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

Richann Watson, Experis, Batavia, Ohio
Lynn Mullins, PPD, Cincinnati, Ohio

ABSTRACT

There are often times when programmers need to merge multiple SAS® data sets to combine data into one single source data set. Like many other processes, there are various techniques to accomplish this using SAS software. The most efficient method to use based on varying assumptions will be explored in this paper. We will describe the differences, advantages and disadvantages, and display benchmarks of using HASH tables, the SORT and DATA step procedures, and the SQL procedure.

INTRODUCTION

Merging data sets together is a common practice that programmers do in order to combine data sets based on key fields. There are a number of possible solutions to merge SAS data sets, including the PROC SORT/DATA step merge, the PROC SQL join, and HASH table lookups. Some of the determinants on which method to use are the size of the data sets, resource availability, and the programmers' experience with the different techniques. This paper will discuss these three methods in detail beginning with the syntax for using HASH table lookups including the options associated with this method. We will then describe the differences in the complexity, memory type, data set size, and other attributes between the three methods. Benchmarks will also be discussed using three data set sizes, small, medium, and large. And lastly, the ideal situations will be described for when to use each method.

INTRODUCTION TO HASH TABLES

Hash tables, also referred to as hash objects, is an in-memory lookup table that can only be accessed from within the DATA step that creates it. Thus, once the DATA step ends, the hash table is deleted. A hash table provides an efficient way to search the data.

The hash object has two parts to it. The first part is the key. The key can consist of a single variable or multiple variables that will be used to perform a lookup. The key part can consist of character and/or numeric values. The second part of a hash object is the data part. The data part is the data value(s) associated with the key. The data part can also consist of character and/or numeric values.

SYNTAX AND SOME METHODS OF HASH TABLES

The hash table is defined in a DATA step and is only available during the DATA step. The syntax of a hash object can be difficult and can take some time getting used to. Once the hash table is defined, then it can be used to add, find, replace, check, remove, and output data. Below is generic code that shows how a hash table is defined.

```
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();
```

```
  /* 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, check, etc. */
  hashobj.check();
  if hashobj.find() then output;

run;
```

Some of the methods that can be used with hash tables along with a description of each and the syntax are in the table below.

Method	Description	Syntax
Add	Adds the data associated with the key to the hash table	***hashobj.***add(); ***hashobj.***add(key: ***keyvar1,*** …, key: ***keyvarN***, data: ***datavar1***, …, data: ***datavarN***);
Check	Checks to see if key is stored in hash table	***hashobj.***check(); ***hashobj.***check(key: ***keyvar1,*** …, key: ***keyvarN***);
Clear	Removes all entries in hash table without deleting the hash table	***hashobj.***clear();
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');
Equals	Determines if two hash tables are equal and stores result in indicated DATA step variable	***hashobj.***equals(hash: '***hashobj1***', results: ***resvar***)
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***');
Ref	Performs a **find** on the current key and if the key is not found it is added to the hash table	***hashobj.***ref(); ***hashobj.***ref(key: ***keyvar1,*** …, key: ***keyvarN***);
Remove	Removes the data associated with the key	***hashobj.***remove(); ***hashobj.***remove(key: ***keyvar1,*** …, key: ***keyvarN***);
Replace	Replaces the data associated with the key with new data	***hashobj.***replace(); ***hashobj.***replace(key: ***keyvar1,*** …, key: ***keyvarN***, data: ***datavar1***, …, data: ***datavarN***);
Sum	Gets the key summary for the indicated key and stores in the indicated DATA step variable	***hashobj.***sum(***sumvar***); ***hashobj.***sum(key: ***keyvar1,*** …, key: ***keyvarN***, sum: ***sumvar***);

Table 1. Hash table methods

ILLUSTRATION OF HASH TABLE

Below is example code of creating a hash table.

```
/* 1. not applicable - sorting is not required when using HASH OBJECTS */
/* 2. combine the result records into one data set by CASEID */

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);

    /*******************************************/
    /*** BEGIN SECTION TO DECLARE HASH OBJECT ***/
    /*******************************************/
    /* 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 SECTION TO DECLARE HASH OBJECT ***/
    /*****************************************/
  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;
```

DIFFERENCES BETWEEN THE THREE METHODS

The differences between the three merging methods (DATA step merge, SQL Procedure, and HASH table) that we benchmarked are highlighted in the following table.

