

Developing County Profiles using PROC SQL

Arlene Slocum

Institute for Public Policy and Business Research, University of Kansas, Lawrence, Kansas

Introduction

In my job I spend a lot of time dealing with the same basic sets of variables (i.e., population, employment, income etc.). I take these same numbers and manipulate them in endless combinations depending on the particular project. As a result, I have wound up literally drowning in differently, transposed, or extracts for a particular geographic region, or extracts over a specific range of years, and on and on and on ...

We want to move away from repeating data in databases and more toward a single repository of non-repetitive data that can be viewed in different ways. The design of the County Profile Reporting System using a combination of base SAS® software and the SAS/SQL-DS®1 software (better known as Proc SQL) is an attempt at trying to reorganize data so it can be better situated for use for a variety of purposes. I had no prior knowledge of SQL before this project, but found that I was able to accomplish a lot in a short period of time by just learning how to do a few basic queries.

In this paper I will discuss two SAS programs. Program 1 reads transactions for county level data and builds several SQL tables. Each table contains all the data for a single variable (example: table EMPLOYED contains the number of employed persons for each Kansas county for several years). These tables also contain computed county rankings and per capita figures which will be printed on the county profile report. A statewide non-metropolitan total is computed for each variable and year and is kept in a separate table with the statewide totals.

Program 2 extracts the data for a single county from the county level tables created by Program 1 and uses the data to print a county profile report.

I tested this software on a DEC™ 5000 series (model 240) RISC based computer running Unix™ and SAS software version 6.07.

Inputs:

STATE table: the transactions

VARIABLE	COUNTY	YEAR	CTYVAL
TOTPOP	1	1980	15654.000
TOTPOP	1	1990	14638.000
...			
TOTPOP	209	1980	172335.000
TOTPOP	209	1990	161993.000
MED AGE	1	1990	35.500
EMPLOYED	3	1990	3589.000
EMPLOYED	3	1991	3610.000

STATE table: State totals and flags for use in printing profile

Example record for MED AGE median age:

Variable	Value	Description
SUBJECT	P	Population Section
VORDER	2	Will print as variable number 2 in Pop Section
VARNAME	MED AGE	the table (variable) name
YEAR	1990	
UNITS		blank here - but could be coded to indicate \$, \$000, acres, sq mi., etc.
DEC_PLG	1	Number of decimal places to use when printing
RANK_FLG	Y	Y means print the county rankings
PCAP_FLG	N	N means don't print per capita median age (it is a useless number!)
KS_FLG	Y	Y means print the statewide value
NMET_FLG	N	N means don't print the non-metro total
KANSAS	32.9	The statewide median age
NONMETRO	.	The non-metropolitan county totals go here

POPS table: total population for each county for all years. This is needed to compute per capita or per 1000.

COUNTY	YEAR	CTYVAL (population)
1	1980	15654
1	1989	.
1	1990	14638
3	1989	.
3	1980	8749
3	1990	7803

Program 1 - Section 1

```

/* get variable names and years to process */
proc sql;
create table varyr as
select distinct varname, year from cty
order by varname, year;

/* make copy of state table */
create table ks as
select * from mwsas.state
order by varname, year;

/* create the rural table which will hold the
computed nonmetro totals */
create table rural
(varname char(8), year num,
nonmetro num format=15.3);
quit;

```

Section 2 is a SAS macro that generates macro variables VNAME and YR which will contain the current variable and year to process. Note that this is done as a separate data step so the macro variable values can be passed to the rest of the program.

Program 1. Section 2.

```
=====
%global NXT DONE;
%let NXT=1;
%let DONE=0;
/* crank will get next var/yr to process */
%macro crank;
  data _null_;
    obsnum=&NXT;
    if obsnum > totrecs then do;
      put 'end of processing';
      call symput('done',1);
    end;
    else do;
      set varyr point=obsnum nobs=totrecs;
      /* generate macro variables vname & yr */
      call symput('vname',varname);
      call symput('yr',year);
      /* point to the next record */
      call symput('nxt', obsnum + 1);
    end;
  stop;
  run;
%mend crank;
```

The following section of code will isolate only the data for a single variable and year (e.g. TOTPOP 1980). We return to Proc SQL to set up the data so we can perform the rank and per capita calculations.

