harmaSUG 2017 - Paper BB08

USING HASH TABLES FOR AE SEARCH STRATEGIES

Vinodita Bongarala, Liz Thomas
Seattle Genetics, Inc., Bothell, WA

ABSTRACT

As part of adverse event safety analysis, adverse events of special interest (AESI) are identified within a study by a variety of means. Most commonly, AESIs can be identified with standardized MedDRA Queries (SMQs), MedDRA System Organ Classes (SOCs), or a customized collection of MedDRA preferred terms (PTs) or lower level terms (LLTs). Analysis of AESIs is similar, regardless of the search strategy used to identify them. Identifying AESIs may involve merging the AE dataset with multiple datasets on keys of different levels (LLT, PT or SOC). Using a hash table as opposed to multiple sorted or unsorted merges can simplify both the code and execution time. Faster processing with SAS® hash tables has considerable utility in large safety databases.

The efficiency gain of using hash tables over other methods of merging (both sorted and unsorted) has been well characterized. This paper demonstrates how to use hash tables to identify AESIs and two methods of structuring the resulting data for rapid subsequent tabulation and analysis, with benchmark comparison to using unstructured SQL merges to obtain the same results.

INTRODUCTION

Adverse Events analysis is most critical in determining the safety profile of any drug. It is therefore monitored frequently during every phase of a clinical trial both internally by the study team and pharmacovigiliance committees and by external bodies like data safety monitoring boards (DSMBs), Safety Monitoring Committees (SMCs) or regulatory authorities. An adverse event is indicative of some kind of negative impact on the patient's health and well-being. Some adverse events are unavoidable and they reflect the risk associated with the treatment.

Due to the level of detail present in the adverse event data, the presence of a medical condition may not be easily visualized when only summarizing Preferred Terms (PTs) or System Organ Classes (SOCs). MedDRA (Medical Dictionary of Regulatory Affairs) developed Standardized MedDRA Queries (SMQs) just to resolve this issue. SMQs provide a validated tool to identify all PTs considered to be related to a single condition or area of interest. The terms included relate to signs, symptoms, diagnoses, syndromes, physical findings, laboratory and other physiologic test data, etc. that are associated with the medical condition or area of interest. SMQs can be nested, and may include an algorithmic determination of a condition requiring comorbidity of multiple PTs.

NATURE OF ADVERSE EVENTS DATA

The complexity of adverse event analysis can be attributed to the fact that AE data is generally considered as hierarchical in nature. SOCs are a broad classification of the area of the disease, and the classification hierarchy is increasingly refined down to the lower level terms (LLTs), as displayed in the chart below.

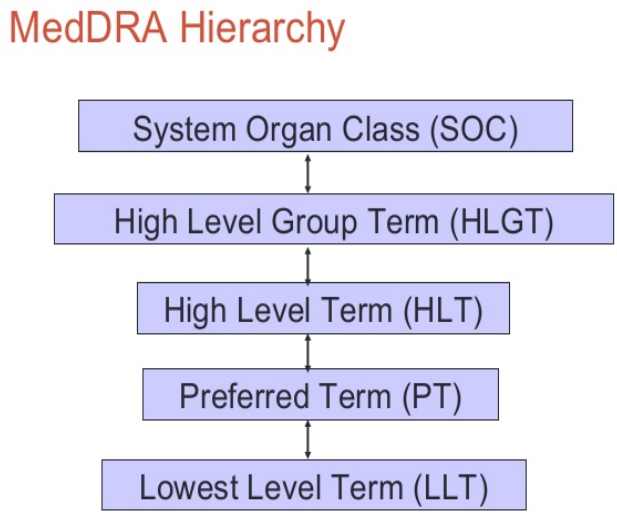

Figure 1: MedDRA Hierarchy

The hierarchy of the adverse events is not quite as linear as the chart above indicates. The classification is unique from LLT up to HLGT. However, the relationship between SOCs and the rest of the hierarchy is not a strict single classification. It is possible that a single PT can map to multiple SOCs. MedDRA terms this phenomenon as 'Multi-axiality'. One of the SOCs is assigned as the primary SOC, and should be used for overall AE summaries. However, when using SOC as a search strategy, all PTs that map to an SOC should be considered. In the below figure, the Influenza PT is classified under two SOCs, but 'Infections and Infestations' is assigned as the primary SOC.

