6 Golden Rules for Teradata SQL Tuning

Author : Roland Wenzlofsky

Date : October 29, 2014

Teradata SQL Tuning Evergreens

1.

Ensure completeness and correctness of Teradata Statistics

This is what I always teach in my performance Teradata SQL tuning workshops without ever disappointing my clients:

The most important optimization task is to ensure statistic completeness and correctness. This is valid for statistics on all database objects, not only for tables (secondary indexes, join indexes).

Detecting missing statistics is easier than you may expect, in your SQL assistant run the command:

DIAGNOSTIC HELPSTATS ON FOR SESSION;

and EXPLAIN your SQL statement. At the end of the explain output a list of statistics will be added which the optimizer would consider helpful in creating a better execution plan. Add them one by one and re-check the execution plan.

To detect stale statistics, there are several methods available. Maybe the easiest way is to split your SQL statement into parts and test each of them by comparing the estimated row count statistics (which the explain output shows you) with the real number of records returned. This approach makes especially sense if the SQL statement which has to be optimized is not runnable in a moderate time. Here is one example:

```
SELECT t01.*
FROM
    table_1 t01
 INNER JOIN
   table_2 t02
ON
   t01.PK = t02.PK
WHERE
   t01.column_1 = 1
   AND t02.column_a = 2
;
```

Above query can be split into two parts for testing:

```
SELECT * FROM table_1 WHERE column_1 = 1;
```

and

```
SELECT * FROM table_2 WHERE column_a = 2;
```

Execute and Explain both queries and compare the number of returned rows against the number of rows the explain output shows you.

If there is a big difference, it could be related to stale statistics or similar problems related with the data distribution.

2.

The Primary Index (PI) Choice

Use primary indexes for joins whenever possible, and specify in the where clause all the columns for the primary indexes. Joining on the complete set of primary index columns is the least resource intense join possibility.

If your environment does not allow you to adjust primary indexes to suit your requirements or you would need a different primary index only for optimizing certain SQL statements, you should think about re-indexing the involved tables by using volatile tables or real temporary tables.

3.

Teradata Indexing Techniques

Using Teradata Indexing Techniques may be another option to improve your SQL statement. For example, secondary indexes could be especially helpful if you have highly selective WHERE conditions.

You could try as well join indexes or even work with partitioning.
Whenever working with indexing techniques you have to keep the overall data warehouse architecture in mind and how your solution fits into this architecture.

If the identified indexes are not used by the optimizer and are not useful in the overall PDM design, drop them again immediately.

4.

Query Rewriting

Many times, queries performance can be improved by rewriting the query in different way.

Examples like using DISTINCT instead of GROUP BY on columns with many different values come to my mind.
Union could be used to break up a large SQL statements into several smaller ones, which may be executed in parallel.

While query rewriting may be very powerful in your query optimization toolkit, it may require to understand the business logic behind (“can this left join be replaced by an inner join?”)

5.

Real Time Monitoring

Watch your query running in real-time. Observing your query while its running in Viewpoint or PMON, helps you to find out the critical steps of your query.

Most performance issues are caused either by query steps with heavy skewing in the AMP’s or by a wrong execution plan caused by stale and missing statistics.

Stale and missing statistics typically lead to wrong decisions in join preparation (copying a table to all AMP’s instead of rehashing) and join types used (product join instead of merge join).

6.

Comparison of Resource Usage

Another very important task in performance optimization is to measure the resources used before and after the optimization. Plain query run times can be misleading as they heavily depend on the current load on the Teradata Server and workload management blocking you may not even notice.
Here is one example query which only needs the DBC.DBQLOGTBL table. Set a different QUERYBAND for each version of the query you are running:

```
SET QUERY_BAND = 'Version=1;' FOR SESSION;

 SELECT
   AMPCPUTIME,
   (FIRSTRESPTIME-STARTTIME DAY(2) TO SECOND(6)) RUNTIME,
   SPOOLUSAGE/1024**3 AS SPOOL_IN_GB,
   CAST(100-((AMPCPUTIME/(HASHAMP()+1))*100/NULLIFZERO(MAXAMPCPUTIME)) AS
INTEGER) AS CPU_SKEW,
   MAXAMPCPUTIME*(HASHAMP()+1) AS CPU_IMPACT,
   AMPCPUTIME*1000/NULLIFZERO(TOTALIOCOUNT) AS LHR
FROM
   DBC.DBQLOGTBL
WHERE
     QUERYBAND = 'Version=1;'
```

Above query gives you detailed insight about how good or bad each step of your query is:

- The total CPU Usage
- The Spool Space needed
- The LHR (ratio between CPU and IO usage)
- The CPU Skew
- The Skew Impact on the CPU

Goal is to reduce total CPU usage, consumed spool space and Skew impact on the CPU.
Further, the LHR is optimally around 1.00