Paper 115-2012

What's New in SAS/ACCESS® and Process Improvements to Apply to Your DBMS

Howard Plemmons, SAS Institute Inc., Cary, NC

ABSTRACT

This paper provides an overview of major new features in SAS/ACCESS and looks at process improvements that can help with DBMS performance. The information provided will help with SAS® execution performance.

INTRODUCTION

The SAS/ACCESS product suite continues to grow, as shown in Table 1. Such broad product offerings provide you with access to your data for analysis.

SAS/ACCESS Interface	Platforms that SAS Supports
Aster *n*Cluster	Linux andWindows
DB2	all
Greenplum	all except HP/PA-RISC and mainframe
Hadoop™	64-bit Linux and 64-bit Windows
Informix	all except Windows and mainframe
Microsoft SQL Server	all except Windows and mainframe
MySQL	all except mainframe
Neoview	all except HP/PA-RISC, Solaris I86, 64-bit Linux,64-bit Windows, and mainframe
Netezza	all except Solaris I86 and mainframe
ODBC	all except mainframe
OLE/DB	32- and 64-bit Windows
Oracle	all
PC Files	all
Sybase	all except mainframe
Sybase IQ	all except mainframe
Teradata	all
IMS IDMS ADABAS DATACOM	mainframe

Table 1: Supported Interfaces and Platforms

Platforms that SAS/ACCESS supports in SAS 9.3M1 and SAS 9.3M2 are HP (PA-RISC and I86), SUN (SPARC and I86), Linux (32- and 64-bit), 64-bit AIX, 32- and 64-bit Windows and mainframe. SAS system requirements support specific hardware and OS requirements for each platform.

SOME HIGHLIGHTS OF WHAT'S NEW

Significant new features span the SAS/ACCESS product portfolio. It was difficult to pick between enhancements that are in SAS 9.3M1 and what is being developed and slated for SAS 9.3M2. The focus of these enhancements are performance improvements and new engines.

SAS/ACCESS INTERFACE TO HADOOP

Released as LA (limited availability) in early March, this new SAS/ACCESS LIBNAME engine lets SAS customers interact with the Hadoop Data Warehouse using Hadoop Hive. To understand how SAS/ACCESS works with Hadoop requires some background knowledge. What is Hadoop?

1. The name of the toy elephant that belonged to Doug's son
2. An Apache project that global contributors use
3. A common set of JAR files

All answers are correct. The Hadoop name came from the name of Doug's son's toy elephant. Global contributors to both the open source and commercial sides of Hadoop use the Apache project . And the Hadoop Common contains the necessary JAR files to start Hadoop. A myriad of Hadoop technical references are available on the Web. To help you get started, see a few free favorites that I have placed in the Resources section of this paper. I suggest you become more familiar with Hadoop before proceeding. Specifically, you need basic knowledge of Hadoop. The explanation of the SAS/ACCESS engine has a SAS focus, and Hadoop terms are not fully explained.

The SAS/ACCESS to Hadoop engine uses HiveQL (Hive Query Language) to interact with Hadoop. This engine shares many of the capabilities of other SAS/ACCESS LIBNAME engines, such as the SQL explicit and implicit pass-through through facility, the infrastructure needed to process SAS DATA step and procedure code, and data loading. For example, this shows how to connect to Hadoop:

```
/*--- Connect to the Hive Service that is running on the Hadoop server ---*/

libname x hadoop server=hxpduped user=sasuser password=sasuser;

/*--- Once connected to Hive, SAS commands can show you what you have ---*/
proc datasets lib=x;
run;
proc contents data=x.sastab;
run;
```

These SAS procedures produce this output:

```
                              The SAS System

                                Directory
               Libref          X
               Engine          HADOOP
               Physical Name   jdbc:hive://dbihadoop:10000/default
               Schema/Owner    default

                      #  Name     Member Type  DBMS Member
                                               Type
                     10  FOO      DATA
                     11  HADOOP1  DATA
```

Figure 1: Sample Hadoop Output from PROC DATASETS

```
                              The SAS System

                          The CONTENTS Procedure
      Data Set Name      X.HADOOP1  Observations           .
      Member Type        DATA       Variables              3
      Engine             HADOOP     Indexes                0
      Created            .          Observation Length     0
      Last Modified      .          Deleted Observations   0
      Protection                    Compressed            NO
      Data Set Type                 Sorted                NO
```

```
Label
Data Representation Default
Encoding            Default
```

```
Alphabetic List of Variables and Attributes
# Variable Type Len Format Informat Label
1 x        Num    8                 x
2 y        Char   7 $7.    $7.      y
3 z        Num    8                 z
```

Figure 2: Sample Hadoop Output from PROC CONTENTS

Once you connect and see what has been defined through Hive using PROC DATASETS and PROC CONTENTS, what's next?

