Paper 260-2009

Automate Dummy Variable Creating from Large Longitudinal RDBMS Tables --- A Pedagogical Example of the SAS/Connect® Piping Facility

Melissa Skanderson, VA Pittsburgh Healthcare System, Pittsburgh, PA
Xiaoqiang Wang, University of Pittsburgh, Pittsburgh, PA

ABSTRACT

Given a list of patient demographic characteristics and the patient lab records in RDBMS tables separated by years, we can create a list of summary variables per specification per person almost automatically using 3 parallel processes: process #1 selects all lab records only for the patients in our ID list and writes them into pipe #1; process #2 reads records from pipe #1, cleans lab results into appropriate numeric values and writes them into pipe #2; process #3 reads lab values from pipe #2 and aggregates them into person-level characteristics. Tricks of SAS/Access for OLEDB, regular expression, Hash object, and macro programming are illustrated too. We also discuss the advantages and limitations of our methods. This paper fills in the gap of the lack of a working example on the piping facility.

INTRODUCTION

VA MEDICAL DATA

At the VA, data is collected at each facility and transferred to a national database that is available to researchers. This paper is focused on the lab data, while our method can be applied on other types of data too. The lab files contain one record per person, per day, per lab with inpatient and outpatient data in separate tables by fiscal year. The patient identifier, common across datasets, is the variable SCRSSN. The number of records per file for fiscal year 2001 through 2006 is in the table below, where the 2nd and 3rd rows are the # of records in our data warehouse, inpatient and outpatient, respectively, and the 4th and 5th rows are the # of records to be selected for our test cohort of 436,726 subjects.

	2001	2002	2003	2004	2005	2006	All Years
Inpatient	20,437,865	22,497,163	24,423,976	31,020,196	32,563,079	38,336,414	169,278,693
Outpatient	68,952,442	82,777,951	94,613,667	108,401,326	117,194,510	152,371,988	624,311,884
Selected IP	1,513,850	1,921,066	2,103,567	2,238,530	2,307,208	2,180,509	12,264,730
Selected OP	4,341,442	5,208,076	5,965,452	6,689,988	7,033,834	7,496,893	36,735,685

Table 1. Sizes of lab tables and # or records to be selected.

For many projects, we're asked to generate person-level summary dummy variables such as diseased or not, # of occasions that a lab test is abnormal, the maximum values of a certain lab test, etc. from the lab tables (transactional tables) for a list of SCRSSNs (master table). The business logic behind these variables are very similar, thus macro programming is in general a great tool to create them in batch. But unfortunately those big transactional tables often reside in a remote relational database system and consist of records for all hospital uses (of a size at least 10 times as big as that of our target records). It takes hours just to select records of the target subjects out of the RDBMS. What's more, the selected records claim tens of gigabytes of hard drive space that clogs the file server, and may cause some data security issues. The SAS piping facility comes into play because it only bypasses records via a TCP socket without writing them to the hard drive. But this nice feature was only briefly covered in the SAS/Connect reference, and we cannot find a good working example in the literature.

PIPELINE PARALLELISM

According to [7], "pipeline parallelism is when multiple steps depend on each other, but the execution can overlap and the output of one step is streamed as input to the next step". SAS 9 introduced a LIBNAME Engine SASESOCK to facilitate data-flow from one process to another via a TCP/IP socket. The syntax of the piping facility is just as easy as

```
LIBNAME libref SASESOCK "port-specifier" <TIMEOUT=# of seconds>;
```

Where *libref* is a valid library name, *port-specifier* could be either a TCP port number (such as :7001 locally or 10.37.214.222:7001 for a port of machine with IP address 10.37.214.222) or an alias (such as :pipe1 locally or 10.37.214.222:pipe1 remotely) of a TCP port registered in file /etc/services under UNIX or %WINNT%\system32\drivers\etc\services under Windows XP/Server. There're some restrictions for SASESOCK pipeline.

- There must be only one process writing to the pipeline, and only one process reading from it.
- The pipeline can be viewed a queue such that one process always writes to the bottom and the other process reads from the top, and it has only limited capacity, so
    - One record in the pipeline is only written once and read once.
    - If the reading process consumes records faster than what the writing process can produe, then the reading process will pause for more records before TIMEOUT reaches.
    - If the writing process has produced more records than what the writing process can consume, then the pipeline will block the reading process from populating more records before TIMOUT reaches.

  In other words, processes linked via pipelines are sequentially dependent on each other.

