Paper SAS039-2014

An Insider's Guide to SAS/ACCESS® Interface to ODBC

Jeff Bailey, SAS Institute Inc., Cary, NC

ABSTRACT

The SAS/ACCESS® Interface to ODBC has been around forever. On one hand, ODBC is very easy to use. That ease hides the flexibility that ODBC offers. This paper shows you how to increase your program's performance and troubleshoot problems. You will learn the differences between ODBC and OLE DB, what the `odbc.ini` file is (and why it is important), and how to discover what your ODBC driver is actually doing. In the past you have scratched your head and wondered, "What is the difference between a native ACCESS engine and SAS/ACCESS Interface to ODBC?" This paper answers that question.

INTRODUCTION

The SAS/ACCESS Interface to ODBC has been around forever. Why write a paper about it now?

The answer is fairly simple: the product is extremely popular and practically ignored. Ignored might be the wrong word. It is taken for granted. The SAS/ACCESS Interface to ODBC just works and is incredibly useful. This paper uses examples to show you how to effectively use this product.

This paper covers these topics:

- the differences between ODBC and OLE DB
- what the `odbc.ini` file is and why it is important
- how to determine what your ODBC driver is actually doing
- specific techniques for increasing your program's performance

This paper includes examples from many ODBC data sources. These examples use data sources from SalesForce.com, Google BigQuery, PostgreSQL, and Pivotal Greenplum DB. No need to worry if your database is not listed. The steps are very similar, and you will be able to apply what you learn to your environment.

AN INTRODUCTION TO ODBC

Open Database Connectivity (ODBC) is a standard that was created by Microsoft. (ODBC 1.0 was released in September 1992.) ODBC has a very ambitious goal: to enable users to easily access data from any relational database using a common interface. ODBC is intended to be the industry standard for universal data access. As this paper shows, this goal might not have been met, but even so, ODBC is very versatile.

Many customers choose the SAS/ACCESS interface to ODBC for one simple reason – cost. Suppose you have five database management systems (DBMS), and you need to read data from each of them using a single SAS/ACCESS product. Your choices are the SAS/ACCESS Interface to ODBC or the SAS/ACCESS® Interface to OLE DB.

BASIC ODBC ARCHITECTURE

There are 4 basic components to an ODBC architecture:

- **Application** – This is the program that calls ODBC, submits SQL statements, and handles the resultant data. The examples in this paper use SAS as the application.
- **ODBC Driver** – The ODBC driver submits SQL statements to the specified data source and returns the resultant sets to the application. The driver is smart enough to modify the syntax of the request to conform to the syntax required by the data source. You need a driver that communicates with the server where your data lives. For example to access data stored in MongoDB, you need a MongoDB ODBC driver. There are many companies that create ODBC drivers. For example, both DataDirect and Simba create ODBC drivers.
- **Driver Manager** – The driver manager loads the ODBC drivers into memory based on the ODBC calls made by the application. The driver manager processes the function calls or passes them to the ODBC driver.
- **Data Source** – If you do not have a data source, all of this is pointless. Teradata, SalesForce.com, Twitter, Microsoft SQL Server, MongoDB, HBase, and Google BigQuery are examples of data sources. There are many more data sources that can be processed using ODBC. In fact, SAS can access data from any database that has an ODBC driver that meets the ODBC standard. This functionality makes the SAS/ACCESS Interface to ODBC

very popular with SAS users. (Note: The terms “data source” and “data source name” are often used interchangeably.)

WHAT IS THE DIFFERENCE BETWEEN ODBC AND OLE DB?

ODBC was designed to handle the data needs of SQL-based data sources. These data sources are typically relational databases. ODBC has been expanded to include non-relational databases that have their data stored in columns and rows.

Also developed by Microsoft, OLE DB is an open standard and was developed to handle data of any format. OLE DB is not SQL based. Here are some examples of non-relational data that OLE DB can handle: hierarchical databases, e-mail, file system stores, text, custom business objects, geographical data, and more.

ODBC is procedural based. OLE DB is component based.

ODBC does not support locking while OLE DB supports several locking models.

OLE DB has slightly better performance characteristics. Note this improved performance might not be true across the board. Be sure to test the specific ODBC drivers and OLE DB providers to determine the performance for your environment.

