Paper BB-03-2015

Just passing through… Or are you?

Determine when SQL Pass-Through occurs to optimize your queries

Misty Johnson, State of Wisconsin Department of Health Services, Madison, WI

ABSTRACT

SAS/ACCESS® has two recommended methods for accessing data within a relational database management system (DBMS), namely, the SAS/ACCESS LIBNAME interface engine and the Structured Query Language (SQL) Pass-Through Facility. This paper describes the use of the open database connectivity (ODBC) LIBNAME engine with 9.3 SAS code that does and does not invoke implicit SQL pass-through and its effect on run time. Also described is the use of the system options DEBUG and SASTRACE to determine if implicit SQL pass-through occurred, what triggers implicit SQL pass-through and the potential time savings. Knowledge of these methods, their triggers, and tracking options enables the intermediate SAS programmer to select the most efficient coding strategy.

INTRODUCTION

SAS/ACCESS is a licensed product separate from base SAS that enables the user to communicate easily with a DBMS (Plemmons, 2010). SAS/ACCESS has two components: the SQL pass-through facility and the LIBNAME interface engine. The first component, the SQL pass-through facility, may be invoked in an explicit or implicit fashion. Explicit pass-through is invoked with a CONNECT statement within a PROC SQL statement. Explicit pass-through allows the user to write code in the SQL dialect native to the DBMS. Implicit pass-through is invoked by use of a SAS/ACCESS LIBNAME and SAS code that triggers implicit SQL pass-through.

The second component of the SAS/ACCESS software, the LIBNAME interface engine, enables the user to create a transparent interface to the DBMS, allowing the user to refer to the data source with a two-level filename. The LIBNAME engine, specific to the DBMS, also determines which queries can successfully be “passed through” to the DBMS for processing, thus saving time and resources. A SAS/ACCESS LIBNAME can be used to communicate with a DBMS without invoking implicit pass-through, thus processing the query completely on SAS. Implicit SQL pass-through only occurs when it’s triggered by an aggregate function or certain keywords, like DISTINCT.

Use of implicit SQL pass-through to process the query on the DBMS may be more efficient than processing in SAS; however, one must first have the knowledge of coding that invokes implicit SQL pass-through to achieve it. A broad overview of the ODBC LIBNAME engine within SAS/ACCESS, ODBC triggers of implicit pass-through, how to determine if implicit SQL pass-through occurred, and potential time-savings of implicit pass-through are the objectives of this paper.

BACKGROUND: SQL PASS-THROUGH FACILITY

The SQL pass-through facility allows SAS/ACCESS to send queries directly to the DBMS for processing. Most programmers are aware of the explicit method of the SQL pass-through facility and its, perhaps, “daunting” coding structure. Explicit pass-through can only be achieved within a PROC SQL statement. The explicit pass-through facility is triggered by the CONNECT statement within the SQL procedure as shown below.

```
LIBNAME OLD 'L:\BFS\CARSVOUCHERS';

PROC SQL;
    CONNECT TO ODBC (DATASRC=FMS_PROD USER=&SYSUSERID. DBPROMPT=YES);
            CREATE TABLE OLD.TABLE2 AS
            SELECT *
            FROM CONNECTION TO ODBC
                    /* THIS CODE PROCESSED ON DBMS */
(SELECT DISTINCT Contract_Year AS year, Profile_ID AS profid, Appn AS app, Proj AS
prj, Resp AS ra
                    FROM DBO.VOUCHER_CODING
                    WHERE CONTRACT_YEAR IN ('14','15','16','17'));
                    /* END CODE PROCESSED ON DBMS */
DISCONNECT FROM ODBC;
QUIT;
```

The code above first establishes a library reference, “OLD”, to a permanent file location. Next, the SQL procedure is invoked, but the first row of code is the CONNECT statement that explicitly tells SAS to establish a connection to a

DBMS, the DBMS engine to use, “ODBC”, and the connection options to use. Since this code will create a permanent table, the next line of coding is the CREATE TABLE clause, followed by the familiar SELECT and FROM clauses. However, instead of a table name in the FROM clause, we see a “CONNECTION TO” and the DBMS name, “ODBC”. The inner query describes the query processed on the DBMS. The outer query selects all columns from this query, and the result is written to the permanent table “TABLE2” in the library named “OLD”.

