Paper SAS3358-2019

Accessing Your Favorite Database with SAS® and JDBC

Salman Maher, Bill Oliver, and Keith Handlon, SAS Institute Inc., Cary, NC

ABSTRACT

Have you ever wished you could use Java Database Connectivity (JDBC) with SAS®? If you have, we have exciting news: SAS/ACCESS® Interface to JDBC! JDBC is one of the most versatile database connectivity APIs available. JDBC is very popular, and it is common for vendors to release JDBC drivers before they release Open Database Connectivity (ODBC) drivers. Some vendors release only JDBC drivers for their products. Now is the time to learn about this important new technology. This paper shows you how to effectively use SAS/ACCESS Interface to JDBC. You will learn how to configure the JDBC driver, connect to your database, query data from a database, and write data to a database. This paper is the jump-start you need to learn how to effectively use SAS and JDBC with your database of choice.

INTRODUCTION

SAS/ACCESS is one of the fundamental components of the SAS system for accessing DBMS data. SAS/ACCESS Interface to JDBC is an extremely versatile product that gives you access to almost any data source for which there is a JDBC 4.0 or later driver. SAS/ACCESS Interface to JDBC was first released in 2018 with SAS Viya 3.4 and again with SAS 9.4M6. Using an example-driven approach to show how to use JDBC with SAS, this paper covers these topics:

- configuration
- connection methods
- data movement

CONFIGURATION

Getting started with SAS/ACCESS Interface to JDBC is simpler than with an ODBC-based data connector. No system options need to be set, and no DSN or ODBC administration tool required. You need only a Java JDBC driver that is accessible from your SAS session. In this paper we use the Teradata JDBC driver from http://downloads.teradata.com/download/connectivity/jdbc-driver to demonstrate using SAS/ACCESS Interface to JDBC with both SAS 9 and SAS Viya.

CONNECTING

Three options are required for SAS/ACCESS Interface to JDBC to connect to a data source.

CLASSPATH (default SAS_ACCESS_CLASSPATH environment variable value)

This option specifies the class path to your JDBC JAR files. The value that you specify for CLASSPATH= overrides the value that might have been set for the SAS_ACCESS_CLASSPATH environment variable.

CLASS (alias: DRIVERCLASS)

This option specifies the fully qualified class name for the JDBC driver to use for your database connection. This class name is loaded using a Class.forName call at connection time. In this paper we use the Teradata JDBC driver class, com.teradata.jdbc.TeraDriver. Currently, SAS/ACCESS Interface to JDBC cannot load a JDBC driver using the service provider mechanisms that were introduced with JDBC 4.

URL

This option specifies the JDBC connection string to use to connect to your data source. It is used as is and is not modified or parsed by SAS/ACCESS Interface to JDBC. The Teradata JDBC connection string used in this paper is as basic as you can get. It is jdbc:teradata://teradata.sas.com.

In addition to those three options, you must set these options for the JDBC driver.

USER (alias: UID)

This option specifies the user name to use to connect to your data source through a call to DriverManager.getConnection(String *url*, Properties *info*). If the user name contains spaces or non-alphanumeric characters, you must enclose it in quotation marks. The user name must be set with this option and cannot be set using the URL= option.

PASSWORD (alias: PWD)

This option specifies the user password to use to connect to your data source. If the password contains spaces or non-alphanumeric characters, you must enclose it in quotation marks. If you do not want to enter your password in plane text, see PROC PWENCODE in the *Base SAS® 9.4 Procedures Guide* for a method to encode it. The user password must be set with this option and cannot be set using the URL= option.

SCHEMA

This option specifies the schema that you want to use when loading and accessing tables in your data store. If the schema name contains spaces or non-alphanumeric characters, you must enclose it in quotation marks. The SCHEMA= option is required if your JDBC driver is older than 4.1 or if your database vendor did not implement the Connection.getSchema method. If SAS/ACCESS Interface to JDBC cannot discover your SCHEMA by calling Connection.getSchema or if the SCHEMA= option is omitted, this error message is displayed in your SAS log.

```
ERROR: java.lang.AbstractMethodError
ERROR: Error retrieving DatabaseMetaData: java.lang.AbstractMethodError
```

PRESERVE_NAMES

Many modern data source today allow for the use of mixed-case table and column names. This option preserves spaces, special characters, and case sensitivity of the table and column names in your data source. Use this option when connecting to data source that allow for mixed case table and column names.

