RUCHITA TRIVEDI

rtrivedi4@horizon.csueastay.edu

Abstract

The different methods and the reasons for choosing a particular method of joining are contrasted and compared  
 Potential problems and limitations when joining data sets are also discussed.

Merges and Joins

Comparing Proc SQL and DATA step

Friday, July 22, 2016

1:19 PM

Contents

[1.1 Abstract 1](#_Toc457690305)

[1.2 Vertical Joining 2](#_Toc457690306)

[1.2.1 Using PROC DATASETS and APPEND 2](#_Toc457690307)

[1.2.2 Vertical Joining with UNION Corresponding 4](#_Toc457690308)

[1.3 Horizontal Joining 4](#_Toc457690309)

[1.3.1 The Inner Join 5](#_Toc457690310)

[1.3.2 The Left Join 6](#_Toc457690311)

[1.3.3 The Right Join 7](#_Toc457690312)

[1.3.4 The Full Join 7](#_Toc457690313)

[1.3.5 Using the COALESCE function and determining the source data set 8](#_Toc457690314)

[1.4 The Data Step Merge 9](#_Toc457690315)

[1.4.1 Sort Ordering and BY Variables 9](#_Toc457690316)

[1.4.2 Merges and IN Variables 10](#_Toc457690317)

[1.5 Comparing a DATA Step Merge with a PROC SQL Join. 10](#_Toc457690318)

[1.6 PROC SQL Joins vs. DATA Step MERGEs 13](#_Toc457690319)

[1.7 More than Two Data Sets at a Time 13](#_Toc457690320)

## Abstract

I have implemented here methods of joining SAS data sets. The different methods and the reasons for choosing a particular method of joining are contrasted and compared. There are two basic types of join, vertical, and horizontal. Vertical joining is appending one data set to another, whereas horizontal joining is using one or more key variables to combine different observations.

First , I created the datasets in a text editor and then opened it via Microsoft Excel, then the excel spreadsheet was imported into SAS via following code,

**PROC** **IMPORT** OUT= work.dosing DATAFILE= "C:\Users\i037805\Google Drive\Summers 2016\SAS Advanced\SAS\_PROJECT1.xlsx"

DBMS=xlsx REPLACE;

SHEET="DOSING";

GETNAMES=YES;

**RUN**;

**PROC** **IMPORT** OUT= work.efficacy DATAFILE= "C:\Users\i037805\Google Drive\Summers 2016\SAS Advanced\SAS\_PROJECT1.xlsx"

DBMS=xlsx REPLACE;

SHEET="EFFICACY";

GETNAMES=YES;

**RUN**;

## Vertical Joining

* Once the data sets are combined at least one of the variables should, in practice, be able to identify which of the source data sets any given observation originated from.

* Another issue may be the sort order. In this example there is no need to sort the resulting data set if the source data sets are in date order.

* When vertically joining data sets, is the issue vertical compatibility. This is whether the corresponding variables in each data set have the same attributes, and if there are any variables which are present in one data set but not in the other

### Using PROC DATASETS and APPEND

* One method of vertical joining is to use the utility procedure PROC DATASETS with the APPEND statement. More than two data sets many be joined in this way, but all of the data sets should be vertically compatible.
* However, vertical incompatibility may be overridden by using the FORCE option. When this option is used, variables which are absent in one data set are created with the same attributes in the resulting data set, but the values are missing in each observation which originated from the data set without those variables.
* Where variable lengths are different the shorter length values are right padded with spaces to equal the longer length. Where data types are different the numeric type is made character. If labels are different the label from the latest data set is used.
* If the FORCE option is not specified and any of the data sets are not completely vertically compatible applicable NOTES and WARNINGS are written to the log file.
* If a variable is present in the DATA data set but is absent in the BASE data set the appending is not done.

**proc** **append** base=dosing data=efficacy force ;

**run**;

**proc** **print** data=dosing ;

**run**;

WARNING: Variable EFFIC\_ID was not found on BASE file. The variable will not be added to the

BASE file.

WARNING: Variable VISIT was not found on BASE file. The variable will not be added to the BASE

file.

