Methods of Storing SAS® Data into Oracle Tables

Lois Levin, Independent Consultant, Bethesda, MD

ABSTRACT

There are several ways to create a DBMS table from a SAS dataset. This paper will discuss the SAS methods that may be used to perform loads or updates, a comparison of times elapsed, and the INSERTBUFF option and the BULKLOAD option that can be used to perform multiple row inserts and direct path loads to optimize performance.

INTRODUCTION

You may need to store data into a database to perform a full database load, to update an existing table, or you may just be creating a temporary table for the purpose of joining to an existing table. But whenever you store data into a database you will need to consider methods that are fast and also safe and auditable to protect the integrity of the database. We will discuss the following 4 methods for creating or updating a table. They are

1) PROC DBLOAD
2) The DATA step
3) PROC SQL
4) PROC APPEND

Then we will see how these methods can be optimized using

1) Multiple row inserts
2) Bulk loads

All examples use Oracle 9i and the native Oracle loader SQL*Loader, Release 9.0.1.0.0. All text will refer to Oracle but most examples will be compatible with other DBMS's unless noted.

All examples were run using SAS V8.2.

All examples were run on a Compaq Tru64 UNIX V5.1 (Rev. 732).

The sample dataset contains 10 million observations and 20 variables.

The database user, password, and path names have been previously defined in %let statements and are referenced here as &user, &pass, and &path.

TRANSACTIONAL UPDATING USING PROC DBLOAD

PROC DBLOAD performs updates using the transactional method of update. That is, it executes a standard Oracle insert, one row at a time. An example of code for loading a sample file of 10 million records and 20 variables is:

```
*-----------------------*;
*    Create a table     *;
*    with PROC DBLOAD   *;
*-----------------------*;
proc dbload dbms=oracle data=testin;
     path  = &path;
     user  = &user;
     pw    = &pass;
     table = lltemp;
     commit= 1000000;
     limit = 0;
     load;
run;
```

Note that the COMMIT statement is set to 1000000, which is a very large number for this example. It indicates when to do a save after inserting a given number of rows. A small number for a COMMIT will save often and a large number will issue fewer saves. This large value for the COMMIT will result in a faster load but not necessarily the safest. You will have to choose the appropriate amount for the COMMIT depending on your data and the system environment.

The log includes the following statements:

```
NOTE: Load completed. Examine statistics below.
NOTE:  Inserted (10000000) rows into table (LLTEMP)
NOTE:  Rejected (0) insert attempts see the log for details.

NOTE: There were 10000000 observations read from the dataset WORK.TESTIN.
NOTE: PROCEDURE DBLOAD used:
      real time           4:58:13.51
      cpu time              47:47.14
```

USING THE LIBNAME ENGINE

In SAS Version 7/8 the LIBNAME engine allows the user to specify the database using the LIBNAME statement. The database can then be referenced by any DATA step or PROC statement as if it were a SAS library and a table can be referenced as a SAS dataset.

So, just as a dataset can be created using the DATA step, a DBMS table can be created in exactly the same way.

Here is an example:

```
*-----------------------*;
*    Create a table     *;
*    with DATA Step     *;
*-----------------------*;
libname cdwdir oracle user = &user
                      password = &pass
                      path = &path;

data cdwdir.lltemp;
     set testin;
run;
```

The log for this example includes:

```
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: There were 10000000 observations read from the data set WORK.TESTIN.
NOTE: The data set CDWDIR.LLTEMP has 10000000 observations and 20 variables.
NOTE: DATA statement used:
      real time           46:14.96
      cpu time            15:25.09
```

Loads can also be performed using the LIBNAME statement and PROC SQL.

Here is an example:

```
*-----------------------*;
*    Create a table     *;
*    with PROC SQL      *;
*-----------------------*;
libname cdwdir oracle user = &user
                      password = &pass
                      path = &path;
```

```
proc sql;
     create table cdwdir.lltemp as
     select * from testin;
quit;
```

The log includes;

```
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: Table CDWDIR.LLTEMP created, with 10000000 rows and 20 columns.
21         quit;
NOTE: PROCEDURE SQL used:
      real time           49:40.27
      cpu time            16:12.06
```

A note about SQL Pass-Through: Normally, using the LIBNAME statement implies that the LIBNAME engine is in effect and therefore all processing is performed in SAS. But since we are writing to the database, it would seem that we should use the SQL pass-through method that moves all processing into the DBMS. In fact when we are doing these loads, since all processing must happen in the DBMS, both approaches are actually equivalent. So for the sake of simplicity, all examples here will use the LIBNAME statement.

The other PROC that may be used is PROC APPEND. PROC APPEND should be used to append data to an existing table but it can also be used to load a new table.