	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

Table 2. Differences between the three methods

BENCHMARKS OF THE THREE METHODS

The summary of real-time, the amount of time spent to process the SAS job, are shown in the tables below. Three different data set sizes (# of observations) have been used for comparison. Real-time is also referred to as elapsed time. The lowest real-time used is displayed in red.

The following three tables display the real-time results using two data sets with many formatted variables.

	Real Time (seconds)		
Step	**Standard DATA Step**	**PROC SQL**	**DATA Step HASH**
1.1 (1st Sort)	0.10	N/A	N/A
1.2 (2nd Sort)	0.14	N/A	N/A
2 (Join)	4.27	7.18	4.31
Total	4.51	7.18	**4.31**

	# 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**

Table 3. Real-Time statistics of small size data sets with many variables

	Real Time (seconds)		
Step	**Standard DATA Step**	**PROC SQL**	**DATA Step HASH**
1.1 (1st Sort)	1.72	N/A	N/A
1.2 (2nd Sort)	1.66	N/A	N/A
2 (Join)	8.10	12.27	7.27
Total	11.48	12.27	**7.72**

	# 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**

Table 4. Real-time statistics of moderate size data sets with many variables

	Real Time (seconds)		
Step	**Standard DATA Step**	**PROC SQL**	**DATA Step HASH**
1.1 (1st Sort)	9.31	N/A	N/A
1.2 (2nd Sort)	9.40	N/A	N/A
2 (Join)	20.58	301.31	38.09
Total	39.29	301.31	**38.09**

	# 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**

Table 5. Real-Time statistics of large size data sets with many variables

The following three tables display the real-time results using three data sets with few variables.

	Real Time (seconds)		
Step	**Standard DATA Step**	**PROC SQL**	**DATA Step HASH**
1.1 (1st Sort*)	0.05	N/A	N/A
1.2 (2nd Sort†)	1.00	N/A	N/A
1.3 (3rd Sort‡)	0.43	N/A	N/A
2.1 (1st Pre-join†)	0.24	0.55	N/A
2.2.1 (2nd Pre-join‡)	0.04	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.02	0.04	N/A
3 (Join)	0.36	0.36	1.10
Total	2.14	**0.95**	1.10

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

*** Compressed using binary option**

Table 6. Real-Time statistics of small size data sets with few variables

	Real Time (seconds)		
Step	**Standard DATA Step**	**PROC SQL**	**DATA Step HASH**
1.1 (1st Sort*)	0.15	N/A	N/A
1.2 (2nd Sort†)	14.79	N/A	N/A
1.3 (3rd Sort‡)	0.42	N/A	N/A
2.1 (1st Pre-join†)	2.09	5.63	N/A
2.2.1 (2nd Pre-join‡)	0.38	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.02	0.06	N/A
3 (Join)	1.96	2.07	4.12
Total	19.81	7.76	**4.12**

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

*** Compressed using binary option**

Table 7. Real-Time statistics of moderate to large size data sets with few variables

	Real Time (seconds)		
Step	**Standard DATA Step**	**PROC SQL**	**DATA Step HASH**
1.1 (1st Sort*)	0.08	N/A	N/A
1.2 (2nd Sort†)	1,257.10	N/A	N/A
1.3 (3rd Sort‡)	0.13	N/A	N/A
2.1 (1st Pre-join†)	82.49	3,004.86	N/A
2.2.1 (2nd Pre-join‡)	1.91	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.04	3.10	N/A
3 (Join)	81.74	104.37	275.36
Total	1,423.49	**3,112.33**	**275.36**

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

*** Compressed using binary option**

Table 8. Real-Time statistics of extremely large size data sets with few variables

WHEN IT IS IDEAL TO USE ONE METHOD OVER THE OTHER

The data sets sizes have an impact on the efficiencies of the different methods. If the data sets to be joined are relatively small, then using a standard sort and DATA step merge would be sufficient. The larger the data sets are the less the DATA Step and PROC SQL methods become.

The number of variables should also be considered when more than just the number of records needs to be considered. Below are recommended approaches of how to determine which method is best to use considering the data set size and number of variables. Again, we see that the small size data sets independent of the number of variables can be merged using a standard sort and DATA step merge but the SQL procedure works just as well. Interestingly, the DATA Step with the HASH table method is more efficient with larger data sets having fewer 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			

Table 9. Recommendation based on number of records and number of variables

LIMITATIONS

These benchmarks were run on a Windows 7 environment using PC SAS v9.4. Different results may occur running on other environments and SAS versions. The SAS option COMPRESS = BINARY was used to make the programs run quicker.

