

Using SAS® with Oracle®: Writing efficient and accurate SQL

Tasha Chapman and Lori Carleton
Oregon Department of Consumer and Business Services
Salem, Oregon

When using SAS to extract data from a live Oracle database, it is important to write accurate and efficient code that will not hamper the performance of other users. The best way to do this is to have Oracle perform the initial data query before bringing the selected records to SAS. To accomplish this, the SAS user must write SQL code that Oracle will understand and compute properly. SAS has included enhancements in Version 9.2, such as the so-called "Un-Put Technologies" that make this easier. This paper will teach SAS users, including those without access to Version 9.2, how to use PROC SQL efficiently and effectively with Oracle and other non-SAS databases.

Our SAS Environment (or "Does this apply to me?")

When using SAS with a non-SAS database management system (DBMS), you need to understand not only how SAS functions, but also how your particular DBMS functions, and how the two interact with each other. It may sound like a lot to keep in your head, but not knowing may have disastrous consequences. You may not get the results you meant to if you aren't accurate with your programming. And you may hamper system resources if you don't write efficient code. This paper will help you sidestep some of the common pitfalls when using SAS with a non-SAS DBMS.

Our work environment has servers dedicated to various processes under a Linux platform, including PC-SAS. Most of the data we analyze is in Oracle tables which are stored on a separate server. We use SAS/ACCESS to "talk" to the Oracle server. Because the Oracle tables are stored remotely, the SAS statements that query on the tables will execute on the Oracle servers and the rest of the program will execute on the SAS server. Although many of the concepts discussed in this paper will apply to systems, this paper was developed using SAS 9.1.3 and Oracle as our DBMS.

One thing that should be noted upfront is that, for the most part, **once the data is put in a SAS dataset, the rules in this paper no longer apply**, even if the data originally came from an Oracle database. SAS is much more forgiving and lenient with its coding rules than other database management systems.

Programming Accurately (or, how to get what you want from a Where clause)

SAS and Oracle all have slightly different ways of treating particular variables. Because of this, the WHERE clause in SQL will give different results depending on whether it is processed by SAS or Oracle. Here are some things to keep in mind when writing SQL statements for Oracle.

Character data and trailing blanks

When SAS creates SQL to pass to Oracle, it removes trailing blanks. To query on a particular code in your SQL, you can either include or exclude the trailing blank (e.g., `event_code = '012 '` (with a space) and `event_code = '012'` produce the same results).

Numeric data and leading zeros

Numeric fields in Oracle follow numeric rules. Leading zeros in a numeric field do not need to be included when selecting a record. For example, a number may be displayed as 0001. However, if it's a numeric field, you can select it with or without the leading zeros (e.g., `ins_number = 0001` and `ins_number = 1` will produce the same results).

Datetime values

Date values in Oracle may actually be datetime values, which include hours, minutes, and seconds (e.g., `'30sep2005:11:37:06'dt` is September 30, 2005, 11:37am and 6 seconds). Therefore, any selection criteria in the WHERE clause may need to account for dates with non-zero times.

Inaccurate:

```
'01sep2005'd le mod_date le '30sep2005'd - or -  
mod_date between '01sep2005'd and '30sep2005'd
```

Using either of these selection criteria will result in any modification date (mod_date) values on September 30th (datetime values after '30sep2005:00:00:00'dt) being excluded.

New options:

```
'01sep2005'd le mod_date lt '01oct2005'd - or -  
'01sep2005:00:00:00'dt le mod_date le '30sep2005:23:59:59'dt
```

Using either of these selection criteria will result in any mod_date values on September 30th (datetime values up to '30sep2005:23:59:59'dt) being included.

Do not use the DATEPART function in the WHERE clause to account for this issue. This may result in the entire Oracle table(s) being passed back to SAS before the query is processed, severely hampering efficiency (see below).

Formatting datetime values

Although you shouldn't use the DATEPART function in your SQL, you may want to use it later in a DATA step to convert Oracle's datetime variable to a more SAS-friendly date variable. Many of the SAS-supplied date formats are meant to be used with a date variable, and cannot be used with datetime variables. If you want to use a SAS-supplied date format (such as year4., mmdyy10., and others), you must first convert the datetime variables to date variables using the DATEPART function in a DATA step.

Null values

In SAS, any selection criteria that includes "not equals" (^=, <>, or ne), "less than" (< or lt), or "less than or equal to" (<= or le) will include null or missing values. For example, the criteria `Where Language ne "English"` will pull records where the language is Spanish, French, Chinese, or blank.

However, this is not the case in Oracle SQL. In Oracle, a null value is non-existent. It doesn't equal or not equal anything. For this reason, comparison operators will not apply for null records. `Where Language ne "English"` in an Oracle SQL statement will only pull records where the language is Spanish, French, or Chinese. Blank or null records will be excluded.