Program 1. Section 3.

```
=====
/* update a variable for one year */
%macro vyrupd;

%put Now processing variable &VNAME year &YR;
/* do existence test for variable table */
/* see Chapter 11 SAS Guide to Macro
   Processing V 6 */
%exist(mwsas.&VNAME)
```

```
proc sql;
```

```
/* generate table of state and cty stuff
combined for one year for one variable */
create table oneyr as
select distinct
  cty.varname,cty.county,cty.year,cty.ctyval,
  ks.units from cty, ks
  where cty.varname = ks.varname
  and cty.varname = "&VNAME" and
        cty.year = ks.year and cty.year=&YR
  order by cty.county, cty.year;

/* alter table to add percapita and rank*/
alter table oneyr
  add
    per_cap num format=10.3,
    c_rank num format=3.;
```

Next we will add all the values in the non-metropolitan counties together to get a non-metropolitan total for the variable and year. We will store this total temporarily in the rural table using the SQL insert command. Note that in Kansas there are 9 metropolitan counties and the rest are "rural" (non-metro).

Program 1. Section 4.

```
=====
/* now lets compute the non metro total */
/* for the current varname and year */
/* and insert it into the rural table */
insert into rural
select distinct varname,year,
  sum(ctyval) as nonmetro from oneyr
  where not (county in
    (045,091,015,079,173,177,209,121,103));
```

The next section of code computes the per capita or per 1000 value. Per capita is computed for money values (where the units code = B, C, or M). When the data are not monetary, then the county value is divided by the population for the county and multiplied by 1000 so you can get figures like "dentists per 1000 persons". Note that the table for the single variable and year is joined to the POPS table which contains county populations for all years. At the end of the statement the resulting table is sorted by county value in descending order so we can be ready to compute the county rankings.

Program 1. Section 5.

```
=====
/* the jtable will have updated percapita */
create table jtable as
select distinct oneyr.county,oneyr.varname,
       oneyr.year,oneyr.ctyval,oneyr.c_rank,
       ((oneyr.ctyval/pop.ctyval)*
       case
/* Per Capita is always computed in dollars*/
when oneyr.units='B' then 1 /* data in $*/
when oneyr.units='C' then 1000
/* thousands of $*/
when oneyr.units='M' then 1000000
/* millions of $*/
else 1000 /* use to compute per 1000 */
end) as per_cap
from oneyr, mwsas.pops as pop
where oneyr.year=pop.year
and oneyr.county = pop.county
/* sort results by county value descending
for ranking */
order by ctyval desc;
quit;
```

A SAS data step was used to do the ranking.

Program 1. Section 6.

```
=====
/* use data step to do ranking modified
sports method */
/* example:      Value      Rank
                100        1
                75         2
                75         2
                50         4          */

data jrank;
set jtable;
drop oldval oldrank;
retain oldval oldrank;
if _N_ = 1 then do;
  c_rank=1; output; oldval=ctyval; oldrank=1;
end;
else do;
  if (ctyval < oldval) then do;
    oldval = ctyval;
    c_rank=_N_;
    oldrank=c_rank;
  end;
  else
    c_rank=oldrank;
  output;
end;
run;
```

The UPDATE statement in the SAS data step was used to update the variable table with data for the new year. There are flaws with the update statement, most notably the fact that it will not change a value to missing unless extra steps are taken to name missing values. There may be better ways to do this using SQL or the new MODIFY statement in the data step.

Program 1. Section 7.

```
=====
/* sort table jrank before applying it as a
transaction file - note that proc sort
would work just as well */
proc sql;
create table j2 as
select *
  from jrank
  order by county, year;
/* if table for a variable does not exist,
then create it */
%if &EXIST=yes %then %do;
  create table onevar as
  select * from mwsas.&VNAME
  order by county, year;
  quit;
  data mwsas.&VNAME;
  update onevar j2;
  by county year;
run;
%end;
%else %do;
  create table mwsas.&VNAME as
  select *
    from j2
    order by county,year;
  quit;
%end;
%mend vyrupd;
```

The next section of code is a macro that will update or create tables for each variable. One variable/year combination is processed at a time. For example: TOTPOP 1980, TOTPOP 1990, MED_AGE 1990, etc. After all the variable tables have been updated, the state table is updated with the non-metro totals that have been collected in the rural table (see Section 4).

Program 1. Section 8.

```
/* run the profile update for all vars and
years on input file */
%macro doit;
%crank /* get a var year to process */
%if (not &DONE) %then
%do %until (&DONE);
  %vyrupd /* update the dataset */
  %crank /* get the next var/yr */
%end;
```

```

/* update the state with non-metro totals */
data mwsas.state;
  update ks rural;
  by varname year;
run;
%mend doit;

/* lets roll! */
%doit

```

Program 2. Print the county profile report.

For the sample input transactions, Program 1 produced as output 3 tables of county level data. The TOTPOP table contained total population for 1980 and 1990. MED_AGE contains the median age for 1990 and EMPLOYED contains the number of employed persons for 1990 and 1991. The STATE table was updated with non-metropolitan totals for these variables.

