SAS® GLOBAL FORUM 2017

April 2 – 5 | Orlando, FL

#SASGF

USERS PROGRAM

Presenters

Paul Dorfman, *Independent Consultant*
Don Henderson, *Henderson Consulting Services, LLC*

Don Henderson and Paul Dorfman have been using SAS® for decades, working and consulting in such industries as software development, finance, pharma, insurance, engineering, and retail. They have been involved, also for many decades, with SUGI/SAS Global Forum and regional conferences as chairs, presenters, and instructors.

Beyond Table Lookup:

The Versatile SAS® Hash Object

Cause and Aim

Usage	Cause	The Hash Object has been in active use for 15+ years. Yet there are still misconceptions about its programming power, flexibility, and scope: 1. Useful for processing "big data" only. 2. Just a table look-up subterfuge, good for matching files.
	Aim	Show that the Hash Object is a versatile, dynamic programming structure.
Learning	Cause	Usual starting points for SAS Hash learners: 1. Code *cannibalism* - centered on specific *tasks* and scales poorly. 2. SAS documentation - centered on *tools* rather than *operations.*
	Aim	Show the hash object power via the prism of *table operations.*
SAS Press Book	Cause	The authors are writing a book on the subject with above points in mind. The paper is a brief compendium on what to expect in the book.
	Aim	Attract attention to the future result of our geeky efforts.

Sample Data and Programs

Papers' sample data and code snippets come from a set of programs available online:

http://www.sascommunity.org/wiki/Beyond_Table_Look-up:_The_Versatile_SAS_Hash_Object

The data is designed to illustrate broad management / reporting functionality.

Techniques	Scope	E.g.
Table Look-Up	Most common problems addressed using the Hash Object	Subsetting, file matching, de-duplication, etc.
Data Load / Management	Processing transactional data	Create DW fact / dimension tables from transactions
Data Aggregation - Basic	Single-level or hierarchical	From structured DW or transaction data
Data Aggregation - Advanced	Multiple-level or non-hierarchical	Enterprise-wide multi-level reporting
Memory Management	Minimizing hash memory footprint	Scenarios of hash memory taken to physical limits

Sample Data – the Bizarro Angle

Description	Generated for a *fictitious game* called *Bizarro Ball* (aka BB)	
	Conceptually similar to *Baseball* - with a few wrinkles	
	Sufficiently rich set of data	
Table Content	Fact	Pitches, Atbats, Runs
	Dimension	Teams, Players, Games
Utility	Illustrate *well-known* and commonly used SAS Hash Object features	
	Highlight *less well-known* and/or underused capabilities	

Bizarro Sample Data - Details

Dimension Tables:

- PITCH_DISTRIBUTION: List of the possible results from each pitch (e.g., Called Strike, Single, . . .)
- PLAYERS - 1,600 Players (2 leagues * 16 teams * 50 players)
- POSITIONS_DIM - 6 different positions (e.g., Infielder, Outfielder, . . .)
- SCHEDULE - 992 games over 31 days (16 morning games and 16 afternoon games for each of the 31 days)
- TEAMS - 32 teams (2 leagues, 16 teams each)

Bizarro Sample Data Details

Transactional Tables:

- BATTERS - 27,776 batter x game combinations (each team has 14 batters in each of the 62 games)
- PITCHERS - 7,856 pitcher x game combinations (each team has 9 pitchers, who each pitch one inning, in each of the 62 games)

Bizarro Sample Data Details

Fact Tables:

- ATBATS - 98,769 plate appearances across all the teams/games with fields for the results of the player at bat. The data includes details on what bases are occupied (including the player identifier) at the completion of each plate appearance/atbat.
- PITCHES - 301,506 pitches for those 98,769 plate appearances.
- RUNS - 23,901 runs scored.

The number of rows generated for each table can be varied based on *random seeds* to determine the results of each pitch.

Autoexec.sas program assigns the seed values used by the programs. Different seed values will produce different result sets.

