

# The Efficiency of Proc SQL in Place of Traditional Procedures for Large Data Sets

Cindy Stroupe, ClinTrials Research, Inc., Cary, NC

## Abstract

ClinTrials Research, Inc. has had a large, ongoing, marketing study for several years. Each year brings additional patients and additional visits for existing patients. A majority of the data sets contain over half a million records. This volume of data creates a problem not only in processing resources but also processing time.

The SQL procedure is a very effective method of sorting, counting, and selecting data that does not require the same amount of processing resources or time as traditional procedures. This paper will give you practical examples of the use of SQL with large data sets.

Key Words: SQL

## Introduction

In clinical research, time is money, timelines are short and studies vary in size from 10 to 30,000 patients. Efficiencies need to be gained from every possible avenue. Programming efficiencies can be gained not only in processing time, but also programmer time, and computer resources.

One of these ways is to use the SQL procedure in place of traditional procedures, especially when large data sets are being processed. The SQL procedure can sort data and perform mathematical calculations in one step. Also, SQL can be up to 10 times faster than traditional procedures, especially when sorting.

## The SQL Syntax

The syntax for the SQL procedure is different from traditional SAS<sup>®</sup> syntax. However, the code for a basic SQL procedure (sort, count, etc...) is easy to learn and modify. Following are examples of Proc SQL code to use when sorting, selecting, or manipulating large data sets.

### *The Basic Sort and Keep*

#### The Traditional Sort

```
proc sort data=indata.demo out=demo;  
by inv pat;  
keep inv pat page age;  
run;
```

```
proc sort data=indata.demo out=demo2 nodupkey;  
by inv pat;  
keep inv pat page age;  
run;
```

#### The SQL Sort

```
proc sql;  
create table demo1 as  
select inv, pat, page, age  
from indata.demo  
order by inv, pat;  
quit;
```

```
proc sql;  
create table demo3 as  
select distinct(pat) as onepat, inv, page, age  
from indata.demo  
order by inv, page, age;  
quit;
```

In the first SQL example you can see the basic structure of SQL code. The 'select' statement is used the same way that the 'keep' statement is used in the traditional sort. An '\*' can also be used in place of individual variable names if all variables are to be selected. The 'order by' statement is the sort order of the final data set.

The second SQL example shows the 'distinct' option of proc SQL which functions the same as the 'nodupkey' option in the traditional sort.

## Manipulating Data

### The Traditional Univariate

```
proc sort data=indata.spirom out=spirom;
by inv pat;
run;
```

```
proc univariate data=spirom noprint;
var fev1 fvc1;
by inv pat;
output out=spirom1 min=fevmin fvcmin
max=fevmax fvcmax;
run;
```

```
proc sort data=indata.drug out=drug;
by inv patinit;
run;
```

```
proc univariate data=drug noprint;
var dose;
by inv patinit;
output out=drug1 mean=meandose;
run;
```

```
proc sort data=indata.antibio out=antibio;
by pat class strtdt;
run;
```

```
proc univariate data=antibio noprint;
var route;
by pat class strtdt;
output out=antibio1 n= rtenum;
run;
```

### The SQL Univariate Functions

```
proc sql;
create table spirom as
select min(fev1) as fevmin, min(fvc1) as fvcmin,
max(fev1) as fevmax, max(fvc1) as fvcmax,
inv, pat
from indata.spirom
group by inv, pat
order by inv, pat;
```

```
create table drug as
select mean(dose) as meandose, inv, patinit
from indata.drug
group by inv, patinit
order by inv, patinit;
```

```
create table antibio as
select count(route) as rtenum, pat, class, strtdt
from antibio
group by pat, class, strtdt
order by pat, class, strtdt;
```

quit;

The above examples show that proc SQL can do both a traditional sort and univariate function in one step. In contrast, data should be sorted before the univariate procedure is used. Even the 'notsorted' option on these procedures does not gain efficiencies. Sixteen different statistical functions can be performed within the SQL procedure. A 'group by' statement must be used when a mathematical function is being performed.

### **The Efficiency Comparison**

One of the best gauges of efficiency is time. In the SAS<sup>®</sup> log file, real time and cpu time are printed for every step processed. If just looking at time, SQL is up to 10 times more efficient than traditional procedures. The table below gives the number of records contained within the data set, the amount of real time, and the amount of cpu time for both a traditional sort and a SQL sort.

# of records	Real time		Cpu time	
	Sort	SQL	Sort	SQL
268115	11:23.08	1:14.42	53.61	10.43
415415	8:46.44	1:22.94	52.20	13.48
428835	9:29.86	1:25.54	53.63	14.32
767202	23:16.14	12:56.02	2:21:53	1:31.17

As you can see, SQL cuts down on both real time and cpu time.

Another measure of efficiency is resources. Traditional procedures use more processing space than the SQL procedure. The table below gives a comparison of the amount of memory used to process a traditional univariate versus using the SQL procedure.

# of records	Memory Used	
	Univariate	SQL
415415	133k	100k
428835	126k	86k
767202	114k	87k

### **Conclusion**

The SQL procedure can be a very efficient tool to use when processing large data sets. It also minimizes the number of steps that need to be written to perform a task. SQL saves processing time, programmer time and memory used. Despite, the coding differences between traditional SAS® code and SQL, the code is easy to learn and modify.

### **Contact Information**

Cindy Stroupe  
ClinTrials Research, Inc.  
P.O. Box 13991  
Research Triangle Park, NC 27709  
Work Phone: (919) 462-2670  
Fax: (919) 462-2773  
Email: [cstroupe@clintrialsresearch.com](mailto:cstroupe@clintrialsresearch.com)