OLE DB was intended to replace ODBC but that has not happened. ODBC has very mature drivers that work reliably in the field. OLE DB is less mature and does not have the reassuring reputation that ODBC enjoys.

CONNECTING TO ODBC DATA SOURCES

An ODBC Data Source stores information that enables a client (such as SAS) to connect to a server. Each data source is given a name. A data source name (DSN) is the name assigned to a set of connection information. You determine the DSN for your connection. The name should be descriptive, easy to remember, and (if it will be used by others) easy to understand. Poorly chosen names can cause confusion.

This connection information can include:

- Data Source Name (DSN), which is chosen by the user
- ODBC driver – the software that allows us to interact with external data
- server name
- port number
- user ID
- password

Here is an example. Assume that you need to connect to Google’s BigQuery using SAS. To do this, you need an ODBC driver (such as BigQuery ODBC Driver from Simba), a Google account (which consists of a user ID and password), and a security token (that you get from Google). Add this connection information to an ODBC data source, and you have a valid ODBC data source. Now, you can access BigQuery data from your SAS programs. In Figure 1, the items in the **Systems Data Sources** box are circled. The first item in the list is Google BigQuery, which is the data source name.

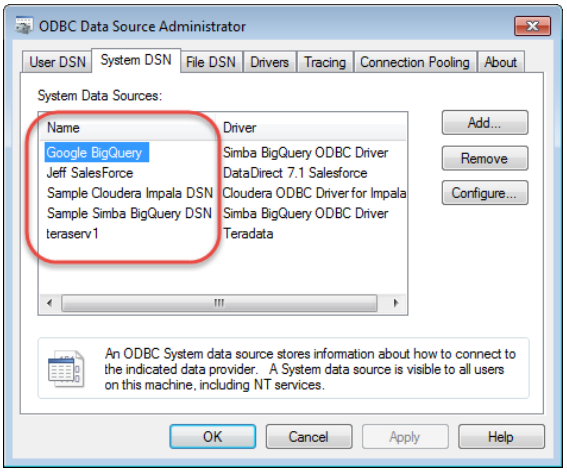


Figure 1. Systems Data Sources Box Shows the Google BigQuery Data Source Name

CREATING AN ODBC DATA SOURCE IN WINDOWS ENVIRONMENTS

Configuring ODBC data sources in Windows environments is fairly easy. Use the ODBC Data Source Administrator to create these data sources. This process is documented in many places, so there is no need to do that here. This paper discusses the portions of the utility that are important (or often overlooked). Figure 2 shows the **System DSN** tab in the ODBC Data Source Administrator.

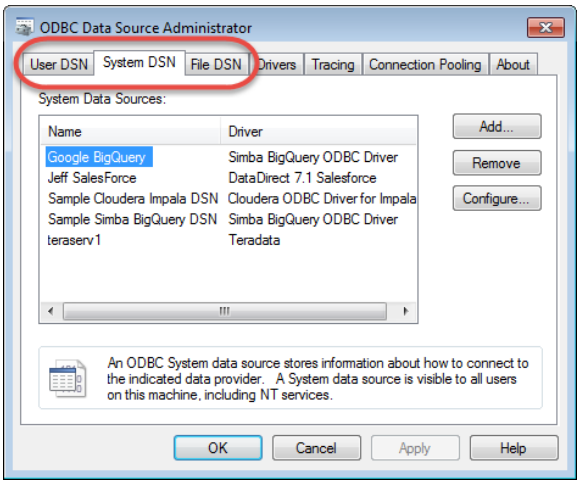


Figure 2. The Systems DSN Tab in the ODBC Data Source Administrator

Use the **User DSN** tab to create data sources that will only be visible to you. This tab is not used very often. Typically, your software creates the default data sources that are listed in this tab. For example when you install ODBC drivers, the drivers create the sample data sources that are listed in this tab.

Use the **System DSN** tab to create data sources that will be visible to all users on the machine. In this case users includes Windows NT services, which could be running on the machine. For an example of the **System DSN** tab, see Figure 2.

Use the **File DSN** tab to create data sources that are stored as files. You can share these data sources with other users who have the specified driver installed. Some organizations use this type of data source to distribute

connection information to their users (typically in support of production applications). Figure 3 shows the **File DSN** tab. In this example, a Teradata file data source is defined.