WARNING: Variable SCORE was not found on BASE file. The variable will not be added to the BASE

file.

WARNING: Variable MED\_CODE was not found on DATA file.

WARNING: Variable DOSE\_ID was not found on DATA file.

WARNING: Variable DOSE\_AMT was not found on DATA file.

WARNING: Variable DOSE\_FREQ was not found on DATA file.

NOTE: FORCE is specified, so dropping/truncating will occur

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Obs | OBS | PATIENT | MED\_CODE | DOSE\_ID | DOSE\_AMT | DOSE\_FREQ |
| 1 | 1 | 1001 | A | 1 | 2 | 2 |
| 2 | 2 | 1003 | A | 2 | 1 | 2 |
| 3 | 3 | 1004 | A | 3 | 1 | 2 |
| 4 | 4 | 1004 | B | 4 | 4 | 2 |
| 5 | 5 | 1006 | B | 5 | 2 | 2 |
| 6 | 6 | 1007 | A | 6 | 2 | 1 |
| 7 | 7 | 1008 | A | 7 | 1 | 2 |
| 8 | 8 | 1009 | A | 8 | 2 | 2 |
| 9 | 1 | 1001 |  | . | . | . |
| 10 | 2 | 1002 |  | . | . | . |
| 11 | 3 | 1004 |  | . | . | . |
| 12 | 4 | 1004 |  | . | . | . |
| 13 | 5 | 1005 |  | . | . | . |
| 14 | 6 | 1009 |  | . | . | . |
| 15 | 1 | 1001 |  | . | . | . |
| 16 | 2 | 1002 |  | . | . | . |
| 17 | 3 | 1004 |  | . | . |  |

### Vertical Joining with UNION Corresponding

* In PROC SQL two or more data sets may be vertically joined used UNION CORRESPONDING ALL. (If the ‘ALL’ is omitted only one of any duplicate observations are kept). This is analogous to APPEND in PROC DATASETS but if the data sets to be joined are not vertically compatible only variables common to both data sets are placed in the resulting table. Check the example below using PROC SQL with UNION CORRESPONDING ALL. (Generally this method is less efficient than using PROC DATASETS with APPEND.)

**PROC** **SQL** outobs=**15**;

CREATE TABLE DATA99 AS

SELECT \* FROM sasusers.admit

UNION CORRESPONDING ALL

SELECT \* FROM sasusers.admitjune;

**QUIT**;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Obs | ID | Name | Sex | Age | Date | Height | Weight | ActLevel | Fee |
| 1 | 2458 | Murray, W | M | 27 | 1 | 72 | 168 | HIGH | 85.20 |
| 2 | 2462 | Almers, C | F | 34 | 3 | 66 | 152 | HIGH | 124.80 |
| 3 | 2501 | Bonaventure, T | F | 31 | 17 | 61 | 123 | LOW | 149.75 |
| 4 | 2523 | Johnson, R | F | 43 | 31 | 63 | 137 | MOD | 149.75 |
| 5 | 2539 | LaMance, K | M | 51 | 4 | 71 | 158 | LOW | 124.80 |
| 6 | 2544 | Jones, M | M | 29 | 6 | 76 | 193 | HIGH | 124.80 |
| 7 | 2552 | Reberson, P | F | 32 | 9 | 67 | 151 | MOD | 149.75 |
| 8 | 2555 | King, E | M | 35 | 13 | 70 | 173 | MOD | 149.75 |
| 9 | 2563 | Pitts, D | M | 34 | 22 | 73 | 154 | LOW | 124.80 |
| 10 | 2568 | Eberhardt, S | F | 49 | 27 | 64 | 172 | LOW | 124.80 |
| 11 | 2571 | Nunnelly, A | F | 44 | 19 | 66 | 140 | HIGH | 149.75 |
| 12 | 2572 | Oberon, M | F | 28 | 17 | 62 | 118 | LOW | 85.20 |
| 13 | 2574 | Peterson, V | M | 30 | 6 | 69 | 147 | MOD | 149.75 |
| 14 | 2575 | Quigley, M | F | 40 | 8 | 69 | 163 | HIGH | 124.80 |
| 15 | 2578 | Cameron, L | M | 47 | 5 | 72 | 173 | MOD | 124.80 |

## Horizontal Joining

* There are four basic types of horizontal join, the inner join, left join, right join, and full join. All such joins are Cartesian products made on specified key variables.
* If there are duplicate matches in either or both tables all of the matching observations are selected, for example if there are two equal key values in each input data set there will be four output observations created.

