ABSTRACT

No matter what type of programming you do in a pharmaceutical environment there will eventually be a need to combine your data with a look-up table. This look-up table could be a code list for adverse events, a list of names for visits, or just one of your own summary data sets containing totals that you will be using to calculate percentages, and you may have your favourite way to incorporate it. This paper will describe, and discuss the reasons for, using 5 different simple ways to merge data sets with look-up tables, so that, when you take over the maintenance of a new program, you will be ready for anything!

INTRODUCTION

All 5 techniques described in this paper use the same 4 SAS data sets to create the same output data set.

Sample Data Sets

The SAS data sets used in this paper are:

• MAIN = Multiple copies of SASHELP.CARS (428 observations and 15 variables) saved in a single WORK data set to increase the size of this data set.

```sas
%LET mult = 1; /* 10, 100, 1000, 2000, 5000 */

DATA main;
  SET sashelp.cars;
  DO i = 1 TO &mult.;
    OUTPUT;
  END;
RUN;
```

• LOOKUP_ORIGIN (3 observations and 3 variables):

```sql
PROC SQL;
  CREATE TABLE lookup_origin AS
  SELECT origin /* key */
    ,COUNT(DISTINCT make) AS make_n
    ,COUNT(DISTINCT type) AS type_n
  FROM sashelp.cars
  GROUP BY origin
  ORDER BY origin
;
QUIT;
```

![Table showing origin, make_n, and type_n values]

1 Asia
2 Europe
3 USA
• **LOOKUP_TYPE** (15 observations and 6 variables):

```sql
PROC SQL;
CREATE TABLE lookup_make AS
   SELECT origin, make /* keys */
      ,COUNT(DISTINCT model) AS make_model_n
      ,COUNT(DISTINCT type) AS make_type_n
      ,MEAN(msrp) AS make_msrp_mean
      ,MAX(horsepower) AS make_horsepower_max
   FROM   sashelp.cars
   GROUP BY
   origin, make
ORDER BY
   origin, make
;
QUIT;
```

```
<table>
<thead>
<tr>
<th>Origin</th>
<th>Type</th>
<th>type_model_n</th>
<th>type_make_n</th>
<th>type_msrp_mean</th>
<th>type_horsepower_max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia</td>
<td>Hybrid</td>
<td>3</td>
<td>2</td>
<td>19920</td>
<td>110</td>
</tr>
<tr>
<td>Asia</td>
<td>SUV</td>
<td>23</td>
<td>11</td>
<td>25663</td>
<td>325</td>
</tr>
<tr>
<td>Asia</td>
<td>Sedan</td>
<td>83</td>
<td>13</td>
<td>27836</td>
<td>340</td>
</tr>
<tr>
<td>Asia</td>
<td>Sports</td>
<td>17</td>
<td>9</td>
<td>32510</td>
<td>300</td>
</tr>
<tr>
<td>Asia</td>
<td>Truck</td>
<td>8</td>
<td>4</td>
<td>20383</td>
<td>305</td>
</tr>
<tr>
<td>Asia</td>
<td>Wagon</td>
<td>11</td>
<td>9</td>
<td>23743</td>
<td>315</td>
</tr>
<tr>
<td>Europe</td>
<td>SUV</td>
<td>10</td>
<td>6</td>
<td>40346</td>
<td>340</td>
</tr>
<tr>
<td>Europe</td>
<td>Sedan</td>
<td>76</td>
<td>8</td>
<td>42982</td>
<td>453</td>
</tr>
<tr>
<td>Europe</td>
<td>Sports</td>
<td>23</td>
<td>5</td>
<td>71996</td>
<td>493</td>
</tr>
<tr>
<td>Europe</td>
<td>Wagon</td>
<td>12</td>
<td>6</td>
<td>37901</td>
<td>340</td>
</tr>
<tr>
<td>USA</td>
<td>Sedan</td>
<td>80</td>
<td>12</td>
<td>24636</td>
<td>302</td>
</tr>
<tr>
<td>USA</td>
<td>Sports</td>
<td>9</td>
<td>6</td>
<td>24527</td>
<td>500</td>
</tr>
<tr>
<td>USA</td>
<td>Truck</td>
<td>18</td>
<td>5</td>
<td>27220</td>
<td>346</td>
</tr>
<tr>
<td>USA</td>
<td>Wagon</td>
<td>7</td>
<td>6</td>
<td>23345</td>
<td>280</td>
</tr>
</tbody>
</table>
```

• **LOOKUP_MAKE** (38 observations and 6 variables):

```sql
PROC SQL;
CREATE TABLE lookup_type AS
   SELECT origin, type /* keys */
      ,COUNT(DISTINCT model) AS type_model_n
      ,COUNT(DISTINCT make) AS type_make_n
      ,MEAN(msrp) AS type_msrp_mean
      ,MAX(horsepower) AS type_horsepower_max
   FROM   sashelp.cars
   GROUP BY
   origin, type
ORDER BY
   origin, type
;
QUIT;
```
DATA STEP MERGE

This is probably the most commonly used technique to merge SAS data sets together. However, it is also one of the least efficient method, as, for each join, the 2 data sets must be sorted the same way. In this example, even though the smaller lookup data sets would be quicker to sort, the large master data set has to resorted each time.

