PharmaSUG China 2017 - Paper 19

Streamline Table Lookup by Embedding HASH in FCMP

Qing Liu, Eli Lilly & Company, Shanghai, China

ABSTRACT

SAS provides many methods to perform a table lookup like Merge Statement, Proc SQL, Format. Starting with SAS 9.3, Even we can define our custom routine to perform table lookup since hashing is available in user-defined subroutines through the FCMP procedure. By embedding a hash object in PROC FCMP functions, simplicity in program can be enhanced. This paper introduces table lookup using HASH in PROC FCMP and demonstrate a practical application in using this technic to solve real world problems.

INTRODUCTION

Hashing is a search algorithm to index a table, thus to store and retrieve associated data by using index hash key. It has been introduced into SAS® since SAS 9. Being considered the fastest way to search large amount information, Hash's main usage is to improve program performance when working with large dataset. While hashing is often used through DATA step hash object in SAS Program, its syntax is often considered complicated and hard to maintain.

The FCMP procedure enables you to create user-defined SAS functions using DATA step syntax that is stored in a data set, this function can be called from the DATA step just as you would any other SAS function. Comparing with modern programming language, such as Python, R, SAS function has very limitation in general programming activity.

However, starting with SAS 9.3, hashing is available in user-defined subroutines through the FCMP procedure. By embedding HASH in FCMP, we can extend the functionality of user-defined SAS functions and join the power of HASH and FCMP. This could streamline your existing programs, make your program more generic, and enhance simplicity. This paper is not intent to teach fundamental HASH or FCMP knowledge, but to introduce a new way to implement HASH and FCMP

TABLE LOOKUP – TRADITIONAL APPROACH VS HASHING FCMP

There are many ways to look up table: merge, SQL join, Format, Data hashing, and etc. Below is an example comparing using traditional approach and hashing FCMP approach in table lookup.

	state	capital
1	North Carolina	Raleigh
2	Virginia	Richmond
3	South Carolina	Columbia
4	Arkansas	Little Rock
5	Mississippi	Jackson
6	Alabama	Montgomery
7	Louisiana	Baton Rouge
8	Tennessee	Nashville
9	Georgia	Atlanta
10	Florida	Tallahassee
11	Kentucky	Frankfort
12	West Virginia	Charleston

	state
1	Virginia
2	North Carolina
3	Texas
4	Alabama
5	Mississippi
6	Georgia
7	Arkansas

Figure 1. Sates and Capital Sample Data

Traditional Approach

1. Data Step

```
/* Data Step approach*/

proc sort data = StatesCapital;  by state;run;
proc sort data = States;         by state;run;

data new;
   merge StatesCapital States;
   by state;
run;
```

2. PROC SQL

```
/* Proc SQL */
proc sql;
   create table new as
   select a.state, b.capital
   from States as a, StatesCapital as b
   where a.state = b.state;
quit;
```

3. Data Hash

```
/* Data Hash */
data new;
   if 0 then set work.StatesCapital;
   if _n_ = 1 then do;
         declare hash hh (dataset: "work.StatesCapital");
        hh.definedata("capital");
        hh.definekey("state");
        hh.definedone();
   end;
   set States;
   rc =hh.find();
   if rc = 0 then capital = capital;
run;
```

Hash FCMP

While many SAS users use Data Step or SQL to perform table lookup in their daily work. Advanced programmer may use hash in their program for efficiency purpose. However, using hash means more syntax, thus the program is much complex and hard to maintain. In this case, wrapping hash in SAS user-defined functions provide a more generic and simple way. And program could be much simplified.

```
/* Define the hash function */
proc fcmp outlib=work.functions.samples;
   function hash_fcmp(state $) $25;
   declare hash hh(dataset: "work.StatesCapital");
   rc=hh.definedata("capital");
   rc=hh.definekey("state");
   rc=hh.definedone();
   rc=hh.find();
   if rc eq 0 then return(capital);
   else return('Not Found');
   endsub;
quit;
options cmplib=work.functions;

/* Call the function */
* In Data Step;
data new;
  set states;
  capital = hash_fcmp(state);
run;

* In Proc SQL;
proc sql;
   create table new as
   select *,hash_fcmp(state) as capital
   from states;
quit;
```

As we can see in above example, embedding HASH in FCMP made table lookup more elegant. By packaging HASH in FCMP, the complicated HASH syntax goes no way.

