Paper SA-11-2016

Accessing Teradata through SAS, common pitfalls, solutions and tips

Kiran Venna, Experis Business Analytics Practice

Abstract

There are some common pitfalls while accessing Teradata from SAS® and Vice Versa. SAS Options and SAS Macro's efficiently handle these pitfalls. Owing to its unique architecture, Teradata primary index has to be designed properly for both space and efficiency of accessing data in Teradata. Data Set option dbcreate_table_opts handles creation of primary index, when Teradata tables are created through SAS. Inefficient data types are created, when Teradata tables are created using SAS. Data Set option dbtype helps in creating efficient data types. SAS Macro can help to automate dbcreate_table_opts and dbtype Data Set options, when SAS is used to create Teradata tables. Case Specificity of Teradata and also right truncation of string data can cause major concern, when Explicit SQL Pass-Through is used. By issuing appropriate mode in connect statement these concerns can be resolved. While creating a SAS Data Set in Explicit SQL Pass-Through with row_number function can make query fail in Teradata 15 with the same query running fine in Teradata 14. Bulk compression on large Data Sets in Teradata can be done in Explicit SQL Pass-Through.

Keywords

SAS, SAS/ACCESS®, Teradata, dbcreate_table_opts, dbtype, mode=option, ANSI vs Teradata mode, case specificity, right truncation of string data, block compression

Introduction

Teradata is very efficient in handling large amounts of data, owing to its parallel architecture. SAS is excellent in Extract Transform and Load (ETL) capabilities and in its analytical power. SAS/ACCESS provides a way to interact with Teradata either through SQL Pass-Through facility or by using libname statement. The interaction of SAS and Teradata to handle large amounts of data is often necessary for doing ETL, analytics and reporting. This interaction usually involves creating tables in Teradata through SAS and vice versa. There are notable differences in architecture and data types of SAS and Teradata, which if not considered carefully, can cause common pitfalls.

This paper covers following topics.

1. Issue of Primary index and dbcreate_table_opts= Data Set option.
2. Data type issues and dbtype= Data Set option.
3. Best practice for creating appropriate primary index and data types.
4. Automating dbcreate_table_opts and dbtype options by using SAS Macros.
5. Issue with case specificity in Explicit SQL Pass-Through and mode= option.
6. Issue with right truncation of string data in Explicit SQL Pass-Through and mode =option.
7. Issues with creating SAS Data Set with row_number in Teradata 14 vs Teradata 15.
8. Bulk compression Teradata tables in Explicit SQL Pass-Through.

Issue of Primary index and dbcreate_table_opts = Data Set option

In Teradata, table rows are distributed on Access Module Processor (AMP). Row distribution is dependent on uniqueness of defined primary index column. More unique the primary index column is better the data distribution and vice versa. Improper distribution of table rows in AMP's will results in skewed data. Data skew causes space wastage and also weakens the parallel processing capabilities of Teradata. To create primary index in Teradata a column is explicitly defined as shown.

```
CREATE TABLE teraschema.employee
(Job_type VARCHAR(9)
,employee_number INTEGER
,SAL integer
)primary index(employee number);
```

When primary index is not explicitly defined, usually first column is selected as primary index. In the above example if the primary index was not explicitly defined, job_type, which is the first column, will be selected as primary index, which may lead to data skewing.

Teradata tables can be created by DATA Step, SQL-pass through or PROC APPEND. Logic and syntax for DATA Step is discussed below and the same can be applied for other methods. In the following DATA Step, a Teradata table is created.

```
data tdtable.employee_scoring;
      set work.employee_scoring(keep= Job_type employee_number Sal Sal_rating);
run;
```

This will create Teradata table with job_type as primary index. If job_type column has few distinct values a skewed table is created. As mentioned earlier, the impact of the skewed table results in wastage of space and loss of parallel capabilities of Teradata. This is common mistake done by novice programmers.

