

Using DATA Step MERGE and PROC SQL JOIN to Combine SAS® Datasets

Dalia C. Kahane, Westat, Rockville, MD

ABSTRACT

This paper demonstrates important features of combining datasets in SAS. The facility to combine data from different sources and create a convenient store of information in one location is one of the best tools offered to the SAS programmer. Whether you merge data via the SAS data step or you join data via PROC SQL you need to be aware of important rules you must follow. By reading this paper you will gain a deeper understanding of how the DATA Step MERGE works and will be able to compare it to parallel PROC SQL JOIN examples. You may then select what is best for your own programming style and data situation.

The paper includes descriptions and examples that cover the following: how to combine data from multiple sources where records share a relationship of one-to-one, one-to-many or many-to-many; the importance of the BY statement in the DATA Step and the WHERE clause in PROC SQL; the dangers inherent in using (or not using) the BY and WHERE; the importance of knowing your data and the fields that are common to all datasets or unique in each; and the syntax for properly performing different types of joins in SQL (inner vs. outer join, left vs. right join, etc.)

INTRODUCTION

The data step language with the MERGE and BY statements provides a powerful method for doing one-to-one combining of data. It can also be used as a powerful look up tool where blocks of observations require combining or looking up information in a corresponding single observation in another data set. This is an important part of SAS programming.

The SQL procedure offers another tool for combining data sets through a JOIN. In particular it offers the inner join, left join, right join, full join and natural join. This paper will look at the two different methods of combining data and compare them. One-to-one, one-to-many and many-to-many relationships will be explored and the importance of the MERGE BY and the JOIN WHERE will be described. But first let's look at some examples of why you might want to combine data:

- Educational data about students appear in multiple files, one per class; you want to combine the data to show a student's performance across all classes;
- Your client wants to see educational performance data where you need to combine teachers' data with that of their students;
- You have research data about physicians and you want to compare individual against national or regional averages; combining individual data with summary data;
- You have two datasets: one which contains data from parents and another which contains data from children; you want to combine them; parents may have multiple children and vice versa.

All the examples used in this paper limit the merge/join to two source datasets, so that the rules can be more easily demonstrated for the beginner programmer. Also, in order to keep things simple, the examples are about toys and children with very few observations in each dataset/table.

DATASETS USED TO ILLUSTRATE COMBINING OF DATA

The datasets that will be used as examples for the purpose of discussing the Merge and Join actions include the following:

- Toy – dataset includes the code, description and company code for various toys

Code	Description	CompanyCode
1202	Princess	1000
0101	Baby Doll	1000
1316	Animal Set	1000
3220	Model Train	1000
3201	Electric Truck	1000
4300	Animal Cards	2000
4400	Teddy Bear	2000

- ToyGenderAge – provides associated gender and recommended-age information for each toy

Code	Gender	AgeRangeLow	AgeRangeHigh
1202	F	6	9
0101	F	4	9
1316	B	3	6
3220	M	6	9
3201	M	6	9
5500	M	2	6

- Company – provides the company code and name for toy manufacturers

CompanyCode	CompanyName
1000	Kids Toys
2000	More Toys

- Factory – provides data on all factories associated with each company

Company Code	FactoryCode	FactoryState
1000	1111	MD
1000	1112	NY
1000	1113	VT
2000	2221	AZ
2000	2222	ME
2000	2223	CA

DATA STEP MERGE

SAS Merge allows the programmer to combine data from multiple datasets. Each observation from dataset one is combined with a corresponding observation in dataset two (and dataset three, etc.)¹ Which observations and which data fields from the source datasets will be included in the resulting dataset is determined by the detailed “instructions” provided by the programmer.

DEFAULT MERGE: ONE-TO-ONE NO MATCHING

The default action taken by SAS when the code requests a merge between two datasets is to simply combine the observations one by one in the order that they appear in the datasets.

