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

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

<table>
<thead>
<tr>
<th></th>
<th>Standard DATA Step</th>
<th>PROC SQL</th>
<th>DATA Step HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax Complexity</td>
<td>Straightforward</td>
<td>Straightforward to</td>
<td>Very Confusing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Moderate</td>
<td></td>
</tr>
<tr>
<td>Memory or Disk-Based</td>
<td>Disk</td>
<td>Disk</td>
<td>Memory</td>
</tr>
<tr>
<td>Ideal size of data sets</td>
<td>Any Can be a</td>
<td>Small to Moderate</td>
<td>Large to Very Large</td>
</tr>
<tr>
<td></td>
<td>resource hog for</td>
<td>Can be a resource</td>
<td></td>
</tr>
<tr>
<td></td>
<td>very large data</td>
<td>hog for very large</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sets and may not</td>
<td>data sets.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>be very efficient.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Allocation</td>
<td>Upfront</td>
<td>Upfront</td>
<td>Only when needed</td>
</tr>
<tr>
<td>Sorting/Indexing Required</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Additional calculations</td>
<td>Yes</td>
<td>Maybe</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Ex 1 Benchmarks – Small

<table>
<thead>
<tr>
<th>Step</th>
<th>Standard DATA Step</th>
<th>PROC SQL</th>
<th>DATA HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Sort</td>
<td>0.10</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2nd Sort</td>
<td>0.14</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Join</td>
<td>4.27</td>
<td>7.18</td>
<td>4.31</td>
</tr>
<tr>
<td>Total</td>
<td>4.51</td>
<td>7.18</td>
<td><strong>4.31</strong></td>
</tr>
</tbody>
</table>

### SMALL

<table>
<thead>
<tr>
<th># of Obs</th>
<th>Vars</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set 1</td>
<td>100</td>
<td>5,754</td>
</tr>
<tr>
<td>Data set 2</td>
<td>100</td>
<td>5,754</td>
</tr>
<tr>
<td>Final</td>
<td>100</td>
<td>11,507</td>
</tr>
</tbody>
</table>

* Compressed using binary option
### Ex 1 Benchmarks – Moderate

<table>
<thead>
<tr>
<th>Step</th>
<th>Real Time (seconds)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard DATA Step</td>
<td>PROC SQL</td>
</tr>
<tr>
<td>1st Sort</td>
<td>1.72</td>
<td>N/A</td>
</tr>
<tr>
<td>2nd Sort</td>
<td>1.66</td>
<td>N/A</td>
</tr>
<tr>
<td>Join</td>
<td>8.10</td>
<td>12.27</td>
</tr>
<tr>
<td>Total</td>
<td>11.48</td>
<td>12.27</td>
</tr>
</tbody>
</table>

**MODERATE**

<table>
<thead>
<tr>
<th>Data set 1</th>
<th>10,847</th>
<th>5,754</th>
<th>488,926</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set 2</td>
<td>10,847</td>
<td>5,754</td>
<td>488,926</td>
</tr>
<tr>
<td>Final</td>
<td>10,847</td>
<td>11,507</td>
<td>216,576*</td>
</tr>
</tbody>
</table>

* Compressed using binary option
## Ex 1 Benchmarks – Large

<table>
<thead>
<tr>
<th>Step</th>
<th>Standard DATA Step</th>
<th>PROC SQL</th>
<th>DATA HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Sort</td>
<td>9.31</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2nd Sort</td>
<td>9.4</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Join</td>
<td>20.58</td>
<td>301.31</td>
<td>38.09</td>
</tr>
<tr>
<td>Total</td>
<td>39.29</td>
<td>301.31</td>
<td><strong>38.09</strong></td>
</tr>
</tbody>
</table>

### Large

<table>
<thead>
<tr>
<th></th>
<th># of Obs</th>
<th>Vars</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set 1</td>
<td>45,103</td>
<td>5,754</td>
<td>2,053,824</td>
</tr>
<tr>
<td>Data set 2</td>
<td>45,103</td>
<td>5,754</td>
<td>2,053,824</td>
</tr>
<tr>
<td>Final</td>
<td>45,103</td>
<td>11,507</td>
<td>890,368*</td>
</tr>
</tbody>
</table>

