PharmaSUG 2013 - Paper BB01

The Hash of Hashes as a "Russian Doll" Structure: Application to Clinical Adverse Events Data Analysis

Joseph Hinson, Princeton, NJ

ABSTRACT

The SAS® DATA Step hash objects have the unique ability to be generated as a nested structure, popularly known as "hash-of-hashes". Such structures are particularly well-suited for the processing and regrouping of hierarchical data, by splitting data into distinct tables within tables. In this paper, this approach has been applied to the processing of clinical adverse events data, which often need to be organized into number of events within preferred terms within system organ classes within treatment groups. Such complex reorganization has been accomplished with a multilevel nested groups of hash objects, all within a single DATA Step, without any sorting or by-processing, and with just one pass through the unsorted raw data. It is yet another demonstration of the powerful flexibility and unique opportunities provided by the hash object technique.

INTRODUCTION

Ever since the introduction in SAS® 9 of DATA Step Component Objects, hash objects programming has gradually been gaining adoption in many programming areas. Primarily, the technique has been popular in programming tasks in which fast table look-ups and sortless merging are advantageous. But novel uses of the technique keep appearing. For instance the present author had demonstrated the use of hash objects for table transposition[1]. Very recently, it has been discovered and demonstrated that hash objects do allow other hash objects as their data[2]. This has opened the door to their application to the processing of hierarchical data. Such data structures require table nesting, analagous to "Russian Dolls", where bigger dolls contain smaller dolls, which also contain even smaller dolls. The phrase "hash of hashes" refers to a nesting structure of hash objects, with objects containing other objects as data, allowing a situation where tables can be organized in a hierarchical manner. Such hash-of-hashes technique has successfully been applied to very complex coding tasks, such as (in order of increasing complexity), the splitting of data sets into multiple files[2], the analyses of stock holdings[3,] and the processing of healthcare claims[4]. Much recently, the author has demonstrated the application of the technique to the creation of XML data structures[5]. We hereby present yet another demonstration of the capabilities of the hash-of-hashes technique, as applied to the processing of clinical adverse events data.

CLINICAL ADVERSE EVENTS AS HIERARCHICAL DATA

The analysis of adverse events is an essential part of safety monitoring and reporting during clinical trials. In the summarization of adverse events, the events are usually evaluated per a combination of subpopulations. Thus events can be counted per subject, per subject by treatment, per subject by system organ class, per subject by system organ class by treatment, per subject by system organ class by preferred term, et cetera. Furthermore, these subpopulations tend to be hierarchically classified. For instance, preferred terms are a subset of a system organ class, which could be a subset of a treatment arm. The processing of this type of hierarchical organization of data presents unique programming challenges, and many have managed the task by relying on a series of DATA steps with a multitude of subtables, and several sorting and merging routines.The merges require the presence of common variables, a situation not easily applicable to hierarchical data. The counts themselves are usually obtained with the FREQUENCY Procedure, which generates its own output table. Often such tables require transposition and further processing.The present paper will demonstrate that the hash-of-hashes technique is well-suited for such types of hierarchical data organization and analysis.For the sake of simplicity, other adverse events classifications are excluded in this paper, such as, "causality" and "severity/intensity.

HIERARCHICAL DATA STRUCTURES

Like a Russian doll, the hierarchy of adverse event data can be considered as containers within containers, as shown in Figure 1. The outermost container would hold the treatment arms, which would then contain sytem organ classes, which would also contain a set of preferred terms. Then each preferred term could store the number of events for that particular term. Because such a layout is actually tables within tables, one can easily see how the structure can be modeled by hash objects containing other hash objects.

