Paper SAS409-2017

An Insider's Guide to Fine-Tuning Your CREATE TABLE Statements Using SAS® Options

Jeff Bailey, SAS Institute Inc.

ABSTRACT

The SAS® code looks perfect. You submit it and to your amazement, there is a problem with the CREATE TABLE statement. You need to change the table definition, ever so slightly, but how? Explicit pass-through? That's not an option. Fortunately, there are a handful of SAS options that can save the day. This paper will cover everything you need to know in order to adjust your SAS-generated create table statements using SAS options.

This paper covers the following SAS options:

- DBIDIRECTEXEC
- DBCREATE_TABLE_OPTS=
- POST_STMT_OPTS=
- POST_TABLE_OPTS=
- PRE_STMT_OPTS=
- PRE_TABLE_OPTS=

We use Hadoop and Oracle examples to show why these options can make your life easier. From there, we use real code to show you how to use them.

INTRODUCTION

One of the great things about using SAS with databases is that it hides so much of the complexity of dealing with database management systems (DBMSs). Seriously, a 4-line SAS program can do a lot of work. This is especially true in the data management world. Let's look at a very simple example:

```
libname oradb oracle path=oradb user=sasuser password=mypasswd;

data oradb.cars;
   set sashelp.cars;
run;
```

These 4 lines of code do the following:

1. Connect to an Oracle database.
2. Create an Oracle table named Cars. This table creation step includes figuring out the data types for the columns in the new table.
3. Insert data from Sashelp.Cars into the new Oracle table.

This paper focuses on the CREATE TABLE code that SAS generates in Step 2. Output 1 shows the CREATE TABLE SQL statement that SAS generated in order to create the cars table:

```
ORACLE_4: Executed: on connection 2
CREATE TABLE CARS(Make VARCHAR2 (52),Model VARCHAR2 (160),Type VARCHAR2
(32),Origin VARCHAR2
(24),DriveTrain VARCHAR2 (20),MSRP NUMBER ,Invoice NUMBER ,EngineSize
NUMBER ,Cylinders NUMBER
,Horsepower NUMBER ,MPG_City NUMBER ,MPG_Highway NUMBER ,Weight NUMBER
,Wheelbase NUMBER
,Length NUMBER )
```

Output 1. Generic CREATE TABLE Statement Generated by SAS.

Notice how much code SAS generated for this statement. It is worth thinking about the fact that our SAS code provided no clues as to how to create the table. SAS just did it. This is all well-and-good until you need to make a change. For example, suppose you need to create the table in a specific Oracle tablespace or suppose you need to tell Hive that you want your data stored using the Parquet file format. How would you do this?

Beginning in the third maintenance release of SAS 9.4, there are SAS data set options that enable you to customize the SQL that is generated to create tables. At first glance, these options look simple, but they have subtle quirks that we must understand to use them effectively. There is no need to worry. Using these options is easy once you see them in action.

The SAS create table options (DBCREATE_TABLE_OPTS=, PRE_STMT_OPTS=, PRE_TABLE_OPTS=, POST_TABLE_OPTS=, and POST_STMT_OPTS=) are supported by the following SAS/ACCESS engines:

- Amazon Redshift
- Aster
- DB2 under UNIX and PC Hosts
- DB2 under z/OS
- Greenplum
- Hadoop
- HAWQ
- Impala
- Informix
- Microsoft SQL Server
- Netezza
- Oracle
- PostgreSQL
- SAP ASE
- SAP IQ
- Teradata
- Vertica

The most important “secret” that you need to know about the SAS create table options is that they enable you to specify text strings that are placed into CREATE TABLE and CREATE TABLE AS SELECT (CTAS) statements that SAS generates. Each option is responsible for placing the text string in a different location in the SQL statement. POST_STMT_OPTS= is interesting because its behavior changes base on whether DBIDIRECTEXEC has been enabled.

Before we dive into the options, it is critical that we understand how SAS generates DBMS-specific SQL code.

PROC SQL IMPLICIT PASS-THROUGH

There is SQL that is generated by SAS, and then there is SQL that is generated by PROC SQL implicit pass-through (PSIP). These concepts are commonly grouped together and called implicit pass-though. There is even a slang version of the name, implicit pass-thru. The common attribute here is that SAS is taking SAS code and translating it into database-specific SQL. In this paper, we refer to this as SAS-generated SQL. This DATA step is an example of SAS-generated SQL, because it uses a LIBNAME statement to access an Oracle DBMS:

```
libname oradb oracle path=oradb user=sasuser password=mypasswd;

data oradb.cars;
   set sashelp.cars;
run;
```

The following PROC SQL code will accomplish the same task:

```
libname oradb oracle path=oradb user=sasuser password=mypasswd;

proc sql;
   connect using oradb;
   create table oradb.cars as
      (select * from sashelp.cars);
quit;
```

Notice that in the previous examples the data we are inserting into our new Oracle table lives in a SAS data set. This fact means that SAS-generated SQL is being used to perform this work. Compare this to the following example:

```
proc sql;
   connect using oradb;
   create table oradb.cars_new
      (select * from ora.cars);
quit;
```

In this example, both the destination and source tables are stored in Oracle, so we want Oracle to do all the work. Specifically, we want the entire CTAS statement to execute on Oracle. Our goal is for PSIP to be used for this operation. Unfortunately, there is a problem. By default, SAS/ACCESS Interface to Oracle does not enable PSIP.

