SESUG Paper 154-2018

Improving query performance by saying "NO" to Heterogeneous Joins

Suryakiran Pothuraju, Unify Solutions Inc., Catonsville, MD
Kiran Venna, SMACT Works Inc., Dublin, OH

ABSTRACT

Heterogeneous joins between SAS® dataset and DBMS table is a common requirement, most SAS® programmers often come across. SAS/ACCESS® support for DBMS temporary tables has the ability to retain DBMS temporary tables across multiple SAS® steps. If the joins process in-database, the overall performance will enhance immensely. In this paper, we will discuss how to query small subset of data from extremely large Database tables efficiently by using key variables from a SAS® Table.

KEYWORDS: Data Management, Performance, Teradata Volatile Tables, MULTISTMT, GLOBAL Connection, SAS/ACCESS®, Heterogeneous and Homogeneous Joins, SQL pass-through, In-Database.

INTRODUCTION

When joining two or more tables which reside in the same schema then that type of join is considered as homogeneous joins. This type of joins have better performance in query processing because they have same hardware and software. It is difficult for most organization to maintain homogeneous environment because of its disadvantages (Risk, Modularity, Failure, Cost, Protection of valuable data, expansion, etc.) and go for different software and hardware which result in heterogeneous environment. Performing a heterogeneous joins will make the system to locate the data sources, perform necessary query translation and also data transmission across network which are main considerations for performance.

SAS® programmers routinely join SAS® datasets with different database tables efficiently by retrieving only the necessary data into SAS® using different approaches like WHERE clause for sub-setting data, OBS to limit the data to read, KEEP and DROP for only the required variables, and copying the database table once and running several queries against it in SAS®.

When the join is heterogeneous, it will result in excess data movement from database to SAS® server than required. SAS® dataset can be pushed into database and process in-database to retrieve only the required records back to SAS® instead of performing a heterogeneous joins which will result in more data movement from database to SAS® server over network.

In this paper we will discuss how to avoid heterogeneous joins between large Teradata table/view and a small SAS® dataset using SAS/ACCESS® support for database temporary tables. The main four topics that we cover in this paper are:

1. GLOBAL CONNECTION TO DATABASE
2. CREATING DATABASE TEMPORARY TABLE
3. LOADING DATA INTO TEMPORARY TABLE
4. IN-DATABASE JOINS

GLOBAL CONNECTION TO DATABASE

SAS/ACCESS® can establish a single physical connection to DBMS that persists across multiple SAS® Procedures and Data steps using CONNECTION = GLOBAL as LIBNAME option. Assign a LIBNAME statement with CONNECTION=GLOBAL and DBMSTEMP=YES allows to create and access the temporary tables in multiple steps.

```
libname td_db teradata user = &td_user
                       pwd = &td_pw
                    server = &td_server
                connection = global
                  dbmstemp = yes;
```

Any pass-through queries after the LIBNAME statement share the same connection and can access the temporary table created within the global connection.

CREATING DATABASE TEMPORARY TABLE

Creating a database temporary table with proper data definitions using SAS® pass-through query with some basic knowledge on native database SQL can control the utilization of temporary tables that best suits the need. An example of temporary table creation in Teradata is shown below.
Now, let's see how an empty temporary table with one column ("id") is created.

```
proc sql;
connect to teradata (server = &td_server
                       user = &td_user
                        pwd = &td_pw
                 Connection = global);
execute(
          create multiset  volatile table td_temp ( id char(8) )
                no primary index
                on commit preserve rows
        ) by teradata;
execute(commit work) by teradata;
quit;
```

In the above query ON COMMIT PRESERVE ROWS is required, since the default is ON COMMIT DELETE ROWS and also mention NO PRIMARY INDEX since by default first column will be selected as primary index and may lead to data skewing.

Data can also be loaded into permanent table instead of temporary table, but this paper is concentrated in utilizing temporary table because most work environments doesn't allow users to create permanent tables in production. Global or Volatile Temporary tables are two types of

temporary tables that can be created in Teradata. Volatile tables are more efficient because data definitions are not stored in Data Dictionary and also access rights checking are not done.

LOADING DATA INTO TEMPORARY TABLE

FASTLOAD option cannot be used to load data into Teradata volatile table, but we can request multi-statement inserts using MULTISTMT= YES option. By default inserts are sent one at a time, when there is large volume of data to insert then performance can be increased significantly by using multi-statement inserts. MULTISTMT=YES will attempt to insert as many inserts as possible that will fit in 64k buffer.

"sas_data" Dataset that contains distinct "id" key values will be loaded into temporary Teradata table (created in above step) using proc append.

```
proc append base=td_db.td_temp(multistmt=yes)
            data=work.sas_data;
run;
```

IN-DATABASE JOIN

In-database joins can be performed between the temporary table and large Teradata permanent table (Homogenous Joins) to create final SAS® Dataset ("id_info").

```
proc sql ;
connect to teradata (server = &td_server
                       user = &td_user
                        pwd = &td_pw
                 Connection = global);
create table id_info as
select *
     from connection to teradata
          (select tmp.id,
                  prm.full_name,
                  prm.address
                  prm.phone
                  prm.email
             from td_temp as tmp
                inner join <schema>.<table/view> as prm
                  on (tmp.id=prm.id)
          );
disconnect from teradata;
quit;
```

Here, native Teradata SQL functions can also be used to manipulate data that will be returned to SAS® Server.

CONCLUSION

There are several ways to enhance performance of query between small SAS® dataset and extremely large database table, on way is by pushing the SAS® dataset into database and process in-database. This will result in less data movement between database server and SAS® server.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the author's for more information:

Suryakiran Pothuraju
Unify Solution Inc.
Catonsville, MD - 21228
suryakiran.pothuraju@analyticscurator.com
www.analyticscurator.com

Kiran Venna
SMACT Works Inc.
Dublin, OH – 43017
kiranvenna@gmail.com

REFERENCES

- SAS/ACCESS® 9.4 for Relational Databases: Reference, Ninth Edition http://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.4&docsetId=acreldb&docsetTarget=titlepage.htm&locale=en
- SAS® 9.4 Product Documentation http://support.sas.com/documentation/94/index.html
- Teradata Documentation https://www.info.teradata.com/HTMLPubs/DB_TTU_16_00/index.html