|  |  |
| --- | --- |
| Teradata Best Practices |  |

|  |
| --- |
|  |

# 1-SET versus MULTISET tables

Tables should be defined as MULTISET with NO FALLBACK unless there is a very specific reason to have a SET and/or FALLBACK protected definition.MULTISET omits a unique row check, saving overhead, while NO FALLBACKensures there are not four copies of the table stored on disk. Teradata isRAID1/mirrored internally by default so a redundant copy already exists in case of drive failure.

# 2-PRIMARY INDEX selection:

The choice of a table's PRIMARY INDEX is the single most important physical design decision. The PRIMARY INDEX is not necessarily the PRIMARY KEY. If it is unique then it should be defined as a UNIQUE PRIMARY INDEX. If the table does not have any good candidate field(s) for the PRIMARY INDEX then defining the table as NO PRIMARY INDEX is acceptable (e.g. some staging tables). Two important considerations for the PRIMARY INDEX are data distribution across the MPP architecture (i.e. the AMPs); and, join access which is the most important. Therefore, reasonable skew of data at rest is acceptable for processing efficiency gains.

# 3-Efficient data typing:

Fields should have an efficient data typing. If they always contain values then they should be defined as NOT NULL. Again, further efficiencies are realized. Examples of efficient data typing include defining code fields (e.g State Code), and other fixed-length character fields as CHARACTER and not VARCHAR. Today, we MVC compress low-cardinality fixed-length fields (prior to the introduction of VLC where variable-length fields can also be compressed). Additionally, date fields are defined as DATE and not TIMESTAMP nor INTEGER. TIMESTAMP is internally a concatenation of Year, Month, Day, Hour, Minute, and second fields, and is not nearly as efficient as DATE. Likewise, turning date values into a look-alike INTEGER saves nothing in storage, but removes date math capabilities inherent with the DATE data type. Likewise, storing numerical values in fields much larger than is required (e.g. if DECIMAL (9,2) suffices then don't decimal(18,2); also, don't exceed DECIMAL(18,x) without a specific reason to do so).

**4-Multi-Value Compression:**

Now that our tables and fields are defined we need to specify our Multi-Value Compression. MVC offers reduced disk space consumption and increased access performance. Typically, MVC can save 30% of required disk space on average. Thinner records equate to more records read perI/O. Less I/O leads to faster executing queries.

**5-Statistics:**

Database statistics are core and fundamental to Teradata. The cost-based optimizer relies on them heavily. Approximately 90% of all performance problems are due to missing or stale/bad statistics. Teradata's rule of thumb about refreshing statistics is only required if the data changes by 10% or more is solely a guideline. I would be far more risk averse and place it at 5% or less. Ultimately, I consider fixed-period collection to work well (e.g. daily, weekly). Additionally, my belief is single-column statistics tend to be less confusing to the optimizer given the overlap of multi-column statistics with single-column versions; therefore, the core is single-column. At a minimum, the following fields should contain statistics:

* the PRIMARY INDEX
* the pseudo field PARTITION
* foreign key fields
* predicates: fields used as criteria in an ON or WHERE clause
* secondary indexes (multi-column statistics if the secondary index is multi-column, and the collection is defined as INDEX)

**6-ForeignKeys:**

While statistics give the optimizer information about a table's demographics, they are intra-table. However, we rarely use solely one table in a query; rather joins to other tables are more common. Therefore, defining FOREIGN KEYs between tables allows the optimizer to understand intended relationships and offers inter-table demographics and extrapolate cardinality. Foreign Key constraints done on a SoftRI (i.e. without database referential integrity checks) basis also allow for Join Elimination to be leveraged. This means that the optimizer may omit joins from a query or view if it deems that the join with not affect the integrity of the query's outcome and the join process would otherwise be superfluous

**7-Locking modifiers:**

Table accessing queries should occur through a view. The base view to the table for accessing queries should contain an access lock: LOCKING TABLE table\_name FOR ACCESS, or the row version at a minimum. Access locks remove another level of overhead and mitigate contention with write locks.

**8-ANSI SQL:**

Queries and views should be written with ANSI SQL if there is an equivalent function to the need. ANSI SQL tends to perform equally well or better and is typically more efficient for accessing queries and views.

**9-OAFs:**

Ordered Analytical Functions can, not always, offer increased efficiency by reducing the number of database objects brought into a query. This is especially true when doing some form of chronological sequencing of the data. Prior to OAFs these types of processes required self-joins of tables to occur. Otherwise, this would require a copy of the table to be spun up into spool for that join to occur. With an OAF it reduces the number of objects/tables involved and in many cases requires only a single pass of specific tables. Less I/O is required to scan through a table once versus many times. Examples of this involve the need to look back or forward in time, across records within a key, for the prior/next chronological value of a field. OAFs can be very powerful for specific tasks.