As a result, only the DATA step and a few procedures (refer to [7]) are supported by the piping facility. Still, the pipeline parallelism enhances efficiency in certain ways as discussed in [5], [7], and [8]. We also noticed a nice feature that one process can write multiple datasets into a pipeline for another process to read, as long as reading and writing in the same order.

PREPARATION

In our example, we're going to create two types of person-level dummy variables based on lab test values from fiscal year 2001 through 2006. These variables are going to be part of the statistical models in the future, either directly or after combination. Each variable of Type 1 basically counts the # of times that a lab value satisfies a certain condition within a certain period of time; while each variable of Type 2 selects a lab value satisfying a certain criterion within a certain period of time. There are some examples in the 2 tables as follows.

Variable	Time w.r.t. baseline	Lab #	Condition	Description
sercrt20_c	±∞	31	result>2.0	# of times that Serum Creatinine > 2.0 within any time
alt40_c	±24 months	45	result>40	# of times that ALT > 40 between baseline – 24 months and baseline + 24 months
hgb13_c	(-∞, +31]	1	result<13 and sex='M' or result<12 and sex='F'	# of times that HGB<13 for men or <12 for women from ever to 31 days after baseline

Table 2. Some Type 1 Variables

Variable	Time w.r.t. baseline	Lab #	Criterion	Description
hgb	-12/+24 months	1	min(abs(date-baseline))	Hemoglobin closest to baseline
sercrtpr	(-∞,0)	31	min(baseline-date)	Serum Creatinine closest to baseline but before baseline
hivrnahi	±∞	38	max(result)	Highest HIV Viral Load any time