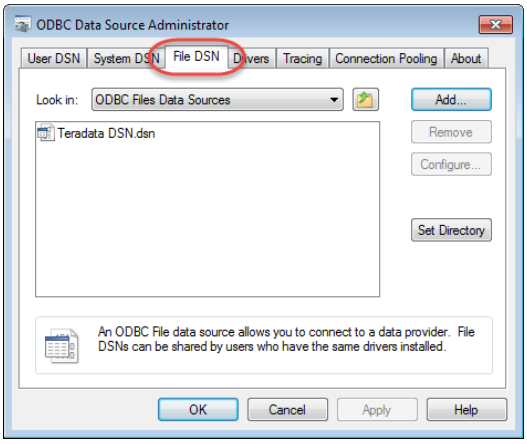


Figure 3. A File Data Source on the File DSN Tab

The **Drivers** tab is one of the most useful tabs in the ODBC Data Source Administrator. Use this tab to determine the version number of a specific ODBC driver. For example, suppose you need to know which version of the Teradata ODBC driver is installed on a machine. Figure 4 shows the Teradata driver, and the version of the driver is 14.10.00.00.

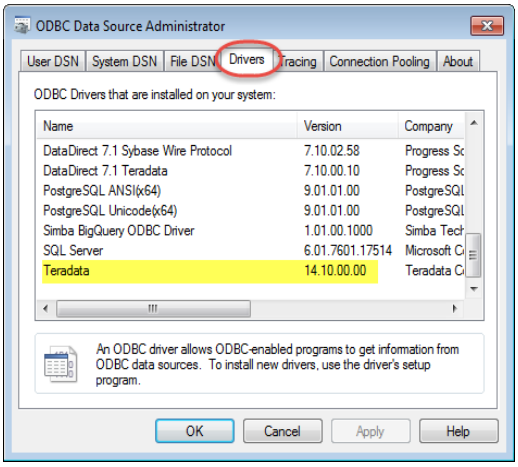


Figure 4. Determining the Version Number of a Driver on the Drivers Tab

Figure 5 shows the **Tracing** tab. This tab shows the ODBC API commands and SQL code generated by the ODBC driver. Use the options in this tab to specify where to write the output log and click **Start Tracing Now**. That is all there is to it. Remember to turn tracing off when you are finished. After tracing starts, the **Start Tracing Now** button changes to the **Stop Tracing Now** button.

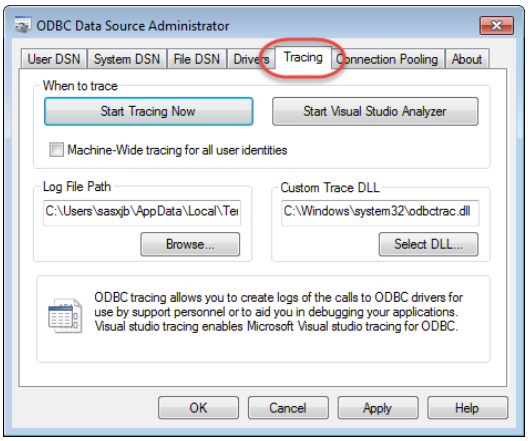


Figure 5. How to Start and Stop ODBC Tracing Using the ODBC Data Source Administrator

CREATING AN ODBC DATA SOURCE NAME ON UNIX

Unlike Windows environments, UNIX environments do not provide GUI tools to help you configure your ODBC connections. It is just you and the `vi` editor. Actually, your systems administrator will probably do this configuration for you. This paper assumes that you have your ODBC drivers installed, your environment variables set properly, and a valid `odbc.ini` file.

Before you start, ask your system administrator for the specifics for your environment. You probably can use an existing configuration file, but for this to work, you must have the environment variables set properly. Setting these environment variables is beyond the scope of this paper, but it is useful to know that you can use the environment variables to find the `odbc.ini` file on your system. The `env` command lists the environment variables for your account. The `ODBCINI=` environment variable shows you the path for the `odbc.ini` file.

