Query Tuning Guidelines

## Query Tuning Checklist

The following checklist can be used as a guide to avoid some of the most frequent problems that may occur when developing and running queries on Teradata. Each step is described in the next sections.

1. Format the Query
2. Use the Explain Plan
3. Eliminate Accidental Product Joins
4. Check Statistics
5. Check the Use of Indexes
6. Check the Estimated Result Set
7. Use Group By Vs DISTINCT
8. Use Date vs. CAST(Timestamp)
9. Check for Default Values
10. Add Indexed Columns to Joins and Filters
11. Check for Matched Column Definitions
12. Check to see if there are sub queries with the IN/NOT IN, EXISTS/NOT EXISTS. Replace
13. them with joins.
14. Rerun Explain
15. ANSI AND TD JOIN Format
16. UNION vs. UNION ALL

## Format the Query

Format the query (CTRL +Q) to a standard SQL format. Qualify all tables with the Database name.

Example:

Select a.column1, a.column2, b.column3, b.column4, sum (b.column5) from table1 a, table2 b where a.column1=b.column1 and a.column2=b.column2

Group by 1,2,3,4;

Can be formatted for better understanding and clarity as

Select

a.column1

,a.column2

,b.column3

,b.column4

,sum (b.column5)

From

databasename.table1 a

,databasename.table2 b

Where

a.column1=b.column1

and a.column2=b.column2

Group by 1,2,3,4;

## Use Explain

Check the EXPLAIN plan for the query. This can be done by placing the word “explain” before the query or by pressing the function key **F6** in Queryman.

Run the following at the beginning of each session in order for the explain plan to include information concerning which columns it is using for redistribution steps:

Diagnostic verbose explain on for session;

Browse through the explain and make note of the estimated step times and row counts for spool files. Use these as sanity checks of the estimated row counts that might be expected based on the query.

Make sure that the times are reasonable for the tables being queried.

Excessive estimated row counts and step times could indicate a problem with the query, the need for query tuning, a lack of statistics on important tables and columns and perhaps a need for a new approach (such as breaking the query into more than one step with statistics collected on intermediate tables).

## Eliminate Accidental Product Joins

Browse through the query to see if there are any accidental products joins or logic errors.

Check for the columns in the SELECT clause and WHERE clause to see if they all are qualified with the table names existing in the FROM clause.

If an alias has been defined for a table, carry the alias all the way through the query. Otherwise, Teradata will assume that these are two copies of the table, one with the alias and one without.

In case of ANSI join formats using the ON clause, check to see if the joins are all on the tables that are referenced are as part of the ON clause. Avoid hybrids like some of the joins are done in ON and some in WHERE. This can sometimes lead to CARTESIAN product.

Anything after the outer join is serialized. By ANSI definition, where clause criteria are applied after any ON clause. This forces the entire table to be built prior to constraining. Check to see if the correct columns are being referenced in the GROUP BY clause.

## Check Statistics

Check to see if the statistics are collected for the columns in join condition or filters.

Run: Diagnostic help stats on for session;

Run an explain on the query.

Any statistics that might be useful in determining the best execution plan are reported at the end of the explain.

If the explain plan estimates are not in line with expectations or the query is not performing well, notify the DBA Group of single column recommendations, particularly if the query is run repeatedly. The DBA Group could add these columns to their statistics collection procedure in order to have statistics collected on an ongoing basis.

## Check the Use of Indexes

Check to see if any of the columns in join conditions (the “ovals” in the data map) are indexed or are part of an index. If they are a part of an index, try to include a join condition on all columns that make up the index. In this way, the index will be used.

If there is an index, check the explain plan to verify that this index will be used. Look for statements such as “…by traversal of index #...” If use of the index is not indicated in the explain plan, the query will not use the index.

Please note that creating unnecessary indexes can result in negative Performance and does impact on the below factors.

1. Takes up a lot of additional space because creating a secondary index actually creates a sub table on all the AMP’s.
2. Makes your DML statements against that table run longer as the index subtable has to be updated.

**Points to consider for creating/defining indexes.**

1. Is the optimizer really using the index created? You can look in the explain plan.
2. Should be defined on columns frequently used in WHERE clause. However should satisfy point 1.
3. Stats should be collected.
4. Can have it on the duplicate columns. Should satisfy points 1, 2.

## Check the Estimated Result Set

Estimate the volume of data that will be retrieved from the query and make sure that the client can handle it and that it is required. Add additional filters or run queries against sampled data placed in temporary tables in order to return a useful and meaningful result.

