PharmaSUG 2017 - Paper BB17

Writing Efficient Queries in SAS® Using PROC SQL with Teradata

Mina Chen, Roche (China) Holding Ltd.

ABSTRACT

The emergence of big data, as well as advancements in data science approaches and technology, is providing pharmaceutical companies with an opportunity to gain novel insights that can enhance and accelerate drug research and development. The pharmaceuticals industry has seen an explosion in the amount of available data beyond that collected from traditional, tightly controlled clinical trial environments. Investing in data enrichment, integration, and management will allow the industry to combine real-time and historical information for deeper insight as well as competitive advantage. On the other hand, pharmaceutical companies are faced with the unique big data challenges collecting more data than ever before. There has never been a greater need for efficient data analytical approach.

Together, SAS and Teradata provide a combined analytics solution that helps big data analysis and reporting. This paper will introduce, based on real study experience, how to connect to Teradata in SAS and how to conduct analysis with SQL in a more efficient way.

INTRODUCTION

Value-based drug development refers to the pharmaceutical companies striving to create value for patients with new therapies. There is an increasing need to maximize usage of real world data (as well as randomized clinical trials data) along with predictive analytics so that development program planning and execution can be achieved with maximum efficiency. From a data analyst's perspective, accelerating value-based drug development means exploring and adapting new and suitable approaches to delivery of big data analysis. A key approach to analysis big data is to use SAS in conjunction with Teradata to support faster decision-making then accelerate drug development. SAS/ACCESS® offers two methods of passing SQL to Teradata for data processing: implicit and explicit SQL pass-through. Implicit SQL pass-through can be identified by the use of a SAS LIBNAME statement pointing at a relational database. As the name suggests, SAS will attempt to convert such code to SQL that the target database can process. On the other hand, explicit SQL pass-through is a coding syntax that allows the user to write and submit database-specific SQL that SAS will pass untouched to that database. These queries execute entirely on the DBMS and return results back to SAS.
This paper will describe those scenarios, as well as how to use SQL Pass-Through more efficiently from a data analyst's perspective in pharmaceuticals.

ABOUT TERADATA

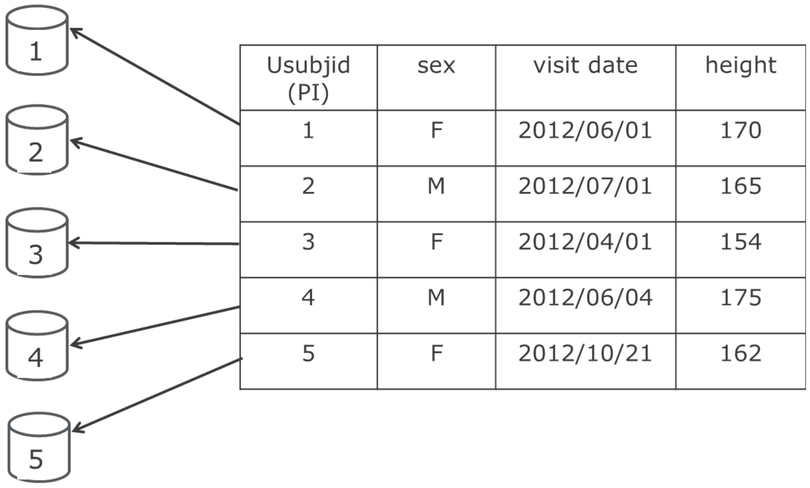
Teradata is very efficient in handling large amounts of data, owing to its parallel architecture. It uses Massively Parallel Processing (MPP) to provide linear scalability of the system by distributing the data across a number of processing units (AMPs). Each record in a table is placed on an AMP. The more evenly the data is distributed across the AMPs for all tables, the better the system performs, because each AMP does an equal amount of work to satisfy a query. When the data is unevenly distributed, the AMPs with the most records work harder, while the AMPs with the fewest records are underutilized. 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. So there're some relational database concepts that you need to know before working with Teradata.

§ **Primary Index**

The primary index (PI) distributes the records in a table across the AMPs, by hashing the columns that make up the PI to determine which records go to which AMP. If no PI is specified when a table is created, the first column of the table will be used as the PI. When creating a table, care needs to be taken to choose a column or set of columns that evenly distribute the data across the AMPs. A PI that distributes data unevenly will at the very least impact the performance of the table, and depending on the size of the table, has the potential to negatively impact the entire system.