This query is explicitly sent to the DMBS for processing, is pretty efficient, but it requires more coding than the LIBNAME interface method.

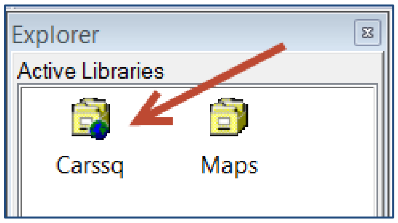
THE LIBNAME ENGINE INTERFACE IN SAS/ACCESS

The LIBNAME statement in Base SAS allows the user to communicate directly with various other data sources on the local machine or network. The LIBNAME statement allows the user to communicate directly with a data source as a SAS Library. Within SAS/ACCESS, the LIBNAME gains an engine which is specific to the DBMS you wish to communicate with, such as ODBC, DB2, Oracle, etc. This engine within the LIBNAME statement allows the user to not only communicate with a DBMS as if it were a SAS Library, but it also optimizes queries by determining if the query can be sent to the DBMS for faster processing. DBMS connection options may be specified for the user in the LIBNAME statement or the user can be prompted for this information. Below is an example of a simple LIBNAME statement using the ODBC engine within SAS/ACCESS.

```
   LIBNAME CARSSQ ODBC DATASRC=FMS_PROD SCHEMA=DBO DBPROMPT=YES;
```

The name given to this LIBREF above is “CARSSQ”. This LIBREF could be any name, as long as it conforms to the SAS naming conventions for librefs of: no more than 8 characters, must start with a letter or underscore, and contains only numbers, letters or underscores. The DBMS connection options DATASRC and SCHEMA are specified in the LIBNAME; these are the database and schema name within the DBMS that you are establishing the connection. The database name, FMS_PROD, contains an underscore, which is allowed. However, if your database name contains anything non-standard, such as a hyphen, it must be quoted as a name literal. The connection option DBPROMPT is set to YES, to prompt the user for the name of the database, schema, username and password when the LIBNAME statement is processed.

Once the LIBNAME statement has been successfully processed by SAS, you can see the tables within the DBMS connection to in the Explorer window of PC SAS as if it were a simple file location. You can easily right-click on any table in the DBMS to see the table and variable attributes just as you would with a regular file library.



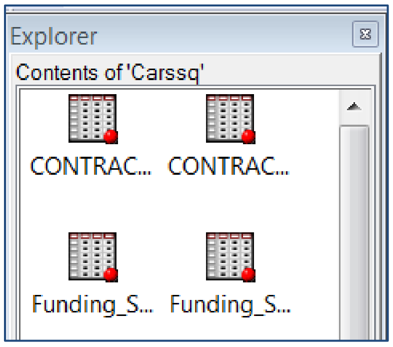


Figure 1. DBMS connection in the explorer window.

You can use any SAS statements you wish with a DBMS LIBREF, such as a DATA step or various procedures. However, the coding structure you choose can impact where SAS does its processing; and where the data is processed can greatly impact your run time.

WHERE DOES PROCESSING OCCUR?

When you submit SAS statements using a SAS/ACCESS LIBNAME engine interface to a DMBS, the query will be run within the SAS session or on the DBMS, depending upon the code you’ve written. The DBMS-specific engine in the SAS/ACCESS LIBNAME statement reviews each query to determine if part or all of it can be sent directly to the DBMS for processing to save time and resources. If the query is run in SAS, it must download the entire table from the DBMS to complete the query; however, if the query can be passed to the DBMS for processing, the DBMS only reads observations necessary for the query, thus maximizing efficiency and minimizing run time. If the query qualifies for processing on the DBMS, SAS/ACCESS translates the query to SQL statements native to the DBMS, submits the query to the DBMS for processing, and sends the results back to SAS; this is known as an implicit SQL pass-through. Using the pass-through facility in the implicit fashion allows the programmer to write code in SAS, instead of the native DBMS SQL, and in much less coding steps than explicit SQL pass-through. However, the programmer must be aware of what SAS code will trigger implicit SQL pass-through to maximize the chances of passing their query to the DBMS.