Here are the ODBC-related environment variables from one of the UNIX systems at SAS:

```
LD_LIBRARY_PATH=/apps/odbc/dd7.10/lib:$LD_LIBRARY_PATH
ODBCHOME=/apps/odbc/dd7.10
ODBCINI=./odbc.ini
```

The `odbc.ini` file is very important because it defines the DSNs and any customizable parameters for your ODBC environment. In UNIX environments, the `odbc.ini` file is a text file. In Windows enviroments, `odbc.ini` is an entry in the registry. The `odbc.ini` file contains information vital to the smooth running of your ODBC connections. Each driver is different, so the information required is also different. Figure 5 shows the **Tracing** tab of the ODBC Administrator in Windows environments. To enable tracing on UNIX, you need to add entries to the `odbc.ini` file.

The `odbc.ini` file is divided into stanzas. A stanza is the section that follows the bracketed text. Each DSN has its own stanza. The opening stanza is `[ODBC Data Sources]`. Tracing is configured, and started, in the `[ODBC]` stanza.

In this `odbc.ini` file, the `[SalesForce]` stanza contains all the information needed to connect to SalesForce.com.

```
[ODBC Data Sources]
Salesforce=DataDirect 7.1 Salesforce

[ODBC]
IANAAppCodePage=4
InstallDir=/dbi/odbc/dd7.10
Trace=0
TraceFile=odbctrace.out
#TraceDll=/dbi/odbc/dd7.10/lib/odbctrac.so
TraceDll=/dbi/odbc/dd7.10/lib/ddtrc26.so

[Salesforce]
Driver=/dbi/odbc/dd7.10/lib/ddsfrc26.so
Description=DataDirect 7.1 Salesforce
ApplicationUsingThreads=1
```

```
BulkLoadAsync=0
BulkLoadBatchSize=1024
BulkLoadConcurrencyMode=1
BulkLoadFieldDelimiter=
BulkLoadPollInterval=10
BulkLoadProtocol=0
BulkLoadRecordDelimiter=
BulkLoadThreshold=4000
BulkLoadTimeout=0
ConnectionReset=0
ConfigOptions=
CreateDB=2
EnableBulkLoad=False
Database=
Extended Options=
FetchSize=100
HostName=
InitializationString=
IANAAppCodePage=
JVMArgs=-Xmx256m
JVMClasspath=
LoadBalanceTimeout=0
LogConfigFile=
LogonDomain=
LoginTimeout=15
LogonID=
MaxPoolSize=100
MinPoolSize=0
Password=
Pooling=0
ProxyHost=
ProxyPassword=
ProxyPort=
ProxyUser=
QueryTimeout=0
ReportCodepageConversionErrors=0
ReadOnly=0
RefreshDirtyCache=1
SecurityToken=
StmtCallLimit=20
StmtCallLimitBehavior=2
TransactionMode=0
WSFetchSize=0
WSRetryCount=0
WSTimeout=120
```

From this example, you can see that this code is more complicated than the code for Windows environments. Fortunately, you do not have to set all the parameters in this example. If you are ever in a situation where you need to configure an ODBC Data Source Name in Linux or UNIX environments, the best idea is to find a working example and copy it.

It is very important to test your ODBC connections. The remainder of this paper is full of examples that show how to connect to DBMSs using ODBC. Select your favorite example and use it for testing. An easy way of testing a DSN is to issue a LIBNAME statement using SAS. If you do not have SAS installed, you can use Microsoft Excel or another ODBC enabled application for testing.

CONNECTING TO AN ODBC DATA SOURCE USING SAS CODE

There are many ways to connect using SAS code, and this paper shows examples of all these connection techniques.

CONNECTING USING PROMPTING

Prompts are helpful when you are using SAS Display Manager and cannot remember the syntax for the ODBC LIBNAME statement. Prompts are also useful when you need to create a DSN. Prompts work in Windows environments and might work for some ODBC driver managers in Linux and UNIX environments.