SQL uses different algorithms to execute different types of joins. The SQL optimizer may choose to execute an inner join using a hash, index, or sort-and-merge technique under different circumstances. In our test, we used a left outer join on the small data sets to add two variables from the lookup table and only kept the lookup table data that matched. We did an inner join on the large data sets because there was a 1-1 match of the key variable in each data set.

Results from a DATA step merge can vary based on the environment (i.e. Compression = ON or OFF and results from PROC SORT can vary based on any options used (i.e. TAGSORT vs. Non-TAGSORT vs. OUT = used).

CONCLUSION

This was just a small test and there are other factors that can be considered when doing benchmarking but for our purposes we only looked at doing a basic DATA step, PROC SQL with a left outer join, and hash object. We looked at various sizes of data sets and number of variables to see which process was the most efficient and then did a comparison of the data sets to make sure that all three processes produced the same results. There are several factors to consider when deciding which approach to use. It may sometimes be worthwhile to learn a new method, even if it is a bit cumbersome, if in the end it will save you a lot of processing time.

REFERENCES

SAS Institute, SAS9 Hash Object Tip Sheet, Available at **http://support.sas.com/rnd/base/datastep/dot/hash-tip-sheet.pdf**.

Burlew, Michele M. 2012. SAS Hash Object Programming Made Easy. Cary, NC: SAS Institute Inc.

Lafler, Kirk P. 2010-2015. Exploring SAS DATA Step Hash Programming Techniques. Software Intelligence Corporation.

Secosky, Jason and Bloom, Janice 2007. “Getting Started with the DATA Step Hash Object”. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

Thanks to Jamie Mabry, Lindsay Dean, Ken Borowiak, David Gray, Richard D’Amato, Lynn Clipstone, and PPD and Experis Management for their reviews and comments. Thanks to our families for their support.

DISCLAIMERS

The contents of this paper are the work of the authors and do not necessarily represent the opinions, recommendations, or practices of PPD or Experis.

CONTACT INFORMATION

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

Richann Watson
Experis
(513) 843-4081
Richann.watson@experis.com

Lynn Mullins
PPD
(910) 558-4343
Lynn.mullins@ppdi.com

APPENDIX A – ADDITIONAL HASH TABLE EXAMPLE

```
/* 1. Add in the LBCAT and LBTESTCD from the lookup table */
data labhash;
  /* specify the lookup table(s) */
  if _n_ = 1 then do;
    /* this will define the attributes for the variables */
    /* to be retrieved in the hash object                */
    /* (i.e. defines attributes for LBCAT and LBTESTCD   */
    if 0 then set indsn.lookup (keep=LBCAT LBTESTCD);

       /*****************************************************************/
       /*** BEGIN SECTION TO DECLARE HASH OBJECT FOR LABS WITH RESULTS ***/
       /*****************************************************************/
       /* declare name for hash table with ascending sort order */
       declare hash lrslt(dataset: "indsn.lookup", ordered: "a");

       /* define variables that will be used a key for lookup (key part) */
       lrslt.definekey ('PANEL', 'TEST', 'UNIT');

       /* define variables that will be retrieved (data part) */
       lrslt.definedata ('LBCAT', 'LBTESTCD');

       /* end definition of hash table */
       lrslt.definedone();
       /*****************************************************************/
       /*** END SECTION TO DECLARE HASH OBJECT FOR LABS WITH RESULTS ***/
       /*****************************************************************/
       /*****************************************************************/
       /*** BEGIN SECTION TO DECLARE HASH OBJECT FOR CANCELLED LABS ***/
       /*****************************************************************/
       /* declare name for hash table with ascending sort order */
       /* want to keep only tests that don't have 'LE' in name  */
       /* this is due heme diffs not being calculated b/c there */
       /* no results to determine the diffs                     */
       declare hash lcncl(dataset: "indsn.lookup (where=(not(LBTESTCD ? 'LE')))",
                          ordered: "a");

       /* define variables that will be used a key for lookup */
       lcncl.definekey ('PANEL', 'TEST');

       /* define variables that will be retrieved */
       lcncl.definedata ('LBCAT', 'LBTESTCD');

       /* end definition of hash table                          */
       lcncl.definedone();
       /*****************************************************************/
       /*** END SECTION TO DECLARE HASH OBJECT FOR CANCELLED LABS ***/
       /*****************************************************************/
  end;

  /* specify the main tables that are going to use the lookup table */
  set indsn.labrslt (in=rslt)
      indsn.labcncl (in=cncl);
  /* if there is a match lxxxx.find() returns a 0 for success */
  /* otherwise it returns non-zero value for failure          */
  if rslt and lrslt.find() = 0 then output;
  if cncl and lcncl.find() = 0 then output;
run;
```