WHAT TRIGGERS IMPLICIT SQL PASS-THROUGH?

Implicit pass-through is triggered by certain key words and aggregate functions in PROC SQL, specific to the DBMS. Refer to the SAS/ACCESS documentation specific to your DBMS application for further details.

Implicit pass-through triggers for ODBC include (reference: Selecting a SAS/ACCESS method):

PROC SQL:

- DISTINCT
- JOIN
- UNION
- COMPUTED
- Aggregate functions (SUM, MAX, etc)

DATA Step:

- WHERE clause

There are certain conditions that will prohibit implicit pass-through, they are:

- Generated SQL syntax not accepted by the DBMS
- Multiple, separate librefs
- Most data set options
- SAS functions in the SELECT clause
- Use of the DIRECT SQL = LIBNAME option

Once you know the triggers of implicit SQL pass-through for your DBMS, you can write code that will cause the SAS/ACCESS engine to attempt the implicit SQL pass-through. But how does one know if implicit pass-through occurred?

DBMS_SELECT & SASTRACE SYSTEM OPTIONS

The system option DEBUG=DBMS_SELECT will help you determine if implicit SQL pass-through occurred. This handy debugging system option shows you the SELECT clause generated by the SAS/ACCESS LIBNAME engine that was *attempted* to be passed through to the DBMS for processing.

SASTRACE is another useful system option. SASTRACE=',,,d' will show you *all* the SQL statements generated by the LIBNAME engine that were attempted to be passed to the DBMS. Remember, when using SASTRACE that you must also declare where you want SAS to write these statements, such as the log: SASTRACELOC=SASLOG. It is also recommended that you use the NOSTSUFFIX option when using SAS trace to suppress information of limited use.

Both DEBUG and SASTRACE show the user helpful information in determining what SQL was generated by the SAS/ACCESS LIBNAME engine, but it also shows some prepare and execute statements that need a little practice interpreting. Let's look at some examples of code that processed on SAS and the DBMS with implicit SQL pass-through to see the feedback from these system options and the effect where processing occurred on run time.

QUERY PROCESSED ON SAS

This SAS code uses a SAS/ACCESS LIBNAME to create a permanent SAS table with data from a table within a DBMS, but this code does NOT achieve implicit pass-through because there is no trigger.

```
OPTIONS DEBUG=DBMS_SELECT SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
LIBNAME OLD 'L:\BFSPROC\CARSVOUCHERS';
LIBNAME CARSSQ ODBC DATASRC=FMS_PROD SCHEMA=DBO USER=&SYSUSERID. DBPROMPT=YES;

PROC SQL;
    CREATE TABLE old.TABLE2 AS
    SELECT Contract_Year AS year, Profile_ID AS profid, Appn AS app,
    Proj AS prj, Resp AS ra, Pcnt AS perct
    FROM CARSSQ.VOUCHER_CODING as Codetable
    WHERE CONTRACT_YEAR IN ('14','15','16','17');
QUIT;
```