## 

### The Inner Join

* The inner join creates observations from data items selected from either input data set where the key values match in both tables. If the key values match in only one table an output observation is not created.
* An ‘inner’ join is a logical AND of the two tables and is therefore commutative, that is the tables can be joined in either order. The following PROC SQL segment creates a table named INNER1 as the inner join between DOSING and EFFICACY on PATIENT. A point to note is that where there are duplicate key values a complete Cartesian product is produced, in this example this happens with Patient 1004. The ‘A’ and ‘B’ characters preceding the variable names are aliases for each of the data set names and the ORDER BY clause sorts the resulting data set in ascending order of PATIENT and MEDCODE.

\*Table INNER1: An INNER JOIN on PATIENT

between DOSING and EFFICACY.;

**PROC** **SQL**;

CREATE TABLE INNER1 AS

SELECT A.\*, B.EFFIC\_ID, B.VISIT,

B.SCORE

FROM DOSING A, EFFICACY B

WHERE A.PATIENT=B.PATIENT

ORDER BY PATIENT;

**QUIT**;

**proc** **print** data=inner1;

**run**;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Obs | OBS | PATIENT | MED\_CODE | DOSE\_ID | DOSE\_AMT | DOSE\_FREQ | EFFIC\_ID | VISIT | SCORE |
| 1 | 1 | 1001 | A | 1 | 2 | 2 | 1 | 1 | 4 |
| 2 | 3 | 1004 | A | 3 | 1 | 2 | 4 | 2 | 1 |
| 3 | 4 | 1004 | B | 4 | 4 | 2 | 4 | 2 | 1 |
| 4 | 3 | 1004 | A | 3 | 1 | 2 | 3 | 1 | 2 |
| 5 | 4 | 1004 | B | 4 | 4 | 2 | 3 | 1 | 2 |
| 6 | 8 | 1009 | A | 8 | 2 | 2 | 6 | 1 | 5 |

* If the WHERE clause were omitted the complete Cartesian product of every observation would be selected, producing 48 (6\*8) observations, hence at least one key variable must be specified when performing any type of horizontal join .

### The Left Join

* A ‘left’ join selects items from all the observations in the first (left) data set regardless of their key values but only observations with matching key values from the second (right) data set. Variables from the second data set where the observation key values do not match the join criteria are assigned missing values in the output data set.

* As with the inner join, where there are multiple key values a complete Cartesian product is created.
* Also, a left join is not commutative, reversing the Coders' Corner order of the source data sets would produce a totally different result.

\*Table LEFT: A LEFT JOIN on PATIENT

between DOSING ('left' data set) and

EFFICACY ('right' data set).;

**PROC** **SQL**;

CREATE TABLE LEFT1 AS

SELECT A.\*, B.EFFIC\_ID, B.VISIT,

B.SCORE

FROM DOSING A LEFT JOIN EFFICACY B

ON A.PATIENT = B.PATIENT

ORDER BY PATIENT;

**QUIT**;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Obs | OBS | PATIENT | MED\_CODE | DOSE\_ID | DOSE\_AMT | DOSE\_FREQ | EFFIC\_ID | VISIT | SCORE |
| 1 | 1 | 1001 | A | 1 | 2 | 2 | 1 | 1 | 4 |
| 2 | 2 | 1003 | A | 2 | 1 | 2 | . | . | . |
| 3 | 3 | 1004 | A | 3 | 1 | 2 | 3 | 1 | 2 |
| 4 | 4 | 1004 | B | 4 | 4 | 2 | 3 | 1 | 2 |
| 5 | 3 | 1004 | A | 3 | 1 | 2 | 4 | 2 | 1 |
| 6 | 4 | 1004 | B | 4 | 4 | 2 | 4 | 2 | 1 |
| 7 | 5 | 1006 | B | 5 | 2 | 2 | . | . | . |
| 8 | 6 | 1007 | A | 6 | 2 | 1 | . | . | . |
| 9 | 7 | 1008 | A | 7 | 1 | 2 | . | . | . |
| 10 | 8 | 1009 | A | 8 | 2 | 2 | 6 | 1 | 5 |

