Paper 333-2013

Nifty Tips For Data Change Tracking

Julie Kilburn and Rebecca Ottesen, City of Hope

ABSTRACT

Best practices for databases include keeping detailed audit trail information about the data. These audit trail tables vary in complexity as well as size. Generally speaking, the larger the database in tables (as well as in observations), the larger the audit trail. We have discovered that leveraging audit trail information in our automated reporting has been a huge resource saver in terms of which observations need to be reprocessed for a report. Even with minimal audit information (such as created by and modified by dates at the data table level), automation processing time can be greatly reduced by taking advantage of a new way of thinking and a few handy SAS® functions.

INTRODUCTION

Our system of SAS batch jobs that generated reports automatically on a regular basis had proven to be quite resource heavy. It became desirable to find more efficient ways to get the same reports. Using audit trail information available with each data table, such as the date a record was created or modified, we were able to not just process large data more quickly, but to reduce the amount of processing required to show the same results. This paper will discuss the issues that we faced and the solutions that we found.

THE PROBLEM

Our automated report process used complex algorithm programming to categorize patients into meaningful reporting groups. As can be seen in Table 1 our automation ran daily, monthly and quarterly, each having a different level of complexity in what the batch job produced. The explosive issue that we encountered was even for the daily runs our automated processing was taking too long. We realized that we were running data on tens of thousands of patients nightly even though they did not have a change in their data from the night before. The quarterly job was only run four times a year; however it was massive in scope and created issues of bumping into other batch jobs that the SAS server needed to handle.

Process	***Usage***	***Original Processing Time (hr:min)***
Daily	Daily patient based reports	3:30
Monthly	Creating data sets for internal use	6:58
Quarterly	Creating data sets for external institution use	13:07

Table 1: A Breakdown of the Automation Processes Run at Various Intervals.

THE SOLUTION(S)

The first solution we deployed was to schedule some of the batch automation jobs to run at times where they were not likely to impact other projects. For example, we moved the monthly and quarterly runs to the weekends, when there were not as many competing jobs running on the server. This helped but did not completely alleviate the processing time component.

When we reflected on our process we realized that no matter how we cleaned the code, modified DATA steps to use PROC SQL, or aligned programs to capitalize on already created temporary data sets, there was one fundamental issue that was holding us back. We were running this process, even daily, on patients whose data did not change from the previous day, or even the month, so their reporting status would also not have a change. We decided that a smarter strategy was to run our process only on those patients who had any change in data in the last 24 hours, regardless of what the changed data was since so many algorithms are deployed. We wanted to tackle the daily automation first since it is run every night, by revamping the code so that only those patients with a change would be sent through the complex algorithms. This would reduce the processing time by a significant amount by only running the algorithm programming on a few hundred patients rather than tens of thousands of patients.

DATE FIELDS

While the database had auditing capabilities before, they were not exactly as we needed. Therefore we added createdby and datecreated, modifiedby and datemodified fields to each table. These fields track when a record has been created and when it has been modified, as well as by which user. Using these fields for our new strategy worked well because we could run a quick query of all tables in the database to see if a patient had a data change in any of the tables in the last 24 hours, then reduce this to a patient level data set that flagged only those patients. However, there were also a couple of issues in using these date fields. The first was that they only track that there was an entry or modification in the table for a patient, not what field was changed. We didn't flinch too much at this because given the number of variables that go into our algorithms we didn't want to take the chance of missing one, so just knowing that the table changed was good enough. The second issue was that our FTP institutions do a complete refresh of data into the database every quarter. This meant that the datecreated and datemodified fields would always be the date of the quarterly FTP submission and would not change again until the next submission. The result was that once a quarter we were back to running every single patient for the FTP sites. However the web based institutions could still use the new 24 hour methodology, which still saved some time compared to the original method of running reporting for every patient. Also for every other night that was not a FTP submission *none* of the FTP site patients would be included in the 24 hour data set, again another time savings.

THE NEW PROCESS

The new process involves several steps. First we run a program to check the datecreated and datemodified fields to reduce the cohort down to only patients who had a change in the last 24 hours. Next we process only these 'change' patients through the algorithms to generate the proper report groupings. We then update the previous night's report groupings with the new changed grouping information only for patients with a 'change', while the other 'no change' patients keep their previous status and information. The change in time savings from this new process was nearly cut in half, as shown in Table 2.

