Paper: 113-2008

Migrating Information in the Amisys Data-Warehouse: Current to Advanced

by Anthony DeGance, Community Care, Tulsa, OK

Abstract

In the course of migrating tables from a current system to an advanced system, the migrated information must be verified. Inevitably this comes down to verifying corresponding information table by table, record by record, field by field. We used SAS's ® Proc Compare with the Out option to carry out the verification process. We report on some generic SAS query code used to do the table by table compare and on the DOS ® .bat syntax to launch the SAS query code batch-wise. And while the details of SAS query code may not be portable, the structure and various pieces of code developed for the process are.

We wrote and debugged queries for eighty tables over a period of six weeks to eight weeks. Evidently, making the comparison output manageable and presentable is essential to the analyst's effort. We managed the output by a combination SAS and DOS and directory structure. The eighty queries were launched by a peculiar DOS .bat syntax in batches of about a dozen or so. We tracked both execution time and table size in an effort to quantify the process. In the first pass through, the eighty queries took about sixteen hours over two days to execute, using a Pentium 4, 512 MB RAM, 1.9 GHz processor. In the last pass the queries ran in a day and took less than ten hours using a dual core, 3.25 GB Ram, 2.66GHz processor.

Introduction

Change is inevitable. How we manage that change reflects who we are.

Amisys Data Warehouse

This is the first paper in a series on migration problems and proposed solutions. Community Care is migrating its current Amisys Windows-based data warehouse to an Amisys Unix-based Oracle system. We sought a way to verify the tables in the current system corresponded to the migrated tables in the advanced system. We solved this problem by using SAS's Proc Compare table by table, comparing record by record, field by field.

Getting Started

There are some four hundred tables in the Amisys data warehouse. Some tables have as few as five or so fields with other tables having as many as one hundred or more fields. Some tables have as few as two hundred records with other tables having more than twenty million records. The task of data warehouse verification was enormous. We resolved this issue by selecting key tables and verifying that the key tables in the current system contained the same data as the corresponding tables in the advanced system. We did this using SAS's Proc Compare.

In order to use Proc Compare the current and advanced sampled tables had to be made comparable. This was a non-trivial task for three reasons. Firstly some field names changed and some field types changed, e.g. claim_ became claim_nbr, character fields changed to numeric. Additional fields were added to the tables of the advanced system while other fields were deleted from the tables of the current system. Secondly, we did not know what key variable or variables to sort the tables by to do the comparison. In many instances the determination of the key variable(s) was a trial and error procedure. Thirdly, many of the tables had in excess of a million records; for tables with in excess of 10,000 records, the average record size was 1.3 million records. Thus sorting, storing and managing of these tables were time-consuming.

We report on how we worked through these difficulties. We believe the method developed can be used outside our particular migration. The kernel of the method is a table by table comparison in which like records of the current table, Tempory.Old&Tble, and advanced table, Tempory.New&Tble, are selected randomly and compared. To carry this out in practice we construct a common base observation table, Tempory.Obs&Tble. We may think of the base observation table as a skinny Tempory.Old&Tble or Tempory.New&Tble; the table contains only the key variable(s) common to both Tempory.Old&Tble, and Tempory.New&Tble and record number. From the base observation table, a query table, Tempory.Query&Tble, of randomly selected records is formed using SAS's RanUni. Herein RanUni randomly