Hadoop has been called the BIG data server. Hadoop's power lies in the vast amounts of storage and memory (RAM) that can be concurrently applied to a problem. With Hadoop clusters containing hundreds or thousands of nodes, data that is persisted can be enormous—represented in petabytes. To consider moving a Hadoop table that could contain trillions of rows out of Hadoop might not be practical. So what do you consider?

Process Development

You have Web log data since the Web began. You want to surface that data to your SAS community so they can find the game-changing information that is hidden inside. Do they need all of the data? Probably not, so you instead create Hive table aggregates to limit the amount of data pull.

SAS User Education

If you have some process development, your next consideration should be what to impart to the SAS user base. Depending on what you have stored in Hadoop and surface for access, some queries might never return. Consider these:

- `select count(*):` To return information on this request, Hadoop reads all data internally to perform the count using the MapReduce model. If you have 1.5 trillion rows of wide data, the impact of the query on the Hadoop server could be very costly. If you are obtaining row counts join processing, it might be safe to assume that the Hadoop table is always the master table.
- `data a; set hadoop.b;:` A straight data pull on a tremendous table might never return. You must use aggregate or limiting functions when accessing data of extreme size.
- joins in SAS: If you write SAS code that boils down to a join in SAS, then the 1.5 trillion rows that are stored in Hadoop are extracted into SAS, resulting in a join that might never complete. ETL (extract, transform, load) processes must consider a Hadoop side-join. PROC SQL assumptions are made for Hadoop that push in-clause joins with a small number of elements directly to Hadoop.

Batch Processing

Hadoop is designed as a batch system. If you expectat it to behave like your OLTP for quick transaction processing, you will be disappointed. It operates on tremendous amounts of data in parallel. For example, you push a HiveQL statement to Hadoop that selects all sales data from California from 200-2012. Hadoop converts that statementto MapReduce statements and processes against every Hadoop data split in parallel. It returns the results of that query to SAS. The amount of concurrent operations that can occur against the data is one aspect that make Hadoop so powerful.

How do I configure the Hadoop system I have so I can use the SAS/ACCESS engine? You need to create Hivetables that reference the Hadoop Distributed File System (HDFS) data files that you want to access with SAS.

- If you have existing Hadoop HDFS files that you want to map to Hive, you could use this code, where what is in the PROC SQL exec is a Hive DDL command.

```
proc sql;
connect to Hadoop <connection info>;
exec( create external table hadoop1( x double, y string, z double) row format
delimited fields terminated by '\001' stored as textfile location
'/tmp/hadoop1_hdfs_file') by hadoop;
quit;
```

Here, the Hive table is mapped to existing HDFS data. If you create and load a Hive table where Hive manages the data internally and then drop the Hive table, the data also goes away. If you create an external Hive table and then drop the table, only the Hive table definition is deleted. Therefore, if you create a Hive table and load the data into storage that Hive manages, the data is deleted when you drop the table. If you want to keep the data, create the Hive table and map it to your HDFS data file.

- If you have SAS data that you want to put into Hadoop. This DATA step code triggers a multistep process :

```
/*--- Load SAS data into Hadoop using the SAS DATA step ---*/

libname x hadoop <connection info>;
data x.hadoop1;
  Set work.my_sas_data;
run;
```

1. What runs under the covers is the following HiveQL CREATE TABLE statement below. SAS File ACCESS method is used as a pipe to get the data to Hadoop then a Hadoop load command to get the data into Hive.

```
CREATE TABLE `HADOOP1` (`x` DOUBLE,`y` STRING,`z` DOUBLE) ROW FORMAT
DELIMITED FIELDS TERMINATED BY '\001'
STORED AS TEXTFILE TBLPROPERTIES ('SASFMT:y'='CHAR(7)')
```

The SAS/ACESS to Hadoop engine recognizes that `SASFMT:` is a tag when it reads the Hivetable properties. It provides SAS with information that is needed to correctly map the HDFS file into a tabular SAS representation. When you load your Hadoop data with SAS, SAS automatically creates table property metadata for you, as shown above.

