

Novel Programming Methods for Working with Large Claims Databases

Richard Read Allen
Peak Stat, Evergreen, CO USA
PhUSE 2012



INTRODUCTION

Working with large claims databases can be quite a challenge.

- These can be many gigabytes in size
- Running queries against them can be extremely time-consuming
- Need to write code that runs as efficiently as possible

We will look at some questions that are common in claims-based research.

To solve these questions, we'll look at

1. Some standard methods commonly used to solve these questions.
2. Some novel methods that can be used to solve the same problems, usually more efficiently.

The novel methods will use

- the hash object
- DOW loop (sometimes called the DO loop of DOrfman and Whitlock)

to attempt to speed up processing.

A third example combining both of these concepts will be presented but not compared to standard solutions.

RESEARCH PROBLEM 1

All prescription records for each of the approximately 6.5 million subjects in this database are stored in dataset P and are sorted by pat_id. We have a second dataset S of subjects with certain characteristics that we are interested in researching. This dataset is smaller, containing 527,627 subjects and is also sorted by pat_id.. The third dataset (N) has NDC codes for a class of drugs that we want to study in these subjects. This dataset has 1,091 records and is sorted by NDC code.

The object is to extract all of the prescription records from P for the class of drugs in N taken by the subjects that are in dataset S.

STANDARD SOLUTION #1

```
/*=====*/
/* Merge, sort and merge again.
/* Sort to get back to original order.
/*=====*/
data P_S;
  merge P
    S(in=KeepIt keep=pat_id);
  by pat_id;
  if KeepIt;
run;

proc sort data=P_S;
  by NDC;
run;

data want1;
  merge P_S(in=I)
    N (in=KeepIt keep=NDC);
  by NDC;
  if I & KeepIt;
run;

proc sort data=want1;
  by pat_id from_dt;
run;
```

STANDARD SOLUTION #2

```
/*=====*/  
/* Informat and filter  
/*=====*/  
data NDCinfmt;  
  set N end=EOF;  
  type='I';  
  fmtname='NDC';  
  start=NDC;  
  label=1;  
  output;  
  if EOF then do;  
    hlo='o';  
    label=0;  
    output;  
  end;  
run;  
  
proc format cntlin=NDCinfmt;  
run;  
  
data want2;  
  merge P  
    S(in=KeepIt keep=pat_id);  
  by pat_id;  
  if KeepIt & input(NDC,NDC.)=1;  
run;
```

STANDARD SOLUTION #3

```
/*=====*/  
/* SQL  
/*=====*/  
proc sql;  
  create table want3 as  
  select p.*  
  from      S, P, N  
  where p.pat_id=s.pat_id & p.ndc=n.ndc  
  order by s.pat_id, p.from_dt;  
quit;
```

Introduction to HASH Object:

The basic pieces of the hash object that we will use are the declare statement and the defineKey, defineData, defineDone and find methods.

The four basic steps for building a hash using these components are

1. **Instantiate the hash object:** Use the DECLARE statement with the keyword HASH followed by the hash name to create the hash object. Options are specified within the parentheses and include DATASET to load the hash object from an existing SAS dataset and ORDERED to specify how the data is returned in key-value order.
2. **Define the key part:** Use the DEFINEKEY method to create a key of one or more variables. The key must be unique.
3. **Define the data part:** Use the DEFINEDATA method to specify zero or more data variables to be associated with the key variables.
4. **Complete the definition of the hash object:** Use the DEFINEDONE method to conclude the definition of the hash object

NOVEL SOLUTION

```
/*=====*/  
/* Double Hash  
/*=====*/  
data want4;  
  if 0 then do;  
    set N(keep=NDC);  
    set S(keep=pat_id);  
  end;  
  if _n_=1 then do;  
    declare hash n(dataset:'N');  
    n.defineKey('NDC');  
    n.defineDone();  
  
    declare hash s(dataset:'S');  
    s.defineKey('pat_id');  
    s.defineDone();  
  end;  
  set P;  
  if n.find()=0 & s.find()=0;  
run;
```

COMPARISON OF SOLUTIONS

To compare the speed of the proposed solutions, programs were run using SAS version 9.3 on a remote server. Each of the programs was replicated 5 times in an attempt to get better estimates.

