Systems Architecture Solutions for an STD Information System

Keith Humphrey, Centers for Disease Control and Prevention
Ray L Ransom, Centers for Disease Control and Prevention

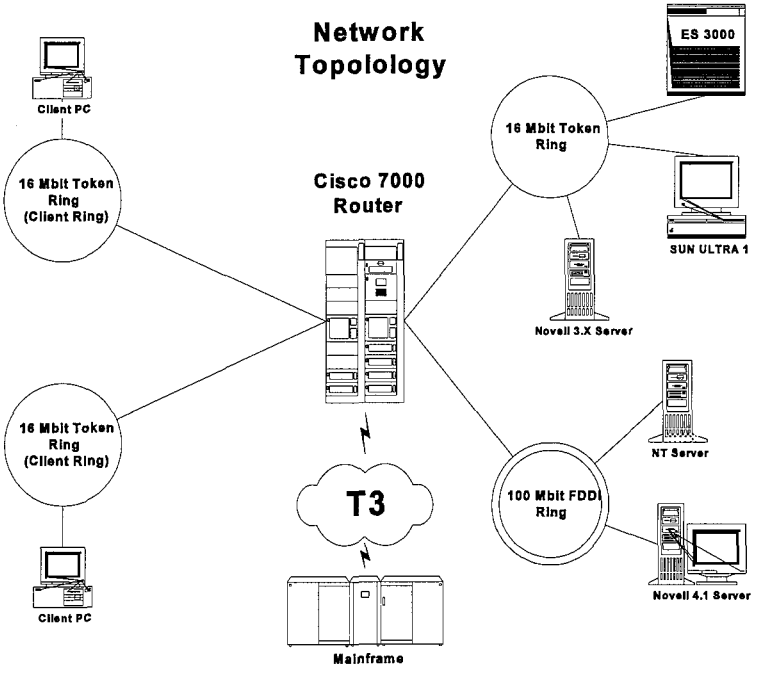
Abstract

The Division of Sexually Transmitted Disease Prevention (DSTDP) of the Centers for Disease Control and Prevention (CDC) collects data on occurrence of STD's and disseminates this information throughout the world public health community. DSTDP downsizing and the resulting need to maximize resources are necessitating the move of programmers into application development roles. Many programmer person-hours are spent on *ad hoc* requests for specific subsets and basic analyses on relatively static surveillance data tables. The tables currently reside on an IBM MVS mainframe. Many epidemiological and behavioral researchers and statisticians in the Division, hereafter referred to as "users", frequently wished to query these tables, but were unfamiliar with the mainframe environment. Therefore, they would request a subset of tables to be downloaded to the SAS® System for the Microsoft Windows® environment. Realizing that these requests were prime candidates for an application, programmers began to plan for an STD surveillance data access and analysis application.

Analysis and design of an STD information application (STDINFO) began in early 1996. Due to extensive experience with the SAS System, existing licenses and the functionality of SAS/AF® (especially the new FRAME technology), the SAS System was the chosen development platform. In addition to application development, another need surfaced at about the same time. As sophistication of analyses increased and multivariate modeling of sexual behavior and demographic variables became common methods, the ability to process relatively large tables using process intensive SAS Procedures required more than our current desktop abilities provided. Using the SAS System on a Windows client remote submitting to a higher-end processor was the obvious solution. The mainframe was considered, but fluctuating queue times made real time processing values unpredictable. A SUN ES3000® enterprise system was purchased and dedicated to SAS application and high-end statistical processing.

The LAN staff were challenged with providing a systems architecture solution that would best utilize current resources for Division programmers. The latest hardware and software technologies were evaluated by comparing real-time values for several distributed applications environments. Processing resources included a diverse LAN/WAN incorporating Windows 3.1, Windows 95 and NT workstations, a Windows NT Server and SQL server, a SUN ES3000 system, an IBM MVS mainframe and a Novell network file server. Emerging technologies researched included the new Scalable Performance Data Server (SPDS®), SUN UltraSPARC® 167MHz processors, pentium pro desktops and state-of-the-art network topology. This paper will discuss the analysis and decision process citing results and the final design decision. We will discuss our methodology and hopefully this information will be useful to other groups of varying needs and resources.

