# Potential Result Set Differences between Relational DBMSs and the SAS System

Fred Levine, Senior Systems Developer, SAS Institute Inc

## Overview

Over the years, the SAS/ACCESS engine development team has made continuous performance improvements to our products by providing our users with the ability to offload to underlying relational data base management systems (RDBMSs) processing that normally would occur in the SAS System. In many cases, one can see significant performance improvements when processing occurs on the RDBMS server. There are three basic mechanisms that provide this functionality:

- The SQL Pass-Through facility, which utilizes special syntax in PROC SQL to allow RDBMS-specific SQL to get passed to the RDBMS server.
- The SAS WHERE clause used with a SAS/ACCESS engine. This is the WHERE clause that can be surfaced from any SAS procedure that operates on rectangular data.
- PROC SQL queries that process tables from a single SAS/ACCESS LIBNAME engine and contain performance-sensitive operations such as joins, distinct processing, and SQL-defined aggregate functions.

For the purposes of this paper, I will only address the latter two performance mechanisms because in these two cases, RDBMS server processing is normally transparent to the user. It is this transparency that at times can be troublesome due to the fact that

- Certain types of processing are handled differently in SAS than in most RDBMSs, which potentially can yield different result sets.
- Users may not always be aware of which software system is processing their SAS jobs. As a result, they may get unexpected results.

## Prerequisites

To get the most benefit from the concepts discussed in this paper, you should already be familiar with the syntax of the SAS/ACCESS LIBNAME statement that controls when queries are passed to the RDBMS. For further information, please refer to the SAS/ACCESS LIBNAME engine documentation.

## RDBMS Null Values Versus SAS Missing Values

How does RDBMS processing differ from SAS processing? The major area of processing where result set differences can occur is when processing null data.

In the SAS System, the element that is most similar to the RDBMS null value is the SAS missing value. Because RDBMS nulls and SAS missing values are conceptually the same, SAS/ACCESS engines will translate SAS missing values to RDBMS nulls when creating RDBMS tables from within SAS and, conversely, translate RDBMS nulls to SAS missing values when reading RDBMS tables into SAS.

So how do RDBMS Nulls and SAS missing values differ?

In most relational RDBMSs, nulls represent the *absence* of data; that is, RDBMS nulls do not sort or compare because there is no data on which to operate. However in SAS, we have the concept of a missing value, which is quite different. A SAS missing value is a special, reserved floating point number that can have 28 possible values represented by the following elements:

- A period ( **.** )
- An underscore and a period ( **_.** )
- A period and an alpha ( **.**A through **.**Z ).

These values make a lot of sense for SAS procedures that utilize survey data where more than one missing value is needed. For example, consider an epidemiological survey that asks a patient about their genetic predisposition to certain illnesses. The patient could decide to leave some of these questions blank for one or more reasons. They may not know the answer or actually refuse to answer. If the survey required a reason for not answering the question, then in this example, two missing values would be needed. For example, the values could be coded as .D for "do not know" and .R for "refuse to answer."

So it is helpful in such cases for software that has its roots in statistical analysis to provide multiple missing values. Because these missing values must be distinguishable from each other, it is important to note that *SAS* missing values **will evaluate with standard comparison operators, unlike RDBMS nulls which only evaluate with special null comparison operators, that is, "where column is null."**

This ability to evaluate missing values with standard comparison operators has significant implications when passing queries to an RDBMS for processing, because the RDBMS will interpret nulls differently than SAS.

In the following examples, I use the value "**.**" as the SAS missing value to illustrate these differences because the SAS "**.**" is the most commonly used missing value and is visually identical to an RDBMS null.

**Example 1 - Evaluating Column < 0 with Null Data**

Data Processed by RDBMS

```
libname ora oracle user=scott pw=tiger;

data ora.tabl1; x=. ;        /* create missing value */
run;
```

```
NOTE: The data set ORA.TABL1 has 1 observations and 1 variables.
NOTE: DATA statement used:
              real time         24.20 seconds
              cpu time          0.20 seconds
```

```
/* WHERE clause passed to Oracle for processing: */

proc print data=ora.tabl1; where x < 0;
run;
```

```
NOTE: No observations were selected from data set ORA.TABL1.
```

Data Processed by SAS

```
libname ora oracle user=scott pw=tiger direct_sql=(nowhere);

/* WHERE clause processed by SAS: */
proc print data=ora.tabl1; where x < 0;
run;
```

```
Obs   X
  1   .

NOTE: There were 1 observations read from the data set ORA.TABL1.
```

In this very simple example, we see a difference in results because in Oracle the null is interpreted as the absence of data and therefore will not evaluate with a '<' comparison operator. However in SAS, the null (SAS missing value) is interpreted as its internal floating point representation whose ordinal value is less than 0. Therefore, in SAS the expression does satisfy the condition of the WHERE clause.

