Paper 255-2011

Building Provider Panels: An Application for the Hash of Hashes

Judy Loren, Health Dialog, Portland, ME
Richard A. DeVenezia, Independent Consultant, Remsen, NY

ABSTRACT

Did you know that an element of a hash table can be another hash table? How cool is that? Kind of like a multidimensional array on steroids. You don't need to know in advance how many elements you will have nor how deep they will nest. Furthermore, you can do this without a single macro variable!

The "hash of hashes" technique provides one solution to a classic challenge in healthcare: how to process claims data to determine which provider within a given specialty is most likely overseeing a patient's care. The algorithm requires examining an unpredictable amount of information and making decisions at defined breakpoints.

This paper will demonstrate the code for a "hash of hashes" solution and compare it to two other approaches: the double DOW and SQL. Along the way we'll make use of some features of hash tables that are new in 9.2. If you love hash tables, you'll flip for the hash of hashes!

INTRODUCTION

Analysts in the health care field often need to infer from only healthcare claims (also called **administrative data**) which **provider** (doctor, nurse practitioner, physician assistant, etc.) has been taking charge of a patient's care. The **attribution** of patient to provider drives activities such as **outreach** (sending a message about a patient's activities or apparent needs to a provider) and **assessment** (calculating rates of specific care over all the patients attributed to a provider, also called the provider's **panel**). Attribution can be limited to finding a patient's **Primary Care Provider** (PCP), or it can be extended to finding the overseeing provider within each **specialty** (cardiology, neurology, oncology, etc.)

Most analysts use some variation of the approach described here. This has been simplified for illustration purposes. As we will see, other techniques can handle the simplified version with greater efficiency. The demonstration of the Hash of Hashes (HoH) approach is presented as the basis for solving more complex rulesets found in messy real world problems.

Here is the simplified ruleset for attributing a patient to one provider within a specialty. For each patient:

1) Select claims from the most recent 18 months.
2) Select only those claims that have procedure codes indicating the provider saw the patient for a face-to-face evaluation and management visit in an outpatient setting.
3) Count the number of visits to each individual provider.
4) Note the date of the most recent visit to each individual provider.
5) Within each specialty, select the provider with the largest number of visits (frequency).
   a. If two providers of the same specialty have the same, largest number of visits, select the provider with the most recent visit.
   b. If multiple providers share the same, largest number of visits and the most recent visit date, choose the one with the maximum value of provider_id (an arbitrary rule for guaranteeing only one provider per specialty).

SAS offers several techniques to encode this logic. This paper presents three of them:

- **SQL** – Structured Query Language, set-based statements and logic familiar to many people
- **DOW** – Do loop of Whitlock, or the Dorfman-Whitlock loop, explained completely in other papers (e.g., Dorfman, 2002 and Dorfman, 2009) and outlined here
- **HoH** – Hash of hashes, used in other languages and posted to SAS-L in May 2004.

SAMPLE INPUT DATA

Consider a set of healthcare claims displayed in Table 1 with the following key fields; Patient_id, Service_date, Provider_id and Specialty. For illustration purposes, we show a single patient (ID 00004) who had multiple visits to

healthcare providers in 2010. This patient consulted with two different chiropractors (specialty = 35): five consultations with provider 124688 and one with 003663. The patient also had several visits to General Practitioners (specialty = GP): two visits with 075224 and two with 097323.

Patient_id	Service_date	Provider_id	Specialty
00004	01/10/2010	124688	35
00004	02/23/2010	124688	35
00004	05/02/2010	075224	GP
00004	06/16/2010	124688	35
00004	07/26/2010	003663	35
00004	07/27/2010	097323	GP
00004	08/29/2010	097323	GP
00004	09/05/2010	124688	35
00004	10/20/2010	075224	GP
00004	12/01/2010	124688	35

Table 1. Sample Input Data

The data file is sorted by Patient_id and then by Service_date. Before starting to write any code, we can manually apply the desired logic to this patient and come up with the following results:

This patient's chiropractor is 124688 by virtue of a frequency count of 5. The patient visited two general practitioners, and each had two visits. The date-based tie-breaker rule is invoked and 075224 wins attribution by virtue of having the patient's most recent visit on 10/20/2010.