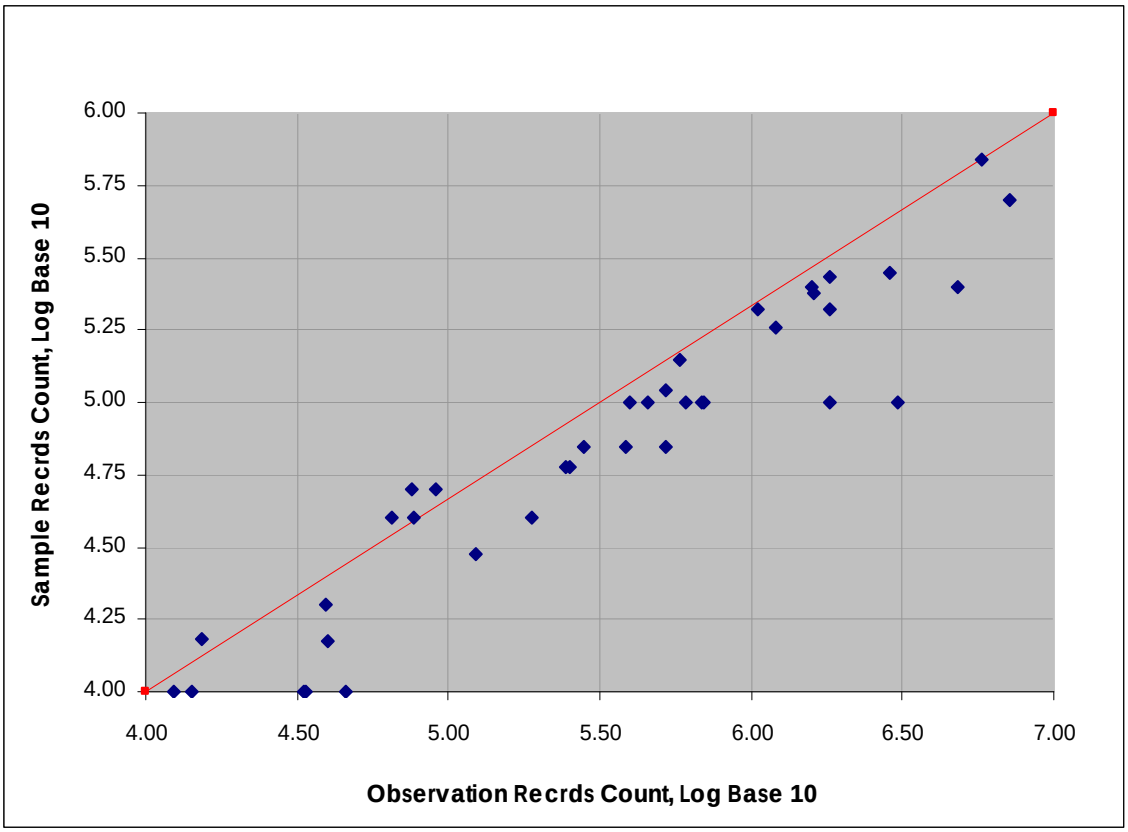
generates the record numbers of the selected records, an appropriate number of times, cf. the Sample Size Eq. With the query table in hand, a format of the key variable is constructed via Proc Format and used to make doable the means of record extraction from the current and advanced table. (The use of the format allows us to avoid sorting some potentially large tables.) With the extracted sample of current and advanced tables in hand, two things must still be done before we do the compare. Firstly, we must change the variable names and variable types in one of the extracted tables so that all the variables are comparable. Secondly we must do a certain amount of aligning so that the current and advanced tables are comparable record by record.

The question naturally arises, 'How large is the query table? How many records do we need to compare?' We give an un-statistical answer here. For tables with less than 10,000 records every record was compared, i.e. the observation table record count equals the query table record count. For tables with in excess of 10,000 records, we tried to balance the need for storing the current and advanced extracted tables as well as the cumbersomeness in sorting and doing the compare against the need for completeness. Initially we chose sample count based on observation count on a query-by-query basis. After several weeks of running queries and guesstimating sample size, we drew a straight line bounding the query-by-query data of sample count versus observation count on a log-log plot. The straight line allowed us to establish a simple equation for the sample record count as a function of the observation record count. In particular, we settled upon the Sample Size Eq., viz.

$$\text{Log}_{10}(\text{Sample Rcrds}) = 1.33333 + 0.666667 * \text{Log}_{10}(\text{Obs Rcrds})$$

We illustrate in Fig 1 the data of sample count versus observation count for forty or so queries performed as well as the Sample Size Eq. For an observation count of 10 million records, the Sample Size Eq. yields 1million records. We note that the Sample Size Eq. was not regressed but drawn to satisfy what we were comfortable with based on experience. As the power of PC's and storage increases, larger query tables will become acceptable.

