Paper 040-2009

You Need to Consolidate Massive Amounts of Data? Select the Fastest Method or Go for Minimal Memory Requirements to Build Your Result

Henri Theuwissen, BI Knowledge Sharing, Bierbeek, Belgium

ABSTRACT

Most applications require consolidated data. With the ever growing volumes of data, CPU time becomes a bottleneck, and performance becomes vital. The most common way to consolidate data is the MEANS or SUMMARY procedure; database addicts prefer the Structured Query Language (SQL) procedure. These procedures, although very powerful and easy, sometimes require too much CPU time or cause out of memory errors.

Learn alternative solutions and get details on options to improve the required resources. See a comparison between several methods for data consolidation:

- The MEANS and SUMMARY procedure
- The SQL procedure
- DATA step processing
- HASH tables

Examine the impact of sorted or indexed data or the usage of system options like SUMSIZE, MEMSIZE, and THREADS.

Learn details of the MEANS and SUMMARY procedures:

- Using the WAYS or TYPES instructions, with a positive impact on memory
- Working with the ID statement, and the impact on CPU time; restrictions for the ID statement
- Working with the BY statement, and the impact on CPU time.

INTRODUCTION

With ever increasing volumes of data in Business Intelligence and Data Warehousing applications, performance can become an issue. Summarizing data might hit the barriers of CPU time limits, space requirements or memory requirements.

This paper illustrates various approaches to get the fastest result in aggregating data, within the limits of the available memory. Performance statistics are collected using the FULLSTIMER option.

Description of FULLSTIMER Statistics

Statistic	Description
Real Time	(Elapsed) Time spent to process the SAS step
User CPU Time	CPU time spent to execute the SAS code
System CPU Time	CPU time spent to perform operating system overhead tasks. These tasks support the execution of SAS code
Memory	Memory required to run the step

SAS offers almost always several possibilities to provide a solution for a reporting or analysis request. Although there are several general guidelines, there is not a single rule that suggests using always the same approach for a specific problem. For production jobs it is important to test several alternatives and select the best solution, based on your CPU time, space requirements or memory limits. The best solution usually is the result of several parameters, both system related parameters and data related parameters. Items to take into account include: RAM size, number of observations in the data set, number of classification variables and cardinality of the data.

The examples discussed in this paper are executed on data sets with different sizes, to illustrate the impact of the volume of records. The table below shows the structure of the data sets and the number of distinct values for each variable.

Description of Source Table

Name	Type	Length	Cardinality
Year_Month	Char	7	12
Distributor	Char	20	48
Area	Char	25	1.000
City	Char	30	10.000
Street	Char	30	100.000
Client	Char	30	1.000.000
Network	Char	10	30
Installation	Char	10	800.000
Consumption	Num	8	N/A

SUMMARIZING DATA

METHODS

Four methods to aggregate data are analyzed. No pre-processing is executed on the data sets. The data are not sorted. Resource requirements are compared. The four methods are:

1. The SUMMARY and MEANS procedure. Summarization is executed using a CLASS statement and a VAR statement. The system variables _TYPE_ and _FREQ_ are removed from the output. Only the N-way summarization is created.
2. The SQL procedure. Summarization is executed using the SUM function and the GROUP BY clause.
3. The DATA step with a HASH table. At the beginning of the execution of the DATA step, a hash table is created, with a parameter indicating that the data will be sorted when written to an output data set. The key variables for the hash table, specified in the DEFINEKEY method are the variables that are specified in the CLASS statement in the PROC SUMMARY. Since these variables must also exist in the output data set, they are also specified in the DEFINEDATA method. A new variable is added in the DEFINEDATA method, containing the total value of the VAR variable from the PROC SUMMARY. When executing the DATA step, a new item is added to the hash table if the key combination of the processed input record does not yet exist in the hash table. When the combination already exists, the numeric value of the input record is added to the existing item from the hash table. After having processed all input records, the content of the hash table is written to the output data set.
4. The SORT procedure, followed by a DATA step, using a RETAIN statement. The data set is first sorted by the classification variables. Then a DATA step is executed with BY processing, and the FIRST. and LAST. options, building total values using a RETAIN statement. For each new combination of the classification values, the total value is initialized to 0. After processing the last record of a combination of the classification values, a record is written to the output data set.