Process	***Usage***	***Original Processing Time***	***24 Hour Processing Time***
Daily	Daily patient based reports, for one study	72 min	38 min

Table 2: A Breakdown of the Reporting Automation Process Including New Methodology.

THE DETAILS

The first step in our 24 hour process is to generate a data set of patients who have changed since the last run. Our daily automation runs Monday through Friday mornings. This means on Tuesday to Friday, we are looking for any changes within the last day, and on Monday, we are looking for changes within the last three days. To accomplish this we utilize a small macro to look for these changes.

```
DATA onlychanged; (1)
 LENGTH study 8 id 8;
 DELETE;
RUN;

%MACRO tablechange(table);

DATA tabletemp; SET &table (KEEP=study id datecreated datemodified);
 IF weekday(TODAY()) IN(2) THEN DO; (2)
   IF (datecreated>=(TODAY()-3)) OR (datemodified>=(TODAY()-3));
 END;
 ELSE IF weekday(today()) IN(3,4,5,6) THEN DO; (3)
   IF (datecreated>=(TODAY()-1)) OR (datemodified>=(TODAY()-1));
 END;
 KEEP study id; (4)
RUN;

PROC APPEND BASE=onlychanged DATA=tabletemp FORCE; (5)
RUN;

%MEND;

%tablechange(PatientStatus); (6)
%tablechange(PatientDemographics);
/* etc... */
```

1. Before the macro runs, we create an empty data set that will hold the patient identifiers for those that had changes. During each iteration of the macro, this data set will be appended with flagged 24 hour change patients from the data tables.

2. Use the weekday function to determine which day of the week today is. If today is weekday 2 (Monday), then look for dateCreated or dateModified within 3 days.

3. If today is weekday 3-6 (Tuesday to Friday), then look for dateCreated or dateModified within 1 day

4. Once the changed patients for the current table are identified, we no longer need to keep the dateCreated and dateModified variables, and we can limit our data set to the study and id that are required keys in our data tables.

5. As this macro will be run for each raw data table we will append the 24 hour results for the current data table to the existing storage data set.

6. We follow this up by running the macro for each data table in the database.

```
PROC SORT DATA=onlychanged NODUP OUT=onlychangedout; 7
 BY study id;
RUN;
```

7. There could easily be duplicate patients with a change across data sets, but we only need to have one record per patient to flag that there was a change. To accomplish this, while sorting, we reduce the data set to one patient per study.

The second step in our new process is to reduce all raw data tables that come directly from the database to just those patients who had a 24 hour change, or who are in the resulting onlychangedout data set from step 1. This is also best handled in another small macro that is run once for each raw data table.

```
%MACRO update(dsn);

PROC SORT DATA=&dsn; 1
 BY study id;
RUN;

DATA &dsn; 2
 MERGE onlychangedout (IN=a)
       &dsn (IN=b);
 BY id;
 IF a AND b;
RUN;

%mend;

%update(PatientStatus);
%update(PatientDemographics);
/* etc... */
```

1. First within the update macro, we sort the current data set to match the order of the keys of the changed data set to ensure proper merging.

2. Next we create a new data set by merging the current data set with the changed data set, keeping each observation only if it is in both.

Now that our raw data sets have been reduced to only those patients with a data change in the last 24 hours we can run the complex algorithm programming with just this small cohort. This algorithm is beyond the scope of this paper but the result is a SAS data set named analyzed that contains the reporting data for only the patients whose data was updated in the last 24 hours. Next we prepare to add this new information to the previous day's reporting file by keeping only the variables that are necessary to be updated in the final data set.

```
DATA prep4merge; SET analyzed;
```

```
 KEEP id study report grouping;
RUN;
```

We conveniently save the previous day's SAS data set for reporting to a permanent folder so that we can access it today. This data set has old information on the 24 hour patients that will need to be updated, as well as information on the no change patients which is considered current since their data did not change. Therefore, we only need to update the data that comes from prep4merge.

We then combine yesterday's data, with today's new updated data. We merge the respective data sets in this order (first the existing data, then the new data) so that when a patient that already existed previously has data that was updated in the last day, the new information will overwrite the old. The resulting output file is a SAS data set that contains the complete reporting output data

```
DATA xx.reporting;
 MERGE xx.reporting (IN=a) /*original reporting file*/
       prep4merge (IN=b) /*overwrites original with updated info*/;
 BY study id;
 IF a OR b;
run;
```

The result is actually a very simple data set that has information which is uploaded directly to the patient reports available in the database.

