
Debunking the myths about redo, undo, commit & rollback

By

Riyaj Shamsudeen

Who am I?

- 15 years using Oracle products
- Over 14 years as Oracle DBA
- Certified DBA versions 7.0,7.3,8,8i &9i
- Specializes in performance tuning, Internals and E-business suite
- Currently consulting for Cingular through Avion systems
- Adjunct faculty – Northlake College
- OakTable member
- Email: rshams4 at yahoo.com



Disclaimer

These slides and materials represent the work and opinions of the author and do not constitute official positions of my current or past employer or any other organization. This material has been peer reviewed, but author assume no responsibility whatsoever for the test cases.

If you corrupt your databases by running my scripts, you are solely responsible for that.

This material should not should not be reproduced or used without the authors' written permission.

blah..blah..

Table structure for test case

- Following structure used in all test cases:

```
create table redo_internals_tbl (  
    char_column      char(5),  
    varchar2_column varchar2(20)  
)
```

“What do you mean rollback is generating excessive redo , due to that failed parallel DML failure? Why would rollback generate redo ? Rollback does not
Generate any redo.”

Incorrect

Myth 1: Rollback does ~~not~~ generate redo

- Changes from SQL statements, generate redo records.
- Redo records contains change vectors for both data and undo segments blocks.
- Change vectors for data blocks specify how to do the change
- Change vectors for undo blocks specify how to undo the change

Myth 1: Rollback does ~~not~~ generate redo

- Test case:
 - Table populated with 1001 rows initially and committed.
 - Rows are updated using the SQL:
update redo_internals_tbl
set varchar2_column = replace(varchar2_column,'X','Y');
 - Redo size measured for the update statement
 - Then rollback;
Rollback;
 - Redo size measured for the rollback.

Myth 1: Rollback does ~~not~~ generate redo

- Test case:

Redo size for the update statement => 121,792 bytes

Redo size for the rollback statement => 63,792 bytes

In this specific case, redo size for the rollback is nearly half of the update statement.

Script: redo_myth_01.sql


Myth 1: Rollback does ~~not~~ generate redo

- Let's look at the micro level redo records inserting one row, followed by a rollback.

Myth 1: Rollback does ~~not~~ generate redo

Redo record for insert statement

Change vector for undo block



```
REDO RECORD - Thread:1 RBA: 0x000014.00000002.0010 LEN: 0x0128 VLD: 0x05
SCN: 0x0000.0008c994 SUBSCN: 1 07/22/2007 09:41:41
CHANGE #1 TYP:0 CLS:32 AFN:2 DBA:0x00800027 OBJ:4294967295 SCN:0x0000.0008c992 SEQ: 1 OP:5.1
ktudb redo: siz: 64 spc: 7110 flg: 0x0022 seq: 0x00c2 rec: 0x09
            xid: 0x0008.002.0000012e
ktubu redo: slt: 2 rci: 8 opc: 11.1 objn: 52562 objd: 52562 tsn: 4
Undo type: Regular undo      Undo type: Last buffer split: No
Tablespace Undo: No
            0x00000000
KDO undo record:
KTB Redo
op: 0x02 ver: 0x01
op: C uba: 0x00800027.00c2.08
KDO Op code: DRP row dependencies Disabled
xtype: XA flags: 0x00000000 bdba: 0x01000221 hdba: 0x0100020c
itli: 1 ispac: 0 maxfr: 4858
tabn: 0 slot: 1(0x1)
```

```
CHANGE #2 TYP:0 CLS: 1 AFN:4 DBA:0x01000221 OBJ:52562 SCN:0x0000.0008c992 SEQ: 3 OP:11.2
KTB Redo
op: 0x02 ver: 0x01
op: C uba: 0x00800027.00c2.09
KDO Op code: IRP row dependencies Disabled
xtype: XA flags: 0x00000000 bdba: 0x01000221 hdba: 0x0100020c
itli: 1 ispac: 0 maxfr: 4858
tabn: 0 slot: 1(0x1) size/delt: 17
fb: --H-FL-- lb: 0x1 cc: 2
null: --
col 0: [ 2] 41 32
col 1: [10] 53 45 43 4f 4e 44 20 52 4f 57
```

Change vector for the data block



Myth 1: Rollback does ~~not~~ generate redo

Redo record for rollback statement

```
REDO RECORD - Thread:1 RBA: 0x000006.00000002.0010 LEN: 0x00bc VLD: 0x05  
SCN: 0x0000.0008c2c2 SUBSCN: 1 07/22/2007 09:01:47  
CHANGE #1 TYP:0 CLS: 1 AFN:4 DBA:0x01000221 OBJ:52556 SCN:0x0000.0008c2c0 SEQ: 1 OP:11.3
```

KTB Redo

op: 0x03 ver: 0x01

op: Z

KDO Op code: DRP row dependencies Disabled

xtype: XR flags: 0x00000000 bdba: 0x01000221 hdba: 0x0100020c

itli: 2 ispac: 0 maxfr: 4858

tabn: 0 slot: 1(0x1)

KDO undo record:

KTB Redo

op: 0x02 ver: 0x01

op: C uba: 0x00800027.00c2.08

KDO Op code: DRP row dependencies Disabled

xtype: XA flags: 0x00000000 bdba: 0x01000221 hdba: 0x0100020c

itli: 1 ispac: 0 maxfr: 4858

tabn: 0 slot: 1(0x1)

Change vector from
update statement shown
for side-by-side
comparison

“Delete generates more redo than inserts..”