Program 2 is only interested in the data for all of the variables for one county at a time. The sample report is small, since it is a profile of 3 variables. A more realistic profile may contain over 100 variables. I have found that the most efficient way to get just one county's data from a table containing all of the counties would be to index the table by county and use the WHERE clause on the SET statement to select the county. Since there are 105 counties in Kansas the index would be efficient since you are accessing only 1 percent of the data in the selection. I then concatenate the tables for all of the variables. A problem occurs when you try to concatenate too many files in this manner in a single SAS data step or SQL statement. There is a limit of only 16 files that can be referenced in a single SET statement or with UNION clauses in Proc SQL. To get around this limitation I used a lot of "%" statements from the *SAS Guide to Macro Processing*. The code presented for concatenation (See Program 2. Section 3) will work for up to 240 variables in a county profile report. I tried this using the UNION statement in PROC SQL and found it ran very slowly. I am not sure if an SQL view with the UNION statement would go faster. This is something I did not have time to experiment with.

Section 1 of Program 2 uses SQL to count the number of years for each variable to print in the report. This is used to anticipate where to put page breaks so the data for a time series for a single variable are not split across pages. SUBORDER is the order that the subject sections will print in the final report. This variable could have been put in the state table or obtained by reading a value label for the subject, but I just wanted to show you how easy it is to set up things like this "on the fly" in SQL.

Program 2 Section 1.

```

/* get the varnames to process */
/* yrs=number of yrs for each variable */
proc sql;
  create table v1 as
  select distinct varname, freq(varname) as yrs
  , subject, vorder
  from mwsas.state
  group by varname;
  alter table v1
    add suborder num;
  update v1
  /* suborder is the order that the subjects
  print */
  set suborder=1 * case
    when subject='P' then 1
    when subject='M' then 2
    else 0
  end;
/* order this the same way it will print */
create table vars as
select *
  from v1
order by suborder, vorder;
quit;

```

In the next section a SAS data step is used to set up a group of macro variables for all of the items that appear in the profile. This data step is performed once and, in the case of the sample profile, creates 3 macro variables VAR1, VAR2, and VAR3 which contain the names of the 3 items in the order they will appear in the profile report. In this case, VAR1 will contain TOTPOP, VAR2 is MED_AGE, and VAR3 is EMPLOYED. NVARS is the total number of variables to process.

Program 2. Section 2.

```

/* get all items to print on profile */
/* make an array of the variable names */
%macro makevar;
/* create macro variables for each varname */
/* and number of yrs for each variable */
/* note: items will be in the same order as
the final report variables */
%global nvars;
data _null_;
  set vars end=lastobs;
  call symput('var'||left(_N_),varname);
  if lastobs then call symput('nvars',_N_);
run;
%mend makevar;

```

Here is the kludge for concatenating potentially hundreds of tables to generate a profile for a single county.

Program 2. Section 3.

```
%macro printit;
/* the whole reason for this is that you
   can't put more than 16 files in a set
   statement */
/* this does &NVARs vars */

%let jend=%eval(&NVARs/15+1);
%do j=1 %to &jend;
%let en=%eval(&j*15);
%let st=%eval(&en-14);
%if &NVARs < 15 %then
%let en=&NVARs;

data c&j;
/* try to combine all the variable tables */
/* max of 16 files per set- we use 15 here*/
set
%do i=&st %to &en;
mwsas.&&VAR&I (where=(county=&CTY))
%end;
; /* end of set command */
run;
%end;

%if &en < &NVARs %then %do;
%let lst=%eval(&en+1);
data clast;
/* do the rest */
set
%do i=&lst %to &NVARs;
mwsas.&&VAR&I (where=(county=&CTY))
%end;
; /* this is the end of the set command */
run;
%end;

/* concatenate the groups of vars- hopefully
we won't have more than 16*15 (240 vars)*/
data mwsas.c&CTY;
set
%do i=1 %to &jend;
c&i
%end;
%if &en < &NVARs %then %do;
clast
%end;
; /* this is the end of the set statement */
run;
```

Now, back to SQL for a quick join to link together data from the county table, the state table, and the vars table created in Program 2 Section 1.

Program 2 Section 4.

```
proc sql;

/* join the county data with the state table
and vars table so we get the flags and sort
order etc */
create table cnty as
select
cty.county, cty.year, cty.ctyval, ks.subject,
ks.vorder, vars.suborder, ks.varname,
ks.units, ks.kansas, ks.nonmetro, ks.dec_plc,
ks.rank_flg, ks.pcap_flg, ks.ks_flg,
ks.nmet_flg, vars.yrs, cty.c_rank, cty.per_cap
from mwsas.c&CTY as cty,
mwsas.state as ks, vars
where cty.varname=ks.varname
and cty.varname=vars.varname
and ks.varname=vars.varname
and cty.year=ks.year
order by vars.suborder, ks.vorder,
cty.year;
```

```
quit;
```

Table CNTY has everything needed to create the county profile and it is all in the proper order for printing. I used a SAS data step and PUT statements to print the report, since this let me be most flexible in formatting the report. The code for the actually printing of the report is not shown here since it is just a series of PUT statements.