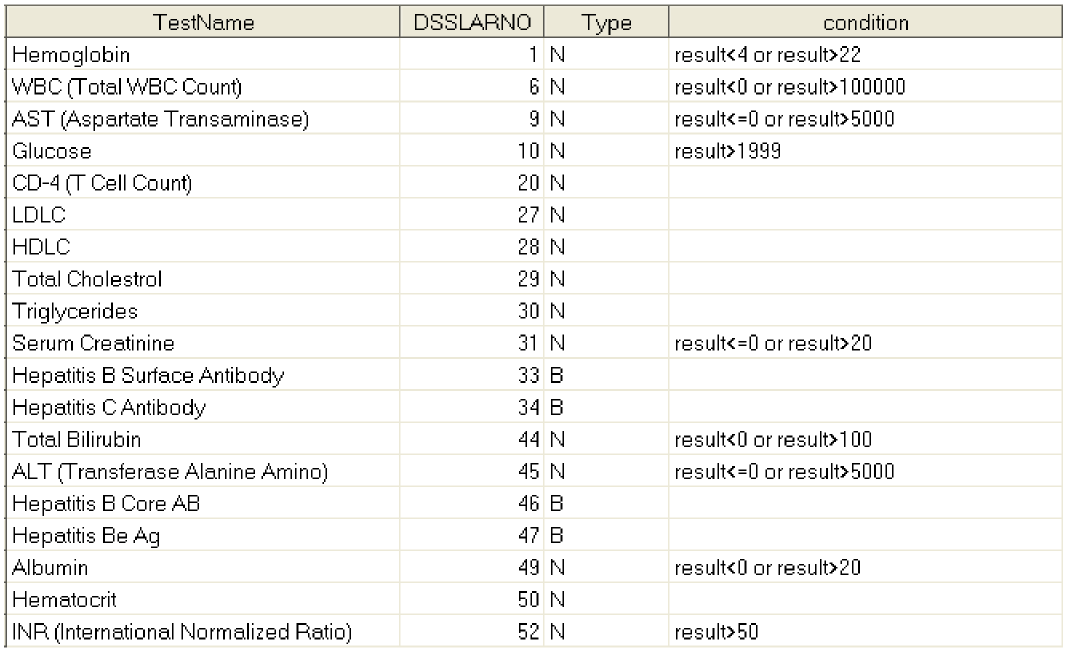
Table 3. Some Type 2 Variables

Then we sort out all the inputs and outputs

INPUTS

After analyzing the requirements, we figured out there were 4 parts of input,

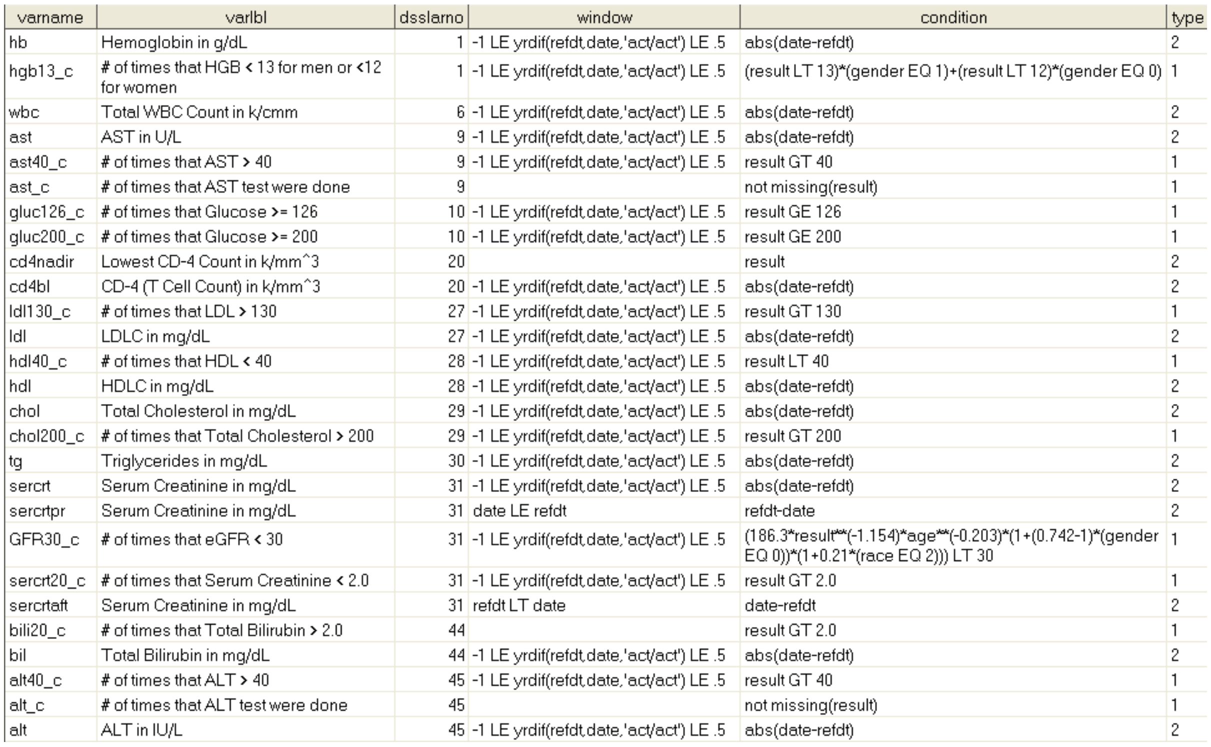
1. Cleaning directions --- table of the rules to clean text lab results.



TestName	DSSLARNO	Type	condition
Hemoglobin	1	N	result<4 or result>22
WBC (Total WBC Count)	6	N	result<0 or result>100000
AST (Aspartate Transaminase)	9	N	result<=0 or result>5000
Glucose	10	N	result>1999
CD-4 (T Cell Count)	20	N	
LDLC	27	N	
HDLC	28	N	
Total Cholestrol	29	N	
Triglycerides	30	N	
Serum Creatinine	31	N	result<=0 or result>20
Hepatitis B Surface Antibody	33	B	
Hepatitis C Antibody	34	B	
Total Bilirubin	44	N	result<0 or result>100
ALT (Transferase Alanine Amino)	45	N	result<=0 or result>5000
Hepatitis B Core AB	46	B	
Hepatitis Be Ag	47	B	
Albumin	49	N	result<0 or result>20
Hematocrit	50	N	
INR (International Normalized Ratio)	52	N	result>50

Figure 1. Instructions to clean text lab results

2. Variable specifications --- table of the detailed specifications of 32 variables. (Figure 2 shows the first 27 of them)



