Paper 252-2012

Selecting All Observations When Any Observation Is of Interest

Christopher J. Bost, MDRC, New York, NY

ABSTRACT

A data set might contain multiple observations per person. Suppose you want to keep all observations for a person if at least one observation for that person meets certain criteria. This paper shows how to use PROC SQL to select all observations when any observation is of interest. The first method uses a subquery; the second method uses the GROUP BY and HAVING clauses. The SQL techniques are compared and contrasted with a traditional DATA step match-merge.

INTRODUCTION

Some data sets have more than one observation per person. For example, a high school data set might have multiple observations per student, one for each course that she takes. WHERE or IF statements can be used to select observations of interest. For example, you might select courses that qualify for Advanced Placement (AP) college credit.

You might, however, want to select all observations for someone if any observation for that person is of interest. For example, you might want to select all courses for a student if she has ever taken any Advanced Placement course. This is typically accomplished with multiple DATA steps and a match-merge.

This paper reviews the DATA step approach and then offers two different ways to produce the same results with PROC SQL.

SAMPLE DATA

SAS® data set COURSES is used in this paper:

studentid	course	ap
1	BIOL101	0
2	BIOL102	1
2	CHEM102	1
3	CALC101	0
3	STAT101	0
4	PSYC201	1
4	HIST102	0
4	PHYS101	1
5	CHEM101	0
5	CALC102	1

Output 1. Data set COURSES

Data set COURSES contains ten observations and three variables:

STUDENTID is a unique identifier for each student
COURSE is a unique code for each class
AP is a dummy variable for Advanced Placement

There is one observation for each class a student takes, so there can be multiple observations per student. Student 1 took one course. Student 2, Student 3, and Student 5 took two courses. Student 4 took three courses.

Student 1 and Student 3 did not take any AP courses (i.e., where AP equals 1). Student 2 and Student 4 took two AP courses. Student 5 took one AP course.

The desired data set will contain all observations in red in Output 1 (i.e., AP courses and non-AP courses for Student 2, Student 4, and Student 5).

SELECTING OBSERVATIONS WITH THE DATA STEP

A common approach to selecting all observations when any observation is of interest is to use a match-merge. (Other approaches are possible.) This involves three steps.

1: SUBSET OBSERVATIONS OF INTEREST

First, subset observations for Advanced Placement courses. All variables are kept for illustrative purposes:

```
data ap;
set courses;
where ap=1;
run;
```

The resulting data set looks as follows:

studentid	course	ap
2	BIOL102	1
2	CHEM102	1
4	PSYC201	1
4	PHYS101	1
5	CALC102	1

Output 2. Data set AP

All Advanced Placement courses are included in data set AP. Note that some values of STUDENTID are repeated (i.e., if the student took more than one AP course).

2: KEEP ONE OBSERVATION PER IDENTIFIER

Second, keep one observation per STUDENTID. Keep only STUDENTID in the output data set, which will serve as a lookup table:

```
proc sort data=ap out=ap_sort;
by studentid;
run;

data ap2;
set ap_sort;
by studentid;
if first.studentid;
keep studentid;
run;
```

The PROC SORT step sorts observations in data set AP by STUDENTID and saves the results in data set AP_SORT.

The DATA step creates data set AP2. The SET statement reads observations from data set AP_SORT.

The BY statement reads observations in order by STUDENTID. This creates two variables, FIRST.STUDENTID and LAST.STUDENTID. The variable FIRST. STUDENTID equals 1 when the observation contains the first occurrence of a STUDENTID and 0 when it does not. Similarly, the variable LAST.STUDENTID equals 1 when the observation contains the last occurrence of a STUDENTID and 0 when it does not.

The IF statement subsets the first occurrence of each STUDENTID. IF STUDENTID; is equivalent to IF STUDENTID=1;

The KEEP statement keeps only the variable STUDENTID in data set AP2. The resulting data set looks as follows:

studentid
2
4
5

Output 3. Data set AP2

STUDENTID values for students who took at least one Advanced Placement course now occur once in data set AP2.

3: MATCH-MERGE OBSERVATIONS

Third, merge the original data set and the lookup data set, keeping only those observations that match a STUDENTID in the lookup data set:

```
proc sort data=courses out=courses_sort;
by studentid;
run;

data anyap;
merge courses_sort ap2(in=inap2);
by studentid;
if inap2;
run;
```

The PROC SORT step sorts observations in data set COURSES by STUDENTID and saves the results in data set COURSES_SORT.

The DATA step merges observations in COURSES_SORT and AP2 (the lookup data set) by STUDENTID.

The IN= data set option in parentheses creates INAP2, a dummy variable that equals 1 when the data set contributes to the current observation and 0 when it does not.

