

## Paper 44-28

**Make Your SAS/ACCESS® Query More Efficient**

Jianming He, Dinesh Jain, Cheng Wang, Ph.D.  
Quest Diagnostics, Inc., Lyndhurst, NJ 07071

**ABSTRACT**

This paper discusses how to query from relational databases efficiently by using SAS/ACCESS. It integrates the idea that database developers perform data query into SAS environment. It also shows some tips for SAS programmers doing data extraction and data manipulation.

**INTRODUCTION**

For some SAS/ACCESS® beginners, they can write elegant SQL scripts for their tasks. Sometimes to their surprise, the SQL scripts do not work as they expected. The query might run forever without anything coming back. However, the database servers might be in jeopardy since the query occupies a lot of computer resources. The problems could be a table scan, improper index, etc.

Extracting large data from relational database (RDB) requires knowledge more than PROC SQL procedure. It needs comprehensive understanding of the RDB. SAS/ACCESS® provides a good tool set for data warehousing. SAS programmers should have sufficient knowledge and experience as database developers do in order to query database efficiently.

Though there are some literatures talking about retrieving data from RDB by SAS/ACCESS, most of them skip the part of RDB programming. With a mind of the database developer performing data query, this paper will discuss the data query strategy briefly<sup>1</sup>. It will show some tips that might be useful for a SAS/ACCESS® programmer by using examples. All examples are for SYBASE environment. Interested readers may easily move them to other relational database environment.

**DATA QUERY METHODOLOGY**

A SQL script is not simply joining with several tables. To optimize the performance, it needs good understanding of how queries are handled and how queries are executed inside the database. This section describes the query execution process and techniques to write an efficient query.

**1) Query Execution Process inside Database server**

Once the query is passed to the database server from the SQL Procedure Pass-Through Facility, the database server parses and normalizes the query. The parser ensures that the SQL syntax is correct. Normalization ensures that all the objects referenced in the query exist. Permissions are also checked during this process to ensure that the user has permission to access all tables and columns in the query. Query preprocessing changes some search arguments to an optimized form and adds optimized search arguments and join clauses.

If no errors are found, the parsed query is passed to the query optimizer. The query optimizer uses statistics about the tables, indexes and columns named in the query, and predicts the cost of using alternative access methods to resolve a particular query. The output of the optimizer is the query plan – the plan that is least costly in terms of I/O. The query plan is compiled with the

code that contains the ordered steps to carry out the query, including the access methods (table scan or index scan, type of join to use, join order, and etc.) to access each table. Once the query execution plan is complete, the database server executes the query and returns the data to the SQL Procedure Pass-Through Facility.

**2) Working with the optimizer**

The goal of the optimizer is to select the access method for each table that reduces the total time needed to process a query. The optimizer bases its choice on the statistics available for the tables being queried and on other factors such as cache strategies, cache and I/O size. A major component of optimizer decision-making is the statistics available for the tables, indexes and columns.

In some situations, the optimizer may seem to make the incorrect choice of access methods. This may be the result of inaccurate or incomplete information (such as out-of-date statistics). In other cases, additional analysis and the use of special query processing options can determine the source of the problem and provide solutions. The query optimizer uses I/O cost as the measure of query execution cost. The significant costs in query processing are the physical I/O (when pages must be read from disk) and the logical I/O (when pages in cache are read for a query). The two significant outputs from the query plan are the table access method and the order of table access.

**3) Table Access method in the optimizer**

For each table in the query the optimizer tries to determine the best path by choosing from the table scan, the index scan (use clustered index) or covering index scan (use non-clustered index).

Other than situations where the table scan is cheaper than the index scan, SAS programmers need to avoid the table scan as much as possible. During the table scan process, the database server reads every data page in the table because no useful indexes are available to help retrieve the data it needs. Every data page access needs disk read, which in turn causes poor response time from the system. It also affects the performance of other queries on the server.

To avoid the table scan for the query, SAS programmers can give hint to the database by choosing a proper index inside the SQL query. This feature can be used when multiple indexes are available on the table and the optimizer picks index based on its statistics, which may not be the fastest one for your query. A hint suggests that the optimizer use the index defined in the query instead of index based on its algorithm.

**4) Order of Table access defined by optimizer**

If multiple tables are accessed through the SQL query, the optimizer defines the sequence in which all tables will be accessed. This order is the most important factor for the performance of a query. The table accessed first returns a set of rows based on the condition defined in the "where" clause. The rest of the tables are accessed iteratively based on the results returned by the first query.

