Paper SA-03-2014

SAS® PASSTHRU to Microsoft SQL Server using ODBC

Nina L. Werner, Madison, WI

ABSTRACT

I wish I could live in SAS World; I do much of my data analysis there. However, my current environment houses all production data in Microsoft SQL Server 2008 R2 data warehouses. Microsoft SQL Server is a relational database with its own dialect of SQL (T-SQL or Transact-SQL.) SAS/ACCESS® Interface to ODBC has several methods of retrieving SQL Server data for analyzing in SAS. Here's an introduction to both implicit and explicit PASSTHRU (or, Pass-Through) with some hints for deciding which to apply. Explicit PASSTHRU will execute T-SQL code syntax inside SQL Server, while implicit PASSTHRU will use your SAS DATA step or PROC SQL code. Several useful options will be explained. No prior experience with ODBC is necessary.

INTRODUCTION

Open Database Connectivity, or ODBC, is the open standard application programming interface (API) which allows SAS to communicate with many databases using an ODBC driver for each database which you define in the operating system. Using ODBC, SAS will give you the same permissions (read only or read/write, etc.) that you have natively in the database. For Windows, use the Data Sources ODBC in System Tools to create and name these drivers; UNIX requires coding (see Bailey, 2014.) Any ODBC-compliant application can access any DBMS for which a driver is installed. While SAS offers targeted SAS/ACCESS Interface modules for many databases such as SAS/ACCESS Interface to Oracle or Teradata, and even SAS/ACCESS Interface to Microsoft SQL Server for UNIX, SAS/ACCESS Interface to ODBC is popular because it is cost-effective to license this one product instead of one for each of your multiple DBMS sources. You can even use ODBC to access Excel spreadsheets if you do not have a license for SAS/ACCESS Interface to PC File Formats or operate outside of Windows. ODBC 1.0 was released by Microsoft in 1992. Although Microsoft developed OLE/DB after ODBC, in recent years it has been deprecated in SQL Server (Factor[1], 2011.) Factor says, "I always feel slightly awkward in talking about ODBC. It is a Once and Future technology, developed before its time, but now showing its value for processing large volumes of data, despite its quirks, poor documentation and lackluster support." (Factor, 2013.) There have been many SAS papers on the subject of SAS/ACCESS Interfaces, so be sure to look over the references.

ISSUES TO KEEP IN MIND

The first issue you may encounter when connecting to SQL Server via ODBC is the 32-character limit for table names. SQL Server tables with longer names will be invisible. If you have authority on the database, you could briefly rename your table to something shorter, read it into SAS, and go back to SQL Server Management Studio to change the name back before anyone noticed. A better method is to CREATE a VIEW of the table with a shorter name in SQL Server. If you do not have authority to do it yourself, you will have to request that your DBA does it for you. I have seen the problem with names that are exactly 32 characters long, so I recommend a new name shorter than 32 characters. It is true that by using a VIEW, you lose access to the indexes on the table, so you might prefer to copy a large, indexed table with a shorter name. This is beyond the scope of this introductory paper, but optimization of your queries is an important subject to address after the basics.

SAS Missing values are not the same as SQL Server NULLs. Investigate how NULL values respond in your queries.

EFFICIENCIES

In order to limit network traffic between SAS and SQL Server, your queries should return only columns needed for further processing in SAS. Let SQL Server select only the rows you need; avoid pulling the entire table into SAS only to discard many rows. Since SQL Server tables are indexed, perform joins there if possible for most efficient processing. Discussion of these important topics must wait for future papers, alas.

[1] You have to agree that *Phil Factor* is a great pen name for a SQL Server expert.

IMPLICIT VS. EXPLICIT PASSTHRU

Implicit PASSTHRU uses SAS syntax in either a DATA step or PROC SQL with an assigned LIBNAME to access the external data source. Explicit PASSTHRU wraps T-SQL SQL Server code with CONNECT TO ODBC, so the query executes in the database and the result set is passed back to SAS. For non-SELECT T-SQL statements, you must use explicit PASSTHRU. Explicit PASSTHRU is also preferable for complex queries combining multiple tables using SQL Server native functions.

ODBC LIBNAME FOR IMPLICIT PASSTHRU

The easiest way to use ODBC is to assign a SAS LIBNAME to each of your SQL Server server/database ODBC Data Sources. If you need to access multiple schemas from a database, you must set a different LIBNAME for each one. As you know, any SAS LIBNAME assigned *name* must be eight or fewer characters and start with a letter or underscore. The ODBC Data Source name can be longer; therefore, your LIBNAME *name* does not have to match the Data Source name. [Similarly, the Data Source name does not have to match your SQL Server database name; it is just a convenient alias.] It is referenced in the LIBNAME statement with the DATASRC option. The example below is **SQLdb**. Also required in the LIBNAME statement is the SCHEMA option.