We can turn on PSIP by setting the DBIDIRECTEXEC system option. Once set, the entire CTAS statement will be executed by Oracle. Where supported, NODBIDIRECTEXEC is the default for all SAS/ACCESS products except SAS/ACCESS Interface to Amazon Redshift.

Fortunately, there is an easy way to determine if your CTAS statement is being passed to the database. The following OPTIONS statement turns on PSIP optimization and has PSIP trace messages written to the SAS log:

```
options sql_ip_trace=note msglevel=i dbidirectexec;
```

When we run the previous PROC SQL code, we see a message in the SAS log telling us that the SQL statement was passed directly to the DBMS:

```
SQL_IP_TRACE: The CREATE statement was passed to the DBMS.
```

If we had not turned on PSIP optimization via DBIDIRECTEXEC, we would have seen this message:

```
SQL_IP_TRACE: None of the SQL was directly passed to the DBMS.
```

The SQL_IP_TRACE messages are very helpful, but they will only take us so far. There are times we will need to see the exact SQL that SAS is submitting to the DBMS. This information is extremely valuable when tuning SQL queries.

The SAS options SASTRACE= and SASTRACELOC= tell SAS to write the generated database-specific SQL to the SAS log.

Here is an example:

```
libname myhdp hadoop server=myhorton database=testing
                     user=myuser password=mypasswd;

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

proc sql;
   connect using myhdp;
   select count(*)
     from myhdp.cars
    where make='Toyota';
quit;
```

Output 2 shows the SELECT statement generated by the PROC SQL SELECT statement. It is important to note that the count(*) function and WHERE clause are passed to Hive. This shows that the SAS is issuing an efficient query – the database is doing a great deal of the work.

```
HADOOP_12: Prepared: on connection 4
 select COUNT(*) from `sgfhivedb`.`CARS` TXT_1 where TXT_1.`make` =
'Toyota'
```

Output 2. SAS-generated SELECT Statement.

DATABASE-SPECIFIC CREATE TABLE STATEMENT SYNTAX

The text strings we supply using the table creation options must conform to valid SQL syntax for the DBMS being used. Unfortunately, CREATE TABLE statement syntax can be complicated. It is helpful to have working examples of CREATE TABLE statements. Examples can be found in the SQL language reference for your specific DBMS and on various blogs.

You might be wondering, what happens if I include incorrect SQL syntax or I include it in the wrong SAS option? The answer, your SAS code will fail. Output 3 shows a Hive example:

```
HADOOP_3: Executed: on connection 2
  ---pre_stmt---  CREATE   >>>pre_table<<<  TABLE
`testing`.`CREATE_TABLE_MAP` (`x`
DOUBLE)  !!!post_table!!!    ***post_stmt***   TBLPROPERTIES ('SAS OS
Name'='X64_10PRO','SAS
Version'='9.04.01M4P11092016')

ERROR: Error attempting to CREATE a DBMS table. ERROR: Execute error: Error
while compiling
       statement: FAILED: ParseException line 1:213 cannot recognize input
near '<EOF>'
       '<EOF>' '<EOF>'.
NOTE: The DATA step has been abnormally terminated.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set MYHDP.CREATE_TABLE_MAP may be incomplete.  When this
step was stopped
         there were 0 observations and 1 variables.
ERROR: ROLLBACK issued due to errors for data set
MYHDP.CREATE_TABLE_MAP.DATA.
NOTE: DATA statement used (Total process time):
      real time           0.24 seconds
      cpu time            0.04 seconds
```

Output 3. Invalid CREATE TABLE Statement Produced By Specifying Invalid SQL Code in SAS Create Table Options.

The error looks menacing but basically means that SQL statement isn't valid. SAS creates the SQL statement and submits it to the database for processing. The CREATE TABLE options we are discussing take programmer specified text as arguments. It is not uncommon for mistakes to find their way into these strings. So, don't worry if you have errors it is not a huge problem. Fix them and move on.

SAS CREATE TABLE OPTIONS

Now that we have covered the required background material, let's look at the individual SAS create table options.

PRE_STMT_OPTS=

The PRE_STMT_OPTS= option places a user-supplied text string before the CREATE keyword in the CREATE TABLE SQL statement.

When I began writing this paper, I could find no example of why anyone would want to use PRE_STMT_OPTS=. Everything I tried caused SQL syntax errors. After a long search, I learned that this option was created for the Teradata database.

