A Join for All Seasons

Caroline Bahler, Meridian Software, Inc.

Abstract

The ability to combine different types of data from multiple hardware and software platforms is a major strength of the SAS® system. SAS has blessed information analysts with a wealth of different options for joining data values from many different data structures. Therefore, an information analyst needs to determine who (parent data structures), what ("software environment"), where ("hardware environment"), how (the tools available to perform the join) and why (the required contents of the child data set) of a join in order to determine which strategy to use. This paper will discuss the joining techniques offered within the SAS system and give examples of their use.

Introduction

Data warehouses can contain data collected and stored in many different physical forms. These data structures (the physical form of the data) can include flat files, database tables, spreadsheets, and SAS data sets. Utilization of this "raw" data by an information analyst can require combining two or more of these data structures through the use of a join (merging and joining are synonymous terms referring to the combination of data structures through the use of common variables/fields). One of the strengths of the SAS system is that it provides many different options for joining data values from many different data structures.

Joining Strategies

Determining the appropriate joining strategy for a given situation requires that the information analyst evaluate the who, what, where, why and how of a join.

- **Who and Why- "Parent" and "Child" Data Structures**

Selection of the "parent" data structures is totally dependent on the required contents of the "child" data structure. In other words, the "parent" data
structures should contain all variables and data values that are needed in the final outcome of the "child" data structure. The "parent" data structures are the SAS data sets, database tables and other data structures contained within a data warehouse. The job of the information analyst is to identify which of these stored data structures need to be joined to create the required new "child" data structure.

- **What - "Software Environment"**

The "software environment" also greatly influences how a group of two (2) or more data structures are joined. Selection of a joining strategy increases in complexity as the number of "software environments" containing data increases. A standard rule of thumb is that data structures from the same environment should be joined within that environment. However, this does not always hold true since system resources and other factors may indicate that it is more "efficient" to join data structures from the same software within a different "environment".

Figure 1 illustrates a common joining strategy with some of the ambiguities involved. The optimum place to join a SAS data set and a flat file is within SAS. However, whether the database tables are joined within the database environment depends upon the type of join required. For instance, outer joins can be very database resource intensive and the "better" choice might be to join the two database tables within the SAS environment.