Figure 1 Sample Records Count versus Observation Records Count



The Sample Size Eq. was subsequently programmed and used in all queries, even in those previously hard coded. In the end we queried and performed Proc Compares for eighty tables. More than sixty tables had observation counts greater than 10,000 records. Clearly the Sample Size Eq. gives us a simple way to determine query table size and transmit the query-by-query experience already learned to the queries not already programmed.

Process and Experience

The trial and error approach to selecting key variables is not readily transferred. However the structure and the pieces of code developed for the compare query are. We illustrate the basic flow chart of any query in the appendix.

There are two fundamental issues aside from sample size. Firstly we must decide from which perspective the observations table is to be constructed. Secondly we need to have a suitable procedure in hand to align the extracted tables so that they may be compared record by record.

Initially we supposed that the migration care-takers would transfer all the information from one system to the next, i.e. every record in the advanced table would have a corresponding record in the current table, and that the failure to transfer information was in fact a short coming of the process. In this instance it would not matter whether or not the observations table was constructed from the current or advanced system. We choose the advanced system simply because of speed of the data transfer, with the advanced being five to ten times faster than the current. Nonetheless we learned subsequently for the smaller tables, i.e. those tables less than 10,000 records which contained mostly descriptive information, that the migration-caretakers were deleting records in an effort to clean up the tables.

Because of this failure to communicate we switched to making the observations table from the current system for the tables with less than 10,000 records. The failure also forced us to introduce the measurement of table size (number of records) and to record the various table sizes for each query in order to quantify the process. In particular we recorded for each query the table size of Tempory.New&Tble, Tempory.Old&Tble, the corresponding Tempory.Goby&Tble and Tempory.Mssng&Tble and the output Tempory.Diffrnce&Tble in addition to Tempory.Obs&Tble and Tempory.Query&Tble. We recorded this information in the single record table Tempory.Recrds&Tble. A simple examination then of the table sizes provided essential information. In particular if Size(Tempory.New&Tble) did not equal Size(Tempory.Old&Tble) we knew information did not get transferred.

Once the Tempory.New&Tble and Tempory.Old&Tble have been extracted, the tables themselves have to be sorted and aligned in order to perform a meaningful Proc Compare. In addition to aligning the records we must align the fields and field types. In our particular case we choose to rename the fields in Tempory.New&Tble to coincide with the field names of Tempory.Old&Tble, outputting the result to Work.New&Tble and to recast the field type of the changed-type variables in Tempory.Old&Tble to coincide with the field type of the changed-type variables in Tempory.New&Tble, outputting the result to Work.Old&Tble. In our case the year-month-day variables changed from character to numeric.

Over a period of several weeks and numerous queries, we established a systematic way to align the records of extracted tables. The three-step method proved itself useful time and again. Step One: Tempory.Query&Tble and Work.New&Tble were merged by matched variables, &matchdkeyvars, keeping only records matched by &matchdkeyvars, as were Tempory.Query&Tble and Work.Old&Tble. An additional count variable, trns, was introduced to facilitate the upcoming merge because for some queries the records were not uniquely identified by &matchdkeyvars. Step Two: Work.Old&Tble and Work.New&Tble were merged, with records matched &matchdkeyvars trns giving Work.Goby&Tble and unmatched records giving Work.Mssng&Tble. Step Three: Work.Goby&Tble and Work.New&Tble were merged, keeping only matched records to give a finalized Work.New&Tble. Similarly, Work.Goby&Tble and Work.Old&Tble were merged to give a finalized Work.Old&Tble. The finalized Work.New&Tble and Work.Old&Tble were then compared using Proc Compare.

The output of Proc Compare using the OutNoEqual option is the Tempory.Diffrnce&Tble. We did not use the by-option with the Proc Compare but prefered to compare the aligned tables record-by-record. By doing this the Tempory.Diffrnce&Tble included the record observation number, _obs_, along with the compared fields. We used _obs_ to extract records from Work.New&Tble and Work.Old&Tble to form Work.CheckNew&Tble and Work.CheckOld&Tble.

Now while Tempory.Diffrnce&Tble is typically a small table of unequal records, it has more fields than we need to examine. In particular all the unequal records have some fields that are equal. In a companion paper we report on a procedure to identify the unequal fields. For the present we suppose only that a method is available to identify those variables that exhibit differences and to give us a list of kept variables, denoted &keepvars.