varname	varlbl	dsslarno	window	condition	type
hb	Hemoglobin in g/dL	1	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
hgb13_c	# of times that HGB < 13 for men or <12 for women	1	-1 LE yrdif(refdt,date,'act/act') LE .5	(result LT 13)*(gender EQ 1)+(result LT 12)*(gender EQ 0)	1
wbc	Total WBC Count in k/cmm	6	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
ast	AST in U/L	9	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
ast40_c	# of times that AST > 40	9	-1 LE yrdif(refdt,date,'act/act') LE .5	result GT 40	1
ast_c	# of times that AST test were done	9		not missing(result)	1
gluc126_c	# of times that Glucose >= 126	10	-1 LE yrdif(refdt,date,'act/act') LE .5	result GE 126	1
gluc200_c	# of times that Glucose >= 200	10	-1 LE yrdif(refdt,date,'act/act') LE .5	result GE 200	1
cd4nadir	Lowest CD-4 Count in k/mm^3	20		result	2
cd4bl	CD-4 (T Cell Count) in k/mm^3	20	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
ldl130_c	# of times that LDL > 130	27	-1 LE yrdif(refdt,date,'act/act') LE .5	result GT 130	1
ldl	LDLC in mg/dL	27	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
hdl40_c	# of times that HDL < 40	28	-1 LE yrdif(refdt,date,'act/act') LE .5	result LT 40	1
hdl	HDLC in mg/dL	28	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
chol	Total Cholesterol in mg/dL	29	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
chol200_c	# of times that Total Cholesterol > 200	29	-1 LE yrdif(refdt,date,'act/act') LE .5	result GT 200	1
tg	Triglycerides in mg/dL	30	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
sercrt	Serum Creatinine in mg/dL	31	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
sercrtpr	Serum Creatinine in mg/dL	31	date LE refdt	refdt-date	2
GFR30_c	# of times that eGFR < 30	31	-1 LE yrdif(refdt,date,'act/act') LE .5	(186.3*result**(-1.154)*age**(-0.203)*(1+(0.742-1)*(gender EQ 0))*(1+0.21*(race EQ 2))) LT 30	1
sercrt20_c	# of times that Serum Creatinine < 2.0	31	-1 LE yrdif(refdt,date,'act/act') LE .5	result GT 2.0	1
sercrtaft	Serum Creatinine in mg/dL	31	refdt LT date	date-refdt	2
bili20_c	# of times that Total Bilirubin > 2.0	44		result GT 2.0	1
bil	Total Bilirubin in mg/dL	44	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2
alt40_c	# of times that ALT > 40	45	-1 LE yrdif(refdt,date,'act/act') LE .5	result GT 40	1
alt_c	# of times that ALT test were done	45		not missing(result)	1
alt	ALT in IU/L	45	-1 LE yrdif(refdt,date,'act/act') LE .5	abs(date-refdt)	2

Figure 2. Variable Specifications

3. A demographics SAS dataset with the unique ID scrssn (char(9)) together with other necessary variables such as refdt (SAS date), gender (1 for male or 0 for female), age (in years) and race (1=white, 2=black, 3=hispanic, 4=other).

4. 12 SQL Server tables --- LAR_IP_FY01 through LAR_IP_FY06 for inpatient labs, LAR_FY01 through LAR_FY06 for outpatient labs. Every LAR table has at least 10 million records. They were recorded chronologically without a simple index on SCRSSN or a composite index with SCRSSN in the first position. We are going to use the columns listed in Table 3.

Column	Description	Type
SCRSSN	Scramble social security number, which is the unique ID per person	char(9)
DSSLARNO	Lab test id, for example, 1=hemoglobin, 31=serum creatinine	Integer
SVC_DATE	Date of test	Datetime
RESULT	Lab result as a text string, for example, “13.0”, “POS”, “NEGATIVE”	char(10)

Table 4. Columns of interest in the LAR tables

DESIRED OUTPUT

We’ll create a person-level dataset with unique ID with all the counters and critical values.

ASSUMPTIONS

We assume that,

1. We’re programming in SAS 9.1.3 under Windows XP.
2. The remote RDBMS is SQL Server 2005. We’ll use SAS/Access for OLE DB to connect to the database.
3. We’re granted only read access to the SQL server database.
4. No index on scrssn was created on any of the LAR tables.

MAIN PROGRAM DISSECTED

We need 3 steps to accomplish our task: step 1, select lab records for the subjects with given SCRSSNs’; step 2, convert messy text lab results into numeric values; step 3, generate person-level summary variables. By using 3 parallel processes and pipes, data flows from one process to the next without creating intermediate datasets. The figure below illustrates the system architecture.

