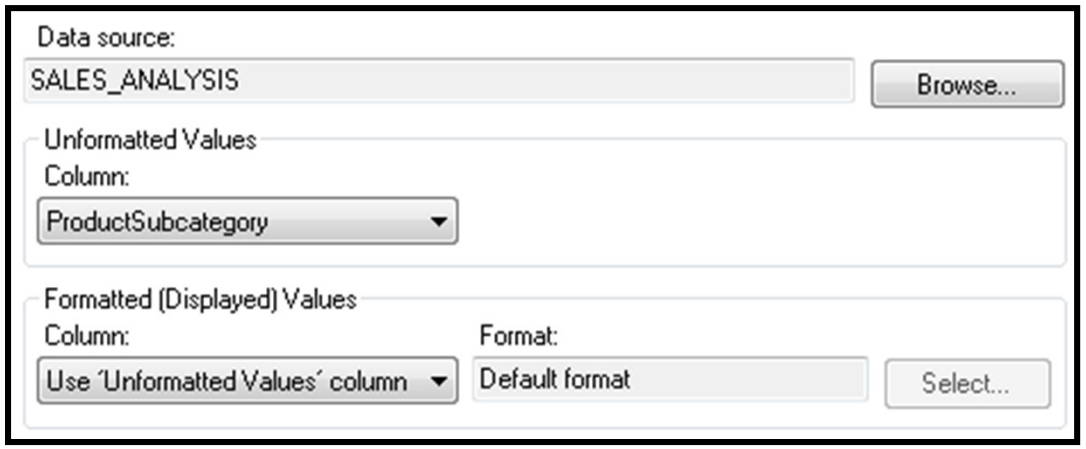
In addition to the different types of prompts provided by the SAS Prompting Framework, several categories of prompts provide additional functionality, including the following:

- dynamic prompts
- cascading prompts
- relative date/time prompts
- range prompts

Dynamic Prompts

Dynamic prompts populate a list of possible values from a dynamic data source. The list is generated at run time rather than at design time.

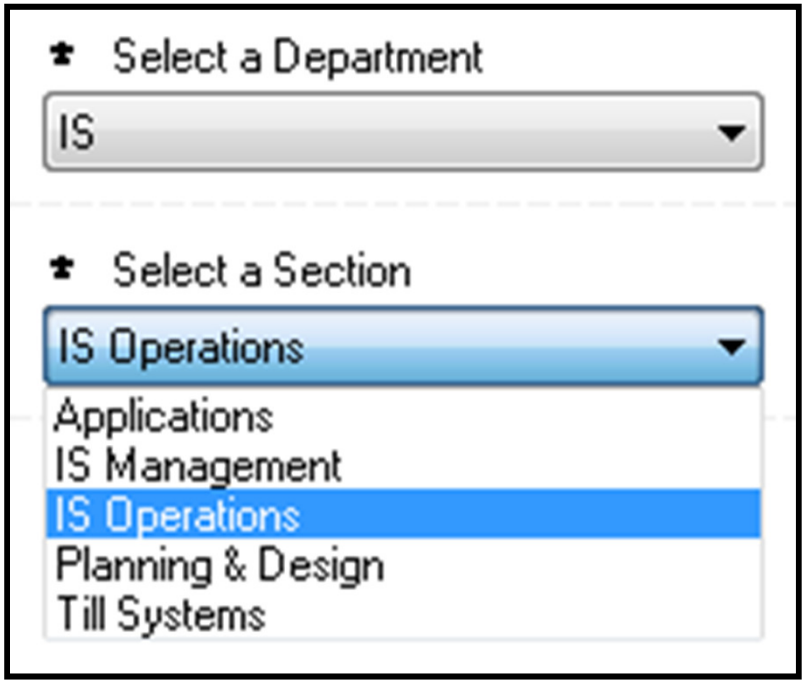
Depending on the SAS application where the prompt is built, the data source can be a physical table or an information map based on relational tables.