Data set option dbcreate_table_opts can define primary index explicitly. dbcreate_table_opts = Data Set option needs a key word primary index followed by column name in parenthesis.

```
data tdtable.employee_scoring (dbcreate_table_opts= 'primary
                                                    index(employee_number)');
      set work.employee_scoring(keep= Job_type employee_number Sal,
                                      Sal_rating);
run;
```

For defining multicolumn primary index by dbcreate_table_opts, each column name should be separated by comma as shown.

```
dbcreate_table_opts= 'primary index(employee_number, Sal_rating)'
```

From Teradata 13.0, concept of no primary index was introduced, which allows to distribute rows randomly and equally across all AMP's. Code for creating Teradata table in SAS with no primary index as shown.

```
dbcreate_table_opts= 'no primary index'
```

Data type issues and dbtype Data set option

When creating a Teradata table in SAS, another important aspect to keep in mind is data type. Data type issues are not just related to Teradata but can occur with any relational databases. SAS has only two data types, num and char; which are mapped to float and char respectively in Teradata. In Teradata,

various data types are available for both numeric and character data, which provides flexibility in terms of space utilization. For string variables, char and varchar data types and for numeric variables different data types like byteint, smallint, decimal, bigint etc are available in Teradata. Assigning appropriate data type is very important in context of space, especially with large datasets.

In the below example, Teradata table is created from SAS with data types of char and float by default.

```
data tdtable.cust_table;
      set work.cust_table;
run;
```

SAS Data Set option dbtype gives the flexibility of creating appropriate data types for Teradata tables. With help of dbtype in below example, numeric variable with smaller length is casted to byteint and char variable with varying length is casted to varchar (500). Byteint data type takes one byte of space as opposed to eight bytes of space taken by float. Saving seven bytes of space can have huge impact on large datasets. Unnecessary padding space is also avoided by casting to varchar instead of default char.

```
data tdtable.cust_table  (dbtype=(cust_value='byteint'
                                  cust_address='varchar(500)'));
      set work.cust_table;
run;
```

Automating dbcreate_table_opts and dbtype options by using SAS Macros

It is evident from earlier discussion that SAS Data Set options dbcreate_table_opts and dbtype can create very efficient tables in Teradata. Automating dbcreate_table_opts and dbtype data set options with the help of SAS macro will avoid manual typing and can also be made easily available to novice programmer through a stored process. This macro is named as %dboptions and macro parameters of the same are discussed below.

Macro parameters for %dboptions

1. lib = libname of SAS dataset
2. dsn =name of SAS dataset
3. dbname= libname of Teradata data set.
4. dbtable = name of sas dataset
5. index = to define primary index of Teradata tables. Default Value is No Primary index.

%dboptions macro is limited to converting char to varchar data type but can be easily applied to num variables. Main purpose of macro %dboptions is to pick name of data set options i.e. dbcreate_table_opts and dbtype along with their values. Values for this Data Set options are created dynamically in macro for dbtype from dictionary.columns and passed through a macro parameter for dbcreate_table_opts. This macro can be explained in 3 steps.

1. First step in this macro is to pick character variables longer than length 5(usually from length 5 char value becomes varying) from SAS Data Set and to be included as values in dbtype = variable varchar(length).
2. The second step is to check, whether index macro variable has default value of no primary index, if it has that default value then use dbcreate_table_opts = default macro variable value and also use dbtype along with its values created in step 1.

3. Third step, if user input value for macro parameter is different from no primary index then primary index value is taken from macro parameter and used to create a primary index with everything same as in step 2.

Code for %dboptions macro

```
%macro dboptions(lib=, dsn=, dbname=,dbtable=, index=%str('No Primary
index'));
  proc sql noprint;
  select
     strip(name)||"="||"'"||"varchar"||"("||strip(put(length,5.))||")"||"'"
    into :a separated by ' '
   from       dictionary.columns
   where      libname = upcase("&lib.")
   and        memname=upcase("&dsn.")
   and        type='char'
   and        length gt 5;
 quit;
    %if &index = 'No Primary index' %then %do;
   data &dbname..&dbtable.( dbcreate_table_opts = 'No Primary index'
                                dbtype =(&a));
         set &lib..&dsn.;
       run;
     %end;
    %else %do;
      data _null_;
        call symput('b',"'"||"primary index"||"("||"&index"||")"||"'");
      run;
      data &dbname..&dbtable.( dbcreate_table_opts = &b dbtype =(&a));
        set &lib..&dsn.;
      run;
   %end;
%mend dboptions;
```

1

2

3

Below are the three macro executions, first one creates Teradata table with no primary index, while second one creates a Teradata table with single primary index and the third one with multiple primary indexes. Dbtype values are selected dynamically in macro from SAS tables with help of dictionary.columns.