Code:

```
data Merged_ToyGA_default;
    merge Toy ToyGenderAge;
run;
```

Log Results:

INFO: The variable Code on data set WORK.TOY will be overwritten by data set WORK.TOYGENDERAGE.

NOTE: There were 7 observations read from the data set WORK.TOY.

NOTE: There were 6 observations read from the data set WORK.TOYGENDERAGE.

NOTE: The data set WORK.MERGED_TOYGA_DEFAULT has 7 observations and 6 variables.

Combined File:

Code	Description	CompanyCode	Gender	RangeAgeLow	RangeAgeHigh
1202	Princess	1000	F	6	9
0101	Baby Doll	1000	F	4	9
1316	Animal Set	1000	B	3	6
3220	Model Train	1000	M	6	9
3201	Electric Truck	1000	M	6	9
5500	Animal Cards	2000	M	2	6
4400	Teddy Bear	2000		.	.

By default the SAS Merge does the following:

- combines in order observation #1 from Toy with Observation #1 from ToyGenderAge, then Observation #2 with Observation #2, etc.; does **not** try to match on common variable(s); simply matches the observations in the random order that they appear in the datasets
- keeps all observations from both datasets
- the system option **MergeNoBy** is set to NOWARN; neither a warning nor an error message is given to alert user that a merge is being performed without matching observations through a BY statement
- in a one-to-one merge, common observations that are not part of BY statement, act as follows: the value coming in from the right dataset override those coming in from the left dataset – see SAS NOTE in Log results above

This is usually **not** what the programmer would want, since it would make much more sense to combine all data related to toy “Princess” together; all data related to “Electric Truck” together; etc.

ONE-TO-ONE MATCH-MERGE KEEPING ALL OBSERVATIONS

A Match-Merge combines observations from multiple datasets into a single observation in the result dataset based on the values of one or more common variables. It is more effective for the programmer to take control of the SAS Merge process and use matching variable(s). In our example, the matching variable is the Toy Code.

It is highly recommended that you do the following:

- set the system option: **MergeNoBy** = ERROR; this ensures that if a MERGE statement is used without a corresponding BY statement the log will present an error message to that effect;
- use a BY statement in the data step to force SAS to put values into a single observation combining data from observations in the source datasets where the BY Variable has the same value;
- make sure to sort all “source” datasets by the matching variable(s) listed in the BY statement; if the datasets are not sorted properly you will get error messages in the log

Note that by default a match-merge keeps all observations from all datasets; but each “final” observation gets its data values only from data variables that are contributed by those datasets with matching Toy Code values; where there is no contributing matching observation the data values are set to missing.

Code:

```
proc sort data=Toy; by Code; run;
proc sort data=ToyGenderAge; by Code; run;
data Merged_ToyGA_ByCode;
    merge Toy (keep=Code Description)
          ToyGenderAge;
    by Code;
run;
```

Log Results:

NOTE: There were 7 observations read from the data set WORK.TOY.

NOTE: There were 6 observations read from the data set WORK.TOYGENDERAGE.

NOTE: The data set WORK.MERGED_TOYGA_BYCODE has 8 observations and 5 variables.

Combined File:

Code	Description	Gender	AgeRangeLow	AgeRangeHigh
0101	Baby Doll	F	4	9
1202	Princess	F	6	9
1316	Animal Set	B	3	6
3201	Electric Truck	M	6	9
3220	Model Train	M	6	9
4300	Animal Cards			
4400	Teddy Bear			
5500		M	2	6

Note that:

- all observations from both datasets are kept
- where identical code value was found on both datasets, all variables get filled with data coming in from the corresponding source datasets; therefore, for codes 4300 and 4400 the combined observations contain Description values from the Toy dataset but no Gender or Age Range values coming in from the ToyGenderAge dataset; for Code 5500, the combined observation contains data values from the ToyGenderAge dataset but no Description from the Toy dataset

ONE-TO-ONE MATCH-MERGE KEEPING SOME OBSERVATIONS

In many cases we may want to further control which observations (of those that match) will actually be included in the final dataset. This is done via a “subsetting if” statement combined with an “in=” data option. There are several different choices such as:

- keep only those observations where a match is found on the BY variable in all “source” datasets
- keep all those that appear in the 1st dataset whether or not a match is found in the 2nd dataset
- keep all those that appear in the 2nd dataset whether or not a match is found in the 1st dataset

Code:

```
data Merged_ToyGA_KeepOnlyMatched;
    merge Toy (in=a keep=Code Description)
          ToyGenderAge (in=b)
          ;
    by Code;
    if a and b;
run;
```

Log Results:

NOTE: There were 7 observations read from the data set WORK.TOY.

NOTE: There were 6 observations read from the data set WORK.TOYGENDERAGE.

NOTE: The data set WORK.MERGED_TOYGA_KEEPPONLYMATCHED has 5 observations and 5 variables.

Combined File:

Code	Description	Gender	AgeRangeLow	AgeRangeHigh
0101	Baby Doll	F	4	9
1202	Princess	F	6	9
1316	Animal Set	B	3	6
3201	Electric Truck	M	6	9
3220	Model Train	M	6	9

There are several things to note here:

- the “in=” option which uniquely identifies each “source” dataset;
- the “by Code” statement which ensure a one-to-one matching;
- the “if a and b” statement, which instructs SAS to keep only those observations with matching code values in both datasets; those observations with code values that appears only in one of the other dataset, get excluded from the final “combined” dataset;
- a statement of “if a” will ensure that all observations and only observations from dataset Toy will be kept in the “final” dataset; the data values coming from the ToyGenderAge will be set to missing when no matching observation with that Code value is found there; result in log
NOTE: The data set WORK.MERGED_TOYGA_KEEPPONLYLEFT has 7 observations and 5 variables.
- a statement of “if b” will ensure that all observations and only observations from dataset ToyGenderAge will be kept in the “final” dataset; the data values coming from Toy will be set to missing when no matching observation with that Code value is found in the Toy dataset; result in log
NOTE: The data set WORK.MERGED_TOYGA_KEEPPONLYRIGHT has 6 observations and 5 variables.

ONE-TO-MANY MERGE

Using our example datasets, an example of a one-to-many merge is to combine the Toys with their Company name.

There are many toys per company.

Code:

```
proc sort data=Toy; by CompanyCode; run;
proc sort data=Company; by CompanyCode; run;
data Merged_ToyCompany;
  merge Toy Company;
  by CompanyCode;
run;
```

Log Results:

- NOTE: There were 7 observations read from the data set WORK.TOY.
- NOTE: There were 2 observations read from the data set WORK.COMPANY.
- NOTE: The data set WORK.MERGED_TOYCOMPANY has 7 observations and 4 variables.

Combined File:

Code	Description	Company Code	Company Name
0101	Baby Doll	1000	Kids Toys
1202	Princess	1000	Kids Toys
1316	Animal Set	1000	Kids Toys
3201	Electric Truck	1000	Kids Toys
3220	Model Train	1000	Kids Toys
4300	Animal Cards	2000	More Toys
4400	Teddy Bear	2000	More Toys

Important issues to note for the one-to-many merge are:

- In DATA Step MERGE the one-to-many and many-to-one merges are the same! The order of the dataset names within the MERGE statement has no significance; the **actual merge action** still combines one observation from a dataset to one observation from another;

- The data from each observation coming from the “one” side is retained through all merges with the “many” side (until a new observation comes in from the “one” side);
- Common variables that are not included in the BY statement should be renamed since bringing them in may lead to errors. In a one-to-one merge a common variable's value coming from a “later” dataset simply overwrites the value from an “earlier” dataset (in the order that the datasets appear in the MERGE statement). This is **not always** true in the case of a one-to-many merge. It is **good practice to always** rename common variables as they come in from the source files and calculate a “final” value in the result dataset. Use the RENAME= option per dataset in the MERGE statement.

