CA Test Data Manager 4.8.1 FDM Scaled Masking Benchmarking and Best Practice

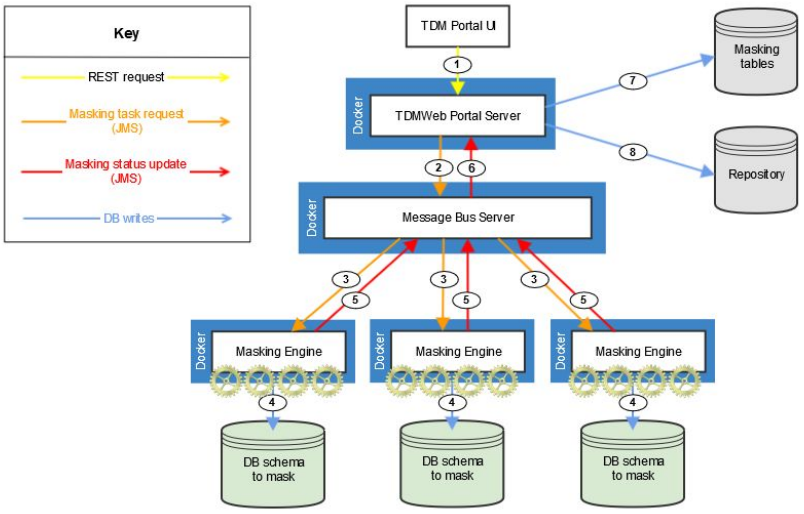
Version: 1.6
Date: Wednesday, 8th January 2020
Authors: Keith Puzey, Abderrahmane Zahrir, Walter Guerrero, Salvator Pilo

Contents

1 Masking and architectural considerations

Scalable masking from the TDM Portal was made available with TDM 4.8. This new feature allows masking jobs to be automatically split across multiple masking engines which are deployed within Docker containers. By utilising additional masking engines a masking job is split into parallel threads which reduces the overall masking time.

The architecture diagram below shows the overall TDM architecture when using scaled masking.



When calculating overall masking time you should consider the following:

1. Overall masking time is dependent on the number of individual cell updates to be masked not just the number of rows. As an example a table with 100 rows and 10 columns will update 1000 cells.
2. Different types of masking function and the use of large seedlists can affect the overall masking time
3. Performance and configuration of the target database can affect the time to complete masking
4. An average 4 Million cell updates per minute can be expected on a moderate size environment. More details can be found in the results section of this document.

The scaled masking feature will dynamically split a masking job to utilise separate FDM engine instances for scale out masking of large tables and it can also be used to split very large tables to utilise multiple threads within a single FDM engine instance. The following explains the default rules when using scaled masking:

1 Each Data Source within a masking job will be allocated a separate FDM masking engine instance

2 Each FDM Docker container will run 4 FDM masking engine instances by default

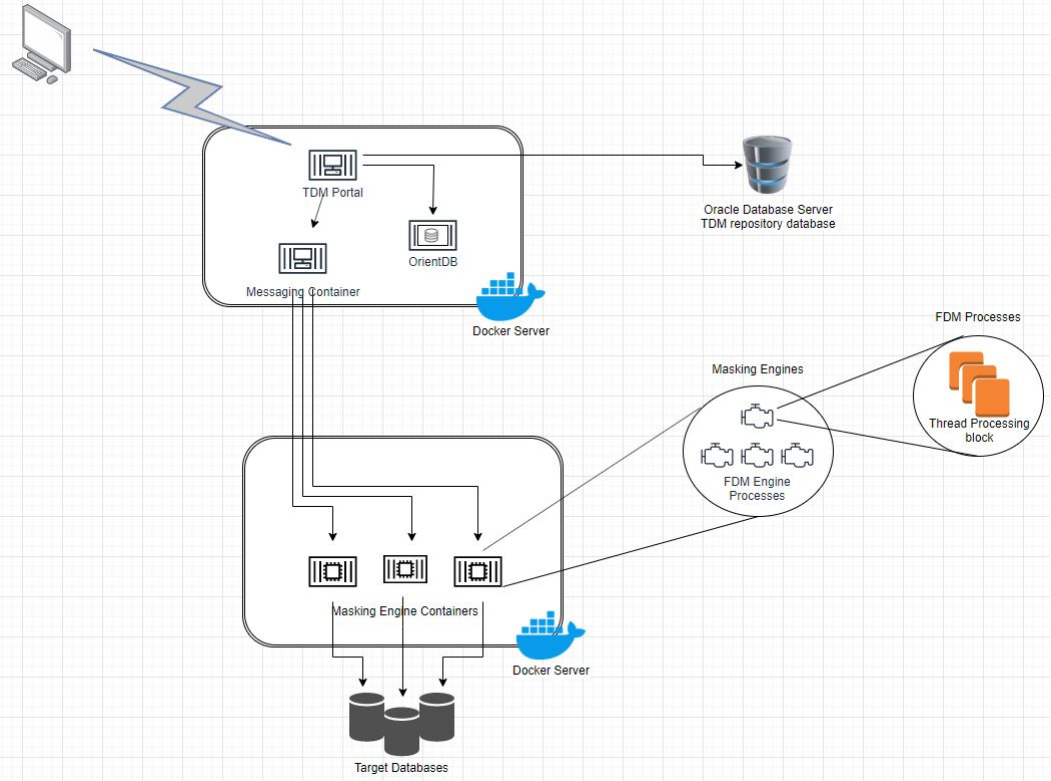
3 The following setting in the application.properties file can be used to control when an individual table is also assigned a separate FDM engine instance. The

