**How to update millions of records and not take years: Part 1**

By Joseph DeArce 02/20/18

**Introduction**

This series of articles will explore how parallel processing works within the Oracle 12c database server and how you can monitor it. In addition, we will cover how logging works in Oracle 12c, what options are available to us to facilitate updates, deletions, inserting records, and **CTASs** (Create Tables as Select)onrecords within very large tables with minimal impact on production systems. There are many solutions for these types of problems where you need to update, delete, insert and **CTAS** records from a table with hundreds of millions of rows. The traditional update statement can take days or months, none of us can take that kind of processing hit. My solution is one where I used the **nologging** and **parallel** options to overcome these processing issues. The **nologging/logging** option is different in Oracle 12c but the ‘**FORCE LOGGING**’ can disable the use **NOLOGGING** option at table level and even though you use an **ALTER TABLE** command or a **CREATE TABLE** command, you will still be in **LOGGING** mode and you will not even know it.

Another solution is using the Oracle package **DBMS\_REDIFINITION** that allows an online production transactional table that has a heavy transactional load to be redefined on the fly. However, this can only be used under certain conditions. This would not work if all you just wanted to do was just remove some rows such as in a delete statement. The **DBMS\_REDIFINITION** package is a multistep process where you would call a series of procedures in a particular order. Our solution would work for this scenario were we just wanted to remove or update some rows. For this to work, you would need to create a copy of the table without the unwanted rows. Then the old and new table can be renamed.

**Hardware**

This is what our test system’s configuration:

* An Acer Laptop running Windows 10 with a Core7i quad processor,
* 16GB of memory.
* 1TB hard drive (not SSD).
* Installed is an Oracle 12c Enterprise database with an SGA of 4GB
* This database has a custom configuration.
* There are five PDB’s on the Oracle 12c R1 instance.

**Logging options in Oracle 12c**

In Oracle Database 12c the architecture of the database has changed, we now have two kinds of databases; "**Pluggable Databases** (**PDB**)" and "**Container Databases** (**CDB**)". In addition, we have a partitioned data dictionary between the **CDB** and **PDB**’s. With these kind of structural database changes there comes Logging Level change, and that's one of the things that can really impact your **nologging** options. You may not even know that it has happened! In the following pages, I will describe the logging levels and the effects that they can have. I will also show examples which will illustrate the various logging levels. These are:

* Table Level Logging/Nologging.
* Index Level Logging/Nologging.
* Tablespace Level Force Logging/Nologging.
* PDB Level Logging/Nologging.
* PDB Level Force No logging/Force logging.
* CDB Force Logging/Force No logging.

**Table Level Logging/Nologing**

Setting the logging at the table level determines whether **Direct SQL\*Loader** and **direct-path** **INSERT** operations against the table and index are logged or not logged. **LOGGING** is the default value.

In non-partitioned indexes this causes the **logging** value to be turned on for both the table and index. When the index is partitioned, it determines:

* The **default value** of all partitions is specified in the **CREATE** statement, unless you specify the **logging\_clause** in the **PARTITION** clause.
* The default value for the segments associated with the index partitions.
* The **default value** for any local index partitions or subpartitions added implicitly during any **ALTER TABLE ADD PARTITION** operations.
* The logging value of the index is independent of the base table.

If you omit the logging clause, then the logging value is that of the tablespace.

searstgi\_admin@PDT16TST> CREATE TABLE TEST\_LG (STUDENT\_ID NUMBER, STUDENT\_NAME VARCHAR2(100) ) LOGGING;

Table TEST\_LG created

Elapsed: 00:00:00.016

searstgi\_admin@PDT16TST>SELECT TABLE\_NAME, LOGGING FROM DBA\_TABLES WHERE TABLE\_NAME='TEST\_LG'

TABLE\_NAME      LOG  
-------------- ---  
TEST\_LG YES

searstgi\_admin@PDT16TST> ALTER TABLE TEST\_LG NOLOGGING;

Table TEST\_LG altered.

Elapsed: 00:00:00.040

searstgi\_admin@PDT16TST> SELECT TABLE\_NAME, LOGGING FROM DBA\_TABLES WHERE TABLE\_NAME='TEST\_LG'

TABLE\_NAME LOG  
--------------- ---  
TEST\_LG NO

**Index Level Logging/Nologging**

Use the **logging\_clause** to change the **logging** value of the index. If you also specify the **REBUILD** clause, then the new setting affects the rebuild operation too. If you specify a different value for logging in the **REBUILD** clause, then Oracle Database uses the last logging value specified.

