Paper 156-2023

Table Lookup by Enclosing Hash in FCMP

Wenyu Hu, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

Table lookup or searching is a common task performed in SAS®. There are many lookup methods such as SET with KEY=, arrays, SQL joins, formats, MERGE with a BY statement, and DATA step hash object. Beginning with SAS® 9.3, hashing is available to user-defined subroutines through the SAS Function Complier (FCMP) procedure. FCMP enables programmers to create and store frequently used complex and repetitive code into reusable and customized functions and subroutines. Embedding a hash object in PROC FCMP functions can enhance a user's capability to handle large data by improving performance while retaining program simplicity. This paper introduces table lookup using hashing in PROC FCMP and demonstrate how it can improve performance and streamline existing programs.

INTRODUCTION

A hash object provides an efficient and convenient mechanism for quick data storage and retrieval. It stores and retrieves data based on lookup keys, and is considered the fastest way to search through a large amount information. While hashing's main advantage is to improve program performance with large datasets, its syntax is often considered complicated and hard to maintain.

FCMP (Function Compiler) enables one to write functions and CALL routines using DATA step syntax and makes these reusable blocks of code available to other SAS procedures or DATA step programs.

Beginning with SAS 9.3, hashing is available in user-defined subroutines through the FCMP procedure. By embedding smart hashing techniques in FCMP, a user can have most features of the hash objects wrapped within a single procedure. This could better streamline and simplify the code.

TRADITIONAL TABLE LOOKUP VS HASHING FCMP

There are many table lookup techniques. Examples are: "Join with PROC SQL", "MERGE with BY statement", "SET statement with KEY= option", "Table Lookup using Formats", "Lookup with Hash Object". We will look at table lookup examples comparing traditional approaches and the hashing FCMP approach.

Searching for a value in one dataset when given a value from another dataset is one of the most frequent data processing operations. This value is often referred to as a "key". For example, given a subject identification, one could retrieve the patient's demographics or other treatment related information from the dataset.

Suppose a lookup dataset SV contains subject identification (USUBJID), visit number (VISITNUM), visit date (SVSTDTC) and planned study day of visit (VISITDY). The master

dataset QS has subject identification (USUBJID), visit number (VISITNUM) and the questions the subject answered at each visit.

JOIN WITH PROC SQL

Using SQL, multiple datasets can be joined without having common variables across all datasets. Datasets do not have to be sorted or indexed.

```
proc sql;
      create table qs_sql
      as select qs.usubjid, qs.visitnum, sv.svstdtc as qsvisdtc, sv.visitdy as qsvisdy,
      qs.qstestcd, qs.qstest, qs.qsorres, qs.qsstresc, qs.qsstresn
      from sdtm.qs qs left join sdtm.sv sv
      on qs.usubjid=sv.usubjid and
      qs.visitnum=sv.visitnum;
quit;
```

SET STATEMENT WITH KEY = OPTION

Lookup operations can be performed using the KEY = option with one or more SET statements if the lookup dataset is indexed by unique key variables. No sorting or indexing is required on the master dataset. The composite key can include a combination of variables with either numeric or character type and multiple values can be returned.

```
proc sql;
      create index patvis
      on sdtm.sv (usubjid, visitnum);  ❶
quit;
data qs_key;
      length qsvisdtc $19;
      set sdtm.qs;
      set sdtm.sv(keep=usubjid visitnum svstdtc visitdy)  ❷
      key=patvis/unique;  ❸
      if _iorc_=0 then do; ❹
            qsvisdtc=svstdtc;
            qsvisdy=visitdy;
      end;
run;
```

❶ PROC SQL is used to create index PATVIS consisting of USUBJID and VISITNUM.

❷ The order of the datasets specified is important. The master dataset must be specified first, and then the lookup dataset.

❸ The UNIQUE option forces SAS to begin at the top of the lookup table each time it performs a lookup operation.

❹ If _IORC_ returns a value of 0, it indicates that SAS found a matching observation.