Teradata has the concept of a query band. A query band enables the Teradata DBA to tag a query so that she can easily monitor it or place it in different priority queues. A query band can also be used for billing purposes. The following example runs a SAS PROC SQL with DBIDIRECTEXEC specified. Because we used PRE_STMT_OPTS= to specify a query band, the generated CTAS statement runs in the Demigod Teradata queue:

```
libname mytera teradata server=vat user=myuser password=mypassword;

options sastrace=',,,d' sastraceloc=saslog nostsuffix;
options sql_ip_trace=note msglevel=i dbidirectexec;

proc sql;
   connect using mytera;
```

```
   create table mytera.sgf_test1 (PRE_STMT_OPTS=
           "set query_band='importance=demigod;' for transaction;")
      as select * from mytera.sgf_data;
quit;
```

Output 4 shows the SET QUERY_BAND statement was included in the PSIP generated CTAS statement and that the entire statement was executed by Teradata. There is no data read back into SAS then reinserted into Teradata.

```
TERADATA_12: Executed: on connection 1
 set query_band='importance=demigod;' for transaction; CREATE MULTISET
TABLE "sgf_test1" as (
select TXT_1."x" from "sgf_data" TXT_1 ) WITH DATA

... snip

SQL_IP_TRACE: The CREATE statement was passed to the DBMS.
```

Output 4. CTAS Statement Generated by SAS Implicit Pass-through Includes the Query Band Specification.

The QUERY_BAND= data set option provides another means of setting a Teradata query band. Unfortunately, using it prevents PSIP from working. In contrast, the QUERY_BAND= LIBNAME statement option works flawlessly – the CTAS is passed to Teradata for execution.

Let's contrast the previous example to what happens when PSIP is turned-off via the NODBIDIRECTEXEC system option.

```
libname mytera teradata server=vat user=myuser password=mypassword;

options sastrace=',,,d' sastraceloc=saslog nostsuffix;
options sql_ip_trace=note msglevel=i nodbidirectexec;
proc sql;
   connect using mytera;
   create table mytera.sgf_test2 (PRE_STMT_OPTS=
           "set query_band='importance=demigod;' for transaction;")
      as select * from mytera.sgf_data;
quit;
```

Output 5 shows the SET QUERY_BAND statement was included in the SAS-generated CREATE TABLE statement. Unfortunately, the SELECT statement brings data back into SAS then are inserted back into Teradata. We know this because the SQL_IP_TRACE note states that none of the SQL was directly passed to the database.

```
SQL_IP_TRACE: None of the SQL was directly passed to the DBMS.

... snip

TERADATA_25: Executed: on connection 4
 set query_band='importance=demigod;' for transaction; CREATE MULTISET
TABLE "sgf_test2" ("x" FLOAT);COMMIT WORK

... snip

TERADATA_27: Executed: on connection 3
SELECT "x" FROM "sgf_data"

TERADATA: trget - rows to fetch: 1

TERADATA_28: Prepared: on connection 4
USING ("x" FLOAT)INSERT INTO "sgf_test2" ("x") VALUES (:"x")
```

Output 5. CREATE TABLE Statement Generated by SAS Includes the Query Band Specification but Does Not Include the Sub-select.

The lesson here is that we want to pay special attention to how SAS and the DBMS are working together. We must ensure that SAS is passing as much of the work as possible to the DBMS. Carefully observing the SASTRACE and SQL_IP_TRACE messages helps us prevent poorly performing SQL from being submitted by SAS.

PRE_TABLE_OPTS=

The PRE_TABLE_OPTS= data set option will insert a text string between the CREATE and TABLE SQL statement keywords. The most common use for this option is create temporary and volatile tables.

For this example, we will create an Oracle global temporary table using PROC SQL:

```
libname myora path=oradb user=myuser password=mypasswd;

options sql_ip_trace=note msglevel=i dbidirectexec;
proc sql;
   connect using myora;
   create table myora.sgf_test1 (pre_table_opts='GLOBAL TEMPORARY')
     as select * from myora.sgf_data;
quit;
```

Output 6 shows the GLOBAL TEMPORARY CTAS statement generated by SAS and passed to Oracle.

```
ORACLE_5: Executed: on connection 1
CREATE GLOBAL TEMPORARY  TABLE sgf_test1 as select TXT_1."X" from SGF_DATA

...snip

SQL_IP_TRACE: The CREATE statement was passed to the DBMS.ORACLE_18:
```

Output 6. CREATE GLOBAL TEMPORARY TABLE via CTAS Generated by SAS Implicit Pass-Through.

Once again, we see that using the DBIDIRECTEXEC system option enables us to pass a SAS-generated CTAS statement completely to Oracle. If we run this PROC SQL code with NODBIDIRECTEXEC, then SAS will generate a CREATE TABLE statement, SELECT * statement, and a separate INSERT

statement. This will result in poor performance. The CREATE TABLE statement will include the value specified by the PRE_TABLE_OPTS= option. So, the table will be created properly.