Below are the “real times” for each of the methods. For each run the bolded italics time is the quickest. The minimum time for each solution is highlighted in green and the median time is highlighted in yellow.

Run Number	Standard Solutions				Novel Solution
	#1	#2	#2A	#3	
1	10:27.24	2:42.36	2:56.58	2:48.25	2:29.35
2	10:56.00	2:44.30	2:48.30	2:36.64	2:44.01
3	10:47.03	2:46.77	2:35.98	2:52.79	2:29.29
4	10:50.02	2:40.22	2:56.73	2:41.23	2:48.14
5	10:37.19	2:23.90	2:27.37	2:22.90	2:39.06
Mean	10:43.50	2:39.51	2:44.99	2:40.36	2:37.97

RESEARCH PROBLEM 2

The diagnosis claims dataset (D) contains diagnosis codes diag1-diag4 for all subjects that have diagnosis claims. There are possibly multiple records per subject/date. This is a large dataset (~67 GB) but it is already sorted by subject (pat_id) so that time cost is not a factor.

The object is to create a dataset of one record per subject with a 0/1 flag for DM and HTN by searching diag1-diag4 on each record for icd-9 codes for these disease states

STANDARD SOLUTION #1

```
/*=====*/  
/* Flag and summarize, DATA step and PROC summary  
/*=====*/  
data D2;  
  set D(keep=pat_id diag:);  
  array dx diag:;  
  DM=0; HTN=0;  
  do i=1 to dim(dx);  
    if dx(i)='250' then DM=1;  
    if dx(i) in ('401','402','403','404','405') then HTN=1;  
  end;  
run;  
  
proc summary data=D2 nway;  
  class pat_id;  
  var DM HTN;  
  output out=comorbs(drop=_) max=;  
run;
```

STANDARD SOLUTION #2

```
/*=====*/  
/* Flag and summarize within same DATA step  
/* then output last record per subject.  
/*=====*/  
data comorbs2(drop=i diag:);  
  retain DM HTN;  
  array diag{4}$ diag1 - diag4;  
  set D(keep=pat_id diag:);  
  by pat_id;  
  if first.pat_id then do;  
    DM=0;  
    HTN=0;  
  end;  
  do i= 1 to 4;  
    DM=max(DM, diag{i}='250');  
    HTN=max(HTN, diag{i} in: ('401','402','403','404','405'));  
  end;  
  if last.pat_id then output;  
run;
```

Introduction to DOW:

The DOW loop is a powerful technique that moves the DATA step SET statement inside of an explicitly-coded DO-loop.

This gives the programmer complete control over retention of variable values and the population of the Program Data Vector (PDV) by allowing a natural isolation of DO-loop instructions related to a certain break-event.

The DOW isolates actions performed before and after the DO-loop from the instructions inside the loop and eliminates the necessity of retaining variables in most applications.

In its most basic and well-known form using a DO UNTIL (LAST.ID) construct, it naturally lends itself to BY-processing of grouped data.

The Basic Structure of the DOW Loop is as follows:

```
data ... ;  
    <stuff done before break-event> ;  
    do <index specs> until ( break-event ) ;  
        set ... ;  
        by ... ;  
        <stuff done for each record> ;  
    end ;  
    <stuff done after break-event... > ;  
run ;
```

There are three separate sections of code in this structure where instructions can be grouped for different types of processing, depending on how you need to handle the data.

1. Between the top of the implied data step loop and before the first record in a by-group is read if the action needs to be done before the by-group is processed.
2. Inside the DOW-loop, for each record in the by-group, if the action needs to be done to each record.
3. After the last record in the by-group has been processed and before the bottom of the implied data step loop, if the action such as summarizing needs to be done after the by-group is processed.

Here's a brief description of how this works:

The DOW-Loop itself begins with the DO UNTIL statement, and takes control from the traditional implicit DATA step loop.

Because the SET statement is inside of the DOW-loop, the loop is not exited until after the last record for the break-event has been processed.

Variables populated inside of the loop are retained while all of the records for the break-event are read in.

A single record for the break-event, containing the filled arrays, is output at the completion of the DOW-loop.

