Paper HE06

Novel Programming Methods for Working with Large Claims Databases

Richard Read Allen, Peak Stat, Evergreen, Colorado, USA

ABSTRACT

This presentation will look at some novel and efficient ways to code against large claims databases. These methods will use the DOW loop and the hash object, separately and together, to determine such measures as comorbidities or other queries about subjects in claims data.

INTRODUCTION

Working with large claims databases can be quite challenging. Since they can be many gigabytes in size, running queries against them can be extremely time-consuming. Therefore it's important to write code that runs as efficiently as possible when running programs to extract information from them. In this paper we will look at some questions that are common in claims-based research. We'll look at some standard methods commonly used to solve two of these questions and at some novel methods that can be used to solve the same problems, usually more efficiently. These novel methods will use the hash object and the 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. We're going to assume some understanding of these concepts in the research examples below, but a brief introduction to each follows.

BASIC INTRODUCTION TO HASH

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 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.

One uses the DATA step Component Interface and object dot notation

ObjectName.Method(Parameters)

to create and manipulate these component objects.

Hash objects read contents of a dataset into memory at the top of the DATA step. Operations on the hash object are run entirely in memory, which is usually faster than disk-based operations like a DATA step merge or PROC SQL.

BASIC INTRODUCTION TO DOW

The DOW-Loop was originally developed by Don Henderson and popularized the past few years on the SAS-L listserv by Paul Dorfman and Ian Whitlock, among others. It is a programming technique that involves taking control of the implicit DO loop inherent in the DATA step by identifying and storing a value in a variable that is retained until all records for a by-group have been processed.

The basic structure of the DOW loop is as follows [D4: Dorfman/Shajenko]:

```
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 the 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 the inner loop 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 initiated as missing and retained while all of the records for the break-event are read in. This eliminates the necessity of using the RETAIN statement.
- A single record for the break-event, containing the populated variables, is output at the completion of the DOW-loop by the "implied" output at the end of the DATA step.

The Dorfman/Shajenko paper [D4] has some detailed explanations of the inner workings of the DOW loop and its variations; including using multiple DOW loops in the same DATA step and using the DOW with the data step hash object. We'll look at a simple example using DOW with a hash later.

DATA SOURCE FOR PROBLEMS 1 and 2

The data used for the first two examples is a 10% random sample from the PharMetrics® (IMS Lifelink™) claims database.

> *The LifeLink™ Health Plan Claims Database is comprised of commercial health plan information obtained from managed care plans throughout the United States. It is paid claims data, which by definition is information collected by the medical plans from medical service providers to facilitate the adjudication and payment of health insurance benefits on behalf of the plan's enrolled members. It is fully adjudicated medical and pharmaceutical claims for over 61 million unique patients from over 98 health plans across the U.S. (approximately 16 million covered lives per year).*
>
> *(SOURCE: LifeLink™ Health Plan Claims Data User Guide & Data Dictionary, September 2009)*

The large claims dataset (>100GB) in this database has been split into pharmacy (P) and diagnosis (D) subset datasets to streamline the coding somewhat. A subset of subjects (S) of interest in the examples has been extracted from the population datasets and the class of drugs (N) we wish to study has also been extracted from the prescription drug dataset. The datasets we will use in these examples are the following:

#	Name	File Size	Observations
1	D	67,289,875,456	438,085,057
2	N	197,632	1,091
3	P	22,610,277,376	165,602,587
4	S	12,866,560	527,627

These datasets are on the small side for these types of databases. The contents of these datasets can be found in the appendix.

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.

Some of the standard methods for solving this problem are the following:

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;
```

This can be modified slightly as follows to get the same result.

```
data want2a;
  merge P(in=Rx where=(input(NDC,NDC.)=1))
        S(in=KeepIt keep=pat_id);
  by pat_id;
  if Rx & KeepIt;
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;
```

NOVEL SOLUTION

The solution below makes use of the hash object reviewed earlier. It sets up two separate hashes, N for the NDC codes and S for the subjects (pat_id). In these hashes we only define keys and match them against the large prescription file by using the find() method and keeping only observations with both keys. Demographic data from S and/or drug data from N could also be added to the prescription file at this point from the hash objects if one wishes.

```
/*=============================================================================*/
/* 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, I ran each of the programs using SAS version 9.3 against the data with the option fullstimer turned on. Since these kinds of comparisons are subject to many factors, I ran each of the programs 5 times in an attempt to get better estimates of how long each would take to do this task. Below are the "real times" for each of the methods. These times may differ on different machines, operating systems or for many other factors but should give us a reasonable picture of how fast each solution runs.

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***

The first standard method (#1) uses sorts and merges. The first sort is very expensive time-wise since the dataset P_S is quite large. If this is a small dataset this method becomes competitive with the others, but that's rarely the case in real life. This method takes over 4 times longer than the others to complete using the sample data.

The second standard method using formats is much faster than the first because it avoids sorting the large P_S dataset. The first option of applying the format in the DATA step (#2) rather than in a where clause on the dataset in the merge (#2A) is better, though that's not really what I expected. This option is competitive with the proposed novel solution as well as SQL and even has the fastest time on one of the runs.

The third standard method using SQL (#3) had the fastest individual time of 2:22.90. It is also very competitive with the proposed novel method using the double hash. In version 9.2 of the SAS system, SQL was much slower than hash but improvements in version 9.3 have improved its performance.

The double hash solution had the best mean and median times of the solutions. I expected that it would be considerably faster than the competition but that was not shown in my tests. There's not much performance difference between using the informat or SQL or the double hash in this simple example. The real strength of the double hash (and the informat method) is that you have access to all DATA step functionality to further process your data once you've identified the records you need. The same processing may also be able to be done in SQL, but I believe it would usually be easier to code in the DATA step for most SAS programmers.

RESEARCH PROBLEM 2

The diagnosis claims dataset (D) contains diagnosis codes diag1-diag4 for all subjects in the 10% random sample 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 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;
```

