Paper RF-02-2013

Does foo Pass-Through? SQL Coding Methods and Examples using SAS® software

Stephen Crosbie, UM (Universal McCann), Birmingham, MI

ABSTRACT

Rapid Fire: Processing data between SAS® software and a DBMS, such as Netezza® or DB2®. Application of the SQL Procedure with the CONNECT Statement (requires SAS/CONNECT® software) and use of the PROC SQL Pass-Through Facility.

While a SAS library reference gives PROC SQL access to DBMS tables (requires SAS/ACCESS® software), using Pass-Through SQL coding instead can often provide faster results. This paper provides useful examples of queries that were passed-through to a DBMS to take advantage of greater processing resources or better query optimization.

INTRODUCTION

This paper includes fundamental information about pass-through SQL for SAS programmers analyzing data stored outside the SAS environment. SAS/ACCESS software for the specific DBMS must be installed for the code to work.

Sections include definitions and various examples of pass-through SQL. Examples follow the best practice of ensuring that data processing is pushed-down to the DBMS as much as possible, before returning data to SAS (Dinsmore et al).

DBMS DATA ACCESS METHODS

This section includes basic examples of SAS code and pass-through SQL. A library reference or the CONNECT statement (as part of the SQL procedure) are valid ways for SAS to retrieve data from a DBMS.

IMPLICIT PASS-THROUGH SQL (USING A LIBRARY REFERENCE)

One easy way to access data from a DBMS is to use the LIBNAME statement:

```
options sastrace=',,,d' sastraceloc=saslog;
LIBNAME nz netezza server=&NZServer database=&NZdb user=&NZUser password=&NZPass
BULKUNLOAD=YES /*connection=global*/;
```

Using the above library reference (libref), the DATA step or SQL procedure can build a SAS data set—as illustrated in Figure 1. SAS/ACCESS Interface for Netezza executes both of these examples in the same way:

1. Open a connection using the settings listed in the libref (assuming the global connection was not specified)
2. Translate the request into DBMS-native SQL and send for in-DBMS processing (where possible)
3. Receive the output from the DBMS and temporarily store it when further processing by SAS is needed
4. Write the data set or output the results

The "behind-the-scenes" translation in step 2 above is known as "implicit pass-through SQL" (Capobianco, p.1)

DATA step with libref to a DBMS Table	PROC SQL with libref to a DBMS Table

`DATA fig_1a;`
`   set nz.srctable;`
`run;`
	`PROC SQL;`
	`   create table fig_1b as`
	`   select *`
	`   from nz.srctable ;`
	`%PUT &SQLXRC; %PUT &SQLXMSG;`
	`QUIT;`

Figure 1. Examples of implicit pass-through SQL (using a library reference)

Although outside the scope of this paper, it is important to note that the SAS/ACCESS interface enables many procedures to efficiently process "in-database" via a libref (SAS Institute 2012, p.74). With SAS/ACCESS version 9.3, even more DBMS interfaces are available, including Hadoop—a "big-data" platform (SAS Institute 2012, p.xi).

EXPLICIT PASS-THROUGH SQL (THE CONNECT STATEMENT)

Similar to a library reference, PROC SQL can be used with the CONNECT statement to retrieve DBMS data, as seen in Figure 2. The CONNECT statement uses similar SAS/ACCESS settings as the libref; however, "netezza" is an alias to the specific DBMS connection made.

In this example, the query syntax between the parentheses is passed directly to the DBMS—without translation. In other words, the query is passed-through "explicitly."

Explicit Pass-Through SQL to Manipulate Data

```
PROC SQL noerrorstop;
connect to netezza(server=&NZServer database=&NZdb user=&NZUser
                   password=&NZPass BULKUNLOAD=YES);
create table fig_2 as
select * from connection to netezza
(select * from srctable
);
%PUT &SQLXRC; %PUT &SQLXMSG;
disconnect from netezza;
QUIT;
```

Figure 2. Explicit pass-through SQL used to extract DBMS data to a SAS data set

A slightly different form of explicit pass-through SQL uses the EXECUTE statement within PROC SQL to allow any valid SQL commands to be passed to the DBMS, as seen in Figure 3. SQL code from a DBMS query tool such as Aginity (for Netezza) or AQT (for DB2) can be pasted inside the EXECUTE statement.