Just for fun let's run the same code, but change the default column type to INTEGER using the DBTYPE= data set option:

```
libname myora path=oradb user=myuser password=mypasswd;

options sql_ip_trace=note msglevel=i dbidirectexec;
proc sql;
   connect using myora;
   create table myora.sgf_test2 (pre_table_opts='GLOBAL TEMPORARY'
                                 dbtype=(x='INTEGER'))
     as select * from myora.sgf_data;
quit;
```

Output 7 shows that SAS was unable to pass the CTAS to Oracle because there was an additional data set option specified. Most data set options, except for those that are covered in this paper, prevent SQL queries from being passed to an external database.

```
SAS_SQL:  Cannot handle dataset options.
SAS_SQL:  Unable to convert the query to a DBMS specific SQL statement due
to an error.
SQL_IP_TRACE: None of the SQL was directly passed to the DBMS.

...snip

SQL_IP_TRACE: None of the SQL was directly passed to the DBMS.

ORACLE_12: Prepared: on connection 1
SELECT * FROM SGF_TEST2

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

ORACLE_13: Executed: on connection 3
CREATE  GLOBAL TEMPORARY TABLE SGF_TEST2(X INTEGER)
```

Output 7. CTAS Fails Because There Is an Extra Data Set Option Specified.

Unfortunately, the CTAS was not pushed-down to Oracle. This means that data is moving from Oracle to SAS and then back into Oracle. If the table contains a large amount of data, this could pose a performance problem. This is compounded by the fact that the job appears to run properly.

POST_TABLE_OPTS=

The POST_TABLE_OPTS= data set option inserts a text string immediately after the table definition portion of the CREATE TABLE statement. The most common use for this option is to define storage parameters on the CREATE TABLE statement. When DBIDIRECTEXEC is specified, the text is inserted immediately after the table definition and before the SQL AS clause.

In this example, we see how to create a Hive table that stores its data in an ORC file format and creates the files in a specific HDFS location. Since we specify DBIDIRECTEXEC, the CTAS construct is executed entirely on the Hadoop cluster:

```
libname myhdp hadoop server=myhorton user=sgfuser schema=sgfhivedb;

options sql_ip_trace=note msglevel=i dbidirectexec;
proc sql;
   connect using myhdp;
```

```
   create table myhdp.sgf_test
       (post_table_opts="STORED AS ORCFILE location '/tmp/sgf_test_table'")
      as select * from myhdp.sgf_data;
quit;

/* display the directory created above */
proc hadoop user="sgfuser" ;
   HDFS LS='/tmp/sgf_test_table';
run;
```

Output 8 shows the CTAS statement that creates a table with data stored in an ORC file stored in the `/tmp/sgf_test_table` directory along with the PROC HADOOP output displaying information about the custom location. The SQL_IP_TRACE note verifies that the CTAS statement runs entirely in Hadoop.

```
HADOOP_73: Executed: on connection 1
CREATE TABLE `sgfhivedb`.`sgf_test` STORED AS ORCFILE location
'/tmp/sgf_test_table'  as
select TXT_1.`x` from `sgfhivedb`.`SGF_DATA` TXT_1

...snip

SQL_IP_TRACE: The CREATE statement was passed to the DBMS.

...snip

61   /* display the directory created above */
62   proc hadoop user="sgfuser" ;
63      HDFS LS='/tmp/sgf_test_table';
-rwxrwxrwx sgfuser  hdfs                   220 2017-02-10 20:26:15
/tmp/sgf_test_table/000000_0
```

Output 8. CTAS Statement Generated by SAS PSIP.

Once again, let's see what happens when the NODBIDIRECTEXEC option is specified.

```
libname myhdp hadoop server=myhorton user=sgfuser schema=sgfhivedb;

options sql_ip_trace=note msglevel=i nodbidirectexec;
proc sql;
   connect using myhdp;
   create table myhdp.sgf_test
       (post_table_opts="STORED AS ORCFILE location '/tmp/sgf_test_table'")
      as select * from myhdp.sgf_data;
quit;
```

Output 9 shows that there is no CTAS statement passed to Hadoop. The CREATE TABLE statement creates an ORCFILE file format table and it is stored in our special location. The SQL_IP_TRACE note tells us that the statement is not passed to the DBMS. The performance issues mentioned apply here, too.

```
SQL_IP_TRACE: None of the SQL was directly passed to the DBMS.

...snip

HADOOP_84: Executed: on connection 4
CREATE TABLE `sgfhivedb`.`SGF_TEST` (`x` DOUBLE) STORED AS ORCFILE location
'/tmp/sgf_test_table'  TBLPROPERTIES ('SAS OS Name'='X64_10PRO','SAS
Version'='9.04.01M4P11092016')
```

Output 9. The CTAS Statement Was Not Passed to Hadoop

The idea that you can specify a file type for an underlying database table is unique to Hive and a handful of other SQL processing engines on Hadoop. This gives Hive a very unique capability – you can define a table on a file in such a way that you can drop the Hive table and the associated file is not deleted. This is called an external table.

