Dear Ms. SAS® Answers: A Guide to Efficient PROC SQL Coding

Jane Stroupe and Linda Jolley, SAS Institute Inc., Cary, NC

ABSTRACT

She's back! Advice columnist Ms. SAS® Answers has received more questions from curious users -- questions such as "What used all that CPU time?", "Which should I use, merge or join?", "Should I use a subquery?", and "Why should I learn SQL, anyway?" She will share her answers to help you harness the potential of Structured Query Language.

QUESTIONS AND ANSWERS

Dear Ms. SAS® Answers,
I can do anything with a DATA step. Why should I bother to learn SQL, anyway?
Signed,
Happy DATA Stepper

Dear Happy DATA Stepper,

Here are some advantages of the SQL procedure:

- PROC SQL follows ANSI standard language definitions, so that you can use SQL knowledge gained from other languages.
- PROC SQL is good at doing in one step what it might take several steps to accomplish in the DATA step.
- PROC SQL code is sometimes simpler to understand than the equivalent DATA step code.

If your data is in a relational database, here are some additional advantages to using the SQL procedure:

- PROC SQL allows you to write database-specific SQL statements and pass them directly to the database for execution.
- While WHERE statements and data set options, KEEP/DROP statements and data set options, BY statements, and some selected SAS functions can be passed to the database for processing no matter where they are coded in SAS, PROC SQL will also pass joins, GROUP BY clauses, and aggregate functions directly to the database for processing where possible.

In addition, PROC SQL joins have their own set of advantages:

- Multiple data sets can be joined without having common variables in all the data sets.
- Data sets do not have to be sorted or indexed.
- Inequality joins can be performed.

Think of the SQL procedure as another tool in your SAS tool belt. It gives you another possibility to use to solve the coding problem at hand.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,

I'm wondering why you have to type your clauses in the SELECT statement in a particular order. Is it because they execute in that order?

Signed,
Out of Order

Dear Out of Order,

The order of the clauses is governed by the American National Standards Institute (ANSI) SQL standard. Here's the order in which you must TYPE the general clauses:

```
proc sql;
   select ...
     from ...
       where ...
          group by ...
             having ...
                order by ...;
quit;
```

You can use one of the following sayings to help you remember the order:

Some **F**rench **w**orkers **g**lue **h**ardwood **o**ften.

San **F**rancisco, **w**here the **G**reatful Dead **H**eads **o**riginated.

or you can make up your own saying: whatever works best to help you remember the order in which to type the general clauses.

The order in which the clauses get processed by PROC SQL is generally as follows:

```
proc sql;
   select ...⑤
     from ...①
       where ...②
          group by ...③
             having ...④
                order by ...;⑥
quit;
```

1. The FROM clause is processed first. This opens the data sets ready for reading.
2. The WHERE clause is next. Just like the WHERE statement in the DATA step, the WHERE clause pre-processes the open data and selects only the rows that meet the WHERE clause predicates criteria. The results of WHERE clause processing are stored in an intermediate table.
3. The GROUP BY clause is third. The GROUP BY clause creates the groups of the rows the WHERE clause selected. A quick note here: SAS has to perform some type of ordering of the data to be able to create the different groups, but that does not necessarily mean that the groups will be placed in ascending order or alphabetical order. If you want to be sure that the final results show up in a particular order, you must specify an ORDER BY clause in addition to the GROUP BY clause.
4. The HAVING clause processes next. This clause subsets the groups created by the GROUP BY clause based on the HAVING clause predicates, and, like the WHERE clause, builds an intermediate table.
5. The SELECT clause selects the columns for the report or table.
6. The ORDER BY clause orders the remaining rows in the intermediate table.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,
Why does SQL take so much more CPU time to execute than the DATA step?
Signed,
Twiddling My Thumb**s**

Dear Thumbs,

The CPU time really depends on what you are asking PROC SQL or the DATA step to do. In general, if what you want is a very simple MERGE or INNER JOIN, the DATA step will finish faster than the equivalent PROC SQL code. In part, this is because of the differences in how the two steps execute.