TEST RESULTS

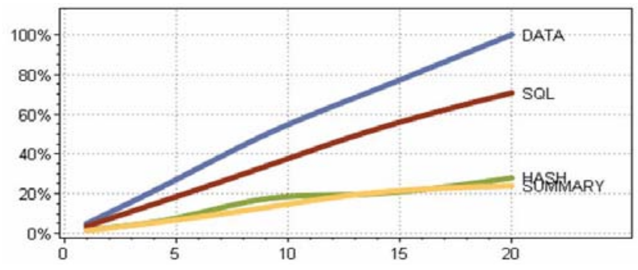
For each of these methods, several tests are executed, with different numbers of classification variables and with different number of records. Each test is executed on a table with 1.000.000, 5.000.000, 9.000.000, 13.000.000 and 20.000.000 records.

The first tests are executed on two classification variables: “Distributor” and “Year_Month”. “Distributor” has 48 unique values, “Year_Month” has 12 unique values. The following code is executed.

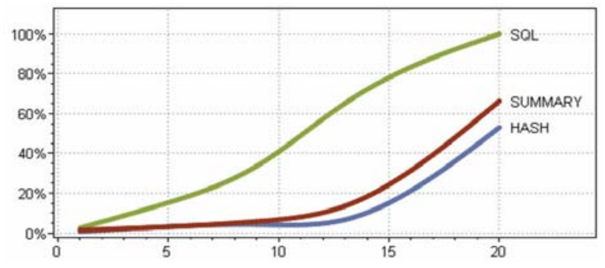
```
/* Traditional SUMMARY procedure                      */
PROC SUMMARY DATA = SGF.SGF_BASE NWAY MISSING;
   CLASS  DISTRIBUTOR YEAR_MONTH;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUMMARY (DROP = _TYPE_ _FREQ_) SUM = TOTAL_CONSUMPTION;
RUN;


/* Summarizing using the SQL procedure                */
PROC SQL;
   CREATE TABLE WORK.SQL AS
   SELECT DISTRIBUTOR,
          YEAR_MONTH,
          SUM(CONSUMPTION) AS TOTAL_CONSUMPTION
   FROM   SGF.SGF_BASE
   GROUP  BY DISTRIBUTOR, YEAR_MONTH;
QUIT;


/* Creating and building a Hash table in a DATA step  */
DATA _NULL_;
   LENGTH TOTAL_CONSUMPTION 8;
   SET SGF.SGF_BASE END = EOF;
   IF _N_ = 1 THEN DO;
      DECLARE HASH HS(HASHEXP:8, ORDERED:'A');
      HS.DEFINEKEY  ('DISTRIBUTOR', 'YEAR_MONTH');
      HS.DEFINEDATA ('DISTRIBUTOR', 'YEAR_MONTH', 'TOTAL_CONSUMPTION');
      HS.DEFINEDONE ();
   END;
   RC = HS.FIND();
   IF RC NE 0 THEN TOTAL_CONSUMPTION = 0;
   TOTAL_CONSUMPTION + CONSUMPTION;
   HS.REPLACE();
   IF EOF THEN RC = HS.OUTPUT(DATASET:'WORK.HASH');
RUN;


/* Sorting the source table and processing the table  */
/* in a DATA step with RETAIN statement               */
PROC SORT DATA = SGF.SGF_BASE
           OUT = WORK.SGF;
   BY DISTRIBUTOR YEAR_MONTH;
RUN;
DATA WORK.DATA;
   SET WORK.SGF;
   BY DISTRIBUTOR YEAR_MONTH;
   RETAIN TOTAL_CONSUMPTION 0;
   IF FIRST.YEAR_MONTH THEN TOTAL_CONSUMPTION = 0;
   TOTAL_CONSUMPTION + CONSUMPTION;
   IF LAST.YEAR_MONTH;
RUN;
```

The graphs show the resource requirements for each method. Instead of showing absolute values for CPU seconds the graphs consider the highest value (DATA step with preceding SORT procedure, for 20 Mio records in the first graph) as 100% and all other values are presented as the ratio to that value. This concept is used for all further graphs in this paper.