### The Right Join

* A ‘right’ join is where all the observations are selected from the second data set and where observations do not match in the first data set the key values themselves are assigned missing values.
* A right join, like a left join, is not commutative neither is a right join the same as reversing the order of the two data sets in a left join. (The ORDER BY PATIENT clause causes the missing values to float to the topmost observations.).

\*Table RIGHT1: A RIGHT JOIN on PATIENT between DOSING('left' data set) and EFFICACY('right' data set). ;

**PROC** **SQL**;

CREATE TABLE RIGHT1 AS

SELECT A.\*, B.EFFIC\_ID, B.VISIT, B.SCORE

FROM DOSING A RIGHT JOIN EFFICACY

B ON A.PATIENT = B.PATIENT

ORDER BY PATIENT;

**QUIT**;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Obs | OBS | PATIENT | MED\_CODE | DOSE\_ID | DOSE\_AMT | DOSE\_FREQ | EFFIC\_ID | VISIT | SCORE |
| 1 | . | . |  | . | . | . | 2 | 1 | 5 |
| 2 | . | . |  | . | . | . | 5 | 1 | 2 |
| 3 | 1 | 1001 | A | 1 | 2 | 2 | 1 | 1 | 4 |
| 4 | 3 | 1004 | A | 3 | 1 | 2 | 3 | 1 | 2 |
| 5 | 4 | 1004 | B | 4 | 4 | 2 | 4 | 2 | 1 |
| 6 | 4 | 1004 | B | 4 | 4 | 2 | 3 | 1 | 2 |
| 7 | 3 | 1004 | A | 3 | 1 | 2 | 4 | 2 | 1 |
| 8 | 8 | 1009 | A | 8 | 2 | 2 | 6 | 1 | 5 |

### The Full Join

* The ‘full’ join selects all the observations from both data sets but there are missing values where the key value in each observation is found in one table only.
* A ‘full’ join is the logical OR of the two tables, but is not commutative because missing key values are assigned to those non-matching observations in the second data set.

\*Table FULL1: A FULL JOIN on PATIENT between DOSING('left' data set) and EFFICACY('right' data set).;

**PROC** **SQL**;

CREATE TABLE FULL1 AS

SELECT A.\*, B.EFFIC\_ID, B.VISIT, B.SCORE

FROM DOSING A FULL JOIN EFFICACY B

ON A.PATIENT = B.PATIENT

ORDER BY PATIENT;

**QUIT**;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Obs | OBS | PATIENT | MED\_CODE | DOSE\_ID | DOSE\_AMT | DOSE\_FREQ | EFFIC\_ID | VISIT | SCORE |
| 1 | . | . |  | . | . | . | 2 | 1 | 5 |
| 2 | . | . |  | . | . | . | 5 | 1 | 2 |
| 3 | 1 | 1001 | A | 1 | 2 | 2 | 1 | 1 | 4 |
| 4 | 2 | 1003 | A | 2 | 1 | 2 | . | . | . |
| 5 | 4 | 1004 | B | 4 | 4 | 2 | 4 | 2 | 1 |
| 6 | 4 | 1004 | B | 4 | 4 | 2 | 3 | 1 | 2 |
| 7 | 3 | 1004 | A | 3 | 1 | 2 | 4 | 2 | 1 |
| 8 | 3 | 1004 | A | 3 | 1 | 2 | 3 | 1 | 2 |
| 9 | 5 | 1006 | B | 5 | 2 | 2 | . | . | . |
| 10 | 6 | 1007 | A | 6 | 2 | 1 | . | . | . |
| 11 | 7 | 1008 | A | 7 | 1 | 2 | . | . | . |
| 12 | 8 | 1009 | A | 8 | 2 | 2 | 6 | 1 | 5 |

### Using the COALESCE function and determining the source data set

* When performing left, right, or full joins where observations do not have a key variable match, nonkey values are assigned missing values. Sometimes there is a need to substitute missing values with other data, either hard coded values or different items from either data set. One way to do this is with a CASE construct, but the COALESCE function is provided specifically for this purpose.
* In this example the variable ADJSCORE (Adjusted Score) contains the Coders' Corner value of SCORE in the observations that match, but where there is no match and SCORE is missing AJDSCORE is assigned the value zero.