APPENDIX B – BENCHMARK METHODS

The statistics compared during the benchmarking and a description of each are highlighted in the table below.

Statistic	Description
Real-Time	The amount of time spent to process the SAS job. Real-time is also referred to as elapsed time.
User CPU Time	The CPU time spent to execute SAS code.
System CPU Time	the CPU time spent to perform operating system tasks (system overhead tasks) that support the execution of SAS code
Memory	The amount of memory required to run a step.
OS Memory	The maximum amount of memory that a step requested from the System.

Table 11. Benchmarks of the three methods

APPENDIX C – FULL BENCHMARK RESULTS

The full benchmarks results using data sets with many formatted variables are displayed in the tables below.

	Real Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort)	0.10	N/A	N/A
1.2 (2nd Sort)	0.14	N/A	N/A
2 (Join)	4.27	7.18	4.31
Total	4.51	7.18	**4.31**

	User CPU Time (seconds)		
Standard DATA Step	PROC SQL	DATA Step HASH	
0.01	N/A	N/A	
0.01	N/A	N/A	
1.73	6.98	1.68	
1.75	6.98	**1.68**	

	System CPU Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort)	0.01	N/A	N/A
1.2 (2nd Sort)	0.06	N/A	N/A
2 (Join)	2.43	0.10	2.55
Total	2.50	**0.10**	2.55

	Memory (k)		
Standard DATA Step	PROC SQL	DATA Step HASH	
7,681.96	N/A	N/A	
7,161.06	N/A	N/A	
35,039.62	60,757.40	33,228.14	
49,882.64	60,757.40	**33,228.14**	

	OS Memory (k)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort)	37,824	N/A	N/A
1.2 (2nd Sort)	37,824	N/A	N/A
2 (Join)	59,448	92,952	58,124
Total	135,096	92,952	**58,124**

	# of Obs	# of Vars
Data set 1	**100**	**5,754**
Data set 2	**100**	**5,754**
Final	**100**	**11,507**

Table 12. Benchmark statistics of small size data sets with many variables

	Real Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort)	1.72	N/A	N/A
1.2 (2nd Sort)	1.66	N/A	N/A
2 (Join)	8.10	12.27	7.72
Total	11.48	12.27	**7.72**

	User CPU Time (seconds)	
Standard DATA Step	PROC SQL	DATA Step HASH
1.31	N/A	N/A
1.27	N/A	N/A
4.80	11.21	4.32
7.38	11.21	**4.32**

	System CPU Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort)	0.40	N/A	N/A
1.2 (2nd Sort)	0.34	N/A	N/A
2 (Join)	3.15	1.43	3.29
Total	3.89	**1.43**	3.29

	Memory (k)	
Standard DATA Step	PROC SQL	DATA Step HASH
7,653.50	N/A	N/A
7,155.93	N/A	N/A
34,783.79	1,510,280.12	580,830.31
49,593.22	1,410,280.12	580,830.31

	OS Memory (k)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort)	34,504	N/A	N/A
1.2 (2nd Sort)	34,504	N/A	N/A
2 (Join)	53,824	92,952	58,124
Total	**122,832**	92,952	58,124

	# of Obs	# of Vars
Data set 1	**10,847**	**5,754**
Data set 2	**10,847**	**5,754**
Final	**10,847**	**11,507**

Table 13. Benchmark statistics of moderate size data sets with many variables

	Real Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort)	9.31	N/A	N/A
1.2 (2nd Sort)	9.40	N/A	N/A
2 (Join)	20.58	301.31	38.09
Total	39.29	30131	**38.09**