HASH OBJECT

When using a DATA step hash object, neither dataset is required to be sorted or indexed. The key may be composite and may simultaneously consist of both numeric and character values and multiple data items can be stored per key.

```
data qs_hash;
      length qsvisdtc $19;
      if _n_=1 then do;
                  if 0 then set sdtm.sv(keep=usubjid visitnum svstdtc visitdy); ❶
                  declare hash hh(dataset: "sdtm.sv");
                  hh.definekey('usubjid', 'visitnum'); ❷
                  hh.definedata('svstdtc', 'visitdy'); ❸
                  hh.definedone();
      end;
      do until (eof2);
                  set sdtm.qs end=eof2;
                  rc=hh.find(key:usubjid, key:visitnum); ❹
                  if rc=0 then do; ❺
                              qsvisdtc=svstdtc;
                              qsvisdy=visitdy;
                  end;
                  else do;
                              qsvisdtc=''; qsvisdy=.;
                  end;
                  output;
      end;
      stop;
run;
```

❶ By using a non-executing SET statement, attributes of hash variables originating from an existing dataset are initialized.

❷ Composite key is defined using DEFINEKEY.

❸ Data elements are defined using DEFINEDATA. Multiple variables can be defined to be either character or numeric.

❹ FIND is used to search for the composite key value of USUBJID and VISITNUM in the QS dataset and retrieve the values of the data variables from the hash object.

❺ If the search is successful, return code RC is set to 0. Multiple numeric/character results to be returned.

HASH FCMP

The following example uses FCMP to define a subroutine that returns the visit date (SVSTDTC) and planned study day of visit (VISITDY) from the SV dataset. Once defined, the FCMP subroutine can be utilized wherever a subroutine is normally used. The hash object can be used to perform a lookup by calling the subroutine LOOKUP_SV. Wrapping a hash in a user-defined subroutine provides a more generic way, and the program becomes easier to maintain.

```
proc fcmp outlib=work.functions.lookup_tbl; ❶
    subroutine lookup_sv(usubjid $, visitnum, svstdtc $, visitdy ); ❷
        outargs svstdtc, visitdy; ❸
        declare hash hh(dataset: "sdtm.sv");  ❹
        rc=hh.definekey('usubjid', 'visitnum');
        rc=hh.definedata('svstdtc', 'visitdy');
        rc=hh.definedone();
        rc=hh.find(); ❺
    endsub;  ❻
quit;
options cmplib=work.functions; ❼
```

❶ The OUTLIB= option tells PROC FCMP where to store the subroutines it compiles, Multiple subroutines and functions can be declared in a single usage of PROC FCMP.

❷ The routine is named as LOOKUP_SV and has two character arguments (USUBJID, SVSTDTC) and two numeric arguments (VISITNUM, VISITDY). Subroutine is used here since multiple values are returned. If only one return value is needed, Function can be used instead.

❸ OUTARGS is used to specify the arguments returned by the subroutine.

❹ The hash object HH is defined within subroutine. It is important to note that the data table SDTM.SV is not loaded when the subroutine is compiled by FCMP.

❺ The FIND method is used to retrieve values from the hash table. It will use composite key values of USUBJID and VISITNUM as index to look up and retrieve values from the hash object.

❻ ENDSUB closes the definition of LOOKUP_SV routine.

❼ Option CMPLIB = specifies where to look for previously compiled functions and subroutines. All procedures that support the use of FCMP functions and subroutines use this system option.

The hash object defined above can be used to perform a lookup by calling the subroutine LOOKUP_SV as shown in the example below.

```
data qs_hash_fcmp;
      length svstdtc qsvisdtc $19;

      do until (eof2);
            svstdtc=''; ❶
            visitdy=.;  ❶
            set sdtm.qs end=eof2;
            call lookup_sv(usubjid, visitnum, svstdtc, visitdy);  ❷
            qsvisdtc=svstdtc;
            qsvisdy=visitdy;
            output qs_hash_fcmp; ❸
      end;
run;
```

❶ The variables to be returned must be defined on the PDV before the routine is called. This is the requirement for any routine that returns values.

❷ LOOKUP_SV is called using the CALL statement. The returned values are added to the PDV.

❸ If LOOKUP_SV returns a match, then QSVISDTC and QSVISDY are updated with the values of the returned variables SVSTDTC and VISITDY respectively in the output dataset QS_HASH_FCMP, otherwise, QSVISDTC and QSVISDY are set to blank in the output dataset.