The PRE_TABLE_OPTS= and POST_TABLE_OPTS= cannot be used to create a Hive external table. Fortunately, there are options specifically for this purpose: DBCREATE_TABLE_EXTERNAL= and DBCREATE_TABLE_LOCATION=. Unfortunately, using these data set options prevents PSIP from working. Performance might not meet expectations.

The following SAS code creates a Hive external table, but does not pass the CTAS statement to Hive:

```
libname myhdp hadoop server=myhorton schema=sgfhivedb
                     user=sgfuser password=mypasswd;

proc sql;
   connect using myhdp;

   create table myhdp.sgf_test2(DBCREATE_TABLE_EXTERNAL=yes
                                DBCREATE_TABLE_LOCATION='/tmp/sgf_test2')
       as select * from myhdp.sgf_data;
quit;
```

Remember, for external Hive tables, dropping the table does not delete the underlying file. The following code will delete the underlying file that was created in the previous example:

```
proc hadoop username="sgfuser" password="mypasswd" verbose;
   hdfs delete='/tmp/sgf_test2';
run;
```

POST_STMT_OPTS=

A Google search for POST_STMT_OPTS= yields 8 hits. This is not going to be good. Like PRE_STMT_OPTS=, there has to be a reason this option was created. It obviously applies to very few of the SAS/ACCESS engines. This raises an uncomfortable dilemma for me. I need an example for this paper, but I am having a difficult time finding one. Neither Oracle nor Hive have a clause that requires this option. DB2 and Greenplum have statements which can go after the CTAS sub-select, but including them in the POST_STMT_OPTS= option causes a DBMS SQL error.

In fact, I cannot find a single database which can take advantage of POST_STMT_OPTS= when running with DBIDIRECTEXEC. Before I give up I am going to try SAS/ACCESS Interface to PostgreSQL.

Wow! It works! Perhaps POST_STMT_OPTS= was developed specifically for PostgreSQL. Back to the paper.

The POST_STMT_OPTS= data set option appends text to the end of a PSIP generated CTAS statement when using PROC SQL with the DBIDIRECTEXEC option. It helps to picture this:

```
create table db.sgf_test
      as select from db.sgf_data ***POST_STMT_OPTS_TEST_GOES_HERE***;
```

The PostgreSQL CTAS statement allows us to place a clause at the end that tells PostgreSQL to create the table but don't include data:

```
create table pgres.sgf_test
    as select from pgres.sgf_data WITH NO DATA;
```

Here is the SAS code that will construct this CTAS statement and pass it to PostreSQL:

```
libname pgres postgres server=localhost database=sgfdb
                       user=sgfuser pw=mypassword;

options sql_ip_trace=note msglevel=i dbidirectexec;
proc sql;
   connect using pgres;
   create table pgres.sgf_test1 (post_stmt_opts='with no data')
      as select * from pgres.sgf_data;
quit;
```

Output 10 shows that the WITH clause was passed to PostgreSQL via the POST_STMT_OPTS= data set option. All of the processing happens in PostgreSQL.

```
POSTGRES_11: Executed: on connection 1
CREATE  TABLE sgf_test1(x) as ( select TXT_1."x" from SGF_DATA TXT_1 )
with no data

POSTGRES: 0 row(s) affected by INSERT/UPDATE/DELETE or other statement.

POSTGRES_12: Prepared: on connection 0
SELECT * FROM SGF_TEST2

SQL_IP_TRACE: The CREATE statement was passed to the DBMS.
```

Output 10. WITH NO DATA Is Successfully Appended to the CTAS Statement Generated by SAS PSIP.

Change DBIDIRECTEXEC to NODBIDIRECTEXEC and you have a problem:

```
libname pgres postgres server=localhost database=sgfdb
                       user=sgfuser pw=mypassword;

options sql_ip_trace=note msglevel=i nodbidirectexec;
proc sql;
   connect using pgres;
   create table pgres.sgf_test2 (post_stmt_opts='with no data')
      as select * from pgres.sgf_data;
quit;
```

Normally, if NODBIDIRECTEXEC is specified, PSIP cannot run, but the code executes successfully and creates the table in SAS and then reinserts it into the database. This gives the illusion of working. Our POST_STMT_OPTS= with NODBIDIRECTEXEC example does not match this pattern. It does not work, period.