Using Windows authentication for my SQL Server databases, I can omit the PASSWORD option and use this SAS automatic macro variable to pass my user credentials instead of having to type them in, USER=&SYSUSERID. You might prompt for credentials or encrypt and store them[2] (Werner, 2006.) For SQL Server, I must always use PRESERVE_TAB_NAMES=YES option which “specifies that table names are read from and passed to the DBMS with special characters, and the exact, case-sensitive spelling of the name is preserved” and CONNECTION=SHARED option which “specifies that all operations that access DBMS tables in a single libref share a single connection.” The final two options I always include on my ODBC LIBNAME specify buffer sizes, INSERTBUFF and READBUFF. Optimal values are based on trial and error using real network connections; your mileage may vary. For ad hoc querying, I tend to use these reliable, successful values and not spend time tuning for the best possible values until the job is too slow or cannot complete. Buffer sizes depend on system memory as well as record size and row count.

Here are LIBNAMES for two schemas on the same server and database:

```
LIBNAME   sqldb   ODBC   DATASRC=SQLdb   SCHEMA=dbo                USER=&SYSUSERID
     PRESERVE_TAB_NAMES=YES   CONNECTION=SHARED   INSERTBUFF=1000   READBUFF=2500;
LIBNAME    nlw    ODBC   DATASRC=SQLdb   SCHEMA="CORP\nlwerner"    USER=&SYSUSERID
     PRESERVE_TAB_NAMES=YES   CONNECTION=SHARED   INSERTBUFF=1000   READBUFF=2500;
```

BONUS: DO YOU KNOW THE PROC SQL (SAS) FUNCTION MONOTONIC()? TRY IT.

MONOTONIC() is the PROC SQL equivalent of coding a new variable from _N_ in a DATA step. It is not a SQL Server T-SQL function. Find it in the last implicit PASSTHRU sample code coming up on the next page. The T-SQL function is IDENTITY(INT,1,1) . Find it in the last explicit PASSTHRU sample code.

```
PROC SQL;
SELECT MONOTONIC() AS ord,  /* creates a column containing row number called ord */
...

DATA a;  SET b;
  ord = _N_;    * creates a column containing row number called ord;
RUN;
```

[2] Ask for my code from **Get Vital Info: Relational Database Password Protection via SAS Macros**.

IMPLICIT PASSTHRU USES SAS DATA STEP OR PROC SQL WITH ODBC LIBNAME

Implicit PASSTHRU uses SAS syntax in either a DATA step or PROC SQL with an assigned LIBNAME to access the external data source. You can combine SQL Server and SAS tables by using multiple LIBNAMES. Caution: using OUTER JOIN or some SAS functions which do not translate into T-SQL in the WHERE clause may cause the entire SQL table to be transported into SAS. Very inefficient.

Read from SQL Server into SAS in DATA step:

```
LIBNAME sqldb ODBC DATASRC=SQLdb SCHEMA=dbo USER=&SYSUSERID
     PRESERVE_TAB_NAMES=YES CONNECTION=SHARED INSERTBUFF=1000 READBUFF=2500;

DATA workcopySASFemales1;
SET sqldb.big_SS_table1  (KEEP=idnum col3 col22 SSdatecol  WHERE=(col10='F') );
RUN;
```

Write from SAS to SQL Server in DATA step:

```
LIBNAME sqldb ODBC DATASRC=SQLdb SCHEMA=dbo USER=&SYSUSERID
     PRESERVE_TAB_NAMES=YES CONNECTION=SHARED INSERTBUFF=1000 READBUFF=2500;

DATA sqldb.new_SS_table_Females_only;
SET workcopySASFemales1;
FORMAT load_date DATETIME.;
* SAS date into SQL Server DATETIME with proper conversion;
load_date = %SYSFUNC(INPUTN( %SYSFUNC (PUTN( %SYSFUNC(dhms(
            %SYSFUNC(INPUTN(&SYSDATE,DATE7.)),0,0,0)), DATETIME.)), DATETIME.));
RUN;


DATA sqldb.'##zips'N (BULKLOAD=YES); * write to SQL Server tempdb;
SET SAShelp.zipcode;
ord = _N_;    * creates a column containing row number called ord;
RUN;
```

For this to succeed, your userid must have write permission as dbo schema on the database defined as Data Source SQLdb.