The IF statement subsets observations that match a STUDENTID in data set AP2. IF INAP2; is equivalent to IF INAP2=1;

The resulting data set looks as follows:

studentid	course	ap
2	BIOL102	1
2	CHEM102	1
4	PSYC201	1
4	HIST102	0
4	PHYS101	1
5	CHEM101	0
5	CALC102	1

Output 4. Data set ANYAP

Data set ANYAP contains all courses for students who registered for at least one Advanced Placement course. This is the desired data set. Note, however, that three DATA steps and two PROC SORT steps were required. PROC SQL can produce the same data set with a single step.

SELECTING OBSERVATIONS WITH PROC SQL

PROC SQL can be used to select all observations when any observation is of interest. Two methods are reviewed.

1: USE A SUBQUERY

The following PROC SQL step uses a subquery to create a lookup table that is used to select observations:

```
proc sql;
create table anyap2 as
select *
from courses
where studentid in (select distinct studentid
                    from courses
                    where ap=1)
order by studentid;
quit;
```

The PROC SQL statement starts the procedure.

The CREATE TABLE clause saves the query results to SAS data set ANYAP2.

The SELECT clause selects all columns (variables).

The FROM clause reads rows (observations) from table (data set) COURSES.

The WHERE clause subsets rows where the value of STUDENTID is IN the list of values in parentheses.

The list of values is produced by a subquery. A subquery (or "inner query") is an SQL query on a WHERE clause. It must be in parentheses and is, therefore, evaluated first. It can include all clauses except ORDER BY and INTO. It must return a single value or multiple values from a single column.

In the above subquery, the SELECT clause picks up values of STUDENTID from data set COURSES where the value of AP equals 1. The DISTINCT keyword picks up each STUDENTID value only once. The subquery creates a list of values that are used by the outer query.

The ORDER BY clause sorts the query results by STUDENTID. Note that PROC SQL may or may not process rows sequentially. Use the ORDER BY clause to sort the query results by the specified column(s).

The resulting data set looks as follows:

studentid	course	ap
2	BIOL102	1
2	CHEM102	1
4	PSYC201	1
4	PHYS101	1
4	HIST102	0
5	CALC102	1
5	CHEM101	0

Output 5. Data set ANYAP2

Data set ANYAP2 contains all courses for students who registered for at least one Advanced Placement course. This is the desired data set. Only one step was required. Note, however, that the subquery and the outer query each require a separate pass through data set COURSES.

2: USE GROUP BY AND HAVING CLAUSES

The following PROC SQL step groups all rows with the same STUDENTID together and keeps the group when at least one row in the group is an Advanced Placement class:

```
proc sql;
create table anyap3 as
select *
from courses
group by studentid
having sum(ap=1) > 0
order by studentid;
quit;
```

The PROC SQL statement starts the procedure.

The CREATE TABLE clause saves the query results to SAS data set ANYAP3.

The SELECT clause selects all columns (variables).

The FROM clause reads rows (observations) from table (data set) COURSES.

The GROUP BY clause groups all rows with the same value of STUDENTID together.

The HAVING clause “post-processes” each group. If the group meets the specified condition, it is selected (i.e., all rows in that group are selected).

The condition includes the SQL summary (or aggregate) function SUM. The expression in parentheses is evaluated for each row. If the expression is true (AP=1), SAS returns a 1. If the expression is false, SAS returns a 0. The SUM function adds the returned values across all rows in the group. If the sum is greater than 0—meaning there was at least one AP course—all rows in that group are selected.

The ORDER BY clause sorts the query results by STUDENTID.

The resulting data set looks as follows:

studentid	course	ap
2	CHEM102	1
2	BIOL102	1
4	HIST102	0
4	PSYC201	1
4	PHYS101	1
5	CALC102	1
5	CHEM101	0

Output 6. Data set ANYAP3

Data set ANYAP3 contains all courses for students who registered for at least one Advanced Placement course. This is the desired data set. Only one step was required. Note, however, that SAS prints the following in the log:

```
NOTE: The query requires remerging summary statistics back with the original data.
```

This note is informational only (i.e., does not indicate a problem). The aggregate function SUM on the HAVING clause remerges the calculated sum with each row in its respective group. This will entail additional processing time.

Note the flexibility of this approach. Any condition (or conditions) of interest may be specified in the parentheses after the SUM function on the HAVING clause.

CONCLUSION

A match-merge, PROC SQL subquery, and PROC SQL query using GROUP BY and HAVING clauses can be used to select all observations when any observation is of interest. The DATA step approach requires multiple steps and more coding than either of the PROC SQL approaches.

Note that while the SQL queries each require a single step and significantly less coding, they are not necessarily more efficient than the match-merge. If processing time is an issue, test all three of these approaches using your data to see which runs the fastest.

The sample data in this paper contain groups of observations for people, but these techniques work for any data with groups of observations with one or more identifiers.

REFERENCES

SAS Institute Inc. *SAS® 9.2 SQL Procedure User's Guide.* Cary, NC: SAS Institute Inc., 2009.

CONTACT INFORMATION

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

Christopher J. Bost
MDRC
16 East 34th Street, 19th Floor
New York, NY 10016
(212) 340-8613 telephone
(212) 684-0832 fax
christopher.bost@mdrc.org
www.mdrc.org