MANY-TO-MANY MERGE

The many-to-many merge refers to the instance where at least two source datasets contain multiple repeats of values in the BY variables used for match-merging. An example of a many-to-many merge using our datasets is to present all possible factories from where all toys of a given company may be shipped. Another way to describe this: show any factory from which any toy may be shipped.

Code:

```
proc sort data=Toy; by CompanyCode Code; run;
proc sort data=factory; by CompanyCode FactoryCode; run;

/* create cross walk dataset to tell us how many factories per company and assigns them numbers */
data Factory(drop=cnt)
  Company_Xwalk(keep=CompanyCode FactoryNumber rename=(FactoryNumber=MaxFactory));
  retain cnt 0;
  set Factory;
  by CompanyCode;
  if first.CompanyCode then cnt=0;
  cnt+1;
  FactoryNumber=cnt;
  output factory;
  if last.CompanyCode then output Company_Xwalk;
run;

/* prepare the toys dataset by merging with the company Xwalk */
data ToyWithFN (drop=i);
  merge Toy(in=a)
        Company_Xwalk(in=b)
  ;
  by CompanyCode;
  if a;
  if not b then put "No Company record found for Toy: " _all_;
  /* per toy - output multiple records - one per factory */
  do i =1 to MaxFactory;
    FactoryNumber=i;
    output;
  end;
run;

/* we now finally merge Toys with Factories */
proc sort data=ToyWithFN; by CompanyCode FactoryNumber Code; run;
proc sort data=factory; by CompanyCode FactoryNumber; run;
data Merged_ToyFactory (drop=MaxFactory FactoryNumber);
  merge ToyWithFN(in=a)
        Factory(in=b)
  ;
  by CompanyCode FactoryNumber;
run;
```

Log Results:

NOTE: The data set WORK.COMPANY_XWALK has 2 observations and 2 variables.

NOTE: The data set WORK.TOYWITHFN has 21 observations and 5 variables.

NOTE: There were 21 observations read from the data set WORK.TOYWITHFN.

NOTE: There were 6 observations read from the data set WORK.FACTORY.

NOTE: The data set WORK.MERGED_TOYFACTORY has 21 observations and 5 variables.

Combined File:

Code	Description	CompanyCode	FactoryCode	FactoryState
0101	Baby Doll	1000	1111	MD
1202	Princess	1000	1111	MD
1316	Animal Set	1000	1111	MD
3201	Electric Truck	1000	1111	MD
3220	Model Train	1000	1111	MD
0101	Baby Doll	1000	1112	NY
1202	Princess	1000	1112	NY
1316	Animal Set	1000	1112	NY
3201	Electric Truck	1000	1112	NY
3220	Model Train	1000	1112	NY
0101	Baby Doll	1000	1113	VT
1202	Princess	1000	1113	VT
1316	Animal Set	1000	1113	VT
3201	Electric Truck	1000	1113	VT
3220	Model Train	1000	1113	VT
4300	Animal Cards	2000	2221	AZ
4400	Teddy Bear	2000	2221	AZ
4300	Animal Cards	2000	2222	ME
4400	Teddy Bear	2000	2222	ME
4300	Animal Cards	2000	2223	CA
4400	Teddy Bear	2000	2223	CA

- Important notes for the many-to-many merge:
- As shown in the SAS code, the goal here is to combine many toys with many factories. The MERGE statement however still ends up combining observations from the 1st dataset with observations of the other datasets, one by one;
 - Before performing the merge, several steps are needed to prepare the source datasets: we use PROC SORT for each source dataset and we create a crosswalk dataset; etc.
 - Much less work is needed to accomplish this task of combining many-to-many in the PROC SQL JOIN (described later in this paper)

PROC SQL JOIN

PROC SQL (which implements the Standard Query Language) allows the user to combine tables through join-queries. Here we use different terminology: tables instead of datasets, join instead of merge, but the concept of combining data from multiple “sources” into a single resulting table is still the same. As described in the “SAS Guide to the SQL Procedure”², the PROC SQL FROM clause is used in a query-expression to specify the table(s), view(s), or query-expressions which are referred to here as the “source” tables, and which can be combined to produce the joined result.