The Hash Object in a Nutshell

Hash Object =	Hash table in memory for rapid data storage and retrieval
	Hash object tools to perform table operations
Environment	DATA step
	DS2 Procedure
Tool Activate (stand-alone or in combinations)	Global table operations: Create, Delete, Truncate, etc.
	Item-level table operations, Search, Insert, Update, etc.
	Data exchange: Hash ↔PDV, PDV ↔Disk storage

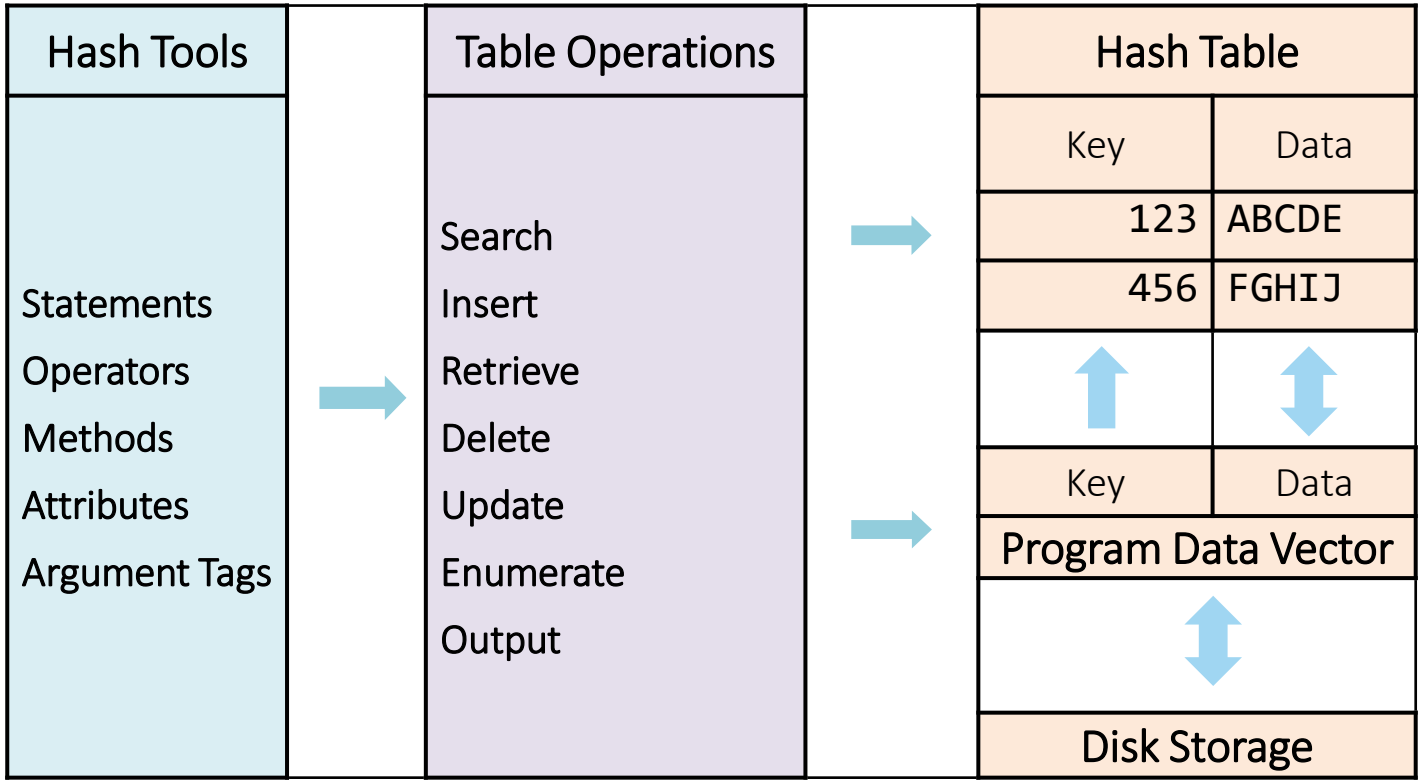
Hash Table - Visual Example

	Hash *Variables*				
Hash *Entry*	***Key* Portion**		***Data* Portion**		
	Numeric	Numeric	$ 12	$ 12	$ 2
	Team_SK	**Player_ID**	**First_name**	**Last_name**	**Position_code**
Hash *Items*	171	12059	Jason	Anderson	RP
	193	14175	Benjamin	Lopez	C
	353	15036	Tyler	Gonzales	IF
	153	16188	Dylan	Jackson	OF
	158	15913	Harold	Clark	OF
	130	17510	Jesse	Parker	UT

Hash Table Nomenclature / Properties

Element	Nomenclature / Property
Data Residence	Memory (RAM, main storage)
Key + Data Portion	Hash *Entry*
Column	Hash *Variable*
Row	Hash *Item*
Key Portion	Required, *simple or composite*
Data Portion	Required, *one or more hash variables*
Key Variable Type	Numeric, character only (i.e. *scalar*)
Data Variable Type	Numeric, character, or of *type object*
Corresponding PDV Variables	*PDV Host* Variables - Required, same names as hash
PDV Host Variables Update	From *Data portion only*
Memory (RAM) Allocation	*Grows / shrinks* with Number of items, run-time