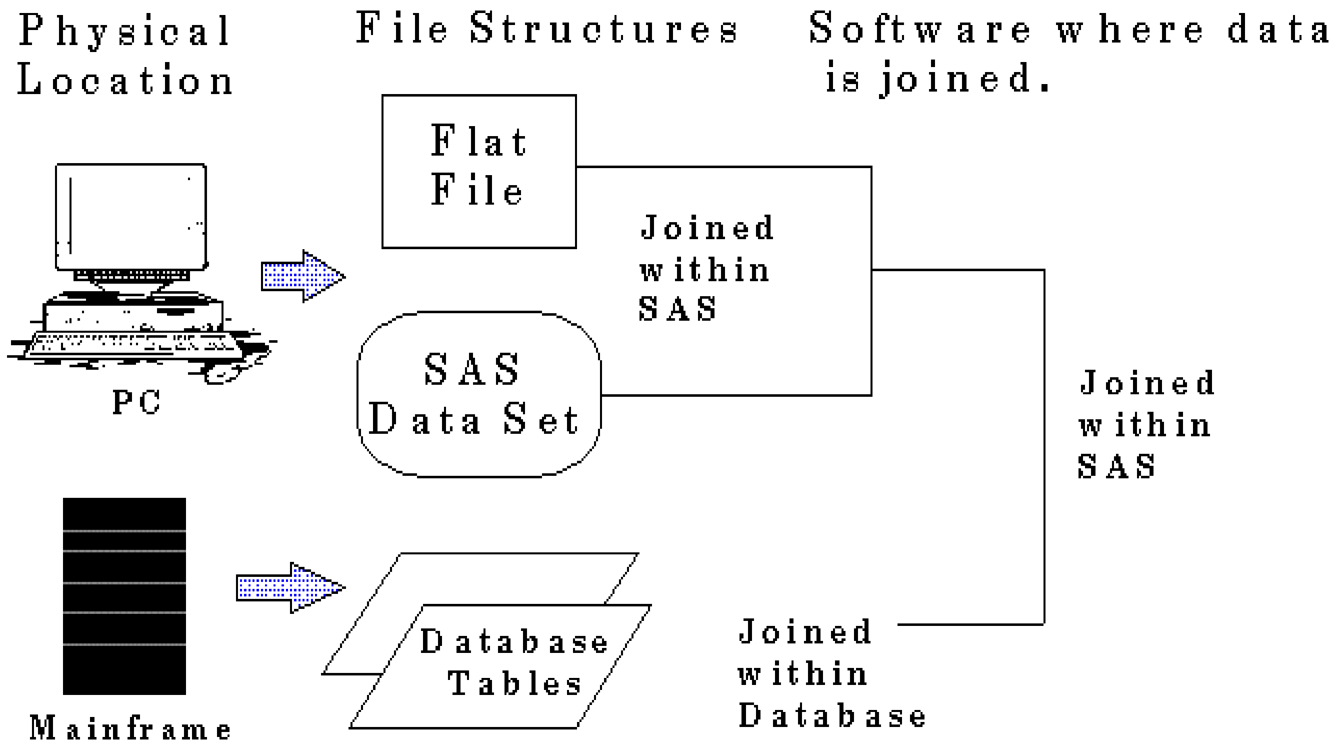

Figure 1. Example of joining strategy.

Therefore, determining whether data structures should be joined within a given “software environment” depends upon the impact of the system resources by this data manipulation and whether the joining tools available within the “environment” produce the required result. If the impact on system resources is acceptable and the tools available produce the required results then the data structures should be joined within that environment. If the answer to the first question is “no” then other “environments” need to be evaluated. However, this decision is not always black and white, and a large draw upon system resources for a given join may be acceptable within certain situations. Therefore, each joining scenario needs to evaluated thoroughly to determine the costs vs. benefits of joining data structures within a given environment.

- **Where - Physical Location of the Join “The Hardware”**

Physical location defines where the data resides (hardware platform) such as two SAS data sets with one residing on a PC and one on a mainframe. Deciding on which hardware platform to join these data sets can be an involved process. The choice is not always clear because advances in PC CPU processor power and speed blurs the line between a mainframe approach and a PC approach. Bench-marking of each hardware system is critical to developing a realistic joining strategy.

- **How - Selection of a “Joining Tool”**

Availability of the proper software joining tool is the last decision required of the information analyst. The rest of this paper will assume that the SAS environment was the correct place for a join to occur.

The following criteria affect joining tool selection:

b) the information requirements of the “child” data set and

c) the type and amount of information contained within the “parent” data structures.

Table 1 below lists examples of different information requirements for a “child” data set and the type of join and SAS tool used to produce that result.

Child Informational Requirements	Type of Parent Data Set Join	SAS Tool
A. All data values from all parent data sets.	Match-Merge or Full Outer Join	• MERGE statement with BY statement[4] • PROC SQL[2]
B. All data values from a single parent data set (base) and all data values from the other parent data set(s) that match the data values of the joining variables within the base.	Non-base parent data set(s) are used as • "Look-up" table(s) • Right or Left outer join.	• PROC FORMAT[1] • SET statement with KEYS= option[3] • PROC SQL[2]
C. Only those data values from all parent data sets that contain the same data values within the joining variables.	Inner Join	• PROC SQL[2] • MERGE statement with IN= option and BY statement4
D. Placement of parent data sets side by side	One-to-one Merge	MERGE statement[4]
E. Expansion of child data set to include all levels of a non-common variable.	Many-to-many Join	PROC SQL[2]

Table 1. The type of parent data set joins required to construct a specific child data set.

Scenario A - Groups of two or more parent data sets are used to build or create a child or output data set. In scenario A all data values from all of the parent data sets are needed to create the child data set (Table 1). The type of joins used to accomplish this are a match-merge or a full join. Missing values are placed within observations that do not occur within all parent data sets. Example 1 illustrates the two SAS tools capable of full joins.

Scenario B - All data values of a base parent data set are kept and only those observations containing matching data values of the common variable(s) are selected from all other parent data sets (Table 1). The terminology right or left join is an indication of which parent data set to use as the base. Example 2 illustrates all three SAS tools available for right or left joins.

Scenario C - An inner join is used when the child data set needs to contain only those data records from the parent data sets where the common variable(s) are identical. Example 3 illustrates an inner join.

Scenario D - One-to-one merging combines all of the parent data sets using a common variable and creates a child data set that is as large as the largest data set within the merge list. The parent data sets are not joined by common variables. Instead parent data sets are placed side by side and each common variable data value is super-imposed by the data values within the last parent data set within the merge. Example 4 illustrates when a one-to-one merge could be used.