Typical Output for SAS &Tble Query Code

In what follows we provide a sanitized sample output from the Proc Compare for table &Tble. We also provide a typical piece of SAS code for the comparison of current table to advanced table in the Appendix. In Figs 2 to 4 we illustrate the comparison of the differences exhibited Old to New for the claims table. Now the claims table has more than fifty fields but we illustrate only those fields with differences, viz. with three of them. Without identifying these three fields, the output from Proc Compare would be unmanageable since the user would have to search over every field for each record. Such a task is readily programmed. Moreover it behooves us to do so because there are eighty queries. We present a solution in our second paper in the series.

Figure 2 A Partial Listing of the Output Diffrnce&Tble

OBS	SOME_CTL_NBR	YMDCOV_EFF_	YMDCOV_END_
1	XXXXXXXXXXXXXX.....		
2	XXXXXXXXXXXXXX.....		
10	XXXXXXXXXXXXXX.....	20010821	20010828
12	.....	20011114	20011114
25	XXXXXXXXX.....	20020523	20020526
27	.....	20060201	20060201

Figure 3 A Partial Listing of the CheckOld&Tble

OBS	CLAIM_	YMDTRANS	SOME_CTL_NBR	YMDCOV_EFF_	YMDCOV_END_
1	000822200492	20060922		0	0
2	000828301583	20060922		0	0
10	010910201418	20060922		0	0
12	011205200450	20060130	5191886001	0	0
25	020605OE0123	20060501		0	0
27	020806XE0979	20060217	I0603290004	0	0

Figure 4 A Partial Listing of the CheckNew&Tble

OBS	CLAIM_	YMDTRANS	SOME_CTL_NBR	YMDCOV_EFF_	YMDCOV_END_
1	000822200492	20060922	1002190006002U	0	0
2	000828301583	20060922	1002190007001T	0	0
10	010910201418	20060922	1006710109001U	20010821	20010828
12	011205200450	20060130	5191886001	20011114	20011114
25	020605OE0123	20060501	950565306	20020523	20020526
27	020806XE0979	20060217	I0603290004	20060201	20060201

In addition to the query tables, a simple MS Word report field by field summary is generated. We bypass the need to report this code but instead focus on tracking the query results and launching the queries.

Batch Automation of SAS Query Code for &Tble

The eighty queries were run several times over the course of the migration. The need to manage and track the eighty queries and their output seems obvious. We did this in two ways. Firstly we constructed a general

bat exec to launch the SAS query. The DOS syntax for the Query &Tble is located in Query &Tble.bat file; it looks like

```
Set YrMthDy=(%date:~-4%-%date:~4,2%-%date:~7,2%)
Set XcutnDirctry=C:\Projects\Migration\Queries
Set FontParms=8 Portrait Letter

Call SASBatchMe "%XcutnDirctry%" "Query &Tble" %FontParms%
```

where SASBatchMe is a generic SAS batch exec, a .bat file which looks like

```
"C:\Program Files\SAS Institute\SAS\V8\SAS.exe" -sasuser "C:\My Documents\My SAS
Files\v8\profile2" -sysin "%~1\%~2.sas" -log "%~1\SASLogs\%~2 %YrMthDy%.log" -sysparm
"%~1\SASListng\%~2 %YrMthDy%.lst" -sysprintfont ("Courier New" %3) -orientation %4 -papersize %5
```

Pretty ugly stuff! We avoid any detailed explanation of the either. It suffices to say that Query &Tble.log and Query &Tble.lst files are generated and placed in folders C:\Projects\Migration\Queries\SASLogs and C:\Projects\Migration\Queries\SASListng and given the names Query &Tble (Yr-Mth-Dy).log and Query &Tble (Yr-Mth-Dy).lst respectively, with the unique date stamp, Yr-Mth-Dy, decoded.