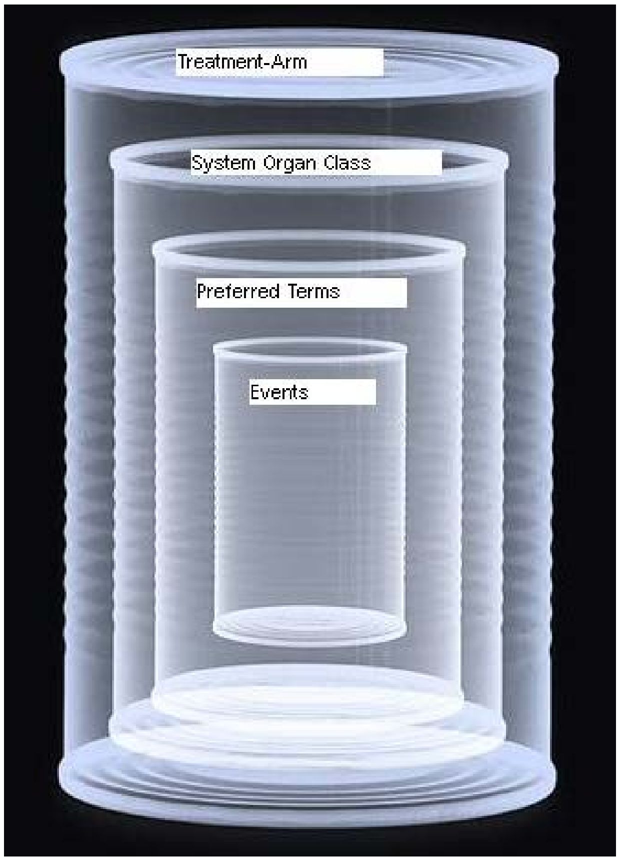


Figure 1. A Hierarchical Model of Adverse Events

HASH OBJECTS

SAS® hash objects are DATA Step Component Objects which are RAM memory-resident tables and are called "objects" because each one possesses its own methods and attributes. Each record in a hash table has a *key* part and a *data* part. Hash objects (also called "associative arrays") permit the quick retrieval of data based on the lookup key. Each lookup key is passed through a "hash" (mathematical function) which maps the key to a record in the hash table. The hash iterator object is a sort of a "table navigator", that allows data items in a hash table to be accessed row by row, without reliance on lookup keys. In this paper, the hash iterator object methods, **object.first()** and **object.next()**, were used for the hierarchical accessing of data for consolidating all the nested hash objects into a final single summary table. (The methods were also used for generating intermediate dataset outputs for illustrative purposes just for this paper).
The hash methods **object.find()** and **object.replace()** were widely used to look for data and to place new data in a hash table. One hash attribute very useful for this paper was **object.num_items**. All adverse event counts in the present paper were obtained with the object.num_items attribute, making it unecessary to rely on PROC FREQ.
Before any hash or hash iterator object can be used, they have to be *declared* and *instantiated*.

The are two ways of accomplishing this:

(a) Declare and instantiate with a single statement:

```
declare hash ta();
```

(b) Declare first, then instantiate when necessary:

```
declare hash soc;
soc=_new_ hash();
```

This second 2-step method is one of the tools that make hash-of-hashes possible.
A hash object is declared once, but reused many times by dynamic instantiation. The re-use allows hash objects *within* hash objects to be filled.

THE HASH-OF-HASHES TECHNIQUE

Until the discovery was made by Richard DeVenezia, it was assumed hash tables could not contain other hash tables. And ever since the introduction of hash objects, the conventional practice had been to let hash tables contain just numeric and character data. But DeVenezia and Dorfman demonstrated that indeed hash tables could contain references to other hash tables as data, using data set splitting to illustrate the concept[2]. The gem of the technique is the ability to use dynamic instantiation of hash objects (using **object=_new_ hash()**) to insert hash objects into other hash objects *only* when a new group of data is encountered. So rather than declaring a fixed number of hash objects in advance, the objects are created on the fly as needed. Such multiple instantiations allow data to be compartmentalized in a nested fashion.

THE VARIABLES AND OBJECTS USED IN THE CODE:

(a) Names of Variables:

trt:	for Treatment Arms (0=Active; 1=Placebo)
aebodsys:	for System Organ Classes (also called Body Systems), eg. "Cardiac Disorders"
aedecod:	for Preferred Terms of adverse events, eg. "Palpitations"
subjid:	for Patient ID numbers

(b) Names of Hash Objects:

ta:	hash container for treated groups for all patients (including patients without adverse events)
sub:	hash container for subject IDs of all treated patients (including patients without adverse events)
drug:	hash container for treated groups for only adverse events patients
soc:	hash container for system organ classes
ae:	hash container for adverse events preferred terms
pat:	hash container for subject IDs of adverse event patients

(c) Names of Hash Iterator Objects (*also called "hiter"*):
(The iterator object allows data to be read from hash tables by navigating the table row by row).

hidrug:	iterator object for navigating the **drug** hash table
hisoc:	iterator object for navigating the **soc** hash table
hiae:	iterator object for navigating the **ae** hash table
hipat:	iterator object for navigating the **pat** hash table