One patient's data would be inadequate to test code fully, but it will serve to illustrate each coding approach.

DESIRED OUTPUT

Another piece of information you need before developing your code is the specification for the output. What information will be required, and in what format? Let's assume the following data structure will meet the user's needs:

Patient_id	Specialty	Attributed_provider	Last_service_date	Visit_count
00004	GP	075224	10/20/2010	2
00004	35	124688	12/1/2010	5

Table 2. Desired output

SQL

SQL allows for processing data in a set[1] based manner. In SQL, rows of a table are members of set and have no inherent ordering. Each member of a set has a collection of associated values which correspond to the columns of a table. A collection of rows are grouped together using the GROUP BY clause.

PROC SQL contains a feature not commonly seen in other products (e.g., Oracle, DB2); the ability of the SELECT statement to use an aggregate (or summary) function on a variable and have the result automatically merged with each contributing detail record. That feature is leveraged several times in the SQL code to accomplish patient-provider attribution.

The set-based nature of SQL requires that sequential business logic be coded as nested sub-queries. In terms of the sample data we will have an innermost query evaluating the highest frequency, wrapped by a middle query focused on visit date, which is wrapped by a query about provider for a final tie breaking evaluation.. Adding more

[1] As in a mathematical set that is a collection of distinct objects.

requirements to the problems, such as choosing randomly, or selecting based on the order in which the claims appear, would pose more of a challenge.

```
libname mylib '/mydata/SGF_2011';

proc sql;
create table mylib.attrib as

select patient_id
     , specialty
     , provider_id as attrib_prv
     , Visit_count as attrib_count
     , last_date as attrib_dt
from
 (select * from

        (select * from

              ( select specialty length=2 format=$2.
                ,patient_id
                ,provider_id
                ,count(*) as visit_count format=4.
                ,max(service_date) as last_date
                    format=yymmdd10.
              from mylib.sample
              group by patient_id, provider_id, specialty)


        group by patient_id, specialty
        having visit_count = max (visit_count))



 group by patient_id, specialty
 having last_date = max(last_date))



group by patient_id, specialty
having provider_id = max(provider_id)

;
quit;
```

This section defines a summary of the claims data by patient, provider and specialty, creating a count of visits and calculating the latest date of any visit of that patient to that provider.

This section looks at the summary defined above and keeps only the records where the count of visits is equal to the maximum count of visits of that patient to any provider in that specialty.

This section reviews the set of records with the maximum count of visits and keeps only the records where the last visit date to that provider is equal to the last visit date of that patient to any provider of that specialty.

This forces the choice of only one provider within a specialty. It operates on a set defined to contain only the providers with the maximum number of visits and the latest visit date within each specialty. If there are two providers who meet those criteria, this one chooses the larger value of provider identifier.

DOW

In a paper written for the 2002 SESUG conference called "The Magnificent DO," Paul Dorfman (crediting Ian Whitlock as the originator) defined a particular use of the DO Loop Structure as a DOW-loop. This technique generally applies to a situation in which the incoming data are sorted and grouped by an identifier (such as, in our case, the patient_id, specialty and provider_id) and the goal is to process all the records within an identifier and perform some action (such as compare with previously seen providers, or output) when you encounter the last record in the by-group. The general case of the DOW-loop looks like this (from Paul Dorfman's paper):

```
data ... ;
  <stuff done before break-event> ;
  do <Index Specs> until ( Break-Event ) ;
    set A ;
    <stuff done for each record> ;
  end ;
  <stuff done after break-event... > ;
run ;
```

Exploiting the DOW-loop for attribution of patients to providers requires the claims data to be sorted by patient_id, specialty and provider_id. Note that the question of efficiency/performance is addressed later in the paper.