Total CPU Time



Real Time



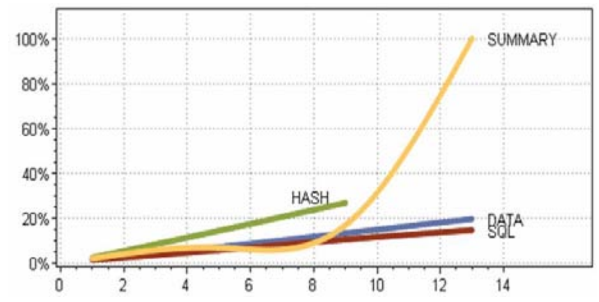
- In CPU time, the SUMMARY procedure and the usage of a hash table are comparable; the CPU time increase for the SQL procedure has a higher slope than the curve for the SUMMARY procedure. The DATA step with preceding sort consumes a lot of CPU time for the SORT step.
- The real time for hash tables is better than for the SUMMARY procedure, mainly due to les I/O.
- The graph for real time does not contain information for the DATA step with preceding SORT, due to the extreme values. By omitting this solution, the differences between the other three methods is better illustrated.

In a second test, the same four methods are executed, on the same data sets, but with four classification variables. “Distributor” has 48 unique values, “Year_Month” has 12 unique values, “Area” has 1.000 unique values and “City” has 10.000 unique values.

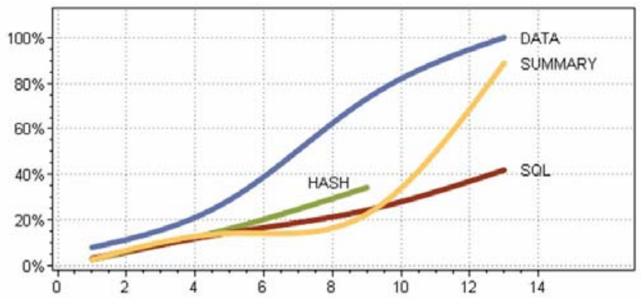
The source code for the SUMMARY procedure is shown below.

```
PROC SUMMARY DATA = SGF.SGF_BASE NWAY MISSING;
   CLASS  DISTRIBUTOR YEAR_MONTH AREA CITY;
   VAR    CONSUMPTION ;
   OUTPUT OUT = WORK.SUMMARY (DROP = _TYPE_ _FREQ_) SUM = TOTAL_CONSUMPTION;
RUN;
```

Total CPU Time



Real Time



- The summarization with the hash table was not executed for 13.000.000 records, due to memory problems. The step aborted with the error message:

```
FATAL: Insufficient memory to execute data step program. Aborted during the
       EXECUTION phase.
```

- The SUMMARY procedure has an exponential increase in CPU time. This can be explained by the following execution note:

```
Note: Processing on disk occurred during summarization. Peak disk usage was
      approximately 1374 Mbytes. Adjusting SUMSIZE may improve performance.
```

- Using a separate disk for WORK files can improve the real time, because not all I/O processing is executed on the same disk.

SUMMARIZING SORTED DATA

Since the result of summarizing data is always sorted on the classification variables, the next section examines the impact of sorted input.

