

PROC SQL: Why Use It When Simple IF/THEN Statements Work?

Susan Myers and Inga Allred
RTI International, RTP, NC

ABSTRACT

Efficiency is an important concern for computer programs that undertake large tasks, especially when the programs are used many times. We compare two methods for updating values in a large database. The first method uses standard SAS "datastepping" with IF statements. The second method takes advantage of some of the power available in the SAS SQL Procedure. A comparison of the two methods on real data sets encountered in operations on a large national household survey, with more than 16,000 records and over 3,000 fields, shows that striking improvement is achieved using the more sophisticated SQL tools.

INTRODUCTION

The SAS DATA Step automatically loops through the entire data set. However, there are occasions when one would like to leave the data set intact with all observations, modifying only a few variables in a small subset of records. The SAS SQL Procedure applies the Structured Query Language (SQL) within base SAS. SQL is a common language used to manipulate data within tables, also known as data sets in SAS. It offers a whole new set of tools to the programmer. In this paper, we will illustrate an approach that uses the Update Query in SQL.

BACKGROUND

Our data are collected on laptops for a large household survey. The identifying variable, QUESTID, is entered by the Field Interviewer (FI). The data are transmitted to RTI, and subsequently processed into SAS data sets. Therefore, each observation represents a survey respondent. A small subset of the variables, including respondent demographics, are broken out and sent to a team of Data Quality specialists. Since each QUESTID is associated with a specific household, the specialists are directed to verify whether the correct QUESTID was entered into the laptop. When errors are found, the specialists edit this identifying variable and an automated process appends a record to a file of QUESTID edit requests. The file is maintained with the following fields as illustrated in Figure 1:

- Original QUESTID
- New QUESTID
- Interview Start Date and Time
- FI Identification Number

Since different FIs could enter the same QUESTID, we determine unique records by the following three fields.

- Original QUESTID
- FI Identification Number
- Interview Start Date and Time

Our task is to update the SAS data set to reflect these edit requests. At the end of the quarter, this data set grows to over 16,000 records and 3,000 variables. Our first program developed to process the QUESTID edits utilized a DATA Step with an IF/THEN statement. While this accomplished our goal, the program took hours to run.

Our situation prohibits separating these records out before the edit and merging them back on for a couple of reasons. First, the variable being edited is the BY variable. Second, we need a complete data set after each edit, because the list of edits is dynamic. The input list of edits must be performed in the order they were added to the file, i.e., sequentially. Accordingly, we do not know initially which records would need to be processed separately.

Figure 1 illustrates why we cannot just subset our original data set for processing. In the first line, we are instructed to change QUESTID from 1111128 to 1111182. When we come to the third line, we need to reverse this edit. If we were to take our entire list and try to subset the master data set for those ids in the first column, some of the ids would not be present since they only exist after the edit is completed. The process needs to be entirely automated with no human intervention.

Figure 1. Input File of Requested Edits

Old ID	New ID	Interview	Time	FIID
1111128	1111182	01/07/2000	14:47:40	522060
9999999	9999998	01/16/2000	13:17:46	538427
1111182	1111128	01/07/2000	14:47:40	522060
1776382	1776874	01/14/2000	08:23:12	537890

The other consideration is the Quality Assurance of the resulting data set. By avoiding merges and subsets, we can be confident that we have the correct data in all resulting observations. Another benefit of either of our two programs is preventing the unnecessary data management steps another method would involve.

PROBLEM DESCRIPTION

Our first solution was to create a program with a basic DATA Step and IF/THEN statement. Although this method worked, it was extremely slow. With Analysts anxiously waiting for “clean” data, we knew we had to explore alternatives.

This runtime issue led us to investigate other programming methods, including SQL. The SQL Update Query seems to provide the perfect solution; accessing only designated records and replacing the value of one variable.

PROGRAM STEPS

The components of each program remain the same with only one changing from DATA Step to PROC SQL at step 5, alternatives indicated as a and b.

1. Index the data set
2. Read input file of edit requests
3. Put the fields into a table of macro variables
4. Loop through the edit requests
5. Edit the data set
 - a. Use IF/THEN statement
 - b. Use PROC SQL with an Update query

Step 1. Index

```
FILENAME editfile "c:\NoteQuestidupdate.txt";
LIBNAME cdrive 'c:\clean\';

%LET ds=Master;

OPTIONS ERRORABEND SYMBOLGEN;

PROC DATASETS LIBRARY=cdrive MEMTYPE=data;
  MODIFY &ds.;
  INDEX CREATE endx=(questid fiid tbegintr);
RUN;
```