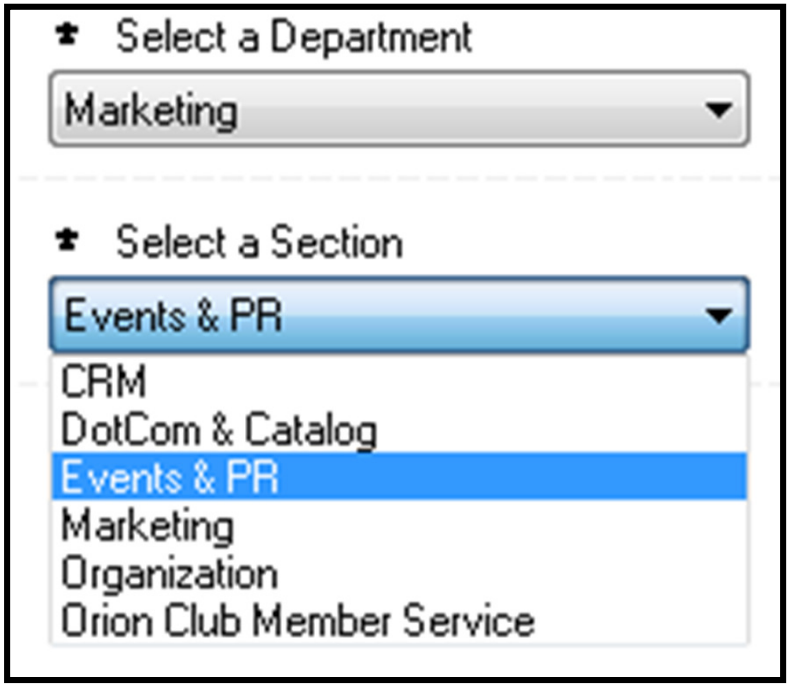


Cascading Prompts

Cascading prompts populate prompt values based on selections in other prompts.

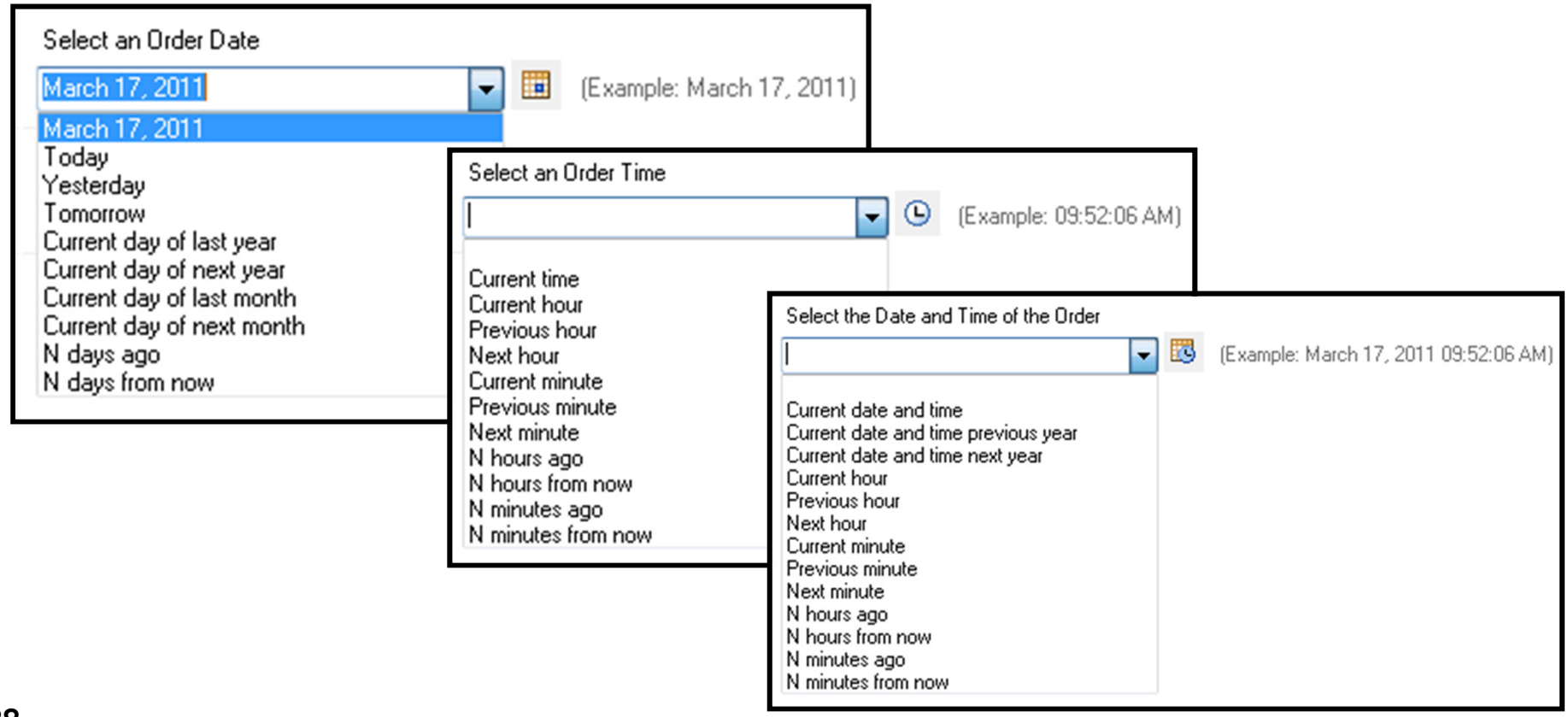
Example: When a department is selected, the list of sections is dynamically generated based on the selected department.





Relative Date/Time Prompts

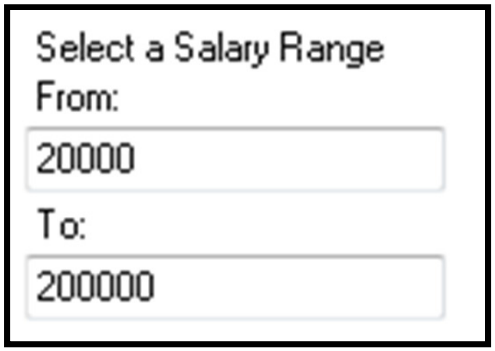
In addition to being able to specify an exact date or time (or both) for a prompt value, relative date/time prompts enable you to incorporate relative time frames into prompting.