NOVEL SOLUTION

This solution makes use of the DOW logic reviewed earlier to speed up the processing. The DM and HTN flags are not assigned at the beginning of the loop so they are set to missing. The retention property is then used set these flags to 0/1 based on the existence of the ICD-9 codes for these diseases as we loop through each subject's diagnoses (diag1-diag4). If it's ever found, it will remain 1 due to the properties of the maximum function.

```
/*=============================================================================*/
/* 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

Once again to compare the speed of these proposed solutions, I ran each of the programs 5 times and looked at the numbers. 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 #1	#2	Novel Solution
1	21:10.92	9:03.48	***8:33.61***
2	18:35.83	9:19.92	***8:19.55***
3	17:36.01	7:51.48	***7:44.61***
4	21:59.63	***8:42.42***	9:04.45
5	19:12.41	***8:37.08***	8:37.44
Mean	19:42.96	8:42.88	***8:27.93***

The first standard method (#1) with the separate proc summary is the slowest of these three methods. Each record of the D dataset is kept in the first DATA step, then that large dataset needs to be summarized.

Both of the single DATA step solutions take less than half the time of the first method and are very comparable. In this simple example, the novel solution using DOW techniques (#3) holds an edge over the standard DATA step solution (#2). This edge is slight but I'd conjecture that as the dataset D gets larger and/or more flags are needed for other comorbid disease states, the DOW would perform significantly better.

DATA SOURCE FOR PROBLEM 3

Hospital claims data from the Premier® Perspective Comparative Hospital Database (PCD) reported from 2003 through 2008 are used for problem #3. The following is a snippet to describe the database from their database description document.

Perspective is unique and powerful in its ability to capture, standardize, and aggregate data from multiple organizations. Non-standard hospital charge description coding schemas often hamper comparisons of inpatient data sets at the resource consumption level. To overcome these obstacles, Premier developed its standard list of common charges for the detailed items consumed by the patient. This standardized charge master catalogue captures over 47,000 items on the patient's detailed inpatient bill providing an unmatched level of granularity to collect and normalize data from hospitals. Premier staff "maps" individual charge master elements to Premier's standardized charge master for each hospital participating in the database.

SOURCE: Premier Prospective Database Overview (December 10, 2004)

The components of this database that we'll be using are the *patbill, chgmstr* and *hospchg* datasets. The *patbill* dataset contains all the detailed inpatient billing records for each admission by service day. This dataset contains two variables that pertain to the charges and supply a description of the billing, std_chg_code and hosp_chg_id. The dataset *chgmstr* is the standardized charge master catalogue referred to in the above description. The std_chg_code variable is mapped to a description in this dataset. The *hospchg* dataset maps the hosp_chg_id to the "Description used in hospital's billing system".

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. I'd think that most alternative solutions would probably run longer but I haven't tested any.

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.

The first step is to extract the codes in *chgmstr* and *hospchg* that pertain to insulin usage. There are 94 records in chgmstr and 23,804 records in hospchg that we have descriptions containing the key words we were looking for. In this case the patbill dataset has been reduced already to only records for the subjects of interest in the study and sorted by pat_key and serv_day. Still there are 34,728,496 observations in the reduced patbill dataset and it's about 3.3 GB in size.

The following section of code is part of a larger program to create an analysis dataset for events occurring on each day of the admission. It uses the double hash to define keys from the records found in the *chgmstr* (sc_ins) and *hospchg* (hc_ins) datasets. Since the data are in pat_key and serv_day order this allows us to then loop through the data for each subject to create a flag for insulin use depending on whether or not one of the codes in either sc_ins or hc_ins appears in a detailed billing record.