NOVEL SOLUTION

```
/*=====*/  
/* DOW  
/*=====*/  
data comorbs3(drop=i diag:);  
  do until(last.pat_id);  
    set D(keep=pat_id diag:);  
    by pat_id;  
    array dx(4) $ diag1-diag4;  
    if sum(dm,htn)=2 then continue;  
    do i=1 to 4;  
      if missing(dx[i]) then leave;  
      DM=max(DM,(dx[i]='250'));  
      HTN=max(HTN,(dx[i] in ('401','402','403','404','405')));  
    end;  
  end;  
run;
```

COMPARISON OF SOLUTIONS

Below are the “real times” for each of the methods. For each run the bolded italics time is the quickest. The minimum time for each solution is highlighted in green and the median time is highlighted in yellow.

Run Number	Standard Solutions		Novel Solution
	#1	#2	
1	21:10.92	9:03.48	<i>8:33.61</i>
2	18:35.83	9:19.92	<i>8:19.55</i>
3	<i>17:36.01</i>	<i>7:51.48</i>	<i>7:44.61</i>
4	21:59.63	<i>8:42.42</i>	9:04.45
5	<i>19:12.41</i>	<i>8:37.08</i>	8:37.44
Mean	19:42.96	8:42.88	<i>8:27.93</i>

Comparing the standard data step approach with the DOW

```
data comorbs2(drop=i diag:);
  retain DM HTN;
  array diag{4}$ diag1 - diag4;
  set D(keep=pat_id diag:);
  by pat_id;
  if first.pat_id then do;
    DM=0;
    HTN=0;
  end;
  do i= 1 to 4;
    DM=max(DM, diag{i}='250');
    HTN=max(HTN, diag{i} in:
      ('401','402','403','404','405'));
  end;
  if last.pat_id then output;
run;
```

```
data comorbs3(drop=i diag:);
  do until(last.pat_id);
    set D(keep=pat_id diag:);
    by pat_id;
    array dx(4) $ diag1-dia4;
    if sum(dm,htn)=2 then continue;
    do i=1 to 4;
      if missing(dx[i]) then leave;
      DM=max(DM,(dx[i]='250'));
      HTN=max(HTN,(dx[i] in:
        ('401','402','403','404','405')));
    end;
  end;
run;
```

RESEARCH PROBLEM 3

For this problem I'll just present the “novel” solution which uses the double hash and the DOW in one DATA step to solve this problem.

The investigator on this study wanted to look at insulin usage by service day found in the detailed inpatient billing records. Both the standardized charge master dataset (*chgmstr*) and the hospital charge dataset (*hospchg*) have records that contain certain key words to indicate insulin usage in their description.

```

data insulin_days(keep=pat_key serv_day insulin);
/*-----*/
/* Define hashes to create keys to flag insulin usage
/*-----*/
if 0 then do;
    set sc_ins(keep=std_chg_code);
    set hc_ins(keep=hosp_chg_id);
end;
if _n_=1 then do;
    declare hash si(dataset:'sc_ins');
    si.defineKey('std_chg_code');
    si.defineDone();
    declare hash hi(dataset:'hc_ins');
    hi.defineKey('hosp_chg_id');
    hi.defineDone();
end;
/*-----*/
/* Loop through records for each patient/day
/* to create a flag for insulin usage
/*-----*/
do until (last.serv_day);
    set COPD.admit_PatBill;
    by pat_key serv_day;
    Insulin=max(Insulin,(si.find()=0 | hi.find()=0));
end;
run;

```

CONCLUSION

There are probably multiple situations where the Hash object, the DOW loop or a combination of both could possibly speed things up in your day-to-day programming against large claims databases.

These are basic examples and one could easily use more than 2 hashes in the same DATA step if more sub setting of the large claims file is required or even multiple DOWs. The hash object can also be used to add data to your datasets.

Avoiding sorts whenever possible, especially on large datasets, will make your code run more efficiently.

If you have programs that are running a very long time, you should consider experimenting with some of these novel methods in your code to speed it up.

You may be surprised at how much time you can save. Plus, as an added bonus, they're different and fun to program.

CONTACT INFORMATION

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

Richard Read Allen
Peak Statistical Services
5691 Northwood Drive
Evergreen, CO 80439 USA

Work Phone: 303-670-5386
Fax: 303-670-5387

Email: rrallen@peakstat.com
Web: www.peakstat.com