That depends

Myth 2: Delete generates ~~more~~ redo than inserts

- That depends.
- Many variables in play
 - Multi row vs single row inserts
 - Multi row vs single row deletes
 - Presence of indices
 - Presence of triggers etc

Myth 2: Delete generates ~~more~~ redo than inserts

Table structure	Insert type Multi/single	Delete Multi/single	Insert Redo size	Delete redo size
Regular table	Multi	Multi	4.4M	27M
Regular table	Multi	Single row Loop based	4.4M	27M
Regular table	Single row loop based	Multi	27M	27M
Regular table	Single row loop based	Single row loop based	27M	27M

Myth 2: Delete generates more redo than inserts

Delete generates almost same amount of redo, but insert redo size increases
For single row inserts

Table structure	Insert type Multi/single	Delete Multi/single	Insert Redo size	Delete redo size
Regular table w/index	Multi	Multi	18M	47M
Regular table w/index	Multi	Single row Loop based	18M	47M
Regular table w/index	Single row loop based	Multi	59M	48M
Regular table w/index	Single row loop based	Single row loop based	59M	48M

“Increasing the log file size does not change redo size..”

Correct

Myth 3: Increasing the log file size ~~increases~~ redo

size

- Log file size does not affect redo size generated from a transaction.
- This myth can be disproved by following test case:
 - Insert into a table with log file size 50MB
 - Create a new log group with log file size 100MB
 - Switch the log file
 - Insert into a table and measure redo size

Myth 3: Increasing the log file size increases redo

size

MEMBER	GROUP#	BYTES	STATUS
C:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO01.LOG	1	52428800	CURRENT

Inserting 11 rows in to a table and measure redo size
Total redo generated ==>2536

-- Adding 100MB log file

```
alter database add logfile group 99  
( 'C:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\redo99.log' ) size 104857600 reuse
```

MEMBER	GROUP#	BYTES	STATUS
C:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\REDO99.LOG	99	104857600	CURRENT

Inserting 11 rows in to a table and measure redo size
Total redo generated ==>2536

Script:redo_myth_03.sql

“Commit forces all dirty buffers to be written, from the buffer cache..”

Incorrect

Myth 4:

Commit ~~forces~~ all dirty buffers to be written

- Commit does not force dirty buffers to be written.
- Commits are not successful until the Log writer writes log buffers to the disk.
- DBWR writes dirty buffers at a different pace, then the log writer.

Myth 4:

Commit ~~forces~~ all dirty buffers to be written

Test case:

A table created in Example tablespace.

-- Checkpoints to make sure all dirty buffers are cleaned.

```
alter system checkpoint;
```

-- Query to count dirty buffers:

```
select file#, dirty , count(*) cnt from v$dbh b
```

```
where b.file# = (select file_id from dba_data_files where tablespace_name='EXAMPLE')
```

```
group by file#, dirty
```

```
/
```

FILE#	DI	CNT
5	N	24

← No dirty buffers initially

Myth 4:

Commit ~~forces~~ all dirty buffers to be written

Test case:

-- Inserting 1001 rows

```
insert into redo_internals_tbl
```

```
  select 'A' || n, rpad( n, 20,'X') from
```

```
    (select level n from dual connect by level <= 1001) d;
```

-- Counting dirty buffers again:

```
select file#, dirty , count(*) cnt from v$dbh b
```

```
where b.file# = (select file_id from dba_data_files where tablespace_name='EXAMPLE')
```

```
group by file#, dirty;
```

FILE#	DI	CNT
5	Y	35
5	N	21

Myth 4:

Commit ~~forces~~ all dirty buffers to be written

-- Committing.

```
Commit;
```

-- Counting # of dirty buffers after the commit.

```
select file#, dirty , count(*) cnt from v$dbh b
```

```
where b.file# = (select file_id from dba_data_files where  
tablespace_name='EXAMPLE')
```

```
group by file#, dirty
```

```
/
```

FILE#	DI	CNT
5	Y	35
5	N	21

As shown here, commit did not force DBW to write dirty buffers. There are still many dirty buffers after commit.

Script:redo_myth_04.sql

“Uncommitted changes are not written by DBWR, if it does a transaction can issue a rollback and that can cause corruption..”

Incorrect

Myth 5:

Uncommitted buffers are ~~not~~ written by DBW

- Dirty buffers can be written by DBW even if the transaction modifying the buffer has not committed yet.
- This means that a transaction could rollback the changes after DBW has written the dirty buffer to disk. There is no real harm since subsequent rollback will undo the change and DBW will write again.
- Log writer plays critical role in data consistency.