default setting is 1 Million so any individual table with over 1 million rows will use a separate FDM instance.

i. tdmweb.TDMMaskingService.tableTaskRowThreshold

TDM Masking Deployment and sizing

The TDM Docker masking containers can generate a high load when running multiple masking jobs so it is recommended to separate the docker masking engines from other TDM components. In the sample architecture below the TDM Portal / Messaging and OrientDB containers are deployed separately from the masking containers. The masking containers are configured to communicate with the messaging container on the primary TDM server.



TDM Masking Components explained

TDM Portal	TDM Manager and UI Server
TDM Repository	Oracle Database storing TDM data
OrientDB	TDM Internal Database (Models and Find and Reserve data)
Messaging Container	Communication bus between TDM Portal and Masking engines
Masking Engines Container	Docker container hosting FDM Engines
FDM Engine Instances	Each Masking engine container can run four separate masking engine instances
FDM Processes	Each FDM Masking engine can spawn additional FDM Processes when using the large table masking functionality

Sizing Considerations

A standard masking container masking a single table will utilise approximately 1.5Gb of memory and 500 millicores (½ CPU). TDM will utilise separate FDM masking engines instances for each table that has over 1 million rows.

The following table shows how memory and CPU consumption increases as more FDM masking engines are used:

Number of Tables	Memory Usage (GB)	CPU Usage (millicores)
1	1.3	550
2	1.75	2070
3	2.9	3100
4	3.5	4100

When using the large table splitting functionality to mask very large tables the masking job will run on a single masking engine but will spawn multiple FDM processes and this will increase the CPU Usage, generally each FDM process will consume 1000 millicores (1 CPU)

2 Masking very large tables

From TDM 4.8.135 and above, when masking a very large table with a primary key or a unique index, you can improve performance by using the following options.

LARGETABLESPLITENABLED

Enables large tables processing.

Set this parameter to Y to enable, and to N to disable.

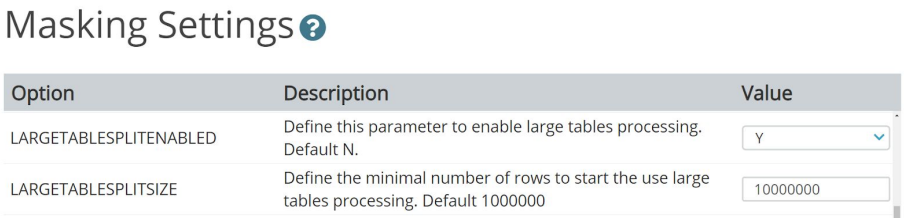
Default: N

LARGETABLESPLITSIZE

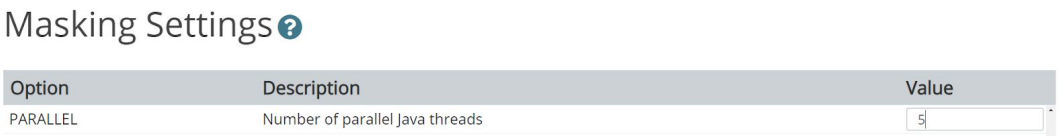
Defines the minimal number of rows for Fast Data Masker to start using large table processing.

Default: 1000000

With this setting, Fast Data Masker processes large tables by generating several blocks, with each block containing *LARGETABLESPLITSIZE* rows to be processed.



The existing option **PARALLEL** defines the number of threads that can run concurrently to process the blocks. If the **PARALLEL** option is not set, and you enable **LARGETABLESPLITENABLED**, then **PARALLEL** is set to 10 by default. If there are more blocks than threads, then remaining blocks are queued for processing and wait for a thread to become available.