Three different options to execute %dboptions macro

```
%dboptions(lib = work, dsn=base_cust, dbname=tera_tbl, dbtable=dem_tbl);
%dboptions(lib =work, dsn=new_cust, dbname=tera_db, dbtable=cust_tbl,
index=cust_cd);
%dboptions(lib = work, dsn=prod_tbl,   dbname=tera_db, dbtable=
cust_tbl,index=%str(cust_cd ,dept_cd));
```

Best practice for creating appropriate primary index and data types

Data Set options dbtype and dbcreate_table_opts, serve the purpose of creating appropriate primary index and data types respectively when creating Teradata tables from SAS. However, the best practice for creating efficient Teradata tables is to first create empty Teradata table through Explicit SQL Pass-Through and then use PROC APPEND to insert the data. The purpose of creating empty table in Explicit SQL Pass-Through is to have a Teradata table with appropriate column attributes and primary index.

Below is data definition language for creating empty Teradata table with appropriate attributes in Explicit SQL Pass-Through.

```
proc sql;
 connect to teradata (server=myserver user=myuserid pw=mypass);
  execute(create table teraschema.employee
          (cust_id decimal(10, 0),
           cust_fname varchar(40),
           cust_lname varchar(50))
         primary index(cust_id)) by teradata;
     execute(commit) by teradata;
  disconnect from teradata;
quit;
```

The code for PROC APPEND is shown below. Base table in PROC APPEND is Teradata table which was created by running its Data Definition Language (DDL) in Explicit SQL Pass-Through and Data table is SAS table from where data needs to be moved. This best practice gives full control of creating efficient tables.

```
proc append
      base= tera_tbl.employee
      data=work.employee;
quit;
```

ANSI vs Teradata mode in Explicit SQL Pass-Through through issues

Mode concept in Explicit SQL Pass-Through is very important topic to understand, especially when connecting to Teradata. Default mode in Explicit SQL Pass-Through is ANSI, where as it is Teradata mode in Teradata tools. Due to difference in default modes in SQL Pass-Through and Teradata tools, which can often leads to varying results for same query. There are 2 Major issues than can arise due to these differences in default modes. First one is case specificity of ANSI mode as opposed to non-case specificity of Teradata mode. Second one is right truncation of string data when user tries to assign a longer string to a shorter string destination in Teradata mode as opposed to the error "Right truncation of string data" in ANSI mode.

Issue of case specificity in Explicit SQL Pass-Through and mode= option

In the example below, where clause is used to search for ‘smith’ in a Teradata tool (SQL Assistant). This will bring all the records with ‘smith’ irrespective of case like ‘Smith’, ‘SMITH’ or with mixed case. Default mode in various Teradata tools is usually Teradata which causes this non case specificity.

```
Select  cust_id,
      fname,
      lname,
from prod_stagedb.customer_table
Where lname = 'smith'
```

In Explicit SQL Pass-Through as shown in the code below, where clause is used to search for 'smith'. This will search for records with exact case matching. Case specificity of the Explicit SQL Pass-Through is because of its default mode, which is ANSI. This may result in different number of records for same query in Teradata tools versus Explicit SQL Pass-Through.

```
proc sql;
connect to teradata (server=myserver user=myuserid pw=mypass);
  execute(Insert into prod_targetdb.customer_table
              Select  cust_id,  fname, lname,
        from      prod_stagedb.customer_table
      Where    lname = 'smith') by teradata;
disconnect from teradata;
quit;
```

This difference in functioning of same query in Teradata tools as opposed to Explicit SQL Pass-Through is due to default mode in those particular systems. To achieve the same results as in Teradata tools, mode = teradata has to be added in connect statement as shown below.

```
connect to teradata(server=myserver user=myuserid pw=mypass mode = teradata);
```

Even though mode = teradata exactly simulates the queries like in Teradata tools, one major issue with Teradata mode is right truncation of string data.