Even distribution of the PI isn't the only criteria to use when choosing a PI. Consideration should also be given to how the data will be queried. If the data can be evenly distributed using different sets of columns, then the determination of which columns to use should be based on how the data will be queried and what other tables it will be joined to. If two tables that are frequently joined have the same PI, then joining them doesn't require the records to be redistributed to other AMPs to satisfy a query.

Figure 1 is an simple example of choosing primary index:



Usubjid (PI)	sex	visit date	height
1	F	2012/06/01	170
2	M	2012/07/01	165
3	F	2012/04/01	154
4	M	2012/06/04	175
5	F	2012/10/21	162

Figure 1. Choosing Primary Index

The PI (usubjid) is unique - rows are distributed more evenly across the AMPs.

§ **Skew Factor**

A perfectly distributed table (same amount of data on each AMP) will have a skew factor of 0%. While a table which has all of its data assigned to a single AMP will be 100% skewed. The higher the skew factor is, the more unevenly data in a table is distributed. As a general rule, tables with a skew factor higher than 50% should be evaluated to determine if a different primary index would distribute the data better, and thereby improve performance.

QUERYING TERADATA USING SAS®

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 data analysis 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. So the following part will introduce the two methods of connecting to Teradata using SAS separately.

Implicit SQL Pass-Through

The first method is known as implicit SQL pass-through. As the name suggests, the programmer using this coding method implicitly expects SAS to convert that code to SQL and pass it to the target relational database for execution. Implicit SQL pass-through can be recognized by virtue of the fact it contains a SAS library reference pointing at the target relational database. For example, the following library reference points at a Teradata-resident database `tdwork`.

```
LIBNAME tdwork TERADATA SERVER="servername" user="youruserid"
password="yourpassword";
```

In the LIBNAME statement, the USER= option provides the Teradata user name, the PASSWORD= option provides the Teradata password associated with the Teradata user name, and the SERVER= option provides the Teradata server name as defined in the HOSTS file.
Behind the scenes, the SAS/ACCESS engine writes SQL code that is passed implicitly to Teradata, causing as much work to be done in the database as possible.
Incorporating this library name, the following code excerpts thus represent two examples of implicit SQL pass-through:

Example 1: `PROC PRINT DATA=tdwork.teradata_table (OBS=5);`
Example 2: `PROC SQL; CREATE TABLE mydataset AS SELECT * FROM tdwork.teradata_table;`
In example 1, the SAS/ACCESS engine writes SQL code on the user's behalf from this PROC PRINT step and DATA

step that is passed implicitly to Teradata. While for the second example, there're sometimes misunderstanding that it may be an explicit pass-through, which will be introduced in the second part.
Once the queries are submitted, SAS via the SAS/ACCESS engine determines the SQL query that is implicitly passed to Teradata on the user's behalf and convert the SAS code to Teradata-specific SQL.
Here's an example:

```
proc sql
  select pt, start_date, type,
    sum(call_dur) as call_dur_sum
  from tdwork.mydata
  where start_date between "01JAN16"d and "30JUN17“d
  group by pt, start_date, type;
quit;
```

The above code will be converted to Teradata-specific SQL as below:

```
SQL_IP_TRACE: passed down query:
select "mydata"."PT", "mydata"."START_DATE",
       "mydata"."TYPE”,
SUM("mydata"."CALL_DUR") as "call_dur_sum" from
"mydata" where "mydata"."START_DATE“
   between DATE'2016-01-01' and DATE'2017-06-30‘
group by
"mydata"."PT", "mydata"."START_DATE", "mydata"."TYPE"
```

By default, there is no indication regarding the success or failure of the SAS/ACCESS engine to generate SQL code that is passed to Teradata. To determine the success or failure of implicit pass-through for a query, you must examine the SQL that the SAS/ACCESS engine submits to the Teradata by using the SASTRACE= SAS system option in the following way:

