Handling Missing Values in the SQL Procedure

Danbo Yi, Abt Associates Inc., Cambridge, MA
Lei Zhang, Domain Solutions Corp., Cambridge, MA

ABSTRACT

PROC SQL as a powerful database management tool provides many features available in the DATA steps and the MEANS, TRANSPOSE, PRINT and SORT procedures. If properly used, PROC SQL often results in concise solutions to data manipulations and queries. PROC SQL follows most of the guidelines set by the American National Standards Institute (ANSI) in its implementation of SQL. However, it is not fully compliant with the current ANSI Standard for SQL, especially for the missing values. PROC SQL uses SAS System convention to express and handle the missing values, which is significantly different from many ANSI-compatible SQL databases such as Oracle, Sybase. In this paper, we summarize the ways the PROC SQL handles the missing values in a variety of situations. Topics include missing values in the logic, arithmetic and string expression, missing values in the SQL predicates such as LIKE, ANY, ALL, JOINs, missing values in the aggregate functions, and missing value conversion.

INTRODUCTION

SQL procedure follows the SAS® System convention for handling missing values. The way it expresses missing numeric values and character values are totally different. A missing numeric value is usually expressed as a period (.), but it also can be stated as one of other 27 special missing value expressions based on the underscore (_) and letters A, B,...,Z, that is, _A, _B,..., _Z. In SAS SQL procedure, a particular missing value is equal to itself, but those 28 missing values are not equal to each other. They actually have their own order, that is _< .< _A< _B,...< _Z. When missing numeric values are compared to non-missing numeric value, the missing numeric values are always less than or smaller than all the non-missing numeric values. A missing character value is expressed and treated as a string of blanks. Missing character values are always same no matter whether it is expressed as one blank, or more than one blanks. Obviously, missing character values are not the smallest strings.

In SAS system, the way missing Date and DateTime values are expressed and treated is similar to missing numeric values.

This paper will cover following topics.
1. Missing Values and Expression
   • Logic Expression
   • Arithmetic Expression
   • String Expression
2. Missing Values and Predicates
   • IS NULL /IS MISSING
   • [NOT] LIKE
   • ALL, ANY, and SOME
   • [NOT] EXISTS
3. Missing Values and JOINs
   • Inner Join
   • Left/Right Join
   • Full Join
4. Missing Values and Aggregate Functions
   • COUNT Functions
   • SUM, AVG and STD Functions
   • MIN and MAX Functions
5. Missing value conversion

In order to see how differently the missing data are handled in SQL procedure, we will produce examples based on a very small data set that can be generated by following SAS codes.

data ABC;
input x1 1-2 x2 4-5
   y1 $7-9 y2 $11-12;
datalines;
-1 2 ABC 12
0 -3 DE 34
1
1 1 CDE 56
The small data set has four variables, X1 and X2 are numeric variables, and Y1 and Y2 are character variables.

1. MISSING VALUES AND EXPRESSION
Since SAS system uses different ways to express the missing numeric values and missing character values, the missing values are treated differently in the logic, arithmetic, and string expression.

LOGIC EXPRESSION
When a missing numeric value appear in the logic expression (not, and, or). The missing numeric value is regarded as 0 or FALSE. In another word, In SAS system, 0 and missing values are regarded as FALSE, and non missing numeric values except 0 are regarded as TRUE. Consider this example,

```
Proc sql;
  Select not x1 as z1, x1 or . as z2,
       x1 and . as z3
  from ABC;
Result:

<table>
<thead>
<tr>
<th>Z1</th>
<th>Z2</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>.</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Notice that missing numeric values behave like FALSE or 0 in NOT and AND logical expression, but in the logical OR expression, the missing values was simply dropped because of the nature of OR operation.

ARITHMETIC EXPRESSIONS
When you use a missing numeric value in an arithmetic expression (+,-,*,/), the SQL procedure always set the result of the expression to period (.) missing value. If you use that result in another expression, the next result is also period (.) missing value. This method of treating missing values is called propagation of missing values. For example,

```
Proc SQL;
  Select x1+1 as z1 from ABC
      where x1<0;
Result:

<table>
<thead>
<tr>
<th>Z1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

Notice that all special missing values are changed into period (.) missing value after the calculation.