SAS Programmers need to make sure of two things in this situation. The table accessed first should be a qualifying table for the query and should return less number of rows or maximum

<sup>1</sup> For further investigation, readers may refer to any database developer's handbook. Please see the reference.

number of result rows. The tables accessed forward should not use table scan and must use a proper index scan, because these tables will be accessed iteratively for the result rows from the first table. For example, assume there are two tables in a query and the first accessed table from the database returns 1000 rows and the second accessed table performs a table scan for every row from the first table. This means that on the second table, the table scan will be performed 1000 times which is not a good thing for the data warehouse, if the number of rows in the second table are significant.

In some cases, the number of qualifying tables may be more than one, where data fetched from first table returns a few rows and the second table returns most of the qualifying results for other tables.

### 5) The definition of a table order in a query

If multiple tables are involved in a query, there are three ways a programmer can control the order of the table in the query.

Firstly, control the order by defining the hint for the tables. The optimizer creates a query plan based on statistics and hints given in a SQL query. If a hint tells the optimizer to use a different index other than specified in its' query plan, it forces the optimizer to re-evaluate everything and reorder table accesses. One can repeat this trick for every table in the query to get a query plan in the right order.

Secondly, use the manual query implementation technique. When performing a join, the query optimizer evaluates all reasonable joins, permutations and estimates the total I/O costs. The number of possible joins is a permutation of the number of tables. The more table joins, the more time the optimizer spends on calculation. In practice, we do not need to join all the required tables at once. A series of less than four tables join, may save a lot of time.

Finally, use the FORCEPLAN option. If none of the previous tricks are effective in getting the query plan in the way you want, then use FORCEPLAN setting for the query. This option once set forces optimizer to ignore everything and create the query plan based on the order and indexes defined in the clause of SQL query. If indexes are defined in the query, optimizer can use its own algorithm to choose the right index for the table. In any case optimizer cannot change the order of the table.

## SOME TIPS ON EXTRACTION DATA USING SAS

SAS always provides more than one solution for a job. To perform a table join in SAS/ACCESS, mostly we use the SQL Procedure Pass-Through Facility, or use ODBC engine treating the database table as a sas data set. After the system environment is set up, we can directly parse the SQL scripts generated following the above method. Since the SAS programmers are going to manipulate data mining after the data pull, in a large database, there are some issues for SAS programmers to think about, like storage space and working space problems, etc.

### 1) Break the data pull into small pieces

The purpose for SAS programmers to pull data from a database is for further data mining and reporting. For the convenience of later manipulation, it is always a good idea to break a big data pull into small data pulls and executes the data pull piece by piece. The advantage is to resume the data pull easily if there is a problem within the database. Small datasets also cost less space and time to manipulate like sorting.

In the SQL Procedure Pass-Through Facility using SAS/MACRO, we may break the data pull into smaller pieces. For instance, for

a data pull for a month, we can break it up by week, so we have four repeated data pulls.

Assuming we have two tables in a database. Table Beta has a field called date\_of\_service. A common key links table Alpha and Table Beta. Our task is to pull one month data. The code can be written as following.

```
proc sql;
connect to sybase
(server= &server database=&database
user=&user password=&pwd);
%macro pull(i, startdate,enddate);
create table data&i
as
select *
from connection to sybase
(
select a.product1, a.product2, b.date_of_service
from alpha a,
beta b
where
a.key=b.key
and b.date_of_service between &startdate
and &enddate
);
%mend;
%pull(i=1,startdate=%nrstr('20030321'),
enddate=%nrstr('20030331'));
```

Below are some notes on this approach.

Firstly, do not assume that if one query returns one month data in X minutes, that the same query, just by changing dates will return three months data in 3X minutes. Different data ranges forces the optimizer to use different indexes and a different order for the tables, which affects query performance.

Secondly, avoid the greater than operator in the range queries, instead of that operator use greater than equal for the operator. The greater than operator starts reading pages from the value defined in the query and scans all the pages until it reaches the page where it finds the value greater than the value in the where clause. Greater than equal to operator always starts from the right values and hence avoids page scans and saves resources. Also, as stated by the SAS online document, it is more efficient to use "BETWEEN" where it applies.

Thirdly, use single quotes when invoking the macro function. If using the double quotation mark, for both variables "startdate" and "enddate", from the above example, resolve to unexpected strings. SAS is looking for a column name like quoted string, like '20030331' other than searching for date like 20030331. Double quotation mark is not well resolved.

### 2) Using the temp table of a relation database

To expedite a data pull, it is common to create a temporary table in the relation database. SAS/ACCESS provides a lot of tools solving such a problem, like PROC DBLOAD, the SQL Procedure Pass-Through Facility, or the ODBC engine, etc.