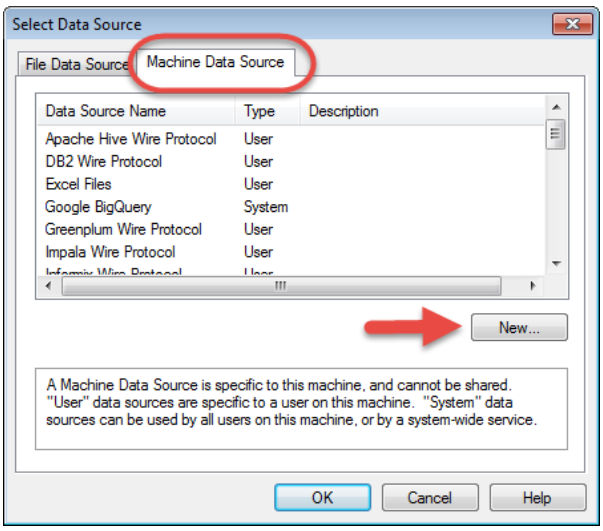
To connect to a Windows ODBC DSN using prompting:

1. In the SAS Display Manager, enter and submit this LIBNAME statement:.

   ```
   LIBNAME myodbc ODBC PROMPT;
   ```

   The Select Data Source dialog box appears.

2. In the Select Data Source dialog box, select the **Machine Data Source** tab. Now, select an existing DSN for your connection, or click **New** to create a DSN.



3. Complete the dialog boxes for your ODBC driver. Be sure to save your DSN. When you are finished, you can connect to the data source, and your SAS library is assigned.

USING A DSN CONNECTION

A DSN connection requires that you have an ODBC DSN defined by the ODBC Manager. You use this DSN in your SAS code. This method of connecting to ODBC is very common.

Here is an example of a DSN connection to a Microsoft SQL Server. Because the user ID and password are not stored in the DSN definition, they must be included in the LIBNAME statement.

```
LIBNAME sqlodbc ODBC DSN="MS_SQL" User=Jeff Password=Jeff123;
```

If the user ID and password are saved with the DSN (which is not recommended), the LIBNAME statement could be very simple.

```
LIBNAME sqlodbc ODBC DSN="MS_SQL";
```

Remember that you must have defined a DSN in order to use this technique. It is also worth mentioning that the various ODBC drivers differ greatly in complexity. So, your DSN connection might be more complicated than this one.

USING A DSN-LESS CONNECTION

There are three reasons why you might want to use a DSN-less connection. First, you can bypass the need to create a DSN using the ODBC Data Source Administrator. Second, because you are bypassing the administration application, the connections do not take as long. Connecting takes time. If you find yourself reconnecting to the same DSN, using the DSN-less technique might enable your jobs to run faster. Third, it is easy to include seldom-used or custom connections. In other words, you can set connections parameters without having to create a DSN or edit the

odbc.ini file.

Creating the DSN-less connection is not difficult. You use the LIBNAME statement with the CONNECT= option. You must look up name of the ODBC Driver to get this to work, but that is easy. You can find this name in the **Drivers** tab in the ODBC Data Source Administrator or in the odbc.ini file. Specific options for the driver can sometimes be found using the "trick" discussed in the "How Do I Find the Options Available for My ODBC Driver?" section of this paper. Unfortunately, this "trick" does not work for SaleForce.com, but do not worry. It is relatively easy to find this information using the Windows registry or an odbc.ini file on UNIX.

Here is a LIBNAME statement that uses a DSN-less connection to SalesForce.com.

```
LIBNAME mySF ODBC COMPLETE=DRIVER={DataDirect 7.1
SalesForce};Database=SFORCE;HostName=login.sas.com;UID=myuser@myemail.com;PWD=MyPassWo
rdsomelongtokenyougetfromSalesForce;
```

PERFORMACE TUNING

Unfortunately, there is no one-size-fits-all guide to performance tuning. However, here are some principles that can help you improve performance:

- The ODBC drivers have options that can be useful for performance tuning.
- You can make your programs run faster by making your SQL run faster.

HOW DO I FIND THE OPTIONS AVAILABLE FOR MY ODBC DRIVER?

There are times that you need to know the specific options that can be used with your ODBC driver. Looking up this information can be painful. Fortunately, you can sometimes use SAS to display these options. It might not work all the time, but it does work most of the time. This example uses prompting to display these options.

To display the options for an ODBC driver:

1. In the SAS Display Manager (or SAS Enterprise Guide), enter and submit the following LIBNAME statement. You do not have to use the PROMPT keyword. You can submit a LIBNAME statement and all the connection options.

