Some Uses of PROC SQL

Noory Kim

05 May 2016

cros nt

The Center of Excellence for Clinical Trial Data

Outline

A. Syntax

B. Coding style tips

C. Using PROC SQL for Tables
   - Counting subjects
   - Tallying categorical variables
   - Summarizing continuous variables

D. Using PROC SQL for Datasets
   - Types of (horizontal) joins
   - Many-to-many joins

A. Syntax

SQL Invocation & Query statement: General form

PROC SQL;

SELECT *column-1<,...column-n>*

FROM *table-1|view-1<,...table-n|view-n>*

<WHERE *expression*>

<GROUP BY *column-1<, ... column-n>>*

<ORDER BY *column-1<,... column-n>>*;

where

PROC SQL
invokes the SQL procedure

SELECT
specifies the column(s) to be selected

FROM
specifies the table(s) to be queried

WHERE
subsets the data based on one or more conditions

GROUP BY
classifies the data into groups based on the specified column(s)

ORDER BY
sorts the rows that the query returns by the value(s) of the specified column(s).

SQL Invocation & Query statement: Example

```
proc sql;
    select usubjid,
           trtpn
        from sasdata.adsl
        where ittfln = 1
    ;
    select *
        from sasdata.adae
    ;
quit;
```

Items listed within each clause are separated by commas. But the clauses themselves are not separated by punctuation.

* includes all variables in the dataset.

QUIT ends the PROC SQL invocation.
Note a single SQL invocation can have multiple query statements, each ending with a semi-colon.
This invocation has two query statements.

Output a query into a table / data set

CREATE TABLE *table-name* **AS**

SELECT *column-1<, ... column-n>*

FROM *table-1 | view-1<, ... table-n | view-n>*

<optional query clauses>;

where

table-name
specifies the name of the table to be created.

SELECT
specifies the column(s) that will appear in the table.

FROM
specifies the table(s) or view(s) to be queried.

optional query clauses
are used to refine the query further and include WHERE, GROUP BY, HAVING, and ORDER BY.

Output a query into a table / data set: Example

```
proc sql;
    create table adsl as
        select usubjid,
               trtpn
        from sasdata.adsl
        where ittfln = 1
    ;
    create table adae as
        select *
        from sasdata.adae
    ;
quit;
```

B. Coding Style Tips

Coding Style Tips

Using one line per variable increases legibility.

```
proc sql;
    select trtpn, sum(v1 = 'Normal') as w11,  sum(v1 not in ('Normal'
           ' ') ) as w12, sum(v2 ne ' ') as w2
    from adef;
quit;

proc sql;
    select trtpn,
           sum(v1 = 'Normal') as w11,
           sum(v1 not in ('Normal' ' ') ) as w12,
           sum(v2 ne ' ') as w2
    from adef;
quit;
```

Coding Style Tips

Indenting helps distinguish between multiple statements, clauses, and variables.

```
proc sql;
    create table summ1 as
        select trtpn,
                put(count(unique usubjid), 8.) as nsubj
        from adsl
     ;
    create table summ1 as
        select trtpn,
                put(count(unique usubjid), 8.) as nsubj_ae
        from adae
    ;
quit;
```

C. Using PROC SQL for Tables

Counting Subjects to Display "(N=##)"

```
proc sql noprint;
    select trtpn,
           count(unique usubjid) into :n1-:n2
    from ADSL
    group by trtpn
    order by trtpn
  ;
quit;
proc report …;
    …
    define column2 / "Treatment 1*(N=&n1)";
    define column3 / "Treatment 2*(N=&n2)";
run;
```

The into clause creates the macro variables &n1 and &n2.

Summary	Treatment 1 (N=36)	Treatment 1 (N=40)

Tallying categorical variables

```
proc sql;
   create table summary_table as
      select trtpn,
             sum(SAFFLN = 1) as nsafety  label='Number of Safety Subjects'
      from ADSL
      group by trtpn
      order by trtpn
   ;
quit;

proc transpose data=summary_table out=transposed_table;
      id trtpn;
      var nsafety;
run;
```

Summary	Treatment 1 (N=36)	Treatment 1 (N=40)
Number of Safety Subjects	30	35

Summarizing continuous variables

```
proc sql;
   create table summary_table as
      select trtpn,
             mean(BMIBL) as BMIBL_mean label='Mean Baseline BMI'
      from ADSL
      group by trtpn
      order by trtpn
   ;
quit;

proc transpose data=summary_table out=transposed_table;
     id trtpn;
     var BMIBL_mean;
run;
```

Summary	Treatment 1 (N=36)	Treatment 1 (N=40)
Mean Baseline BMI	20.3	21.4

D. Using PROC SQL for Datasets

Inner Join

SELECT *column-1<,...column-n>*

FROM *table-1 | view-1, table-2 | view-2<,...table-n | view-n>*

WHERE *join-condition(s)*

<AND*other subsetting condition(s)*>

<other clauses>;

where

join-condition(s)
refers to one or more expressions that specify the column or columns on which the tables are to be joined.

other subsetting condition(s)
refers to optional expressions that are used to subset rows in the query results.