PROC SUMMARY ON SORTED OR INDEXED DATA

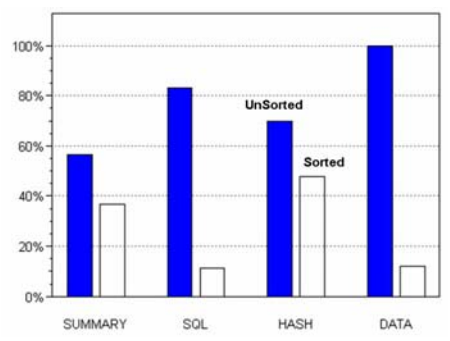
The four summarization methods, discussed in the first topic are used to consolidate data on three classification variables: “Distributor”, “Year_Month” and “Area”. The summarization is executed on the input source data set with a random sequence of the classification values and on a sorted input data set. The tests were executed on 5.000.000 records and on 13.000.000 records.

```
/* Traditional SUMMARY procedure                      */
PROC SUMMARY DATA = SGF.SGF_BASE NWAY MISSING;
   CLASS  DISTRIBUTOR YEAR_MONTH AREA;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUMMARY (DROP = _TYPE_ _FREQ_) SUM = TOTAL_CONSUMPTION;
RUN;


/* Summarizing using the SQL procedure                */
PROC SQL;
   CREATE TABLE WORK.SQL AS
   SELECT DISTRIBUTOR,
          YEAR_MONTH,
          AREA,
          SUM(CONSUMPTION) AS TOTAL_CONSUMPTION
   FROM   SGF.SGF_BASE
   GROUP BY DISTRIBUTOR, YEAR_MONTH, AREA;
QUIT;


/* Creating and building a Hash table in a DATA step  */
DATA _NULL_;
   LENGTH TOTAL_CONSUMPTION 8;
   SET SGF.SGF_BASE END = EOF;
   IF _N_ = 1 THEN DO;
      DECLARE HASH HS(HASHEXP:8, ORDERED:'A');
      HS.DEFINEKEY  ('DISTRIBUTOR', 'YEAR_MONTH', 'AREA');
      HS.DEFINEDATA ('DISTRIBUTOR', 'YEAR_MONTH', 'AREA',
                     'TOTAL_CONSUMPTION');
      HS.DEFINEDONE ();
   END;
   RC = HS.FIND();
   IF RC NE 0 THEN TOTAL_CONSUMPTION = 0;
   TOTAL_CONSUMPTION + CONSUMPTION;
   HS.REPLACE();
   IF EOF THEN RC = HS.OUTPUT(DATASET:'WORK.HASH');
```

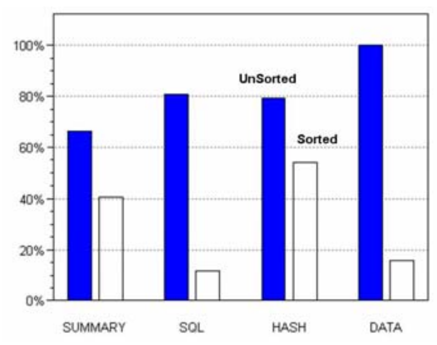
```
RUN;


/* DATA step with BY and RETAIN statement              */
DATA WORK.DATA;
   SET WORK.SGF;
   BY DISTRIBUTOR YEAR_MONTH AREA;
   RETAIN TOTAL_CONSUMPTION 0;
   IF FIRST.AREA THEN TOTAL_CONSUMPTION = 0;
   TOTAL_CONSUMPTION + CONSUMPTION;
   IF LAST.AREA;
RUN;
```

The bar charts compare the CPU time requirements of the sorted versus the unsorted input data sets.

CPU Time – 5.000.000 records



CPU Time – 13.000.000 records



- The SQL procedure and the solution with DATA step with RETAIN statement benefit the most from sorted input data.
- Comparing the CPU time between the 2 data sets shows that for the SQL procedure the CPU time almost doubles, whereas for the other solutions the increase is 150% to 200%. (Note: this is not illustrated in the bar charts).

SORTING DATA BEFORE PROC SUMMARY

Given the benefit of having a sorted or indexed input data set for a summarization task, the next topic examines whether it is advisable to precede the SUMMARY procedure by a SORT procedure. The test is executed on a data set with 2.000.000 records, with four classification variables.

```
/* Unsorted */
PROC SUMMARY DATA = SGF.SGF_BASE NWAY MISSING;
   CLASS  AREA YEAR_MONTH  DISTRIBUTOR CLIENT ;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUM1 (DROP = _TYPE_ _FREQ_) SUM =;
RUN;


/* Sorted */
PROC SORT DATA = SGF.SGF_BASE OUT = WORK.SGFID;
   BY AREA YEAR_MONTH DISTRIBUTOR CLIENT;
RUN;
PROC SUMMARY DATA = WORK.SGFID NWAY MISSING;
```

```
   CLASS  AREA YEAR_MONTH DISTRIBUTOR CLIENT;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUM2 (DROP = _TYPE_ _FREQ_) SUM = ;
RUN;
```

Resource Requirements

	Unsorted	Sorted		
	Summary	Sort	Summary	Total
Memory	206,162.00	302,764.00	206,158.00	302,764.00
Real Time	22.87	33.04	13.38	46.42
System CPU	1.15	1.42	0.87	2.29
User CPU	14.90	8.39	9.07	17.46

- For the memory requirements, for the second method (with preceding SORT procedure) the maximum value of both steps is taken as total requirement.
- The resource statistics show that the SUMMARY procedure is faster on a sorted input data set, but the gain is not high enough to justify a preceding PROC SORT.