Stopping here is not sufficient because this only bypasses the need for the analyst to launch the SAS interactively. What makes the process doable is a peculiar piece of DOS syntax wrapped around the Call SASBatchMe. Here is the syntax that can launch a thousand SAS ships:

```
Set YrMthDy=(%date:~-4%-%date:~4,2%-%date:~7,2%)
Set XcutnDirctry=C:\Projects\Migration\Queries
Set FontParms=8 Portrait Letter

For /F "Delims==" %%A in (SAS Ships.txt) Do (
Echo %%A
Call SASBatchMe "%XcutnDirctry%" "%%A" %FontParms%
)
```

where the text file SAS Ships.txt is a simple listing of the SAS query files free of file extension, e.g.

```
Query Contrct Span
Query Contrct
Query Mbr Condtn
Query Mbr Info
Query Mbr Prev Info
Query Mbr Span
Query Mbr
Query Remarks
Query Sumry
```

Thus the DOS syntax for the Query &Tble is .bat file has been modified placing the Call SASBatchMe exec inside a For-Do loop with an unusual counter %%A which is assigned line by line from SAS Ships.txt. This text, as most DOS, has little flexibility, viz. spacing and names appear to be overly restrictive.

The eighty queries were broken into a half a dozen buckets and the queries were assigned to one or another of the buckets. In each of the buckets a single piece of SAS code, Query Sumry, was placed as the last piece of executable SAS code in the bucket. Query Sumry task was to append the Tempory.Recrds&Tble for all the tables in the Tempory folder into a single table, denoted AllRecrds. The code looks something like

```
   Proc Sort Data = SASHelp.VSTable Out = SASTables (Keep = libname memname) ;
     by libname memname ; where libname eq "TEMPORY" ; Run ;

   Data Records ; Set SASTables end = last ;
     where memname like "RECR%"  and memname ne 'RECRDS' ;
     Label memname = '  '  libname = '  ' ;
     table = Left(Tranwrd(memname,'RECRDS','')) ;
     tble = 'tble' || Left(_n_) ; Call SymPut(tble,Trim(memname)) ;
     If last then Call SymPut('NmbrTbles',Trim(Left(_n_))) ; Run ;

  %Macro AllThemRecs(Nmbr) ;
     Data AllRecrds ; Set Tempory.&Tble1 ; If _n_ eq 0 ; Run ;
    %Do da = 1 %to &Nmbr ;
       Proc Append Base = AllRecrds Data = Tempory.&&Tble&da Force ; %end ; %Mend ;
```

```
  %AllThemRecs(&NmbrTbles) ;
   Proc Summary Data = AllRecrds nway missing ; var lapstime ;
   OutPut Out = TotalTime (Drop = _:) Sum = ; Run ;

   Data AllRecrds ; UpDate Tempory.AllRecrds AllRecrds ; by table ; Run ;
```

Conclusion

We have developed a powerful method to verify migrated information based upon SAS's Proc Compare. Eighty queries were performed over the course of a single day. The method has the potential to be used elsewhere.

Contact Information

Name: Anthony E. DeGance
Company: Community Care, HMO, Inc
Address: 218 W 6th St, Tulsa, OK 74119
E-mail:ADeGance@CCOK.com
Work Phone: 918.595.5295, x 4170
Fax: 918.594.5371

Appendix

Typical SAS Query Code for &Tble

The macros OthrVars, RenameOthrVars, YMDVars and RenameYMDVars used herein are application specific and we do not report them. The macro HSize(&Limit,&Load) calculates the first prime number greater than &Limit/&Load and is taken from Paul Dorfman's paper on key-indexing, cf. Dorfman, P. 2001, "Table Look-up by Direct Addressing Key-Indexing – Bitmapping – Hashing", SUGI 26, Long Beach, CA, p008.26. Hey, the code is not complete.