2. Pipe the SAS data to Hadoop and put it in a tmp file. If you have a LIBNAME to another data source in the SAS SET statement, you would be sending that data to Hadoop to be loaded into the Hivetable. The results of Step 2 are a file that resembles this one:

```
/tmp/HADOOP1_2012-02-17-12-35-26-194
```

3. Load the data that is piped over into the newly created Hive table.

```
LOAD DATA INPATH '/tmp/HADOOP1_2012-02-17-12-35-26-194'
OVERWRITE INTO TABLE `HADOOP1`
```

This process creates the internal Hivedata store that the HADOOP1Hivetable references.

- If you have an existing Hive table for which you want to provide SAS metadata so that you can persist the data accurately in SAS, you could use this code:

```
proc sql;
connect to Hadoop(<connection info>);
exec (alter table hadoop2 ('SASFMT:y'='CHAR(10)' );
quit;

data a;
set lib.hadoop2;
run;
```

Why do you need accurate Hive table character length properties? When you execute a HiveQL describe of the Hivetable the maximum length of the string column is not returned. Therefore, when you extract the data into SAS, you get the default column length of 32K bytes, which can cause significant data expansion. To accurately extract data into a SAS tabular form requires maximum column-length metadata for all string columns.See the Resources section for more information about HiveQL.

PROC SQL

The ability to write implicit pass-through SQL code that can execute on the DBMS has significant gain. To enable processing to flow to the DBMS, we have implemented additional PROC SQL changes. PROC SQL provides two types of interfaces to your DBMS. The first is implicit pass-through SQL , where the SAS SQL that you write is first parsed and textualized into DBMS SQL. This SAS code represents Oracle implicit pass-through SQL.

```
/*--- Implicit pass-through SQL ---*/
libname x oracle user=sasuser password=sasuser path=ora_path;
proc sql;
select * from x.dbtab where employee_num < 2500;
quit;
```

The second PROC SQL implementation is explicit pass-through SQL , where the SQL that you write is passed directly to the DBMS for processing. This SAS code represents Teradata explicite pass-through SQL.

```
/*--- Explicit pass-through SQL ---*/
proc sql;
connect to teradata (user=sasuser password=sasuser);
select * from connection to teradata (select * from dbtab);
exec (drop table dbtab) by teradata;
quit;
```

For 9.3M1 we have provided additional parse/textualization processes to support implicite pass-through SQL. We added these components to PROC SQL to enable push-down of processing into the DBMS. Here are some examples of the changes that now execute in the DBMS.

DELETE, using impliciat pass-through

```
/*--- Connect to your DBMS ---*/
libname x <dbms> server=xx user=sasuser password=sasuser;

/*--- Delete all rows in the DBMS table using a WHERE clause ---*/
proc sql;
delete from x.dbtabx;
delete from x.dbtab2 where x < 10;
run;
```

End-to-End Example Using Implicit Pass-Through

```
/*--- End-to-end example ---*/
libname user db2 user=sasuser pass=sasuser dsn=sasuser;

proc delete data=db2.sqltab;run;
proc delete data=db2.rdtab;run;
proc delete data=db2.new_sqltab;run;

data db2.sqltab;
     format c $17.;
     do cnt=1 to 1000;
          n=cnt*10;
            c=cnt||'_char'; * 'char_'||cnt;
            output;
      end;
run;
```

```
data db2.rdtab;
     num=82; output;
     num=730; output;
     num=105; output;
     num=233; output;
     num=956; output;
     num=411; output;
     num=899; output;
     num=627; output;
     num=333; output;
     num=563; output;
run;

%dbtrace(',,,d');
options dbidirectexec;

/* Updating using sub-selects */
/* Updates 399 rows           */
proc sql;
     update db2.sqltab
        set n = (select num
                   from db2.rdtab
                  where num between 300 and 400),
            c='new'
      where cnt > 100 and cnt < 500;
quit;

/* Inserting multiple rows at once */
/* Adds 3 rows                     */
proc sql;
     insert into db2.sqltab values ('char_1001', 1001, 10010,)
                        values ('char_1005', 1005, 10050,)
                                    values ('char_1007', 1007, 10070,);
quit;

/* Passing down a function */
/* deletes 7 rows          */
proc sql;
     delete from db2.sqltab
     where cnt in (select ceil(num / 2)
                      from db2.rdtab
                             where num between 200 and 900);
quit;

/* Passing down a CREATE TABLE AS SELECT with a data set option */
/* The option doesn't prevent implicit pass-through SQL
   from executing. */
/* Supported by Oracle and Teradata at 9.3.                     */
/* adds 500 rows                                                */
proc sql;
     create table db2.new_sqltab (dbcreate_table_opts='primary index (cnt)')
      as select * from db2.sqltab
     where cnt > 300
          and not exists (select 1
                              from db2.rdtab
                                    where n*10 between 50010 and 70040);
quit;

%dbtrace(',,,');

proc delete data=sqltab;run;
proc delete data=rdtab;run;
proc delete data=new_sqltab;run;
```