BY STATEMENT IN A PROC SUMMARY

When summarizing a sorted input data set, important resource improvements are achieved with BY-group processing. Instead of specifying all classification variables in a CLASS statement, some of the variables can be moved to a BY statement. As a result, summarization will be executed within each BY-group value, instead of within the full data set.

To analyze the impact of the BY statement, an input data set with 5.000.000 records is consolidated on four classification variables. The data set is sorted on the first classification variable (“Area”). First a SUMMARY procedure is executed with all four classification variables in the CLASS statement, then PROC SUMMARY is executed with one variable in a BY statement. This variable (“Area”) has 1.000 distinct values.

Afterwards, the same test is executed on a data set with 13.000.000 records.

```
/* Sorted      */
PROC SUMMARY DATA = WORK.SGFID NWAY MISSING;
   CLASS  AREA YEAR_MONTH DISTRIBUTOR CLIENT;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.CLASS (DROP = _TYPE_ _FREQ_) SUM = ;
RUN;


/* Sorted + BY */
PROC SUMMARY DATA = WORK.SGFID NWAY MISSING;
   BY     AREA;
   CLASS  YEAR_MONTH DISTRIBUTOR CLIENT;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.BY (DROP = _TYPE_ _FREQ_) SUM = ;
RUN;
```

Resource Requirements

	13 M		5 M	
	CLASS	BY	CLASS	BY
Memory	984,453	2,055	453,497	930
Real Time	526	115	154	74
System CPU	33	7	4	2
User CPU	93	49	32	23

- The resource statistics show that the BY statement has an important performance improvement, both on CPU requirements and on memory usage.
- For the test on 13.000.000 records, the difference is even higher, due to the fact that processing occurred on disk during summarization.

```
Note: Processing on disk occurred during summarization. Peak disk usage was
      approximately 1374 Mbytes. Adjusting SUMSIZE may improve performance.
```

COMBINATIONS OF CLASSIFICATION VARIABLES

The SUMMARY procedure allows creating the N-way summary and subtotals for combinations of classification variables. There are several ways to build these subtotals:

- Execute the SUMMARY procedure without the NWAY option in the PROC statement, and subset the result to the required combinations of the classification variables by using a WHERE option on the values of the _TYPE_ variable. Notice that the subset is already created in the OUTPUT statement of the PROC SUMMARY and not in an additional DATA step.
- Use the WAYS or TYPES statement in the SUMMARY procedure to select the combinations of classification variables that are required.

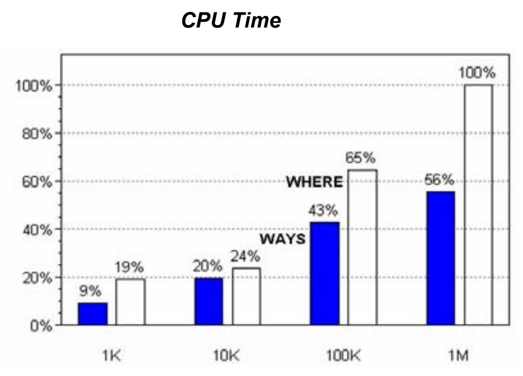
Both methods are analyzed using the following code: the total value of the VAR variable is created for each combination of the values of the four classification variables (N-way summary) and also subtotals are calculated for the values of each individual classification variable.