If a value of SCORE was missing in a matching observation the value of ADJSCORE would also be set to zero.

* COALESCE may be used with either a character or numeric data type, but the second argument must be of that same data type.

\*COALESCE function example;

**PROC** **SQL**;

CREATE TABLE LEFT1A(DROP=DOSE\_ID) AS

SELECT A.\*, B.VISIT, B.SCORE,

COALESCE(B.SCORE,**0**) AS ADJSCORE,

CASE when (A.PATIENT=B.PATIENT) THEN "Match"

when (A.PATIENT^=B.PATIENT) THEN "Miss"

ELSE " "

END AS INVAR LENGTH=**5**

FROM DOSING A LEFT JOIN

EFFICACY B

ON A.PATIENT = B.PATIENT

ORDER BY A.PATIENT;

**QUIT**;

## The Data Step Merge

A DATA step MERGE joins two data sets inside a DATA step to produce a resulting data set. A DATA step MERGE differs from a PROC SQL in two important ways.

(1) The sort order of the key variables is important because matching is performed sequentially on an observation-by-observation basis.

(2) Cartesian products are not evaluated, the merge is performed sequentially by input observation and then the resulting observation is placed in the Program Data Vector (PDV). When there are more key matching observations in one data set than the other, the non-key data from the last of the fewer matching observations is retained in the PDV for each remaining match of the more numerous observations. This is called the *implied retain.*

## 

### Sort Ordering and BY Variables

* BY variables are the ‘key’ variables on which the merge matching is to be performed. They must have been sorted in either ascending (default) order or descending order using PROC SORT or an ORDER BY statement in a prior PROC SQL.
* The DATA step performing the merge must contain an applicable BY statement, matching the BY statement in the preceding PROC SORT or PROC SQL ORDER BY statement of each corresponding set. If a key variable has been sorted in descending order that variable must be specified as DESCENDING in the BY statement of the merge.
* If the BY values are unique in both data sets the merge is a ‘one to one’ merge. If there are observations with duplicate BY values in one data set and only one matching observation in the other data set that single observation is joined with all of the matching BY variables in the first data set because of the implied retain. This is a ‘one to many’ merge .
* This aspect of merging is commutative, in that performing a ‘many to one’ merge with the order of the data sets reversed produces the same result .
* A ‘Many to many’ merge is where there are corresponding duplicate BY variables in both data sets.

Such a merge does not result in a Cartesian product because the observations are joined in sequence where they match. (See what happens to Patient 1004 in the examples listed below). When ‘many to many’ situations are encountered the following message is written to the log file: NOTE: MERGE statement has more than one data set with repeats of BY values.

### Merges and IN Variables

* An IN variable is a Boolean flag, which applies to an input data set. The IN variable is set to ‘true’ (1) or ‘false’ (0) depending on whether that data set contributed data to the current PDV contents or not.
* When two data sets are being merged at least one of the IN variables must be ‘true’, both IN variables are ‘true’ if the BY variables match. IN variables are most useful for testing for such matches, as shown in the examples listed below.

## Comparing a DATA Step Merge with a PROC SQL Join.

* This first example performs a merge using the key variable PATIENT and outputs an observation when the patient numbers are equal, and hence the IN variables are both true.
* This merge corresponds to the PROC SQL inner join, but with one important difference, no Cartesian products are generated because the merging process is sequential by observation. Hence, the data set INNER2 has only two observations for Patient 1004 instead of the four in INNER1. (For this reason ‘many to many’ merges should be avoided in practice).
* Another point to note is that when testing IN variables an IF statement must be used, a WHERE clause will not work because the IN variables are calculated within the DATA Step and are not from the source data.