Tools and Operations



Standard Table Operations

Table Level	Create	Declare hash or iterator object, define its entries, and *create an instant* of it.
	Delete	Delete hash or iterator object. Deleting hash also deletes its iterator.
	Truncate	Remove all hash items *en masse without deleting the hash table* itself.
	Output	Write each item's *data portion variables* to a SAS data set as observations.
Item Level	Search	Check if key is in table. *Do not overwrite* PDV host variables.
	Insert	If key not is in table, *add new item* with this key.
	Delete	If key is in table, *remove all or some items* with this key.
	Retrieve	If key is in table, *overwrite host PDV variables* with their *data portion* variables.
	Update	If key is in table, *overwrite data portion variables* with their PDV host variables.
	Enumerate	*List one or more items serially* from table start, end, or given key. For each item listed, overwrite host PDV variables with data portion variables.
	*Key*Numerate	If key is in table, *list and retrieve one or some items* with this key.

Examples: Overview

- The examples in this *paper* are designed to illustrate the *Task-Operation-Tools* concept.
- Refer to the *paper* to see *examples for each table operation* listed above.
- The *examples selected for the presentation* are driven by 3 vectors:
  - ✓Highlight the concept of thinking *in terms of operations first*, given a task.
  - ✓Underscore the *dynamic and versatile nature of the hash object* as programming structure.
  - ✓ Spur the *desire to learn more* about the hash object by showing how it does things you may not have thought possible.

Table Lookup as Explicit *Retrieve* Operation

```
data withNames(drop=Player_ID);
 dcl hash dim(dataset:'bizarro.players');
 dim.DEFINEKEY('Player_ID');
 dim.DEFINEDATA('First_name'
                ,'Last_name'
                ,'Position_code')
 ;
 dim.DEFINEDONE();
  do until(lr);
   set bizarro.atbats end=lr;
   if dim.find(key:Batter_ID) = 0
     then output;
  end;
  stop;
  /* set up PDV host variables */
  set bizarro.players(drop=team_sk);
run;
```

	Batter_ID	Inning	onFirst
1	16660	1	
2	15650	1	
3	14302	1	
4	15643	2	
5	15652	2	156
6	15673	2	156
7	14334	2	156

Batter	Inning	onFirst	First_Name	Last_Name	Position_Code
16660	1	.	Justin	Green	OF
15650	1	.	Jerry	Jackson	IF
14302	1	.	Brandon	Garcia	C
15643	2	.	Larry	Allen	IF
15652	2	15652	Charles	Watson	IF
15673	2	15652	Gerald	Nelson	IF
14334	2	15652	Jack	Sanders	C

Basic Data Aggregation Scheme

```
data _null_;
   dcl hash H  (ordered:"a");   *Create H;
   H.defineKey ("Batter_ID");
   H.defineData("Batter_ID","Games","RBIs");
   H.defineDone();
   dcl hash U  ();              *Create U;          Count
   U.defineKey ("Batter_ID","Game_SK");            Distinct
   U.defineDone();
   do until (lr);
      set bizarro.atbats end = lr;
      rc = H.FIND();            *Retrieve sums from H;
      Games + (U.ADD() = 0);    *Sum in PDV / Insert into H;
      RBIs  + Runs;             *Sum in PDV;
      H.REPLACE();              *Update H with new sums;
   end ;
   H.OUTPUT(dataset:"Stats");   *Output aggregate table H;
   stop;
run;
```

Data-Driven Aggregation

- Create data aggregates (e.g, BA = Batting Average) at *multiple key levels*.
- Do not want to *duplicate code* for each level of the required aggregation.
- Want to make it easy to *add new key aggregation levels*.
- In Baseball Terminology these aggregates are called "*splits*".

```
/* Aggregation Key Levels Driver Table */

data agg_levels;
 input (Table Key KeyAsData) (:$32.);
 datalines;
 Overall _N_        0
 Team    Team_SK    1
 Batter  Batter_ID  1
run;
```

Augment Atbats Data (via a Join)

```
data _null_ ;
   dcl hash players(dataset:'bizarro.players') ;
   players.defineKey('Player_ID') ;
   players.defineData('Team_SK','Last_Name','First_Name') ;
   players.defineDone() ;
```

For each Aggregation Level – Its Own Hash Instance via Hash of Hashes (HoH)

```
dcl hash HoH() ;
HoH.defineKey ('Table') ;
HoH.defineData('H','Table') ;
HoH.defineDone() ;
/* Need to scan through separate instances of H */
dcl hiter i_HoH ('HoH') ;
/* No parens: Instances of H will be created later */
dcl hash H ;
```

- Variable H will contain an instance of H for each key level.
- Variable H data type is *not scalar* (not numeric or character) .
- Variable H of *type Hash Object*.
- This type can be stored *only in the data portion* of table HoH.