(d) The Hash Hierarchy:

```
Level-0:  Hash drug (for treatment groups (0 and 1)):
                                    ---contains--
          Level-1:  Hash soc (for System Organ Classes (SOC)):
                                              ---contains--
                    Level-2:  Hash ae (for Preferred Terms (AE)):
                                                        ---contains--
                              Level-3:  Hash pat (for Subject IDs):
```

THE CODING STRATEGY

1. Empty hash and iterator containers (objects) are first declared for the various levels of nested tables
2. Four pieces of data are read from each observation of AEDATA (adverse events data set):
   subjid, trt, aebodsys, aedecod
   and allowed to be segregated into various hash containers, forming a multilevel nested groups of hash tables:
   (a) the keys of the various hash objects determine which data item would be captured.
   (b) objects within other hash objects are dynamically created and filled with data

(Hash objects can be defined for re-use by the definition statement, ***object=_new_ hash()*** which allows dynamically created hash objects. Thus the same object can be initialized for different groups but with different keys and data.
For instance, the same ***soc*** hash object can be initialized to hold cardiac disorders, and then re-initialized to contain psychiatric disorders another time, etc.
Through the re-initializion of hash objects with the ***object=_new_ hash()*** method, data get segregated into different hash tables in a key-based manner. This is the core tool of the hash of hashes technique).

Below are actual code fragments to illustrate the process:

i. The "trt" value is used as key to put a "**soc**" hash and "**hisoc**" hiter in the ***drug*** hash object:

```
drug.defineKey("trt");
drug.defineData("trt","soc","hisoc");
```

ii. That "**soc**" hash object itself gets an "**ae**" hash object and "**hiae**" hiter using "aebodsys" as key:

```
        soc= new  hash();
        soc.defineKey("aebodsys");
        soc.defineData("aebodsys", "ae","hiae");
```

iii. The "**ae**" hash object similarly acquires "**pat**" and "**hipat**" objects using "aebodsys" and "aedecod" as keys:

```
                ae=_new_ hash();
                ae.defineKey("aebodsys","aedecod");
                ae.defineData("aedecod","pat","hipat");
```

iv. The final hash table, **pat**, is made to contain only numeric data, "subjid":

```
                        pat= new  hash();
                        pat.defineKey("k","aebodsys","aedecod","subjid");
                        pat.defineData("subjid");
```

In the above examples, the instantiations of hash objects with " ***new hash"*** are done only if the objects do not already exist. The **object.find()** method is used for that purpose, and returns a code "0" if the object exists.

3. The nested hash tables are consolidated, by reading data, level by level, to create a single hash summary table.
(The nested hash tables can also be individually converted to data sets for output. This has been done in this paper for illustrative purposes but not included in the code).