Output 11 shows the SQL generated by the POST_STMT_OPTS= data set option when NODBIDIRECTEXEC is set. We see that the SAS-generated CREATE TABLE statement fails due to a syntax error. The table is not created and no data is moved.

```
SQL_IP_TRACE: None of the SQL was directly passed to the DBMS.

...snip

POSTGRES_11: Executed: on connection 3
CREATE TABLE SGF_TEST2 (x DOUBLE PRECISION) with no data

POSTGRES: COMMIT performed on connection 3.
ERROR: Error attempting to CREATE a DBMS table. ERROR: CLI execute error:
ERROR: syntax error
       at or near "no"; Error while executing the query.
POSTGRES: ROLLBACK performed on connection 3.
POSTGRES: COMMIT performed on connection 3.
WARNING: File deletion failed for PGRES.SGF_TEST2.DATA.
```

Output 11. POST_STMT_OPTS= and DBIDIRECTEXEC Have Some Side Effects.

POST_TABLE_OPTS= and POST_STMT_OPTS= appear to be very similar. As seen in Output 11, POST_STMT_OPTS= might cause some side effects. If you make heavy use of PSIP, it is a good idea to stick with POST_TABLE_OPTS=.

THE ILLUSION OF SUCCESS

If there is one take away from this paper, it is that using these CREATE TABLE options with DBIDIRECTEXEC is very powerful. However, it is easy to have PSIP fail and back down to SAS-generated SQL code. As we have seen many times, this results in data being read back into SAS and then inserted into the database. This is a performance killer. Given how we tend to create code – develop with a small amount of data then put it in production – this can sneak up on us. The key is to use your system options carefully, especially SQL_IP_TRACE, SASTRACE, MSGLEVEL, and DBIDIRECTEXEC, and review your log file to ensure processing is occurring where you intend.

Let's look at a one more example so that we can see the impact of using POST_STMT_OPTS= and DBIDIRECTEXEC.

First up a complicated Oracle example where we create a partitioned table using the POST_TABLE_OPTS= option:

```
options sql_ip_trace=note msglevel=i dbidirectexec;
proc sql;
   create table myora.sales1 (post_table_opts="PARTITION BY RANGE (date_id)
(PARTITION SALES_2011 VALUES LESS THAN (TO_DATE('01-JAN-2011','DD-MON-YYYY')),
 PARTITION SALES_2012 VALUES LESS THAN (TO_DATE('01-JAN-2012','DD-MON-YYYY')),
 PARTITION SALES_2013 VALUES LESS THAN (TO_DATE('01-JAN-2013','DD-MON-YYYY')),
 PARTITION SALES_2014 VALUES LESS THAN (TO_DATE('01-JAN-2014','DD-MON-YYYY')),
 PARTITION SALES_2015 VALUES LESS THAN (TO_DATE('01-JAN-2015','DD-MON-YYYY')),
 PARTITION SALES_2016 VALUES LESS THAN (TO_DATE('01-JAN-2016','DD-MON-YYYY')),
 PARTITION SALES_max  VALUES LESS THAN (MAXVALUE))")

         as select * from myora.sales;
quit;
```

Output 12 shows that POST_STMT_OPTS= works when used with DBIDIRECTEXEC. Oracle handles all the processing. All is well. Ignore the note about the quoted string (unless you actually have an unquoted string) because partitioning code tends to be much longer than 262 characters.

```
214  options sql_ip_trace=note msglevel=i dbidirectexec;
215  proc sql;
216     create table myora.sales1 (post_table_opts="PARTITION BY RANGE
(date_id)

snip...
NOTE: The quoted string currently being processed has become more than 262
characters long.
      You might have unbalanced quotation marks.
220   PARTITION SALES_2014 VALUES LESS THAN (TO_DATE('01-JAN-2014','DD-MON-
YYYY')),

...snip
ORACLE_36: Executed: on connection 1
CREATE TABLE sales1 PARTITION BY RANGE (date_id)(PARTITION SALES_2011 VALUES
LESS THAN
(TO_DATE('01-JAN-2011','DD-MON-YYYY')), PARTITION SALES_2012 VALUES LESS THAN
(TO_DATE('01-JAN-2012','DD-MON-YYYY')), PARTITION SALES_2013 VALUES LESS THAN
(TO_DATE('01-JAN-2013','DD-MON-YYYY')), PARTITION SALES_2014 VALUES LESS THAN
(TO_DATE('01-JAN-2014','DD-MON-YYYY')), PARTITION SALES_2015 VALUES LESS THAN
(TO_DATE('01-JAN-2015','DD-MON-YYYY')), PARTITION SALES_2016 VALUES LESS THAN
(TO_DATE('01-JAN-2016','DD-MON-YYYY')), PARTITION SALES_max  VALUES LESS THAN
(MAXVALUE))  as
select TXT_1."DATE_ID", TXT_1."PROD_ID", TXT_1."CUST_ID", TXT_1."CHANNEL_ID",
TXT_1."PROMO_ID", TXT_1."QUANTITY_SOLD", TXT_1."AMOUNT_SOLD" from SALES TXT_1

...snip
SQL_IP_TRACE: The CREATE statement was passed to the DBMS.
```

Output 12. POST_TABLE_OPTS= and DBIDIRECTEXEC Works Perfectly.