LIBNAME STATEMENT

The SAS/ACCESS Interface to JDBC LIBNAME statement extends the SAS global LIBNAME statement so that you can assign a libref to your data source. This feature lets you reference a table in your data store directly in a DATA Step or SAS procedure. You can use it to read from and write to your data source as if it were a SAS data set. This example shows the basic LIBNAME statement to connect to Teradata.

```
libname libTera jdbc classpath="/opt/teradata/client/JDBC"
  class="com.teradata.jdbc.TeraDriver"
  URL="jdbc:teradata://teradata.sas.com";
  user="teraUser"
```

```
    password="TeraUser1"
    schema=db;
```

In this case we had to set the SCHEMA= option because the Teradata JDBC driver does not implement the Connection.getSchema method. The SCHEMA= option replaces any schema values that are typically specified in the URL= option.

SAS VIYA CASLIB STATEMENT

To access your data source directly from the CAS server in SAS Viya, you use a CASLIB statement, which is very similar to a LIBNAME statement. One major difference is that the CASLIB statement does not try to connect to your data source when the statement is executed. The connection is deferred until the first data source request is made by a CAS action. You also specify data connector options within the DATASOURCE= option. Using the above JDBC LIBNAME statement, here is how to translate it into a caslib:

```
caslib clibTera sessref=mysess
  datasource=(srctype=”jdbc”,
    classpath=”/opt/3eradata/client/JDBC”,
    driverclass=”com.teradata.jdbc.TeraDriver”,
    URL=”jdbc:teradata://teradata.sas.com”,
    username=“teraUser“,
    password=“TeraUser1“,
    schema=“db“);
```

The name that you give to your JDBC casLib must be unique within the session that you specified with the SESSREF= option on the JDBC CASLIB statement. If your JDBC casLib name is the same as a global caslib, the global caslib is effectively hidden from use within your session.

Note: The JDBC caslib does not honor the SAS_ACCESS_CLASSPATH environment variable, which makes the CLASSPATH= option required.

PROC FEDSQL

You can connect to your data source in two ways using the FedSQL product. The most common approach is to use a LIBNAME statement and reference tables in FedSQL with a libref as you would for any other SAS product. The less common second method is with the CONN= option in conjunction with the NOLIBS option. Using NOLIBS with CONN= overrides the default data source connection with the specified connection string. This approach lets you make a data source connection that is scoped only to the FedSQL job. This is very similar to the CONNECT TO option in PROC SQL. If you specify the NOLIBS option without the CONN= option, the PROC FEDSQL job fails to execute. This example makes the same JDBC-to-Teradata connection as in the LIBNAME and CASLIB statement sections above, and it returns all available tables in the DB schema.

```
proc fedsql nolibs conn="DRIVER=JDBC;
  classpath='/opt/teradata/client/JDBC';
  CLASS=com.teradata.jdbc.TeraDriver;
  URI='jdbc:teradata://teradata.sas.com';
  userid=teraUser;
  password='TeraUser1';
  schema=db;
  CATALOG=jdbc";
    select * from dictionary.tables;
QUIT;
```

Two additional options are used, DRIVER= and CATALOG=, that are specific to the PROC FEDSQL. DRIVER= specifies the underlying SAS data source data connector that you want the PROC FEDSQL provider to use—in this case, the JDBC data connector. CATALOG= specifies an arbitrary identifier for an internal SQL catalog to group logically related schemas.