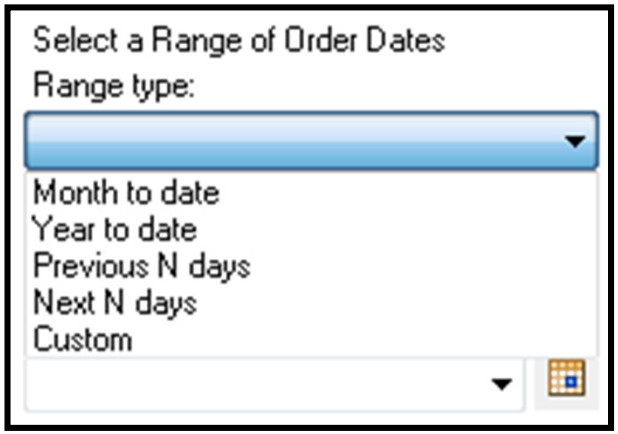


Range Prompts

Range prompts enable users to enter ranges of values, such as minimum and maximum, in one combined prompt.

Range prompts provide the user with one question to answer instead of two and ensure that both values are entered.





Date/time range prompts enable you to select from different range types.

Shared Prompts

In addition to prompts being defined for a specific use, prompts can be shared.

Shared prompts are stored in metadata and can be used in multiple applications.

The benefits of shared prompts include the following:

- a single point of maintenance
- the ability to create prompts with complex configurations one time
- sharing prompt functionality across multiple applications within the organization

What Is a SAS Stored Process?

A *SAS Stored Process* is a special type of SAS program.

Stored processes enable you to run a SAS program and view the results in many different types of SAS applications.

Stored processes consist of a SAS program file along with a metadata definition that describes how the stored process should execute.

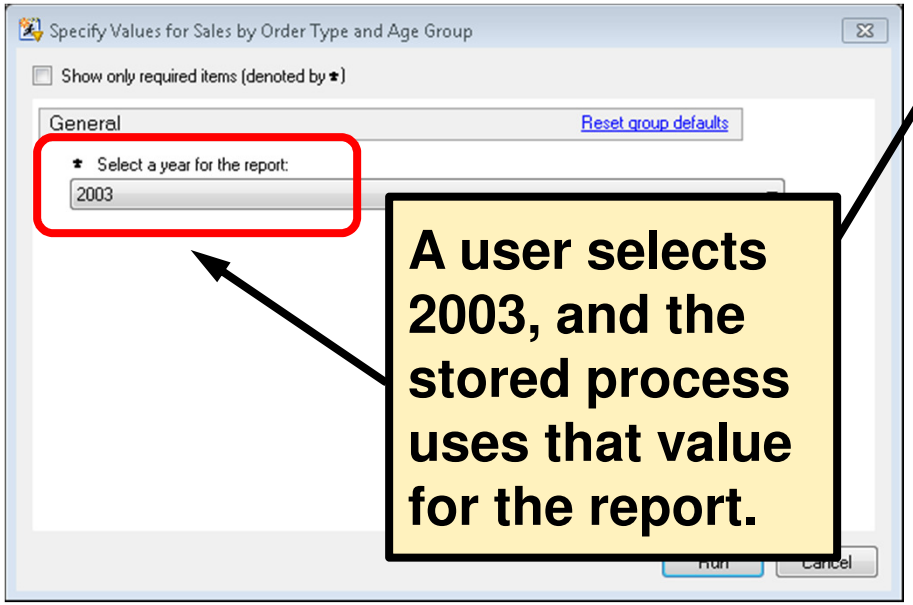
Advantages of Stored Processes

SAS Stored Processes have several advantages over traditional SAS programs.

- Stored processes can prompt users for input through parameters. This allows for code that is not static and can be easily run with different values.
- Because stored process code is not embedded into client applications, there is only one copy of the code to maintain.
- Every application that runs a stored process always gets the latest version of the results.
- Stored process programs use security to ensure that each user has access only to the information that he or she is allowed to see.

Running a SAS Stored Process

A stored process without parameters will execute and immediately return results to the requesting client application. However, if a stored process is defined with parameters, the user is prompted to select parameter values. The stored process uses those values, as coded in the stored process program, and results are then displayed.