Let's change POST_TABLE_OPTS= to POST_STMT_OPTS= and see what happens:

```
options sql_ip_trace=note msglevel=i dbidirectexec;
proc sql;
   create table myora.sales2 (post_stmt_opts="PARTITION BY RANGE (date_id)
(PARTITION SALES_2011 VALUES LESS THAN (TO_DATE('01-JAN-2011','DD-MON-YYYY')),
 PARTITION SALES_2012 VALUES LESS THAN (TO_DATE('01-JAN-2012','DD-MON-YYYY')),
 PARTITION SALES_2013 VALUES LESS THAN (TO_DATE('01-JAN-2013','DD-MON-YYYY')),
 PARTITION SALES_2014 VALUES LESS THAN (TO_DATE('01-JAN-2014','DD-MON-YYYY')),
 PARTITION SALES_2015 VALUES LESS THAN (TO_DATE('01-JAN-2015','DD-MON-YYYY')),
 PARTITION SALES_2016 VALUES LESS THAN (TO_DATE('01-JAN-2016','DD-MON-YYYY')),
 PARTITION SALES_max  VALUES LESS THAN (MAXVALUE))")

         as select * from myora.sales;
quit;
```

We wanted SAS to create a new table and put data into it. That is exactly what happened. This is a success, right? No, this is the illusion of success. Once again, the table was created using the partitioning that we specified, and data was read into SAS and inserted back into Oracle.

Output 13 shows the problem, but you have to hunt for it. It is not easy to see. The ORACLE_42 section shows that SAS placed the partitioning code at the end of the CTAS statement. Oracle threw an error (ORA-00906) because the statement is syntactically incorrect.

Next we see the SQL_IP_TRACE note stating that none of the SQL was directly passed to the DBMS.

Now the fun starts. SAS PSIP processing backs off and SQL-generation takes over. The table is created in a single step. This makes the contents of the POST_STMT_OPTS= option syntactically correct. Next SAS reads the data from Oracle then inserts it into the new table.

```
ORACLE_42: Executed: on connection 1
CREATE TABLE sales2 as select TXT_1."DATE_ID", TXT_1."PROD_ID",
TXT_1."CUST_ID",
TXT_1."CHANNEL_ID", TXT_1."PROMO_ID", TXT_1."QUANTITY_SOLD",
TXT_1."AMOUNT_SOLD" from SALES
TXT_1 PARTITION BY RANGE (date_id)(PARTITION SALES_2011 VALUES LESS THAN
(TO_DATE('01-JAN-2011','DD-MON-YYYY')), PARTITION SALES_2012 VALUES LESS
THAN
…snip
(TO_DATE('01-JAN-2016','DD-MON-YYYY')), PARTITION SALES_max  VALUES LESS
THAN (MAXVALUE))

ORACLE:  *-*-*-*-*-*-* COMMIT *-*-*-*-*-*-*
ERROR: ORACLE execute error: ORA-00906: missing left parenthesis.
SQL_IP_TRACE: None of the SQL was directly passed to the DBMS.

...snip

ORACLE_45: Executed: on connection 2
CREATE TABLE SALES2(DATE_ID DATE,PROD_ID NUMBER ,CUST_ID NUMBER ,CHANNEL_ID
VARCHAR2
(128),PROMO_ID NUMBER ,QUANTITY_SOLD NUMBER ,AMOUNT_SOLD NUMBER ) PARTITION
BY RANGE
(date_id)(PARTITION SALES_2011 VALUES LESS THAN (TO_DATE('01-JAN-2011','DD-
MON-YYYY')),
PARTITION SALES_2012 VALUES LESS THAN (TO_DATE('01-JAN-2012','DD-MON-
YYYY')), PARTITION
SALES_2013 VALUES LESS THAN (TO_DATE('01-JAN-2013','DD-MON-YYYY')),
PARTITION SALES_2014
VALUES LESS THAN (TO_DATE('01-JAN-2014','DD-MON-YYYY')), PARTITION
SALES_2015 VALUES LESS THAN
(TO_DATE('01-JAN-2015','DD-MON-YYYY')), PARTITION SALES_2016 VALUES LESS
THAN
(TO_DATE('01-JAN-2016','DD-MON-YYYY')), PARTITION SALES_max  VALUES LESS
THAN (MAXVALUE))

...snip

ORACLE_47: Prepared: on connection 2
INSERT  INTO SALES2
(DATE_ID,PROD_ID,CUST_ID,CHANNEL_ID,PROMO_ID,QUANTITY_SOLD,AMOUNT_SOLD)
VALUES
(TO_DATE(:DATE_ID,'DDMONYYYY:HH24:MI:SS','NLS_DATE_LANGUAGE=American'),:PRO
D_ID,:CUST_ID,:CHANN
EL_ID,:PROMO_ID,:QUANTITY_SOLD,:AMOUNT_SOLD)
```

Output 13. POST_STMT_OPTS= and DBIDIRECTEXEC Have Some Side Effects.