The above example shows how hashing is integrated within FCMP procedure to provide a user defined subroutine to perform table lookup. This can dramatically reduce the steps involved in conventional hash methods, with an added advantage of minimal use of system resources. Storing and compiling user defined functions/subroutines can be easily referenced to the respective library locations. This gives programmers more flexibility to create, store and compile user defined subroutines/functions. By using the HASH FCMP approach, we eliminate multiple sorting and merging steps through DATA step or using PROC SQL and make table lookup more elegant.

FCMP HASH SUPPORTED STATEMENTS AND METHODS

Please note that not all hash methods and statements available in the regular DATA step are supported by FCMP. The following list highlights the commonly used hash methods and statements that are available in FCMP.

Method	Syntax	Description
DECLARE	declare hash myhash(dataset: "table");	Create a new instance of hash object
DEFINEDATA	rc = myhash.DEFINEDATA('data 1','data 2');	Define data to be stored in hash object
DEFINEKEY	rc = myhash.DEFINEKEY('key 1','key 2') ;	Define key variables to hash object
DEFINEDONE	rc = myhash.DEFINEDONE() ;	Indicate all data definitions and keys are complete
FIND	rc = myhash.FIND('key1','key2');	Search hash object based on values of defined keys
ADD	rc = myhash.ADD(key:'key1', key: key2, data: 'data1', data: 'data2');	Add data associated with keys to hash object
REMOVE	rc = myhash.REMOVE(key:'key1', key: key2);	Remove data associated with keys from hash object
REPLACE	rc = myhash.REPLACE(key:'key1', key: key2, data: 'data1', data: 'data2');	Replace data associated with the specified key with new data
CHECK	rc = myhash.CHECK(key:'key1', key: key2);	Return only a value that indicates whether the key is in hash object
CLEAR	rc=myhash.CLEAR();	Remove all items from hash object without deleting the hash object instance

HASH FCMP USAGE

When a hash object is incorporated into a user defined subroutine/function through the FCMP procedure, it dramatically reduces the use of traditional DATA steps or SQL procedures, instead a simple subroutine/function is called. Multiple DATA steps or procedures can be integrated into the FCMP procedure, and the programmer can decide the desire location to store and compile subroutine/function. The steps involved in creating subroutine/function with a hash object are complex, but when it is stored and compiled in a library, it can be accessed easily and compiled accordingly when needed. It extends the functionality of user-defined subroutines/functions and streamlines existing programs to make them simpler and generic.

It is most efficient to use hash when the lookup dataset is relatively small compared with the master dataset and fits in the memory. Since any hash object is placed in the system memory, searching, or finding data that are lined to its key occurs faster compared with data read from disk.

CONCLUSION

This paper discusses various techniques for managing table lookup problems. It primarily explores the usage of user-defined subroutines/functions in conjunction with hash objects to handle large datasets. With the help of FCMP subroutines/functions, we can store hash tables in memory and retrieve them through subroutine/function calls. By extending the functionality of user-defined SAS functions, we can enhance the efficiency of our code and streamline our daily work.

REFERENCES:

1. PROC FCMP and DATA Step Component Objects https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lecompobjref/n0kyejvk2wd2qyn1gch1awbzsyiy.htm
2. Arthur L. Carpenter (2018) “Using Memory Resident Hash Tables to Manage Your Sparse Lookups” Proceedings of WUSS 2018: https://www.lexjansen.com/wuss/2018/41_Final_Paper_PDF.pdf
3. Premanand Kumaravel (2019) “An Antidote for Huge Clinical Trial Data Processing: Enclosing Hash in FCMP” Proceedings of PhUSE US Connect 2019 https://www.lexjansen.com/phuse-us/2019/ct/CT13.pdf
4. Wenyu Hu (2008) “Using Lookup Tables to Match Data” Proceedings of SESUG 2008: https://analytics.ncsu.edu/sesug/2008/CC-024.pdf

TRADEMARKS

CONTACT INFORMATION

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

Wenyu Hu
Merck & Co., Inc.,
351 North Sumneytown Pike,
P.O. Box 1000,
North Wales, PA 19454
Wenyu_hu@merck.com