The DATA step does a sequential read of all data sets on your MERGE statement. It starts with observation 1 in each data set and moves down through them, comparing BY values to determine which comes next and when to read from more than one data set.

Theoretically, PROC SQL creates a Cartesian product (joins all rows in all data sets) and eliminates rows that do not meet the WHERE or ON clause predicates. In fact, there are optimization routines that will pre-sort your input data sets, if needed, and create matching "chunks" of data to form the Cartesian product when doing a join on equal conditions (an equijoin).

Where PROC SQL really shines compared to the DATA step is if you have to pre-process any of the data to get it into shape for merging (sorting, etc.).

You can often accomplish in one step what it takes several steps to do with the DATA step.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,
Which is more efficient, the DATA Step MERGE or PROC SQL?
Signed,
Putting Them Together

Dear Together,

Well, that depends on your data and what you mean by "efficient".
You have to know

- the relationship between the tables
- the sparseness or density of matches
- the size of the tables
- the availability of an index or sort flag.

When data sets are large and unsorted, the SQL inner join might outperform SORT and MERGE. If you have a long series of SORT and DATA steps, the SQL inner join might be easier to code and read.

In most cases, a DATA step MERGE statement generally out performs an SQL outer join, even taking sort resources into account.

One exception is a very sparse match join when you only want the observations with matching key values.

Keep in mind that the SQL procedure and the DATA step MERGE do not provide the same results if you have a many-to-many match or non-matching data.

- ONE-TO-ONE matches produce identical results.

- ONE-TO-MANY matches produce identical results.

- MANY-TO-MANY matches produce different results.

- NON-MATCHING data produces different results.

Here's a handy comparison chart of the DATA step match MERGE and the SQL inner join:

Match Merge	**SQL Inner Join**
There is no limit to the number of data sets nor the size of the data sets other than disk space.	The maximum number of tables that can be joined at one time is 256.
Data is processed sequentially so that observations with duplicate BY values are joined one-to-one.	Data is processed using a Cartesian product for duplicate BY values.
Multiple data sets can be created.	Only one data set can be created with one CREATE TABLE statement.
Complex business logic can be incorporated using IF-THEN or SELECT/WHEN logic.	CASE logic can be used for business logic; however, it is not as flexible as DATA step syntax.
The data sets being merged must be sorted or indexed on the BY variable(s).	The data sets being joined do not have to be sorted nor indexed.
An exact match on the BY-variable(s) value(s) must be found.	Inequality joins can be performed.
Like-named BY variables must be available in all data sets.	Common variables do not have to be in all data sets.

Since there are no hard and fast rules about the efficiency of MERGE vs. SQL, you'll just have to benchmark them to find out how they perform with your data.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,

We have tons of users trying to retrieve subsets from a huge data set. Is there any way to make their response time faster?

Signed,
Dealing with Restless Users

Dear Dealing with Restless Users,

There are several techniques that you can use to make your WHERE clauses work better. I've listed a couple for you to try.

- If the subsets are small, indexing the SAS data set can make the query faster. An index is an optional file that can be associated with a SAS data set and enables SAS to directly access observations based on the value of the indexed variable.

  For example, if you have an index on the variable SSN, a WHERE clause such as "where SSN='123-45-6789'" first checks the index file for record identifiers for that particular value and can access the appropriate observations without having to read all observations in the data.

  You can create the index by using PROC SQL, PROC DATASETS, SAS Management Console, or the data set INDEX= option.

  Here's a PROC SQL example:

```
proc sql;
   create index SSN
       on sasuser.people(SSN);
quit;
```

- If indexing the data is impractical, then you could store the data in sorted order. In that case, SAS only queries the data until the WHERE conditions are satisfied, then it stops searching.