The code that will accomplish attribution via the DOW-loop contains multiple DO loops because tracking variables must be initialized and checks accomplished at different break points.

The first loop repeats for all the records within a given patient. Prior to processing each specialty, the date of attribution and count of visits for that specialty are initialized.

The second loop executes for all the records within a given specialty. Before we start each new provider, we have to initialize the counters that we will use to record the number of visits and latest date for that provider.

The third, innermost loop is where we examine all the claims from a given provider. We count them and figure out the latest visit date. After we have seen all the records for that provider, we compare the visit count and latest date to what we have seen before for providers within that specialty. If this provider has more visits, or an equal number of visits but a later visit date, than the best provider we have seen before for this specialty, we update the information about the best provider and then loop back up to examine the next provider.

After we have seen all the records for a given specialty, we know who the best provider is, and we simply output the record.

If there are more records (another specialty) for this same patient_id, we go through the specialty loop again.

If there are no more records for this patient, we reach the normal end of the data step. There is no automatic output at this point because we have an explicit output statement at the end of each specialty. As long as there are more records in the incoming dataset, the data step code will be executed again from the top.

```
libname mylib '/mydata/SGF_2011';
data mylib.attrib_dow;
   do until (last.patient_id);                          * First loop *;
     attrib_dt = .;
     attrib_count = .;
     do until (last.specialty);                         *Second loop *;
        vis_count = 0;
        last_date = .;
        do until (last.provider_id);                    * Third loop*;
           length specialty $ 2;
           set mylib.claims;
           by patient_id specialty provider_id;
           vis_count + 1;
           if service_date > last_date then
            last_date = service_date;
        end;
```

```
      if  vis_count > attrib_count then do;
         attrib_count = vis_count;
         attrib_prv = provider_id;
         attrib_dt = last_date;
      end;
      else
      if vis_count = attrib_count and last_date > attrib_dt then do;
         attrib_prv = provider_id;
         attrib_dt = last_date;
      end;
      else
      if vis_count = attrib_count and last_date = attrib_dt and
         provider_id > attrib_prv then do;
         attrib_prv = provider_id;
      end;
   end;
  output;
end;
```

one attrib record per patient_id specialty;

```
run;
```

HOH: HASH OF HASHES

If you have not used hash tables to join data before, you should read Jason Secosky and Janice Bloom's paper "Getting Started with the DATA Step Hash Object." In that paper, they offer a succinct description of the hash object:

> "The hash object is an in-memory lookup table accessible from the DATA step. A hash object is loaded with records and is only available from the DATA step that creates it. A hash record consists of two parts: a key part and a data part. The key part consists of one or more character and numeric values. The data part consists of zero or more character and numeric values.
>
> Once a hash object is loaded with records, a lookup occurs by passing a key to the hash object's FIND method. If a record with the particular key is found, the data part of the record is copied into DATA step variables. In addition to being able to add and find records, there are methods to replace records, remove records, and output records to a data set."

In addition to hash tables, the code below makes use of hash iterator objects. Secosky and Bloom have a paper on this topic as well, "Getting Started with the DATA Step Hash Iterator". Quoting from that paper:

> "The hash iterator works with a hash object and allows another means to access values in the hash object without using a key lookup. Methods are used with the hash iterator to point to an item in the hash object based upon location, not value."

Armed with those definitions, we will proceed with the code.

The next technique exploits **hash tables** and **hash iterators** in an unusual way. As described above, a hash record consists of a key part and a data part. In the following code, records in the hash object **specialties** consist of a key part plus a reference to another hash object and reference to an iterator. Each record in **specialties** refers to a hash table **providers**, which will be loaded for each patient with the claims for a particular specialty, and to an iterator **providersIterator**, which will be used to look at each record in the **providers** hash table for that specialty sequentially to select the appropriate provider for that patient within that specialty. When the work for a particular patient is finished, the hash objects will be cleared of records in preparation for the next patient's data.