Scenario E - Expansion of the child data set occurs when one data set has multiple levels of a non-common variable. An example would be where one data set contains a listing of names by city and the other contains city information. The child data set required contains all of the names for each city plus the city information. In this case the data sets are joined by the common variable city and all names within a city are transferred to the child data set (Example 5).

SAS Tool	Pros	Cons
PROC FORMAT[1]	Creates a "look-up" table using either single variables or multiple variables for the key and label components of the format.	The key values must be unique, no duplicate values can occur within the data set used to create the format. Format can be applied to only one data set at a time.
MERGE statement[4]	• Used only for one-to-one merges no sorting required. • Two (2) or more data sets can be joined.	The data sets are not joined by any common variables. If there is a common variable between the data sets, then the common variable will contain the values of the common variable in the last data set joined.
MERGE statement with BY statement[4]	• Two or more data sets can be joined by common variables. • All values of all variables within all data sets will be retained.	All data sets must be sorted by common variable.
MERGE statement with IN= option and BY statement[4]	• Two (2) or more data sets can be joined by common variables. • All data from each data set will be read before subsetting criteria applied.	All data sets must be sorted by common variable.
PROC SQL[2]	• Data sets do not need to be pre-sorted. • Inner joins can occur between two (2) or more data sets.	• All outer joins occur between two data sets only. • Inner join contains only those values of the common variable that match between the data sets joined.
SET statement with KEYS= option[3]	Two (2) or more data sets can be joined by common variables.	Data sets need to be indexed by common variables.

Table 2. Pros and cons for using each type of SAS joining tool.

Conclusions

The pros and cons of using the different joins are listed in Table 2.

The selection of a joining tool is dependent on the environment of the parent data structures, the required contents of the child data set and what tool is the most system resource efficient. During the application development process, careful bench-marking of joining tools is required to ensure selection of the correct environment and tool for the job.

Literature Cited

1. SAS Institute Inc.. SAS Procedures Guide, Version 6, Third Edition. Cary, NC: SAS Institute Inc., 1990. 275-312 pp.
2. SAS Institute. SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition. Cary , NC: SAS Institute Inc., 1989. 142-147 pp.
3. SAS Institute. SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07. Cary, NC: SAS Institute Inc., 1991. 91, 207-217 pp.
4. SAS Institute. SAS Language: Reference, Version 6, First Edition. Cary, NC: SAS Institute Inc., 1990. 147-155 pp.
5. Bahler, C. and Clos, S.. To Format or Merge ... That is the Question. Proceedings

of the Southeast SAS Users Group. 3:363-367.

Trademarks

Contact Information

Caroline Bahler
Meridian Software, Inc.
12204 Old Creedmoor Road
Raleigh, NC 27613
(919) 518-1070
merccb@meridian-software.com

Example 1 - Scenario A: examples of both a match-merge and SQL full outer joins.

Data sets used in this example:

REPS	
REGION	**REPNAME**
Southeast	Jones
Southeast	Smith
Southeast	Doe
Northeast	Harris
Northeast	James
Northeast	Finch

RGN		
CITY	**ST**	**REGION**
Atlanta	GA	Southeast
New York	NY	Northeast
Portland	OR	Northwest

Match-Merge: Both data sets must be have the same variables sorted in the same order. All data values from both parents are present.

```
PROC SORT DATA=REPS;
 BY REGION;
RUN;

PROC SORT DATA=RGN;
 BY REGION;
RUN;

❶DATA REPRGN;
   MERGE REPS RGN;
    BY REGION;
  RUN;
```

Below are two examples of Full Outer Joins using PROC SQL (structured query language). These two joins differ because of the order of the variables in the select portion of the SQL. As in the match-merge all values from both parent data sets are present.

```
PROC SQL;
❷CREATE TABLE RGNREPSA AS
 SELECT A.REGION, B.CITY, B.ST, A.REPNAME
   FROM REPS A FULL JOIN RGN B
     ON A.REGION=B.REGION;

❸CREATE TABLE RGNREPSB AS
 SELECT A.REGION,
        B.CITY, B.ST,
        A.REPNAME
   FROM RGN A FULL JOIN REPS B
     ON A.REGION=B.REGION;
QUIT;
```

❶Match-Merge Results			
REGION	**REPNAME**	**CITY**	**ST**
Northeast	Harris	New York	NY
Northeast	James	New York	NY
Northeast	Finch	New York	NY
Northwest		Portland	OR
Southeast	Jones	Atlanta	GA
Southeast	Smith	Atlanta	GA
Southeast	Doe	Atlanta	GA