The output data set will include 4 new calculated variables:

- make_msrp_flag: which is set to 1 if msrp > mean msrp by make.
- make_horsepower_pct: which is set to the percentage of the maximum horsepower by make.
- type_msrp_flag: which is set to 1 if msrp > mean msrp by type.
- type_horsepower_pct: which is set to the percentage of the maximum horsepower by type.

```sas
PROC SORT DATA = main OUT = datastepmerge1;
   BY origin make;
RUN;

DATA datastepmerge2;
   MERGE datastepmerge1 lookup_origin;
   BY origin;
RUN;

DATA datastepmerge3;
   MERGE datastepmerge2 lookup_make;
   BY origin make;
   IF msrp > make_msrp_mean THEN make_msrp_flag = 1;
   ELSE make_msrp_flag = 0;
   make_horsepower_pct = 100 * horsepower / make_horsepower_max;
RUN;
```
PROC SORT DATA = datastepmerge3 OUT = datastepmerge4;
   BY origin type;
RUN;

DATA datastepmerge5;
   MERGE datastepmerge4 lookup_type;
   BY origin type;
   IF msrp > type_msrp_mean THEN type_msrp_flag = 1;
       ELSE type_msrp_flag = 0;
   type_horsepower_pct = 100 * horsepower / type_horsepower_max;
RUN;

SQL JOIN

The strange fact about PROC SQL is that it becomes less efficient with increasing data, and yet it is rarely used with clinical data, where the data volumes are low, but widely used with financial data, where the data volumes are high.

PROC SQL;
   CREATE TABLE sqljoin1 AS
      SELECT a.*
         ,b.make_n
         ,b.type_n
         ,c.make_model_n
         ,c.make_type_n
         ,c.make_msrp_mean
         ,(CASE
            WHEN a.msrp > c.make_msrp_mean THEN 1
            ELSE 0
            END) AS make_msrp_flag
         ,c.make_horsepower_max
         ,(100 * a.horsepower / c.make_horsepower_max)
         AS make_horsepower_pct
         ,d.type_model_n
         ,d.type_make_n
         ,d.type_msrp_mean
         ,(CASE
            WHEN a.msrp > d.type_msrp_mean THEN 1
            ELSE 0
            END) AS type_msrp_flag
         ,d.type_horsepower_max
         ,(100 * a.horsepower / d.type_horsepower_max)
         AS type_horsepower_pct
   FROM main a
   LEFT JOIN
      lookup_origin b
   ON a.origin = b.origin
   LEFT JOIN
      lookup_make c
   ON a.origin = c.origin AND a.make = c.make
   LEFT JOIN
      lookup_type d
   ON a.origin = d.origin AND a.type = d.type
   ;
QUIT;
GENERATED SAS FORMATS

Using SAS formats is inherently more efficient than joining data sets directly, as the format data is stored in memory, rather than on disk. There is a small downside, as you have to convert the data sets into formats, but as these data sets are relatively small, there is a significant benefit to using SAS formats as lookup tables. The available memory is going to be a limiting factor in the usable size of the format, but formats in excess of 50,000 entries are perfectly acceptable.

```sas
DATA format_origin;
  LENGTH fmtname $7 start $80 label 8 type hlo $1;
  SET lookup_origin;
  type = 'I';
  hlo = ' ';
  start = origin;
  fmtname = 'originm';
  label = make_n;
  output;
  fmtname = 'origint';
  label = type_n;
  output;
RUN;

PROC SORT DATA = format_origin NODUPKEY;
   BY fmtname start;
RUN;

PROC FORMAT CNTLIN = format_origin;
RUN;

%MACRO generate_format(level1=, level2=);
  DATA format_&level1. ;
  LENGTH fmtname $7 start $80 label 8 type hlo $1;
  SET lookup_&level1. (RENAME = (&level1.=level1));
  type = 'I';
  hlo = ' ';
  start = CATX('|', origin, level1);
  fmtname = "&level1.c";
  label = &level1._model_n;
  output;
  fmtname = "&level1.x";
  label = &level1._&level2._n;
  output;
  fmtname = "&level1.p";
  label = &level1._msrp_mean;
  output;
  fmtname = "&level1.h";
  label = &level1._horsepower_max;
  output;
RUN;

PROC SORT DATA = format_&level1. NODUPKEY;
   BY fmtname start;
RUN;

PROC FORMAT CNTLIN = format_&level1. ;
RUN;
%MEND generate_format;
```
To save all of the sorting and memory usage why not just generate Data Step code to add the extra information from the lookup data sets.  In this case IF .. THEN .. ELSE statements are generated from the lookup data sets, and stored as text records in a SAS Catalog Source entry in a WORK catalog, so they are deleted automatically at the end of the SAS session.