The final section of program 2 can be a loop if you are running profile reports for more than one county. The MAKEVAR macro is run once to set up the macro variable list. The %LET statement for setting the county code and the %printit macro can be repeated for each county profile to generate.

Program 2 Section 5.

```
%makevar
%let CTY=045;
filename rpt "profile.&CTY";
/* do the report for 1 county */
%printit
```

Here is the resulting report for the test data set.

1990-1991 Kansas Statistical Abstract: County Profile Report						Page 1
Douglas County, KANSAS						Per 1000
	Year	County	Kansas	Non-Metro Kansas	County Rank	or \$PerCapita
Section 1: Population						
P 1 DECENNIAL CENSUS:TOTAL POPULATION	1980	67,640	2,364,236	1,180,248	5	.
	1990	81,798	2,477,574	1,144,646	5	.
P 2 MEDIAN AGE	1990	25.8	32.9	.	104	.
Section 2: Employment and Earnings						
M 1 EMPLOYED (KDHR)	1990	42,780	1,243,000	545,324	5	523.0
(Place of Residence)	1991	43,396	1,238,000	543,578	5	.

COUNTY:county value. KANSAS:state value. COUNTY RANK:highest county value ranked #1.(\$000):thousands of dollars
 NON-METRO KANSAS: Kansas total less (Johnson,Miami,Wyandotte,Leavenworth,Douglas,Shawnee,Sedgwick,Butler,Harvey)
 PER 1000: For non-money amounts, PER 1000=(County value / County Population for the same year as the data) x 1000.
 \$PERCAPITA: (Computed in Dollars): (County Value / County Population for the same year as the data).
 Institute for Public Policy and Business Research, University of Kansas, Lawrence, KS 66045 11JUL92

SUMMARY

One does not become an SQL expert overnight. There are lots of different ways to do the same things using SQL. I cannot state any hard and fast rules on when to use SQL versus when to use SAS data step processing. A lot of it had to do with what I was most comfortable doing at the time. Much of my SQL coding came about out of my own curiosity. I would say to myself: "What would happen if I coded this in SQL" or even "I wonder if it is possible to do this in SQL?". I will summarize some general things that I have learned while getting used to SQL.

1. Code and test SQL statements in small pieces with really tiny test data sets that have only a few variables. Always print the results of any intermediate tables produced after even the smallest change in the procedure.
2. Play around with the joins. See how much repetitive data you can get rid of. Of course, some things are better off being repeated because the overhead of the join isn't worth it, but playing with joins is understanding the heart of SQL.
3. Warning: one little mistake- such as leaving out a comma or a semicolon or even the words "proc sql;" can cause your program to hang in never never land for a long time before you get an error report. I have produced 20,000 line SAS logs with errors when all I did was forget a comma in a variable list! If things crank for too long, kill the job and check the log.
4. The summing and grouping features in SQL are very powerful, but check the results carefully; they may be summing a different group of records from what you planned. Make at least one hand calculation of a computed amount from the original data to be sure.

5. If you work with lots of data and SQL you will probably generate many, many tables. You can cut down on the tables by using views, but then you have to manage views. The numbers of tables, views and sql procedures to do the 'little things' could become an administrative nightmare to organize. But, if you are turning to SQL to assist in data management you probably are already in a nightmare situation with your data.
6. Some SAS programmers who love the data step will tell you that SQL is inefficient and dangerous to use. Efficiency will become less of an issue when more people have access to fast systems like the RISC based Unix systems. In any event, the corporate world is moving to some flavor of SQL for database manipulation. SQL will probably be here for a while. We might as well get used to it now.

The complete software for the sample county profiles program is available from me via email. If you are not connected, send me a diskette in a self-addressed stamped mailer.

Trademark Notice:

SAS and SAS/SQL-DS1 are registered trademarks of SAS
 Institute, Inc. Cary, NC, USA
 Unix is a trademark of AT&T
 DEC is a trademark of Digital Equipment Corporation

Author: Arlene Slocum, Institute for Public Policy and Business
 Research, University of Kansas, 607 Blake Hall, Lawrence,
 Kansas 66045-2960 Phone: (913) 864-3701
 Internet: arlene@ukansvm.cc.ukans.edu