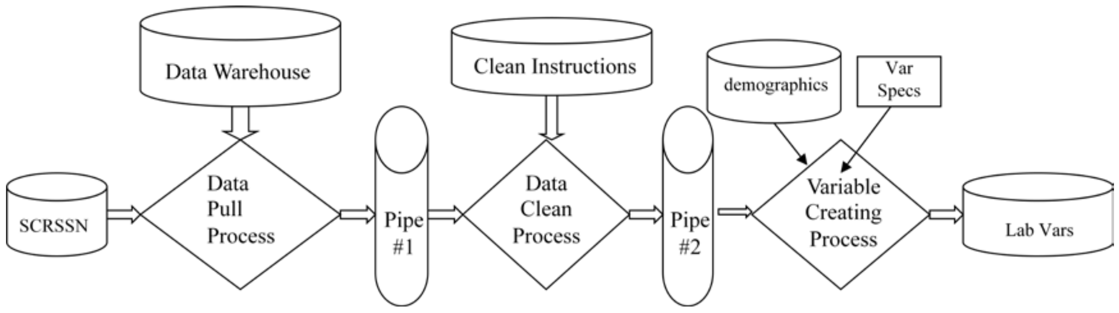


Figure 3. System Architecture

The challenges we face in each step are, (1) querying efficiently, (2) converting messy text lab results into proper continuous or binary values, and (3) generating person-level summaries with one-pass of data and without sorting by SCRSSN.

We’ll address the key points in the 3 processes.

PROCESS #1

In the 1st process, the given list SCRSSNs’ is uploaded into the SQL Server before it’s joined with each of the 12 lab tables.

```
signon[1] task1 sascmd=’!sascmd’;
%syslput[2] start_year = &start_year / remote=task1;
%syslput end_year = &end_year / remote=task1;
%syslput timeout = &timeout / remote=task1;
rsubmit[3] remote=task1 wait=no inherit[4]=(work=locwork);
libname sqlsrv oledb init_string=”Provider=……” connection[5]=shared;
libname outLib sasesock[6] ‘:3006’ timeout=&timeout[7];
data sqlsrv.’#scrssn’n[8] (insertbuff[9]=2000);
    set locwork.demographics(keep=scrssn);
%macro union_all(start_year=, end_year=);
%do fy=&begin_year %to &end_year;
    %let yr = %substr(&fy, 3);
    create table outLib.ip&fy as ……;
    create table outLib.op&fy as
    select a.scrssn, b.dsslarno, b.result as txtresult, b.svc_date, b.testunit
           from sqlslrv.'#scrssn'n as a, sqlsrv.lar_fy&yr as b
           where a.scrssn=b.scrssn and b.dsslarno in (&dsslarnolist);
%end;
quit;
%mend;
proc sql;
    select distinct dsslarno into :dsslarnolist[10] separated by ','
from locwork.vars_config;
%union_all(start_year=&start_year, end_year=&end_year)
quit;
endrsubmit;
```

Here we omitted the OLEDB connection string because it's too long. Also we omitted the code to create table *outLib.ip&fy* because it's similar to the creation of *outLib.op&fy*. We had chosen LIBNAME instead of explicit SQL Pass-Through to access the database because it's much harder to upload the SCRSSN's within the SQL Pass-Through facility. We don't have the right to bulk load, and we cannot insert multiple records per INSERT statement in SQL Server 2005. By using the LIBNAME option, we'll let SAS choose the SQL statements send to the database. The following option displays the SQL statements generated by SAS in the log.

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
```

For the sake of simplicity, we always signon to the local machine using the sascmd= option, but one can easily change it to connect to a remote server.The marked points,

1. Before submiting a program as a separate process, we must sign on to either a remote SAS server or the local SAS. For simplicity, all 3 processes here connect to the same PC as the parent session. For examples to connect remotely, refer to [6].
2. Because the parent session and the 3 processes all have their own workspace including the macro symbol table, %syslput has to be used to pass a macro symbol from the parent session to the other 3 processes.
3. What's enclosed between 'rsubmit' and 'endrsubmit' is a program to be run by the remote SAS server specified by the REMOTE= option.
4. 'inherit=(work=locwork)' will make the work library of the parent session accessible within process #1 by library name 'locwork'.
5. The CONNECTION=SHARED option instructs SAS/ACCESS not to issue a new connection for the update operations. (refer to page 25 of [2])
6. SASESOCK is the engine name for the piping facility.
7. By default, the TIMEOUT= option is 10 seconds, which is not always enough time to upload all SCRSSN's into the SQL Server.
8. Remember that we don't have the right to write into the SQL Server except the tempdb. And the name of a temporary table must begin with a pound sign via OLEDB. (refer to page 16 of [1])
9. The INSERTBUFF= option is set at 1 by default. According to our experience, a number between 1000 and 2000 is optimal.
10. Macro symbol *&dsslarnolist* is generated as a comma-separated list of DSSLARNOs needed.

PROCESS #2

In the 2nd process, records are read out of pipe #1, then converted into either continuous or binary SAS numeric values depending on the type of each lab test. The numeric values are then passed into pipe #2. The types of all lab tests and their cutoff points are provided by SAS dataset 'clean_directions' in the WORK library of the parent session.