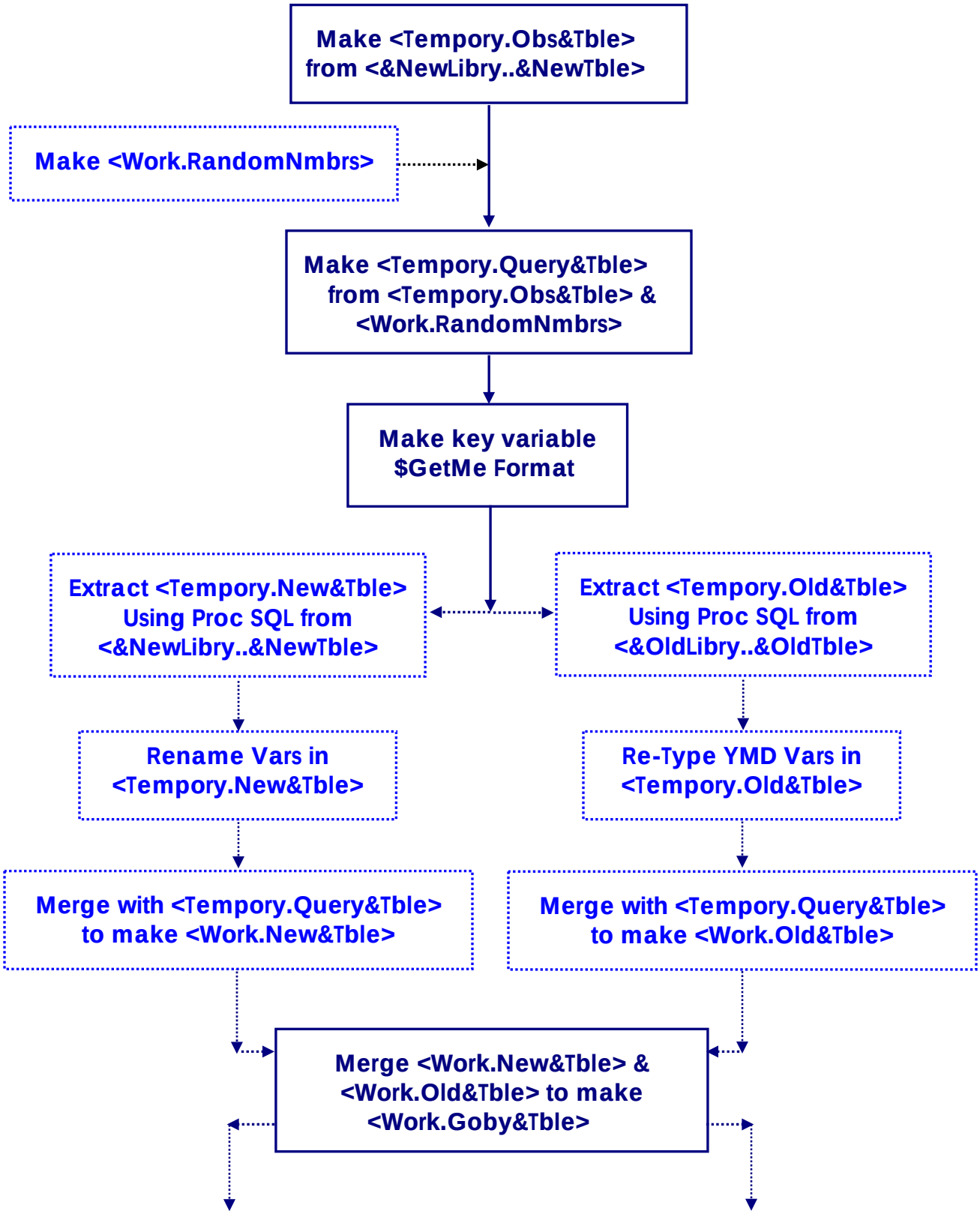
```
  %Let StarTime = %SysFunc(DateTime()) ;
  %Let Rnd = 10000 ;
  %Let MaxRecs = %Eval(150*&Rnd) ; %Put .. &MaxRecs ;
  %Let MatchdKeyVars = &keyvar &keyvaradd &keyymdvars ;

/*
   Fix observations table                                                    */

   Data Tempory.Obs&Tble (Rename = &keyvar.nbr = &keyvar) ;
     Set &NewLibry..&NewTble (Keep = &keyvar.nbr &keyvaradd &keyymdvars) end = last ;
     where ymdeff between &YMDMin and &YMDMax ; obs = _n_ ;
     If last then Call SymPut('keyvarlen',Trim(Left(Length(&keyvar.nbr)))) ; Run ;

   Data _null_ ; Call SymPut("NmbrObs",Left(MyObs)) ; Stop ;
     Set Tempory.Obs&Tble nobs = MyObs ; Run ; %Put .. &NmbrObs ;

/*
   Fix size of query table                                                   */

  %Let LogRecs = %SysEvalF(1.33333 + 0.666667*%SysFunc(Log10(&NmbrObs))) ;
  %Let SmpleRecs = %SysEvalF(10**&LogRecs) ;
  %Let RundRecs = %SysFunc(Round(&SmpleRecs,100)) ;
  %Let Limit = %SysFunc(Min(&RundRecs,&MaxRecs,&NmbrObs)) ;
  %HSize(&Limit,0.5) ;

/*
   Construct table of random numbers                                         */

   Data RandomNmbrs (Keep = obs) Cnt (Keep = obs cnt hash) ;
     Array obsrv (0:&hashme) _Temporary_ ; Retain seed cnt 0 ;
     Do Until (cnt ge &Limit) ;
       randy = RanUni(seed) ; obs = Int(randy*&nmbrobs) ; hash = Mod(obs,&hashme) ;
```

```
      Do dat = hash by -1 Until (obsrv(dat) eq '' or obsrv(dat) eq obs) ;
        If dat < 0 then dat = &hashme - 1 ; end ;
      If obsrv(dat) eq '' then cnt = cnt + 1 ;
      obsrv(dat) = obs ; OutPut RandomNmbrs Cnt ; end ;
  Proc Sort Data = RandomNmbrs NoDups ; by obs ; Run ;

/*
  Construct query table                                                          */



  Data Tempory.Query&Tble ; Do Until (endfile) ;
    Set RandomNmbrs end = endfile ;
    Set Tempory.Obs&Tble point = obs ; OutPut ; end ; Stop ; Run ;
  Proc Sort NoDups ; by &matchdkeyvars ; Run ;
/*
  Make &keyvar format                                                            */

  Data Frmt_Query / View = Frmt_Query ; Length &keyvar $&keyvarlen..  ;
    Set Tempory.Query&Tble ; by &keyvar ; If first.&keyvar ;
    fmtname = '$GetMe' ; label = 1 ; OutPut ;
    If _n_ eq 1 then do ; &keyvar = 'Other' ; label = 0 ; OutPut ; end ;