The code includes some elements that are available only in SAS 9.2; these are noted. The technique itself does work in SAS 9.1 but the efficiency aspects must be handled more manually. A schematic model that might help you think about the hash table relationships is presented in Figure 1. The colors in the figure correspond to the highlights in the code below. Note: The figure is not intended to represent how the hash objects or iterators are actually implemented.

```
libname mylib '/mydata/SGF_2011';
data mylib.attrib(keep=patient_id specialty attrib_prv attrib_dt attrib_count);
```

```
  length specialty $ 2 provider_id $ 15;
  declare hash specialties ();
  specialties.defineKey ('specialty');
  specialties.defineData ('specialty', 'providers',
                          'providersIterator');
  specialties.defineDone ();

  declare hiter specialtiesIterator('specialties');
  declare hash providers;
  declare hiter providersIterator;
  do until (endOfDataset);
```

** hash of hashes;*

** data element **providers** is an anonymous hash object and **providersIterator** an anonymous hash iterator;*

** anonymous hash: no key or data defined yet;*
** anonymous hash iterator;*
** loop over entire dataset*;*

```
    do until (last.patient_id);
      set mylib.claims end = endOfDataset;
      by patient_id;

      if specialties.find() ne 0 then do;


        providers = _new_ hash (ordered:'a');
        providers.defineKey ('provider_id');
        providers.defineData ('provider_id', 'specialty',
                              'rcnt_dt', 'count');
        providers.defineDone ();

        providersIterator = _new_ hiter('providers');
        specialties.add();
      end;
      if providers.find() = 0 then do;
        count = count + 1;
        rcnt_dt = max(rcnt_dt, service_date);
      end;
      else do;
        count = 1;
        rcnt_dt = service_date;
      end;


      providers.replace ();

    end;
```

** When SET is executed the variable 'specialty' receives a value from mylib.claims;*

** `FIND()` is used to check whether a **providers** hash already exists for this specialty;*

** create a hash for the providers of this specialty;*

** insert an element whose data are the references to the hash and its iterator;*
** Retrieve from the hash table the count and most recent date for this provider_id. If this provider_id has been seen before then increment count and update host variable rcnt_dt *;*
** If this provider has not been seen before, start the count at 1 and initialize the rcnt_dt to the service_date of this claim*;*

** update the data elements of the specialty's anonymous hash;*

** end **do until** last.patient_id;*

** Now we need code to walk the HoH table and select the appropriate provider for each of the hash table entries (one hash table per specialty) *;*

```
  do while(specialtiesIterator.next() = 0);



    if providers.num_items > 0 then do;
      max = 0;
```

** Each data element in **specialties** contains a reference to a hash containing providers of a specialty. `NEXT()` points to each record of **specialties** in succession and updates the host variables **providers** and **providersIterator**;*

**If there are any provider records in this table, proceed with the logic to select one *;*

```
      do while (providersIterator.next() = 0);
        if count > max then do;
          attrib_prv = provider_id;
          attrib_dt = rcnt_dt;
          attrib_count = count;
          max = count;
        end;
        else if count = max then do;
          if rcnt_dt > attrib_dt then do;
            attrib_prv = provider_id;
            attrib_dt = rcnt_dt;
            attrib_count = count;
          end;
          else if rcnt_dt = attrib_dt then do;
            if provider_id > attrib_prv
            then attrib_prv = provider_id;
          end;
        end;
      end;
      OUTPUT;
      providers.clear();

*The clear() method is not available in 9.1—it was introduced in 9.2  For 9.1, clear the
table manually as instructed in Secosky Iterator paper.*;
    end;

  end;

end;
run;
```

*attribution by highest frequency;

* attribution by date based tie breaker;

* same count, same last date, take max prv_id *;

*one record per patient_id per specialty *;

* Remove the rows from the hash table for this specialty, but do not delete the structure.