Here is an example using PROC APPEND:

```
*------------------------*;
*    Create a table      *;
*    with PROC APPEND    *;
*------------------------*;
libname cdwdir oracle user = &user
                      password = &pass
                      path = &path;

proc append base=cdwdir.lltemp
            data=testin;
run;
```

The log includes:

```
NOTE: Appending WORK.TESTIN to CDWDIR.LLTEMP.
NOTE: BASE data set does not exist. DATA file is being copied to BASE file.
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: There were 10000000 observations read from the data set WORK.TESTIN.

NOTE: The data set CDWDIR.LLTEMP has 10000000 observations and 20 variables.
NOTE: PROCEDURE APPEND used:
      real time           43:48.65
      cpu time            15:45.76
```

Note the comment generated by the LIBNAME engine. If your SAS data set includes labels, formats, and lengths, they cannot be written to the DBMS table. These are SAS constructs and are not understood by any DBMS. If your data set contains indices, they also cannot be loaded into the DBMS. You will have to remove the indices, load the table, and then rebuild the indices using the DBMS software.

MULTIPLE ROW INSERTS

The loads we have just seen are all transactional loads and insert one row (observation) at a time. Loads using the LIBNAME engine can be optimized by including the INSERTBUFF option in the LIBNAME statement. This speeds the update by allowing you to specify the number of rows to be included in a single Oracle insert.

The INSERTBUFF option uses the transactional insert method and is comparable to using the Oracle SQL*Loader Conventional Path Load.

Note that this is different from the COMMIT statement in DBLOAD. The COMMIT statement just executes a save after each individual row insert. INSERTBUFF actually inserts multiple observations within one Oracle insert activity.

To use the INSERTBUFF option, just include it in the LIBNAME statement like this:

```
libname cdwdir oracle user = &user
                      password = &pass
                      path = &path
                      INSERTBUFF=32767;
```

This example uses 32767, which is the maximum buffer size. In most cases using that maximum can actually slow the system and it is suggested that a smaller value be used. The exact size of the buffer will depend on your operating environment and you will have to experiment to find the best value.

INSERTBUFF with the DATA STEP

If we use the INSERTBUFF option, and then load with a DATA step, the log is as follows:

```
18         data cdwdir.lltemp;
19              set testin;
20         run;

NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: There were 10000000 observations read from the data set WORK.TESTIN.
NOTE: The data set CDWDIR.LLTEMP has 10000000 observations and 20 variables.
NOTE: DATA statement used:
      real time           14:30.71
      cpu time            8:33.30
```

INSERTBUFF with PROC SQL

If we use the INSERTBUFF option, and load with PROC SQL, the log includes:

```
18  proc sql;
19       create table cdwdir.lltemp as
20       select * from testin;
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: Table CDWDIR.LLTEMP created, with 10000000 rows and 20 columns.

21    quit;
NOTE: PROCEDURE SQL used:

      real time           11:41.02
      cpu time            8:40.06
```

INSERTBUFF with PROC APPEND

If we use the INSERTBUFF option and PROC APPEND, the log includes:

```
18    proc append base=cdwdir.lltemp
19                data=testing;
20    run;

NOTE: Appending WORK.TESTIN to CDWDIR.LLTEMP.
NOTE: BASE data set does not exist. DATA file is being copied to BASE file.

NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: There were 10000000 observations read from the data set WORK.TESTIN.
NOTE: The data set CDWDIR.LLTEMP has 10000000 observations and 20 variables.
NOTE: PROCEDURE APPEND used:
      real time           19:29.32
      cpu time             8:19.73
```

BULKLOAD

In Version 8, the SAS/ACCESS engine for Oracle can use the BULKLOAD option to call Oracle's native load utility, SQL*Loader. It activates the DIRECT=TRUE option to execute a direct path load which optimizes loading performance.

Direct path loading, or bulk loading, indicates that SQL*Loader is writing directly to the database, as opposed to executing individual SQL INSERT statements. It is a direct path update in that it bypasses standard SQL statement processing and therefore results in a significant improvement in performance.

The BULKLOAD option may also be used with DB2, Sybase, Teradata, ODBC, and OLE.DB. In all cases it makes use of the native load utility for that particular DBMS. Bulk loading is available with the OS/390 operating system in SAS Version 8.2.

BULKLOAD is not available with the DBLOAD procedure (except with Sybase where you may use the BULKCOPY option). DBLOAD can only use the transactional method of loading and BULKLOAD uses the direct load method. BULKLOAD can be used with a DATA step, PROC SQL and with PROC APPEND to load or update data into tables.