\*Table INNER2: A MERGE between DOSING and

EFFICACY where equal values of PATIENT

occur in both input data sets.;

**DATA** INNER2;

MERGE DOSING(IN=A) EFFICACY(IN=B);

BY PATIENT;

IF (A=B);

**RUN**;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Obs | OBS | PATIENT | MED\_CODE | DOSE\_ID | DOSE\_AMT | DOSE\_FREQ | EFFIC\_ID | VISIT | SCORE |
| 1 | . | . |  | . | . | . | 2 | 1 | 5 |
| 2 | . | . |  | . | . | . | 5 | 1 | 2 |
| 3 | 1 | 1001 | A | 1 | 2 | 2 | 1 | 1 | 4 |
| 4 | 3 | 1004 | A | 3 | 1 | 2 | 3 | 1 | 2 |
| 5 | 4 | 1004 | B | 4 | 4 | 2 | 4 | 2 | 1 |
| 6 | 4 | 1004 | B | 4 | 4 | 2 | 3 | 1 | 2 |
| 7 | 3 | 1004 | A | 3 | 1 | 2 | 4 | 2 | 1 |
| 8 | 8 | 1009 | A | 8 | 2 | 2 | 6 | 1 | 5 |

This second example performs the same merge as the first example but only outputs an observation whenever the DOSING data set contributes a value to the PDV. Observations with values of PATIENT which are present only in EFFICACY and not in DOSING are not output and missing values are substituted in the non-key variables not from DOSING. This merge corresponds to the PROC SQL left join, but without the Cartesian product of duplicate key values (Patient 1004).

\*Table LEFT2: All values of PATIENT are taken from DOSING and only matching values from EFFICACY.;

**DATA** LEFT2;

MERGE DOSING(IN=A) EFFICACY;

BY PATIENT;

IF A;

**RUN**;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Obs | OBS | PATIENT | MED\_CODE | DOSE\_ID | DOSE\_AMT | DOSE\_FREQ | EFFIC\_ID | VISIT | SCORE |
| 1 | 1 | 1001 | A | 1 | 2 | 2 | 1 | 1 | 4 |
| 2 | 2 | 1003 | A | 2 | 1 | 2 | . | . | . |
| 3 | 3 | 1004 | A | 3 | 1 | 2 | 3 | 1 | 2 |
| 4 | 4 | 1004 | B | 4 | 4 | 2 | 4 | 2 | 1 |
| 5 | 5 | 1006 | B | 5 | 2 | 2 | . | . | . |
| 6 | 6 | 1007 | A | 6 | 2 | 1 | . | . | . |
| 7 | 7 | 1008 | A | 7 | 1 | 2 | . | . | . |
| 8 | 6 | 1009 | A | 8 | 2 | 2 | 6 | 1 | 5 |

In this third example only observations in the PDV from the EFFICACY data set are output. Values in DOSING which do not match, including the key value (Patient) are output as missing values. This corresponds to a right join in PROC SQL. Note that both the order in which the data sets are specified and which IN variable is tested are important. In this example ‘merge dosing efficacy(in=b)’ is not the same as ‘merge efficacy(in=b) dosing’ or ‘merge dosing(in=b) efficacy’.

\*Table RIGHT2: A MERGE between DOSING and EFFICACY where all values of PATIENT are taken from EFFICACY and

only matching values from DOSING.;

**DATA** RIGHT2;

MERGE DOSING EFFICACY(IN=B);

BY PATIENT;

IF B;

**RUN**;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Obs | OBS | PATIENT | MED\_CODE | DOSE\_ID | DOSE\_AMT | DOSE\_FREQ | EFFIC\_ID | VISIT | SCORE |
| 1 | 1 | 1001 | A | 1 | 2 | 2 | 1 | 1 | 4 |
| 2 | 2 | 1002 |  | . | . | . | 2 | 1 | 5 |
| 3 | 3 | 1004 | A | 3 | 1 | 2 | 3 | 1 | 2 |
| 4 | 4 | 1004 | B | 4 | 4 | 2 | 4 | 2 | 1 |
| 5 | 5 | 1005 |  | . | . | . | 5 | 1 | 2 |
| 6 | 6 | 1009 | A | 8 | 2 | 2 | 6 | 1 | 5 |

In this next and final example all observations in the PDV are output regardless of a match or not, hence IN variables are not needed. This corresponds to the PROC SQL full join.