To include null records in the selection criteria, this must be stated explicitly, i.e., `Where Language ne "English" or Language is null`

Order of operations in the WHERE clause

Just as mathematical equations are solved in a particular order (multiplication and division first, followed by addition and subtraction)¹, Oracle will apply logical operators to your query in a very specific order. If you don't write your SQL query accordingly, you could end up with unexpected results.

For example, let's say you wanted to pull records that met either criteria A or criteria B and also met criteria C. You might write your WHERE clause like this:

```
Where A or B and C;
```

Logically you meant `Where (A or B) and C;` but what you don't know is that Oracle applies AND operators before OR operators. So what you ended up with was `Where A or (B and C);`

To fix this problem, apply parentheses accordingly. Just like in mathematical equations, Oracle will apply the parenthetical portion first, before moving to the outer portion of the query.

For future reference, when the WHERE clause contains multiple operators:

- Arithmetic operators will be resolved first.
- Comparison operators (<, >, =, LIKE, etc.) are resolved next.
- Logical operators are evaluated last, in the order of NOT, AND, and finally OR.

Programming Efficiently (or how to not bring your organization to its knees)

If you are running your programs from a live database, that is being used by multiple users at once, you must make sure your SAS programs are as efficient as possible. This means writing SQL that Oracle understands and can compute. Here are some tips for making an efficient SAS program.

¹ Remember PEMDAS? Please Excuse My Dear Aunt Sally?

SAStrace

SAStrace writes information in the SAS log about the SQL that is passed to Oracle, to determine the efficiency of your code. At the beginning of your program include the following syntax:

```
option sastrace = ',,,d' sastraceloc=saslog nostsuffix;
```

The information included in the SAS log by SAStrace can be very useful in determining if your SQL is efficient.

If you are only bringing in one table:

When your SAS code has a where clause (a key to efficiency), and is **efficient**, your SAS log will show that your WHERE clause was passed to Oracle:

```
Prepared: SELECT 'ui_number', 'year' WHERE ('year' = 2008)
```

When your SAS code is **not efficient**, the WHERE clause will not be passed to Oracle:

```
Prepared: SELECT 'ui_number', 'year'
```

If you are joining two tables, you may also see messages from the SAS/ACCESS engine:

When your SAS code is **efficient**, your SAS log will say something like this:

```
ACCESS ENGINE: SQL statement was passed to the DBMS for fetching data.
```

When your SAS code is **not efficient**, your SAS log will say something like this:

```
SAS_SQL: Unable to convert the query to a DBMS specific SQL statement due to an error.
```

```
ACCESS ENGINE: SQL statement was not passed to the DBMS, SAS will do the processing.
```

Testing

If possible, test your programs against a test or development database before you unleash your SAS code onto the rest of the world. If you notice inefficiencies, fix them before moving to a production environment.

If your SQL code is inefficient, here are some possible causes:

SAS functions

Oracle does not understand SAS functions, so they should not be used in the SQL procedure that is pulling data from an Oracle database. Some common SAS functions to look out for include DATEPART, COALESCE, PUT, INPUT, CALCULATED, SUBSTR, etc. Instead of using functions in the SQL procedure, use them later in a DATA step.

There are some exceptions to this rule if you are using Version 9.2 or later (more on this below).

Oracle reserved words

Field name aliases can be used for renaming variables in SQL. In SAS SQL, field name aliases follow an AS keyword (e.g., `select ui_number as bin`). If you must use aliases, do not use Oracle reserved words in your SQL. Oracle has certain reserved words that are commonly used for Oracle functions. If these words are used inappropriately Oracle will get confused. For example, **name** is a reserved word. Using name as an alias in your SQL will cause Oracle to send the SQL back to SAS for processing. (A web search for "Oracle reserved words" will provide helpful references on which words not to use.)

Joining Oracle tables with a SAS dataset

Many SAS users like to write SQL statements in steps, because they are used to writing SAS syntax in steps. For example:

```
proc sql;
create table claimAB as
select b.ownership_list
from oracle.employers as b;

create table claimC as
select claimAB.*, e.naics
from claimAB left join oracle.locations as e on claimAB.emp_no=e.empnr;
quit;
```

In the above example, the programmer is creating a SAS dataset (claimAB) from an Oracle table (oracle.employers), and then joining this dataset with another Oracle table (oracle.locations). However, Oracle doesn't have access to claimAB and doesn't know what's in it, so it will have to pass the second query back to SAS to process the join. Instead of programming SQL in steps, combine as many queries as possible into one statement. The above example should have been coded like so:

```
proc sql;
create table claimAB as
select b.ownership_list, e.naics
from oracle.employers as b left join oracle.locations as e
      on b.emp_no=e.empnr;
quit;
```

Some additional tips for improving efficiency:

Clean up your SAS program! Include only the tables and fields that you need in the SQL.