  If your users' queries generally retrieve data from the second half of your data set, you can use the DESCENDING option when sorting the data.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,

I learned in a SAS class that if you are using IF…THEN/ELSE processing, you should type the conditions in descending order of frequency and the IF…THEN/ELSE works faster. Is that true of the CASE clause in SQL? I love that CASE clause.

Signed,
Is It True?

Dear Is It True,

First, let's look at what the CASE clause does.

The CASE expression selects values if certain conditions are met. Here is an example:

```
proc sql;
   select Make, Model, Type,
      case DriveTrain
         when 'Rear' then 'Rear Wheels'
         when 'Front' then 'Front Wheels'
         when 'All' then 'Four Wheels'
         else ' '
      end as WheelType
   from sashelp.cars;
quit;
```

Now to answer your question:

Distribution	WHEN Expression Order
Uniform	Doesn't matter
Skewed	Most frequently occurring to least frequently occurring

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,

What's up with those SET operators? They seem to take so l-o-n-g to process. Is there any way to make them faster?

Signed,
Set in My Ways

Dear Set in My Ways,

The SET operators INTERSECT, UNION, and EXCEPT all return unique rows by default. If your queries are taking a long time to process, try using the ALL keyword under these circumstances:

- You do not care if you return duplicate rows with those SET operators.
- You know that the rows in the tables are unique.

Here's an example. Suppose you want a list of all the employees who made charitable donations in both of the last two years.

```
proc sql;
   select employee_id
      from orion.donations2007
   intersect all
   select employee_id
      from orion.donations2008;
quit;
```

The ALL keyword does not have to read the data sets multiple times in order to eliminate duplicates, so the ALL keyword can make your query more efficient.

Even if you know that you have no duplicates, use the ALL keyword. PROC SQL does not know that there are no duplicates, so without the ALL keyword PROC SQL processes the data twice.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,
I use the TODAY() function to subset my data. Is there any way I can create a variable from the TODAY function once without using a separate DATA step or creating a macro variable and reference that variable in SQL?
Signed,
Today is the Day

Dear Today,

I'm glad you asked that question! There is the SAS® 9.2 CONSTDATETIME option in the PROC SQL statement that you can use to evaluate the TODAY, DATE, TIME, and DATETIME functions in a query once and use the resulting value in the query.

Here's an example:

```
proc sql constdatetime;
   select *
       from orion.Sales_History
          where Order_Date = today();
   select *
      from orion.Employee_Payroll
         where Employee_Hired_Date = today();
quit;
```

Advantages of the CONSTDATETIME option include the following:

- It saves CPU resources.
- It ensures consistent results when the function is used multiple times in the query.
- If you are accessing database tables, the date can be passed to the database for processing.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,
I was reading a SAS Global Forum paper where they talked about an In-Line View. So what is it, exactly, and why would I use it?
Signed,
Curious to Know

Dear Curious,
An in-line view is a virtual table that is created by coding a SELECT statement on a FROM clause. It can be quite useful to prevent creating an intermediate SAS data set to solve complex problems.

Let's use this code as an example:

```
proc sql;
   create table temp as
       select Job_Title,
              sum(salary) as Total_Salary
           from orion.Employee_organization O,
                orion.Employee_Payroll P
               where O.Employee_ID = P.Employee_ID;
   create table pct as
      select Employee_ID,
             Sales.Job_Title,
             Salary,
             Salary/Total_Salary as Percent format = percent7.1
          from orion.Sales,
               temp
              where Sales.Job_Title = temp.Job_Title
                 order by Job_Title;
quit;
```

You could create a view, TEMP_VIEW, instead of a table:

```
proc sql;
   create view temp_view as
       select Job_Title,
              sum(salary) as Total_Salary
           from orion.Employee_organization O,
                orion.Employee_Payroll P
               where O.Employee_ID = P.Employee_ID;
   create table pct as
      select Employee_ID,
             Sales.Job_Title,
             Salary,
             Salary/Total_Salary as Percent format = percent7.1
          from orion.Sales,
               temp
              where Sales.Job_Title = temp.Job_Title
                 order by Job_Title;
quit;
```