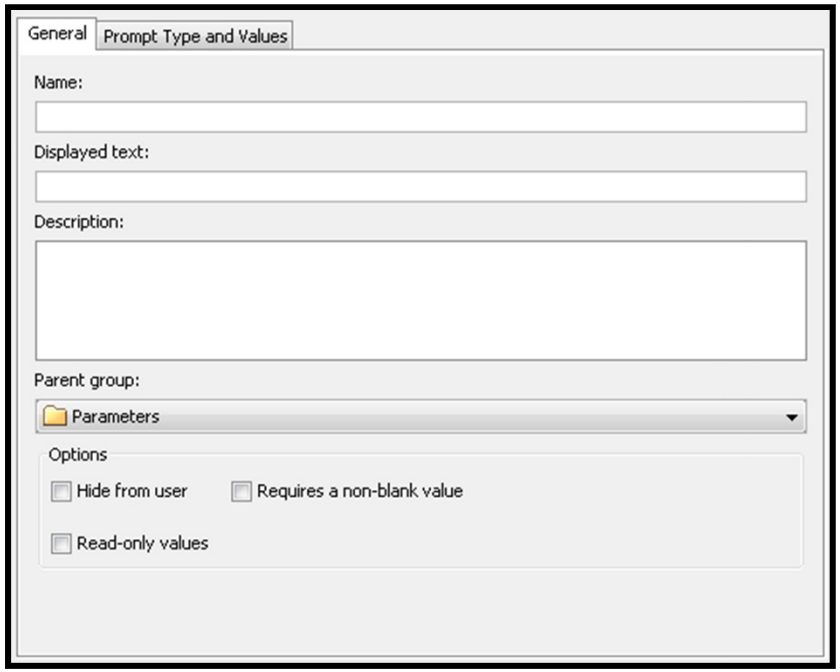


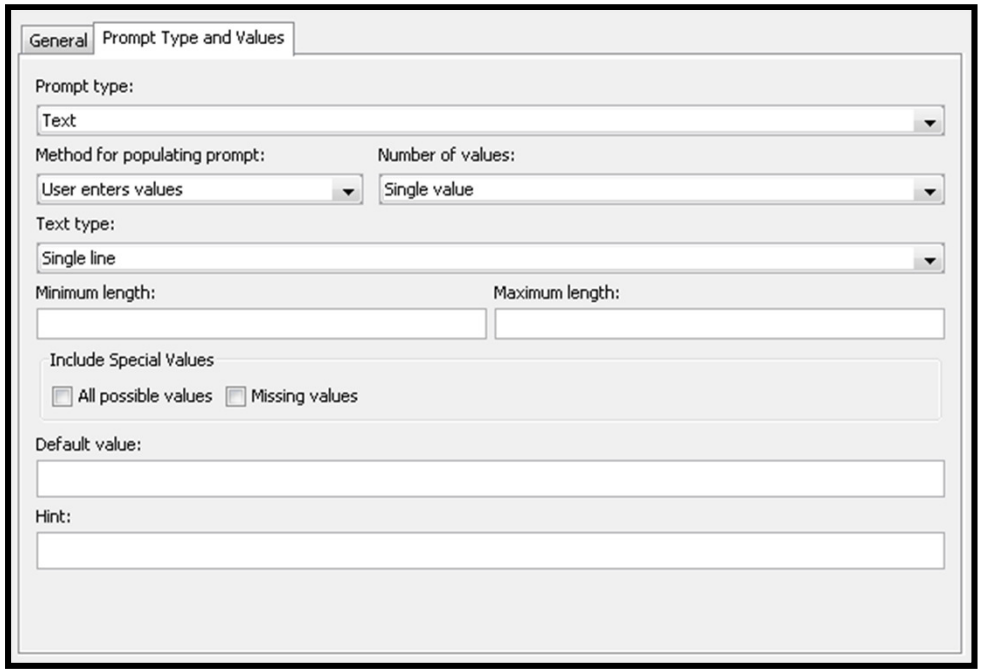
2003 Sales by Order Type and Age Group

Order Type	Age Group	Quantity	Total Sales
Catalog Sale	15-30 years	13,850	$1,238,474
	31-45 years	10,479	$907,574
	46-60 years	10,392	$899,263
	61-75 years	10,052	$860,035
Internet Sale	15-30 years	11,138	$993,449
	31-45 years	8,472	$767,835
	46-60 years	7,999	$691,286
	61-75 years	457	$46,098
Retail Sale	15-30 years	63,267	$5,195,396
	31-45 years	48,661	$4,047,945
	46-60 years	45,837	$3,740,596
	61-75 years	30,555	$2,461,834

Creating Prompts

When creating prompts, you begin by specifying general information. You then define the prompt type and specify how the prompt values are populated.