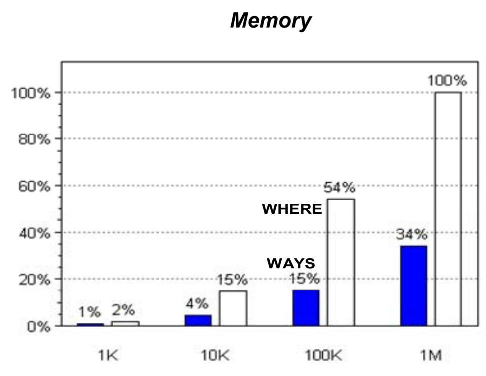
```
PROC SUMMARY DATA = SGF.SGF_BASE MISSING;
   CLASS  DISTRIBUTOR YEAR_MONTH NETWORK CLIENT;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUM_WHERE (WHERE = (_TYPE_ IN (1, 2, 4, 8, 15))) SUM = ;
RUN;

PROC SUMMARY DATA = SGF.SGF_BASE MISSING;
   CLASS  DISTRIBUTOR YEAR_MONTH NETWORK CLIENT;
   VAR    CONSUMPTION;
   WAYS   1, 4;
   OUTPUT OUT = WORK.SUM_WAYS (DROP = _FREQ_ ) SUM = ;
RUN;
```

- The WAYS 1, 4; statement requests the consolidation for all combinations of one classification variable and for all combinations of four classification variables.
- The same result can be created using a TYPES statement instead of a WAYS statement.

```
   TYPES DISTRIBUTOR
         YEAR_MONTH
         NETWORK
         CLIENT
         DISTRIBUTOR * YEAR_MONTH * NETWORK * CLIENT;
```

All tests were executed on a data set containing 1.000.000 records. Different tests were executed, with 1.000, 10.000, 100.000 and 1.000.000 different values for the variable Client. The graphs below show CPU time and Memory requirements. Notice that for both resources, the solution with the WAYS statement gives superior results compared to the subsetting WHERE statement.





A final test was executed with a 2-pass processing: first the N-way summary is calculated and the output data set of this PROC SUMMARY is used as input for a second PROC SUMMARY, to calculate the subtotals. Both output data sets are concatenated with the APPEND procedure

```
PROC SUMMARY DATA = SGF.SGF_BASE MISSING NWAY;
   CLASS  DISTRIBUTOR YEAR_MONTH NETWORK CLIENT;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUM_NWAY (WHERE = (_TYPE_ IN (1,2,4,8,15))) SUM = ;
RUN;
PROC SUMMARY DATA = WORK.SUM_NWAY MISSING;
   CLASS  DISTRIBUTOR YEAR_MONTH NETWORK CLIENT;
   WAYS   1;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUM_DTLS SUM = ;
RUN;
PROC APPEND BASE = WORK.SUM_NWAY DATA = WORK.SUM_DTLS FORCE;
RUN;
```

- The results are not better than any of the first two tests. Resources requirements are higher than both methods running only one PROC SUMMARY. The main reason is that using the last method requires processing the data twice.

Consider a request to consolidate data to create three different combinations of classification variables. Such a request is often treated in three different PROC steps, as illustrated below.

```
PROC SUMMARY DATA = SGF.SGF_1M_WAYS NWAY MISSING CHARTYPE;
   CLASS  COUNTRY DISTRIBUTOR;
   VAR    CONSUMPTION;
   OUTPUT OUT = CONSOLIDATED1 (DROP = _FREQ_ _TYPE_) SUM =;
RUN;
PROC SUMMARY DATA = SGF.SGF_1M_WAYS MISSING CHARTYPE;
   CLASS  YEAR_MONTH AREA;
   VAR    CONSUMPTION;
   OUTPUT OUT = CONSOLIDATED2 (DROP = _FREQ_ _TYPE_) SUM =;
RUN;
PROC SUMMARY DATA = SGF.SGF_1M_WAYS NWAY MISSING;
   CLASS  DISTRIBUTOR YEAR_MONTH NETWORK;
   VAR    CONSUMPTION;
   OUTPUT OUT = CONSOLIDATED3 (DROP = _FREQ_ _TYPE_) SUM =;
RUN;
```

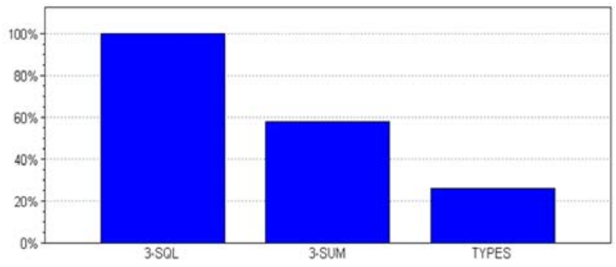
The disadvantage of this approach is that SAS has to read three times through the complete data set.

