If I'm not a database user, what is the SQL procedure good for?

Ann Olmsted, Syntex Research, Palo Alto, California

Abstract

SAS* procedure SQL may offer the simplest way to perform several common and not-so-common tasks, such as taking Cartesian products of SAS data sets (combining every observation in one data set with every observation in another), creating macro variables, adding summary statistics to a SAS data set, producing simple reports, and performing set operations (union, intersection, difference). Complete examples are given, plus a quick reference card, plus a list of PROC SQL traps for novices (so you won't make the same mistakes I did).

Introduction

Here is a simple PROC SQL program and its output:

```
   data dummy ; retain x '1' ;

   proc sql feedback ;
      select 'Hello, world' as message
         from dummy ;
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 MESSAGE
 ------------
 Hello, world
```

SQL is strange. Nothing is actually being selected from the dummy data set. Nevertheless, the dummy data set must have at least one variable, even though the SAS System will let you create a SAS data set with zero variables. This is because PROC SQL will produce the following error message if it encounters a data set with zero variables:

```
 ERROR: Table WORK.DUMMY doesn't have any columns. PROC SQL requires
 each of its tables to have at least 1 column.
```

Here is a useful PROC SQL program and its output:

```
proc sql feedback ;
   select sum( (.z < a.score < b.score) + 0.5*(.z < a.score = b.score) )
   as J label='Jonckheere statistic'
      from sasfil.dpeptalk as a, sasfil.dpeptalk as b
      where a.trt < b.trt ;
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 sasfil.dpeptalk for checking
 TRT    SCORE
  2       77, 69, 80, 89, 41, 76

  1       75, 81, 44, 72, 79, 53

  0       70, 21, 39, 52, 68, 67

 Jonckheere
  statistic
 ----------
         81
```

A single statement calculates the Jonckheere statistic for testing for an upward (or downward) trend.

The morals of these 2 examples are:

PROC SQL's ways are strange and can be annoying
PROC SQL is by far the best way to do certain things

For me, the wonderful : annoying ratio has been about 3 : 1 or 4 : 1. The aim of this paper is to raise it for you to about 4 : 1 or 5 : 1, by first describing some of the things PROC SQL is best at, and then listing some of its traps so you can avoid them (or resign yourself to them).

Translating: SQL <--> SAS

This has 2 parts:

ordinary SAS code <--> PROC SQL code
ordinary SAS terms <--> SQL terms

Appendix 1 illustrates translating DATA steps to PROC SQL statements. (Note that a single PROC SQL statement is usually the equivalent of an entire DATA step.) The table below illustrates the second kind of translation:

SQL term	SAS term
base table	SAS data file
table (can be view)	SAS data set (can be data file or data view)
row, n-tuple	observation
column	variable
heading	a SAS data set's variable-name:variable-type (character or numeric) pairs
union-compatible, type-compatible	having the same heading (standard SQL); having no type conflicts and having headings whose intersection is nonempty (PROC SQL)

What PROC SQL is (particularly) good for

Forming cross products

Syntax:

```
select comma-list
   from A, B, C, ... ;
```

to form all possible horizontal concatenations of 1 observation from A, 1 from B, 1 from C, ..., where A, B, C, ... need not be distinct.

Example 1:
File WT has 1 observation per animal and file DATE has 1 observation (variables DATE1 DATEF). To calculate each animal's rate of gain:

```
select *, (WTF - WT1)/(DATEF - DATE1) as ADG label='ADG (lb/d)'
   from WT, DATE ;
```

Example 2:
Given a set of n (x,y) pairs, Theil's nonparametric slope estimate is the median of the n choose 2 pairwise slopes. File DTENNIS has variables RACKET, HEAD (head area) and SPOT (sweet spot index), and a linear relationship between head area and sweet spot index is hypothesized:

```
create table WORK as
select
A.racket as racket1,
B.racket as racket2,
(B.spot - A.spot)/(B.head - A.head) as slope
   from sasfil.DTennis as A, sasfil.DTennis as B
   where A.racket < B.racket
   order by slope ;
proc univariate data=WORK plot ; * to find the median slope ;
   var slope ;
```