   ```
   LIBNAME myodbc ODBC PROMPT;
   ```

2. In Select Data Source dialog box, select the DSN to use for the connection. You must have the specific connection information (which includes the server name, user ID, and password) required by the database.

3. Enter the following SAS macro code into SAS Display Manager (or SAS Enterprise Guide) and submit it.

   ```
   %PUT %SUPERQ(sysdbmsg);
   ```

Error! Reference source not found. shows the output in the SAS log after connecting to Aster Data.

```
1    libname adata odbc prompt;
NOTE: Libref ADATA was successfully assigned as follows:
      Engine:        ODBC
      Physical Name: nCluster
2    %put %superq(sysdbmsg);
ODBC:
DSN=nCluster;DATABASE=indb;SERVER=asterdata.somename.com;PORT=2406;UID=user;PWD=use
rpw1;SSLmode=disable;ReadOnly=0;Protocol=7.4;FakeOidIndex=0;ShowOidColumn=0;RowVers
ioning=0;ShowSystemTables=0;ConnSettings=;Fetch=100;Socket=4096;UnknownSizes=0;MaxV
archarSize=8000;MaxLongVarcharSize=8190;Debug=0;CommLog=0;Optimizer=1;Ksqo=1;UseDec
lareFetch=0;TextAsLongVarchar=1;UnknownsAsLongVarchar=0;BoolsAsChar=1;Parse=0;Cance
lAsFreeStmt=0;ExtraSysTablePrefixes=dd_;;LFConversion=1;UpdatableCursors=1;Disallow
Premature=0;TrueIsMinus1=0;BI=0;ByteaAsLongVarBinary=0;UseServerSidePrepare=0;Lower
CaseIdentifier=0;
```

Output 1. Log Output from the %PUT Macro

The output shows that there are a lot of parameters available for the Aster Data ODBC driver. Fortunately, you are not required to set them all. Any parameters that you do not specifically set use a default value.

If this trick does not work for your ODBC driver, do not worry because there are still ways to find this information. In Windows environments, look at `odbc.ini` entry in the registry. The registry key is `Computer\HKEY_.`. In Linux and UNIX environments, look in the `odbc.ini` file. Figure 6 shows the ODBC information for the DataDirect SalesForce.com driver.