Working with a TYPES or WAYS statement in the SUMMARY procedure requires only one pass through the data set, with a positive impact on resources.

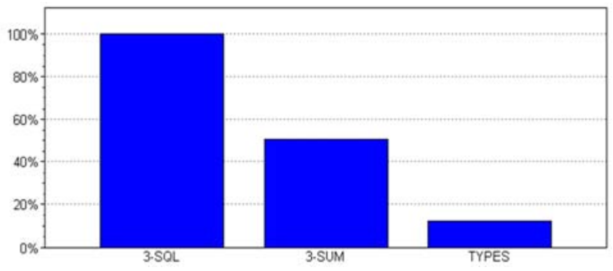
```
PROC SUMMARY DATA = SGF.SGF_1M_WAYS MISSING CHARTYPE;
   CLASS  COUNTRY AREA DISTRIBUTOR YEAR_MONTH NETWORK;
   VAR    CONSUMPTION;
   OUTPUT OUT = CONSOLIDATED (DROP = _FREQ_) SUM =;
   TYPES  COUNTRY * DISTRIBUTOR
          YEAR_MONTH * AREA
          DISTRIBUTOR * YEAR_MONTH * NETWORK;
RUN;
```

Both methods are tested on a data set with 1.000.000 records. Also the alternative with three SQL procedure executions is tested.

CPU Time



Real Time



- The improvement on the real time is even more important than the improvement on CPU time, due to the I/O reduction.

ID STATEMENT IN PROC SUMMARY

Sometimes a data set contains classification variables that are related to each other, with a 1-to-N relation. Examples include: a person lives in one street, in one city, in one country. When you require summarized data by person and city, you will find only one city for each person. Instead of specifying a CLASS statement containing both the variables person and city, you can move the city variable into an ID statement. When executing the SUMMARY procedure, SAS consolidates by the CLASS variable(s) and will add the values of the ID statement to the result.

It is important to make sure to have unique values for the variables, specified in the ID statement, to avoid incorrect results. This is illustrated in the example below.

Consider the table below.

Name	Age	Value
Marc	25	100
Marc	25	200
John	25	300
Peter	30	400

Every “Name” has a unique value for “Age”. For some “Age” values there are multiple “Name” values.

Execute a PROC SUMMARY to calculate totals by “Name” and add “Age” to the result, through an ID statement.

```
PROC SUMMARY DATA = WORK.PEOPLE NWAY;
   CLASS  NAME;
   ID     AGE;
   VAR    VALUE;
   OUTPUT OUT = WORK.SUMMED1 (DROP = _TYPE_ _FREQ_) SUM =;
RUN;
```

The result is correct, as shown below.

Name	Age	Value
John	25	300
Marc	25	300
Peter	30	400

Execute another PROC SUMMARY to calculate totals by “Age” and add “Name” to the result, by using an ID statement.

```
PROC SUMMARY DATA = WORK.PEOPLE NWAY;
   CLASS  AGE;
   ID     NAME;
   VAR    VALUE;
   OUTPUT OUT = WORK.SUMMED2 (DROP = _TYPE_ _FREQ_) SUM =;
RUN;
```

The result is incorrect, as shown below.

Age	Name	Value
25	Marc	600
30	Peter	400

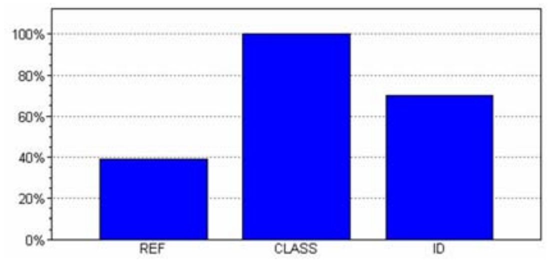
To illustrate the impact of the ID statement on the performance of a PROC SUMMARY, the next example is executed on 10.000.000 records.