	User CPU Time (seconds)	
Standard DATA Step	PROC SQL	DATA Step HASH
8.62	N/A	N/A
8.73	N/A	N/A
15.83	44.03	19.61
33.18	44.03	**19.61**

	System CPU Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort)	0.62	N/A	N/A
1.2 (2nd Sort)	0.59	N/A	N/A
2 (Join)	4.07	31.2	4.64
Total	5.28	31.2	**4.64**

	Memory (k)	
Standard DATA Step	PROC SQL	DATA Step HASH
7,981.21	N/A	N/A
8,005.25	N/A	N/A
34,730.93	1,111,128.57	2,058,676.87
49,593.22	1,410,280.12	580,830.31

	OS Memory (k)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort)	34,516	N/A	N/A
1.2 (2nd Sort)	34,516	N/A	N/A
2 (Join)	53,780	1,137,572	2,077,800
Total	**122,812**	1,137,572	2,077,800

	# of Obs	# of Vars
Data set 1	**45,103**	**5,754**
Data set 2	**45,103**	**5,754**
Final	**45,103**	**11,507**

Table 14. Benchmark statistics of large size data sets with many variables

The full benchmarks results using data sets with few variables are displayed in the tables below.

	Real Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	0.05	N/A	N/A
1.2 (2nd Sort†)	1.00	N/A	N/A
1.3 (3rd Sort‡)	0.43	N/A	N/A
2.1 (1st Pre-join†)	0.24	0.55	N/A
2.2.1 (2nd Pre-join‡)	0.04	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.02	0.04	N/A
3 (Join)	0.36	0.36	1.10
Total	2.14	**0.95**	1.10

User CPU Time (seconds)		
Standard DATA Step	PROC SQL	DATA Step HASH
0.01	N/A	N/A
0.23	N/A	N/A
0.03	N/A	N/A
0.25	0.39	N/A
0.00	N/A	N/A
0.00	0.01	N/A
0.24	0.26	0.31
0.76	0.66	**0.31**

	System CPU Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	0.01	N/A	N/A
1.2 (2nd Sort†)	0.14	N/A	N/A
1.3 (3rd Sort‡)	0.01	N/A	N/A
2.1 (1st Pre-join†)	0.00	0.31	N/A
2.2.1 (2nd Pre-join‡)	0.00	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.01	0.01	N/A
3 (Join)	0.01	0.03	0.10
Total	0.18	0.35	**0.10**

Memory (k)		
Standard DATA Step	PROC SQL	DATA Step HASH
1,153.03	N/A	N/A
109,511.75	N/A	N/A
4,850.21	N/A	N/A
1,859.43	118,646.15	N/A
1,321.50	N/A	N/A
1,880.75	15,496.85	N/A
2,186.62	2,185.84	17,482.34
122,763.29	136,328.84	**17,482.34**

	OS Memory (k)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	35,564	N/A	N/A
1.2 (2nd Sort†)	143,656	N/A	N/A
1.3 (3rd Sort‡)	39,568	N/A	N/A
2.1 (1st Pre-join†)	35,308	151,740	N/A
2.2.1 (2nd Pre-join‡)	35,564	N/A	N/A
2.2.2 (2nd Pre-join‡)	35,308	48,512	N/A
3 (Join)	35,568	35,568	51,468
Total	360,536	235,820	**51,468**

	# of Obs.	# of Vars
*Test Code Lookup	86	5
†Lab Results	40,210	19
‡Cancelled Lab Test	786	19
Final	40,996	21

Table 15. Benchmark statistics of small size data sets with few variables

	Real Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	0.15	N/A	N/A
1.2 (2nd Sort†)	14.79	N/A	N/A
1.3 (3rd Sort‡)	0.42	N/A	N/A
2.1 (1st Pre-join†)	2.09	5.63	N/A
2.2.1 (2nd Pre-join‡)	0.38	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.02	0.06	N/A
3 (Join)	1.96	2.07	4.12
Total	19.81	7.76	**4.12**

	User CPU Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	0.01	N/A	N/A
1.2 (2nd Sort†)	2.68	N/A	N/A
1.3 (3rd Sort‡)	0.00	N/A	N/A
2.1 (1st Pre-join†)	1.80	3.82	N/A
2.2.1 (2nd Pre-join‡)	0.00	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.00	0.01	N/A
3 (Join)	1.71	1.73	1.93
Total	6.20	5.56	**1.93**