FCMP HASH OBJECT METHOD AND SYNTAX

Method	Syntax	Description
DECLARE	DECLARE hash object-name	Create a new instance of hash object, create at parse time.
DEFINEKEY	rc = object.DEFINEKEY('key 1','key n')	Set up key variables for hash object
DEFINEDATA	rc = object.DEFINEDATA('dataset 1','dataset n')	Define data to be stored in hash object
DEFINEDONE	rc = object.DEFINEDONE()	Indicate the key and data specification is completed
DELETE	rc = object.DELETE()	Delete the hash object and free any resources allocated
FIND	rc = object.FIND('key1','keyn')	Search a hash object based on the values of defined key, If look up is successful, defined data variable are updated
CHECK	rc = object.CHECK()	Search a hash object based on the values of defined key, data will not be updated whether If look up is successful
NUM_ITEMS	rc = object.NUM_ITEMS()	Return the number of items in hash object
ADD	rc = object.ADD(key: value1, key: value n)	Add data with associated key to hash object
REMOVE	rc = object.REMOVE(key: value1, key: value n)	Remove data with associated key to hash object

Table 2. HASH in FCMP Method and Syntax

PRACTICAL APPLICATION

In a Proportional font, such as the font this article is set in, different letters occupy different amount of horizontal space. For instance, the letter "I" is much narrower than the letter "W". Most books, magazines and other printed materials are set in proportional fonts. similarly, the graphical user interface of many programs uses a proportional font for titles, menus and other text. Examples of commonly-used proportional fonts are Times New Roman, Verdana, Arial, Georgia and Comic Sans. For my experience, there are no exist method in SAS to get a string widths directly. The method used in the following example demonstrate a way to calculate the display width of a proportional font string by joining the power of HASH and PROC FCMP.

WHY SHOULD WE GET PRINT WIDTH OF A STRING

The idea to calculate the display width of a string comes from two of my projects.

The one is using SAS to draw annotation in rectangular box in CRF which is in PDF format in the development of a aCRF automation tool, and we need to decide how long the rectangular box should be. The width of rectangular box varied from different string. Unfortunately, the string needs in proportional fonts and we can't get the rectangular box width just by counting characters in string.

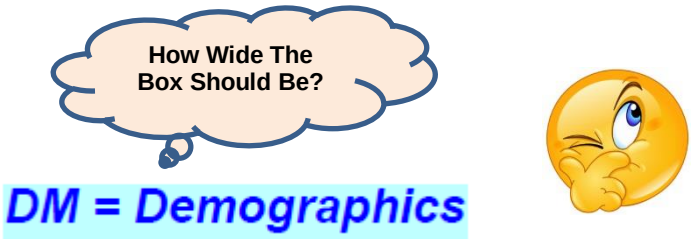


Figure 2. Determine the width of rectangular box

Another is to avoid unwanted wrap up in RTF output since SAS let Microsoft Word to determine the print width. We need to get exact column width for each column so that text would not wrap up. This is especially important in some report which contains many columns and we cannot simply imagine the column width.

