Exploring HASH Tables vs. SORT/DATA Step vs. PROC SQL

Richann Watson, Experis
Lynn Mullins, PPD

Introduction to HASH Tables

What are HASH Tables

- Sometimes referred to as hash objects
- Only accessed with a DATA step
- Provides an efficient way to search the data

- Two parts to a hash table
  - Key: variable(s) used to perform the lookup
  - Data: data value(s) associated with the key

Some Methods and Syntax of HASH tables

Methods

Method	Description	Syntax
Check	Checks to see if key is stored in hash table	***hashobj.***check(); ***hashobj.***check(key: ***keyvar1,*** …, key: ***keyvarN***);
Definedata	Defines the data that is to be stored in hash table	***hashobj.***definedata(); ***hashobj.***definedata(***datavar1***, …, ***datavarN***);
Definedone	Indicates that the key part and data part of the hash table are complete	***hashobj***.definedone();
Definekey	Defines the variables that will be used as the key in the hash table	***hashobj***.definekey(***keyvar1,*** …, ***keyvarN***); ***hashobj***.definekey(all: 'yes');
Find	Determines if key is stored in hash table	***hashobj.***find(); ***hashobj.***find(key: ***keyvar1,*** …, key: ***keyvarN***);
Output	Creates a data set which will contain data from hash table	***hashobj.***output(dataset: '***lib.outdsn***');

General Syntax

```
data _null_;
  define attributes for variables that will be retrieved, i.e., data part
  if _n_ = 1 then do;
       /* declare name for hash table with ascending sort order */
        declare hash hashobj(dataset: "lib.indsn", ordered: "a");
       /* define variables that will be used a key for lookup (key part) */
        hashobj.definekey ('keyvar1', 'keyvar2', 'keyvar3');
       /* define variables that will be retrieved (data part) */
        hashobj.definedata ('datavar1', 'datavar2');
       /* end definition of hash table */
        hashobj.definedone();
  end;
  /* specify the main table(s) that are going to use the lookup table */
  set inlibnm.indsn;
  /* one or more hash methods can be used to add, find, replace, etc. */
  hashobj.check();
  if hashobj.find() then output;
run;
```

Illustration of Hash Table

Why Use Hash Tables?

Example

```
data _null_;

  /* specify the lookup table */
  if _n_ = 1 then do;
    /* define the attributes for the variables that are added to main data set */
    if 0 then set indsn.femresp1 (drop=CASEID);

    /* declare name for hash table with ascending sort order */
          declare hash fresp(dataset: "indsn.femresp1", ordered: "a");

    /* define variables that will be used a key for lookup (key part) */
          fresp.definekey ('CASEID');

    /* define variables that will be retrieved (data part) */
          fresp.definedata (all: 'yes');

    /* end definition of hash table */
          fresp.definedone();
  end;

  /* specify the main tables that are going to use the lookup table */
  set indsn.femresp end=eof;

  /* if there is a match fresp.find() returns a 0 for success */
  /* otherwise it returns non-zero value for failure          */
  /* at the end of the file output the hash table to data set */
  if eof and fresp.find() = 0 then fresp.output(dataset: 'femresp_hash');
run;
```

Why Use Hash Tables?

	Standard DATA Step	PROC SQL	DATA Step HASH
Syntax Complexity	Straightforward	Straightforward to Moderate	Very Confusing
Memory or Disk-Based	Disk	Disk	Memory
Ideal size of data sets	Any Can be a resource hog for very large data sets and may not be very efficient.	Small to Moderate Can be a resource hog for very large data sets.	Large to Very Large
Memory Allocation	Upfront	Upfront	Only when needed
Sorting/Indexing Required	Yes	No	No
Additional calculations	Yes	Maybe	Yes

Ex 1 Benchmarks – Small

	Real Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1st Sort	0.10	N/A	N/A
2nd Sort	0.14	N/A	N/A
Join	4.27	7.18	4.31
Total	4.51	7.18	**4.31**

SMALL	# of Obs	# of Vars	Size (KB)
Data set 1	100	5,754	5,311
Data set 2	100	5,754	5,311
Final	100	11,507	4,096*

* Compressed using binary option

Ex 1 Benchmarks – Moderate

	Real Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1st Sort	1.72	N/A	N/A
2nd Sort	1.66	N/A	N/A
Join	8.10	12.27	7.27
Total	11.48	12.27	**7.27**