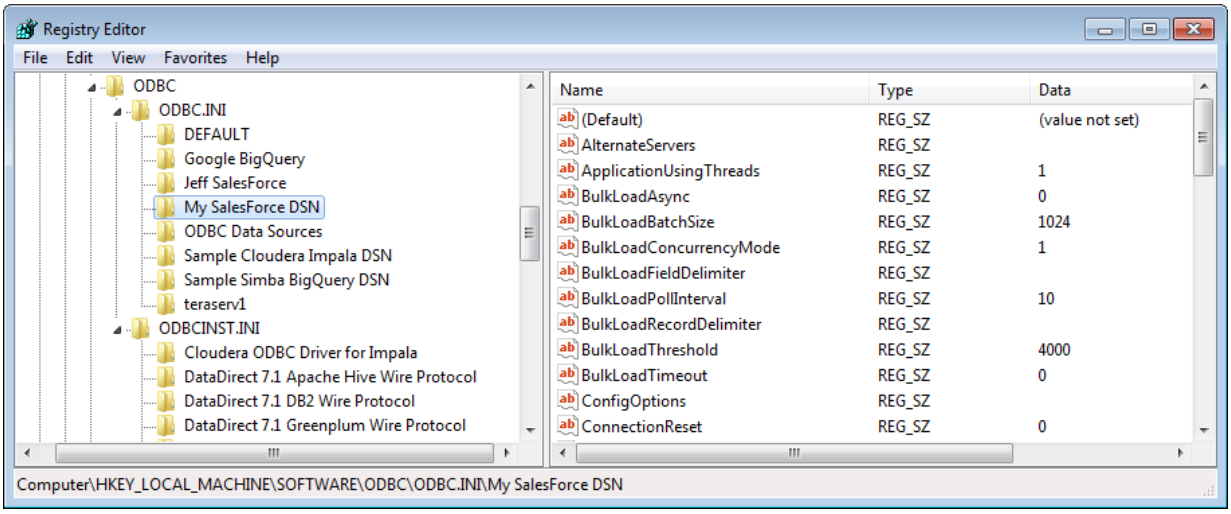


Figure 6. ODBC Driver Options in the Windows Registry

As previously discussed, performance tuning is not a one-size-fits-all situation. Each driver has options that can improve performance. When you have performance issues with SAS/ACCESS to ODBC, the first thing to do is find the list of options for your ODBC driver. Research any option that determines buffersizes or number of rows for an insert or read. Output 1 shows that the Aster Data ODBC driver has a `Fetch=100` option. Experiment with this option by increasing its value and running tests. Ideally, the performance improves. Make sure that you read the SAS/ACCESS Interface to ODBC chapter in the SAS/ACCESS 9.4 for Relational Databases: Reference. Pay close attention to these SAS options.

DBCOMMIT= number-of-rows

> This option affects how UPDATE, DELETE, and INSERT statements are made permanent (committed) in the DBMS. Setting DBCOMMIT=0 commits all changes after the procedure or DATA step completes. Zero is often the best setting, but experiment with this value in order to choose a good one. Note that the DBCOMMIT= and INSERTBUFF= options interact. For more information, see the SAS/ACCESS documentation.
>
> Here are the default values for the DBCOMMIT= option:
>
> - 1000 when a table is created and rows are inserted in a single step.
> - 0 when rows are inserted, updated, or deleted from an existing table.

INSERTBUFF= number-of-rows

> This option helps improve performance by increasing the number of rows in a single DBMS insert. The default is one row per insert. You should increase this value. The maximum value is DBMS dependent and is seldom the best choice. What is the best choice? It depends on the amount of memory that is available. Finding an acceptable value requires experimentation. Remember that this option interacts with the DBCOMMIT= option. For more information, see the SAS/ACCESS documentation.

READBUFF= number-of-rows

> This option helps improve performance by increasing the number of rows read during a fetch. The default is one row per fetch. You should increase this value. The maximum value is 32,767 rows per fetch, but this value is seldom the best choice. What is the best choice? It depends on the amount of memory that is available. Finding an acceptable value requires experimentation.

The UseServerSidePrepare= PostgreSQL ODBC driver parameter is particularly interesting if you are accessing PostGreSQL. Setting this to 1 can greatly increase query performance. This option can also increase performance of the SAS/ACCESS® Interface to PostgreSQL, which also uses ODBC.

The bad news is that there is no magic bullet. Each data source requires different tuning. Fortunately, just seeing the list of available options can help you with performance tuning. It is the helping hand to get you started. If your data source is a database, there are also things you can do outside of ODBC.

HOW DO I MAKE MY SQL QUERIES RUN FASTER?

This paper does not cover query tuning. Other papers (see "The SQL Tuning Checklist: Making Slow Queries a Thing of the Past" in the References section) have been written about that topic. One of the first steps in tuning a query is to run an OPTIONS statement in order to understand what SAS is asking the database to do.

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=saslog NOSTSUFFIX;
```

Here is an example of an OPTIONS statement in your SAS code:

```
LIBNAME tdodbc ODBC DSN="Teradata ODBC" schema=eecdata;


OPTIONS SASTRACE=',,,d' SASTRACELOC=saslog NOSTSUFFIX;

PROC SQL NOEXEC;
   SELECT a.store_id, a.transaction_id, b.date_id, b.employee_rk
     FROM tdodbc.order_fact a
          , tdodbc.order_fact b
    WHERE a.store_id = 1182041
      AND b.date_id = '31JAN2004'd;
QUIT;
```

Output 2 shows the results of running this code. The SQL that is being submitted to Teradata appears in bold. After you have this SQL code, you can tune it. Remember that a lack of indexes and missing (or out-of-date) statistics cause a lot of SQL performance problems. (For more information, see "The SQL Tuning Checklist: Making Slow Queries a Thing of the Past" in the References section.)