Fine points: What happens if the files have variables in common? For instance, what happens if file A has variables X Y and file B has variables Y Z? A SELECT statement will list all 4-tuples (xi, yi, yj, zj) where (xi, yi) is the ith observation from A and (yj, zj) is the jth observation from B. A SAS file created using the SELECT statement will contain all 3-tuples (xi, yi, zj).

Creating macro variables

Syntax:

```
proc sql feedback noprint ;
   select x, y, x, ...
      into :x, :y, :x, ...
      from ... ;
```

```
Example 1:
* create conversion-factor macro variable to convert g per English ton
  of 90% dry matter feed to mg per kg dry matter ;
proc sql noprint ;
   create table dummy (x character(1)) ; insert into dummy set x='1' ;
   select 2.204622622 / (0.9 * 2) into :convert from dummy ;
%put convert=/&convert/;

Example 2:
proc sql feedback noprint ;
   select count(*) as n
      into :n
      from sasfil.DTennis
      where head is not missing and spot is not missing ;
proc plot data=sasfil.DTennis ;
   plot spot*head / box ;
   title1 'sasfil.DTennis for checking' ;
   footnote1 "data for &n rackets plotted" ;
```

Fine points: If the result file has more than 1 observation, macro-variable values come from the first observation. With the DATA step CALL SYMPUT equivalent, values come from the last observation.

Quick fixes

```
Syntax:
   delete
      from A
      where ... ;
or
   update A
      set name = expression, name = expression, name = expression, ...
      where ... ;

Example 1:
   update A
      set x = x+0 ;
```

to convert special missing values to ordinary missing values. (Note: x = --x will work in a DATA step but not in PROC SQL. Use the FEEDBACK option to see why. Question: why is PROC SQL clever enough to realize --x = x but not clever enough to realize x+0 = x?)

```
Example 2:
   delete
      from A
      where eartag is missing ;
```

to delete observations for animals with missing ID numbers.

Quick reports

When all you need is a quick PROC PRINT, except the observations are in the wrong order and you don't need to print all of them and you need to add a calculated variable and ....

```
Syntax:
   select comma-list
      from ...
      where ...
      order by ... ;

Example 1:
   select name, source label='Contrast', prob label='p' format=putp.
      from OUTSTAT /* created by PROC GLM */
      where type='CONTRAST'
      order by prob desc ;
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 NAME      Contrast                  p
 ------------------------------------
 G13       3 vs 2                .41
 G13       2 vs 5                <.001
 G13       3 vs 5                <.001

Example 2:
If you want to display the same value formatted in different ways, you
can do this more easily with PROC SQL than PROC PRINT:

   select
   x format=best.    label='F=best.',
   x format=wordf20. label='F=wordf20.',
   x format=words40. label='F=words40.'
      from WORK ;
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 F=best.            F=wordf20.                                F=words40.
 ------------------------------------------------------------------------
   19.95  nineteen and 95/100   nineteen and ninety-five hundredths
    9.5   nine and 50/100       nine and fifty hundredths
    8.49  eight and 49/100      eight and forty-nine hundredths

Set operations (union, intersection, difference)
Syntax:
   select ...
   except corr | intersect corr | union corr
   select ...

Example 1:
To see if two files with the same heading are identical:

%macro fc(a,b);
 /*********************************************************************/
 /*  DESCRIPTION:                                                     */
 /*                                                                   */
 /*  Create global mac var EQ with value 1 if SAS files A & B contain */
 /*  the same observations and value 0 if they do not.                */
 /*********************************************************************/
```

```
%global eq;
proc sql feedback noprint ;
   select count(*) as n1 label="#obs in &a but not &b"
      into :n1
      from ( select * from &a except corr select * from &b ) ;
   select count(*) as n2 label="#obs in &b but not &a"
      into :n2
      from ( select * from &b except corr select * from &a ) ;
%let eq = %eval(¬(&n1 + &n2));
%put a=/&a/ b=/&b/ n1=/&n1/ n2=/&n2/ eq=/&eq/;
%mend  fc;
```

Simulations

Given a parameter file with n observations, how to run a large number of simulations per observation without bringing a shared system to its knees?