There is some overhead involved in setting up the loader so BULKLOAD may not be as efficient for small table loads as it is for very large tables. You will need to test this with your particular application.

BULKLOAD with the DATA Step

Here is an example using the BULKLOAD option with the DATA step:

```
*----------------------*;
*    BULKLOAD          *;
*    with DATA Step    *;
*----------------------*;
libname cdwdir oracle user = &user
                      password = &pass
                      path = &path;

data cdwdir.lltemp(BULKLOAD=YES
                   BL_DELETE_DATAFILE=YES);
     set testin;
run;
```

The BULKLOAD=YES option invokes the DIRECT=TRUE option for SQL*Loader. Without this, a conventional load will be performed.

Although there is a LIBNAME statement, we are not actually using the LIBNAME engine in SAS. The BULKLOAD=YES statement directs Oracle SQL*Loader to take over. When this happens, we are no longer in SAS. All of the log statements and messages will come from Oracle. When SQL*Loader is finished, SAS takes over again.

BULKLOAD will create several permanent files. They are the .dat file, the .ctl file and the .log file. The .dat file is a flat file containing all of the data to be loaded into Oracle. The .ctl file contains the Oracle statements needed for the load. The .log file is the SQL*Loader log. These files will remain after the load is completed, so if you don't want to keep them, you must remove them explicitly. By default they will be named BL_<*tablename*>_0.dat, BL_<tablename>_0.log, etc. and they will be located in the current directory. The BULKLOAD options BL_DAT= and BL_LOG= allow you to specify fully qualified names for these files and therefore you can use any name and any location you like.

The BL_DELETE_DATAFILE=YES option is used here because the .dat file can be very large (as large as your data). You need to be sure you have enough storage space to hold it but you probably do not want to keep it after the load is complete, so this option will delete it automatically after loading. (Other DBMS's generate different default data file names (.ixt in DB2) and set the default for the BL_DELETE_DATAFILE option to YES so you don't have to specify it yourself.)

The SAS log file for a bulk load is much longer than a standard SAS log because it will contain the SQL*Loader log statements as well as the SAS log statements. The log includes the following:

```
***** Begin:  SQL*Loader Log File *****
.
.     (statements omitted)
.
**** End:    SQL*Loader Log File *****n

NOTE:
************************************
 Please look in the SQL*Loader log file for the load results.
 SQL*Loader Log File location: -- BL_LLTEMP_0.log --
 Note: In a Client/Server environment, the Log File is located on the Server. The log
 file is also echoed in the Server SAS log file.
**********************************

NOTE: DATA statement used:
      real time           15:26.99
      cpu time            2:19.11
```

BULKLOAD with PROC SQL

Here is an example using BULKLOAD with PROC SQL. The BL_LOG= option is included to specify the name and location for the BULKLOAD log.

```
*----------------------*;
*    BULKLOAD          *;
*    with PROC SQL     *;
*----------------------*;
libname cdwdir oracle user = &user
                      password = &pass
                      path = &path;

proc sql;
     create table cdwdir.lltemp(BULKLOAD=YES
                                BL_LOG="/full/path/name/bl_lltemp.log"
                                BL_DELETE_DATAFILE=YES)
     as select * from testin;
quit;
```

The SAS log for this example includes:

```
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.

***** Begin:  SQL*Loader Log File *****
.
.     (statements omitted)
.
***** End:    SQL*Loader Log File ****%n
```

```
NOTE: Table CDWDIR.LLTEMP created, with 10000000 rows and 20 columns.
NOTE:
************************************
 Please look in the SQL*Loader log file for the load results.
 SQL*Loader Log File location: -- /full/path/name/bl_lltemp.log --

 Note: In a Client/Server environment, the Log File is located on the Server.

The log file is also echoed in the Server SAS log file.
************************************

22         quit;

NOTE: PROCEDURE SQL used:
      real time           15:21.37
      cpu time            2:34.26
```

BULKLOAD with PROC APPEND

This next example shows the same load using PROC APPEND. Here we have also included the SQL*Loader option ERRORS= to tell Oracle to stop loading after 50 errors. (The default is 50 but it is included here for an example.) All SQL*Loader options are proceeded by the BL_OPTIONS= statement and the entire option list is enclosed in quotes. (Note that the option ERRORS= should be spelled with an 'S'. The documentation indicating ERROR= is incorrect.) Further information on the SQL*Loader options can be found in the Oracle documentation.