Here is the abbreviated log generated from running this code:

```
9    LIBNAME CARSSQ ODBC DATASRC=FMS_PROD SCHEMA=DBO USER=&SYSUSERID. DBPROMPT=YES;
NOTE: Libref CARSSQ was successfully assigned as follows:
      Engine:        ODBC

11   <PROC SQL statements here>
ODBC: AUTOCOMMIT is NO for connection 0
ODBC: AUTOCOMMIT turned ON for connection id 0
DBMS_SELECT: SELECT * FROM "DBO"."VOUCHER_CODING"

ODBC_1: Prepared: on connection 0
SELECT * FROM "DBO"."VOUCHER_CODING"

DBMS_SELECT: SELECT  "Contract_Year", "Profile_ID", "Appn", "Proj", "Resp", "Pcnt  FROM
"DBO"."VOUCHER_CODING"  WHERE  (("Contract_Year" IN  ('14','15','16','17')))

ODBC_2: Prepared: on connection 0
SELECT  "Contract_Year", "Profile_ID", "Appn", "Proj", "Resp", "Pcnt"  FROM
"DBO"."VOUCHER_CODING"  WHERE  (("Contract_Year" IN  ('14','15','16','17')))

ODBC_3: Executed: on connection 0
Prepared statement ODBC_2
NOTE: Table OLD.TABLE2 created, with 2406 rows and 13 columns

17   QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           11.12 seconds
      cpu time            0.18 seconds
```

Narrative:

The note following Line 9 tells us that it successfully established a SAS/ACCESS LIBNAME with an ODBC engine. Feedback from the SASTRACE is highlighted in blue and that of DEBUG=DBMS_SELECT is highlighted in green. The first two lines of feedback from SASTRACE show the call to the DBMS from the ODBC engine to secure the connection.

Next, we see an initial call to the DBMS using a SELECT all clause. The last two messages are from SASTRACE and are prefixed with the engine name, ODBC, and highlighted in blue. The ODBC LIBNAME engine generated a SQL statement and attempted to pass it through to the DBMS. However, since we see no positive affirmation that implicit SQL pass-through occurred, we can assume that the query was processed on SAS. SAS ran the query and created the permanent SAS table "OLD.TABLE2" in about 11 seconds.

QUERY PROCESSED ON THE DBMS

The keyword DISTINCT will fetch rows with unique values for all the variables in the SELECT clause. Recognizing the fact that the keyword DISTINCT could be added to this query without changing the results and also knowing that this keyword is a pass-through trigger for ODBC, can greatly improve the run time for this code. Here is the code and log after adding the keyword DISTINCT to the SELECT clause:

```
OPTIONS DEBUG=DBMS_SELECT SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
LIBNAME OLD 'L:\BFSPROC\CARSVOUCHERS';
LIBNAME CARSSQ ODBC DATASRC=FMS_PROD SCHEMA=DBO USER=&SYSUSERID. DBPROMPT=YES;

PROC SQL;
    CREATE TABLE old.TABLE2 AS
    SELECT DISTINCT Contract_Year AS year, Profile_ID AS profid, Appn AS app,
    Proj AS prj, Resp AS ra, Pcnt AS perct
    FROM CARSSQ.VOUCHER_CODING as Codetable
    WHERE CONTRACT_YEAR IN ('14','15','16','17');
QUIT;
```

Here is the abbreviated log generated from running this code:

```
9    LIBNAME CARSSQ ODBC DATASRC=FMS_PROD SCHEMA=DBO USER=&SYSUSERID. DBPROMPT=YES;
NOTE: Libref CARSSQ was successfully assigned as follows:
      Engine:        ODBC
10
11   <PROC SQL statements here>
ODBC: AUTOCOMMIT is NO for connection 0
ODBC: AUTOCOMMIT turned ON for connection id 0
DBMS_SELECT: SELECT * FROM "DBO"."VOUCHER_CODING"

ODBC_1: Prepared: on connection 0
SELECT * FROM "DBO"."VOUCHER_CODING"

ODBC: AUTOCOMMIT is NO for connection 1
ODBC: AUTOCOMMIT turned ON for connection id 1
DBMS_SELECT:  select distinct Codetable."year", Codetable."profid", Codetable."app",
Codetable."prj", Codetable."ra", Codetable."perct" from "DBO"."VOUCHER_CODING" Codetable where
(Codetable."Contract_Year" in ('14','15','16','17'))) Codetable

ODBC_2: Prepared: on connection 1
 select distinct Codetable."year", Codetable."profid", Codetable."app", Codetable."prj",
Codetable."ra", Codetable."perct" from ( select Codetable."Contract_Year" as "year",
Codetable."Profile_ID" as "profid", Codetable."Appn" as "app", Codetable."Proj" as "prj",
Codetable."Resp" as "ra", Codetable."Pcnt" as "perct" from "DBO"."VOUCHER_CODING" Codetable
where (Codetable."Contract_Year" in ('14','15','16','17'))) Codetable

DEBUG: SQL Implicit Passthru stmt has been prepared successfully.

ODBC_3: Executed: on connection 1
Prepared statement ODBC_2

DEBUG: SQL Implicit Passthru stmt used for fetching data.
ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.
NOTE: Table OLD.TABLE2 created, with 2406 rows and 13 columns.

17   QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           5.80 seconds
      cpu time            0.21 seconds
```