Resources



Client Hardware and Software

- Low end client — 486/50 MHZ PC/16M Ram — ISA Token Ring adapter
- High end client — Pentium Pro 200/32M Ram — PCI Token Ring adapter
- Windows 95 OS
- The SAS System, Version 6.11 Wave 2 loading from a Novell Application server

Server Hardware and Software

- Sun ES3000 Solaris 2.5.1 — 256M RAM — 167 MHz Risc processors
- Pentium 66 NT Server 4.0 — 133M RAM
- Intel 486/50 Novell Netware 3.X — 133M RAM

Network

- 16 Mbit Token Ring for local client rings
- 100 Mbit CDDI/FDDI ring for center campus backbone
- Cisco 7000 routers
- T3 WAN
- Secondary 16 Mbit Token Ring for center campus backbone
- SUN Ultra 1 as backup for ES3000
- IPX/SPX and TCP/IP protocols

SPDS Setup

The SPDS is a client/server software package that will allow the SAS System to take advantage of a multiple processor system. SPDS is available for several OS platforms such as Sun Solaris 1 and 2, AIX, HP-UX, Microsoft Windows 3.11, Windows 95, NT Workstation and OS/2.

Client Setup

The client software is copied to the !SASROOT/SPDS directory. The following modifications need to be made to the SAS invocation command:

```
path    !SASROOT/SPDS
sasmsg !SASROOT/SPDS
```

This will allow SAS to load the proper program files when it recognizes an SPDS reference in the LIBNAME statement. The proper protocol to access SPDS must be loaded on the client for SPDS to communicate.

Server Setup

Once the SPDS software is installed, the system must be configured to service incoming requests from SPDS clients. The SPDS communication is built around the RPC (Remote Procedure call) API. A remote procedure call is a client/server interface that allows clients to process on remote systems. If the server is running on a Unix system, the only information that the clients need to know to access the remote server is the IP address and the port address where the procedure is mapped. The SPDS has two mandatory processes, the SPDS name server and the server process. The ODBC process is only needed when the client uses the ODBC interface. The SPDS LIBNAME engine on the client connects to the SPDS name server process on the Unix system. The name server must be on a Unix non-well known port that does not conflict with another service on the system. The */etc/services* file is where the name server and ODBC services are recorded. Edit */etc/services* and add the following lines:

```
spdsname        5001/tcp #SPDS name server
spdsodbc        5002/tcp #ODBC if needed
```

Usually port numbers over 5000 are non-well known port and can be selected without a conflict. Once the services have been registered the process can be initialized. This is done by running the rc.spds script file located the *spds/site* directory. Edit the file and make sure the INSTDIR parameter is set to the directory where SPDS is installed.

```
INSTDIR=/opt/SPDS
```

Execute the rc.spds script file and when it is complete check the processes by running the *ps -ef | grep spds* command. Examine the output to see if the spdsserv, spdsnsrv and spdsodbc (if selected) process are loaded. When the processes are initialized, startup information is recorded in log files located in the *SPDS/log* directory. Examine the log files to determine if the name server process loaded at port number you selected in */etc/services*. If you are remote submitting on the same system on which SPDS is located, be sure you have installed the client software to that system's SAS directory as described in the Client Setup section above. SPDS also allows clients access to data on the host by defining a libref in the file *SPDS/site/libnames.parm*

Example syntax: libname=bench pathname=/data/bench;

and then using this same name in a LIBNAME statement in the client session.

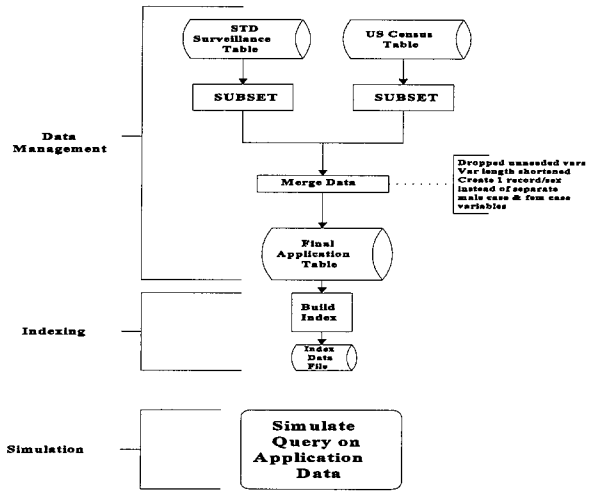
Example syntax: LIBNAME *d1* SASSPDS 'bench';