In this example, the syntax does not allow for query results to be returned to SAS.

Explicit Pass-Through SQL for DBMS Execution

```
PROC SQL noerrorstop;
connect to netezza(server=&NZServer database=&NZdb user=&NZUser
                   password=&NZPass /*BULKUNLOAD=YES*/);
execute(
create temporary table fig_3 as /*NOTE: TABLE CREATED INSIDE DBMS*/
select count(*) as cnt
from srctable
) by netezza;
%PUT &SQLXRC; %PUT &SQLXMSG;
disconnect from netezza;
QUIT;
```

Figure 3. Explicit pass-through SQL used to send DBMS execution instructions

While executing command-line code such as RUNSTATS on a table is beyond the scope of this paper, note that explicit pass-through SQL cannot accomplish this. A properly configured SAS server can do it via the X command.

EFFICIENT PASS-THROUGH SQL EXAMPLES

While the examples from Figures 1 and 2 perform the simple task of creating a SAS data set from one DBMS table, tasks requiring more data manipulation on the DBMS-side are most efficiently written with some form of explicit pass-through SQL. The examples in this section emphasize the best practice of completing DBMS-side processing before returning results to SAS (Dinsmore et al).

AVOIDING A SAS-NATIVE FUNCTION IN A WHERE CLAUSE

A well-known foible of implicit pass-through SQL is that not all SAS-native functions have translations to the DBMS-native SQL. When no translation exists for a function located in the where clause, SAS/ACCESS usually downloads the entire table from the DBMS for local processing and takes a really long time to return results (Rhoads, p.10).

Even when SAS translates the function properly, hiccups can prevent obtaining the desired result. In this case, explicit pass-through SQL should be used with the equivalent SQL function to ensure DBMS-side processing.

In this example, SAS returns a count of 0 (zero) records for the implicit pass-through SQL that uses the SAS-native INDEX function; however, the explicit pass-through SQL using the Netezza-native strpos function works as expected. Upon further inspection of the log (using the SASTRACE option), the INDEX function was translated properly.

Implicit Pass-Through SQL	Explicit Pass-Through SQL

PROC SQL;
select count(*) as cnt_recs
from nz.internetHits
where hitmonth="01JUN2013"d
AND
INDEX(DOMAIN_NAME,'mwsug.org')>0
;
%PUT &SQLXRC; %PUT &SQLXMSG;
QUIT;
	PROC SQL noerrorstop;
	connect to
	netezza(server=&NZServer
	database=&NZdb user=&NZUser
	password=&NZPass BULKUNLOAD=YES);
	select * from connection to
	netezza
	(select count(*) as cnt_recs
	 from nz.internetHits
	 where hitmonth='2013-06-01'
	AND
	strpos('mwsug.org',DOMAIN_NAME)>0
	);
	%PUT &SQLXRC; %PUT &SQLXMSG;
	disconnect from netezza;
	QUIT;

RESULT: 0	**RESULT: 483**

Figure 4. SAS-native function inhibits results, while explicit pass-through SQL does not

KEEPING TABLE CREATION ON THE DBMS-SIDE

In this example, a DBMS table is created from a join of two other DBMS tables. The approach with the DATA step will take much longer because SAS must download and sort each table, and then upload the new table back to the DBMS. It makes sense to use explicit pass-through SQL when all the data to be processed resides on the DBMS.