The Parallel option is used to define the maximum number of separate threads that the FDM instance will utilise, each thread will consume additional CPU cycles as can be seen from the table in the masking example section.

3 Masking Best practices

Masking Examples

Data Sources	1	1	1	1
Number of Tables	10	10	1	1
Rows to mask per table	500,000	1,000,000	10,000,000	100,000,000
Columns to mask per table	10	10	10	10
FDM Engine instances used	1	10	1	1
Large Table Split	N	N	N	Y
Number of FDM threads per instance	1	1	1	8
Cells to Mask	50,000,000	100,000,000	100,000,000	1,000,000,000
Estimated Masking Time	1 Minute 14 seconds	2 Minutes 15 Seconds	22 Minutes	1 Hour 7 Minutes
Cells Masked Per Minute	4,166,666	44,444,444	4,545,455	15,151,515
Memory Usage (Gb)	1.3	8	1.8	2
CPU Usage (mcores)	330	7000	1000	5000

Note: 1000 mcores = 1 CPU core

Best Practice and Advice:

When masking a large amount of data it is essential that the Database server has been configured with the appropriate amount of temporary and data storage

A FDM Docker container by default will run between one and four docker instances. A single FDM instance will utilise approximately 1.8Gb and this can grow to over 4Gb when four instances are in use.

When utilising the FDM parallel option to split very large tables and run these are separate threads from an FDM instance the Docker container will consume approximately the same amount of memory but the container will consume additional CPU cores

4 Environment

TDM Environment

TDM 4.8.1 deployed using Docker containers running on RedHat 7.6.1810 with three scaled masking engines / 4 CPU's and 16 GB Memory. The TDM 4.8.1 FDM Docker containers were also deployed to a CentOS 6.9 with ten scaled masking engines/ 8 CPU's and 32 GB Memory.

Target Database

Oracle Database 19c Standard Edition Release 19.0.0.0.0 / Windows 4 CPU's / 16 GB Memory

50 Tables each with 100 Columns, Tables have 1 Million rows and 1 Table with over 100 Million rows

Additionally an Oracle 11.2 Enterprise Editions release 11.2.0.1/Linux with 8 CPU's / 16 GB Memory

FDM Configuration Settings

BATCHSIZE=37500

BLANKSASNULLS=Y

COMMIT=37500

EMPTYASNULL=Y

FETCHSIZE=75000

GETTABLEROWCOUNTS=N

ORDERBY=N

PARALLEL=10

LARGETABLESPLITENABLED=Y

5 Masking Results

Broadcom Lab environment

Masking functions used

FIXED
HASHLOV
TRANSLATE

Number of Data Sources	Number of Tables	Number of Rows being Masked	Number of Columns being Masked	Time Taken (Minutes)	Rows Masked per minute	Cells Masked per minute	Number of FDM Instances	Total Cells Masked
1	1	1,000,000	5	1.2	833,333	4,166,667	1	5,000,000
1	1	1,000,000	10	1.98	505,051	5,050,505	1	10,000,000
1	4	4,000,000	40	2.68	1,492,537	22,276,676	4	16,000,000
1	5	5,000,000	50	3.57	1,400,560	26,129,855	5	25,000,000
1	1	13,000,000	10	32	406,250	4,062,500	1	130,000,000
1	1	100,000,000	10	67	1,351,351	13,513,514	1	1,000,000,000

Customer Environment

RHEL Enterprise V7.x
6 vCPU
50 GB of RAM
400 GB of local disk allocated to /TDM

Number of Data Sources	Number of Tables	Number of Rows being Masked	Number of Columns being Masked	Time Taken (Minutes)	Rows Masked per minute	Cells Masked per minute	Number of FDM Instances	Total Cells Masked
1	1	9,493,732	12	20	474,687	5,696,239	1	113,924,784
1	5	229,982,033	67	348	658,603	10,060,611	5	3,501,092,768

6 Database Considerations

As the number of rows to be masked start getting into the millions of rows, the users need to take into account the following database conditions.

Oracle

For a successful masking of very large tables (tables over 100 million rows) in Oracle, we would suggest that you follow the recommendations:

TEMP Tablespace

To improve the performance of the temporary segment management within a temporary tablespace, the following recommendations are suggested.

Create a new “BIG FILE” temporary tablespace as the example shown:

CREATE **BIGFILE** TEMPORARY TABLESPACE "TEMP01"

TEMPFILE '/opt/ora112/oradata/masker/temp0101.dbf' SIZE 15G

AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL **UNIFORM SIZE 264k**;

In the above example, we need to take into account the value of the “uniform size”, where in our example is 264k. The formula is UNIFORM as follows (N* S + B).