Read from SQL Server into SAS in PROC SQL:

```
LIBNAME sqldb ODBC DATASRC=SQLdb SCHEMA=dbo USER=&SYSUSERID
     PRESERVE_TAB_NAMES=YES CONNECTION=SHARED INSERTBUFF=1000 READBUFF=2500;

PROC SQL;
CREATE TABLE workcopySASF2 AS
  SELECT idnum, col3, col22, SSdatecol
   FROM sqldb.big_SS_table1
   WHERE col10='F';
QUIT;
```

Write from SAS to SQL Server in PROC SQL:

```
LIBNAME sqldb ODBC DATASRC=SQLdb SCHEMA=dbo USER=&SYSUSERID
     PRESERVE_TAB_NAMES=YES CONNECTION=SHARED INSERTBUFF=1000 READBUFF=2500;

PROC SQL;
CREATE TABLE sqldb.new_SS_table_F2 AS
  SELECT MONOTONIC() AS row_number, idnum, col3, col22, SSdatecol
   FROM workcopySASF2;
QUIT;
```

For this to succeed, your userid must have write permission as dbo schema on the database defined as Data Source SQLdb.

EXPLICIT PASSTHRU USES PROC SQL WITH CONNECT TO ODBC, T-SQL SYNTAX

Explicit PASSTHRU wraps T-SQL SQL Server code with CONNECT TO ODBC / DISCONNECT FROM ODBC, so the query executes in the database and the result set is passed back to SAS. This can be much more efficient than implicit PASSTHRU if you are combining multiple database tables with complex conditional logic and subsetting the result set.

For non-SELECT T-SQL statements, you must use explicit PASSTHRU. All purple text below is T-SQL syntax for SQL Server to parse and execute.

You can display the return code and message from SQL Server for each query when using explicit PASSTHRU through the SAS automatic macro variables &SQLXRC and &SQLXMSG.

Here is the SAS wrapper code that you need for explicit PASSTHRU to write a SAS dataset:

```
PROC SQL;
CONNECT TO ODBC(DATASRC=SQLdb USER=&SYSUSERID) ;

/* Explicit PASSTHRU with SELECT */
CREATE TABLE workcopySASother AS
SELECT *
FROM CONNECTION TO ODBC (
   SELECT b.idnum b.col3 b.col22 o.[SSdatecol] AS mydate
   FROM dbo.big_SS_table1 b
   LEFT JOIN dbo.other_SStable o
          ON b.idnum = o.memberid
   WHERE o.otherdatecol >= '2014-10-06'
   --This is a T-SQL comment that works inside SQL Server
                             ) ;
;
%PUT &SQLXRC. &SQLXMSG. ; * SQL Server query return code and message;
DISCONNECT FROM ODBC ;
QUIT;
```

Here is the SAS wrapper code that you need for explicit PASSTHRU non-SELECT and combined statements:

```
PROC SQL;
CONNECT TO ODBC(DATASRC=SQLdb USER=&SYSUSERID) ;


/* Explicit PASSTHRU with non-SELECT T-SQL code, no SAS output */
EXECUTE (
   DROP TABLE dbo.new_Small_SS_table
   -- delete the old table first since this does not have REPLACE capability
           ) BY ODBC ;
%PUT &SQLXRC. &SQLXMSG. ;
;
EXECUTE (
   CREATE TABLE dbo.new_Small_SS_table ( ... )
           ) BY ODBC ;
%PUT &SQLXRC. &SQLXMSG. ;
;
EXECUTE (
   INSERT INTO dbo.new_Small_SS_table
   SELECT IDENTITY(INT,1,1) as newID, b.*
   FROM dbo.big_SS_table1 b
   LEFT JOIN dbo.other_SStable o
          ON b.idnum = o.memberid
   WHERE o.otherdatecol >= '2014-10-06'
           ) BY ODBC ;
;
%PUT &SQLXRC. &SQLXMSG. ;
DISCONNECT FROM ODBC ;
QUIT;
```

OPTIONS FOR MORE INFORMATION ABOUT DATABASE PROCESSING

SASTRACE=',,,d' reports what statements SAS has passed to SQL Server for CREATE, DROP, INSERT, DELETE, UPDATE, or SELECT, so you can confirm that your query is being properly translated. You can learn more about the other options available with SASTRACE. SASTRACELOC=SASLOG writes that information to the SAS Log. This can be very helpful if you have an issue with the result set being returned by your query and need to contact SAS Technical Support. NOSTSUFFIX limits the amount of unnecessary information written.

In the Windows environment, the WORK directory path created for your SAS session looks like *_TD9999*, where the temporary directory number "*9999*" is your Task ID. This is good to know, so I write my WORK path to the SAS Log every session. For example, if SQL Server times out passing rows to SAS, or your SAS window gets closed, there can be a zombie task still running in SQL Server with no connection for sending the result set back to SAS, wasting resources. If this happens, you will want to be able to report the Task ID to IT, so they can kill the zombie task in SQL Server.