* END if providers.num_items > 0;

* END do while (specialties.next() > 0);

* END do until endofDataset *;

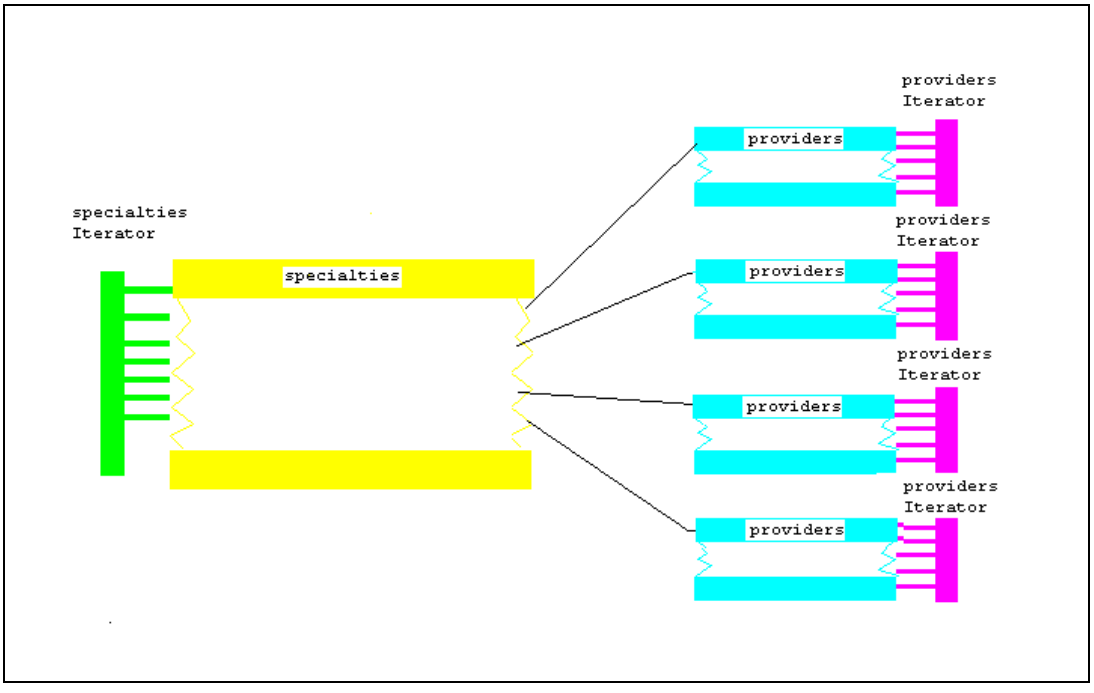

Figure 1. Schematic model of Hash of Hashes

PERFORMANCE COMPARISON

While it's interesting to explore a variety of techniques for implementing logic, sooner or later someone will ask, "Which one is best?" Almost any definition of "best" would include fastest execution time. Table 3 compares CPU and real execution times for the three techniques above on a Red Hat Linux server running 32-bit SAS 9.2

Technique	Obs IN	Obs OUT	CPU Time	Real time
SQL	2,777,502	1,192,711	11.93 seconds	9.27 seconds
DOW	2,777,502	1,192,711	4.29 seconds	4.56 seconds
HoH	2,777,502	1,192,711	18.73 seconds	18.85 seconds

Table 3. Execution times

Of the three techniques, DOW has the fastest execution time. However, that approach depends on having the incoming data sorted by patient_id, specialty and provider_id. Doing that sort on the test data used here took 26.1 seconds (real time). This might be too much of a burden for a one-time analysis, but repeated use of data after the sort would justify the investment.

SQL performs well on unsorted data, but it has limitations in terms of the information it can return in that time. For example, if we have to keep track of which rule was required for each attribution, the SQL would be more complex and take longer to run.

The HoH technique took the longest to run. It also depends on having the data sorted, but it's more flexible in that it can handle any sort order within patient_id.

ADDING COMPLEXITY

What if the rules for determining the appropriate provider within a specialty are more complicated? How would that affect each of the techniques?