An approach that works well is:

- Build the n-observations parameter file and a zero-observations file to hold simulation results.
- Use SELECT INTO to select 1 observation's parameter values into macro variables. Delete the observation, run the simulations, and append the results to the results file.
- Repeat until the parameter file has no more observations. Print out the results file.

Example 1:

```
proc sql feedback ;
   select r, k, seed into :r, :k, :seed from sasfil.params (obs=1 ) ;

<lots of simulation SAS code goes here>

proc sql feedback ;
   create table WORK as
   select
   &r as r,
   &k as k,
   &seed as seed,
   &B as B,
   sum(.z < prob < &alpha) as x
      from OUTSTAT
      where source = '34 vs 12' ;

proc append base=sasfil.results data=WORK ;
data _null_ ; if ( 4 < &syserr ) ; call execute('endsas ;') ; run ;

proc sql feedback ;
   delete from sasfil.params where r=&r and k=&k ;
   select count(*) as n label='No. of observations left'
      from sasfil.params ;
```

Adding summary statistics to a SAS data set

Syntax:

```
   select group-by-list, expressions involving summary functions,
   expressions that are multi-valued per group
      from ...
      where ...
      group by group-by-list
      having ...
      ;
```

Example 1:

```
* list observations with nonunique values of variable ID in WORK ;
proc sql ;
   title1 "Multiple occurrences of ID in WORK" ;
   select * from WORK group by ID having count(*)>1 order by ID ;
   title1 ;
```

Example 2:

Macro to perform Levene's test of equality of variances for a 1-way layout:

```
%macro Levene(data=WORK,class=trt,var=adg13);
 /*****************************************************************/
 /*  DESCRIPTION:                                                 */
 /*                                                               */
 /*  Input is SAS file &DATA with class variable &CLASS and       */
 /*  response variable &VAR.  Macro performs ANOVA of absolute    */
 /*  deviations from &CLASS means and outputs SAS file OUTSTAT    */
 /*  containing p-value PROB.                                     */
 /*****************************************************************/
run ;
title1 " Levene:  data=/&data/, class=/&class/, var=/&var/ " ;
proc sql feedback ;
   create table Levene as
   select &class, &var, avg(&var) as mean,
   abs(calculated mean - &var) as AbsDev
   label='Absolute deviation from mean'
      from &data
      group by &class ;
proc glm data=Levene order=internal outstat=OutStat ;
   class &class ;
   model AbsDev = &class / ss3 ;
   lsmeans &class / stderr pdiff ;
   title2 'ANOVA of absolute deviations from the mean' ;
proc print data=OutStat ;
   where ( _source_="%upcase(&class)" ) ;
   title2 'OutStat for checking' ;
%mend  Levene;
```

Fine points: Automatic remerge of summary statistics is not permitted by the SQL/92 standard (Date 1993, p. 143).