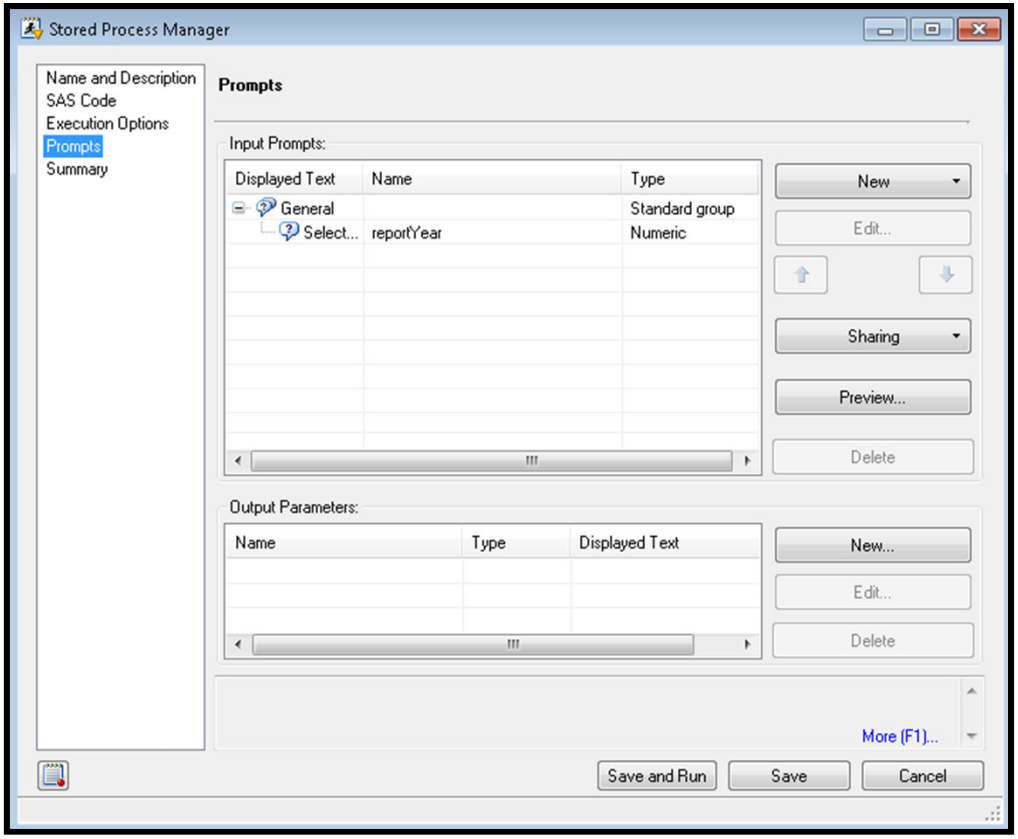




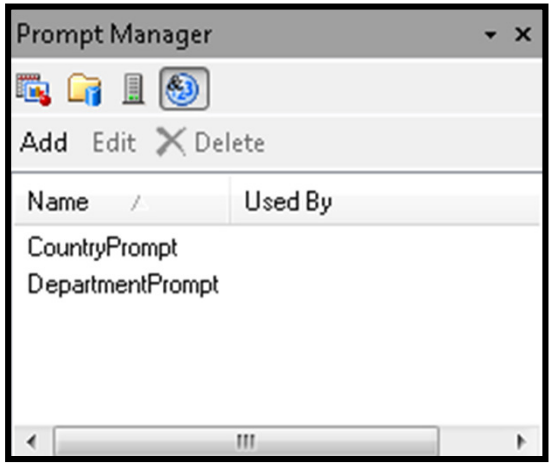
Creating Prompts in SAS Enterprise Guide and SAS Stored Processes

SAS Enterprise Guide enables you to create prompts for the project as well as prompts for stored processes.