```
1    LIBNAME tdodbc ODBC DSN="Teradata ODBC" schema=eecdata;
NOTE: Libref TDODBC was successfully assigned as follows:
      Engine:        ODBC
      Physical Name: Teradata ODBC
2
3    OPTIONS SASTRACE=',,,d' SASTRACELOC=saslog NOSTSUFFIX;
4
5    PROC SQL NOEXEC;
6       SELECT a.store_id, a.transaction_id, b.date_id, b.employee_rk
7         FROM tdodbc.order_fact a
8            , tdodbc.order_fact b
9        WHERE a.store_id = 1182041
10         AND b.date_id = '31JAN2004'd;
ODBC: AUTOCOMMIT turned ON for connection id 0

ODBC_1: Prepared: on connection 0
SELECT * FROM eecdata.order_fact


ODBC_2: Prepared: on connection 0
SELECT * FROM eecdata.order_fact


ODBC_3: Prepared: on connection 0
 select a."STORE_ID", a."TRANSACTION_ID", b."DATE_ID", b."EMPLOYEE_RK" from
eecdata.order_fact
a, eecdata.order_fact b where (a."STORE_ID" = 1182041)
and (b."DATE_ID" =  {d '2004-01-31' })

NOTE: Statement not executed due to NOEXEC option.
11   QUIT;
```

```
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds
```

Output 2. Output from the SASTRACE= and SASTRACELOC= Options

WHAT IS THE DIFFERENCE BETWEEN AN ODBC-BASED ACCESS ENGINE AND SAS/ACCESS TO ODBC?

The SAS/ACCESS Interface to ODBC is a general use SAS/ACCESS engine designed to work with the ODBC drivers at your site. Here are some SAS/ACCESS engines for specific databases that use ODBC underneath the covers:

- SAS/ACCESS® Interface to Aster
- SAS/ACCESS® Interface to Greenplum
- SAS/ACCESS® Interface to Impala
- SAS/ACCESS® Interface to Microsoft SQL Server
- SAS/ACCESS® Interface to MySQL
- SAS/ACCESS® Interface to PostgreSQL
- SAS/ACCESS® Interface to SAP HANA
- SAS/ACCESS® Interface to Vertica

These SAS/ACCESS engines differ from SAS/ACCESS to ODBC in many ways:

- They are shipped with a SAS branded ODBC driver.
- They provide support for bulk loading using a vendor supplied utility.
- They support SQL statements (for example, UPDATE and DELETE) that are not support by the standard SAS/ACCESS engine.
- They support internationalization (I18N).
- They are capable of pushing more functions to the database, which makes the SQL processing faster.

CONCLUSION

The SAS/ACCESS Interface to ODBC is a very versatile product that enables you to access data from many data sources. Using ODBC data sources with SAS is not difficult, but there are lots of things to remember. This paper has shown many examples of how to connect to an ODBC data source using SAS LIBNAME statements and the SQL procedure. This paper has also demonstrated the DSN and DSN-less connection techniques and techniques that you can use to tune the performance of your SAS/ACCESS to ODBC programs. Your next step is to use these tricks and techniques to explore the ODBC data sources that you use every day. Make sure you read the SAS/ACCESS documentation so that you can get the most from SAS/ACCESS to ODBC.

REFERENCES

Bailey, Jeff. 2010. "Troubleshooting SAS and Teradata Query Performance Problems." Cary, NC: SAS Institute, Inc. Available at http://support.sas.com/resources/papers/TroubleshootingSASandTeradataQueryPerformanceProblems.pdf. Accessed on February 19, 2014.

Bailey, Jeff. Petrova, Tatyana. 2013. "The SQL Tuning Checklist: Making Slow Database Queries a Thing of the Past." *Proceedings of the SAS Global Forum 2013 Conference*. Cary, NC: SAS Institute Inc. Available at https://support.sas.com/resources/papers/proceedings13/080-2013.pdf.

ACKNOWLEDGMENTS

The author extends his heartfelt thanks and gratitude to the following individuals:

David Baccanari Jr., Progress Software Corporation

Pat Buckley, SAS Institute Inc.

Chris DeHart, SAS Institute Inc.

Marie Dexter, SAS Institute Inc.

Salman Maher, SAS Institute Inc.

Tom Newton, Simba Technologies Inc.

Tatyana Petrova, SAS Institute Inc.

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. Contact the author:

Jeff Bailey
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Jeff.Bailey@sas.com
http://www.sas.com