In addition to the various types of joins (inner and outer) that are described in this section, the SQL procedure also offers the feature of “equi-join”. The equi-join refers to the constraint of a matching condition which may be added to any type of join. The true significance of an equi-join is speed. There may be different conditions which help speed and shape the result of a join. Examples include: equality between column values coming from the tables being joined; comparison between calculated values; etc. The WHERE clause or ON clause contains the conditions under which some rows are kept or eliminated in the result table. WHERE is used to select rows from inner joins. ON is used to select rows from inner or outer joins.

DEFAULT JOIN

When the user specifies a simple join-query (i.e. defines a FROM clause with multiple “source” tables but no WHERE or ON clause) the result is a **Cartesian Product**. When two tables are joined, each row of table A is matched with **all the rows** of table B thereby creating a result table that is equal to the product of the number of rows in each of the source tables.

Code:

```
PROC SQL;3
    create table Joined_ToyGA_default as
        select toy.*, tga.*
            from Toy as toy, ToyGenderAge as tga
    ;
quit;
```

Log Results:

NOTE: The execution of this query involves performing one or more Cartesian product joins that can not be optimized.

WARNING: Variable Code already exists on file WORK.JOINED_TOYGA_DEFAULT.

NOTE: Table WORK.JOINED_TOYGA_DEFAULT created, with **42 rows** and 6 columns.

Combined File:

Only the combined data for the Toy rows with the 1st row of the ToyGenderAge are shown here – 7 rows (for 6 Toy row combined with 7 ToyGenderAge rows there are a total of 42 rows)

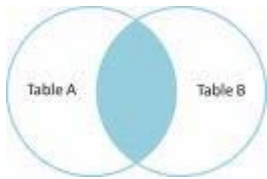
Code	Description	Company Code	Gender	AgeRange Low	Age Range High
1202	Princess	1000	F	6	9
0101	Baby Doll	1000	F	6	9
1316	Animal Set	1000	F	6	9
3220	Model Train	1000	F	6	9
3201	Electric Truck	1000	F	6	9
4300	Animal Cards	2000	F	6	9
4400	Teddy Bear	2000	F	6	9

Please note that :

- SAS indicates that a Cartesian product was involved and implicitly recommends that, when appropriate, indexes should be created and maintained;
- The select statement includes “toy.* and tga.*”; instead an exact list of data fields should have been specified for each of these tables in order to avoid the ugly WARNING that appeared on the log
- Unlike the data step Merge default, which creates a one-to-one row-to-row result, here we get a row-combined-with-all-rows product
- The Cartesian product is also produced if the SQL procedure **cross join** is requested

INNER JOIN

An inner join returns a result table for all the rows in a table that have one or more matching rows in the other table(s), as specified by the sql-expression. A two-table inner join may be viewed as the intersection between table A and table B as shown in the following Venn diagram.⁵



Code:

```
create table Joined_ToyGA_InnerJoin as
  select toy.Code, toy.Description, tga.Gender
  from Toy as toy, ToyGenderAge as tga
  where toy.Code=tga.Code
;
```

Log Results:

NOTE: Table WORK.JOINED_TOYGA_INNERJOIN created, with 5 rows and 3 columns.

Combined File:

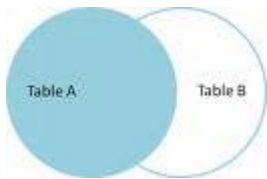
Code	Description	Gender
1202	Princess	F
0101	Baby Doll	F
1316	Animal Set	B
3220	Model Train	M
3201	Electric Truck	M

Here it is important to note that:

- the result here is similar to the “Match-Merge Keeping Some Observations” example shown above (i.e. it is similar to a MERGE with BY and “if a and b”);
- an inner-join may involve up to 32 tables⁴

LEFT JOIN

A LEFT JOIN is one type of OUTER JOIN where the result table includes all the observations from the left table, whether or not a match is found for them on any of the tables specified to the right. A LEFT JOIN between two tables may be represented graphically as shown in the following Venn diagram.