```
signon task2 sascmd='!sascmd';
%syslput start_year = &start_year / remote=task2;
%syslput end_year = &end_year / remote=task2;
%syslput timeout = &timeout / remote=task2;
rsubmit remote=task2 wait=no inherit=(work=locwork);
libname inLib[1] sasesock ':3006' timeout=&timeout;
libname outLib sasesock ':3007' timeout=&timeout;
%macro cleanfile(start_year, end_year);
proc sql noprint;
    select dsslarno into :continuous_labs[2] separated by ','
           from locwork.clean_directions where type eq 'N';
    select dsslarno into :Boolean_labs[3] separated by ','
```

```
            from locwork.clean_directions where type eq 'B';
        select compress(put(count(*), best.)) into :N[4] from locwork.clean_directions;
        select dsslarno,condition into :dsslarno1-:dsslarno&N[5],:condition1-:condition&N[6]
            from locwork.clean_directions;
    quit;
    data outLib.all_labs(keep=scrssn dsslarno result date);
        re[7] = prxparse('/^(\+|-)?\d*(\.)?\d+/');
        re_neg[8] = prxparse('/(NEG)|(^(I|U)ND)|(^NO(T|N))|(^NDET)/');
        re_pos[9] = prxparse('/(POS)|(REA)|(^DET)/');
        do until(eof);
            set
            %do fy=&start_year %to &end_year;
                inLib.IP&fy  inLib.OP&fy
            %end;
            open=defer[10] end=eof[11];
            result = . [12];
            if dsslarno in (&continuous_labs) then do;
                call prxsubstr(re, txtresult, re_start, re_len) [13];
                if re_len>0 then result = input(substr(txtresult,1,re_len),10.);
            end;
            else if dsslarno in (&boolean_labs) then do;
                if prxmatch(re_neg, txtresult) [14] then result=0;
                else if prxmatch(re_pos, txtresult) [15] then result=1;
            end;

            select (dsslarno);
            %do k=1 %to &N;
                %if %length(&&condition&k) %then %do; [16]
                when (&&dsslarno&k) if (&&condition&k) then result = .;

                %end;

            %end;
                otherwise;
            end;

            if not missing(result) then do; [17]
                date = datepart(longdate);
                output;
            end;
        end;
        stop;
    run;
    %mend cleanfile;
    %cleanfile(&start_year, &end_year);
    endrsubmit;
```

You may notice that the 12 datasets generated by process #1 are opened one by one in a single data step and cleaned. The marked points,

1. *inLib* refers to TCP port # 3006, which is the output destination of process #1. A pipe must have exactly one process that writes to it and one process that reads from it.
2. Macro symbol *&continuous_labs* is a comma-separated list of continuous lab #'s.
3. Macro symbol *&boolean_labs* is a comma-separated list of Boolean lab #'s.
4. Macro symbol *&N* is the # of records in table clean_directions.
5. Macro symbols *&dsslarno1* through *&&dsslarno&N* are the unique lab #'s.
6. Macro symbols *&condition1* through *&&condition&N* are the outlier conditions with one for each lab.
7. *re* is the pattern for a decimal number matched from the beginning of a text string.
8. *re_neg* is the pattern indicating negativity.
9. *re_pos* is the pattern indicating positivity.
10. Option 'open=defer' defers the opening of one dataset. Without it, SAS will attempt to open all 12 datasets at the same time, but it is impossible because these 12 datasets are created one by one, sequentially.

11. Option ‘end=eof’ will generate a Boolean variable eof to indicate whether the end of the 12 input datasets has been reached.
12. Initialize the numeric *result* to missing.
13. The CALL PRXSUBSTR routine searches the variable *txtresult* with the pattern of a decimal number, returns *re_start* as the position of the start of the substring and *re_len* as the length of the substring that is matched.
14. The PRXMATCH function searches the variable *txtresult* with the pattern of negativity, and returns the position at which the match begins or 0 if not matched. The numeric *result* is 0 - if a match is found indicating a negative test result.
15. Similar to 14, if a match of positivity is found, then assign 1 to numeric *result*.
16. If a range for acceptable values was specified, then set *result* to missing if *result* lies out of the range.
17. Only non-missing *result* gets populated into pipe #2.

PROCESS #3

In the 3rd process, numeric lab values are read from pipe #2 before we aggregate them into person-level characteristics per specification. Like in the 2nd process, the variable specifications are given as a temporary dataset in WORK library of the parent session.