THE COMPLETE CODE:

```
data aedata;

        input subjid $ 1-3 trt 5 aebodsys $ 6-31 aedecod $ 36-53 ;
        datalines;
101 1 Cardiac disorders                  Atrial flutter
101 0 Gastrointestinal disorders Constipation
102 1 Cardiac disorders                          Cardiac failure
102 1 Psychiatric disorders                      Delirium
103 1 Cardiac disorders                          Palpitations
103 1 Cardiac disorders                          Atrial flutter
103 1 Cardiac disorders                          Tachycardia
104 1
105 0
106 0
107 1
108 1
109 0
110 1
```

```
111 0
112 0
113 0
114 1
115 0 Gastrointestinal disorders  Abdominal pain
115 0 Gastrointestinal disorders        Anal ulcer
116 0 Gastrointestinal disorders  Constipation
117 0 Gastrointestinal disorders  Dyspepsia
118 0 Gastrointestinal disorders  Constipation
119 0 Gastrointestinal disorders  Constipation
120 1
121 1
122 0
123 1
124 0
125 1
126 1
127 0
128 1
129 1
130 1 Nervous system disorders          Convulsion
131 1 Nervous system disorders          Dizziness
132 0 Nervous system disorders          Essential tremor
133 1
134 0
135 1 Psychiatric disorders                    Delirium
136 1
137 0
138 1
139 1
140 1 Psychiatric disorders                    Delirium
140 1 Psychiatric disorders                    Sleep disorder
141 1 Cardiac disorders                        Palpitations
142 0
143 1
144 0
145 1
146 1
147 0
148 1
149 1
150 1
;
run;

data _null_;
       k=0;
       *-------------------------------------------------------------------------------
             [a] C R E A T I O N   O F   F I X E D   H A S H   O B J E C T S
                          (for treatment arms and SOCs)
       -------------------------------------------------------------------------------;
       if (1=2) then set aedata;

       ****FOR ALL-TREATED TOTALS;
       declare hash ta();
              ta.defineKey("trt");
              ta.defineData("sub");
              ta.defineDone();
       declare hiter hita("ta");

       ****FOR SOC PROCESSING;
       declare hash drug();
              drug.defineKey("trt");
              drug.defineData("trt","soc","hisoc");
              drug.defineDone();
       declare hiter hidrug("drug");

       *---------------------------------------------------------------------------
       [b] C R E A T I O N   O F   R E U S A B L E   H A S H   O B J E C T S
                          (for making hashes within hashes)
       ---------------------------------------------------------------------------;
       declare hash sub;
       declare hash soc;
       declare hiter hisoc;
       declare hash ae;
       declare hiter hiae;
       declare hash pat;
       declare hiter hipat;

       *---------------------------------------------------------------------------
       [c]  G R O U P  I S O L A T I O N  B Y  H A S H - W I T H I N - H A S H E S
       ---------------------------------------------------------------------------;
       do until(done);
              set aedata end=done;
              k=k+1;
              ****FOR ALL TREATED PROCESSING;
              rcta=ta.find();
              if rcta ne 0 then
                            do;
                                   sub=_new_ hash();
                                   sub.defineKey("k","trt");
                                   sub.defineData("subjid");
                                   sub.defineDone();
                                   rctax=ta.replace();
                            end;
                            rcsub=sub.replace();

              if (aebodsys ne '') then
              do;
              ****FOR SUB-GROUP PROCESSING;
                     if drug.find() ne 0 then
                     do;
                            soc=_new_ hash();
                            soc.defineKey("aebodsys");
                            soc.defineData("aebodsys", "ae","hiae");
                            soc.defineDone();
                            hisoc= new  hiter("soc");
                            drug.replace();
                     end;
              if soc.find() ne 0 then
                     do;
                            ae=_new_ hash();
                            ae.defineKey("aebodsys","aedecod");
                            ae.defineData("aedecod","pat","hipat");
                            ae.defineDone();
                            hiae=_new_ hiter("ae");
                            soc.replace();
                     end;
              if ae.find() ne 0 then
                     do;
                            pat=_new_ hash();
                            pat.defineKey("k","aebodsys","aedecod","subjid");
                            pat.defineData("subjid");
                            pat.defineDone();
                            hipat= new  hiter("pat");
                            scheck=ae.replace();
```

```
                end;
                        r2check=pat.replace();
        end;*[if aebodsys...];
end;*[do until done…set aedata...];
call missing(trt,aebodsys,aedecod,subjid);

*---------------------------------------------------------------------------
[d]  P R O C E S S I N G   O F   D A T A   W I T H I N   I N D I V I D U A L
                           H A S H   T A B L E S
---------------------------------------------------------------------------;
****FOR SUB-GROUPS;
do while (hidrug.next() eq 0);
        trtsum=drug.num_items;

                do while (hisoc.next() eq 0);
                        socsum=soc.num_items;
                        aesum=ae.num_items;

                        do while (hiae.next() eq 0);
                                wum=pat.num_items;
                                end;*hiae;
                end;*hisoc;
end;*hidrug;

*---------------------------------------------------------------------------
[e] C O N S O L I D A T I O N   O F   H A S H   T A B L E S   I N T O   O N E
                        S U M M A R Y   T A B L E
---------------------------------------------------------------------------;
length hash1 $10 hash2 $30 hash3 $50 hash4 $10;

declare hash aesumm(ordered:"a");
        aesumm.defineKey("row","hash1","hash2","hash3","Events","hash4");
        aesumm.defineData("row","hash1","hash2","hash3","Events","hash4");
        aesumm.defineDone();
        call missing(Hash1,Hash2,Hash3,count,Hash4);

        row=0;
        Hash1="TREATMENT";
        Hash2="SYSTEM ORGAN CLASS";
        Hash3="PREFERRED TERM";
        Events=.;
        Hash4="SUBJECT ID";
        rca=aesumm.add();
        call missing(Hash1,Hash2,Hash3,Events,Hash4);

****FOR SUB-GROUPS;
do while (hidrug.next() eq 0);
        tt=choosec((trt+1),"Active","Placebo");
        Hash1=tt;
        k=trt+1;
        row=row+1;
        hita.next();
        Events=sub.num_items;
        rc2=aesumm.add();
        call missing(Hash1,Hash2,Hash3,Events,Hash4);

do while (hisoc.next() eq 0);
        hash2=aebodsys;
        row=row+1;
        Events=ae.num items;
        rc2=aesumm.add();
        call missing(Hash1,Hash2,Hash3,Events,Hash4);
```