❸Full Outer Join Results - B					❷Full Outer Join Results - A			
REGION	**CITY**	**ST**	**REPNAME**		**REGION**	**CITY**	**ST**	**REPNAME**
Northeast	New York	NY	Harris		Northeast	New York	NY	Harris
Northeast	New York	NY	James		Northeast	New York	NY	James
Northeast	New York	NY	Finch		Northwest	Portland	OR	
Northwest	Portland	OR			Southeast	Atlanta	GA	Jones
Southeast	Atlanta	GA	Jones		Southeast	Atlanta	GA	Smith
Southeast	Atlanta	GA	Smith		Southeast	Atlanta	GA	
Southeast	Atlanta	GA			Northeast	New York	NY	Finch

Example 2 - Scenario B: examples of a “look-up” table, SET statement with KEYS= option and SQL left join.

Creation of “look-up” table using PROC FORMAT:

```
➀DATA CALENDAR;
 LENGTH MON $10.;
 WEEK=1;
 DO DAY = ‘01JAN96’ TO ‘31DEC97’D;
   IF WEEKDAY(DAY) = 1 THEN  WEEK= WEEK + 1;
   IF WEEK=53 THEN WEEK=1;
   DATE=LEFT(PUT(DAY,WORDDATE18.));
   MON=LEFT(SCAN(DATE,1));
   MONTH=MONTH(DAY);
   YEAR=YEAR(DAY); FISCYEAR=YEAR;
   IF MONTH=12 AND WEEK=1 THEN FISCYEAR=YEAR + 1;
   KEEP DATE WEEK MONNAME FISCYEAR;
  OUTPUT;
 END;
RUN;
```

Format Creation - The code below creates variables required by PROC FORMAT[5].

```
DATA WORDDATE;
 SET CALENDAR END=EOF;
  HLO=’ ’;
  START=DATE;
  LABEL=PUT(FISCYEAR,Z4.)||MON||PUT(WEEK,Z2);
  FMTNAME=’YMW’;
  TYPE=’C’;
  OUTPUT WORDDATE;
  IF EOF THEN DO;
    HLO=’O’;
    START=’OTHER’;
    LABEL=’ ’;
    OUTPUT WORDDATE;
 END;
RUN;

PROC FORMAT CNTLIN=WORDDATE;
RUN;
```

The following data step uses the format created in the previous PROC FORMAT statement to create two (2) new variables containing data values from the calendar data set.

```
ƒDATA SALESA;
 LENGTH FISCYEAR $4 MONNAME $10 WEEK $2;
  SET ,SALES;
FISCYEAR=SUBSTR(PUT(DATE,$YMW.),1,4);
MONNAME=SUBSTR(PUT(DATE,$YMW.),5,10);
WEEK=SUBSTR(PUT(DATE,$YMW.),11);
RUN;
```

The following Left Inner Join creates the same child data set.

```
ƒPROC SQL;
  CREATE TABLE SALEC AS
   SELECT B.FISCYEAR,
          B.MON,
          B.WEEK,
          A.DATE,
          A.SALES
     FROM SALES A LEFT JOIN
            CALENDAR B
       ON A.DATE=B.DATE;
 QUIT;
```

Use of the Set Statement with Key= Option to join the SALES and CALENDAR data sets.

```
PROC DATASETS LIB=WORK;
 MODIFY CALENDAR;
 INDEX CREATE DATE;
QUIT;

DATA SALEB;
  SET SALES;
  SET CALENDAR KEY=DATE;
DROP FISCYEAR;
RUN;
```

①Sample of CALENDAR Data Set

MONNAME	WEEK	DATE	MONTH	FISCYEAR
January	1	January 1, 1999	1	1999
January	1	January 2, 1999	1	1999
January	1	January 3, 1999	1	1999
January	1	January 4, 1999	1	1999
January	1	January 5, 1999	1	1999
January	1	January 6, 1999	1	1999
January	1	January 7, 1999	1	1999

Sample of SALES Data Set

DATE	SALES
January 1, 1999	67561.59
January 2, 1999	1162.28
January 3, 1999	52629.84
January 4, 1999	34110.43
January 5, 1999	25372.64
January 6, 1999	15320.03
……..	…….

ƒSample of child data sets created by the Format and Left Join techniques.

FISCYEAR	MON	WEEK	DATE	SALES
1999	April	14	April 1, 1999	3922.42
1999	April	15	April 10, 1999	2438.21
1999	April	15	April 11, 1999	3194.39
1999	April	15	April 12, 1999	5690.00
1999	April	15	April 13, 1999	1357.40
1999	April	16	April 14, 1999	2648.88
1999	April	16	April 15, 1999	4963.07
1999	April	16	April 16, 1999	3274.21

Sample of SALEB Data Set.