QUERYING DATA

DATA QUERYING WITHIN SAS 9

DATA Step

One of the great things about using SAS to access a third-party data source is that it hides so much of the complexity of dealing with the data source. You can again see this when using the SAS DATA Step with SAS/ACCESS Interface to JDBC. The four-line SAS program below can do a lot of work. It creates a stocks SAS data set and moves all stock data for IBM from a Teradata database using the JDBC LIBNAME defined above.

```
data work.IBM;
  set tera.stocks(keep=stock date close);
  where stock = 'IBM';
run;
```

PROC SQL

PROC SQL is the standard way to query a data source with SQL. Here is an example that shows how to do the same thing as above using PROC SQL.

```
proc sql;
     create table work.IBM as
       select stock, date, close
       from tera.stocks
       where stock='IBM';
quit;
```

Using SAS/ACCESS Interface to JDBC with PROC SQL is as any other SAS/ACCESS products.

Explicit Pass-Through

You can use the PROC SQL explicit pass-through feature to communicate directly with your data source by using data source-specific SQL and DDL statements. Two connection methods exist for using explicit pass-through: the CONNECT USING syntax (recommended) and the CONNECT TO syntax.

CONNECT USING

This example uses explicit pass-through with the CONNECT USING syntax with the JDBC LIBNAME that was defined above. The PROC SQL job executes two statements. The first creates the sasteracars table in the WORK LIBNAME and populates it with a data from the teracars table in the Teradata database using the JDBC LIBNAME that is defined. In this case, the SELECT statement is passed to the Teradata database as is without SAS processing. The second statement drops the teracars table in the Teradata database.

```
proc sql;
  CONNECT USING tera;
  create table work.sasteracars as select * from
    connection to tera (select * from teracars);
  execute (drop table teracars) by tera;
  disconnect from tera;
quit;
```

In both statements above, the SQL is passed directly to the data source as is without SAS processing. Therefore, SQL must be valid SQL for your target data source.

CONNECT USING is also useful when the LIBNAME is preassigned or when the LIBNAME statement is not available to the user. This is because, once the LIBNAME statement is assigned, the user does not need to know the LIBNAME connection options to use the CONNECT USING statement.

CONNECT TO

The CONNECT TO syntax is very similar to using the PROC FedSQL NOLIB and CONN= options. With this syntax you must provide all connection information in the PROC SQL statement. This PROC SQL example is the same as above but uses CONNECT TO syntax with all connection information that is needed to make a JDBC connection to a Teradata data source.

```
  proc sql;
    CONNECT TO JDBC as cttera (classpath="/opt/teradata/client/JDBC'"
        class="com.teradata.jdbc.TeraDriver"
        user="teraUser" password="TeraUser1"
        schema=db URL="jdbc:teradata://vatcop1.unx.sas.com");
     create table work.sasteracars as select * from
        connection to cttera (select * from teracars);
     execute (drop table teracars) by cttera;
        disconnect from tera;
  quit;
```

If you use shared connections (CONNECTION=SHARED), the options that are defined in the CONNECT TO statement must match exactly the options that are defined in your LIBNAME or a new connection is created. This is critical to know if you plan to work with temporary tables.

PROC FEDSQL

The FedSQL language is the SAS implementation of the ANSI SQL-99 core standard. It provides a common SQL syntax across all data sources that it supports. FedSQL is a vendor-neutral SQL dialect that accesses data from various data sources without having to submit queries in the SQL dialect that is specific to the data source. To keep things simple and show that you can use the JDBC LIBNAME with PROC FEDSQL, this example rewrites the above PROC SQL job using FedSQL.

```
  proc fedsql libs=(work tera);
    create table work.IBM as
    select st.stock, st.date, st.close
    from tera.stocks as st
    where stock='IBM';
  quit;
```

You can see in the code that we had to use a table alias to qualify the column names in the SELECT statement because close is a SQL-99 reserved word. You can also see in the PROC

SQL section above that we did not have to do that. To learn more about FedSQL please see the *Base SAS 9.4 Procedures Guide*.

QUERYING DATA WITHIN SAS VIYA

You can use two CAS actions with SAS data connectors to request data from third-party data sources: LOADTABLE and FedSQL EXECDIRECT.