This requires that you set the DBIDIRECTEXEC option, as described in the SAS/ACCESS documentation. The power of the PROC SQL implicit pass-through facility continues to improve performance as we push more data down to the DBMS for processing.

PC FILES

For SAS 9.3M1, we added many features and functions to the PC Files product in many different areas. One enhancement focus area is the Microsoft Office XLSX format processing. With the progression to 64-bit Windows and 64-bit Microsoft Office, the interface to XLSX becomes more important. Here are specific enhancements when converting to XLSX format processing:

- Increased extraction speed: Reading XLSX data occurs much faster than reading using the JET or ACE driver.
- Enhanced support for internationalization, which include DBCS for Japanese and Chinese and other unusual text combinations
- XLS support restriction for mixed-case columns has been eliminated
- XLS support restriction for maximum number of columns, which was 255 has been eliminated

You can use this code to run SAS to access a Microsoft XLSX data source.

```
/*--- Read XLSX data into SAS ---*/
proc import datafile='filename.xlsx'
out=work.a dbms=xlsx;
getnames=yes;
sheet='tab_1';
run;

/*--- Write data from SAS to XLSX ---*/
proc export data=work.a
outfile='filename2.xlsx' dbms=xlsx;
sheetname='tab_1';
run;
```

As the interface to XLSX grows, you will see additional components and features have been added to complete the interface to this PC Files data type. This would include a LIBNAMEengine, the ability to output to multiple sheets per Excel file, and more more enhancements to national language support (NLS).

SAS/ACCESS ENGINE READ PERFORMANCE

Read performance time has always been a concern with SAS/ACCESS products. In 1984, if we could not have read the data that you needed to analyze, it would have been a problem. In 2012, if we cannot read the data fast enough for you to analyze, it could still be a problem. Given what has happened and is happening in the industry, speedy data access iscritical. To help meet this challenge requires some new approaches. The first of those, slated for SAS 9.3M2, are pipeline data reads. The secret sauce is to increase read performance by reading ahead and preparing data for the pipeline while you process your SAS job. Figure 3 demonstrates what we have seen in our experiments. Of course, as with any performance statement, your numbers can vary.

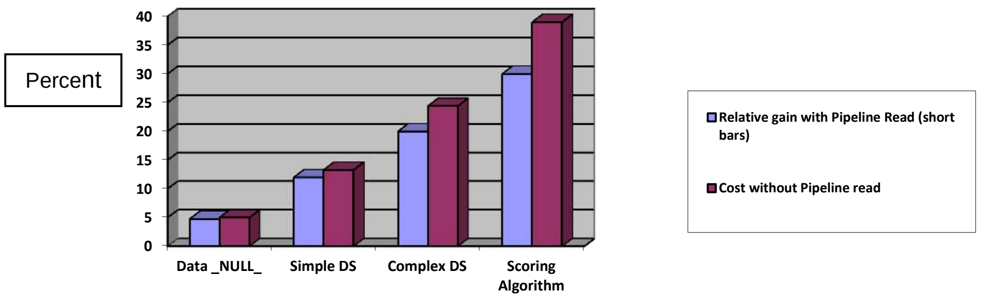

Figure 3: Performance With and Without Pipeline Read

The takeaway is that the more you process data in SAS, the faster it can be pipelined into the SAS/ACCESS read process. Running one of our SAS Enterprise Miner® scoring algorithms showed a 30 percent gain when using pipeline reads.

Here is an example of an additional performance improvement that you might realize by tuning your ETL processes from multiple steps to a single one.

1. Evaluate the extraction data process if you are pulling data into SAS
2. If the extraction and compute step could be combined you will see a greater performance increase over the two step process.