Consider the following enhancement: If a patient has seen only one provider of a certain specialty in the last 12 months (not the last 18 months), and the patient has seen that provider at least twice in the last 12 months, choose that provider, regardless of what happened prior to the most recent 12 months. Otherwise, continue with the ruleset described initially.

This requirement obviously complicates the code because now instead of simply relying on filtering the incoming dataset on date, the code itself has to consider the date range to differentiate between last 12 months and last 18 months. It also multiplies the sets required in the SQL approach because instead of just choosing all the providers that meet certain criteria at each step and continuing to narrow one original set, the choice now depends on comparing elements in multiple sets.

Another requirement that would change the comparison among the techniques in both coding ease and performance would be keeping track of which rule resulted in the attribution. Is this provider a 12-month lone provider? Or a tie-breaker for whom there is another provider of equal claim to the attribution?

It is beyond the scope of this paper to accomplish these and other enhancements. They are presented to put the performance comparisons in context. The Hash of Hashes is complicated to code, but it may be just the solution to a particular set of requirements and data configuration.

CONCLUSION

The purpose of this paper was to put into the body of user group papers the Hash of Hashes technique. The particular problem chosen for the illustration is fairly common among analysts in the healthcare field. The simplified ruleset can be implemented a number of ways. SQL is compact, elegant, and more efficient in solving the simplified problem, and it doesn't require the incoming data in a particular sorted order. The DOW technique is blazingly fast if the data are already sorted in patient_id, specialty, provider_id order. The addition of requirements can change the relative performance. This doesn't invalidate any of the techniques; it only speaks to the need to have multiple tools at your fingertips when you are faced with complex programming tasks. Techniques added in SAS 9.2, such as the .clear() method, improve the efficiency of both runtime and coding time for the Hash of Hashes approach.

REFERENCES

- Chakravarthy, Venky . 2003. “The DOW (not that DOW!!!) and the LOCF in Clinical Trials .” *Proceedings of the SAS Users Group International 2003 Conference*. Seattle, WA: SAS Institute. Available at http://www2.sas.com/proceedings/sugi28/Proceed.pdf
- Dorfman, Paul. 2002. “The Magnificent DO.” *Proceedings of the SouthEast SAS Users Group 2002 Conference*. Savannah, GA: SESUG. Available at http://www.lexjansen.com
- Dorfman, Paul and Vyverman, Koen . 2009. “The DOW-Loop Unrolled .” *Proceedings of the SAS Global Forum 2009 Conference*. Washington, DC: SAS Institute. Available at http://support.sas.com/resources/papers/proceedings09/TOC.html
- DeVenezia, Richard A.. “SAS Code Samples.” *Data Processing, hash-6*. January 7, 2011. Available at http://www.devenezia.com/downloads/sas/samples/.
- Secosky, Jason and Bloom, Janice. “Getting Started with the DATA Step Hash Object.” *Proceedings of the Pacific Northwest SAS Users Group 2006 Conference*. Seaside, OR: PNWSUG. Available at http://www.lexjansen.com/pnwsug/2006/Secosky-PNWSUG06.pdf.
- Secosky, Jason and Bloom, Janice. “Getting Started with the DATA Step Hash Iterator. SAS Institute. Available at http://support.sas.com/rnd/base/datastep/dot/iterator-getting-started.pdf (accessed February 5, 2011)

ACKNOWLEDGMENTS

RECOMMENDED READING

- Paul Dorfman has a number of papers on the hash object which will be helpful to any user trying to learn more about it.
- Loren, Judy. 2008. “How Do I Love Hash Tables? Let Me Count the Ways!” *Proceedings of the SAS Global Forum 2008 conference*. San Antonio, TX: SAS Institute. Available at http://www2.sas.com/proceedings/forum2008/029-2008.pdf

CONTACT INFORMATION

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

Judy Loren
Health Dialog
2 Monument Square
Portland, ME 04101
jloren@healthdialog.com

Richard A. DeVenezia
Independent Consultant
9949 East Steuben Road
Remsen, NY 13438
rdevenezia@gmail.com