Narrative:

The note following Line 9 tells us that it successfully established a SAS/ACCESS LIBNAME with an ODBC engine. Feedback from the SASTRACE is highlighted in blue and that of DEBUG=DBMS_SELECT is highlighted in green. The first two lines of feedback from SASTRACE show the call to the DBMS from the ODBC engine to secure the connection.

We again see an initial call to the DBMS using a SELECT all clause. This time, we notice that the LIBNAME engine opens another connection and both DBMS_SELECT and SASTRACE generate a more specific SQL statement to attempt implicit SQL pass-through. The SQL statement was accepted to be processed on the DBMS via implicit pass-through, and we receive positive affirmation of this from both SASTRACE and DEBUG options. It then ran the query on the DBMS and sent the results back to SAS. Notice that the same table was created, in almost half the time! The permanent SAS table “OLD.TABLE2” was created in less than 6 seconds.

FORCE THE PROCESSING ON SAS

The use of the keyword DISTINCT enabled SAS to make the table faster that without it and doesn’t show a clear comparison to the initial query processed on SAS. So, to demonstrate the effect of the ability of SAS to use the SAS/ACCESS LIBNAME to pass the query through to the DBMS for processing, let’s run the same query and disable the implicit pass-through ability with the PROC SQL option NOIPASSTHRU:

```
OPTIONS DEBUG=DBMS_SELECT SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
```

```
LIBNAME OLD 'L:\BFSPROC\CARSVOUCHERS';
LIBNAME CARSSQ ODBC DATASRC=FMS_PROD SCHEMA=DBO USER=&SYSUSERID. DBPROMPT=YES;

PROC SQL NOIPASSTHRU;
    CREATE TABLE old.TABLE2 AS
    SELECT DISTINCT Contract_Year AS year, Profile_ID AS profid, Appn AS app,
    Proj AS prj, Resp AS ra, Pcnt AS perct
    FROM CARSSQ.VOUCHER_CODING as Codetable
    WHERE CONTRACT_YEAR IN ('14','15','16','17');
QUIT;
```

Here is the abbreviated log produced:

```
9    LIBNAME CARSSQ ODBC DATASRC=FMS_PROD SCHEMA=DBO USER=&SYSUSERID. DBPROMPT=YES;
NOTE: Libref CARSSQ was successfully assigned as follows:
      Engine:        ODBC
10
11   <PROC SQL statements here>;
ODBC: AUTOCOMMIT is NO for connection 0
ODBC: AUTOCOMMIT turned ON for connection id 0
DBMS_SELECT: SELECT * FROM "DBO"."VOUCHER_CODING"

ODBC_1: Prepared: on connection 0
SELECT * FROM "DBO"."VOUCHER_CODING"

DBMS_SELECT: SELECT  "Contract_Year", "Profile_ID", "Appn", "Proj", "Resp", "Pcnt"
FROM "DBO"."VOUCHER_CODING"  WHERE  ( ( "Contract_Year" IN  ('14','15','16','17')))

ODBC_2: Prepared: on connection 0
SELECT  "Contract_Year", "Profile_ID", "Appn", "Proj", "Resp", "Pcnt"
FROM "DBO"."VOUCHER_CODING"  WHERE  ( ( "Contract_Year" IN  ('14','15','16','17')))

ODBC_3: Executed: on connection 0
Prepared statement ODBC_2
NOTE: Table OLD.TABLE2 created, with 2406 rows and 13 columns.

24   QUIT;
NOTE: PROCEDURE SQL used (Total process time):
      real time           9.28 seconds
      cpu time            0.25 seconds
```