The Pipeline read feature is designed to give you an automated way to gain read performance without having to tweak and experiment with the READBUFF option. Note, READBUFF, has been a performance enhancer for years by buffering data transmission between the client and DBMS server. This feature is slated for SAS 9.3M2.

TERADATA

The SAS/ACCESS engine for Teradata contains many features and functions that might go unnoticed unless you experience a specific need and consult SAS Technical Support. Here are some highlights.

Temporal Data Type Support

The Teradata engine continues to evolve as the DBMS evolves. The implementation of temporal data support gives SAS users new methods to explore when evaluating data that is stored in Teradata. Consider this use case.

```
 /* The PERIOD datatypes require the Teradata V13 server or later */
libname x teradata user=testuser pw=testpw server=td13;

/* Create a table with the PERIOD(DATE) data type.
   The period data type represents a time duration. */
data x.mytest(DBTYPE=(validity='PERIOD(DATE)'));
i=1; validity='(1973-02-03, 9999-12-31)'; output;
run;

/* Read from a table with a PERIOD data type? */
proc print data=x.mytest;
```

```
run;

/* Use Fastload to load a table with a PERIOD data type. */
proc delete data=x.mytest;
run;

data x.mytest(DBTYPE=(validity='PERIOD(TIMESTAMP)')
   FASTLOAD=YES TPT=NO);
i=1; validity='(1970-01-05 01:02:03.123, 1970-01-05 05:06:07.456)';
   output;
run;

/* Temporal support starts in Teradata V13.10. */
libname x teradata user=testuser pw=testpw server=td1310;

/* Create a table with the PERIOD(DATE) data type.
   Adding the VALIDTIME qualifier makes the table a temporal table
   and Teradata will maintain information with respect to time */
data x.mytest(DBTYPE=(validity='PERIOD(DATE) VALIDTIME'));
i=1; validity='(1973-02-03, 1999-12-31)'; output;
i=2; validity='(2000-01-01, 2011-01-01)'; output;
i=3; validity='(2011-01-02, 9999-12-31)'; output;
run;

/* Can we read a PERIOD data type?
   You must select the row with i=2. */
proc print data=x.mytest(TEMPORAL_QUALIFIER='CURRENT VALIDTIME');
run;

/* Consider data as of 1995-01-01. */
libname x teradata user=testuser pw=testpw server=td1310
          TEMPORAL_QUALIFIER='VALIDTIME AS OF DATE '1995-01-01' '

/* Row with i=1 is returned. */
proc print data=x.mytest(DBSLICEPARM=ALL);
run;
```

If you process time series data or need to organize and review data based on historical content, you should look into temporal data types in Teradata.

Query Banding

The ability to apply a tag to a LIBNAME or data set option has been available since SAS 9.3. Some users have implemented processes to help track consumers of the Teradata system. Consider these tips as you construct or build SAS code for use by your consumers.

```
/* Turn on query logging */
proc sql;
  connect to teradata (server=dbc user=joe pass=XXXXXX
                      mode=teradata);
  execute (BEGIN QUERY LOGGING LIMIT SQLTEXT=10000 ON joe) by teradata;
quit;

data tdlib.test_tpt_4 (multistmt=yes   tpt_pack_maximum=yes
query_band="GROUP=Stat22;JOBID=Job99;IMPORTANCE=BatchAnalytics");
  set work.big10K;
run;

/* Turn off query logging */
proc sql;
  connect to teradata (server=dbc user=joe pass=XXXXXX
                      mode=teradata);
  execute (END QUERY LOGGING ON joe) by teradata;
```

```
   quit;
```

Query Interrupt

The ability to stop a Teradata query running on the Teradata server has been in demand for some time. With SAS 9.3M2, you will be able to cancel submission of a Cartesian-product Teradata SQL statement from SAS Enterprise Guide®, SAS Data Managment Studio, or the SAS Explorer window without database administrator intervention. You can cancel the Teradata SQL Query by interrupting the process that SAS is running. For example, if you submit a long-running query from DataFlux® Data Managment Studio and need to stop it,you press the BREAK button at the top of the window and select the Cancel Submitted Statements option.