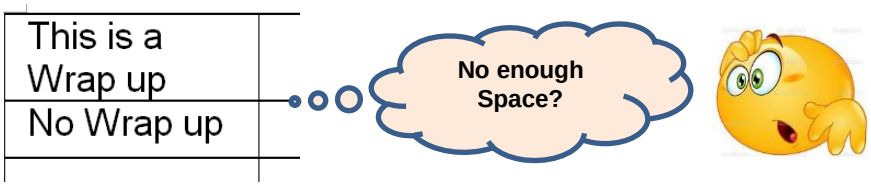
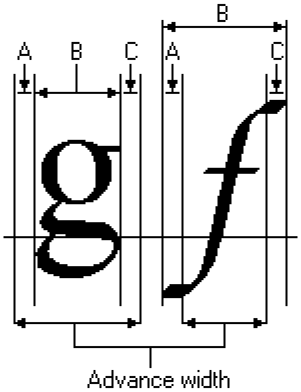


Figure 3. Determine the width of RTF output

THE COMPONENTS OF CHARACTER WIDTH



A: the width to add to the current position before placing the character
B: the width of the character itself
C: the white space to the right of the character
The total width is determined by calculating the sum of A + B + C

Figure 4. Characters Width

SAS APPROACH TO GET WIDTH

The process to get a print character width is listed below.

1. Determine the formula: $String\ Width = Sum(Unique\ Character\ Width)$
2. Got unique character width and stored in table
3. Retrieve using HASH and sum up using FCMP

Get unique Character width

Completed tables regarding print character width is not available in the website, thus I made a method approximating character width by using SAS. And it worked well. The method is listed below.

1. Output characters in RTF format in specific style, and make sure one character fill with a line(Code could be found in the appendix)

`` ```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

2. Read RTF file back line by line, approximate character width using the following formula
$$Width = \frac{\text{Length of the Line}}{\text{Count of Chracter}}$$
3. Summarize The Hash Table by repeating above steps for different font style. Store the data in a dictionary table

char	_font_	_style_	_font_size_	_width_
'	Times	Nomal	10	2.268
|	Times	Nomal	10	2.268
	Times	Nomal	10	2.5060773481
,	Times	Nomal	10	2.5060773481
.	Times	Nomal	10	2.5060773481
/	Times	Nomal	10	2.8
:	Times	Nomal	10	2.8
;	Times	Nomal	10	2.8
\	Times	Nomal	10	2.8
i	Times	Nomal	10	2.8

Figure 4. The character width dictionary

MAKE THE HASH FUNCTION

```
* Customed SAS Function;
proc fcmp outlib=width.functions.GetWidth;
    function GetWidth(string $ ,_font_ $,_style_ $,_font_size_ $);
    length id $ 1;

    * Define Hash Table with Character Width Dataset;
    declare hash Calculate(dataset: "width.CharWidth");
    rc = Calculate.defineKey("char","_font_","_style_","_font_size_");
    rc = Calculate.defineData("_width_");
    rc = Calculate.defineDone();

    width_tot = 0;

    * Retrieve Data from Hash and Sum up;
    do i = 1 to lengthn(string);
        char = substr(string,i,1);
        rc   = Calculate.find();
        if rc eq 0 then width_tot = width_tot+_width_;
    end;
    return(width_tot);
    endsub;
run;
```

SIMPLE USAGE

```
*------------------- Hash in PROC FCMP Example Usage -------------------;
* Using in Data Step;
data Width;
    set TEXT;
    CharWidth   = GetWidth(Value, 'Times','Normal','10');
run;

* Using in Proc SQL;
proc sql;
      create table Width as
      select GetWidth(Value, 'Times','Normal','10') as CharWidth
      from TEXT;
quit;

* Using in Proc MEANS;
proc means data = TEXT;
      where GetWidth(Value, 'Times','Normal','10') > 99;
      var Value;
run;
```

SIMPLE OUTPUT

text	font	style	font_size	TextWidth
MDX	Times	Normal	10	23.704258065
RSX Type S 2dr	Times	Normal	10	66.876593001
TSX 4dr	Times	Normal	10	34.967230228
TL 4dr	Times	Normal	10	28.180830925
3.5 RL 4dr	Times	Normal	10	43.913405145
3.5 RL w/Navigation 4dr	Times	Normal	10	101.3058582
NSX coupe 2dr manual S	Times	Normal	10	102.88078652

CONCLUSION

Table lookup by embedding Hash in PROC FCMP is not for efficiency, it's to streamline your daily work, enhance program simplicity and extend the functionality of user-defined SAS functions. And repetitive coding can be minimized.