Architecture Analysis Methods

Going into the analysis, DSTDP development staff made a few key assumptions which were clearly supported by the literature including SAS documentation and previous SUGI presentations. Other assumptions made were based on the needs and requirements of DSTDP's research staff. The first assumption is that indexing tables on key variables is more efficient than constant resorting of tables because of the reduction in processing time due to decreased I/O requirements. Another assumption was that SQL might provide an advantage in AF development because of the ability of SAS Screen Control Language to submit SQL statements directly to the SQL processor without loading the SQL Procedure, thus eliminating the need for submit block processing. Also, all tables had to be located in an environment accessible by all network clients. Finally, the application processing has to run at speeds similar to other querying applications at CDC to be well received by the users.

Two different programs were used for evaluating performance times across systems architecture platforms. They both perform the same basic operations (figure 1), but differ in methodology and syntax. The first program BENCH1.SAS (Appendix 1) is very similar to the original

Fig 1 **SAS Program Flowchart**



mainframe programs used for previous *ad hoc* queries except that tables accessed by the program are now indexed on key variables and *Where clause* and *BY-group processing* are used to improve efficiency and reduce processing times. The other program, BENCH2.SAS (Appendix 2) uses SQL syntax entirely except for the indexing and printing steps. Since each program performs 3 basic steps:

1. Creation of table from master tables (data management)
2. Indexing of final table
3. Simulation of an application query

real time values were recorded in seconds for each of these 3 steps and measured across different systems architecture configurations for both programs and entered into a spreadsheet (Fig 2). The observed values and spreadsheet are discussed below.

Results

Fig 2

Architecture	Processing	Operating System	Program	Engine	Index Type	Data Manage-ment	Indexing	Simulation
Sas loading from Novell Server Data on Server	Pentium Pro 200 RAM=32 Meg	Windows 95	BENCH 1	V 611	B-Tree	260.34	61.85	21.31
Sas loading from Novell Server Data on Hard Disk	Pentium Pro 200 RAM=32 Meg	Windows 95	BENCH 1	V 611	B-Tree	84.11	20.82	4.65
Sas loading from Novell Server Data on SUN ES 3000	SUN ES 3000 (rsubmit)	Solaris 2.5.1	BENCH 1	V 611	B-Tree	26.412	14.083	2.304
Sas loading from Novell Server Data on SUN ES 3000	SUN ES 3000 (rsubmit)	Solaris 2.5.1	BENCH 1	SPDS	B-Tree	26.863	8.207	1.571
Sas loading from Novell Server Data on SUN ES 3000	SUN ES 3000 (rsubmit)	Solaris 2.5.1	BENCH 1	SPDS	Bit Map Index	26.757	78.356	0.965
Sas loading from Novell Server Data on Server	Pentium Pro 200 RAM=32 Meg	Windows 95	BENCH 2	V 611	B-Tree	386.36	59.71	19.81
Sas loading from Novell Server Data on Hard Disk	Pentium Pro 200 RAM=32 Meg	Windows 95	BENCH 2	V 611	B-Tree	307.54	20.48	4.16
Sas loading from Novell Server Data on SUN ES 3000	SUN ES 3000 (rsubmit)	Solaris 2.5.1	BENCH 2	V611	B-Tree	107.649	13.687	2.105
Sas loading from Novell Server Data on SUN ES 3000	SUN ES 3000 (rsubmit)	Solaris 2.5.1	BENCH 2	SPDS	B-Tree	110.814	8.221	1.384
Sas loading from Novell Server Data on SUN ES 3000	SUN ES 3000 (rsubmit)	Solaris 2.5.1	BENCH 2	SPDS	Bit Map Index	105.483	77.136	0.383
Sas loading from Novell Server Data on Server	486 50 MHz RAM=16M	Windows 95	BENCH 1	V 611	B-Tree	576.7	179.88	66.11
Sas loading from Novell Server Data on Hard Disk	486 50 MHz RAM=16M	Windows 95	BENCH 2	V 611	Bit Map Index	1890.16	172.68	65.9
Sas loading from Novell Client Data on NT 4.0 Server	Pentium 66MHz Ram=122 Meg	NT Server 4.0	BENCH 1	V611	B-Tree	104.48	24.92	12.5
Sas loading from Novell Client Data on NT 4.0 Server	Pentium 66MHz Ram=122 Meg	NT Server 4.0	BENCH 2	V611	B-Tree	683.48	35.43	12.15