Each Key Level – Separate Instance - Separate Set of Keys
Create the Instances - Add Them to HoH

```
do until(lr) ;
   /* define keys for each H instance using driver AGG_LEVELS */
   do q = 1 by 1 until (last.Table) ;
      set agg_levels end = lr ;
      by Table notsorted ;
      if q = 1 then h = _new_ hash (ordered:'a') ; /* new instance    */
      h.defineKey(Key) ;                           /* add a key       */
      if KeyAsData = 1 then h.defineData(Key) ;    /* add data as key */
   end  ;
   /* now add the needed data fields */.
   h.defineData('ABs') ;
   h.defineData('Hits') ;
   h.defineData('BA') ;
   h.defineDone() ;     /* finish creating next instance */
   hoh.add() ;          /* add the instance to HoH       */
end ;
```

Aggregate at All Key Levels - Single Pass

```
format BA 4.3;
/* Loop to read AtBats data once */
do lr = 0 by 0 until (lr) ;
   set bizarro.atbats end = lr ;
   /* Get players data - explicit Retrieve operation */
   _iorc_ = players.find(key:batter_id) ;
   /* Enumerate instances of H - each has its own aggregation key set */
   do until (i_hoh.next()) ;
      /* Aggregate for H instance just Retrieved from HoH by iterator */
      if h.find() ne 0 then call missing (ABs, Hits, BA) ;
      Abs  = sum (Abs, Is_An_AB) ;
      Hits = sum (Hits, Is_A_Hit) ;
      BA   = Divide (Hits, ABs);
      /* Update hash data for current H instance */
      h.replace() ;
   end ;
end ;
```

Create Output Summary Results

```
   /* Each H instance is an aggregate table for its own key level */
   /* Make each H instance current using hash iterator I_HOH      */
   /* Output each instance to its own data set                    */
   /* Data set name is created dynamically using variable Table   */
   /* Its name corresponds to key level from driver AGG_LEVELS    */

   do while (i_hoh.next() = 0) ;
       h.output (dataset: Table) ;
   end ;

   stop ;
   /* Create needed PDV host variables at compile time */
   set bizarro.atbats ;
run ;
```

Multi-Level Aggregation Results

Running the program results in the following SAS log notes:

```
NOTE: There were 1600 observations read from the data set
BIZARRO.PLAYERS.
NOTE: The data set WORK.OVERALL has 1 observations and 3 variables.
NOTE: The data set WORK.TEAM has 32 observations and 4 variables.
NOTE: The data set WORK.BATTER has 924 observations and 4 variables.
NOTE: There were 3 observations read from the data set WORK.TABLES.
NOTE: There were 98769 observations read from the data set
BIZARRO.ATBATS.
```

Multi-Level Aggregation Post-Discussion

- The aggregates for the 3 key levels dictated by driver AGG_LEVELS are computed in a *single pass.*
- There is no need for a separate pass through the data for every key level.
- Nor there is a need to create 3 separate tables for each aggregate level with hard-coded names.
- No other method to avoid hard-coding and make code dynamic (such as a macro or DATA step / %INCLUDE) is needed, either.
- In this case, a similar result can be attained by using the MEANS procedure with the corresponding TYPES.
- *However*, the program is made simple intentionally, just to illustrate the concept.
- It can be enriched in the mold of the Basic Aggregation example to *simultaneously compute* non-additive *"count distinct"* aggregates.
- This functionality is not available from the MEANS procedure.
- It is available from SQL only at the expense of *multiple passes* through the input.

Conclusion

- This is the end of the presentation.
- The authors thank everyone who have attended, read the paper or will read the paper.
- They are eager to answer any presentation or paper-related questions anyone might have in the remaining time or after the session.
- Please do not hesitate to contact the authors using the coordinates indicated in the paper.

THE SAS HASH OBJECT ROCKS!

Don't Forget to Provide Feedback!

1. Go to the Agenda icon in the conference app.
2. Find this session title and select it.
3. On the sessions page, scroll down to Surveys and select the name of the survey.
4. Complete the survey and click Finish.