FURTHER COMPLICATIONS

While the time savings were impressive we were left with several unforeseen issues. After deploying the new method we realized that we had to accommodate several tricky situations. The first issue we needed to address was patients that became flagged as deleted from the database so that they could be taken out of the reporting. Our database allows users to flag a patient as deleted for analytic purposes; however these deleted patient records remain in the database since we allow reinstatement for certain scenarios. Therefore we needed to be sure that if a deleted patient hit the scene in the last 24 hours, we took them out of the base reporting file that we were appending to.

Next we found we had to deal with patients who, due to a change in their data, need to be removed from the report. These patients needed to be removed from a given report in the new 24 hour run, which meant that the corresponding 'old' reporting information also needed removed from the base data set.

The last issue that we discovered was that because our algorithm programming calls many other clinical algorithms, i.e., SAS macro programs, we needed to address scenarios where there was a change in one of the macros due to a change in the clinical algorithm. These changes typically consist of additional code and/or simply updating code, but to be conservative we would need to refresh the entire cohort nonetheless. We needed our program to identify if any of our SAS macro files were updated, and if so, to run the entire reporting programming on all patients once again. This was a much bigger issue and left us scratching our heads for several days.

PATIENT CHANGE SCENARIOS

To deal with the various patients scenarios that need updating for the 24 hour change we sat down to list all possible patient change scenarios for our database, as shown in Table 3. We realized that our 24 hour methodology only covered the first three scenarios and the final two still needed to be addressed.

	Patient Change Scenarios
☑	New patients
☑	Existing patients with no 24 hour change
☑	Existing patients with a 24 hour change that remained on a report
	Existing patients with 24 hour change that dropped off a report
	Existing patients who were flagged as deleted in the last 24 hours

Table 3: The Various Patient Change Scenarios That Needed to be Addressed.

To address this situation we modified our process.

```
DATA existing; (1)
 MERGE xx.reporting (IN=a) (2)
       onlychangedout (IN=b) /*changed id file*/;
 BY study id;
 IF a AND ^b; (3)
RUN;

DATA xx.reporting;
 MERGE existing (IN=a)
       prep4merge (IN=b)
       delete_today (IN=c); (4)
 BY study id;
 IF (a OR b) AND ^c; (5)
RUN;
```

(1) First we add a new DATA step before creating the final reporting data set to remove patients with any change regardless of what it is.

(2) xx.reporting was the original reporting data from the day before. Onlychangedout is the sorted data set which contains, per study and id, records for those with a 24 hour change.

(3) By limiting this new data set to records from the original file with no 24 change, we are deleting all patients that had a 24 hour change. This means any new patient, existing patients with a 24 hour change, or existing patients that dropped out of a report grouping are deleted.

(4) To create the new final reporting file we proceed as before by updating the newly modified old file with the new reporting information However we add in the delete_today data set which contains a flag for any patient that was deleted in the last 24.

(5) Finally we include only the existing patients with no 24 change in information and add in the information for the patients with a 24 hour change *who stayed in a reporting group.* We also remove any patients who were flagged to be deleted.

ALGORITHM CHANGES

Up to now we were happy. The 24 hour change process was working and we had addressed the various patient change scenarios that make someone fall in or out of a report grouping. However we became unhappy very quickly when we realized that we overlooked an important issue. What happens when one of the many algorithms changes? We would need to run all patients through the entire reporting programming again in case the change in algorithm resulted in a change in their status. It would be too complicated to try to run subsets of patients depending on which algorithm changed. We also realized that it would be complicated to rely on human intervention in terms of running the entire job manually; therefore we much preferred that SAS figure it out.