```
signon task3 sascmd=’!sascmd’;
%syslput timeout = &timeout / remote=task3;
rsubmit remote=task3 wait=no inherit=(work=locwork);
libname inLib[1] sasesock ‘:3007’ timeout=&timeout;
%macro doit(outset);
proc sql noprint;
    select compress(put(count(*), best.)) into :N[2] from locwork.vars_config;
    select varname, dsslarno, window, condition, type into
       :varname1-:varname&N[3], :dsslarno1-:dsslarno&N[4], :window1-:windows&N[5],
       :condition1-:condition&N[6], :type1-:type&N[7]
       from locwork.vars_config;
quit;
data &outset;
    set locwork.demographics;
    %do k=1 %to &N;
           length
           %if &&type&k eq 1 %then &&varname&k 3; %else &&varname&k 8 obj&k 8;; [8]
           %if &&type&k eq 1 %then retain &&varname&k 0;; [9]
    %end;
data temp;
    if 0 then set &outset;
proc contents data=temp noprint out=xxx(keep=name);
proc sql noprint;
    select cats(“’”, name, “’”) into :mycolumns separated by ‘,’ from xxx;
quit;

data _null_;
    if 0 then set temp;
    declare hash ht(dataset:”&outset”, hashexp:16);
    ht.definekey(‘scrssn’);
    ht.definedata(&mycolumns);
    ht.definedone();
    do until(eof);
           set inLib.all_labs end=eof;
           if ht.find(key:scrssn) = 0 then do;
           %do k=1 %to &N;
                  if dsslarno eq &&dsslarno&k then
                  %if %length(&&window&k) %then %str(if &&window&k then);
                  %if &&type&k eq 1 %then %do;
                  if &&condition&k then do;
                         &&varname&k++1;
                         ht.replace();
                  end;
                  %end;
                  %else %do;
                  if missing(obj&k) or (&&condition&k) lt obj&k then do;
```

```
                        &&varname&k = result;
                        obj&k = (&&condition&k);
                        ht.replace();
                    end;
                    %end;
                %end;
                end;   *end if ht.find()=0;
            end;   *end do until(eof);
            ht.output(dataset: “&outset”);
            stop;
    run;
    %mend doit;
    %doit(locwork.labvars);
    endrsubmit;
```

The marked points,

1. *inLib* refers to TCP port # 3007, which is the output destination of process #2.
2. Macro symbol *&N* is the # of records in table vars_config.
3. Macro symbols *&varname1* through *&&varname&N* are names of the variables to be created.
4. Macro symbols *&dsslarno1* through *&&dsslarno&N* are the underlying lab #'s corresponding to each variable.
5. Macro symbols *&window1* through *&&window&N* are the time windows for each variable. *&&window&k* is either empty or a logical expression.
6. Macro symbols *&condition1* through *&&condition&N* are either the conditions or the criteria for the variables. When *&&type&k*=1, *&&condition&k* is a logical expression and we should increment counter *&&varname&k* if *&&condition&k* is true. When *&&type&k*=2, *&&condition&k* is an arithmetic expression and we should update *&&varname&k* with the current lab result if *&&condition&k* becomes smaller.
7. Macro symbols *&type1* through *&&type&N* are the types of all variables. *&&type&k=1* for counter *&&varname&k*, or *&&type&k=2* means that variable *&&varname&k* is the lab result satisfying a certain criterion.
8. For each counter, we need only 1 numeric variable of length 3. But to select a critical value, we'll need two variables of length 8, one for the critical value and the other to keep the up-to-date smallest value of the objective function.
9. Counters are initialized to 0.

TEST RESULTS

Due to the lack of indexes, the running-time of the above program is not very sensitive to the # of SCRSSN's given. It takes 10 minutes and 21 seconds to generate the variables for 10,000 subjects using 15MB memory, or 21 minutes 42 seconds for 436,726 subjects using 494MB memory.

CONCLUSION

Our implementation with multi-processing and piping has some advantages over the traditional way:

1. It bypasses intermediate datasets for the subset of patients with given SCRSSN's, so saves hard drive space and the hassle of data security.
2. Shorter running-time. The running-time is cut by at least half due to faster I/O via piping instead of hard drives.
3. The source code is easier to manage, because everything now is packed in a macro that requires only 3 datasets as inputs and all the variables will be done as if it were one step.

Of course there're major limitations as well:

1. Because the piping facility allows only sequential one-pass of data, more complicated variables (that need at least two passes of data) cannot be derived by our method.
2. The program is really hard to debug with multi-processing and intensive macro programming. There's really an alternative way using SAS views so opening tables can be deferred till the last step. The views will also encapsulate codes by functionality thus clearly separate the 3 layers fore-mentioned. For example, instead of creating dataset outLib.op&fy for pipe #1, we can create a view as follows.