LOADTABLE Action

The LOADTABLE action directs the server on your behalf to load a table from a specified caslib data source into the CAS server. At a minimum, this action takes a CASLIB= option to describe the origin of your table to load, a PATH= option that is the table name of your DBMS table to load, and a CASOUT= option to specify the location for the output table. For example, to load the cars table in Teradata into CAS with the name mycars, you could use this LOADTABLE statement with the above-defined caslib:

```
proc cas;
  session mysess;
  action loadTable / caslib=”clibTera” path=”cars” casout=”mycars”;
quit;
```

FedSQL EXECDIRECT Action

The fedSql.execDirect action was updated so that you can submit queries to DBMS caslib data sources. This enables implicit pass-through with both PROC FEDSQL and the fedSql.execDirect action. Implicit pass-through is the process of translating a FedSQL query into the equivalent DBMS-specific SQL so that it can be executed completely in the DBMS. This improves query response time and helps limit the amount of data transferred to the CAS server. When a FedSQL query is submitted to a single data source, an attempt is made to implicitly pass the full query down to the data source for processing. If the targeted data source cannot prepare the query, implicit pass-through fails and the request is processed locally on the CAS server.

Here is what is required to be eligible for implicit pass-through:

1. All tables in the FedSQL request must exist in the same caslib.
2. All tables in the FedSQL request must reside on the data source. They cannot already have been loaded into the CAS session.
3. The target data source must be able to prepare the SQL request.

To submit your FedSQL action request to a caslib you must fully qualify the table using your caslib in the FROM clause. For example, this FedSQL query returns all 2016 Fords with six cylinders from a cars table stored in a Teradata database using the caslib that was defined above.

```
proc cas;
  session mysess;
  action fedsql.execdirect
    query="select *
      from ""clibTera"".""cars""
      where make='Ford' and Cylinders=6";
run;
```

As you can see the table is referenced in the FROM clause as caslib.tablename—in this case, clibTera.cars. Note the two sets of double quotation marks around the caslib name and the table name. This is required when using the JDBC caslib because table names and column names are case sensitive. If the FedSQL query is successfully pushed down to the database, the log contains a success note. Here is the log for the above:

```
73         proc cas;
74            session mysess;
75            action fedsql.execdirect
76                 query="select * from ""clibTera"".""cars"" where make='Ford' and
Cylinders=6";
77         run;
NOTE: Active Session now mysess.
NOTE: Added action set 'fedsql'.
NOTE: The SQL statement was fully offloaded to the underlying data source via full
pass-through
78         quit;
NOTE: PROCEDURE CAS used (Total process time):
real time          0.74 seconds
cpu time           0.10 seconds
```

In addition to submitting FedSQL queries with the FedSQL EXECDIRECT CAS action, you can also do this with PROC FEDSQL by setting the SESSREF= option to your CAS session and fully qualifying the table using your caslib.

```
proc fedsql session mysess;
  create table public.fords as select * from "pgLib"."cars"
  where make='Ford' and Cylinders=6;
run;
```

The above example shows how to save the results of the FedSQL query into a new CAS table called fords under the public caslib.

WRITING DATA

WRITING DATA WITHIN SAS 9

Write usage patterns are very similar to the query usage pattern that is shown above. In almost all cases the only difference is in the output data source. For write use cases, the output data source is the JDBC libref.

DATA Step

The DATA Step example above created the IBM temporary table in the WORK LIBNAME space. To write the IBM table back to your data source, you need to change only work.IBM to tera.IBM. The data set creates a new IBM table in your data source with the three columns: stock, date, and close. It then inserts all rows from the stocks table that meet the condition in the WHERE clause.

```
data tera.IBM;
  set tera.stocks(keep=stock date close);
  where stock = 'IBM';
run;
```

Writing large amounts of data back to your data source can result in long delays. One option that SAS/ACCESS Interface to JDBC provides to manage the time that is needed to perform a write action is INSERTBUFF=.