FILENAME srcif CATALOG "work.generateif";

DATA _NULL_;  
SET lookup_origin END = eof;
FILE srcif(origin.source);
IF _N_ = 1 THEN PUT "IF origin = '" origin +(-1) "' THEN DO;";
ELSE PUT "ELSE IF origin = '" origin +(-1) "' THEN DO;";
PUT "make_n = " make_n ";";
PUT "type_n = " type_n ";";
PUT "END;";
RUN;

%MACRO generate_if(level1=, level2=);  
DATA _NULL_;  
SET lookup_&level1.. END = eof;
FILE srcif(&level1..source);
IF _N_ = 1 THEN PUT "IF origin = '" origin +(-1) "' THEN DO;";
ELSE PUT "ELSE IF origin = '" origin +(-1) "' AND &level1. = '" 
&level1. +(-1) "' THEN DO;";
PUT "&level1._model_n = " &level1._model_n ";";
PUT "&level1._&level2._n = " &level1._&level2._n ";";
PUT "&level1._msrp_mean = " &level1._msrp_mean ";";
PUT "IF msrp > &level1._msrp_mean THEN &level1._msrp_flag = 1;";
PUT " ELSE &level1._msrp_flag = 0;";
RUN;
GENERATED SELECT .. WHEN .. OTHERWISE

In this case SELECT .. WHEN .. OTHERWISE statements are generated from the lookup data sets, and stored as text records in a SAS Catalog Source entry in a WORK catalog, so they are deleted automatically at the end of the SAS session. The advantage of these statements over IF .. THEN .. ELSE is that the OTHERWISE statement forces an action if none of the previous test are satisfied, so it can be used to highlight any omissions.

FILENAME srcsel CATALOG "work.generateselect";

DATA _NULL_;  
SET lookup_origin END = eof;  
FILE srcsel(origin.source);  
IF _N_ = 1 THEN PUT "SELECT;";  
PUT "WHEN (origin = ' origin +(-1) '') DO;";  
PUT "make_n = " make_n ";";  
PUT "type_n = " type_n ";";  
PUT "END;";  
IF eof THEN DO;  
PUT "OTHERWISE;";  
PUT "END;";  
END;  
RUN;

%MACRO generate_select(level1=, level2=);  
DATA _NULL_;  
SET lookup_&level1.. END = eof;  
FILE srcsel(make.source);  
IF _N_ = 1 THEN PUT "SELECT;";  
PUT "WHEN (origin = ' origin +(-1) '') AND &level1.. = ' &level1.. +(-1) '') DO;";  
PUT "&level1.._model_n = " &level1.._model_n ";";  
PUT "&level1.._&level2.._n = " &level1.._&level2.._n ";";  
PUT "&level1.._msrp_mean = " &level1.._msrp_mean ";";  
PUT "IF msrp > &level1.._msrp_mean THEN &level1.._msrp_flag = 1;";  
PUT " ELSE &level1.._msrp_flag = 0;";  
PUT "&level1.._horsepower_max = " &level1.._horsepower_max ";";  
PUT "&level1.._horsepower_pct = 100 * horsepower"  
" / &level1.._horsepower_max;";  
PUT "END;";  
RUN;  
%MEND generate_select;
IF eof THEN DO;
    PUT "OTHERWISE;";
    PUT "END;";
END;
RUN;
%MEND compare_select;

%generate_select(level1=make, level2=type);
%generate_select(level1=type, level2=make);

DATA generateselect1;
    SET main;
    %INCLUDE srcsel(origin.source);
    %INCLUDE srcsel(make.source);
    %INCLUDE srcsel(type.source);
RUN;

CONCLUSIONS

Comparing techniques using CPU time shows that Data Step Merge and SQL Joins are comparable, Formats are quicker, and the Data Step statement-generating techniques are the quickest.

![Lookup Results](image)

**Lookup Results**

**type=CPU time (secs)**

<table>
<thead>
<tr>
<th>Duplicates of SASHELP.CARS</th>
<th>Recorded values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>2000</td>
<td>4</td>
</tr>
<tr>
<td>3000</td>
<td>6</td>
</tr>
<tr>
<td>4000</td>
<td>8</td>
</tr>
<tr>
<td>5000</td>
<td>10</td>
</tr>
</tbody>
</table>

**Key:**
- Data Step Merge
- SQL Join
- Formats
- Generate Select
- Generate Select 2

8
However, comparing techniques using elapsed time is more interesting, as all of the techniques show a linear increase in elapsed time, apart from SQL Join, which increases dramatically as the volume increases after being fairly fast as low data volumes. This is because, at low data volumes, PROC SQL carries out most of its data processing in memory. At higher data volumes it is forced to use the WORK library to store intermediate data, which is much less efficient. Both Data Step Merge and SQL Join are significantly less efficient than the other 3 techniques, which minimise the number of data passes.

Are you now considering a new technique for using lookup tables?

REFERENCES

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

Author Name       Philip R Holland
Company           Holland Numerics Ltd
Address           94 Green Drift
City / Postcode   Royston, Hertfordshire, SG8 5BT, United Kingdom
Work Phone:       +44 7714 279085
Fax:              +44 1763 242486
Email:            phil@hollandnumerics.com
Web:              www.hollandnumerics.com/SASPAPER.HTM