Discussion

Although we include the results of data stored on the local drive of the clients to reflect the impact of network traffic to the process, this architecture would not allow all clients access to the same data. Though queries on some configurations posted comparable response times, note that bench2.sas queries consistently took less time than bench1.sas. However, bench2.sas was atrocious for the data management process. This suggests that SQL syntax is preferred for querying while it provides no advantages over traditional Data step/Procedure programming on subsetting and merging non-indexed tables.

With SAS loading from a Novell Application server and processing on PC Clients, query processing times ranged from 65.9 seconds on the low end clients to 21.31 seconds on the high end clients. By remote submitting to a SAS session on the SUN System, we can provide consistently fast queries (measured here as 2.105 seconds) to all clients. This configuration provides the fastest processing times without taking advantage of SPDS. Just by re-evaluating our code and taking advantage of indexing, Where clauses, and SQL features of the SAS System, we were able to achieve impressive improvements in processing time.

Efficient programming, as defined above, is necessary to recognize the advantages of SPDS. SPDS documentation stresses the advantages of indexes and Where clause, BY-variable, and SQL features of the SAS System. Therefore, simply installing the server on your system will not guarantee performance improvements and at times, as shown above, may actually slow certain processes down. The default index offered by the DATASETS Procedure® is a B-tree index. SPDS introduces a new index called a Bitmap Index. If a key variable to be indexed has low cardinality a Bitmap Index should be used. Low cardinality is defined as the number of unique values of this variable being less than a number between 100 and 1000. Although Bitmap Indexes take longer to create (creates a separate file for each variable), the SAS System reads these smaller files quicker and results in impressive query times. This is when we were able to see speed increases as reported by The SAS Institute® as 5-6 times faster than without SPDS. This is certainly an impressive difference in processing time.

Conclusion

This analysis process has been very beneficial to DSTDP. First the re-evaluation of programming techniques improved processing times substantially. This combined with dedicating a SUN system for SAS system processing provided all clients with impressive query times. Although SPDS reduces these times even more, the processing times seen without SPDS are considerably faster than any application we have seen at CDC. At $25,000 for the initial year for our system configuration and a yet undetermined yearly renewal cost, DSTDP cannot justify purchasing SPDS for the time differences. Since the cost of the SPDS license is based on the number of processors, it is inherent that purchasing SPDS makes future processor purchases more expensive. Since DSTDP's tables are relatively small and our processors are relatively few, we cannot fully realize the benefit of SPDS. If our application environment becomes one of many processors running at high utilization, we would definitely consider the SPDS. For organizations with gigabytes or terabytes of information, the benifit of SPDS is obvious.

References

SAS Institute Inc. (1996), *Installation Instructions for Scalable Performance Data Server Software (SPDS), Version 1, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1989), *SAS Guide to the SQL Procedure, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1994), *SAS Screen Control Language, Reference, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1996), *Scalable Performance Data Server User's Guide, Version 1, First Edition*, Cary, NC: SAS Institute Inc.

Acknowledgments

The authors thank the following people at CDC for their contributions to this paper:
Sharon Hixon, Van Munn and Brenda Sullivan

Keith Humphrey
Centers for Disease Control and Prevention, MS E-63
1600 Clifton Road
Atlanta, GA 30333
kbh3@cpsstd1.em.cdc.gov

Ray L Ransom
Centers for Disease Control and Prevention, MS E-02
1600 Clifton Road
Atlanta, GA 30333
rlr1@cpsstd1.em.cdc.gov