Table FULL2: A MERGE between DOSING and EFFICACY where all values of PATIENT are taken from both data sets regardless of whether they match or not.

\*Table FULL2: A MERGE between DOSING and EFFICACY where all values of PATIENT are taken from both data sets

regardless of whether they match or not.;

**DATA** FULL2;

MERGE DOSING EFFICACY;

BY PATIENT;

**RUN**;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Obs | OBS | PATIENT | MED\_CODE | DOSE\_ID | DOSE\_AMT | DOSE\_FREQ | EFFIC\_ID | VISIT | SCORE |
| 1 | 1 | 1001 | A | 1 | 2 | 2 | 1 | 1 | 4 |
| 2 | 2 | 1002 |  | . | . | . | 2 | 1 | 5 |
| 3 | 2 | 1003 | A | 2 | 1 | 2 | . | . | . |
| 4 | 3 | 1004 | A | 3 | 1 | 2 | 3 | 1 | 2 |
| 5 | 4 | 1004 | B | 4 | 4 | 2 | 4 | 2 | 1 |
| 6 | 5 | 1005 |  | . | . | . | 5 | 1 | 2 |
| 7 | 5 | 1006 | B | 5 | 2 | 2 | . | . | . |
| 8 | 6 | 1007 | A | 6 | 2 | 1 | . | . | . |
| 9 | 7 | 1008 | A | 7 | 1 | 2 | . | . | . |
| 10 | 6 | 1009 | A | 8 | 2 | 2 | 6 | 1 | 5 |

## PROC SQL Joins vs. DATA Step MERGEs

There are several important differences between PROC SQL joins and DATA Step MERGES.

* A SQL join creates a Cartesian product out of multiple occurrences of key values. When there are more matching key observations in one data set than in the other (‘one to many’ or ‘many to one’ merges) the contents of the last matching observation from the data set with fewer matches are retained in the PDV. This is the implied retain. The result is the remaining non-key values from the data set with fewer matches appear in the corresponding excess output observations. The implied retain does not occur in the above examples with Patient 1004 because there are the same number of observations for this patient in both data sets. If one of the Patient 1004 observations is deleted from either DOSING or EFFICACY an implied retain would then take place with the second observation.
* A ‘many-to-many’ MERGE does not produce a complete Cartesian product with duplicate key values in both data sets (‘many to many’). A note indicating repeating BY values is written to the log file. ‘Many-to-many’ merges are also expensive in terms of processing time and resources. Hence ‘many-to-many’ merges should be avoided.
* When using a SQL join the observations in either data set do not have to be sorted, a DATA step MERGE requires the key variables (BY variables) to have been sorted in a corresponding order, either using PROC SORT, or an ORDER BY in a prior PROC SQL.
* A PROC SQL join can use aliases to identify the data set, which is contributing a particular variable.
* A DATA Step MERGE can be used with a logical IN variable to identify which data set contributed the key values to the current PDV contents. Specific non-key variable names must be unique to each data set since their source data set cannot be identified with an alias.
* Data set options such as KEEP, DROP, and RENAME may be used in both PROC SQL or in a DATA step. To subset data from either input data set a WHERE statement may be used in parenthesis after the data set name (This is more efficient than using the WHERE statement in a CREATE TABLE block or a DATA step.)
* Runtime benchmark tests show that a PROC SQL join is faster than a DATA Step MERGE. Using indexes improves performance still further, but PROC SQL indexes and DATA Step indexes are implemented differently internally by the SAS system and may conflict with each other curtailing performance.

## More than Two Data Sets at a Time

In any of the examples shown above more than two data sets may be joined. However, due to the increased complexity, such as possible large Cartesian products when performing PROC SQL joins, joining more than two data sets at a time should be avoided. When the requirement is to join many data sets a ‘result’ data set should be created from a join of the first two and then this ‘result’ data set be joined with each other data set one at a time.