PROC SQL annoyances

Comma-separated lists

For a SAS coder, PROC SQL's number 1 annoyance is the comma-separated list. For instance:

```
proc sql feedback ;
   select bunk, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12,
   d13, d14, d15, d16, d17, d18, d19, d20, d21, d22, d23, d24, d25,
   d26, d27, d28
      from sasfil.admil
      order by bunk ;
```

If through force of habit you slip up and type:

```
select bunk, d1-d28
...
```

PROC SQL will obediently subtract d28 from d1 for you. You could write a macro like Comma to ease the pain:

```
%macro Comma(text);
%local i;
%let i=1;
%do %while ( %scan(&text,&i+1,%str( ))¬= ); %scan(&text,&i,%str( )), %let
i=%eval(&i + 1); %end; %scan(&text,&i,%str( ))
%mend  Comma;
```

```
select %Comma(bunk d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12 d13 d14
d15 d16 d17 d18 d19 d20 d21 d22 d23 d24 d25 d26 d27 d28) ...
```

However, except for one-time-only jobs it's probably best to resign yourself to editing in all those commas. WARNING: You might think you can use abbreviated variable lists in SAS function calls. You can't:

```
* this will work ;
select i, sum( x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12,
x13, x14, x15, x16, x17, x18, x19, x20 ) as sum
   from WORK
   order by i ;

* this won't work ;
select i, sum(of x1-x20) as sum
   from WORK
   order by i ;
```

Merging files by more than one variable requires a lot of typing

The PROC SQL equivalent of:

```
   * merge A and B by x y z, reporting match failures ;
   data WORK ;
      merge A (in=InA )
            B (in=InB )
            ;
      by x y z ;
      if ¬(InA and InB) then do ;
         put (x y z) (=) "inA=" InA "inB=" InB ;
      end ;
      if (InA and InB) ;
```

is:

```
   create table WORK as
   select *
      from A, B
      where a.x=b.x and a.y=b.y and a.z=b.z ;
```

You could write a macro like USING to save some typing (and conform to the SQL/92 standard):

```
%macro Using(x);
 /*********************************************************************/
 /*  DESCRIPTION:                                                     */
 /*                                                                   */
 /*  Convert an argument like x y z to a.x=b.x and a.y=b.y and a.z=b.z./
 /*  Use this macro for PROC SQL joins.  For example:  a join b       */
 /*  %using(x y z) ;                                                 */
 /*********************************************************************/
<macro statements>
%mend  Using;
```

However, it's probably best to resign yourself to typing out the WHERE condition.

First character of variable label must be a letter, a digit, or a blank

Unlike PROC SQL, ordinary SAS has no restrictions (that I know of) on variable label characters. Note that PROC SQL does not discard the offending first character (as the PROC CONTENTS output shows), it simply doesn't display it:

```
   proc sql feedback ;
      create table WORK as
      select trt, count(score) as n label='# of scores'
         from sasfil.dpeptalk
         group by trt
         order by trt ;
      select * from WORK ;
   proc contents data=WORK ;
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
<PROC SQL output>
    0=Low,
 1=Medium,
    2=High   of scores
 ---------------------
         0           .6
         1            6
         2            6
<PROC CONTENTS output>
 #    Variable    Type    Len    Pos    Label
 ------------------------------------------------------------
 2    N           Num       8      2    # of scores
 1    TRT         Num       2      0    0=Low, 1=Medium, 2=High
```

Ordinary SAS and PROC SQL type conversion rules differ

Below, LOT is a 3-digit character variable. Ordinary SAS converts it, but PROC SQL won't:

```
data _null_ ;
   set WORK ;
   blk = ceil((lot - 379)/4) ;
   put lot= blk= ;
proc sql feedback ;
   select lot, ceil((lot - 379)/4) as blk
      from WORK ;
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
<DATA step log messages>
 NOTE: Character values have been converted to numeric values at the
       places given by: (Line):(Column).  62:16
 LOT=380 BLK=1
 LOT=381 BLK=1
 LOT=382 BLK=1
 ...
<PROC SQL step log messages>
 ERROR: Expression using subtraction (-) requires numeric types.
```

You get only 1 formats catalog per step

If you use LIBNAME statements to change formats catalogs, the change appears to have taken place, but the original formats are used throughout the step:

```
proc sql ;
   libname library 'a.sas' ;
   <PROC SQL statements using formats stored in a.sas>

   libname library clear ;
   libname library 'b.sas' ;
   <PROC SQL statements using formats stored in a.sas, not b.sas
   as desired>
```

This feature of PROC SQL confuses me, and I may not have described it correctly.