Code:

```
create table Joined_ToyGA_LeftJoin as
  select toy.Code, toy.Description, tga.Gender
  from Toy as toy LEFT JOIN ToyGenderAge as tga
  on toy.Code=tga.Code
;
```

Log Results:

NOTE: Table WORK.JOINED_TOYGA_LEFTJOIN created, with 7 rows and 3 columns.

Combined File:

Code	Description	Gender
0101	Baby Doll	F
1202	Princess	F
1316	Animal Set	B
3201	Electric Truck	M
3220	Model Train	M
4300	Animal Cards	
4400	Teddy Bear	

Note that the result of a LEFT JOIN, when using a unique key and an equality condition, is similar to the “Match-Merge” with the BY statement and “if a” demonstrated above.

RIGHT JOIN

The RIGHT JOIN is identical to the LEFT JOIN except the result table includes all the observations from the right table, whether or not a match is found for them on any of the tables specified to the left. Therefore, the result is similar to the “Match-Merge” with a BY statement and “if b”. It is important to note that in SQL the LEFT and RIGHT have specific meaning but in SAS MERGE, they do not. In the latter, there is no importance to the order in which the tables appear. What’s important in the subsetting criteria is the IN= value which identifies which table is responsible for the inclusion criteria.

Code:

```
create table Joined_ToyGA_RightJoin as
  select tga.Code, toy.Description, tga.Gender
    from Toy as toy RIGHT JOIN ToyGenderAge as tga
      on toy.Code=tga.Code
```

;

Log Results:

NOTE: Table WORK.JOINED_TOYGA_RIGHTJOIN created, with 6 rows and 3 columns.

Combined File:

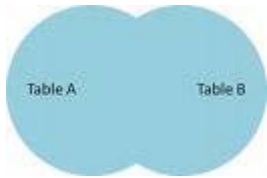
Code	Description	Gender
0101	Baby Doll	F
1202	Princess	F
1316	Animal Set	B
3201	Electric Truck	M
3220	Model Train	M
5500		M

It is important to note that:

- for this Join, the tga.Code is purposely preserved (instead of the toy.Code) because otherwise, those observations that contribute data only from the right table will only contribute values of Gender and nothing else; i.e. Code value will stay missing too
- a better option is to use the SAS COALESCE function (as shown in the next example); the COALESCE function overlays the two Code columns (returns the first value that is a SAS nonmissing value)

FULL JOIN

When a join is specified as a FULL JOIN, the result table includes all the observations from the Cartesian product of the two tables for which the sql-expression is true, plus rows from each table that do not match any row in the other table. The visual representation of the full outer join is shown in the following Venn diagram.



Code:

```
create table Joined_ToyGA_FullJoin as
  select coalesce (toy.Code, tga.Code) as Code, toy.Description, tga.Gender
  from Toy as toy FULL JOIN ToyGenderAge as tga
  on toy.Code=tga.Code
;
```

Log Results:

NOTE: Table WORK.JOINED_TOYGA_FULLJOIN created, with 8 rows and 3 columns.

Combined File:

Code	Description	Gender
0101	Baby Doll	F
1202	Princess	F
1316	Animal Set	B
3201	Electric Truck	M
3220	Model Train	M
4300	Animal Cards	
4400	Teddy Bear	
5500		M

Note that:

- a full join which uses a unique key and an equality condition is similar to the default match-merge which keeps all observations from all datasets
- a full join is limited to two source tables

ONE-TO-MANY JOIN

The following code performs an inner-join where the many toys per company are combined with the company name data. Only records that match on Company Code are kept. The one-to-many join may be performed using outer joins as well. What is significant here is that the value of Company Name from the company table is added to each matching toy record.