	System CPU Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	0.00	N/A	N/A
1.2 (2nd Sort†)	1.49	N/A	N/A
1.3 (3rd Sort‡)	0.04	N/A	N/A
2.1 (1st Pre-join†)	0.09	2.09	N/A
2.2.1 (2nd Pre-join‡)	0.01	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.01	0.03	N/A
3 (Join)	0.03	0.14	0.32
Total	1.67	2.26	**0.32**

	Memory (k)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	1,153.03	N/A	N/A
1.2 (2nd Sort†)	920,448.01	N/A	N/A
1.3 (3rd Sort‡)	4,850.18	N/A	N/A
2.1 (1st Pre-join†)	1,865.40	915,295.29	N/A
2.2.1 (2nd Pre-join‡)	1,321.31	N/A	N/A
2.2.2 (2nd Pre-join‡)	1,881.18	15,494.03	N/A
3 (Join)	2,196.40	2,194.93	22,629.90
Total	933,715.51	932,984.25	**22,629.90**

	OS Memory (k)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	35,564	N/A	N/A
1.2 (2nd Sort†)	955,048	N/A	N/A
1.3 (3rd Sort‡)	39,568	N/A	N/A
2.1 (1st Pre-join†)	35,308	948,256	N/A
2.2.1 (2nd Pre-join‡)	35,564	N/A	N/A
2.2.2 (2nd Pre-join‡)	35,308	48,512	N/A
3 (Join)	35,568	35,568	56,600
Total	1,171,928	1,032,336	**56,600**

	# of Obs.	# of Vars
*Test Code Lookup	86	5
†Lab Results	337,453	21
‡Cancelled Lab Test	786	19
Final	338,239	23

Table 16. Benchmark statistics of moderate size data sets with few variables

	Real Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	0.08	N/A	N/A
1.2 (2nd Sort†)	1,257.10	N/A	N/A
1.3 (3rd Sort‡)	0.13	N/A	N/A
2.1 (1st Pre-join†)	82.49	3,004.86	N/A
2.2.1 (2nd Pre-join‡)	1.91	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.04	3.10	N/A
3 (Join)	81.74	104.37	275.36
Total	1423.49	3112.33	**275.36**

User CPU Time (seconds)		
Standard DATA Step	PROC SQL	DATA Step HASH
0.00	N/A	N/A
120.60	N/A	N/A
0.00	N/A	N/A
59.20	219.54	N/A
0.00	N/A	N/A
0.00	0.04	N/A
55.66	65.00	70.20
235.46	284.58	**70.20**

	System CPU Time (seconds)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	0.01	N/A	N/A
1.2 (2nd Sort†)	59.99	N/A	N/A
1.3 (3rd Sort‡)	0.01	N/A	N/A
2.1 (1st Pre-join†)	2.85	153.20	N/A
2.2.1 (2nd Pre-join‡)	0.00	N/A	N/A
2.2.2 (2nd Pre-join‡)	0.03	0.10	N/A
3 (Join)	3.04	3.35	14.15
Total	65.93	156.65	**14.15**

Memory (k)		
Standard DATA Step	PROC SQL	DATA Step HASH
1,672.59	N/A	N/A
1,054,363.43	N/A	N/A
4,849.71	N/A	N/A
1,976.84	1,058,804.53	N/A
1,321.75	N/A	N/A
1,880.59	15,489.28	N/A
2,308.15	2,308.06	22,629.78
1,068,373.06	1,076,601.87	**22,629.78**

	OS Memory (k)		
Step	Standard DATA Step	PROC SQL	DATA Step HASH
1.1 (1st Sort*)	35,308	N/A	N/A
1.2 (2nd Sort†)	1,088,704	N/A	N/A
1.3 (3rd Sort‡)	39,312	N/A	N/A
2.1 (1st Pre-join†)	35,308	1,092,548	N/A
2.2.1 (2nd Pre-join‡)	35,564	N/A	N/A
2.2.2 (2nd Pre-join‡)	35,308	48,768	N/A
3 (Join)	35,824	35,824	56,856
Total	1,305,328	1,177,140	**56,856**

	# of Obs.	# of Vars
*Test Code Lookup	86	5
†Lab Results	10,620,791	21
‡Cancelled Lab Test	786	19
Final	10,621,577	23

Table 17. Benchmark statistics of large size data sets with few variables