Where

- N = positive number
- S = SORT_AREA_SIZE = 262144 = 264k
- B = DB_BLOCK_SIZE = 8192 (this is usually the default value) = 8k

So the formula breaks down to (1 * 264 + 8), which equals 264k. This should be the minimum recommended value, since your “SORT_AREA_SIZE” value will drive this value.

This temporary tablespace is useful for the management of the large number of temporary segments that will be used during the masking process, as well as utilizing a “BIGFILE” definition for generating a very large file that can grow to a maximum size of 32TB (per 8k db block size).

Please make sure that the user running the masking jobs in the Oracle database instance, defaults to this new temporary tablespace with the following command:

Alter user <masking_user>

Temporary tablespace temp01;

UNDO Tablespace

You might need to set the following parameter in the UNDO tablespace to work in coordination with the temporary segments management.

```
CREATE BIGFILE UNDO TABLESPACE "UNDOTS2" DATAFILE
'/opt/ora112/oradata/masker/undots101.dbf' SIZE 100G AUTOEXTEND ON NEXT 100M MAXSIZE
UNLIMITED RETENTION GUARANTEE;

ALTER SYSTEM set undo_tablespace=UNDOTS2 scope=both sid='*';

DROP TABLESPACE UNDOTBS1;
```

This will allow the Oracle database instance to automatically manage the segments in the UNDO tablespace, just like what is happening in the BIGFILE temporary tablespace that we created.

The recommendation is to change the system level “undo_tablespace” parameter to reflect the newly created undo tablespace and drop the prior undo tablespace.

Oracle DB Instance Parameters

The following database instance parameters need to be modified to at least the values below or higher.

- SORT_AREA_SIZE=262144
- SORT_AREA_RETAINED_SIZE=262144
- UNDO_MANAGEMENT=AUTO
- UNDO_RETENTION=75000
- RESULT_CACHE_MAX_SIZE=0
- RESULT_CACHE_MODE=AUTO

For the “undo_retention” value, you will arrive at the required value by running the following query

```
select max(maxquerylen) from v$undostat;

select (<maxquerylen-value>/60)/60 query,(25000/60)/60 retention from dual;
```

Where the “maxquerylen” value will be provided to the second query, this will provide you with a number for the “undo_retention” value.

Microsoft SQL Server

Database Files

As the database is defined, the LOG files need to be at 80% of the ROWS files.

TEMP Database

Make sure that the TEMP database has at least 90% of the available capacity of the client's database where the large tables to be masked are located.

Shrink Database Files

Prior to executing the masking in the large tables, perform a database shrink command via the SSMS and select the client database or TEMP database to shrink, you can also issue the T-SQL command "

```
SELECT name,
file_id,
type_desc,
size * 8 / 1024 [TempdbSizeInMB]
FROM tempdb.sys.database_files
ORDER BY type_desc DESC,
file_id;
```

After you have reviewed the database size, you can issue the "DBCC SHRINKFILE(temp,10)".

7 Database Statistics

Oracle Statistics

The following queries need to be executed to find out the collection of statistics at the database instance (SYS – DBA_*) objects and the fixed (X$) objects loaded into the SGA.

```
prompt 'Statistics for SYS tables'

SELECT NVL(TO_CHAR(last_analyzed, 'YYYY-Mon-DD'), 'NO STATS') last_analyzed, COUNT(*) dictionary_tables

FROM dba_tables

WHERE owner = 'SYS'

GROUP BY TO_CHAR(last_analyzed, 'YYYY-Mon-DD')

ORDER BY 1 DESC;


prompt 'Statistics for Fixed Objects'

select NVL(TO_CHAR(last_analyzed, 'YYYY-Mon-DD'), 'NO STATS') last_analyzed, COUNT(*) fixed_objects

FROM dba_tab_statistics

WHERE object_type = 'FIXED TABLE'

GROUP BY TO_CHAR(last_analyzed, 'YYYY-Mon-DD')

ORDER BY 1 DESC;
```

If you find out that the statistics are old, you need to run the following Oracle packages.

```
-- gathering statistics for SYS

exec dbms_stats.gather_schema_stats('SYS');

exec dbms_stats.gather_database_stats (gather_sys=>TRUE);

exec dbms_stats.gather_dictionary_stats;


-- gathering statistics for fixed objects

exec dbms_stats.gather_fixed_objects_stats;
```

If you need to re-compute the statistics for the given table (greater than 100m rows), you can use the following Oracle package

```
exec DBMS_STATS.GATHER_TABLE_STATS (ownname => '"LARGESET"',tabname => '"CDBAADMTST"', estimate_percent => 50);
```

Troubleshooting Oracle Errors

If you are encountering ORA-30036 errors, you can use the following scripts to help you solve this type of ORA error.