Or you could use the SELECT statement from the view on the FROM clause of the second query as an in-line view.

```
proc sql;
   create table pct as
      select Employee_ID,
             Sales.Job_Title,
             Salary,
             Salary/Total_Salary as Percent format=percent7.1
          from orion.Sales,
               (select Job_Title,
                       sum(salary) as Total_Salary
                   from orion.Employee_organization O,
                        orion.Employee_Payroll P
                      where O.Employee_ID = P.Employee_ID
                  group by Job_Title) as TotSal
             where Sales.Job_Title = TotSal.Job_Title
                order by Job_Title;
quit;
```

The in-line view will execute once and return its result set to the outer query, which will then execute.

One additional advantage of in-line views is that you can test the in-line view query on its own to determine if it is returning the result set you expected.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,

My co-worker has created a bunch of SQL views for us to use, but they are much slower than if I had created a table. What's the problem?

Signed,
Viewing in Slow Motion

Dear Viewing in Slow Motion,

Use the DESCRIBE VIEW statement to look at the stored SELECT statement in your log.

```
proc sql;
   describe view orion.shoes;
quit;
```

Here's the log. Notice the ORDER BY clause in the view.

```
508  proc sql;
509     describe view shoes;
NOTE: SQL view WORK.SHOES is defined as:

         select Supplier_Name, Supplier_Country, Product_ID,
         Group_Name, Category_Name,
         Mfg_Suggested_Retail_Price from
         ORION.SHOE_VENDORS order by Supplier_Name;

510  quit;
```

The presence of the ORDER BY clause means that the data must be ordered by Supplier_Name every time it is used, even if the data does not need to be returned in sorted order.

Have your co-worker re-create the view, leaving off the ORDER BY clause. I think you'll find your queries run much faster.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,

My subquery is taking forever to execute. What's up?

Signed,
Sitting Here Waiting

p.s. Here's the code that is the problem. The two data sets are huge, by the way.

```
proc sql;
  select *
      from orion.Employee_Addresses
      where 'Payroll Deduction'=
         (select Paid_By
         from orion.Employee_Donations
              where
            Employee_Addresses.Employee_ID =
            Employee_Donations.Employee_ID);
quit;
```

Dear Sitting Here Waiting,

The problem with this code is that the inner query is a correlated subquery. Correlated subqueries can take a while to execute since they require that values are passed to the inner query from the outer query and must be evaluated for each row in the outer query.

You would probably be better off changing this from a correlated subquery into a join. Here's the new code:

```
proc sql;
   select a.*
      from orion.Employee_Addresses as a,
           orion.Employee_Donations as d
           where a.Employee_ID = d.Employee_ID
              and Paid_By = 'Payroll Deduction';
quit;
```

One advantage of the join is that the data set orion.Employee_Donations can be subset for the variable Paid_By, and the resulting data can be used for the join.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,
How does SQL optimize a join?

Signed,
Curious to Kno**w**

Dear Curious to Kno**w**,

How PROC SQL optimizes a join is complicated. SAS uses one of the following methods based on the relative size of the input data sets and the information SAS can gather from the descriptor portion of the data sets:

Optimization Method	When used
Indexed	There are any appropriate indexes. The observations are matched directly via that index.
Merge	One or both of the datasets are already sorted. In this case, only the groups of data with the same key values are combined.
Hash	One of the datasets will fit into memory. In this case, PROC SQL processes the rows from the other table sequentially, using table-lookup techniques to locate matching rows.
Sort Merge algorithm	This is the last resort, brute force method. PROC SQL sorts the incoming data sets and uses the MERGE method.

If you want to know the method being used, you can use the _METHOD option in the PROC SQL statement.

PROC SQL _METHOD;
query code
QUIT;

Here's a partial list of the abbreviations that might be in the log when you use the _METHOD option.