```
create view op&fy as
select a.scrssn, b.dsslarno, b.result, b.svc_date
from sqlsrv.’#scrssn’n a, sqlsrv.lar_fy&yr
```

```
    where a.scrssn=b.scrssn and b.dsslarno in (&dsslarnolist);
```

And later when we clean the lab results, we create a view again, so the actual processing of the first 2 steps will be deferred until the last step.

```
  data all_labs / view=all_labs;
    set
    %do fy=2001 %to 2006;
          ip&fy  op&fy
    %end;
    open=defer;
```

After all the change, the program is running just as efficient but appears much easier. But if the large raw data tables were SAS datasets, then our approach will have some advantage.

3. We used the DATA step Hash object, so our program is subject to the memory limit. Usually it's just fine, because normally the cohort won't be as big as the one we tested. Also by creating an index on SCRSSN in every SQL Server lab table, we can then sort & merge to eliminate the use of Hash object. After this paper was accepted, we finally talked our DBA into creating indexes on SCRSSN, and our program got an immediate boost, especially when the cohort size is below 50,000.
4. We haven't exploited all the potentials of multi-processing due to our system constraints. We tried to use one sequence of processes to create variables with data from fiscal year 2001 to 2003, and another parallel sequence of processes to create the same variables using fiscal year 2004 to 2006 data, but got no improvement in running-time because our database server doesn't support concurrent requests very well.

But anyway, it's a good program to show how the piping facility works.

If you're interested in our program, our source code is available at http://stat.wvu.edu/~xiawang/260-2009.zip.

REFERENCES

[1] SAS Institute Inc., SAS/ACCESS 9.1 Supplement for OLE DB, http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_91/access_oledb_7366.pdf

[2] SAS Institute Inc., SAS/ACCESS 9.1.3 for Relational Databases: Reference, http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/access_rdbref_9297.pdf

[3] SAS Institute Inc., SAS Programs and Macro Processing, http://support.sas.com/onlinedoc/913/getDoc/en/mcrolref.hlp/a002047087.htm

[4] SAS Institute Inc., Hash and Hash Iterator Object Attributes, Methods, and Operators, http://support.sas.com/documentation/cdl/en/lrdict/59540/HTML/default/a003143739.htm

[5] Russ Lavery, An Animated Guide to Basic Parallel Processing, http://www2.sas.com/proceedings/sugi30/224-30.pdf

[6] Michael G. Sadof, Connecting Simply with SAS/CONNECT, http://www2.sas.com/proceedings/sugi30/023-30.pdf

[7] M. Michelle Buchecker, Pipeline Parallelism Performance Practices, http://support.sas.com/rnd/papers/sugi30/pipeline.pdf

[8] Gerhardt Pohl, et al, Pipes and Threads: Performance Testing of Advanced Scalability Feature in SAS v9, http://www2.sas.com/proceedings/forum2007/196-2007.pdf

ACKNOWLEDGMENTS

The authors owe thanks to Sunil Gupta, the section chair, for his assistance of all regards. Also thanks Dr. Adeel Butt, whose big Hepatitis-C Virus cohort inspired our work.

RECOMMENDED READING

[1] Lois Levin, Methods of Storing SAS Data Into Oracle Tables, http://www2.sas.com/proceedings/sugi29/106-29.pdf

[2] SAS Publishing, SAS Certification Prep Guide: Advanced Programming for SAS 9

CONTACT INFORMATION

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

Melissa Skanderson or Xiaoqiang Wang
VA Pittsburgh Healthcare System (646)
7180 Highland Drive (151C-H
Work Phone: (412) 954-5257
E-mail: xiw38@pitt.edu