The only way I've found to use a new formats catalog is to start a new PROC SQL step.

It's hard to bail out of a PROC SQL step

In a DATA step, you can code something like:

```
if <input is bad> then abort ;
```

Here's an awkward PROC SQL equivalent (dummy is any SAS data set having 1 observation and 1 or more variables):

```
proc sql feedback ;
%let stmt=;
   * if WORK file id values are not unique, end the job ;
   select 'endsas ;' as stmt
      into :stmt
      from dummy
      where exists( select '47' from WORK group by id having count(*)>1
      ) ;
&stmt
   <more statements>
```

PROC SQL is too clever to evaluate clauses in the obvious order

This is the obvious (?) order:

```
select        6
   into       7
   from      .1
   where      2
   group by   3
   having     4
   order by   5
```

That is, first PROC SQL forms the product of the FROM-clause tables, then it applies the WHERE-clause restriction, then it groups, then it applies the HAVING-clause restriction to the groups, then it sorts, then it eliminates unwanted variables and adds calculated variables, then it stores values in macro variables. However, a user of the pseudo-random number functions will discover that PROC SQL is too efficient to perform operations in the obvious order:

```
proc sql feedback ;
   create table WORK as
   select a.seq as eartag label='Eartag', a.trt,
   round(b.mu + b.sigma*RANNOR(&seed),0.01) as x label='Dummy variable'
      from sasfil.drand as a, sasfil.ddummy as b
      where a.trt=b.trt
      order by eartag ;
```

Variable TRT values are 1, 2, 3, 4. File drand has variables SEQ TRT and 200 observations (50 per TRT value). File ddummy has variables TRT MU SIGMA and 4 observations (1 per TRT value). The intention was to call the RANNOR function 200 times. However, PROC SQL cleverly observes that variable X depends only on the population parameter values in file ddummy, and calls RANNOR 4 times. Moral: remember that PROC SQL may optimize.

Summary

Recommendations: To form cross products, use PROC SQL. To match-merge by x, use the DATA step MERGE ... ; BY ... ; statements if the files are sorted by x, otherwise use PROC SQL. To match-merge by several variables, use the DATA step MERGE ... ; BY ... ; statements.

To create macro variables without being limited to %LET's integer arithmetic, use PROC SQL. To create simple reports when you need to subset, sort, or add calculated variables to the input file, or display the same variable using more than one format, use PROC SQL.

To merely display summary statistics, use PROC MEANS, SUMMARY, TABULATE, or UNIVARIATE. To add them to a SAS data set, use PROC SQL, unless you need a WEIGHT statement.

Warnings: Do not attempt to use abbreviated variable lists (for instance, x1-x10). Do not begin labels with special characters. Code type conversions explicitly. Use the FEEDBACK option to minimize surprises, and remember that PROC SQL optimizes.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. * indicates USA registration.

All the examples above were copied from working (or error-message-generating) code run on IBM Model 9021s. The release was MVS SAS 6.08 TS404.

Some of the examples use actual Syntex-trial data. To preserve confidentiality, when examples are based on Syntex-trial data the data are not listed and the trials are not identified. Other examples are taken from the course notes of a 3-day course in Nonparametric Statistical Methods given by Professors R. Randles and D. Wackerley of the University of Florida.

Suggested reading

Date, C.J. (1993). A guide to the SQL Standard : A user's guide to the standard relational language SQL, Third Edition. Addison-Wesley.
Describes SQL/92. Read Chapter 11 (Table Expressions). PROC SQL extends and subtracts from the SQL/92 standard, but Chapter 11 will give you a conceptual model of what PROC SQL SELECT statements do.

SAS Institute Inc. (1989). SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition. SAS Institute Inc., Cary, NC.
You need this.