- The first execution of the procedure only contains the CLASS variable “Installation” with 900.000 unique values. This step is executed to have a reference base for the other two executions.
- The second execution of the SUMMARY procedure adds three additional CLASS variables. Every “Installation” has a unique value for each of these extra CLASS variables.
- The final PROC SUMMARY only contains “Installation” in the CLASS statement; the other variables are moved to a ID statement.

```
PROC SUMMARY DATA = SGF.SGFID NWAY MISSING;
   CLASS  INSTALLATION;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.REF (DROP = _TYPE_ _FREQ_) SUM = ;
RUN;
```

```
PROC SUMMARY DATA = SGF.SGFID NWAY MISSING;
   CLASS  INSTALLATION DISTRIBUTOR SERVICE_AREA SERVICE_GROUP;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUMMED2 (DROP = _TYPE_ _FREQ_) SUM = ;
RUN;


PROC SUMMARY DATA = SGF.SGFID NWAY MISSING;
   CLASS  INSTALLATION;
   ID     DISTRIBUTOR SERVICE_AREA SERVICE_GROUP;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUMMED3 (DROP = _TYPE_ _FREQ_) SUM = ;
RUN;
```

CPU time



- Using the ID statement to add the extra information to the CLASS variable, instead of specifying this information in the CLASS statement, reduces the CPU time with more than 25%.

SYSTEM OPTIONS

THREADING

The SUMMARY procedure is one of the procedures that have been modified so that they can thread the processing through multiple CPUs, if they are available.

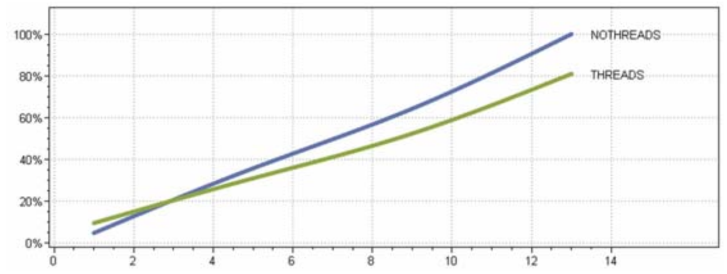
Threading is activated through a general OPTIONS statement or by setting the THREADS option in the PROC SUMMARY statement.

A test is executed with three classification variables, on a data set with 1.000.000, 5.000.000 and 9.000.000 records, with and without threads.

```
OPTIONS THREADS CPUCOUNT = 2;
PROC SUMMARY DATA = SGF.SGF_BASE NWAY MISSING;
   CLASS  DISTRIBUTOR YEAR_MONTH AREA;
   VAR    CONSUMPTION;
   OUTPUT OUT = WORK.SUMMARY (DROP = _TYPE_ _FREQ_) SUM = TOTAL_CONSUMPTION;
RUN;
```

- CPUCOUNT = 2 specifies the number of CPUs that can be used by the SUMMARY procedure.

Real Time



- For small data sets, the overhead for threading is higher than the gain in execution time, so using threading is negative in that case.
- The bigger the data sets, the bigger the difference in execution time.
- The total CPU time will not decrease, but this CPU time is split over multiple CPUs.

SUMSIZE, MEMSIZE

Memory usage for the SUMMARY procedure can be controlled by the options MEMSIZE and SUMSIZE. The SUMSIZE option can be specified as a global option in an OPTIONS statement, or can be used as an option in the PROC SUMMARY statement. Usually the value is by default already set to MAX.

CONCLUSION

When summarized data is required in an application, the SUMMARY procedure is the easiest to code. However, when resource optimization is required, you can consider the following:

- Since the SQL and SUMMARY procedures are I/O intense, using a hash table in a DATA step can improve the resource requirements.
- Hash tables are limited by memory. You can hit the limits of available memory.
- When the input data is already sorted, the SQL procedure takes most advantage of it.
- When your data is already sorted, consider using a BY statement.
- When some classification variables have a 1-to-N relation with other classification variables, consider using an ID statement.
- For large volumes, the THREAD option can reduce the execution time of the summarization task.

When selecting a summarization method, follow these guidelines:

- Test several alternatives and select the best performing one.
- Execute your tests in the same circumstances as your final production job: use the same data set size, the same number of variables, etc.
- The better you know your data, the better you can select the best methods.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Theuwissen Henri
BI Knowledge Sharing
Sterrenlaan 40
B-3360 Bierbeek
Belgium
Phone: +32 495 54 52 53
Fax: +32 16 46 37 74
E-mail: Henri.Theuwissen@BIKnowledgeSharing.be
Web: www.BIKnowledgeSharing.be