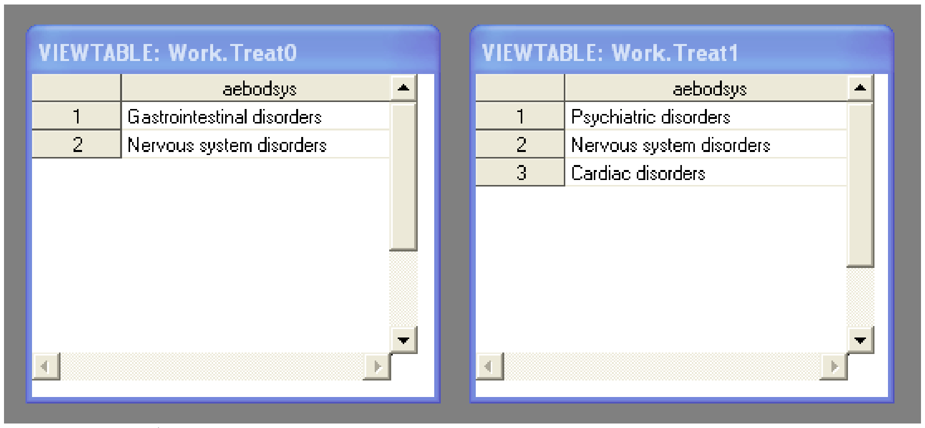
```
                    do while (hiae.next() eq 0);
                          Events=pat.num_items;
                          hash3=aedecod;
                          row=row+1;
                          rc3=aesumm.add();
                          call missing(Hash1,Hash2,Hash3,Events,Hash4);

                                do while (hipat.next() eq 0);
                                      hash4=subjid;
                                      row=row+1;
                                      rc4=aesumm.add();
                                      call missing(Hash1,Hash2,Hash3,Events,Hash4);
                                end;*hipat;
                    end;*hiae;
              end;*hisoc;
        end;*hidrug;
*-------------------------------------------------------------------------------------
             [f] F I N A L   O U T P U T   ( S U M M A R Y   T A B L E )
-------------------------------------------------------------------------------------;

rcs=aesumm.output(dataset:"aesummary");
stop;
run;
```

OUTPUT DISPLAYS

(1) INTERMEDIATE HASH TABLES:


VIEWTABLE: Work.Treat0

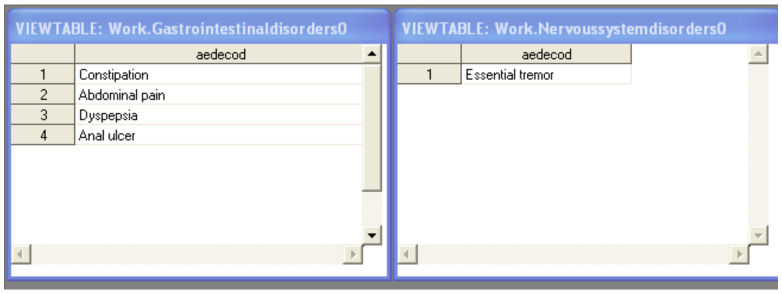
	aebodsys
1	Gastrointestinal disorders
2	Nervous system disorders

VIEWTABLE: Work.Treat1

	aebodsys
1	Psychiatric disorders
2	Nervous system disorders
3	Cardiac disorders

Display A. [1st Level Hash]: Treatment Hash Tables containing System Organ Class Hash Tables

Treatment-0:



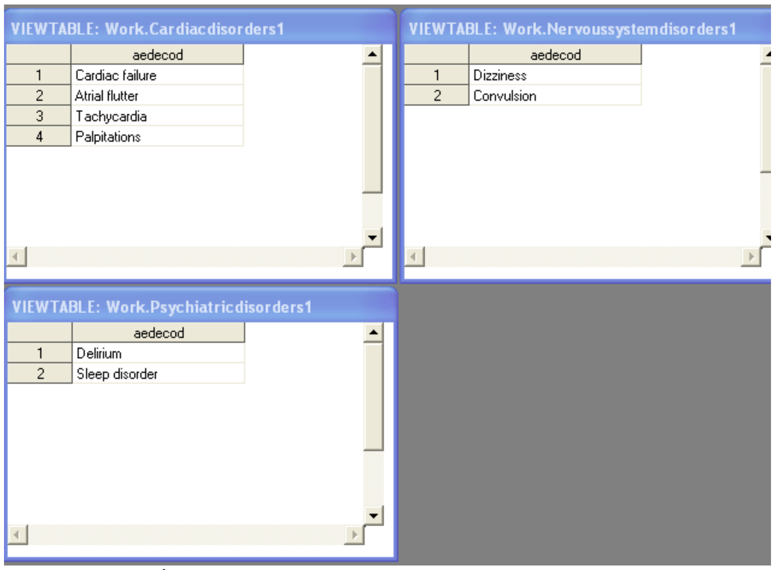
VIEWTABLE: Work.Gastrointestinaldisorders0

	aedecod
1	Constipation
2	Abdominal pain
3	Dyspepsia
4	Anal ulcer

VIEWTABLE: Work.Nervoussystemdisorders0

	aedecod
1	Essential tremor

Treatment-1:



VIEWTABLE: Work.Cardiacdisorders1

	aedecod
1	Cardiac failure
2	Atrial flutter
3	Tachycardia
4	Palpitations

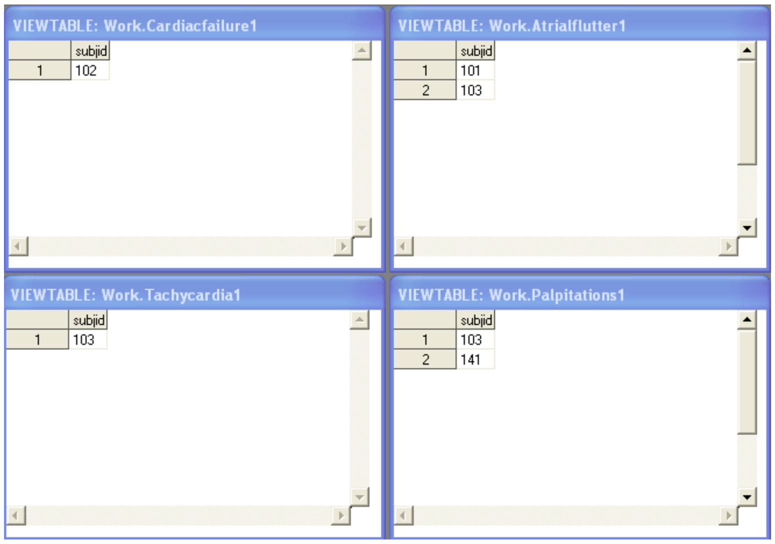
VIEWTABLE: Work.Nervoussystemdisorders1

	aedecod
1	Dizziness
2	Convulsion

VIEWTABLE: Work.Psychiatricdisorders1

	aedecod
1	Delirium
2	Sleep disorder

Display B. [2nd Level Hash]: System Organ Class Hash Tables containing Preferred Terms Hash Tables