I) Check free space in the undo tablespace**.**

```
select sum(bytes) from dba_free_space where tablespace_name='<undo tablespace>';

select sum(bytes) from dba_data_files where tablespace_name='<undo tablespace>';
```

II) Check whetherUndo tablespace datafile is autoextensible.

```
select autoextensible from dba_data_files where tablespace_name='<undo tablespace>;
```

III) Check whether unexpired extents are available in the same segment as the current transaction.

```
SELECT DISTINCT STATUS, SUM(BYTES), COUNT(*) FROM DBA_UNDO_EXTENTS GROUP BY STATUS;
```

In case no undo space is left, then we try to use unexpired extents (Undo Extent required to honor UNDO_RETENTION).

This sometimes results in ORA-1555 errors. Now if you do not have unexpired extents also, then you need to add space to undo tablespace.

IV) Check the status of the Undo extents.

```
SELECT DISTINCT STATUS, SUM(BYTES), COUNT(*),TABLESPACE_NAME FROM DBA_UNDO_EXTENTS GROUP BY STATUS,TABLESPACE_NAME;
```

If there are no Expired extents that can be re-used then its possible to encounter ORA-30036. If we see mostly Active extents then this is most likely Undo sizing issue. In this case, check if Undo Tablespace is correctly sized. The following query calculates the number of bytes needed (based on the current workload):

```
        SELECT (UR * (UPS * DBS)) + (DBS * 24) AS "Bytes"

FROM (SELECT value AS UR FROM v$parameter WHERE name = 'undo_retention'),

(SELECT (SUM(undoblks)/SUM(((end_time - begin_time)*86400))) AS UPS FROM v$undostat),

(select block_size as DBS from dba_tablespaces where tablespace_name=

(select value from v$parameter where name = 'undo_tablespace'));
```

*(UR) UNDO_RETENTION in seconds

*(UPS) Number of undo data blocks generated per second

*(DBS) Overhead varies based on extent and file size (db_block_size)

Sizing an UNDO tablespace requires three pieces of data.

(UR) UNDO_RETENTION in seconds

(UPS) Number of undo data blocks generated per second

(DBS) Overhead varies based on extent and file size (db_block_size)

The undo space needed is calculated as:

UndoSpace = UR * (UPS * DBS)

Two of the pieces of information can be obtained from the instance configuration: UNDO_RETENTION and DB_BLOCK_SIZE. The third piece of the formula requires a query being run against the database. The maximum number of undo blocks generated per second can be acquired from V$UNDOSTAT.

The following formula calculates the peak undo blocks generated per second:

```
SELECT undoblks/((end_time-begin_time)*86400) "Peak Undo Block Generation" FROM v$undostat
WHERE

undoblks=(SELECT MAX(undoblks) FROM v$undostat);
```

Column END_TIME and BEGIN_TIME are DATE data types. When DATE data types are subtracted, the resulting value is the # of days between both dates. To convert days to seconds, you multiply by 86400, the number of seconds in a day (24 hours * 60 minutes * 60 seconds).

The following query calculates the number of bytes needed to handle a peak undo activity:

```
SELECT (UR * (UPS * DBS)) AS "Bytes"

FROM (SELECT value AS UR FROM v$parameter WHERE name = 'undo_retention'),

(SELECT undoblks/((end_time-begin_time)*86400) AS UPS

FROM v$undostat

WHERE undoblks = (SELECT MAX(undoblks) FROM v$undostat)),

(SELECT block_size AS DBS

FROM dba_tablespaces

WHERE tablespace_name = (SELECT UPPER(value) FROM v$parameter WHERE name =
'undo_tablespace'));
```

For 10g and Higher Versions where Tuned undo retention is being used, please use below query:

```
SELECT (UR * (UPS * DBS)) AS "Bytes"

FROM (select max(tuned_undoretention) AS UR from v$undostat),

(SELECT undoblks/((end_time-begin_time)*86400) AS UPS

FROM v$undostat

WHERE undoblks = (SELECT MAX(undoblks) FROM v$undostat)),

(SELECT block_size AS DBS
```

```
    FROM dba_tablespaces

    WHERE tablespace_name = (SELECT UPPER(value) FROM v$parameter WHERE name =
'undo_tablespace'));
```