While using PROC DBLOAD or ODBC engines, make sure that join columns in the query are of the exact same type. Different data type columns in join may force the database to perform a table scan.

The most convenient way is to use ODBC engine. Data step and any SAS procedure which produces output dataset could be used for creating a table. To get the right type of field, DBTYPE option may be used. There are also some other methods to change the data type in the database. We may choose from either the SQL Procedure Pass-Through Facility or PROC DBLOAD type

options.

The option BULKCOPY can save a lot of time if we are going to insert huge amounts of data into the database. This utility groups rows so that they are inserted as a unit into a SYBASE table. The table loading process performance can be significantly increased.

```

Libname mydblib sybase database=tempdb server=&server
user=&user password=&password ;

/* Indata is a sas dataset with a key variable */;
/* Master is the table in database to pull variables from */;
/* Outdata is the sasdata set for output */;
/* Pullvar is the variable in database we are interested */;

%macro pull(indata, key, master, outdata, pullvar);

proc sort data=&indata(keep=&key where=(&key=*))
nodupkey out=mydblib.temp (bulkcopy=yes
dbtype=(&key='data type'2)); by &key ;

proc sql;
connect to sybase
(server= &server database=&server user=&user
password=&pwd);
execute (create index ind on tempdb..temp (&key))
by sybase;

create table &outdata as
select *
from connection to sybase
(
select
&pullvar
from &master b,
tempdb..temp a (index ind)
where a.&key=b.&key
);
quit;
proc delete data=mydblib.b;
%mend;

```

In the above example, it tries to upload a SAS data set into the database "tempdb" to create a table called "temp", creates an index on the new table and joins with another existing table in the relational database.

### 3) Other tips for data extraction

Here are some tips worth mentioning:

- Always check the query plan first for the query before running.
- Keep statistics about data extraction times for the query. It will work as a guideline for the rest of the group.
- If you have multiple good query plans, then choose one that uses the maximum number of processes. Multiple processes run parallel inside database.
- Index definition in the SQL query does not force optimizer to use that index, it is just a hint to the optimizer. When you are forcing indexes, the query plan may be misleading. Some other database tools should be used to make sure that the indexes shown in query plan are used, when query is run.
- Notify DBA instantly if any SAS/ACCESS session is killed in client's side. Otherwise, the SAS/ACCESS is still running in the server.
- Always deleting the temporary table after you finish the data pull.

- Since order by, group by and distinct clause in the PROC SQL statement incurs additional processing and I/O, we suggest to extract the data first and perform the data summary in the next stage. To save the large data extraction time, data manipulation can be put into SAS environment. The Database also needs temporary statistics to merge that information back to the original data.
- If the search argument is not indexed, one solution is that we may ask DBA to create one temporarily. If there is sufficient space and we perform such a query frequently, it might be a good idea to download the whole table into SAS dataset.
- If having space, a SAS data mart with longitudinal data can be created for research convenience.

## SUMMARY

It is proven that SAS /ACCESS® is a very convenient tool for data warehousing. SAS programmers need to be familiar with both RDB warehousing techniques and SAS coding skills to have their job done. As we have seen, deep research on the query can save SAS programmers or database valuable time. A good habit of SAS/ACCESS program coding also helps other people working in the database using other tools.

## Reference:

- 1) Rankins, R., et al (1996), *Sybase SQL Server 11 unleashed*, Indianapolis, IN: Sams publishing.
- 2) Loren, J. (1996), "SAS Connections to DB2: Tools and Techniques", *Proceeding of the Twenty-First Annual SUGI Conference*, 21,498-507
- 3) SAS institute Inc. (1999), *SAS Online Document*®, Cary, NC: SAS Institute Inc.
- 4) Wang C., He J., Mallon P. (2001), "The application of SAS data mart in Clinical Laboratory Testing", *PharmSUG 2001: Conference Proceedings*

## ACKNOWLEDGMENTS

Thanks Darlene Garaffa for proofreading  
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. Other brand and product names are trademarks of their respective companies.

Jianming He  
Quest Diagnostics  
1200 Wall St. west  
Lyndhurst, NJ 07071  
(201)729-8716

Dinesh Jain  
Quest Diagnostics  
1200 Wall St. west  
Lyndhurst, NJ 07071  
(201)729-7524

Cheng Wang, Ph.D.  
Quest Diagnostics  
1200 Wall St. west  
Lyndhurst, NJ 07071  
(201)729-8694

<sup>2</sup> Check the data type of the variable in your database