Code:

```
create table Joined_ToyCompany as
  (select toy.*, c.*
  from Toy as toy, Company as c
  where toy.CompanyCode=c.CompanyCode
  )
;
```

Log Results:

NOTE: Table WORK.JOINED_TOYCOMPANY created, with 7 rows and 4 columns.

Combined File:

Code	Description	CompanyCode	CompanyName
1202	Princess	1000	Kids Toys
0101	Baby Doll	1000	Kids Toys
1316	Animal Set	1000	Kids Toys
3220	Model Train	1000	Kids Toys
3201	Electric Truck	1000	Kids Toys
4300	Animal Cards	2000	More Toys
4400	Teddy Bear	2000	More Toys

MANY-TO-MANY JOIN

The many-to-many join functions in the same manner as a default join with equality condition. Since the 1st step in any SQL JOIN is building internally the Cartesian product of the tables, an automatic many-to-many join is performed which is then simply restricted by the WHERE clause.

Code:

```
create table Joined_ToyFactory as
  select toy.*, f.FactoryCode, f.FactoryState
  from Toy as toy, Factory as f
  where toy.CompanyCode=f.CompanyCode
;
```

Log Results:

NOTE: Table WORK.JOINED_TOYFACTORY created, with 21 rows and 5 columns.

Combined File:

Code	Description	CompanyCode	FactoryCode	FactoryState
1202	Princess	1000	1111	MD
1202	Princess	1000	1112	NY
1202	Princess	1000	1113	VT
0101	Baby Doll	1000	1111	MD
0101	Baby Doll	1000	1112	NY
0101	Baby Doll	1000	1113	VT
1316	Animal Set	1000	1111	MD
1316	Animal Set	1000	1112	NY
1316	Animal Set	1000	1113	VT
3220	Model Train	1000	1111	MD
3220	Model Train	1000	1112	NY
3220	Model Train	1000	1113	VT
3201	Electric Truck	1000	1111	MD
3201	Electric Truck	1000	1112	NY
3201	Electric Truck	1000	1113	VT
4300	Animal Cards	2000	2221	AZ
4300	Animal Cards	2000	2222	ME
4300	Animal Cards	2000	2223	CA
4400	Teddy Bear	2000	2221	AZ
4400	Teddy Bear	2000	2222	ME
4400	Teddy Bear	2000	2223	CA

COMPARING MERGE TO JOIN

The following tables provide a quick reference for comparing the functionality of the MERGE and JOIN.⁶

Characteristics which dominate how one should think of these two methods for combining data:

<p style="text-align: center;">MERGE</p> <p>One-to-one oriented (consequently limited but offers very tight control) Observations are read once (SAS data retained) BY variables must have same name, type and length in all source tables The order of the data sets should not matter; therefore no variables (other than the BY variables) should be in common; variables should be renamed (if needed) IN= variables are not reset by the system unless new data is read Has a missing value concept (supports special missing values so it is possible for each special missing value to represent a different meaning for numeric variables)</p>
<p style="text-align: center;">JOIN</p> <p>Cartesian product oriented SQL expects you to use WHERE or ON conditions to control the matching process (different column names and manipulations are possible) Has a single null value (foreign to SAS and cannot be used in arithmetic expressions)</p>

Similarities:

MERGE	JOIN
Match-merge of two datasets (merge with "BY Common variable(s)" but without a subsetting "IF"	Full-outer-join using the ON clause with equality as the matching condition
Merge of two datasets with "BY common variables(s)" and "If a and b" (where a indicates left dataset and b indicates right dataset)	Inner-join using a WHERE clause with equality as the matching condition
Merge with "BY common variables(s)" and "If a"	Left-join using the ON clause with equality as the matching condition
Merge with "BY common variables(s)" and "If b"	Right-join using the ON clause with equality as the matching condition

Differences:

MERGE	JOIN
Default is one-to-one "outer-join" using the given order of observations in the source datasets	Default is Cartesian Product; joining all rows from one table with all the rows in the other table
Match-merge results by default in a special case of "full-outer-join"	Match-join results by default in an inner-join
"Outer-joins" may be done on multiple source datasets	Outer-joins can be done only on two source tables
Need to sort or index the source datasets before the data step match-merge	No need to sort or index the source tables before join
Requires variable or variables that are identical on all datasets	May join on columns which are named differently ⁷

Differences (continued):

MERGE	JOIN
In a one-to-one match-merge, common variables that are not included in the match condition (i.e. are not part of the BY statement): the value from the latter dataset sometimes overwrites the value coming from the left-more dataset	Common columns are not overwritten unless this is specifically managed by the code via the use of the COALESCE function
Impossible to do a many-to-many merge due to one-to-one behavior of MERGE	Relatively easy to do a many-to-many join
Fewer advantages when working with tables stored in database servers since databases are designed for extensive matching tools	Most advantageous when working with tables stored in Database Servers because databases are designed for SQL processing
Usually more efficient when combining small datasets but too much overhead on large unsorted datasets!	More efficient than Merge when dealing with multiple large datasets (or when combining an indexed large table with a small table)

If you are trying to choose between using the Merge and the Join, take into account the items listed in comparing the two methods but also select the method that is easier for you to code and to maintain! If you are planning to run the code repeatedly in a production environment, it's a good idea to test both methods for the specific conditions of the task, and make the decision after evaluating the results and the performance indicators.

CONCLUSION

The DATA Set Merge and the PROC SQL JOIN are viewed as two alternatives for use in combining multiple data sources. This paper provides basic information about both for beginning SAS programmers. It is intended to be used as a starting point and a reference. Programmers need to do a lot of trial-and-error experimentation with both of these methods of combining data. Most importantly, programmers need to know their data well before attempting to combine observations together.

NOTES

¹ Refer to the "SAS Language: Reference", chapter 9, description of the Merge statement

² Refer to the "SAS Guide to the SQL Procedure", chapter 5, joined-table section

³ In the 1st example of the SQL Join code segment the "PROC SQL;" and "Quit;" statements are included; they are implicit in all later example code segments

⁴ Refer to the "SAS Help" for SAS 9.3, section "the SQL Procedure Joined-Table"

⁵ all Venn diagram images were taken from www.codinghorror.com by Jeff Atwood

⁶ Other excellent comparisons of the MERGE and JOIN (looking at other aspects of code and performance) are provided in papers by Kirk Lafler and Malachy J. Foley; Also, for a comprehensive description of the DATA Step MERGe please review paper by Howard Schreier.

⁷ Refer to short description by Stephen Philp

REFERENCES

Foley, Malachy J. (2005), "MERGING vs. JOINING: Comparing the DATA Step with SQL", Proceedings of the 30th Annual SAS Users Group International Conference

Lafler, Kirk Paul (2006), "A Hands-on Tour Inside the World of PROC SQL," Proceedings of the 31st Annual SAS Users Group International Conference, Software Intelligence Corporation, Spring Valley, CA, USA.

Philp, Stephen (2008), "Data Steps: SAS SQL Join," datasteps.blogspot.com/2008/04/sas-sql-join.html

Schreier, Howard (2005), "Let Your Data Power Your DATA Step: Making Effective Use of the SET, MERGE, UPDATE, and MODIFY Statements", Proceedings of the 30th Annual SAS Users Group International Conference

SAS Institute Inc. (1990), SAS Language: Reference, Version 6, 1st Edition, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1989), SAS Guide to the SQL Procedure: Usage and Reference, Version 6, 1st Edition, Cary, NC: SAS Institute Inc.

DISCLAIMER

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

ACKNOWLEDGEMENTS

I would like to thank Ian Whitlock who reviewed my paper, provided comments, and in his usual fashion made this too a true learning experience for me.

CONTACT INFORMATION

If you have any questions or comments about the paper, you can reach the author at:

Dalia Kahane, Ph.D.
Westat
1650 Research Blvd.
Rockville, MD 20850
Kahaned1@Westat.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.