REFERENCES

- Andrew Henrick, Donald Erdman, and Stacey Christian (2013). "Hashing in PROC FCMP to Enhance Your Productivity", Proceeding of the SAS Global Forum 2013 Conference. http://www.lexjansen.com/wuss/2013/141_Paper.pdf

- Sample 47224: Load a SAS data set into a Hash Object using PROC FCMP http://support.sas.com/kb/47/224.html

- PROC FCMP and DATA Step Component Objects http://documentation.sas.com/?docsetId=proc&docsetVersion=9.4&docsetTarget=n03uc8c8fkguxqn1i5iapv1auqrz.htm&locale=en

- Kirk Paul Lafler (2016). "A Hands-on Introduction to SAS® DATA Step Hash Programming Techniques", Procceding of the MidWest SAS Users Group(MWSUG) Conference. http://www.lexjansen.com/mwsug/2016/HW/MWSUG-2016-HW02.pdf

CONTACT INFORMATION

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

Name: Qing Liu
Enterprise: Eli Lilly & Company
Address: 19F Tower 1 HKRI, Taikoo Hui, No. 288 Shi Men Yi Road
City, State ZIP: Shanghai, 200041
Work Phone: 13167017220
E-mail: MeetQingLiu@gmail.com

Source Code Sample

```
* Dummy all characters in ASCII;
data null;
    length text $200;
    do i =160,32 to 127;
        text = repeat(byte(i),300);
        output;
    end;
run;

* Define the output font style;
proc template;
    define style global.text;
        parent=styles.rtf;
        replace fonts /
                'fixedfont'           = ("Times",10pt,Bold Normal)
                'docfont'             = ("Times",10pt,Bold Normal)
                ;
    end;
run;

* Out RTF file and read this file back;
options orientation=portrait topmargin=1in bottommargin=1in leftmargin=1in
rightmargin=1in nodate;
title '';
ods rtf file='test.rtf' style=global.text;
proc report data=null NOHEADER nowd;
    column text;
    define text/ display;
run;
ods rtf close;

data width_1;
    infile "test.txt";
    input;
    length text $200;
    retain blank;
    if _n_ = 1 then blank=count(_infile_,'|');
    if ^missing(_infile_);
    text = strip(_infile_);
    if _n_ ne 1;
run;

proc sql;
    create table width_2 as
    select distinct *, lengthn(text) as length
    from width 1
    group by substr(text,1,1)
    having lengthn(text)=max(lengthn(text));
quit;

data width_3;
    set width 2;
    width tot = 6.3;
    if _n_ =1 then do;
```

```
        char = ' ';
        width = (width_tot/blank)*72;
        output;
    end;
    char  = substr(text,1,1);
    width = (width_tot/length)*72;
    output;
    keep char width;
run;
proc sort;by width;run;

data width;
    set width_3;
    length _font_ _style_  _font_size_ $20;
    _font_ = 'Times';
    _style_ = 'Bold Normal';
    _font_size_ = '10';
    _width_ = width;
    keep char _font_ _style_  _font_size_ _width_;
run;

data charwidth;
    set width;
run;

* Customed SAS Function;
proc fcmp outlib=work.functions.GetWidth;
    function GetWidth(string $ ,_font_ $,_style_ $,_font_size_ $);
    length id $ 1;
    * Retrieve Character Width Using HASH;
    declare hash Calculate(dataset: " work.CharWidth");
    rc = Calculate.defineKey("char","_font_","_style_","_font_size_");
    rc = Calculate.defineData("_width_");
    rc = Calculate.defineDone();

    width_tot = 0;

    * Sum all width up for a string;
    do i = 1 to lengthn(string);
        char = substr(string,i,1);
        rc   = Calculate.find();
        if rc eq 0 then width_tot = width_tot+_width_;
    end;
    return(width_tot);
    endsub;
run;

options cmplib=work.functions;

data a;
    set sashelp.cars;
    b=GetWidth(strip(model),'Times','Normal','10');
run;
```