LOGDB for Teradata Protocol Transporter (TPT)

The primary purpose of LOGDB is to route log and error tables to a database other than the target database. Teradata recommends using a spool reserve database to hold the log and error tables away from the database where the target tables are stored. Without LOGDB, the only way to do this would be tedious, and each error and log table must be named manually, as shown below. It is also not possible to redirect the target table to a database other than the default for the user—zoom, in this example.

```
   libname trlib teradata user=zoom password=wizard;
   /* Redirecting the log and error tables to spool-reserve
      database zoom2 */
   data trlib.sgftest1 ( fastload=yes tpt=yes
                    TPT_ERROR_TABLE_1=zoom2.abc123_ET
                     TPT_ERROR_TABLE_2=zoom2.abc123_UV
                     TPT_LOG_TABLE=zoom2.abd123_RS
                     TPT_WORK_TABLE=zoom2.abc123_WT
                     TPT_TRACE_LEVEL=2
                     TPT_TRACE_OUTPUT=tpttrace);

     do i=1 to 20;
        output;
     end;

   run;
```

With LOGDB, you can redirect all tables as needed.

```
   /* Target table goes to the zoom5 database, the log and error tables go to zoom2 */

       libname trlib teradata user=zoom password=wizard schema=zoom5 logdb=zoom2;
       /* Re-directing the log and error tables to spool-reserve database zoom2 */
       data trlib.sgftest1 ( fastload=yes tpt=yes);
         do i=1 to 20;
            output;
         end;
   run;
```

DB2

DB2 V10 on z/OS

SAS has been verified using DB2 V10. Some new features that SAS can take advantage of include temporal table support so that you can create and query business, system, and BITETEMPORAL tables and manipulate them through SAS DATA steps and implicit pass-through SQL. Here is an example.

```
   libname user db2;
   %dbtrace(',,,d');

   /* create system temporal table */
   proc sql;
      create table policy (dbnull=(_all_=no) temporal=system)
              (id int,
         vin varchar(10),
   annual_mileage int,
          rental_car char(1),
         coverage_amt int);
   quit;
```

Here is the resulting SAS log.

```
DB2_2: Executed: on connection 1

CREATE TABLE POLICY (id FLOAT NOT NULL, vin CHAR(10) NOT NULL,
   annual_mileage FLOAT NOT NULL,
rental_car CHAR(1) NOT NULL, coverage_amt FLOAT NOT NULL,
sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END NOT NULL,
trans_start TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION
   START ID IMPLICITLY HIDDEN,
PERIOD SYSTEM_TIME (sys_start, sys_end))

DB2_4: Executed: on connection 1

COMMIT WORK

DB2_5: Executed: on connection 1

CREATE TABLE POLICY_history LIKE POLICY

DB2_6: Executed: on connection 1

COMMIT WORK

DB2_7: Executed: on connection 1

ALTER TABLE POLICY ADD VERSIONING USE HISTORY TABLE POLICY_history
```

Extended Join Support:

Before SAS 9.3, the maximum number of DB2 tables that you could specify in a join was limited to 32. That limit has been raised to 64.

AUTHDOMAIN Support

This provides the ability to retrieve credential information from the SAS metadata repository. It lets you store user and password credentials in a secure location so they're not exposed in SAS code or SAS logs. If you need to modify credentials, you change them in the SAS metadata repository, whichdoes not require a change in SAS code or jobs. Here is an example.

```
libname x db2 authdomain=acme;
```

ID and Password Support on the LIBNAME Statement

DB2 on z/OS does not require a user ID and password to connect to the DBMS becase it uses the TSO login ID. Before SAS 9.3, if you wanted to use another ID to connect to DB2 on z/OS, you had to log off and log in using that ID. This is no longer required. With SAS 9.3, you can specify connection information on the LIBNAME statement:

```
libname x db2 user=xx1 password=xx1;
```

The user and password are not validated when you enter the LIBNAMEstatement. Validation occurs when the first statement is run against the DB2 DBMS.

Correlation ID Provided with the DB2 Connection Using RRSAF

Before SAS 9.3, database connections that were established through the RRSAF facility could not be identified through the DB2 `-display thd(*)` command. That capability has been integrated into SAS to enable database administrators.

GREENPLUM

Bulkload Support Using Pipes on UNIX

Bulkloading capabilities have been extended on UNIX platforms to loadtables load pipes. To leverage this new feature, you must add the BL_USE_PIPE=YES option when starting the bulkload facility:

```
data user.dbtab (bulkload=yes
                  bl_format='CSV'
                  bl_quote='"'
                  bl_host='mypc.na.acme.com'
                  bl_port='8080'
                  bl_protocol='gpfdist'
                  BL_USE_PIPE=YES);
     set work.worktab;
run;
```

We have seen a 30 percent average performance gain in load speed when using pipes rather than traditional flat files. Greenplum does not currently support pipes on Windows platforms.