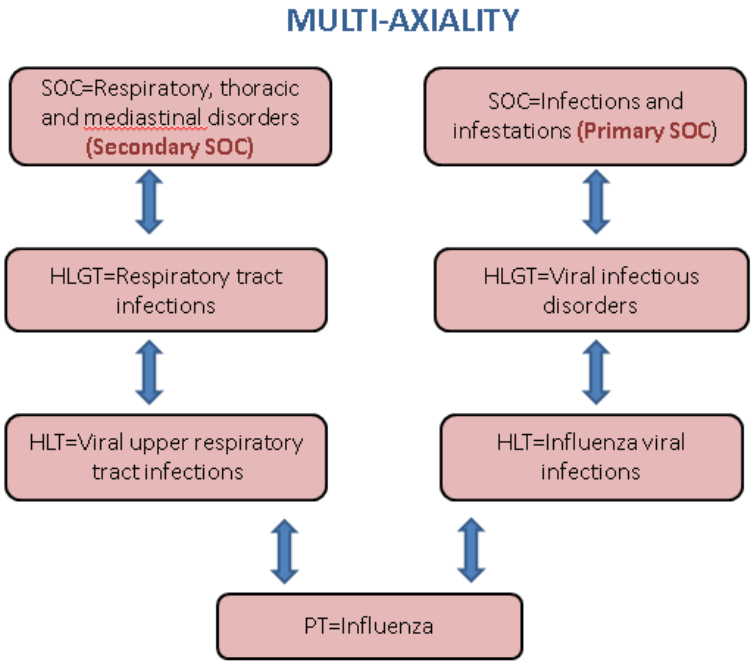

Figure 2: Multi-Axiality of the Influenza PT

HASH OBJECTS

Hash objects have been available for use in a DATA step since SAS® 9. They provide a nifty means of quick retrieval and storage of data. They reside in the memory and hence are quickly retrievable with reference to the lookup keys like array elements. Hash objects do not require the data to be sorted or

indexed. These can be interpreted as lookups that provide far faster results compared unsorted joins with proc SQL. This paper explores how hash objects can be efficiently used in the form of hash tables for implementing AE search strategies.

The structure of a hash object can be divided into two components: ***key*** component and ***data*** component. The key component maps the key values to data rows and it can be numeric or character in nature. The data component has the ability to store multiple data values per key value and it can consist of numeric or character values. The values of a hash table can be loaded from a SAS® dataset or hard-coded values and these exist only for the duration of the DATA step.

USING HASH TABLES TO IMPLEMENT AE SEARCH STRATEGY

Suppose a search strategy encompasses three levels: SMQs, SOCs and PTs. Since SMQs may be broken down by LLT, the only previous way to do this was with three separate joins:

- one on LLT level
- one on SOC level
- one on PT level

The same results can be obtained using the hash tables instead. Instantiating and referencing three hash tables in a single data step, we can pull out the LLT, SOC and PT matches with one pass through the data. In order to accomplish this assume three datasets have been created:

SMQ dataset (with LLT codes and search name)

SOC dataset (with SOCs and search name)

PT dataset (with PTs and search name)

Search name refers to the intent of a search, such as the AESI name or a description of the unifying search strategy (e.g. hepatotoxicity). Using a search name accommodates doing multiple searches in a single pass, creating outputs for multiple AESIs.

Example code:

```
data search_strategy(drop=rc llt_code aellt aeseq recordposition);

      if _n_=1 then do;
            length search_name $100 llt_code $160  scope $8
            soc_name $480 pt_name $480;
            declare hash smq (dataset: 'work.smq', multidata: 'y');
            rc=smq.defineKey('llt_code');
            rc=smq.defineData('search_name', 'scope');
            rc=smq.defineDone();
            declare hash sch (dataset: 'work.ssoc', multidata: 'y');
            rc=sch.defineKey('soc_name');
            rc=sch.defineData('search_name');
            rc=sch.defineDone();
            declare hash pth (dataset: 'work.spt', multidata: 'y');
            rc=pth.defineKey('pt_name');
            rc=pth.defineData('search_name');
            rc=pth.defineDone();
            call missing(search_name, scope);
      end;
      set ae(keep=subject aeseq aeterm startd aestdtc_raw strttime aeser related
      aesev aerel aerelbn aeout aeendtc_raw stttime llt_code pt_name soc_name
  recordposition);
      rc=smq.find(key:llt_code);
      do while(rc=0);
            output;
            rc=smq.find_next(key:llt_code);
      end;
      rc=sch.find(key:soc_name);
      do while(rc=0);
            output;
            rc=sch.find_next(key:soc_name);
      end;
      rc=pth.find(key:pt_name);
      do while(rc=0);
            output;
            rc=pth.find_next(key:pt_name);
      end;
  run;
```

Section 1: defining the hash table

- Use a length statement to initialize variables that will be joined to the table:

  ```
  length search_name $100 llt_code $160  scope $8 soc_name $480 pt_name $480;
  ```

- Declare the hash with a name (smq), a source dataset (dataset), and indicate if it's possible for a single key to match multiple values (multidata):

  ```
  declare hash smq (dataset: 'work.smq', multidata: 'y');
  ```

- Use a dummy variable (rc) to try the assignment of the key (which should match with your dataset), the data to join, and end the define block:

  ```
  rc=smq.defineKey('llt_code');
  rc=smq.defineData('search_name', 'scope');
  rc=smq.defineDone();
  ```

- After declaring and creating the hash object, in order to avoid an uninitialized note from SAS, be sure to use the call missing routine to assign missing values to the variables.