SAS-Side Processing (DATA step)	Explicit Pass-Through SQL

PROC SORT data=nz.srctable1
           out=srctable1;
   by thekey;
run;
PROC SORT data=nz.srctable2
           out=srctable2;
   by thekey;
run;
DATA nz.fig_5a;
   merge srctable1 (in=a)
      srctable2 (in=b);
   by thekey;
   if a AND b;
run;
	PROC SQL noerrorstop;
	connect to
	netezza(server=&NZServer
	database=&NZdb user=&NZUser
	password=&NZPass
	BULKUNLOAD=YES);
	execute(
	create table fig_5b as
	select a.*
	       , b.*
	from srctable1 a
	   inner join srctable2 b
	      using(thekey)
	) by netezza;
	%PUT &SQLXRC; %PUT &SQLXMSG;
	disconnect from netezza;
	QUIT;

Figure 5. SAS-side processing is slow compared to explicit pass-through SQL

PROCESSING BETWEEN TWO DBMS USING SAS

The following example includes a mixture of coding methods illustrating how SAS can work between two DBMS, taking advantage of their processing power. The objective of this code was to apply suppressions to a large contact list stored in one DB2 server, using a small-sized suppression list from a separate DB2 server.

```
libname cust db2 dsn=&myDSN uid=&DB2User pwd=&DB2Pass; /*Customer database*/
libname calls db2 dsn=&myDSN uid=&DB2User pwd=&DB2Pass; /*Call-center database*/
/*LOAD SMALL SUPPRESSIONS LIST INTO BIG CUSTOMER DATABASE*/
DATA cust.list_suppress;
   set calls.list_suppress;
run;
```

```
/*CREATE A NEW TABLE WITH CUSTOMERS APPROVED FOR CONTACT*/
PROC SQL noerrorstop;
CONNECT to db2 (NOPROMPT="DSN=&myDSN;UID=&DB2User;PWD=&DB2Pass;");
execute(
create table SCHEMA.APPROVED_CONTACT (CUSTOMER_ID DECIMAL(10,0)) in myTABLESPACE
) by db2;
execute(
insert into table SCHEMA.APPROVED_CONTACT (
   select foo.*
   from LIST_CUSTOMERS foo
      left join LIST_SUPPRESS bar on foo.CUSTOMER_ID=bar.CUSTOMER_ID
   where coalesce(bar.CUSTOMER_ID,99999)=99999
   )
) by db2;
DISCONNECT from db2;
QUIT;
```

CONCLUSION

Explicit pass-through SQL is an essential part of SAS/ACCESS that enables programmers to process data within another DBMS and return results to SAS for further manipulation. Even when a programmer writes a DATA step to access DBMS data, implicit pass-through SQL is created behind-the-scenes by SAS.

For the programmer seeking efficient processing of data, the constant challenge is to follow best practices by ensuring that data manipulation on DBMS tables is done in the DBMS, wherever possible.

REFERENCES

- Capobianco, Frank. Explicit SQL Pass-Through: Is It Still Useful? SGF 2011 Proceedings, Paper 105-2011
  http://support.sas.com/resources/papers/proceedings11/105-2011.pdf
- Dinsmore et al. “Leveraging IBM Netezza Data Warehouse Appliances with SAS: Best Practices Guide for SAS Programmers”
  http://www.sas.com/partners/directory/ibm/NetezzaDWAppliances-withSAS.pdf
- Rhoads, Mike. Avoiding Common Traps When Accessing RDBMS Data. SGF 2009 Proceedings, Paper 141-2009
  http://support.sas.com/resources/papers/proceedings09/141-2009.pdf
- SAS Institute Inc. 2012. *SAS/ACCESS® 9.3 for Relational Databases: Reference, Second Edition*. Cary, NC: SAS Institute Inc.
  http://support.sas.com/documentation/cdl/en/acreldb/65247/PDF/default/acreldb.pdf

CONTACT INFORMATION

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

Name: Stephen Crosbie
Enterprise: UM (Universal McCann)
Address: 805 E. Maple Rd
City, State ZIP: Birmingham, MI 48009
Work Phone: 248-554-4374
Fax:
E-mail: Stephen.crosbie@umww.com
Web: http://www.umww.com