Step 2. Read In Edit Requests

```
DATA cdrive.editfile ;
  INFILE editfile TRUNCOVER;
  INPUT @1 quest1 $char7.
        @9 quest2 $char7.
        @17 date $char19.
        @37 fiid $char6.;

RUN;
```

Step 3. Create MACRO variables

```
DATA NULL;
  SET cdrive.editfile END=last;
  CALL SYMPUT ('qa' || LEFT(_n_),
    TRIM(LEFT(quest1)));
  CALL SMYPUT ('qb' || LEFT(_n_),
    TRIM(LEFT(quest2)));
  CALL SMYPUT ('d' || LEFT(_n_),
    TRIM(LEFT(date)));
  CALL SMYPUT ('f' || LEFT(_n_),
    TRIM(LEFT(fiid)));

  IF last THEN
    CALL SYMPUT ('index', TRIM(LEFT(_n_)));
RUN;
```

Step 4. Loop through Edits

```
DATA NULL;
  CALL SYMPUT ('found', 1);
RUN;

%MACRO READfile (i);
  %DO %UNTIL (&i > &index or &found = 0);
    %chgqst (&qa&i, &qb&i,
      &f&i, &d&i);
    %LET i = %eval(&i+1);
  %END;

%MEND READfile;

/* Start the process ! */
%READfile (1);
```

Step 5a. SAS IF/THEN Version of Editing

```
%MACRO chngqst (q1,q2,fi,date);  
  
DATA cdrive.&ds.;  
SET cdrive.&ds. END=last;  
LENGTH datetime $19.;  
datetime=TRIM(LEFT(tbegintr));  
RETAIN found 0;  
finish=end;  
IF (TRIM(questid) = TRIM("&q1") AND  
(TRIM(fiid)=TRIM("&fi") AND  
(datetime=TRIM(LEFT("&date")))) THEN  
DO;  
questid=TRIM("&q2");  
CALL SYMPUT ('found',1);  
found=1;  
END;  
ELSE IF (finish=1) AND (found=0) THEN  
DO;  
%PUT &SYMPUT ('found',0);  
END;  
RUN;  
  
%MEND;
```

Step 5b. PROC SQL Version of Editing

```
%MACRO chngqst (q1,q2,fi,date);  
  
PROC SQL;  
UPDATE cdrive.s00001 SET QUESTID = "&Q2"  
WHERE (questid = "&q1") AND (fiid="&fi")  
AND (SUBSTR(LEFT(tbegintr),1,19)="&date");  
%IF &sqllobs=0 %THEN  
%DO;  
DATA NULL;  
CALL SYMPUT ('found',0);  
RUN;  
%END;  
  
%ELSE %DO;  
DATA NULL;  
CALL SYMPUT ('found',1);  
RUN;  
%END;  
%MEND;
```

It should be mentioned that using an index improved the performance of the PROC SQL, however it did not help the IF/THEN approach. A general rule of thumb is to use an index only if 15% or fewer observations are accessed. In our situation, we typically edit approximately .01% of the total records. By definition of an IF/THEN statement, all records are examined.

RESULTS

Both programs were run in the same environment, the same data set and file locations were used. Only step 5 changed between processes. The difference in the results of the two approaches was striking. With 100 edits to a file containing 16,676 observations and 2,941 variables, our processing time went from 2 hours using the IF/THEN approach to just 34 seconds using the SQL approach!

Figure 2 contains the excerpt from the log file for the IF/THEN technique, which took 1.14 minutes to complete a single QUESTID edit. The SQL Update query took .04 seconds for the same task (Figure 3). Because we typically must edit 200 records quarterly, this translates into a difference of two hours or more.

Figure 2. SAS Log with IF/THEN method

NOTE: DATA statement used:	
real time	1:14.26
cpu time	12.37 seconds

Figure 3. SAS Log with PROC SQL method

NOTE: PROCEDURE SQL used:	
real time	0.04 seconds
cpu time	0.01 seconds

SUMMARY

The typical SAS DATA Step would result in undesirable effects whether we try to subset the data set or use conventional IF/THEN statements. The subset, if we were to implement a Where clause in our SET statement would then result in a data set of one observation. Even if we were able to merge it back on to the Master data set, the sorting and merging would be CPU intensive. The alternative method that we originally used, IF/THEN statement, was confirmed to be inefficient.

By using the SQL Update query, we are able to access only observations that need editing. At the same time the data set remains intact with all original observations and variables.

CONTACT INFORMATION

Susan Myers, smyers@rti.org, (919) 541-7441
Inga Allred, irb@rti.org, (919) 541-7072

Each author can be reached at the mailing address:

RTI International
P.O. Box 12194
Research Triangle Park, NC 27709-2194.

TRADEMARK NOTICE

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.