For example, to view a few hundred rows of data from a large table, use:

Select top 50 \* from databasename.tablename;

Rather than:

Select \* from databasename.tablename;

Note that the top clause works efficiently when selecting directly from a table or from a view does not contain filters or joins. Selecting from a view with filters and/or joins builds the entire result set before taking the sample.

## Use Group By vs. Distinct

Use GROUP BY rather than DISTINCT. Teradata typically processes group by more efficiently than distinct.

For example, the query:

**Sel distinct col from db.tbl;**

will typically performance better as:

**Sel col from db.tbl group by 1;**

## Use Date vs. CAST (Timestamp)

Use the DATE column instead of casting the TIMESTAMP, where possible.

For example, the query:

**Sel \* from db.tbl where cast(transaction\_created\_ts as date) ='2006-12-04'**

Explain:

1) First, we lock db.tbl for access.

2) Next, we do an all-AMPs RETRIEVE step from

Db.tbl by way of an all-rows scan with a

condition of (

"(CAST((Db.tbl.TRANSACTION\_CREATED\_TIMESTAMP) AS

DATE))= DATE '2006-12-04'") into Spool 1 (group\_amps), which is

built locally on the AMPs. The input table will not be cached in

memory, but it is eligible for synchronized scanning. The result

spool file will not be cached in memory. The size of Spool 1 is

estimated with no confidence to be 202,945,318 rows. The

estimated time for this step is **27 minutes and 43 seconds**.

3) Finally, we send out an END TRANSACTION step to all AMPs involved

in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of

statement 1. The total estimated time is 27 minutes and 43

seconds.

Can be written as: (Note that this also leverages the index defined on transaction\_created\_date.)

**Sel \* from db.tbl where transaction\_created\_date=’2006-12-04'**

Explain:

1) First, we lock db.tbl for access.

2) Next, we do an all-AMPs RETRIEVE step from

Db.tbl by way of index # 4

"db.tbl .Field\_1045 = DATE '2006-12-04'" with no

residual conditions into Spool 1 (group\_amps), which is built

locally on the AMPs. The input table will not be cached in memory,

but it is eligible for synchronized scanning. The size of Spool 1

is estimated with high confidence to be 2,232,309 rows. The

estimated time for this step **is 2 minutes and 25 seconds**.

3) Finally, we send out an END TRANSACTION step to all AMPs involved

in processing the request.

-> The contents of Spool 1 are sent back to the user as the result of

statement 1. The total estimated time is 2 minutes and 25 seconds.

## Add Indexed Columns to Joins and Filters

If there are no indexes used by the query, double check the join logic and attempt to add filters or join condition on the indexed columns, including the primary index.

## Check for Matched Column Definitions

Check to see if the column type definitions are the same.

If they are not the same, one of the columns will be cast to a different type by Teradata and redistributed based on the new type. This could lead to unnecessary redistributions.

Integers and decimals (n, 0) hash can be considered the same type. However DECIMAL (7,x) and DECIMAL (7,y) hash differently even if they have the same number and should be considered different types.

Make a note of this and inform the Data modeler/ Data Architect (DM PDM Team).

## Using Joins instead of using clauses NOT EXISTS, EXIST and NOT IN.

Using joins rather than using clauses like NOT EXISTS, EXISTS, IN, NOT IN etc. will perform better when the Inner join query returns a result set that has more than 5 values.

## Rerun Explain

Run the EXPLAIN plan after any changes to the query and look for changes and improvements.

* 1. ***Combo usage of ANSI AND TD SQL in the same query***

Please browse through the query and see if you are using the combination of ANSI-SQL and TD SQL in the joins of the same query. If you had used, please change the query to be consistent in the entire query to use either TD SQL or ANSI SQL. Please note that the hybrid/combo usage might leads to product joins in some cases.

**Example of USAGE of ANSI-SQL and TD SQL USAGE in one query**:

**Sel** …………….

**from** table1**join**table2

**on** table1.a=table2.a, table3

**where** table1.a=table3.a

You need to change the above as below. (TD SQL)

**Sel** ………..

**from** table1

**join**table2

**on** table1.a=table2.a

**join**table3

**on** table3.a=table1.a

## UNION vs UNION ALL

Union operator can be used to combine the result set from two select result set into one. However, the final result set will have the data type of the columns specified in the *first* SELECT statement in the union.

Please note: By default, duplicate rows are eliminated from each result set and from the final result if you don’t explicitly specify the ALL option. However, it is not encouraged to use ALL option with UNION operator as the result set will contain all the duplicate rows also.