Appendix 1

```
/*****************************************************************/
/* NAME: BENCH1.SAS                                              */
/* PURPOSE: Prepare data table for STDINFO application and create */
/* index on key variables.  Finally, this program will simulate one */
/* query from the application.  This query will subset table and  */
/* calculate disease rates by requested BY variables.            */
/*****************************************************************/
/* AUTHOR: Ray L Ransom                                          */
/* DATE  : 12/31/96                                              */
/*****************************************************************/
/*******  MODIFICATION HISTORY  ************************************/
/* DATE     PROGRAMMER        DESCRIPTION                        */
/* -------- ---------------  ------------------------------------*/
/*                                                               */
/*****************************************************************/
options pageno=1;
/* table storage will vary according to architecture */
libname app 'c:\bench';
/* create efficient storage of variables */
/*DSTD 2638 Yearly STD Report Form Table */
data app.std(drop=totcase pctage pctrace pctboth areaname);
set app.std(where=(area=2 and source in(1 2) and fipscode<=56));
drop area;
attrib year length=4 disease race fipscode age source length=3;
if year>=80 then year=year+1900;  ****make 20th century dates;
/* US Census data for rate calculations  */
data app.pop(drop=totpop areaname);
set app.pop(where=(area=2 and fipscode<=56));
drop area;
attrib year length=4 race fipscode age length=3;
if year>=80 then year=year+1900;  ****make 20th century dates;
/* sort files before merging                                     */
proc sort data=app.std; by fipscode year age race;
proc sort data=app.pop; by fipscode year age race;
/*  merge std and population tables and keep intersection */
data app.data2638; merge app.pop(in=pop) app.std(in=std);
by fipscode year age race;  if std & pop;
/*  recode year to 4 digit, store variables in more efficient  */
/*  lengths, and create 1 record per sex with a new sex, cases */
/*  and pop variables                                          */
data app.data2638; set app.data2638;
drop fempop malepop malecase femcase;
attrib sex length=3;
label sex='Gender'
      cases='Number of cases of disease'
      pop='Number of persons in underlying population' ;
sex=1; cases=malecase; pop=malepop; output;
sex=2; cases=femcase; pop=fempop; output;
/*  sort data set by variables before indexing to reduce       */
/*  "randomness" of pointers in index to follow                */
PROC SORT; BY disease fipscode YEAR AGE race sex source;
/* index data on variables in order of most likely subsetting  */
/* selection.  This is the last step of the data management    */
/* that will be done by a cron job and scheduled according to  */
/* frequency of updates to master data files on mainframe      */
proc tables lib=app;
   modify data2638;
   index create data2638=( disease fipscode year age race sex source );
/* from here down, the program imitates a selection from the   */
/* STDINFO AF application.  Here the user has selected a subset*/
/* of syphilis data for black and white males, 15-39 years old */
/* from GA, FL, MS, AL, NC, SC, and TN for the years 1988-1995.*/
/* The select BY variables are fipscode year and age so rates  */
/* will be calculated for each stratum of this three way table.*/
/* subset selected data                                        */
DATA app.subset; SET APP.data2638
(where=(DISEASE=310 AND RACE IN(1 2) AND SEX=1 AND
FIPSCODE IN(13 12 28 1 37 45 47) AND AGE IN( 15 20 25 30 35) AND
YEAR IN (1988 1989 1990 1991 1992 1993 1994 1995)));
/* sum cases by BY-variables and create output table cases   */
PROC MEANS DATA=APP.subset NOPRINT SUM;
VAR CASES;
BY FIPSCODE YEAR AGE;
OUTPUT OUT=CASES SUM=CASES;
PROC SORT; BY FIPSCODE YEAR AGE;
/* sum pop by BY-variables and create output table pop       */
PROC MEANS DATA=APP.subset NOPRINT SUM;
VAR POP;
BY FIPSCODE YEAR AGE;
OUTPUT OUT=POP SUM=POP;
PROC SORT; BY FIPSCODE YEAR AGE;
/* merge case and pop tables and calculate rate.  We divide  */
/* the rate by 2 because there is one record per source but the*/
/* underlying population for each is the same.  If a user      */
/* selects only PRIVATE or only PUBLIC provider type then the  */
/* application will not divide population by 2.                */
DATA RATES; MERGE CASES POP; BY FIPSCODE YEAR AGE;
RATE=((CASES/(POP/2))*100000);
label rates='Disease rate per 100,000 persons';
/* print rates by select BY variables                          */
PROC PRINT; VAR fipscode year age CASES pop RATE;
RUN;
```