```
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;
```

In five test runs of this dataset, the code ran in 45.26, 56.40, 52.37, 44.42 and 46.50 seconds for an average of 48.99 seconds. Pretty quick!

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.

REFERENCES

DOW:

1. *2 PROC TRANSPOSEs = 1 DATA Step DOW-Loop*, Nancy Brucken, PharmaSUG 2007 Proceedings
2. *One-Step Change from Baseline Calculations*, Nancy Brucken, PharmaSUG 2008 Proceedings
3. *The DOW (not that DOW!!!) and the LOCF in Clinical Trials*, Venky Chakravarthy, SUGI 28 Proceedings
4. *The DOW loop unrolled,* Paul Dorfman & Lessia Shajenko, PharmaSUG 2008 Proceedings
5. *Practical Uses of the DOW Loop in Pharmaceutical Programming,* Richard R Allen, PhUSE 2009 Proceedings.

HASH:

1. Getting Started with the DATA Step Hash Object, Jason Secosky & Janice Bloom, PharmaSUG 2007 Proceedings
2. Getting Started with the DATA Step Hash Iterator, Janice Bloom & Jason Secosky, support.sas.com
3. Hashing: Generations, Paul Dorfman & Gregg Snell, SUGI 28 Proceedings
4. DATA Step Hash Objects as Programming Tools, Paul Dorfman & Koen Vyverman, SUGI 30 Proceedings
5. Think FAST! Use Memory Tables (Hashing) for Faster Merging, Greg Snell, SUGI 31 Proceedings
6. A Hash Alternative to the PROC SQL Left Join, Ken Borowiak, NESUG 2005 Proceedings
7. Dorfman, Paul M and Shajenko, Lessia S (2011). "Hash + Point = Key." NESUG 2011 Proceedings.
8. *The SAS® Hash Object: It's Time To .find() Your Way Around*, Peter Eberhardt, SAS Global Forum 2011 Proceedings
9. Merge vs. Join vs. Hash Objects: A Comparison using "Big" Medical Device Data, James Johnson, PharmaSUG 2012 Proceedings

ACKNOWLEDGMENTS

Thanks to John King (AKA, data _null_) of Ouachita Clinical Data Services, Gerald Pulver from the School of Medicine at UC Denver and Brenda Beatty of COHO/COR at UC Denver for their input and suggested solutions. Thanks also to Gerald for his editorial advice.

RECOMMENDED READING

Anything by Paul Dorfman.

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

APPENDIX

Structure of datasets from PharMetrics database (IMS Lifelink™) used in research problems 1 and 2.

Dataset D:

Alphabetic List of Variables and Attributes						
#	**Variable**	**Type**	**Len**	**Format**	**Informat**	**Label**
3	ALLOWED	Num	8	11.2	9.2	Allowed Amount
20	SRV_UNIT	Num	4			Service Units
9	charge	Num	8	11.2	9.2	Charge
17	cluspvid	Char	16			ETG Cluster Provider id
18	conf_num	Char	10			
11	diag1	Char	5			Diagnosis Code 1
12	diag2	Char	5			Diagnosis Code 2
13	diag3	Char	5			Diagnosis Code 3
14	diag4	Char	5			Diagnosis Code 4
5	from_dt	Num	8	YYMMDD10.		From Date
8	paid	Num	8	11.2	9.2	Paid
1	pat_id	Char	16			Pharmetric's encrypted short Pat_id
16	pos	Char	2			Place Of Service
10	proc_cde	Char	5			Proc Code(cpt4)
7	prov_id	Char	16			Short Provider id
15	ptypeflg	Char	1			Assigned Prov Type
4	rec_ix	Num	8			Record Joining Variable
19	rec_spec	Char	8			Record Level Specialty
2	rectype	Char	1			ETG Record Type
6	to_dt	Num	8	YYMMDD10.		To Date

Dataset N:

Alphabetic List of Variables and Attributes					
#	**Variable**	**Type**	**Len**	**Format**	**Label**
2	brand_name	Char	60		
4	gen_nm	Char	60		
3	gpi	Char	14		
1	ndc	Char	11	$CHAR11.	National Drug Code
6	strength	Num	8		
5	unit	Char	11		

Dataset P:

Alphabetic List of Variables and Attributes						
#	**Variable**	**Type**	**Len**	**Format**	**Informat**	**Label**
3	ALLOWED	Num	8	11.2	9.2	Allowed Amount
4	DAYSSUP	Num	8	4.		Days Supply
5	QUAN	Num	8	4.		Quantity
11	charge	Num	8	11.2	9.2	Charge
15	cluspvid	Char	16			ETG Cluster Provider id
7	from_dt	Num	8	YYMMDD10.		From Date
12	ndc	Char	11	$CHAR11.		National Drug Code
10	paid	Num	8	11.2	9.2	Paid
1	pat_id	Char	16			Pharmetric's encrypted short Pat_id
14	pos	Char	2			Place Of Service
9	prov_id	Char	16			Short Provider id
13	ptypeflg	Char	1			Assigned Prov Type
6	rec_ix	Num	8			Record Joining Variable
16	rec_spec	Char	8			Record Level Specialty
2	rectype	Char	1			ETG Record Type
8	to_dt	Num	8	YYMMDD10.		To Date

Dataset S:

Alphabetic List of Variables and Attributes				
#	**Variable**	**Type**	**Len**	**Label**
2	der_sex	Char	1	Derived Sex
3	der_yob	Num	4	Derived Yob
1	pat_id	Char	16	Pharmetric's encrypted short Pat_id
4	pat_region	Char	2	Derived Patient Region