STRING EXPRESSIONS
When you use a missing character value in a string expression concatenated by ||, the SQL procedure sets the missing character value to a string of blanks, the length of which is equal to the length of the variable. Here is an example,

```
Proc SQL;
  Select 'AA'||y1||y2 from ABC;
Result:

<table>
<thead>
<tr>
<th>Z1</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAABC12</td>
</tr>
<tr>
<td>AADE 34</td>
</tr>
<tr>
<td>AA</td>
</tr>
<tr>
<td>AACDE56</td>
</tr>
<tr>
<td>AA</td>
</tr>
<tr>
<td>AAABC90</td>
</tr>
</tbody>
</table>

```

Notice that missing numeric values behave like FALSE or 0 in NOT and AND logical expression, but in the logical OR expression, the missing values was simply dropped because of the nature of OR operation.
2. MISSING VALUES AND PREDICATES

In SQL procedure predicates test some conditions that have the truth-value TRUE, FALSE, and NULL, and returns TRUE and FALSE. It also has special predicates to handle missing values, that is IS NULL and IS MISSING. Although missing character values are expressed as a string of blanks, but a string of blanks are always regarded as missing values. You will find this situation in a LIKE predicate below.

**IS [NOT] NULL / IS [NOT] MISSING**

IS [NOT] NULL and IS [NOT] MISSING predicates are two predicates especially designed to deal with missing values. IS [NOT] NULL is a SQL standard predicate and IS [NOT] MISSING is SAS SQL predicate. They are generic because they can handle both numeric and character variables. For example,

Proc SQL;
Select x1, y1 from ABC
Where x1 is not null and y1 is not missing;

Result:

<table>
<thead>
<tr>
<th>X1</th>
<th>Y1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>ABC</td>
</tr>
<tr>
<td>0</td>
<td>DE</td>
</tr>
<tr>
<td>1</td>
<td>CDE</td>
</tr>
</tbody>
</table>

**LIKE**

As mentioned above, missing character values are always the same but there is one exception. In SAS SQL LIKE predicate, missing values are regarded as a string of blanks, they are not missing values. If the length of a character column variable is 3, then a missing character value is like one blank, two blanks, and three blanks, but not like four blanks or more than four blanks. Consider following SQL statement, which returns the same results because the length of y1 is 3.

```
proc sql;
/* One Blank */
select x1, y1
from ABC
where y1 like ' ';  
/* Two Blanks */
select x1, y1
from ABC
where y1 like '  ';
/* Three Blanks */
select x1, y1
from ABC
where y1 like '   ';
```

Same result:

<table>
<thead>
<tr>
<th>X1</th>
<th>Y1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

However, the following SQL statement returns nothing.

```
proc sql;
/* Four Blanks */
select x1, y1
from ABC
where y1 like '    ';  
```

For two character variables, say A, and B, with different size, if size(A) > size(B), then predicate 'A like B' is TRUE, but predicate 'B like A' is FALSE when they both have missing values. Note that predicate 'A=B' is
always TRUE when both have missing
values. For example,

```
Proc sql;
    select x1, y1, y2
    from ABC where y1 like y2;
```

will return the following results:

<table>
<thead>
<tr>
<th>X1 Y1 Y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

However,  
```
Proc sql;
    select x1, y1, y2
    from ABC where y2 like y1;
```

Returns nothing.

**ALL, ANY, AND SOME**

ALL, ANY and SOME are special predicates that are oriented around subqueries. They are used in conjunction with relational operator. Actually, there are only two because ANY and SOME in SQL procedure are the same. ALL, ANY, and SOME are very similar to the IN predicate when it is used with subqueries; they take all the values produced by the subquery and treat them as a unit. However, unlike IN, they can be used only with subqueries.

In SQL procedure, ALL and ANY differ from each other in how they react if the subquery procedures have no values to use in a comparison. These differences can give your queries unexpected results if you do not account for them. One significant difference between ALL and ANY is the way they deal with the situation in which the subquery returns no values. In SQL procedure, whenever a legal subquery fails to produce output, the comparison operator modified with ALL is automatically TRUE, and the comparison operator modified with ANY is automatically FALSE. This means that the following queries

```
Proc SQL;
    select x2
    from ABC
    where x2 > ALL
    (select x1 from ABC
     where x1 >3);
```

```
select x2
from ABC
where x2 < ALL
( select x1 from ABC
  where x1 >3);
```

```
select x2
from ABC
where x2 NE ALL
( select x1 from ABC
  where x1 >3);
```

would produce the entire list of X2 values as following results.