**Example 2 - Comparing Nulls for Equality**

In the following two exhibits, PROC PRINT is used to show regional sales employees' salaries and commissions for a Midwest region and a South region. Note that not all these employees were given commissions, so the value of commissions can be null.

```
proc print data=ora.midwest;
run;
```

```
Obs   EMPID   SALARY   COMMISSION

 1    100     30000        1200
 2    101     35000           .
 3    102     33000           .

NOTE: There were 3 observations read from the data set ORA.MIDWEST.
NOTE: PROCEDURE PRINT used:

      real time         1.14 seconds
      cpu time          0.13 seconds
```

```
proc print data=ora.south;
run;
```

```
Obs    EMPID    SALARY    COMMISSION

 1      200      32500          .
 2      201      34000        800
 3      202      41000        250

NOTE: There were 3 observations read from the data set ORA.SOUTH.
NOTE: PROCEDURE PRINT used:

      real time         0.05 seconds
      cpu time          0.04 seconds
```

Using these two employee tables, we can find out which Midwestern sales employees made the same commission as the Southern sales employees. To do this, construct a simple inner join by first passing the query to Oracle and then to SAS. Here is the code for the Oracle processing along with the output:

```
/* WHERE clause passed to Oracle for processing: */

libname ora oracle user=scott pw=tiger;

 proc sql;

select midwest.empid, south.empid from ora.midwest, ora.south
      where midwest.commission=south.commission;
```

```
NOTE: No rows were selected.
```

No rows were selected. This is the result one would expect because, as one can see from the above output, there are no Midwestern employees who made the same commission as any of the Southern employees.

Next, here is the same query, but this time SAS does the processing:

```
/* WHERE clause processed by SAS: */

libname ora oracle user=scott pw=tiger direct_sql=no;

select midwest.empid, south.empid from ora.midwest, ora.south
      where midwest.commission=south.commission;
```

```
EMPID     EMPID
----------------------
  102         200
  101         200
```

Note in the output that the Midwestern employees identified as 101, 102, and the Southern employee identified as 200 did not make commissions. As a result, all of these employees had null values for commission.

The reason that SAS found these matches is that SAS missing values are interpreted as real ordinal numbers and therefore will match when one SAS missing value equals another. However, the result set from the SAS processing does not truly reflect the intent of the query. If I wanted to know which employees did **not** make a commission, I would have specified "where commission is null" as part of the query. (For more information about revising SAS code in these circumstances, see the section "Work-arounds for Null Data Problems.")

### Example 3 - Internally Comparing Nulls for Equality with Outer Joins

In the two examples so far, we have been using data that contains null values. However, it is also possible to get different result sets from SAS and an RDBMS when submitting outer joins where the data **does not contain nulls** but the internal processing **generates** nulls for intermediate result sets.

To illustrate, below are four simple Ingres tables that will construct a four-table outer join:

Table 1 - ing.q

```
proc print data=ing.q;
run;
```

```
Obs    x

 1     1
 2     3
 3     4
 4     6
```

Table 2 - ing.q2

```
proc print data=ing.q2;
run;
```

| Obs | x |
|---|---|
| 1 | 1 |
| 2 | 3 |
| 3 | 4 |
| 4 | 6 |

Table 3 - ing.y

```
proc print data=ing.y;
run;
```

| Obs | x |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |

Table 4 - ing.z

```
proc print data=ing.z;
run;
```

| Obs | x |
|---|---|
| 1 | 1 |
| 2 | 3 |
| 3 | 5 |

Here is the code used to pass the outer join query to Ingres:

```
/* SQL query passed to Ingres for processing: */

libname ing ingres database=clifftop;

proc sql;
    select * from ing.y left join ing.q  on y.x = q.x
                        left  join ing.z  on  z.x=y.x
                        right join ing.q2 on q.x = z.x;
```

Here are the results when Ingres processes the query:

```
 x      x      x      x
-----------------------------
 .      .      .      1
 .      .      .      3
 .      .      .      4
 .      .      .      6
```

Here is the same outer join query but this time processed by SAS:

```
/* SQL query processed by SAS: */

libname ing ingres database=clifftop direct_sql= no;

proc sql;
    select * from ing.y left join ing.q  on y.x = q.x
                        left  join ing.z  on  z.x=y.x
                        right join ing.q2 on q.x = z.x;
```

Here are the results from SAS:

```
 x      x      x      x
-----------------------------
 2      .      .      1
 2      .      .      3
 2      .      .      4
 2      .      .      6
```

None of these tables contain nulls. So why do we get different results? The answer lies in the fact that this query generates intermediate result sets. Therefore, let's analyze each intermediate result set to see how these tables are being joined.