MODERATE	# of Obs	# of Vars	Size (KB)
Data set 1	10,847	5,754	488,926
Data set 2	10,847	5,754	488,926
Final	10,847	11,507	216,576*

* Compressed using binary option

Ex 1 Benchmarks – Large

	Real Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1st Sort	9.31	N/A	N/A
2nd Sort	9.4	N/A	N/A
Join	20.58	301.31	38.09
Total	39.29	301.31	**38.09**

LARGE	# of Obs	# of Vars	Size (KB)
Data set 1	45,103	5,754	2,053,824
Data set 2	45,103	5,754	2,053,824
Final	45,103	11,507	890,368*

* Compressed using binary option

Ex 2 Benchmarks – Small

	Real Time (seconds)		
Step	**Standard DATA Step**	**PROC SQL**	**DATA Step HASH**
1st Sort *	0.05	N/A	N/A
2nd Sort †	1.00	N/A	N/A
3rd Sort ‡	0.43	N/A	N/A
1st Pre-join †	0.24	0.55	N/A
2nd Pre-join (step 1) ‡	0.04	N/A	N/A
2nd Pre-join (step 2) ‡	0.02	0.04	N/A
Join	0.36	0.36	1.10
Total	2.14	**0.95**	1.10

SMALL	**# of Obs**	**# of Vars**	**Size (KB)**
*** Lookup**	**86**	**5**	**128**
† Lab Result	**40,210**	**19**	**98,784**
‡ Cancelled	**786**	**19**	**2,156**
Final	**40,996**	**21**	**20,480***

*** Compressed using binary option**

Ex 2 Benchmarks – Moderate

	Real Time (seconds)		
Step	**Standard DATA Step**	**PROC SQL**	**DATA Step HASH**
1st Sort *	0.15	N/A	N/A
2nd Sort †	14.79	N/A	N/A
3rd Sort ‡	0.42	N/A	N/A
1st Pre-join †	2.09	5.63	N/A
2nd Pre-join (step 1) ‡	0.38	N/A	N/A
2nd Pre-join (step 2) ‡	0.02	0.06	N/A
Join	1.96	2.07	4.12
Total	19.81	7.76	**4.12**

MODERATE	**# of Obs**	**# of Vars**	**Size (KB)**
*** Lookup**	**86**	**5**	**128**
† Lab Result	**337,453**	**21**	**844,000**
‡ Cancelled	**786**	**19**	**2,156**
Final	**338,239**	**23**	**92,160***

*** Compressed using binary option**

Ex 2 Benchmarks – Large

	Real Time (seconds)		
Step	**Standard DATA Step**	**PROC SQL**	**DATA Step HASH**
1st Sort *	0.08	N/A	N/A
2nd Sort †	1,257.10	N/A	N/A
3rd Sort ‡	0.13	N/A	N/A
1st Pre-join †	82.49	3,004.86	N/A
2nd Pre-join (step 1) ‡	1.91	N/A	N/A
2nd Pre-join (step 2) ‡	0.04	3.10	N/A
Join	81.74	104.37	275.36
Total	1,423.49	3,112.33	**275.36**

LARGE	**# of Obs**	**# of Vars**	**Size (KB)**
*** Lookup**	**86**	**5**	**128**
† Lab Result	**10,620,791**	**21**	**26,552,200**
‡ Cancelled	**786**	**19**	**2,156**
Final	**10,621,577**	**23**	**2,580,480***

*** Compressed using binary option**

Ideal Usage

When is one method better than the other?

Recommendations based on number of records and variables

Scenario	Standard DATA Step	PROC SQL	DATA Step HASH
Small size data sets with many variables	✓	✓	✓
Moderate size data sets with many variables	✓	✓	✓
Large size data sets with many variables	✓	✗	✓
Small size data sets with few variables	✓	✓	✓
Moderate size data sets with few variables	✗	✓	✓
Extremely Large size data sets with few variables	✗	✗	✓

The use of the indicated method is not recommended.

Use caution with the indicated method(s) in this scenario

Ideal method(s) for the indicated scenario

Limitations

When is one method better than the other?

What are the limitations

- Operating environments
- SAS version
- SAS COMPRESS = BINARY option
- SQL optimizer

Conclusions

What are the conclusions

- This was a small test
- Compared basic DATA step, PROC SQL, and HASH table
- Various size data sets and number of variables were compared
- Several factors to consider for method

Richann Watson
Experis
richann.watson@experis.com

Lynn Mullins
PPD
lynn.mullins@ppdi.com