<table>
<thead>
<tr>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>Z</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

Whereas those queries

```
Proc SQL;
    select x2
    from ABC
    where x2 > SOME
    ( select x1 from ABC
      where x1 >3);
```

```
select x2
from ABC
where x2 < SOME
( select x1 from ABC
  where x1 >3);
```

```
select x2
from ABC
where x2 NE SOME
( select x1 from ABC
  where x1 >3);
```

would produce no output. Of course, neither of these comparisons is very meaningful.
[NOT] EXISTS

In [NOT] EXISTS predicate, the way missing values are dealt with in SQL procedure is different from that in most SQL databases because missing values are regarded as comparable values in SQL procedures. Consider following example.

```sql
Proc SQL;
  select x1
  from ABC as a
  where exists (select x1
                from ABC as b
                where a.x1=b.x1);
```

would return an entire list of X1 values, including all the missing values, but in most SQL databases, it will only return non-missing values. Because in most SQL databases, if the missing values are used in the predicate of the subquery, the predicate is made unknown in every case. This means the subquery will produce no values, and EXISTS will be false. This, naturally makes NOT EXISTS true. However, in SQL procedure, missing values are normal comparable values that would be used in evaluation of subquery.

3. MISSING VALUES AND JOINS

In SQL procedure, a missing value equals to itself. When joining tables with missing values, the results are most likely different from those from most of ANSI-compatible SQL databases such as Oracle, Sybase because missing values are never equal to each other in those database system.

For INNER JOIN, SAS SQL will probably produce more observations and for FULL JOIN, SAS SQL will probably have fewer observations if joining tables have missing values. Here are two examples:

Example 1

```sql
proc sql number;
  select T1.x1, T2.x2
  from ABC as T1
  Inner join ABC as T2
  on (T1.x1=T2.x2);
```

Result:

<table>
<thead>
<tr>
<th>Row</th>
<th>X1</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

Example 2

```sql
proc sql number;
  select T1.x1, T2.x2
  from ABC as T1
  LEFT join ABC as T2
  on (T1.x1=T2.x2);
```

Result:

<table>
<thead>
<tr>
<th>Row</th>
<th>X1</th>
<th>X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>_</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

4. MISSING VALUES AND AGGREGATE FUNCTIONS

SQL procedure supports most common aggregate functions, or statistical summary functions such as count, average, sum, min, max, and standard deviation. They are functions that work on a set of values. Aggregate functions first construct a column variable as defined in the parameter. The parameter is usually a single column variable, but it can be an arithmetic expression with scalar functions and other column variables. Once the working column is constructed. The aggregate function performs its operation on a set of known values and the unknown values, or missing values will be given a special treatment dependent on the individual function.
COUNT FUNCTION
Count function works on all SAS data type. Count(*) returns the number of rows in a table. It is the only aggregate function that uses as asterisk (*) as a parameter. The missing values in the table are counted because this function deals with observations or rows and not individual values. For an empty table, COUNT(*) returns zero.

The COUNT([ALL]< sql-expression>) returns the number of non-missing values in the <sql-expression> set. The missing values have been excluded before the counting take place. It returns zero when <sql-expression> set is empty or consisted of only missing values. The COUNT(DISTINCT <sql-expression>) returns the number of unique non-missing values in the <sql-expression> set. The missing values have been excluded before the counting take place, and then all redundant duplicates are removed. Again, it returns zero when <sql-expression> set is empty or consisted of only missing values.

If COUNT ([ALL | DISTINCT] <sql-expression>) is used to count the number of values in character variable, the blank strings will be excluded, which is undesirable in some situations. SAS function N, and FREQ have the exact functionality as COUNT ([ALL | DISTINCT] <sql-expression>), but there exist no N(*) and FREQ(*) functions in the SQL procedure. The other related SAS function is NMISS, which counts the number of missing values in a <sql-expression> set variable. You can even use NMISS(DISTINCT <sql-expression>) to count the number of distinct missing numeric values. Below are two examples:

```sql
proc sql;
create table DUAL (nothing num);
insert into DUAL values(1);
select count(*) from ABC as C1,
     (select count(x1) from ABC) as C2,
     (select count(all x1) from ABC) as C3,
     (select count(distinct x1) from ABC) as C4,
from DUAL;
result:
  C1  C2  C3  C4
  ---- ---- ---- ----
   7   4   4   3
```

```sql
proc sql;
create table DUAL (nothing num);
insert into DUAL values(1);
select
     (select N(distinct x1) from ABC) as C5,
     (select freq(distinct x1) from ABC) as C6,
     (select NMISS(distinct x1) from ABC) as C7
from DUAL;
result:
  C5  C6  C7
  ---- ---- ----
   3   3   3
```