An index segment can have logging values different from those of the base table and different from those of other index segments for the same table.

searstgi\_admin@PDT16TST> CREATE INDEX TEST\_LG\_IDX ON TEST\_LG(STUDENT\_ID) LOGGING;

Index created.

searstgi\_admin@PDT16TST> SELECT INDEX\_NAME, LOGGING FROM DBA\_INDEXES WHERE TABLE\_NAME='TEST\_LG' ;

INDEX\_NAME LOG

--------------- ---

TEST\_LG\_IDX YES

searstgi\_admin@PDT16TST> ALTER INDEX TEST\_LG\_IDX NOLOGGING;

Index altered.

searstgi\_admin@PDT16TST> SELECT INDEX\_NAME, LOGGING FROM DBA\_INDEXES WHERE TABLE\_NAME='TEST\_LG' ;

INDEX\_NAME LOG

--------------- ---

TEST\_LG\_IDX NO

searstgi\_admin@PDT16TST>

**Tablespace Level Logging/Nologging**

Is used to specify the default logging values of all the tables, partitions, indexes, materialized views, and materialized view logs in a tablespace. This clause is **not valid for a temporary or undo** tablespace and you will get an error if you try.

If you omit this clause, then the default is **LOGGING**. The exception is creating a tablespace in a **PDB**. In this case, if you omit this clause, then the tablespace uses the logging value of the **PDB**.

The **tablespace-level logging** value can be overridden by logging specifications at the table, partition, index, materialized view, and materialized view log.

**sys@DATA16PR> CREATE TABLESPACE SEARS\_DATA1 DATAFILE 'C:\app\OraDt16\oradata\data16pr\PDT16TST\Sears2x.ora'**

**2 SIZE 256M AUTOEXTEND ON NEXT 2M DEFAULT**

**3 STORAGE ( INITIAL 200M NEXT 202M MAXEXTENTS UNLIMITED PCTINCREASE 1 ) LOGGING;**

**Tablespace created.**

**sys@DATA16PR> CREATE TABLESPACE SEARS\_DATA1 DATAFILE 'C:\app\OraDt16\oradata\data16pr\PDT16TST\Sears2x.ora'**

**2 SIZE 256M AUTOEXTEND ON NEXT 2M DEFAULT**

**3 STORAGE ( INITIAL 200M NEXT 202M MAXEXTENTS UNLIMITED PCTINCREASE 1 ) LOGGING;**

**sys@DATA16PR> SELECT TABLESPACE\_NAME, LOGGING, FORCE\_LOGGING FROM DBA\_TABLESPACES WHERE TABLESPACE\_NAME='SEARS\_DATA1';**

**TABLESPACE\_NAME LOGGING FOR**

**------------------------------ --------- ---**

**SEARS\_DATA1 LOGGING NO**

**sys@DATA16PR> ALTER TABLESPACE SEARS\_DATA1 NOLOGGING;**

**Tablespace altered.**

**sys@DATA16PR> SELECT TABLESPACE\_NAME, LOGGING, FORCE\_LOGGING FROM DBA\_TABLESPACES WHERE TABLESPACE\_NAME='SEARS\_DATA1';**

**TABLESPACE\_NAME LOGGING FOR**

**------------------------------ --------- ---**

**SEARS\_DATA1 NOLOGGING NO**

**Tablespace Level Force Logging/Nologging**

You can use the logging clause to put the tablespace into **FORCE LOGGING** mode. Oracle Database will log all changes to all objects in the tablespace except changes to temporary segments, **overriding** any **NOLOGGING** setting for individual objects. The database must be open and in read write mode.

This setting does not exclude the **NOLOGGING** value. You can specify both **FORCE LOGGING** and **NOLOGGING** with that clause for objects subsequently created in the tablespace, but this default value is in effect for as long as the tablespace or the database is in **FORCE LOGGING** mode. If you subsequently take the tablespace out of **FORCE LOGGING** mode, then the **NOLOGGING** default is once again enforced.

What is **FORCE LOGGING?** How can it impact the use of the **NOLOGGING** option in an Oracle database? The **FORCE LOGGING** option can be set at database creation or later using the alter database command for the hole database or on tablespace level.

To set **FORCE LOGGING** it can be done during the database creation, by issuing a database **CREATE DATABASE** **FORCE LOGGING** with the option.

To enable **FORCE LOGGING** after the database is created, use the following command:

**ALTER DATABASE FORCE LOGGING;**

The **FORCE LOGGING** option has become a **best practice** with some **DBA**s and in certain cases it is the safest way to ensure that no data is lost when someone turns off logging by accident or on the wrong table which will allow the loss of Redo log activity.