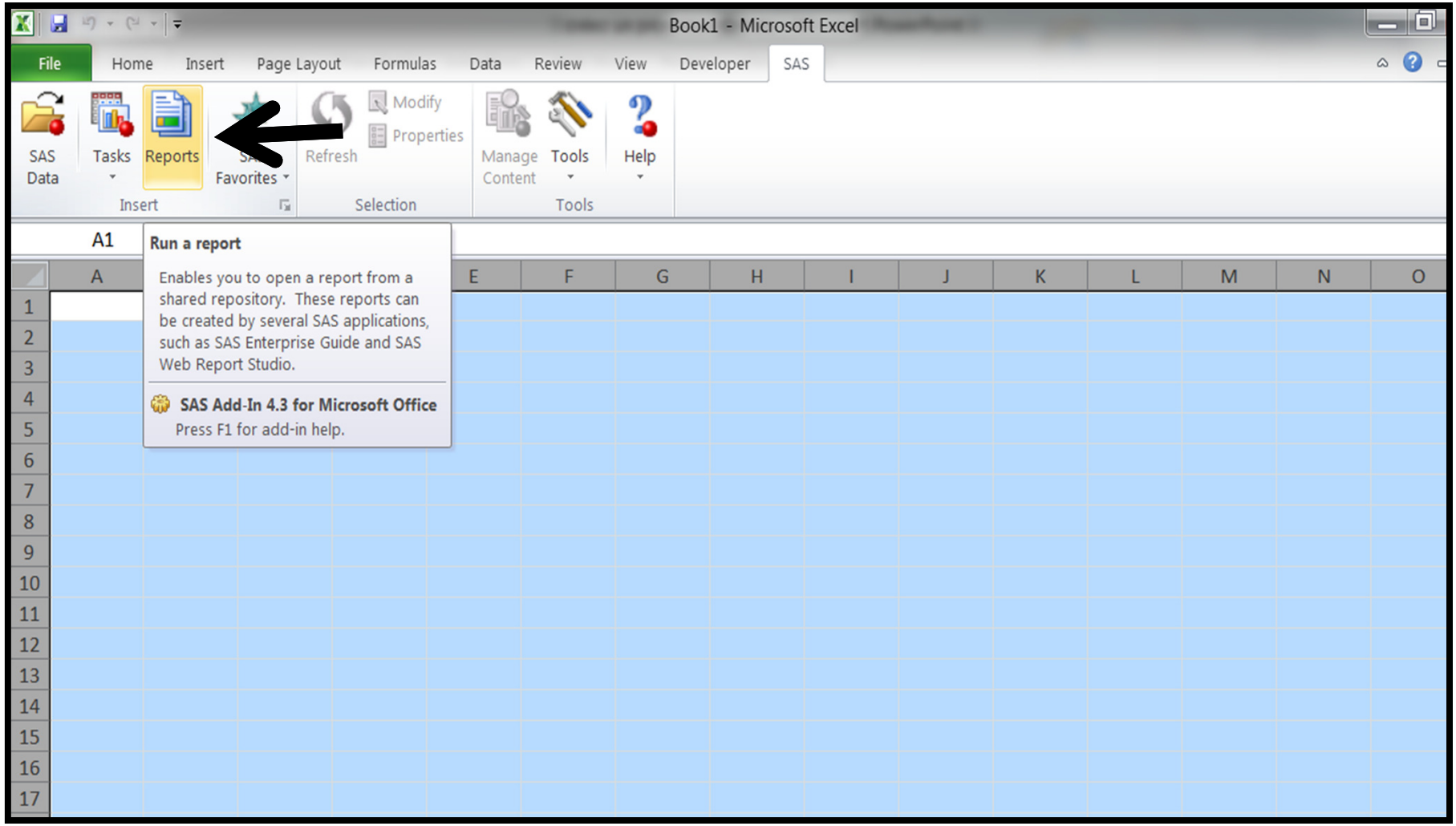
The Prompts selection in the Stored Process Manager enables you to create stored process prompts.



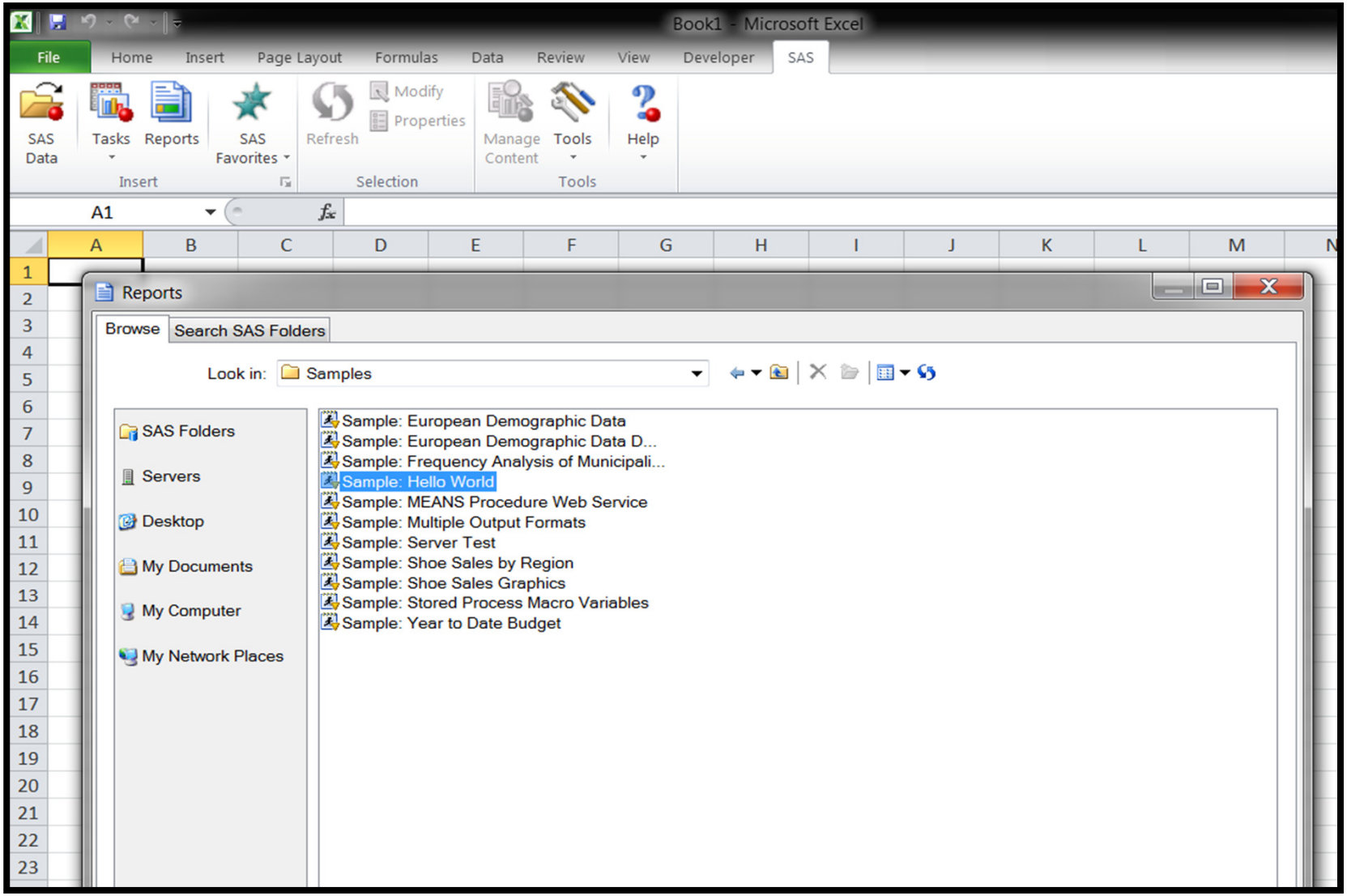
The Prompt Manager enables you to create project prompts.