First Intermediate Result Set

{select * from ing.y left join ing.q on y.x = q.x}

Tables Joined:

| y.x |
|---|
| 2 |
| 4 |
| 6 |
| |

| q.x |
|---|
| 1 |
| 3 |
| 4 |
| 6 |

Generates:

```
 y.x      q.x
 ----     ----
  2        .
  4        4
  6        6
```

Second Intermediate Result Set

{left join ing.z on z.x=y.x}

Tables Joined:

| y.x | q.x |
|---|---|
| 2 | . |
| 4 | 4 |
| 6 | 6 |

| z.x |
|---|
| 1 |
| 3 |
| 5 |

Generates:

```
 y.x      q.x      z.x
-----------------------------
 2        .        .
 4        4        .
 6        6        .
```

Third Intermediate Result Set

{right join ing.q2 on q.x = z.x;}

Tables Joined:

| y.x | q.x | z.x |
|---|---|---|
| 2 | . | . |
| 4 | 4 | . |
| 6 | 6 | . |

| q2.x |
|---|
| 1 |
| 3 |
| 4 |
| 6 |

Here is where this query gets interesting. Note that the ON clause in this last intermediate result set does not reference a column from the **q2** table being joined. This is where the problem lies. Notice the result set so far. In the first row, **q.x** does equal **z.x** when *SAS* does the processing because missing values do compare in SAS. As a result, the first row constitutes a match in SAS and is joined to the first row of **q2**. Because there are no other rows that match here, the final SAS-processed result set is as follows:

| y.x | q.x | z.x | q2.x |
|---|---|---|---|
| 2 | . | . | 1 |
| 2 | . | . | 3 |
| 2 | . | . | 4 |
| 2 | . | . | 6 |

When this same query gets passed to Ingres, the final result set looks different. Here are the relevant tables again just prior to the last join. When Ingres (or any RDBMS) processes this query, **q.x** will not equal **z.x** because these nulls have no value and therefore do not compare.

Tables Joined:

| y.x | q.x | z.x |
|---|---|---|
| 2 | . | . |
| 4 | 4 | . |
| 6 | 6 | . |

| q2.x |
|---|
| 1 |
| 3 |
| 4 |
| 6 |

Here is the final result set for Ingres. Note that because there are no matches, we only see nulls in the non-preserved tables.

Final Ingres Result Set:

| y.x | q.x | z.x | q2.x |
|---|---|---|---|
| . | . | . | 1 |
| . | . | . | 3 |
| . | . | . | 4 |
| . | . | . | 6 |

This example is a bit unusual because outer join queries usually reference at least one column from the tables being joined in their respective ON clauses. However, we can now see how different results can occur even when the data do not contain any nulls.

**Work-arounds for Null Data Problems**

A generic solution for the null data issue does exist that will always produce consistent results regardless of whether SAS or an underlying RDBMS is doing the processing. When specifying WHERE and ON conditions, one can add: "**and *<expression>* is not null**." This additional statement will prevent SAS from evaluating nulls thereby making SAS produce the same results as an underlying RDBMS. This extra statement will have no effect when passed to the RDBMS for processing.

When applied to the three examples discussed earlier, the revised code appears as follows. The additional "is not null" expressions are in red:

Revised Code for Example 1 - Evaluating Column < 0 with Null Data

```
proc print data=ora.tabl1; where x < 0 and x is not null;
run;
```

Revised Code for Example 2 - Comparing Nulls for Equality

```
select midwest.empid, south.empid from ora.midwest, ora.south
      where midwest.commission=south.commission
      and midwest.commission is not null;
```

Revised Code for Example 3 - Internally Comparing Nulls for Equality with Outer Joins

```
select * from ing.y left join ing.q  on (y.x = q.x and y.x is not null)
                    left  join ing.z  on (z.x=y.x and z.x is not null)
                    right join ing.q2 on (q.x = z.x and q.x is not null);
```

Note that in examples 2 and 3 in which we compare two columns for equality, you only need to add the "is not null" expression for one of the contributing columns because if one of them is not null, we could never be comparing two nulls for equality.

In all these examples, identical results would be returned regardless of whether SAS or the underlying RDBMS was doing the processing. The third example is admittedly cumbersome because a user may know very well that the underlying tables do not contain any null data and would only generate such a query to account for nulls in generated intermediate result sets that are not surfaced to the user.

Adding the “is not null” expression to all WHERE clauses and all ON clauses may not always be the most efficient thing to do, so I am not suggesting that it be specified all the time. Depending on the query it can add a lot of code complexity. However, it is a solution that will solve this problem and users should use their own judgement and knowledge of their own data to determine when it would be prudent to apply the “is not null” expression.

## Conclusion

The examples in this paper illustrate how it is possible to get different result sets depending on whether SAS or an underlying RDBMS (via a SAS/ACCESS LIBNAME engine) is doing the processing.

Although in many cases the differences illustrated in these examples will not present a problem, it is important for SAS/ACCESS users to understand how these result set differences can occur so they can tailor their underlying data and subsequent querying to achieve the desired results.

## Contact Information

Fred Levine
Senior Systems Developer
SAS Institute Inc
Cary, NC 27513
Fred.Levine@sas.com