But what about the use case where you are reloading a table from scratch? When you create a target table in **ODI** and reload the mapping data. In cases like these it does not work. and In some DSS applications or data warehousing, but not where you have a slowly changing dimension and you wish to trap those changes.

At work, I am a Senior DBA working with **Data Warehousing, APEX** and **ODI**, I turned on the **NOLOGGING** option at my company and guess what happened: **nothing**. Even though we issued an alter table command it was still in logging mode, our command was ignored. This is wrong, Oracle 12c should at least give you an error or a warning message so that you don’t waste time and know that the database is in **FORCE LOGGING** mode. I did not realize this until I got the results back from the run and the times were the same.

Then we found out that the infrastructure **DBA** group had used **FORCE LOGGING** as a best practice for all our databases. Because we are a major user of Data Guard, currently we use Data Guard 12c even on development systems so we can’t use the **NOLOGGING** option but maybe you can. Oracle suggests using **FORCE LOGGING** on any systems which use Data Guard and have a standby database or you can have problems, such as listed below.

This an excerpt from the Oracle manual ‘**Oracle Data Guard Concepts and Administration 12c R1**’ see below:

URL: <https://docs.oracle.com/database/121/SBYDB/E48552-07.pdf>

10.3.6 **NOLOGGING or Unrecoverable Operations**

When you perform a **DML** or **DDL** operation using the **NOLOGGING** or **UNRECOVERABLE** clause, the standby database is invalidated and may require substantial **DBA** administrative activities to repair. You can specify the **SQL** **ALTER DATABASE** or **ALTER TABLESPACE** statement with the **FORCE LOGGING** clause to override the **NOLOGGING** setting. However, this statement will not repair an already invalidated database.

I started to see comments about **NOLOGGING** and Data Guard as far back as 2008 in the Ask Tom column and other forums. Data Guard could not handle this option, and would lead to errors in the standby database.It would make the standby database invalid. However, at that time it didn’t affect me, so I didn’t take note of it.

Force logging is a new feature that was added to the logging choices for Oracle 12c. Before the existence of **FORCE LOGGING**, Oracle provided two options: **logging** and **nologging**. These two options have higher precedence at the schema object level than the tablespace level; therefore, it was possible to override the logging settings at the tablespace level with **nologging** setting at schema object level.

The following statement will put a tablespace in **FORCE LOGGING** mode:

**ALTER TABLESPACE <tablespace name> FORCE LOGGING;**

The **FORCE LOGGING** mode can be cancelled at the database level using the following statement:

**ALTER DATABASE NO FORCE LOGGING;**

The **FORCE LOGGING** mode can be cancelled at the tablespace level using the following statement:

**ALTER TABLESPACE <tablespace name> NO FORCE LOGGING;**

Temporary tablespaces and temporary segments have no **FORCE LOGGING** because it has no effect because these objects **do not generate any redo**. Undo tablespaces are in **FORCE LOGGING** mode by default, so they cannot be put into **FORCE LOGGING** mode. Oracle will generate an error if an attempt is made to put a temporary tablespace or undo tablespace into **FORCE LOGGING** mode.

**ALTER TABLESPACE UNDOTBS1 FORCE LOGGING**

**ERROR** at line 1:

ORA-30021: Operation not allowed on undo tablespace

The **FORCE\_LOGGING** column of **v$database**view can be queried to verify that the database is **in FORCE LOGGING** mode. Similarly, the **FORCE\_LOGGING** column of **dba\_tablespaces** view provides the same logging information for each tablespace.

**SELECT FORCE\_LOGGING FROM V$DATABASE;**

**SELECT FORCE\_LOGGING FROM DBA\_TABLESPACES;**

Force logging mode is persistent across database startups, but it is not maintained when a control file is recreated unless the **FORCE LOGGING** clause is specified in the create **controlfile** statement. Also, a tablespace in the **FORCE LOGGING** mode, when transported to another database, does not maintain this mode.

In these situations, the **FORCE LOGGING** mode would have to be re-enabled. The primary database should remain in **FORCE LOGGING** mode as long as there is at least one Oracle instance in use.  Putting a database in **FORCE LOGGING** mode will have some performance impact.

**Conclusion**

The conclusion we can draw is that new logging options in Oracle 12c can be used to reinforce the use of **Nologging** option or disable it. The **Nologging** option has a legitimate use case in data warehousing where you are creating a table and logging is not required since the table can be recreated at any time from a source system. One concern is when **Nologging** is used on a Dataguard on standby **Nologging** can make it invalid. **FORCE LOGGING** is good in most cases and is the best way to guaranty that no data is lost through error or ignorance.