Sample code is:

```
*----------------------*;
*    BULKLOAD          *;
*    with PROC APPEND  *;
*----------------------*;
libname cdwdir oracle user = &user
                      password = &pass
                      path = &path;

proc append base=cdwdir.lltemp (BULKLOAD=YES
                                BL_OPTIONS='ERRORS=50'
                                BL_DELETE_DATAFILE=YES)
            data=testin;
run;
```

The SAS log includes:

```
NOTE: Appending WORK.TESTIN to CDWDIR.LLTEMP.
NOTE: BASE data set does not exist. DATA file is being copied to BASE file.

NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: There were 10000000 observations read from the data set WORK.TESTIN.
NOTE: The data set CDWDIR.LLTEMP has 10000000 observations and 20 variables.

**** Begin:  SQL*Loader Log File *****
.
.     (statements omitted)
.
**** End:    SQL*Loader Log File ***%n

NOTE:
************************************
```

```
Please look in the SQL*Loader log file for the load results.
 SQL*Loader Log File location: -- BL_LLTEMP_0.log --
 Note: In a Client/Server environment, the Log File is located on the Server.
      The log file is also echoed in the Server SAS log file.
************************************
NOTE: PROCEDURE APPEND used:
      real time           15:20.35
      cpu time            2:02.73
```

VERIFICATION and DEBUGGING

If you need to debug a process or if you are just interested in what is going on between SAS and the DBMS, you can use the SASTRACE option to see all of the statements that are passed to the DBMS. Of course they will vary with the DBMS used. It is recommended that you use SASTRACE with a small test dataset so that you do not see every single internal COMMIT. You may also wish to direct the output to the log or you might get excessive screen output.

Here is how to invoke the trace and direct it to the log file.

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

BULKLOAD OPTIONS

The BULKLOAD statement allows several options to control the load and the output from the load. The examples above show the use of BL_DELETE_DATAFILE to delete the .dat file after loading, BL_LOG_= to name and locate the .log file, and BL_OPTIONS to begin the list of SQL*Loader options. Note that the BULKLOAD options and SQL*Loader options are different. The BULKLOAD options are documented in the SAS documentation. The SQL*Loader options are documented in Oracle documentation. The list of BULKLOAD options follows.

BL_BADFILE
- creates file *BL_<tablename>.bad* containing rejected rows

BL_CONTROL
- creates file *BL_<tablename>.ctl*
  containing DDL statements for SQLLDR control file

BL_DATAFILE
- creates file *BL_<tablename>.dat*
  containing all of the data to be loaded.

BL_DELETE_DATAFILE
- =YES
  will delete the .dat file created for loading
- =NO
  will keep the .dat file.

BL_DIRECT_PATH
- =YES
  Sets the Oracle SQL*Loader option DIRECT to TRUE, enabling the SQL*Loader to use Direct Path Load to insert rows into a table. This is the default.
- =NO
  Sets the Oracle SQL*Loader option DIRECT to FALSE, enabling the SQL*Loader to use Conventional Path Load to insert rows into a table.

BL_DISCARDFILE
- creates file *BL_<tablename>.dsc*
  containing rows neither rejected nor loaded.

BL_LOAD_METHOD
- INSERT when loading data into an empty table
- APPEND when loading data into a table containing data
- REPLACE and TRUNCATE override the default of APPEND

BL_LOG
- creates file *BL_<tablename>.log* containing the SQL*Loader log

BL_OPTIONS
- begins list of SQL*Loader options. These are fully documented in the Oracle documentation.

BL_SQLLDR_PATH
- path name of SQLLDR executable. This should be automatic.

SUMMARY

Here is a summary table of the load times for all of the different methods provided. Please note that these programs were run on a very busy system and the real times vary greatly depending on the system load. The CPU time is more consistent and so is a better unit of comparison.

	TRANSACTIONAL	MULTI-ROW	BULKLOAD
DBLOAD			
Real Time	4:58:13.51		
CPU Time	47:47.14		
DATA Step			
Real Time	46:14.96	14:30.71	15:26.99
CPU Time	15:25.09	8:33.30	2:19.11
PROC SQL			
Real Time	49:40.27	11:41.02	15:21.37
CPU Time	16:12.06	8:40.06	2:34.26
PROC APPEND			
Real Time	43:48.65	19:29.32	15:20.35
CPU Time	15:45.76	8:19.73	2:02.73

CONCLUSION

Optimizing with multiple-row inserts and bulk loading is much faster than the transactional method of loading and far more efficient than the old DBLOAD process.

BULKLOAD uses much less CPU time and is the fastest method for loading large databases.

REFERENCES

SAS® OnlineDoc, Version 8

Levine, Fred (2001), “Using the SAS/ACCESS Libname Technology to Get Improvements in Performance and Optimization in SAS/SQL Queries”, *Proceedings of the Twenty-Sixth Annual SAS ® Users Group International Conference, 26.*

CONTACT INFORMATION

The author may be contacted at
Lois831@hotmail.com