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
```

Explicit SQL Pass-Through

Explicit pass-through is Teradata-specific SQL that passed by SAS directly to the Teradata system through the SAS/ACCESS engine. Teradata will parse and optimize the query, there is no SAS intervention such as error validation or others.
Explicit pass-through allows users to leverage full power of Teradata's massively parallel processing capability and the wide range of mathematical, statistical, and data reorganization functions available, via usage of native Teradata SQL, combine SAS programming features and Teradata-specific features in your query and save the results of a query as a SAS data set or a PROC SQL pass-through view.
Although explicit SQL pass-through is embedded in PROC SQL code, it is not synonymous with PROC SQL. As the name suggests, Explicit SQL pass-through is based on the premise that SAS will not alter or translate the code, but rather will submit what the programmer has coded verbatim to the relational database for execution.
There are two forms of syntax typically associated with explicit SQL pass-through:
One is the SELECT statements, which produce some outputs to SAS (SQL procedure pass-through queries). For example:

```
proc sql;
   connect to teradata (user="youruserid" password="yourpassword"
server="servername" mode=teradata);
   create table temp as
   select * from connection to teradata (
      TERADATA-QUERY EXPRESSION
      );
  disconnect from teradata;
quit;
```

The CONNECT statement establishes a connection to Teradata and retrieves data directly from Teradata. The DISCONNECT statement terminates the connection to Teradata. Note that the SELECT statement that embraces a native Teradata-SQL code enclosed in parentheses to be passed to Teradata.

Another component is the EXECUTE statements, which perform all other non-query SQL statements that do not produce output. By default, the Teradata connection is established in ANSI-SQL mode for Teradata, hence after each or a series of execute request, an explicit commit request is required to be submitted.
For example:

```
proc sql;
 connect to teradata (user="youruserid" password="yourpassword"
mode=teradata  server="servername" connection=global);
 execute(
 create volatile table temp as (
 TERADATA-QUERY EXPRESSION
 )
 with data primary index (id)
 on commit preserve rows
 ) by teradata;
quit;
```

`TERADATA-QUERY EXPRESSION` is the SQL statement passed to Teradata. In this example a volatile table TEMP was created by using the EXECUTE statement.

CHOOSEING THE RIGHT METHOD FOR THE RIGHT ANALYSIS

It was known that the enhancements to implicit SQL pass-through have greatly expanded the variety and quantity of SAS code that can be automatically translated to SQL and passed to a database. At the same time, explicit SQL pass-through has great value in reducing the risk of an unwanted download of large amounts of data.
Based on my experience in real study analysis, the difference is that if we use LIBNAME statement, it would be used to create a library to store and access the SAS files or datasets in the specified particular folder. We can use this one with the DATA and PROC step to run program and to access datasets but should not be used to access the databases. If we are trying to join more than one table means multiple tables and looking for only aggregate result, in this case explicit SQL pass-through facility is a much better option, which we use as a part of PROC SQL to connect to Teradata and passes the statements directly to Teradata for execution.
So the difference with using LIBNAME engine (implicit pass-through) and explicit SQL pass-through would be from few hours to few minutes based on different analysis situation.

CONCLUSION

As the increasing needs of analyzing real world data as well as RCT data, new data analysis approaches like the integration of SAS and Teradata are used more and more widely. There’re two main querying methods from SAS to Teradata, how to choose the right method for the right analysis is a key to enable faster decision-making and data interpretation. This paper provided a brief introduction of querying Teradata from SAS and the scenarios of different querying methods (implicit SQL pass-through and explicit SQL pass-through) based on current usage of analyzing real world data. As the customer demands evolving, we programmers need to explore and adapt more efficient methods and ways to meet the needs then to accelerate drug development.

REFERENCES

SAS/ACCESS® Interface to Teradata - SAS Support. SAS Institute, Inc. Available at:
https://support.sas.com/resources/papers/teradata.pdf
Explicit SQL Pass-Through: Is It Still Useful. Frank Capobianco, Teradata Corporation. Available at:
http://support.sas.com/resources/papers/proceedings11/105-2011.pdf

CONTACT INFORMATION

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

Name: Mina Chen
Enterprise: Roche (China) Holding Ltd.
Address: 4F, Building 11, 1100 Longdong Avenue, Pudong New District City, State ZIP: Shanghai, China
Work Phone: +86-021-2892 3475, 201203

E-mail: mina.chen@roche.com