Abbreviation	Method
sqxcrta	Create table as SELECT
sqxslct	SELECT
sqxjsl	Step loop join (Cartesian product)
sqxjm	Merge join
sqxjndx	Index join
sqxjhsh	Hash join
sqxsort	Sort
sqxsrc	Source rows from table
sqxfil	Filter rows
sqxsumg	Summary statistics (with GROUP BY)
sqxsumn	Summary statistics (not grouped)
sqxuniq	Distinct rows only

Here's a sample log (with my comments) for you to see the information provided by the _METHOD option.

```
382  proc sql _method ;
383     select Employee_Addresses.*
384        from orion.Employee_Addresses,orion.Employee_Donations
385        where Employee_Addresses.Employee_ID = Employee_Donations.Employee_ID
386        and Paid_By = 'Payroll Deduction';

NOTE: SQL execution methods chosen are:

      sqxslct ← the SELECT clause
          sqxjhsh ← the Hash Join
              sqxsrc( ORION.EMPLOYEE_ADDRESSES ) ← Source Rows from table 1
              sqxsrc( ORION.EMPLOYEE_DONATIONS ) ← Source Rows from table 2
387  quit;
```

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,

You said I could send you some code that my users wrote and submit almost every hour. It's tying up our system to the point that no one can do anything else!

```
proc sql;
    select distinct Product_ID
     from orion.Orders;
quit;
```

There are tons of Product_ID's in that Orders data set. What am I to do?

Signed,
Frustrated Manager

Manager Dear,

Don't be frustrated, please. In SAS 9.2, there's a wonderful enhancement! PROC SQL will utilize an index when processing a SELECT DISTINCT statement. All you have to do is index the data set orders on the variable Product_ID.

Happy SQLing,
Ms. SAS® Answers

Dear Ms. SAS® Answers,

It's me again. Your last suggestion was perfect for solving the problem. But now I have another one. Here's a example of the queries that folks submit all the time and complain that it takes too long. Any hope of a solution?

```
proc sql;
   select Manager_ID,
          DeptSal, sum(DeptSal) as GrandTot,
          DeptSal/calculated GrandTot as percent
                    format=percent8.2
       from orion.totalsalaries;
quit;
```

Signed,
Not Quite so Frustrated Manager

Manager Dear,

Glad the frustration is better. And there's not much you can do to prevent this particular query from taking so long. The problem is, SQL has to total the value of DeptSal then re-merge it with the detail data. I know your data sets are huge, so I can imagine your pain.

But, you can turn on an option to prevent the query from running. Actually, there are two options, one a PROC SQL option, NOREMERGE. The other is a system option, NOSQLREMERGE, that goes on an OPTION statement.

If remerging is attempted when the NOMERGE option or the NOSQLREMERGE system option is set, an error is written to the SAS log

Happy SQLing,
Ms. SAS® Answers

CONCLUSION

Structured Query Language is an extremely powerful tool that you can use to query your data. Knowing the differences between the SQL procedure and the DATA step enables you to make a good decision when choosing a technique for managing data. Understanding the optimization techniques employed by PROC SQL helps structure

your data and indexes to take advantage of the power of SQL. Learning PROC SQL options and SAS system options that provide additional functionality can increase your comfort with submitting SQL code.

Ms. SAS® Answers may not exist in real life, but there are many places that you can find assistance. When you have questions, help is just a phone call, e-mail, or Web search away. Contact SAS Technical Support with questions, or check out SAS training that is offered on the Web or in the classroom. Happy efficient SQL coding!

CONTACT INFORMATION

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

Linda Jolley, Technical Training Specialist
SAS Institute Inc.
Kansas City Regional Office
Phone: (913) 491-1166
Email: linda.jolley@sas.com

Jane Stroupe, Technical Training Specialist
SAS Institute Inc.
Chicago Regional Office
Phone: (847) 367-7216
Email: jane.stroupe@sas.com