**10-EXPLAIN plan**:

After the tables are created, the data is loaded, the statistics and FKs are collected, and the views and queries built, we need to validate it, not only from a results perspective, but from a processing perspective as well. The EXPLAIN plan defines the exact steps, and their sequence. It can provide valuable information about where a process may deviate from what was originally anticipated and help with debugging. It is a powerful tool that should be used to ensure efficiency and quality.

**11-Fastoad**

You use the Fastload process on empty tables, such as loading staging tables and in initial loads where the tables are empty.

When the Fastload process starts loading, it locks the target table, which means that processes (for example, lookups) cannot access that table. One solution to this problem is to specify dummy SQL for the look up overrides at the session level.

1:   If a session fails during a Fastload process, use SQL Assistant to run a simple SQL command (for example, count (\*)), to determine whether the table is locked by a Fastload process.

If a table is locked (for example, (for W\_SAMPLE\_DS), use the following script to release the lock:

LOGON SDCNCR1/Sample\_qa1, sqa1;

BEGIN LOADING Sample\_qa1.W\_SAMPLE\_DS

ERRORFILES

Sample\_qa1.ET\_W\_SAMPLE\_DS, Sample\_qa1.UV\_W\_SAMPLE\_DS;

END LOADING;

If you save the above text in a file called test.ctl, you would run this process by entering the following command at a command prompt:

C:\fastload\test.ctl

2:   To create a load script for a table, edit the test.ctl script above to change the login information, and replace all occurrences of W\_SAMPLE\_DS with the required target table name.

After a load process script runs successfully, you should be able to run the command 'select count (\*)' on the target table. If you are not able release the lock, you might need to drop and re-create the table to remove the lock. If you do so, you must re-create the statistics.

3:   Fastload is typically used in piped mode to load staging tables and initial loads. In the event of errors, reload the entire data.

Fload uses 2 error tables  
Error table 1: where format of data is not correct.

Error table 2: violations of UPI

**12-Mload**

The Mload process is slower than Fastload but quicker than Tpump. The Mload process can work on both empty tables as well as on tables with data. In the event of errors when running in piped mode, you cannot recover the data.

MULTILOAD will create 2 error tables, 1 work table if it fails.  
  
1-To unlock the main table in case of acquisition phase:

RELEASE MLOAD DBNAME.Tablename;

2-To release lock in application phase failure:

RELEASE MLOAD DBNAME.Tablename IN APPLY;

You should be very cautious using the RELEASE command. It could potentially leave your table half updated. Therefore, it is handy for a test environment, but please don’t get too reliant on it for production runs. They should be allowed to finish guaranteeing data integrity.

Mload also uses 2 error tables (ET and UV), 1 work table and 1 log table

1. ET TABLE - Data error

MultiLoad uses the ET table, also called the Acquisition Phase error table, to store errors found during the acquisition phase of a MultiLoad import task.

2. UV TABLE - UPI violations

MultiLoad uses the UV table, also called the [Applicationhttp://images.intellitxt.com/ast/adTypes/icon1.png](http://www.geekinterview.com/question_details/62746) Phase error table, to store data errors found during the application phase of a MultiLoad import or delete task

3. WORK TABLE - WT

Mload loads the selected records in the work table

4. LOG TABLE

A log table maintains record of all checkpoints related to the load job, it is essential/mandatory to specify a log table in Mload job. This table will be useful in case you have a job abort or restart due to any reason.

**13-Tpump**

The Tpump process is slower than Mload but faster than ODBC. The Tpump process does row commits, which enables you to recover processed operations, even if you use piping mode. In other words, if you re-start the process, Tpump starts loading data from the last committed data.

Tpump can be used in the following modes:

1-Tpump\_Insert: Use to do insert.

2-Tpump\_Update: Use to do updates (this mode requires you to define the primary key in the Informatica target table definition).

3-Tpump\_Upsert: Use to do update otherwise insert (this mode requires you to define the primary key in the Informatica target table definition).

4-Tpump\_Delete: Use to do deletes (this mode requires you to define the primary key in the Informatica target table definition).

Informatica uses the actual target table name to generate the error table and log tables to be used as part of its control file generation. If you have two instances of Tpump loading into same target table at the same time, you need to modify the session to use a different error table and log table name.

The Tpump load process in piped mode is useful for incremental loads, and where the table is not empty. In the event of errors, restart the process and it starts re-loading from the last committed data.

**14-Stage Table Design**

* Stage table should be created as like source tables
* No lookups are encouraged during stage load.
* No index, secondary constraints are encouraged in stage tables
* Recommended load is FLOAD,( truncate and load )
* No records are encouraged in rejection ( Error1 and error2 ) tables

**15-Target Table Design**

* All the lookups and calculation has to be done while moving data from stage to target.

**16-Query Usage Techniques**

* Avoid computation with null value details
* Avoid the usage of NOT IN with AND condition
* Avoid the usage of select \* query in large table without the where condition