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
%PUT >>> WORK path is %SYSFUNC(PATHNAME(WORK)) <<<;     /* in AUTOEXEC.SAS */
```

CONCLUSION

Microsoft SQL Server is a relational database with its own dialect of SQL (T-SQL or Transact-SQL.) SAS/ACCESS® Interface to ODBC has several methods of retrieving SQL Server data for analyzing in SAS. Use ODBC to access data in SQL Server databases, schemas, and tables. The simplest way is to use implicit PASSTHRU by assigning a SAS ODBC LIBNAME to your SQL Server database ODBC Data Source. You can get to SQL Server data by coding a DATA step or PROC SQL and accessing the LIBNAME with SAS syntax; no T-SQL knowledge needed. This is likely to be inefficient, but for smaller tables or single use data pulls, it works.

With sufficient user permissions on the server and database, you can CREATE, ALTER, DROP, INSERT, DELETE, UPDATE, or SELECT SQL Server tables from within SAS using explicit PASSTHRU. If you have JOIN logic combining many tables or complex WHERE conditions to return a small subset of data, explicit PASSTHRU coding is a better choice, only returning the result set to SAS. You wrap T-SQL code, which does the heavy lifting inside SQL Server, into your SAS program with FROM CONNECTION TO and EXECUTE BY ODBC.

Set system OPTIONS to see the SQL code being passed to the database, and be sure to check the SQL Server return code and message by writing them to the Log..

Create a Windows Data Source ODBC driver and get started with this code in your environment.

KEYWORDS

Nina L. Werner, SAS/ACCESS® Interface to ODBC, LIBNAME,
CONNECT TO ODBC, FROM CONNECTION TO ODBC, EXECUTE BY ODBC, DISCONNECT FROM ODBC,
&SQLXRC, &SQLXMSG, MONOTONIC(), SASTRACE, SASTRACELOC=SASLOG, NOSTSUFFIX

REFERENCES

- Bailey, Jeff. An Insider's Guide to SAS/ACCESS® Interface to ODBC. SGF 2014 Proceedings, Paper SAS039-2014 http://support.sas.com/resources/papers/proceedings14/SAS039-2014.pdf
- 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
- Crosbie, Stephen. Does foo Pass-Through? SQL Coding Methods and Examples using SAS® software. MWSUG 2010 Proceedings, Paper RF-02-2013 http://www.lexjansen.com/mwsug/2013/RF/MWSUG-2013-RF02.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
- Factor, Phil. Now you ODBC me now you don't. SQL Server Central, Editorial 2011/09/09 http://www.sqlservercentral.com/articles/Editorial/75947/Printable
- Factor, Phil. Getting Data between Excel and SQL Server using ODBC. Simple Talk 16 August 2013 https://www.simple-talk.com/content/print.aspx?article=1856
- He, Jianming, Jain, Dinesh, Wang, Cheng. Make Your SAS/ACCESS® Query More Efficient. SUGI 28 Proceedings, Paper 44-28 http://www2.sas.com/proceedings/sugi28/044-28.pdf
- Riley, Candice. Getting SAS® to Play Nice With Others: Connecting SAS® to Microsoft SQL Server Using an ODBC Connection. SGF 2008 Proceedings, Paper 135-2008 http://www2.sas.com/proceedings/forum2008/135-2008.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
- Rhodes, Dianne Louise. Talking to Your RDBMS Using SAS/ACCESS® . SGF 2007 Proceedings, Paper 239-2007 http://www2.sas.com/proceedings/forum2007/239-2007.pdf
- Wang, Xiaoqiang. # or ## - how to reference SQL server temporary tables? NESUG 2010 Proceedings, Paper CC30 http://www.lexjansen.com/nesug/nesug10/cc/cc30.pdf
- Werner, Nina L. Get Vital Info: Relational Database Password Protection via SAS Macros, SAS Users Group, Madison, WI, May 11, 2006.

ACKNOWLEDGMENTS

I am enormously grateful to Misty Johnson for encouraging me to submit this paper for consideration and making it possible for me to attend and present.

RECOMMENDED READING

SAS/ACCESS 9.4 for Relational Databases: Reference, Third Edition. Available at http://support.sas.com/documentation/cdl/en/acreldb/66787/PDF/default/acreldb.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Feel free to ask for more code snippets as you become more comfortable with ODBC. Contact the author at:

Name: Nina L. Werner
City, State: Madison, WI
Phone: 608 212-0689
E-mail: Ninja.Werner@me.com

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.