<other clauses>
refers to optional PROC SQL clauses.

Inner Join

```
proc sql;
    select *
        from one as w,
             two as t
        where w.x = t.x
    ;
quit;
```

For convenience you set define an alias for each dataset using “as”.

One

X	A
1	a
2	b
4	d

Two

X	B
2	x
3	y
5	v

X	A	X	B
2	b	2	x

Note that the result shows the key variable **X** from both input datasets.

Inner Join

```
proc sql;
    select coalesce(w.x, t.x) as x,
                a,
                b
        from one as w,
                two as t
        where w.x = t.x
    ;
quit;
```

You can use the COALESCE() function to combine the key variable column.

One

X	A
1	a
2	b
4	d

Two

X	B
2	x
3	y
5	v

X	A	B
2	b	x

Outer Join

General form, SELECT statement for outer join:

SELECT *column-1<,...column-n>*

FROM *table-1 | view-1*

LEFT JOIN | RIGHT JOIN | FULL JOIN,

table-2 | view-2

ON *join-condition(s)*

<other clauses>;

not 'OUTER'

where

LEFT JOIN, RIGHT JOIN, FULL JOIN
are keywords that specify the type of outer join.

ON
specifies *join-condition(s)*, which are expression(s) that specify the column or columns on which the tables are to be joined.

<other clauses>
refers to optional PROC SQL clauses.

Left Outer Join

```
proc sql;
    select w.x,
           a,
           b
      from one as w
           left join
           two as t
      on w.x = t.x
    ;
quit;
```

Select the key variable from the input dataset listed **first** in the FROM clause.

One

X	A
1	a
2	b
4	d

Two

X	B
2	x
3	y
5	v

X	A	B
1	a	
2	b	x
4	d	

Right Outer Join

```
proc sql;
    select t.x,
           a,
           b
      from one as w
           right join
           two as t
      on w.x = t.x
    ;
quit;
```

Select the key variable from the input dataset listed **second** in the FROM clause.

One

X	A
1	a
2	b
4	d

Two

X	B
2	x
3	y
5	v

X	A	B
2	b	x
3		y
5		v

Full Outer Join

```
proc sql;
    select *
        from one as w
                full join
                two as t
        on w.x = t.x
    ;
quit;
```

Note that the FROM clause has no commas.

Outer joins have an ON clause instead of a WHERE clause.

Full Outer Join

X	A
1	a
2	b
4	d

X	B
2	x
3	y
5	v

DATA Step Match-Merge Output

Table Merged

X	A	B
1	a	
2	b	x
3		y
4	d	
5		v

PROC SQL Full Outer Join Output

Table Merged

X	A	B
.		y
.		v
1	a	
2	b	x
4	d	

PROC SQL may result in missing values for the key variable that are not shared by both datasets.

You can avoid this problem by using the COALESCE() function.

Full Outer Join

```
proc sql;
    select coalesce(w.x, t.x) as x,
           a,
           b
      from one as w
           full join
           two as t
      on w.x = t.x
    ;
quit;
```

X	A
1	a
2	b
4	d

X	B
2	x
3	y
5	v

X	A	B
1	a	
2	b	x
3		y
4	d	
5		v

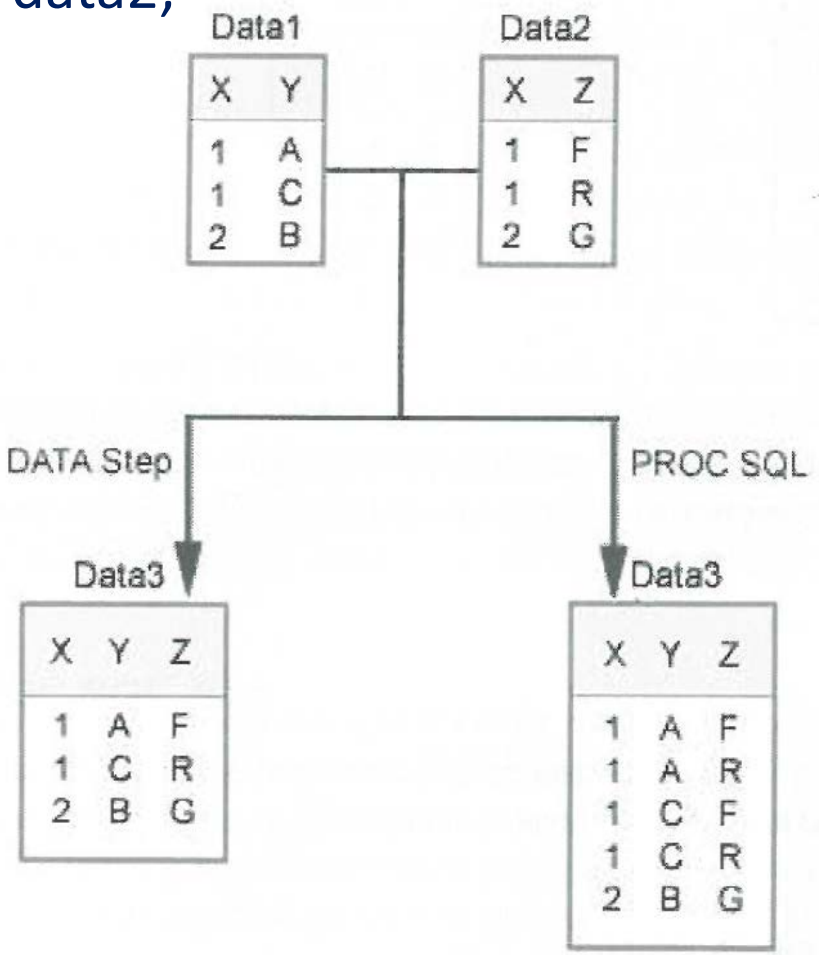
You can avoid missing values for the key variable by using the COALESCE() function.