* Compressed using binary option
# Ex 2 Benchmarks – Small

<table>
<thead>
<tr>
<th>Step</th>
<th>Real Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard DATA Step</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; Sort *</td>
<td>0.05</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Sort †</td>
<td>1.00</td>
</tr>
<tr>
<td>3&lt;sup&gt;rd&lt;/sup&gt; Sort ‡</td>
<td>0.43</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; Pre-join †</td>
<td>0.24</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Pre-join (step 1) ‡</td>
<td>0.04</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Pre-join (step 2) ‡</td>
<td>0.02</td>
</tr>
<tr>
<td>Join</td>
<td>0.36</td>
</tr>
<tr>
<td>Total</td>
<td>2.14</td>
</tr>
</tbody>
</table>

### SMALL

<table>
<thead>
<tr>
<th>SMALL</th>
<th># of Obs</th>
<th># of Vars</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Lookup</td>
<td>86</td>
<td>5</td>
<td>128</td>
</tr>
<tr>
<td>† Lab Result</td>
<td>40,210</td>
<td>19</td>
<td>98,784</td>
</tr>
<tr>
<td>‡ Cancelled</td>
<td>786</td>
<td>19</td>
<td>2,156</td>
</tr>
<tr>
<td>Final</td>
<td>40,996</td>
<td>21</td>
<td>20,480*</td>
</tr>
</tbody>
</table>

* Compressed using binary option
## Ex 2 Benchmarks – Moderate

<table>
<thead>
<tr>
<th>Step</th>
<th>Standard DATA Step</th>
<th>PROC SQL</th>
<th>DATA Step HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; Sort *</td>
<td>0.15</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Sort †</td>
<td>14.79</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3&lt;sup&gt;rd&lt;/sup&gt; Sort ‡</td>
<td>0.42</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; Pre-join †</td>
<td>2.09</td>
<td>5.63</td>
<td>N/A</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Pre-join (step 1) ‡</td>
<td>0.38</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Pre-join (step 2) ‡</td>
<td>0.02</td>
<td>0.06</td>
<td>N/A</td>
</tr>
<tr>
<td>Join</td>
<td>1.96</td>
<td>2.07</td>
<td>4.12</td>
</tr>
<tr>
<td>Total</td>
<td>19.81</td>
<td>7.76</td>
<td>4.12</td>
</tr>
</tbody>
</table>

---

**MODERATE**

<table>
<thead>
<tr>
<th># of Obs</th>
<th># of Vars</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Lookup *</td>
<td>86</td>
<td>5</td>
</tr>
<tr>
<td>† Lab Result</td>
<td>337,453</td>
<td>21</td>
</tr>
<tr>
<td>‡ Cancelled ‡</td>
<td>786</td>
<td>19</td>
</tr>
<tr>
<td>Final</td>
<td>338,239</td>
<td>23</td>
</tr>
</tbody>
</table>

* Compressed using binary option
## Ex 2 Benchmarks – Large

<table>
<thead>
<tr>
<th>Step</th>
<th>Standard DATA Step</th>
<th>PROC SQL</th>
<th>DATA Step HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; Sort *</td>
<td>0.08</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Sort †</td>
<td>1,257.10</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3&lt;sup&gt;rd&lt;/sup&gt; Sort ‡</td>
<td>0.13</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; Pre-join †</td>
<td>82.49</td>
<td>3,004.86</td>
<td>N/A</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Pre-join (step 1) ‡</td>
<td>1.91</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; Pre-join (step 2) ‡</td>
<td>0.04</td>
<td>3.10</td>
<td>N/A</td>
</tr>
<tr>
<td>Join</td>
<td>81.74</td>
<td>104.37</td>
<td>275.36</td>
</tr>
<tr>
<td>Total</td>
<td>1,423.49</td>
<td>3,112.33</td>
<td><strong>275.36</strong></td>
</tr>
</tbody>
</table>

**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

The table below shows the recommended methods for different scenarios based on the size and number of variables in the data sets:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Standard DATA Step</th>
<th>PROC SQL</th>
<th>DATA Step HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small size data sets with many variables</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Moderate size data sets with many variables</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Large size data sets with many variables</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Small size data sets with few variables</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Moderate size data sets with few variables</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Extremely Large size data sets with few variables</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

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