INSERTBUFF

This option helps improve performance by increasing the number of rows in a single insert statement. The default is 100 rows per insert, which is subject to change in future versions of SAS. You should increase this value. The maximum value is data source-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. The examples below show how INSERTBUFF= can impact the time it takes to write 30,000 rows from the SASHELP.ZIPCODE table with the JDBC LIBNAME to Teradata.

```
10   data tera.zipcode; /* INSERTBUFF=100 default value*/
11       set sashelp.zipcode(obs=30000);
12   run;

NOTE: DATA statement used (Total process time):
      real time           1:32.48
      user cpu time        15.37 seconds
      system cpu time    14.89 seconds

14   data tera.zipcode(insertbuff=500);
15       set sashelp.zipcode(obs=30000);
16   run;

NOTE: DATA statement used (Total process time):
      real time            1:07.50
      user cpu time         12.68 seconds
      system cpu time     11.36 seconds

18   data tera.zipcode(insertbuff=1000);
19       set sashelp.zipcode(obs=30000);
20   run;

NOTE: DATA statement used (Total process time):
      real time             58.16 seconds
      user cpu time       13.81 seconds
      system cpu time  13.89 seconds

22   data tera.zipcode(insertbuff=10000);
23       set sashelp.zipcode(obs=30000);
24   run;

NOTE: DATA statement used (Total process time):
      real time             59.22 seconds
      user cpu time       12.34 seconds
      system cpu time  10.26 seconds
```

In the above example, though changing the INSERTBUFF= value from 100 (default) to 1000 significantly impacts write time, changing from 1000 to 10000 has no impact.

PROC SQL and PROC FEDSQL

As in the above DATA Step example, you need to change only the target destination of your PROC SQL or PROC FEDSQL job to be your JDBC libref to write to your data source. Here is an example.

```
proc sql;
   create table tera.IBM as
      select stock, date, close
      from tera.stocks
      where stock='IBM';
quit;

proc fedsql;
   drop table tera.IBM;
   create table tera.IBM as
      select st.stock, st.date, st.close
      from tera.stocks as st
      where stock='IBM';
quit;
```

WRITING DATA WITHIN SAS VIYA

With SAS Viya, you use the SAVE table action to write in-memory CAS tables back to your data source. At a minimum, the action takes a casLib to write the table, the name of the table to create, and a table option that describes the source table to write out. For example, to write back the mycars table that is loaded into CAS from the above LOADTABLE action to your Teradata DBMS as myCarsFromCAS, you could use this save action statement:

```
proc cas;
   session mysess;
   action save / caslib="clibTera" name="myCarsFromCAS"
      table={caslib="clibTera" name="mycars"};
 run;
```

Executing the SAVE table action results in a CREATE TABLE statement being execute against your data source. This action always creates a new table. If a table already exists in your DBMS with the same name, the action fails unless you specify the REPLACE=TRUE option.

You can use the WHERE= and VARS= options in the table option to filter data and columns without creating a temporary table. For example, this action subsets the cars table to for car makes from only 2016, and it creates a new table in your data source named my2016Cars with only one column named makes:

```
proc cas;
   session mysess;
   action save / caslib="clibTera" name="my2016Cars"
      table={caslib="clibTera" name="mycars"
         vars={“make”}
         where=”year = 2016”};
run;
```

CONCLUSION

SAS/ACCESS Interface to JDBC brings the world of JDBC connectivity to SAS. Using SAS and JDBC is not difficult. This paper has detailed configuration requirements, connecting with SAS LIBNAME statements, connecting with the CASLIB statement, the SQL procedure, and DATA Step. Your next step is to use this new knowledge in your everyday work. Make sure you read the SAS/ACCESS documentation. It can help you get the most from your SAS investment.

REFERENCES

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 authors extend their heartfelt thanks and gratitude to these individuals:

Jeff Bailey, SAS Institute Inc., Cary, NC

Greg Otto, Teradata, San Diego, CA

RECOMMENDED READING

- SAS Institute Inc. 2016. "Working with SAS Data Connectors." *SAS Cloud Analytic Services: Language Reference*. Available at http://documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=casref&docsetTarget=p1ez56aqp5uvukn1f96jscujvplm.htm
- SAS Institute Inc. 2016. "Quick Reference for Data Connector Syntax." *SAS Cloud Analytic Services: Language Reference*. Available at http://documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=casref&docsetTarget=n0wxsz0rzamws5n1ldbepubijenk.htm

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

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

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

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