  Proc Format CntlIn = Frmt_Query (Rename = &keyvar = start) ; Run ;
/*
  Extract and store New&Tble and Old&Tble using &keyvar format                   */

  Proc SQL ; Create Table Tempory.New&Tble as Select *
             from &NewLibry..&NewTble where InPut(Put(&keyvar.nbr,$GetMe.),F1.) ;
  Proc SQL ; Create Table Tempory.Old&Tble as Select *
             from &OldLibry..&OldTble where InPut(Put(&keyvar,$GetMe.),F1.) ; Quit ;
  Proc Sort Data = Tempory.Old&Tble ; by &matchdkeyvars ; Run ;
 %Let LapsTime = %SysEvalF(%SysFunc(DateTime()) - &StarTime) ; %Put .. &LapsTime ;
/*
  Identify New&Tble fields to rename and rename New&Tble fields                  */

 %OthrVars(&NewLibry,&NewTble) ;
 %RenameOthrVars(&NmbrOthr,Tempory,New&Tble,Vrble,Lable) ;

  Proc Sort Data = Tempory.New&Tble ; by &matchdkeyvars ; Run ;
/*
  Identify Old&Tble YMD fields to change from character to numeric and do so     */

 %YMDVars(&NewLibry,&NewTble) ;
  Data Old&Tble (Drop = cnt &ymdvars) ; Set Tempory.Old&Tble ; by &matchdkeyvars ;
    Array ymd(&nmbrymd) &ymdvars ; Array yymmdd(&nmbrymd) ;
    Do cnt = 1 to &nmbrymd ;
     yymmdd(cnt) = InPut(ymd(cnt),F8.) ;
     yymmdd(cnt) = Max(yymmdd(cnt),0) ; end ; Run ;
 %RenameYMDVars(&NmbrYmd,Work,Old&Tble,yymmdd) ;
/*
  Establish Old&Tble and New&Tble using Query&Tble                               */

  Data Old&Tble ; Merge Tempory.Query&Tble (in = qry) Old&Tble (in = old) ;
    by &matchdkeyvars ; If qry*old ;
  Data Old&Tble ; Set Old&Tble ; by &matchdkeyvars ;
    If first.&ymdlastvar then trns = 1 ; else trns + 1 ; Run ;

  Data New&Tble ; Merge Tempory.Query&Tble (in = qry) Tempory.New&Tble (in = new) ;
    by &matchdkeyvars ; If qry*new ; Run ;
  Data New&Tble ; Set  New&Tble ; by &matchdkeyvars ;
    If first.&ymdlastvar then trns = 1 ; else trns + 1 ; Run ;
/*
  Establish Goby&Tble                                                            */

  Data Goby&Tble (Keep = &matchdkeyvars trns) Mssng&Tble ;
    Merge New&Tble (in = new) Old&Tble (Keep = &matchdkeyvars trns in = old) ;
    by &matchdkeyvars trns ; Format newold Binary2. ; newold = Sum(2*new,old) ;
    If new*old then OutPut Goby&Tble ; Else OutPut Mssng&Tble ; Run ;
/*
  Align Old&Tble and New&Tble using Goby&Tble                                    */