Section 2: Using the Hash

Use a dummy variable to try to find the key in the hash table. If no match can be found, the variable rc will take on a nonzero value.

```
rc=smq.find(key:llt_code);
do while(rc=0);
       output;
       rc=smq.find_next(key:llt_code);
end;
```

This code selectively outputs only records that match a value from the hash table. Specifically, this outputs AEs whose llt_code is in the SMQ hash. A single AE may match more than one SMQ and will output multiple times with different lookup values. Structuring the data in this way will allow the use of by-group processing to rapidly repeat similar tabulations for different search strategies.

Alternate structure

Once a signal has been found with an established search strategy for an AESI, the AESI may need to be flagged at the ADAE dataset level. A similar use of hash tables can make this possible for simple searches with one pass through the data and no sorting, simply by creating separate hash tables for each search element of the SIs that need to be flagged, at the corresponding key variable levels.

Example code:

Suppose two AESIs have been identified, AESI_A identified by a list of PTs and AESI_B by a combination search strategy of PTs and LLTs:

```
data adae(drop=rc llt_code aellt aeseq recordposition);
    if _n_=1 then do;
           length llt_code $160 pt_name $480 FL_A $1 FL_B $1;
           declare hash a_pt (dataset: 'work.a_pt');
           rc=a_pt.defineKey('pt_name');
           rc=a_pt.defineDone();
           declare hash b_pt (dataset: 'work.b_pt');
           rc=b_pt.defineKey('pt_name');
           rc=b_pt.defineDone();
           declare hash b_llt (dataset: 'work.b_llt');
           rc=b_llt.defineKey('llt_code');
           rc=b_llt.defineDone();
           call missing(FL_A, FL_B);
    end;
    set adae;
    rc=a_pt.find(key:pt_name);
    if rc=0 then fl_a='Y';
    rc=b_pt.find(key:pt_name);
    if rc=0 then fl_b='Y';
    else do;
           rc = b_llt.find(key: llt_code);
           if rc=0 then fl_b='Y';
    end;
run;
```

This will result in a copy of ADAE that has every record with a PT matching a_pt flagged with fl_a='Y' and every record matching either PT in b_pt or LLT in b_LLT flagged with fl_b='Y'.

Efficiency gain:

When datasets are small, an n-fold efficiency gain is an interesting theoretical result. However, since AE search strategies are routinely applied across large safety databases or to pooled AE data from many different trials, a multiplicative gain in efficiency can mean the difference between getting same-day results or requiring a job to run overnight.

We ran a simple example on a relatively small dataset comparing a SMQ hash lookup on LLTs to a single unsorted SQL join:

Hash:

```
real time              0.03 seconds
cpu time               0.03 seconds
```

Unsorted SQL join:

```
real time              0.14 seconds
cpu time               0.14 seconds
```

CONCLUSION

For the purpose of adverse event analysis, the use of hash tables can provide much faster results in comparison to SQL joins. The efficiency gain has been found to be considerably large. The paper described two methods to identify AESIs using hash tables while discussing how AEs that match with multiple SMQs can be identified.

REFERENCES

- SAS® Certification Prep Guide: Advanced Programming for SAS®9
- http://www.pharmasug.org/proceedings/2013/BB/PharmaSUG-2013-BB01.pdf
- http://www.pharmasug.org/proceedings/2013/HO/PharmaSUG-2013-HO02.pdf

ACKNOWLEDGMENTS

We would like to acknowledge Rajeev Karanam for reviewing our paper and providing insightful comments.

CONTACT INFORMATION

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

Name: Vinodita Bongarala
Enterprise: Seattle Genetics, Inc.
Address: 21823 - 30th Drive S.E. Bothell, WA 98021
Work Phone: 425-527-4288
E-mail: vbongarala@seagen.com

Name: Liz Thomas
Enterprise: Seattle Genetics, Inc.
Address: 21823 - 30th Drive S.E. Bothell, WA 98021
Work Phone: 425-527-2370
E-mail: ethomas@seagen.com