Bulkload Support to Load Tables from an Existing Data Set

The new BL_DATAFILE_EXISTS bulkload option supports loading tables from an existing flat file. Valid values are NO (the default) and YES. When set to YES, no flat file is created, and the one that is specified through the BL_DATAFILE option is used to load the table.

```
data user.dbtab (bulkload=yes
                  bl_format='CSV'
                  bl_quote='"'
                  bl_datafile='dbtab.dat'
                  bl_host='mypc.na.acme.com'
                  bl_port='8080'
                  bl_protocol='gpfdist'
                  BL_DATAFILE_EXISTS=YES
                  bl_delete_datafile=no);
     set work.worktab;
run;
```

Bulkload Support to Add Octal Escape Strings

Values for the BL_ESCAPE, BL_DELIMITER, and BL_QUOTE options can now be expressed in octal format. It lets you use a wider range of characters to quote or delimit text while avoiding load failures if it's possible that the same characters might be present in the data.

```
proc sql;
create table x.GPBLKTAB
  (BULKLOAD=YES
   BL_PORT='8080'
   bl_host='mypc.na.acme.com'
   BL_PROTOCOL="gpfdist"
   bl_format='CSV'
   BL_DELIMITER="E'\24'"
   BL_QUOTE="E'\1'"
   bl_datafile='dbtab.dat'
   bl_delete_datafile=no )
   as select * from work.GPBLKDAT;
quit;
```

SOME DBMS PEARLS

With so many knobs to turn when tuning SAS and your DBMS, here are some areas where you might want to invest your time.

Fresh DBMS Table Statistics

If you modify tables that you access with SAS, ensure that the table statistics are accurate. The DBMS query optimizer and even SAS data integration jobs rely on fresh statistics to aid query performance.

PROC SQL Implicit Pass-Through

If you write SAS SQL (PROC SQL implicit pass-through), ensure that the aggregation components and WHERE components are pushed to the DBMS. One way to do this is to remove the PROC SQL statement in question and follow these steps.

1. Turn on SASTRACE= to see what is passed to the DBMS:

   ```
   options sastrace=',,d,d' sastraceloc=saslog nostsuffix;
   ```

2. Set the LOOPS parameter on the PROC SQL statement to indicate no back-down processing:

   ```
   proc sql loops=0;
   select avg(a) form lib.db where b > 1000;
   …
   ```

Loading data into the DBMS

SAS provides many options to help you load data into your DBMS from other DBMSs, data sources, and SAS data sets. We have seen optimal performance when executing loads into a DBMS by using these methods.

- Pipe load: Pipes data to the DBMS server from SAS, does not require an intermediate client file.
- Bulkload: Loads data and control files, then executes the load; requires an intermediate file.
- MultiStatement insert: Batches inserts into multiple statements in one buffer.
- Insert statement: Should only be considered for adding small amounts of data.

If you have data in the DBMS, consider the statements that will move data from one table to another:

- CREATE TABLE AS SELECT
- INSERT INTO AS SELECT

This class of SQL statements is supported in both PROC SQL explicit pass-through and implicit pass-through.

CONCLUSION

SAS/ACCESS products continue to grow and evolve to meet customer demands. SAS/ACESS engines and functionality enhancements expand the data you can access and process. Hadoop brings new, yet not unknown data challenges. Those challenges include reading and writing data of size. The experiences of our Hadoop customers will help shape the engine in both the SAS 9.3M2 and SAS 9.4 releases. Consider DBMS pearls as you interact with your DBMS when using SAS. Investment in understanding how the DBMS interacts with SAS is key.

RESOURCES

- *Base SAS® Procedures Guide* (see PROC SQL)
- *SAS/ACCESS for Relational Databases: Reference*
- *SAS® Interface to Hadoop*
- Hadoop references: http://hadoop.apache.org (see "What Is Apache Hadoop?")
- Hadoop Hive reference information: https://cwiki.apache.org/confluence/display/Hive/LanguageManual
- Hadoop HiveDDL statements: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL

CONTACT INFORMATION

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

Howard Plemmons
SAS Campus Drive
SAS Institute Inc.
E-mail: Howard.Plemmons@sas.com