Our solution was to come up with a mechanism where SAS could analyze file modification dates and then make a decision about what to do. We started by reviewing our SAS macros to see if any have been modified in the last 24 hours. Following an example in the SAS Online documentation for the FINFO function, we run each filename through a new algorithm that looks at the properties of the file and grabs the filename and last modified date. We add these elements to a SAS data set, where each SAS program is an observation. Then if we find that any of the relevant SAS macros were changed, SAS is told to run the reporting grouping again for all patients, rather than the 24 hour subset.

```
DATA programlist; (1)
 LENGTH moddate 8 pgmname $ 60;
 FORMAT moddate date9.;
 DELETE;
RUN;

%MACRO pickdate(filepath);
DATA info; (2)
 LENGTH info_id info_value $60;
 DROP rc fid info_qty i close;
 rc=FILENAME("alg",&filepath);
 fid=FOPEN("alg");
 info_qty=FOPTNUM(fid);
```

```
  DO i=1 TO info_qty;
    info_id=FOPTNAME(fid,i);
    info_value=FINFO(fid,info_id);
    OUTPUT;
  END;
  close=FCLOSE(fid);
RUN;

DATA info2; SET info (WHERE=(info_id="Last Modified")); ❸
 moddate=DATEPART(INPUT(info_value,datetime18.));
 FORMAT moddate date9.;
 DROP info_id info_value;
RUN;

DATA info3; SET info (WHERE=(info_id="Filename")); ❹
 pgmname=info_value;
 DROP info_id info_value;
RUN;

DATA info4; ❺
 MERGE info2 info3;
RUN;

PROC APPEND BASE=programlist DATA=info4 FORCE;
RUN;
%MEND;

%pickdate("createdata.sas");
%pickdate("runformats.sas");
/* etc... */
```

❶ Create an empty data set to hold the file information for filename and modified date.

❷ The first DATA step comes from the FINFO documentation and basically stores external file information into a SAS data set so that we can manipulate it further using SAS. Our file contains two character variables: info_id (the name of the piece of file information) and info_value (the value of the piece of file information). Of the several pieces of available file information, we are only interested in the date last modified and the file name. (Note: A proc print of this file would show all of the information available.)

❸ First we take the observation where the name is ”Last Modified.” The value is a character representation of the date and time last modified, so we read it in as a date/time value, and only keep the date part.

❹ Next we take the observation that contains the SAS program’s file name from the same data set and create a variable to store it. We use this to verify which program changed by matching to the list of macro filenames.

❺ Finally we put this information back together and append it to our programlist data set for every time the macro is run on each SAS program that we need to evaluate.

Using a similar approach as proposed for finding changes in the last 24 hours, we can modify our code to look for changes in files in the last 24 hours in the programlist data set. We create a flag within the data set to identify the files changed within the appropriate timeframe, and use this to sort, and determine if there was any change at all in any file. From this we get a data set (programlist2) where observations (ie.SAS macro program files) with a 24 hour change, if any, are listed first. If the first observation contains a change_flag with a value of 1, there was at least one change. If the change_flag value is 0, then there were no file changes. In the data set programlist3, we create a macro variable with the change_flag value to signal if there was a last modified file change.

```
DATA programlist2; SET programlist;
 IF weekday(TODAY())=2 THEN DO;
  IF(TODAY()-moddate)<=3 THEN change_flag=1;
  ELSE change_flag=0;
 END;
 ELSE IF weekday(TODAY()) IN(3,4,5,6) THEN DO;
```

```
  IF (TODAY()-moddate)<=1 THEN change_flag=1;
  ELSE change_flag=0;
 END;
RUN;

PROC SORT DATA=programlist2;
 BY DESCENDING change_flag;
RUN;

DATA programlist3; SET programlist2;
 IF _N_= 1 THEN DO;
   CALL SYMPUT('algchange',change_flag);
 END;
RUN;
```

We now have a handy macro variable called &algchange that can be used to decide if SAS should run reporting on all patients because there was at least one macro program change (&algchange = 1). Otherwise SAS should only run the modified 24 hour change programming that takes into account all patient scenarios as described earlier because there were no macro programming changes.

CONCLUSION

Change is good but often leads to complexities that go unforeseen until they raise their ugly heads. In this case we continued to strive to improve our process and accommodate every possible situation that arose.

Currently we check a whole list of macros that may only impact certain parts of the automated reporting. Enhancements to this could be identifying which macros impact specific reports so that we do not have to run unnecessary patient cohorts. Expanding this methodology to other reporting could also improve the efficiency of our automation process and would involve looking at similar scenario and algorithm considerations as were covered in this paper.

REFERENCES

"Functions and Call Routines: FINFO Function." *SAS Help and Documentation v. 9.2*, SAS Institute Inc., n.d. Web. 20 Jul. 2012.

Kilburn J. and Ottesen R. (2012), Nuisances of Report Automation that Involves Change Tracking, Proceedings of the Western Users of SAS Software Conference

Ottesen R. and Kilburn J. (2012), Updating Reporting Data Sets for Recent Changes, Proceedings of the Western Users of SAS Software Conference

CONTACT INFORMATION

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

Julie Kilburn
City of Hope
Department of Biostatistics
1500 East Duarte Rd.
Duarte, Ca, 91010
jkilburn@coh.org