SAS Institute Inc. (1990). SAS Procedures Guide, Version 6, Third Edition. SAS Institute Inc., Cary, NC.
See Chapter 34. Anyone not already familiar with SQL who can learn to use PROC SQL by reading this chapter is very clever or very determined.

SAS Institute Inc. (1991). SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07. SAS Institute Inc., Cary, NC.
See Chapter 37. The most useful section is the one about DICTIONARY-tables and SASHELP views. Sample use:

```
* create macro variable exist with value 1 if SAS file s.rand
  exists and 0 if it doesn't ;
select count(*) as count
   into :exist
   from dictionary.tables
   where libname="S" and memname="RAND" ;
```

Appendices

A1-- Inner, left, right, and full joins and their ordinary SAS equivalents

Appendix 1

Inner, left, right, and full joins and their ordinary SAS equivalents

3 animals, eartagged 1 to 3, are weighed at the beginning of the pasture season. At the end of the season, when the animals are reweighed, tag 1 has disappeared and a new animal, tag 4, has been rounded up together with tag 2 and tag 3:

wtl for checking

EARTAG	WTl
1	600
2	620
3	640

wtf for checking

EARTAG	WTF
2	880
3	910
4	940

The 4 ordinary SAS data steps below correspond to the 4 PROC SQL join types, inner, left, right, and full:

```
* merge wtl and wtf by eartag ;
data WORK ;
   merge wtl (in=InA ) wtf (in=InB ) ;
   by eartag ;
   if (InA and InB) ;

data WORK ;
   merge wtl (in=InA ) wtf (in=InB ) ;
   by eartag ;
   if (InA) ;

data WORK ;
   merge wtl (in=InA ) wtf (in=InB ) ;
   by eartag ;
   if (InB) ;

data WORK ;
   merge wtl (in=InA ) wtf (in=InB ) ;
   by eartag ;

proc sql feedback ;
   titlel 'inner join using eartag (animal ID)' ;
   select A.eartag, A.wtl, B.wtf, B.wtf - A.wtl as glf label='Gain'
      from wtl A inner join wtf B on A.eartag=B.eartag ;
```

```
title1 'left join using eartag (animal ID)' ;
select A.eartag, A.wt1, B.wtf, B.wtf - A.wt1 as glf label='Gain'
   from wt1 A left join wtf B on A.eartag=B.eartag ;

title1 'right join using eartag (animal ID)' ;
select B.eartag, A.wt1, B.wtf, B.wtf - A.wt1 as glf label='Gain'
   from wt1 A right join wtf B on A.eartag=B.eartag ;

title1 'full join using eartag (animal ID)' ;
select coalesce(A.eartag,B.eartag) as eartag,
A.wt1, B.wtf, B.wtf - A.wt1 as glf label='Gain'
   from wt1 A full join wtf B on A.eartag=B.eartag ;
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
<PROC SQL output>
 inner join using eartag (animal ID)
          Initial      Final
 Animal    weight     weight
 ID          (lb)       (lb)       Gain
 ------------------------------------
 2            620        880        260
 3            640        910        270

 left join using eartag (animal ID)
          Initial      Final
 Animal    weight     weight
 ID          (lb)       (lb)       Gain
 ------------------------------------
 1            600          -          -
 2            620        880        260
 3            640        910        270

 right join using eartag (animal ID)
          Initial      Final
 Animal    weight     weight
 ID          (lb)       (lb)       Gain
 ------------------------------------
 2            620        880        260
 3            640        910        270
 4              -        940          -

 full join using eartag (animal ID)
          Initial      Final
           weight     weight
 EARTAG      (lb)       (lb)       Gain
 ------------------------------------
 1            600          -          -
 2            620        880        260
 3            640        910        270
 4              -        940          -
```

The SAS code is clearer and cleaner. Note that the COALESCE function was needed to perform the PROC SQL version of the all-animals join.