Types of situations regarding key variables

- One-to-one
  - Both datasets have a key variable with a unique value for each row.
- One-to-many
  - One dataset has a key variable with a unique value for each row.
- Many-to-many
  - Neither dataset has a key variable with a unique value for each row.

Many-to-Many Joins: DATA MERGE vs PROC SQL

```
data data3;
   merge data1 data2;
   by x;
run;
```



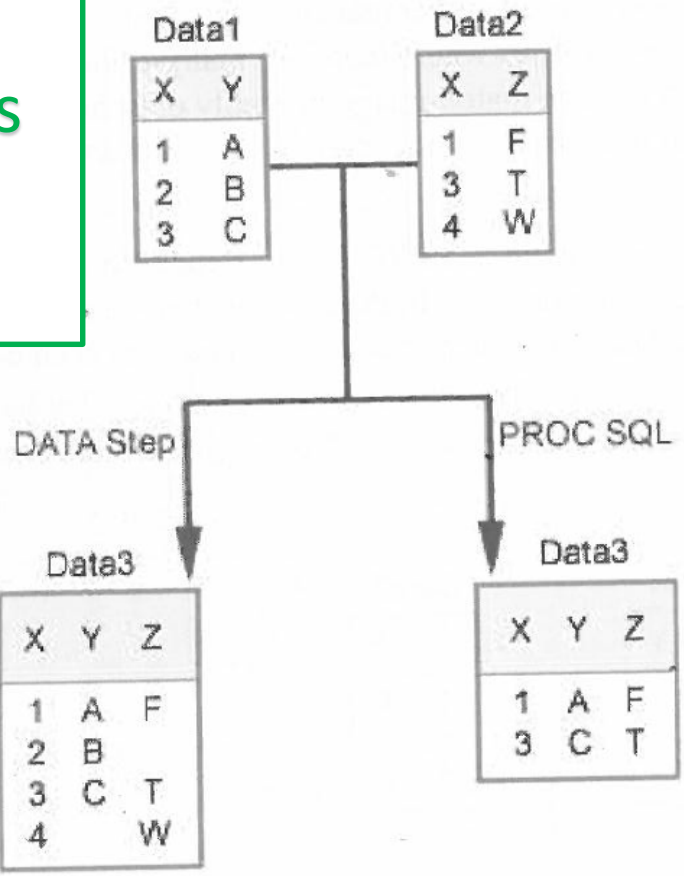

```
proc sql;
   create table data3 as
      select *
      from data1 as a
         full join
         data2 as b
      on a.x = b.x
   ;
quit;
```

DATA MERGE mishandles repetitions of key variables.

Only PROC SQL can produce a Cartesian product.

Many-to-Many Joins: DATA MERGE vs PROC SQL

DATA MERGE keeps all values of the key variable.



PROC SQL may drop values and/or rows for values of the key variable that are not shared by both datasets.

Q: What type of SQL join was chosen in this case?

Use COALESCE() for the key variable as needed.

Tips

Avoid many-to-many merges whenever possible.

- Use a combination of key variables that are unique for all rows in at least one of the data sets.

If you cannot avoid a many-to-many merge, here are a couple of methods to consider:

- PROC SQL. If any rows are missing a key variable, try using a FULL JOIN with the COALESCE() function.
- Subset out and set aside rows missing a key variable. Add those rows to the result from joining/merging rows with non-missing values for the key variables.

Some References / Further Reading

Malachy J. Foley (1997), “Advanced MATCH-MERGING: Techniques, Tricks, and Traps”, SUGI 22, Paper 39.

Malachy J. Foley (1998), “MATCH-MERGING: 20 Some Traps and How to Avoid Them”, SUGI 23.

Malachy J. Foley (2005), “MERGING vs. JOINING: Comparing the DATA Step with SQL”, SUGI 30, Paper 249-30.

James Lew and Joshua Horstman (2012), “Avoiding Pitfalls when Merging Data”, MWSUG Paper S111-2012.

SAS Institute Inc. (2011), *SAS Certification Prep Guide: Advanced Programming for SAS 9*, Third Edition.