MIN AND MAX FUNCTIONS
MIN and MAX functions work on numeric, DATE, DATETIME and character variables. MAX(<sql-expression>) returns the greatest known value in the <sql-expression> set, and MIN(<sql-expression>) returns the smallest known value in the <sql-expression> set if the <sql-expression> set is not empty or not consisted of only missing values. The missing values have been excluded from the operation before the functions take place. It returns a period (.) for DATE, DATETIME and numeric values, and return blanks for character variables when the <sql-expression> set is empty or consisted of only missing values.
The MAX() for a set of known numeric values is the largest one. The MAX() for a set of known DATE and DATETIME data is the one farthest in the future or most recent. The MAX() for a set of non-blank character strings is the last one in the ascending sort order. Likewise, the MIN() for a set of known numeric values is the smallest one. The MIN() for a set of known DATE and DATETIME data is the least recent one. The MIN() for a set of non-blank character strings is the first one in the ascending sort order. Here is an example:

```sql
proc sql;
    Select min(x1) as minx,
        max(x1) as maxx,
        min(y1) as miny,
        max(y1) as maxy
    from ABC;
Result:
  MINX MAXX MINY MAXY
---------------------------
   -1    1    ABC    ER
```

5. MISSING VALUES CONVERSION

Missing numeric values and character values can be converted into each other by SAS INPUT and PUT function, and they also can be converted into character values in macro variables by using INTO clause in the SELECT statement. A missing numeric value can be converted into a character value such as a period (.), underscore (-), or characters A-Z depending on the corresponding missing value form. A missing character value is always converted into a period (.) missing value. Here is an example.

```sql
proc sql;
    select x1 into :valx1 from ABC
    where x1 = ._;
    select x1 into :valx2 from ABC
    where x1 = .;
    select x1 into :valx3 from ABC
    where x1 = .A;
    select y1 into :valy1 from ABC
    where y1=' ';
%put valx1=**&valx1** is %length(&valx1);
%put valx2=**&valx2** is %length(&valx2);
%put valx3=**&valx3** is %length(&valx3);
%put valy1=**&valy1** is %length(&valy1);
```

SUM AND OTHER RELATED FUNCTIONS

SUM only works on numeric values. SUM(ALL<sql-expression>) returns the numeric total of all known numeric values. It returns a missing value when the <sql-expression> is empty or consisted of only missing values.

SUM(DISTINCT<sql-expression>) returns the numeric total of all known, unique numeric values. The missing values and all duplicates have been removed before the summation took place. It returns a period (.), or a missing value if the <sql-expression> set is empty or consisted of only missing values. The other related SAS SQL functions such as AVG, MEAN, RANGE, STD, STDERR, CV, CSS use the same mechanism as SUM function to handling the missing numeric values. Here is an example.

```sql
proc sql;
    Select sum(x1) as sumx,
        avg(x1) as avgx,
        std(x1) as stdx,
        range(x1) as rangex
    from ABC;
Result:
  SUMX AVGX STDX RANGEX
-------------------------
     1   0.25  0.957427      2
```
Log output:

| Val x1=**       _** is 1 |
| Val x2=**       .** is 1 |
| Val x3=**       A** is 1 |
| Val y1=**       ** is 0 |

**CONCLUSION**

In this paper we have discussed the ways the SAS procedure handle missing values in a variety of situations. Although SQL procedure is a very powerful tool and can save a lot of programming time, you must have a good understanding of how missing values are handled in the SAS SQL procedure in order to avoid the possible pitfalls.

SAS and SAS/BASE mentioned in this paper are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.
® indicates USA registration

**REFERENCES**


**CONTACT ADDRESS**

Questions pertaining to this article should be addressed to:

Danbo Yi  
Abt Associates Inc.  
55 Wheeler Street  
Cambridge, MA 02138-1168, USA  
Tel: 617-349-2346  
Fax: 617-520-2940  
e-mail: danbo_yi@abtassoc.com

Lei Zhang  
DomianPharma Inc.  
10 Maguire Road  
Lexington, MA 02140, USA  
Tel: 781-778-3880  
Fax: 781-778-3700  
e-mail: lzhang@domainpharma.com