Narrative:

This log looks very similar to that of the first query, which did not contain a trigger to achieve implicit SQL pass-through. Feedback from SASTRACE, highlighted in blue, shows the initial connection and the initial SELECT all clause prepared. Feedback from DEBUG=DBMS_SELECT, highlighted in green, also demonstrates the initial SELECT all clause, and a more specific SQL statement generated to attempt to be passed through to the DBMS. However, we see no notification that implicit pass-through occurred, as we expected with the use of the NOIPASSTHRU option. This query was ran on SAS and took just over 9 seconds to run. This query was able to run faster with the DISTINCT keyword, but not as fast as the query that was processed on the DBMS.

EFFECT IMPLICIT SQL PASS-THROUGH ON RUN TIME

Notice in all three benchmarking runs, we've created the same table of 2406 observations and 13 variables. Use of the keyword DISTINCT improved the run time by 2 seconds over the code without it, but the real efficiency comes with processing the query on the DBMS instead of SAS.

Method	Processing Location	Real Time (seconds)
No distinct, no implicit pass-through	SAS	11.1
Distinct, no implicit pass-through	SAS	9.3
Distinct, implicit pass-through	DBMS	5.8

Table 1. Effect of processing location on run time.

SUMMARY

Where your data is processed can greatly impact run time. Processing data on the DBMS instead of in SAS is more efficient in most cases. Achieving implicit SQL pass-through with use of a SAS/ACCESS LIBNAME engine interface is a relatively easy way to pass queries to the DBMS, as long as you understand the coding triggers specific to your DBMS. System options DEBUG and SASTRACE offer valuable information to determine if implicit pass-through occurred. Knowing the implicit SQL pass-through triggers for your DBMS will enable you to optimize your queries by writing queries that will be processed on the DBMS with greater efficiency than SAS can typically provide.

REFERENCES

- Plemmons, Howard. 2010. What's New in SAS/ACCESS®. SAS Global Forum 2010. Available at http://support.sas.com/resources/papers/proceedings10/302-2010.pdf
- SAS Institute Inc. The LIBNAME Statement for Relational Databases. DBPROMPT= LIBNAME Option. Available at http://support.sas.com/documentation/cdl/en/acreldb/65247/HTML/default/viewer.htm#p0bu3zsz1a08ton1msxdx1jo45np.htm
- SAS Institute Inc. Usage Note 207: New DEBUG Options for SAS/ACCESS Interfaces for Version 7+. Available at http://support.sas.com/kb/00/207.html
- SAS Institute Inc. Usage Note 6850: DEBUG=DBMS_SELECT returns the SELECT clause passed to a relational DBMS. Available at http://support.sas.com/kb/6/850.html
- SAS Institute Inc. 2008. Tactics for Pushing SQL to the Relational Databases. Available at http://support.sas.com/resources/papers/TacticsForPushingSQLtoRelationalDatabases.pdf
- SAS Institute Inc. 2011. *SAS® Certification Prep Guide: Base Programming for SAS®9, Third Edition*. 44. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. 2014. *SAS/ACCESS® 9.4 for Relational Databases: Reference, Sixth Edition*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/acreldb/67589/PDF/default/acreldb.pdf

ACKNOWLEDGMENTS

The author acknowledges Dr. LeRoy Bessler for inspiration on the paper topic, as well as Bruce Lund and Robin High, Beyond the Basic SAS Section Chairs, for acceptance of this paper.

RECOMMENDED READING

- Tactics for Pushing SQL to the Relational Databases
- SAS/ACCESS documentation

CONTACT INFORMATION

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

Name: Misty Johnson

Enterprise: State of WI Department of Health Services
E-mail: mistyA.johnson@wi.gov
Web: www.linkedin.com/in/MistyJohnson4