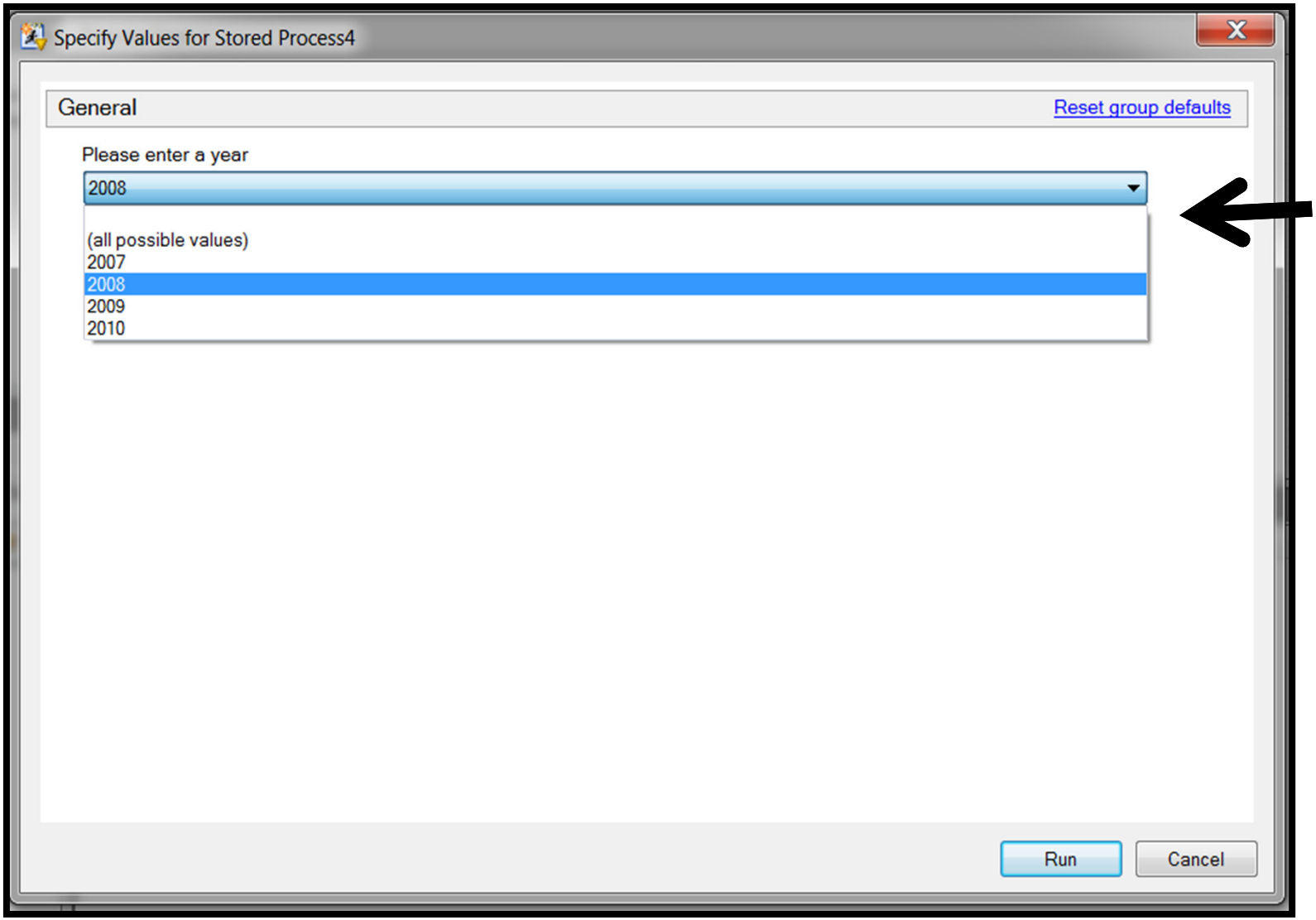
SAS Add-In for Microsoft Office



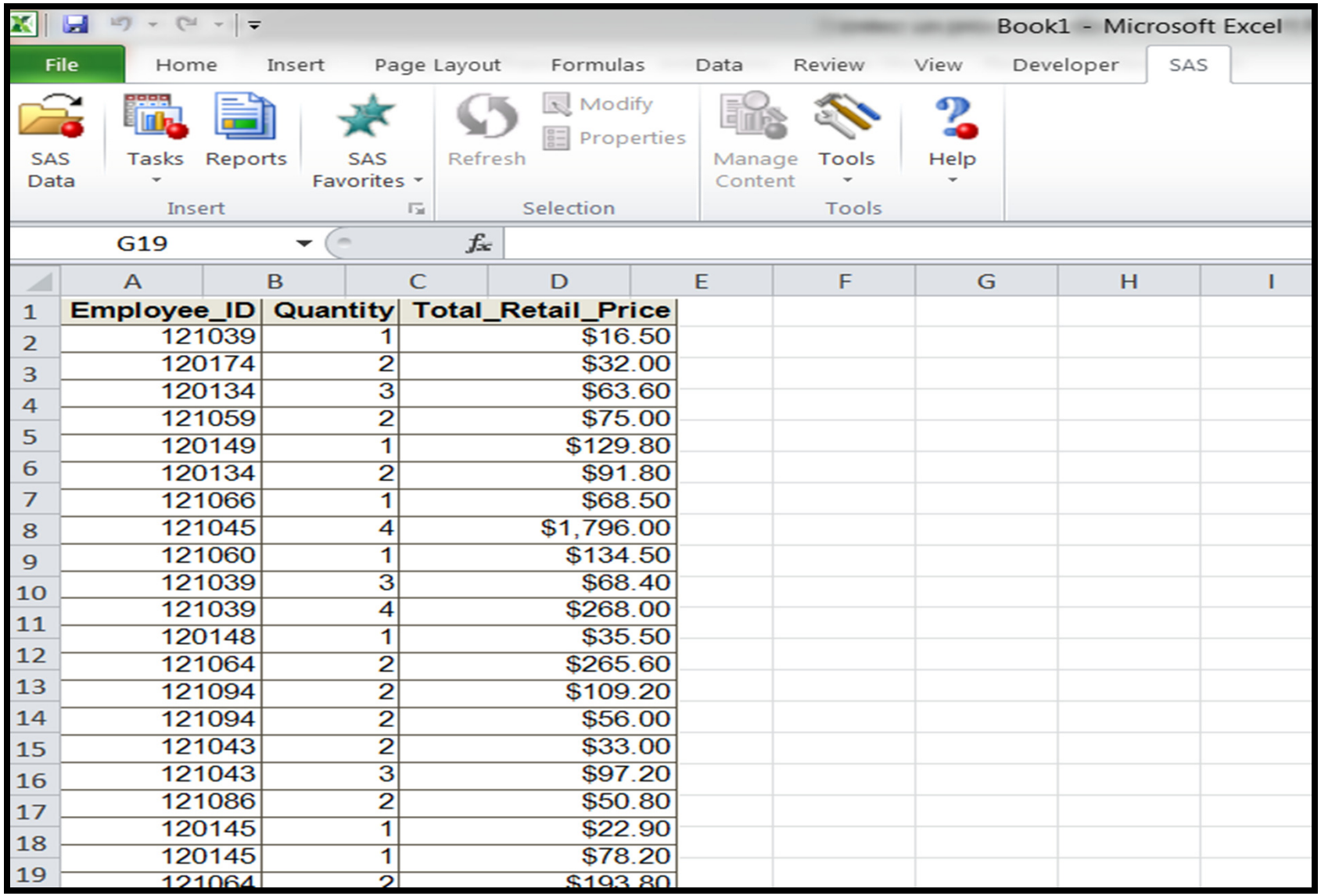
SAS Add-In for Microsoft Office



SAS/Stored Process



SAS/Stored Process



	A	B	C	D	E	F	G	H	I
1	Employee_ID	Quantity	Total_Retail_Price						
2	121039	1	$16.50						
3	120174	2	$32.00						
4	120134	3	$63.60						
5	121059	2	$75.00						
6	120149	1	$129.80						
7	120134	2	$91.80						
8	121066	1	$68.50						
9	121045	4	$1,796.00						
10	121060	1	$134.50						
11	121039	3	$68.40						
12	121039	4	$268.00						
13	120148	1	$35.50						
14	121064	2	$265.60						
15	121094	2	$109.20						
16	121094	2	$56.00						
17	121043	2	$33.00						
18	121043	3	$97.20						
19	121086	2	$50.80						
20	120145	1	$22.90						
21	120145	1	$78.20						
22	121064	2	$193.80						

To learn more

- **Courses:**

  Programming 3:

  Advanced Techniques and Efficiences

  SAS Macro Language 1: Essentials

- **www.sas.com/canada**

  For documentation, papers and examples

Questions?

Thank you for attending!

josee.ranger-lacroix@sas.com