Use the WHERE clause. Try to include a WHERE clause with as many constraints as possible. For example, if you just need a specific year of data and specific codes, then add that to the WHERE clause (`WHERE year = 2008 and code in ('X', 'Y', 'Z')`).

Steer clear of ORDER BY. Use PROC SORT instead and make SAS do the work.

Avoid field name aliases. Aliases can be useful for renaming variables, but they can sometimes make programs less efficient. Try renaming variables using PROC DATASETS instead.

New enhancements to SAS 9.2 (or why we love SAS developers)

SAS recognizes the need to make SAS work well with other DBMS, such as Oracle, and are working on ways to make this relationship better. One of the new enhancements of SAS 9.2 is something known as "unPut Technology." In certain cases, SAS 9.2 will translate your SAS functions into something the DBMS can understand before passing on the code.

For example, some users prefer writing queries based on formatted values of a field:

```
Proc format;
Value sizefmt
1-3 = 'Small'
4-6 = 'Medium'
7-9 = 'Large';

Proc SQL;
Select style as SmallStyles
From db.clothes
Where put(size, sizefmt.) = 'Small';
Quit;
```

In versions prior to 9.2, this will result in inefficient code. SAS would attempt to send the DBMS "`Where put(size, sizefmt.) = 'Small';`" The DBMS would not understand this statement (because PUT is a SAS specific function), and would send the data back to SAS for processing. Starting in 9.2, SAS will translate this function before sending the query to the DBMS. In other words, when you write the above statement, Oracle (or whatever DBMS you are using) will only see this:

```
Select style as SmallStyles
From db.clothes
Where (1 <= size and size <=3);
```

The DBMS will understand this, and will send only the needed data. This improves efficiency, uses minimal resources, and in the end, your organization is happy.

More information about the unPUT technology can be found in:

- **A Sampler of What's New in Base SAS 9.2** by Jason Secosky, SAS Institute
- **New Performance Optimizations to Enhance Your SAS Client and Solution Access to the Database** by Mike Whitcher, SAS Institute

Using SQL in a DATA step (or a simple cheat for DATA Step-ers)

You can combine SQL and SAS functionality in one step by creating your entire query in a DATA step. For example, if you want to bring in an Oracle table and you want to perform some SAS functions, this can all be done in one DATA step.

So instead of this:

```
proc sql;
  create table test as
  select employer_name1,
         employer_name2,
         create_date
  from   oracle.xyz_emptbl
  where  status in (2,5,7);
quit;

data test2;
  set test;
  dba=catx(' ', employer_name1, employer_name2);
  cre8dt=datepart(create_date);
run;
```

You can do this and get equivalent results:

```
data test3;
  set oracle.xyz_emptbl
    (where=(status in (2,5,7))
     keep=status employer_name1 employer_name2 create_date);
  dba=catx(' ', employer_name1, employer_name2);
  cre8dt=datepart(create_date);
run;
```

The **SET** statement is equivalent to the **FROM** statement in SQL.

The **KEEP** statement is equivalent to the **SELECT** statement in SQL.

The **WHERE** statement is equivalent to the **WHERE** statement in SQL.

SAS converts the SET statement, and all of the dataset options within it (WHERE, KEEP, etc.) into SQL that the Oracle server can interpret. Oracle processes the SQL, returns the records set requested and sends it back to the SAS server to process the other statements in the DATA step.

There is one important difference between SQL and SAS that should be noted. When using SQL, it is not necessary to put the fields that are referenced in your WHERE clause in the SELECT statement if you don't wish to keep them in your table. However, in the DATA step, any field that is referenced in the WHERE clause must also be referenced in the KEEP statement (you can use a DROP statement in your DATA step if you don't want to keep that variable).

To select all the fields in a table, you can use `keep=_all_` in the KEEP statement, which is equivalent to `select *` in an SQL statement.

Contact info

We hope you've found these tips useful. Your comments and questions are valued and encouraged. If you have additional questions, feel free to contact us at:

Tasha Chapman

Oregon Department of Consumer and Business Services, Salem, Oregon
Tasha.L.Chapman@state.or.us

Lori Carleton

Oregon Department of Consumer and Business Services, Salem, Oregon
Lori.A.Carleton@state.or.us

Special thanks to Gary Helmer and Chuck Forbes, whose initial work lead to this paper.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Oracle is a registered trademark of the Oracle Corporation.
All other brands and product names are trademarks of their respective companies.