VIEWTABLE: Work.Cardiacfailure1

	subjid
1	102

VIEWTABLE: Work.Atrialflutter1

	subjid
1	101
2	103

VIEWTABLE: Work.Tachycardia1

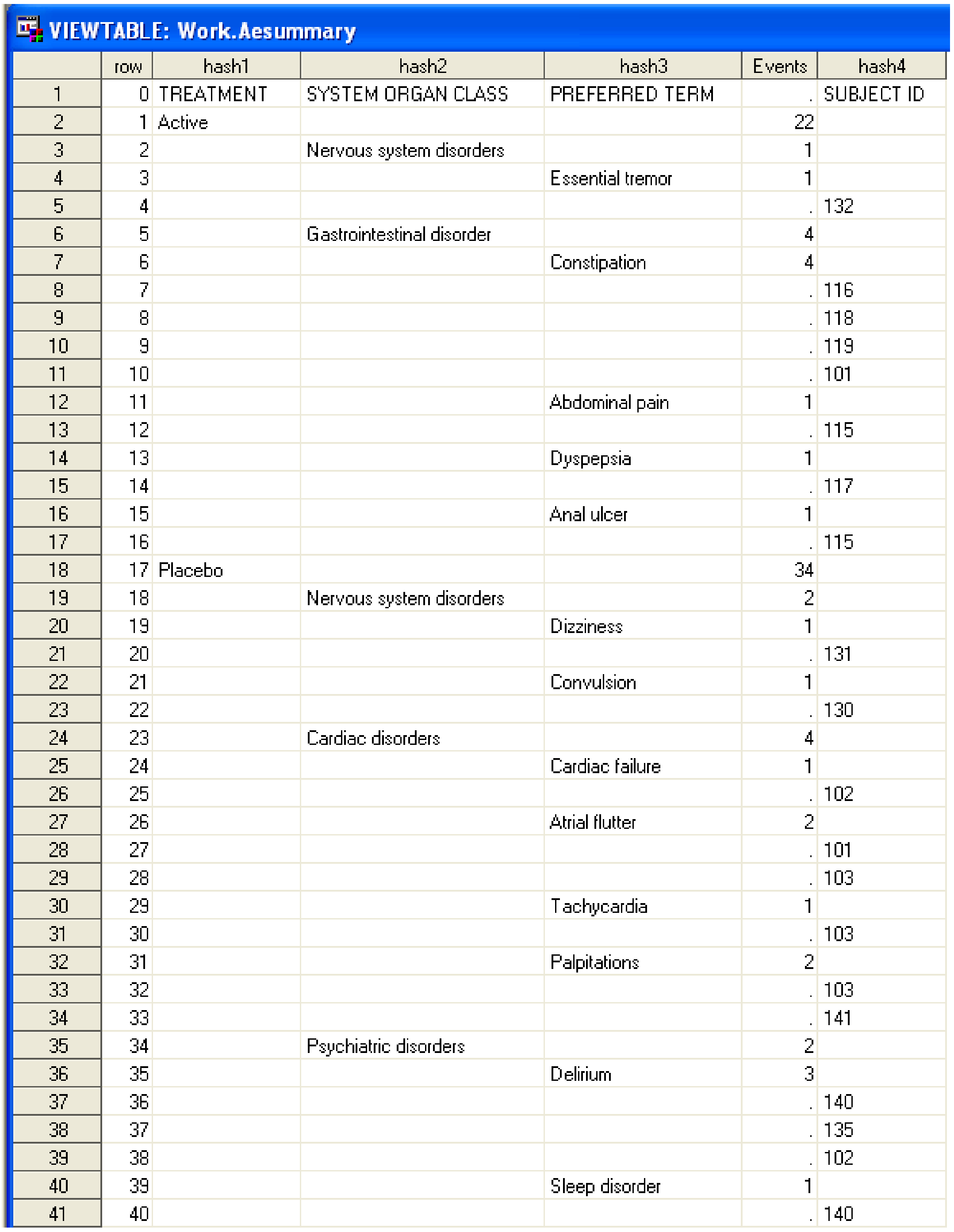
	subjid
1	103

VIEWTABLE: Work.Palpitations1

	subjid
1	103
2	141

Display C. [3rd Level Hash]: Preferred Terms Hash Tables (Cardiac Disorders) containing Subject ID

(2) FINAL OUTPUT:



VIEWTABLE: Work.Aesummary

	row	hash1	hash2	hash3	Events	hash4
1	0	TREATMENT	SYSTEM ORGAN CLASS	PREFERRED TERM	.	SUBJECT ID
2	1	Active			22	
3	2		Nervous system disorders		1	
4	3			Essential tremor	1	
5	4				.	132
6	5		Gastrointestinal disorder		4	
7	6			Constipation	4	
8	7				.	116
9	8				.	118
10	9				.	119
11	10				.	101
12	11			Abdominal pain	1	
13	12				.	115
14	13			Dyspepsia	1	
15	14				.	117
16	15			Anal ulcer	1	
17	16				.	115
18	17	Placebo			34	
19	18		Nervous system disorders		2	
20	19			Dizziness	1	
21	20				.	131
22	21			Convulsion	1	
23	22				.	130
24	23		Cardiac disorders		4	
25	24			Cardiac failure	1	
26	25				.	102
27	26			Atrial flutter	2	
28	27				.	101
29	28				.	103
30	29			Tachycardia	1	
31	30				.	103
32	31			Palpitations	2	
33	32				.	103
34	33				.	141
35	34		Psychiatric disorders		2	
36	35			Delirium	3	
37	36				.	140
38	37				.	135
39	38				.	102
40	39			Sleep disorder	1	
41	40				.	140