Appendix 2

```
/*****************************************************************/
/* NAME: BENCH2.SAS                                              */
/* PURPOSE: Prepare data table for STDINFO application and create */
/* index on key variables.  Finally, this program will simulate one */
/* query from the application.  This query will subset table and  */
/* calculate disease rates by requested BY variables.            */
/*****************************************************************/
/* AUTHOR: Ray L Ransom                                          */
/* DATE  : 12/31/96                                              */
/*****************************************************************/
/*******  MODIFICATION HISTORY  ************************************/
/* DATE     PROGRAMMER        DESCRIPTION                        */
/* -------- ----------------  -----------------------------------*/
/*                                                               */
/*****************************************************************/
options pageno=1;
/* table storage will vary according to achitecture */
libname app 'c:\bench';
/* create efficient storage of variables */
/* SQL Procedure to create table to be used by application from  */
/* DSTD 2638 Yearly STD Report form and US Census data (for rate */
/* calulations) tables*/
proc sql stimer;
/* merge tables and create subset of US states, pub and pvt provider type */
  create table work.newstd1 as
    select s.disease  length=3, s.fipscode length=3, s.year length=4,
           s.age length=3, s.race length=3, s.source length=3,
           s.malecase, s.femcase, p.fempop, p.malepop
        from app.std as s, app.pop as p
        where s.fipscode=p.fipscode and s.year=p.year
        and s.age=p.age and s.race=p.race
          and s.area=2 and s.source in (1 2) and s.fipscode<=56;
/* create table with one record per male with new cases,sex, and pop vars  */
/* and make year 4 digit 20th century years                                */
  create table work.males as
    select disease, fipscode, year, age, race, source,
           malecase as cases, malepop as pop
        from work.newstd1;
    alter table work.males add sex num(3);
    update work.males set sex=1, year=year+1900;
 create table work.males as select * from work.males;
/* create table with one record per female with new cases,sex, and pop vars*/
/* and make year 4 digit 20th century years                                */
  create table work.females as
    select disease, fipscode, year, age, race, source,
           femcase as cases label='Number of cases of disease',
           fempop as pop label='Number of persons in underlying pop'
        from work.newstd1;
    alter table work.females add sex num(3);
    update work.females set sex=2, year=year+1900;
  create table work.females as select * from work.females;
/* concatenate male and female tables into one                             */
  create table app.app1data as
    select disease  length=3 label='STD Code',
           fipscode length=3 label='FIPS Code',
           year     length=4 label='Reporting Year',
           age      length=3 label='Age Group',
           race     length=3 label='Race/Ethnicity',
           source   length=3 label='Reporting Source',
           sex      length=3 label='Gender',
           cases             label='Number of Cases of Disease',
           pop               label='Number of persons in Underlying Pop'
        from work.males
    outer union corresponding
    select disease  length=3, fipscode length=3, year     length=4,
           age      length=3, race     length=3, source   length=3,
           sex      length=3, cases, pop
        from work.females
    order by disease, fipscode, year, age, race, sex, source;
/* index data on variables in order of most likely subsetting  */
/* selection.  This is the last step of the data management    */
/* that will be done by a cron job and scheduled according to  */
/* frequency of updates to master data files on mainframe      */
proc datasets lib=app;
   modify app1data;
   index create app1data=( disease fipscode year age race sex source );
/* from here down, the program imitates a selection from the   */
/* STDINFO AF application.  Here the user has selected a subset*/
/* of syphilis data for black and white males, 15-39 years old */
/* from GA, FL, MS, AL, NC, SC, and TN for the years 1988-1995.*/
/* The select BY variables are fipscode year and age so rates  */
/* will be calculated for each stratum of this three way table.*/
proc sql stimer;
   create table work.rates as
      select fipscode, year, age, sum(cases) as totcase, sum(pop) as totpop
         from app.app1data
            where disease=310 and fipscode in(13 12 28 1 37 45 47)
             and year in (1988 1989 1990 1991 1992 1993 1994 1995)
             and age in (15 20 25 35) and race in(1 2) and sex=1
      group by fipscode, year, age;
   alter table work.rates
      add rate dec(8,2) label='Disease rate per 100,000 persons';
   update work.rates set rate=((totcase/(totpop/2))*100000);
   create table work.rates as
      select * from work.rates;
proc print data=work.rates;var fipscode year age totcase totpop rate;
run;
```