Right truncation of string data in Teradata mode

Teradata mode in connect statement in Explicit SQL Pass-Through is useful to emulate character specificity achieved in Teradata tools but this can cause issue of right truncation of string data. When string data is inserted in target table with shorter length than source string column, silent right truncation of data will happen in Teradata mode. In a similar scenario, truncation of the data does not happen in ANSI mode and will cause insert to fail. The main problem involved in silent right truncation of string data is, it gives an impression that everything is all right with the query, unless someone notices the data. If someone is concerned with truncation of string data, they should not use mode = Teradata option.

Impact of ROW_NUMBER in Teradata 15 versus Teradata 14 on SAS tables

Row_number function in Teradata is mainly used to create unique sequential numbers, starting from number one, within a partition or a whole Data Set. Order of the sequential number depends on ordering column. Syntax for row_number with a partition with order by is given below. Row_number with no partition and with order clause operates on whole Data Set.

```
proc sql ;
connect to teradata (server=server user=user pw=pw  );
 create table work.emp as
      (select *
       from connection to teradata
        (select a.*,
       row_number()over(partition by deptno order by hiredate) as rn  from
prod_targetdb.customer_table a
      ));
 disconnect from teradata;
quit;
```

Row_number was defaulted to integer in Teradata version 14 and was easily modified to num when data was moved from Teradata to SAS. But in Teradata 15 Row_number was defaulted to Bigint in Teradata, which can cause your jobs to fail with below error message.

```
ERROR: At least one of the columns in this DBMS table has a datatype that
is not supported by this engine.
```

To avoid above problem row_number has to be casted to char(20) as shown below.

```
cast(row_number()over(partition by deptno order by empno)as char(20)) as rn
```

Compressing Teradata tables in Explicit SQL Pass-Through

This topic is a tip and is for people, who are interested to block compress their Teradata tables in a SAS Job. Block compression often reduces space by nearly 80% of original size of Teradata Table. Teradata tables which are not very frequently accessed are excellent candidates for block compression. Block compression in Teradata tables can be use by giving QUERY_BAND = 'BLOCKCOMPRESSION=YES;' in a connect statement while insert in an empty table for first time. Code for compressing Teradata tables in Explicit SQL Pass-Through is shown below.

```
proc sql;
 connect to teradata (server=myserver user=myuserid pw=mypw
                          query_band = 'blockcompression=yes;');
  execute(Insert into prod_targetdb.customer_table
           Select  * from prod_stagedb.customer_table) by teradata;
disconnect from teradata;
quit;
```

Summary

SAS options give the required control, while using SAS/ACCESS Interface to create Teradata Tables. When creating Teradata table by using SAS DATA Step, PROC APPEND or Implicit SQL Pass-Through, Data Set options dbcreate_table_opts and dbtype are very important. This options play an important role in creating very efficient Teradata tables both with respect to space and also with regard to Teradata parallel capabilities. Dbcreate_table_opts should always be used, when a Teradata table is created from SAS. Appropriate index or no primary index should be mentioned in the dbcreate_table_opts. dbtype should also be used to create appropriate data type, while creating tables in Teradata from SAS. dbcreate_table_opts and dbtype options can be embedded in SAS macros to create Teradata tables from SAS. While using Explicit SQL Pass-Through mode = option is very important factor. ANSI Mode gives no case specificity, whereas Teradata mode gives case specificity and can also truncate the string data. Appropriate casting for row_number() column might be necessary when upgrading from Teradata 14 to Teradata 15. Bulk compression can be done in Explicit SQL Pass-Through by using appropriate options in connect statement.

ACNOWLEDGEMENTS

I would like to thank Vikas Chhabra, Sarika Elisetti, Ravi Ganesana and Laura Oliver for their valuable inputs.

References

1. SAS Institute Inc. 2014. SAS/ACCESS® 9.4 for Relational Databases: Reference, Sixth Edition. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/acreldb/67589/PDF/default/acreldb.pdf
2. http://www.info.teradata.com/HTMLPubs/DB_TTU_15_00/index.html

Contact Information

Kiran Venna
kiranvenna@gmail.com