DATE	SALES	MON	WEEK	MONTH
January 1, 1999	67561.59	January	1	1
January 2, 1999	1162.28	January	1	1
January 3, 1999	52629.84	January	1	1
January 4, 1999	34110.43	January	1	1
January 5, 1999	25372.64	January	1	1
January 6, 1999	15320.03	January	1	1

Example 3 - Scenario C: examples of SQL inner joins and MERGE statement with IN= options.

Inner join using PROC SQL.

```
PROC SQL;
CREATE TABLE RGNREP AS
 SELECT A.REGION,
            B.CITY,  B.STATE,
            A.REPNAME
   FROM REPS A, RGN B
  WHERE A.REGION=B.REGION;
QUIT;
```

Match-merge restricting the data values kept.

```
DATA REPRGN;
 MERGE REPS(IN=A) RGN;
  BY REGION;
IF A;
RUN;
```

Results from both the Inner Join and Merge

REGION	CITY	ST	REPNAME
Northeast	New York	NY	Harris
Northeast	York	NY	James
Northeast	New York	NY	Finch
Southeast	Atlanta	GA	Jones
Southeast	Atlanta	GA	Smith
Southeast	Atlanta	GA	Doe

Example 4 - Scenario D: example of one-to-one merge.

```
PROC SUMMARY DATA=SALES;
 VAR SALES;
 OUTPUT OUT=SALESTOT(DROP=_TYPE_  _FREQ_)
       SUM=TOTSALES;
RUN;
```

Output of above summary procedure:
TOTSALES
3199002.11

```
DATA SUMMARY;
   MERGE REPS SALETOTS;
   RETAIN SALES;
   SALES=TOTSALES;
  RUN;
```

›Results from One-to-One Merge

REGION	REPNAME	TOTSALES	SALES
Northeast	Harris	3199002.11	3199002.11
Northeast	James	.	3199002.11
Northeast	Finch	.	3199002.11
Southeast	Jones	.	3199002.11
Southeast	Smith	.	3199002.11
Southeast	Doe	.	3199002.11

Note: This type of merge is only useful in adding one or more total variables for use in calculations.

Example 5 - Scenario E: example of a many to many join.

Parent data sets.

REPS

REGION	**REPNAME**
Southeast	Jones
Southeast	Smith
Southeast	Doe
Northeast	Harris
Northeast	James
Northeast	Finch

RGN

CITY	**ST**	**REGION**
Atlanta	GA	Southeast
Miami	FL	Southeast
Boston	MA	Northeast
New York	NY	Northeast

Match-merge of the two (2) parent data sets.

```
PROC SORT DATA=REPS;
 BY REGION;
RUN;

PROC SORT DATA=RGN;
 BY REGION;
RUN;

  DATA REPRGN;
 MERGE REPS RGN;
  BY REGION;
RUN;
```

Many-to-many Join

```
PROC SQL;
   CREATE TABLE RGNREPS AS
  SELECT A.REGION, A.CITY, A.ST,
              B.REPNAME
    FROM RGN A, REPS B
   WHERE A.REGION=B.REGION ;
QUIT;
```

or

```
PROC SQL;
  CREATE TABLE RGNREPS AS
  SELECT A.REGION, A.CITY, A.ST,
              B.REPNAME
    FROM RGN A LEFT JOIN REPS B
             ON A.REGION=B.REGION ;
QUIT;
```

Match-merge results in a child data set that matches rep to a city in each region.

REGION	**REPNAME**	**CITY**	**ST**
Northeast	Harris	Boston	MA
Northeast	James	New York	NY
Northeast	Finch	New York	NY
Southeast	Jones	Atlanta	GA
Southeast	Smith	Miami	FL
Southeast	Doe	Miami	FL

The “child” data set of a many-to-many join contains a record for each rep in each city.

REGION	**CITY**	**ST**	**REPNAME**
Northeast	Boston	MA	Harris
Northeast	Boston	MA	James
Northeast	Boston	MA	Finch
Northeast	New York	NY	Harris
Northeast	New York	NY	James
Northeast	New York	NY	Finch
Southeast	Atlanta	GA	Jones
Southeast	Atlanta	GA	Smith
Southeast	Atlanta	GA	Doe
Southeast	Miami	FL	Jones
Southeast	Miami	FL	Smith
Southeast	Miami	FL	Doe