  Data New&Tble ; Merge Goby&Tble (in = goby) New&Tble ;
    by &matchdkeyvars trns ; If goby ; unique = first.&ymdlastvar*last.&ymdlastvar ;
```

```
   Data Old&Tble ; Merge Goby&Tble (in = goby) Old&Tble ;
     by &matchdkeyvars trns ; If goby ; unique = first.&ymdlastvar*last.&ymdlastvar ; Run ;
/*
   Construct Diffrnce&Tble                                                              */

   Proc Compare Base = Old&Tble Compare = New&Tble OutNoEqual NoPrint
                Out = Tempory.Diffrnce&Tble ; Run ;

  %Let LapsTime = %SysEvalF(%SysFunc(DateTime()) - &StarTime) ; %Put .. &LapsTime ;

   Data Diffrnce&Tble (Keep = _obs_ &keepvars) ChekObs (Keep = _obs_) ;
     Set Tempory.Diffrnce&Tble ; Run ;

   Data CheckOld&Tble (Keep = _obs_ &keepvars &matchdkeyvars) ;
     Do Until (Endofit) ;
       Set work.ChekObs end = Endofit ; obs = _obs_ ;
       Set Old&Tble point = obs ; OutPut ; end ; Stop ; Run ;

   Data CheckNew&Tble (Keep = _obs_ &keepvars &matchdkeyvars) ;
     Do Until (Endofit) ;
       Set work.ChekObs end = Endofit ; obs = _obs_ ;
       Set New&Tble point = obs ; OutPut ; end ; Stop ; Run ;

  %Let LapsTime = %SysEvalF(%SysFunc(DateTime()) - &StarTime) ; %Put .. &LapsTime ;
/*
   Make informational Recrds&Tble                                                       */

   Data Tempory.Recrds&Tble (Drop = datset) ;
     Format sascode $48. table newtble oldtble $32. transdate YYMMDD10. lapstime Time11.2 ;
     sascode = "&SASCode" ;
     table = UpCase("&tble") ; newtble = UpCase("&newtble") ; oldtble = UpCase("&oldtble") ;
     datset = Open("Tempory.Obs&Tble",'I') ;   obsrecs = Attrn(datset,'nobs') ;
     datset = Open("Tempory.Query&Tble",'I') ; queryrecs = Attrn(datset,'nobs') ;
     datset = Open("Tempory.New&Tble",'I') ;   newrecs = Attrn(datset,'nobs') ;
     datset = Open("Tempory.Old&Tble",'I') ;   oldrecs = Attrn(datset,'nobs') ;
     datset = Open("Work.GoBy&Tble",'I') ;     gobyrecs = Attrn(datset,'nobs') ;
     datset = Open("Work.Mssng&Tble",'I') ;    mssngrecs = Attrn(datset,'nobs') ;
     datset = Open("Tempory.Diffrnce&Tble",'I') ; diffrncerecs = Attrn(datset,'nobs') ;
     transdate = today() ; lapstime = DateTime() - &StarTime ; Run ;
```

Appendix

Information Flow Diagram



Information Flow Diagram