Display D. Putting it All Together: FINAL SUMMARY TABLE – "aesummary"

CONCLUSION

This paper has demonstrated that the hash-of-hashes technique can be applied in clinical trial data analysis, particulary to process data of the hierarchical type such as adverse events. When adverse events data are analyzed, counts of events are derived in various ways: counts by treatment groups, by system organ classes, or by preferred terms. Routinely, this is accomplished by many steps: data sorting, BY processing, and counts by the Frequency Procedure. The tables produced by PROC Freq often need to be restructured, sorted, and merged with other tables. Thus adverse event data processing can appear quite unwieldy. As demonstrated, the hash of hashes technique can be a useful alternative, channeling the data into various groups for counting, all within a single DATA step, without any sorting, merging, and the use of procedures.
Above all, the number of tables do not have to be known in advance, as would be the case if one had relied on DATA step By-processing or the SQL PROCEDURE. With the hash-of-hashes approach, tables are dynamically created at run-time as needed, and the same code without any modification would generate a different number of tables from a different set of adverse events.
It is expected that with the approach described in this paper, any hierarchical data type can be processed with the hash-of-hashes technique,

Potential Limitations:
Being memory-resident tables, hashes within other hashes could potentially create memory problems and performance issues when applied to very large data. However, by their very nature, adverse event data are expected to be relatively lean in size. (In fact, it is the ultimate dream of a drug sponsor that there would be ZERO adverse events)! The learning curve for the technique could also be much steeper, considering the unfamiliar dot notation and other "strange" syntax associated with object programming. Blank records also tend to create problems - a situation yet to be investigated by the author, and in this paper, the technique was made to apply to just records with adverse event data.

REFERENCES

1. Joseph Hinson and Changhong Shi
   “Transposing Tables from Long to Wide: A Novel Approach Using Hash Objects“
   Proceedings of the Pharmaceutical SAS Users Group, PharmaSUG 2012, Paper PO14

2. Dorfman Paul and Vyverman, Koen.
   “The SAS® Hash Object in Action”
   Proceedings of the SAS Global Forum 2009, Paper 153-2009
   http://support.sas.com/resources/papers/proceedings09/153-2009.pdf

3. Mark Keintz
   "From Stocks to Flows: Using SAS® HASH objects for FIFO, LIFO, and other FO’s"
   Proceedings of the 2011 Northeast SAS Users Group Conference, Paper
   http://www.nesug.org/Proceedings/nesug11/fi/fi03.pdf

4. Judy Loren and Richard A. DeVenezia
   "Building Provider Panels: An Application for the Hash of Hashes"
   Proceedings of the SAS Global Forum 2011, Paper 255-2011
   http://support.sas.com/resources/papers/proceedings11/255-2011.pdf

5. Hinson J, ”Processing of hierarchical data with hash objects Part 1 - Creation of XML documents”. *Pharmaceutical Programming*, Volume 5, Numbers 1-2, December 2012 , pp. 10-28(19), London, UK: Maney Publishing.

ACKNOWLEDGMENTS

The author wishes to thank his manager while at Merck: Margaret Coughlin, and his mentor Changhong Shi, for providing an intellectually stimulating environment for learning and doing creative work.

Thanks are also due to Dr. Paul Dorfman for providing a wealth of information and tutorials about the hash technique, and for generating so much enthusiasm and ideas for novel applications.

The author also wishes to acknowledge Richard DeVenezia for his innovative programming style, and for discovering the hash-of-hashes technique, providing many elegant examples, and making it possible for the author to develop code and produce this paper.

CONTACT INFORMATION

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

Name:	Joseph Hinson, PhD
City, State ZIP:	Princeton, NJ, 08542
Work Phone:	1-609-540-1309
E-mail:	jwhinson@gmail.com

sas Certified Advanced Programmer for SAS 9

sas Certified Clinical Trials Programmer Using SAS 9