This gives us something to think about. What would happen if we did not have SASTRACE= and SQL_IP_TRACE= turned on? The code would have appeared to work, perfectly. Output 14 shows this.

```
2    options dbidirectexec;
3    proc sql;
4       create table myora.sales3 (post_stmt_opts="PARTITION BY RANGE (date_id)
5    (PARTITION SALES_2011 VALUES LESS THAN (TO_DATE('01-JAN-2011','DD-MON-YYYY')),
6     PARTITION SALES_2012 VALUES LESS THAN (TO_DATE('01-JAN-2012','DD-MON-YYYY')),
7     PARTITION SALES_2013 VALUES LESS THAN (TO_DATE('01-JAN-2013','DD-MON-YYYY')),
NOTE: The quoted string currently being processed has become more than 262
characters long.
      You might have unbalanced quotation marks.
8     PARTITION SALES_2014 VALUES LESS THAN (TO_DATE('01-JAN-2014','DD-MON-YYYY')),
9     PARTITION SALES_2015 VALUES LESS THAN (TO_DATE('01-JAN-2015','DD-MON-YYYY')),
10    PARTITION SALES_2016 VALUES LESS THAN (TO_DATE('01-JAN-2016','DD-MON-YYYY')),
11    PARTITION SALES_max  VALUES LESS THAN (MAXVALUE))")
12
13            as select * from myora.sales;
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: Table MYORA.SALES3 created, with 24 rows and 7 columns.

14   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           3.31 seconds
      cpu time            0.12 seconds
```

Output 14. POST_STMT_OPTS= and DBIDIRECTEXEC Have Some Side Effects.

It is very likely that we would focus on the note about 262 characters and be totally oblivious as to the catastrophe taking place. When dealing with SAS PSIP, take the time to ensure that processing is happening in the DBMS.

CONCLUSION

We have seen that the SAS CREATE TABLE options (PRE_STMT_OPTS=, PRE_TABLE_OPTS=, POST_TABLE_OPTS=, and POST_STMT_OPTS=) are very powerful. They make SAS an extremely versatile data management tool. Although they are simple text substitution options, we must be careful with them because used improperly they can create the illusion of working while secretly killing performance.

The key is to understand the role that DBIDIRECTEXEC plays in the process. Recall that DBIDIRECTEXEC is a SAS system option that turns on PROC SQL implicit pass-through optimization for CREATE TABLE AS SELECT and DELETE statements. The goal is to have the DBMS do as much work as possible so that data is not read by SAS then reinserted into the DBMS.

If we want to create a database table and load it with data from the same database, then we must treat POST_TABLE_OPTS= and POST_STMT_OPTS= as being totally different. POST_TABLE_OPTS= is the option of choice and will save us from unforeseen trouble.

If we want to create a database table and load it from a SAS data set, then we can treat POST_TABLE_OPTS= and POST_STMT_OPTS= (and its alias DBCREATE_TABLE_OPTS=) as being the same. But, should we? I have developed a strong bias against using POST_STMT_OPTS=. It is too easy to change the source data from a SAS data set to a database table (on the same DBMS) and once you do, you have trouble.

Output 14 shows us how hidden these issues can be. I always include SQL_IP_TRACE=NOTE in my SAS code. The cost of not doing this is just too high.

I began this paper stating that I would use Hadoop and Oracle for all the examples. I quickly determined that these options do not apply to all the supported databases. Very few DBMSs require us to use PRE_STMT_OPTS= and POST_STMT_OPTS=.

REFERENCES

SAS/ACCESS® 9.4 for Relational Databases: Reference, Ninth Edition. Available at http://support.sas.com/documentation/cdl/en/acreldb/69580/PDF/default/acreldb.pdf.

Oracle Database SQL Language Reference. Available at https://docs.oracle.com/cd/E11882_01/server.112/e41084/toc.htm.

Hive Language Manual. Available at https://cwiki.apache.org/confluence/display/Hive/LanguageManual.

PostgreSQL 9.1 Documentation. Available at https://www.postgresql.org/docs/9.1/static/index.html

ACKNOWLEDGMENTS

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

Meredyth Bass, SAS Institute Inc.

Pat Buckley, SAS Institute Inc.

Sue Her, SAS Institute Inc.

Chris Lysholm, SAS Institute Inc.

Salman Maher, SAS Institute Inc.

Greg Otto, Teradata.

CONTACT INFORMATION

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

Jeff Bailey
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Jeff.Bailey@sas.com
www.linkedin.com/in/jeffreydbailey
https://www.github.com/Jeff-Bailey