Myth 5:

Uncommitted buffers are ~~not~~ written by DBW

Session #1:

-- Query to count dirty buffers:

```
select file#, dirty , count(*) cnt from v$dbh b
where b.file# = (select file_id from dba_data_files where tablespace_name='EXAMPLE')
group by file#, dirty;
```

FILE#	DI	CNT
5	N	24

Inserting 1001 rows in to a table

1001 rows created.

of dirty buffers BEFORE the commit

FILE#	DI	CNT
5	Y	35
5	N	21

Myth 5:

Uncommitted buffers are ~~not~~ written by DBW

Session #2: alter system checkpoint;


Session #1:

-- Still this transaction has not committed yet..

of dirty buffers BEFORE the commit

FILE#	DI	CNT
-----	---	-----
5	N	56

Transaction in session #1 hasn't
Committed yet. But, the dirty buffers have
Been written by DBW



Script:redo_myth_05.sql

“Increase the log buffer to 50M, but beware that transaction redo size can increase..”

Incorrect

Myth 6:

Increasing log buffer size ~~will~~ increase the redo size

- Log buffer Size does not alter redo size from DML statements.
- This myth can be disproved by measuring redo size for different log buffer size.

Log buffer	# of rows	Redo size (bytes)	Insert mode
7MB	11	708	Conventional
11MB	11	708	Conventional
23MB	11	708	Conventional

Script:redo_myth_06.sql

“To improve performance, just set nologging in all tables and indices, and change DB mode to noarchivelog mode and that will eliminate redo..”

Incorrect

Myth 7:

Nologging inserts generates ~~no~~ redo

- Direct mode or nologging inserts generates minimal redo.
- Blocks are preformatted, populated and written directly to disk during direct mode inserts, above HWM.
- An invalidation redo generated invalidating these new blocks, generating minimal redo.
- Many preconditions must be met.

Myth 7: Nologging inserts generates ~~no~~ redo

- In this test case, rows inserted in to a regular table in conventional and direct mode.
- This shows that, of course, direct mode generates lot less redo.

Case	# of rows	Structure	Redo size	Insert mode
#1	100,000	Regular table	4.4M	Conventional
#2	100,000	Regular table	5,668 bytes	Direct

Script:redo_myth_07.sql

Myth 7: Nologging inserts generates no redo

Corollary: Adding an index disables much benefits of direct mode inserts.

```
create index redotest.redo_i1 on redotest.redo_internals_tbl  
  (varchar2_column) nologging;
```

Case	# of rows	Structure	Redo size	Insert mode
#3	100,000	Regular table w/index	24 MB	Conventional
#4	100,000	Regular table w/index	14.2 MB	Direct

After adding index,

redo size increased from 4.4M to 24MB for conventional

redo size increased from 5668b to 14MB for direct mode

Myth 7: Nologging inserts generates no redo

Corollary: Direct logging inserts is disabled for index organized table.

```
create table redotest.redo_internals_tbl (  
    char_column      char(10) primary key,  
    varchar2_column  varchar2(20)  
) organization index
```

Case	# of rows	Structure	Redo size	Insert mode
#5	100,000	Index organized table	41.3MB	Conventional
#6	100,000	Index organized table	41.2MB	Direct

Myth 7: Nologging inserts generates no redo

Corollary: Adding a foreign key constraint disables direct mode inserts and resorts to conventional mode logging.

```
alter table redotest.redo_internals_tbl add
```

```
(foreign key (char_column) references redotest.redo_fk_tbl (char_column) );
```

Case	# of rows	Structure	Redo size	Insert mode
#7	100,000	Table with out foreign key	44.7MB	Conventional
#8	100,000	Table with foreign key	44.7MB	Conventional
#9	100,000	Table with foreign key	44.7MB	Direct
#10	100,000	Table with out foreign key	5,778 bytes	Direct

Script:redo_myth_07a.sql

Myth 7: Nologging inserts generates no redo

Corollary: Row level triggers disable many redo optimization techniques. Direct mode inserts are disabled too, and works like a conventional mode.

Case	# of rows	Structure	Redo size	Insert mode
#11	100,000	Table with row level trigger	27MB	Conventional
#12	100,000	Table with row level trigger	27MB	Direct
#13	100,000	Table w/out a row level trigger	4.4MB	Conventional
#14	100,000	Table w/out a row level trigger	5,668 bytes	Direct

Script:redo_myth_07b.sql

“Use global temporary tables. DML against GTTs do not generate any redo..”

Incorrect

Myth 8: DML on global temporary tables does not produce redo

- Physical segments for Global temporary tables allocated in TEMP tablespace, no redo generated for these segments.
- But GTT supports rollback, and changes to rollback segments generates redo.
- Direct mode inserts into GTT, generates even less redo.

Case	# of rows	Structure	Redo size	Insert mode
#1	10,001	Regular table	442 KB	Conventional
#2	10,001	Regular table	4680	Direct
#3	10,001	Global temporary table	32K	Conventional
#4	10,001	Global temporary table	444	Direct

Script:redo_myth_08.sql

“During application upgrade process, just change your database to noarchivelog mode and that should eliminate redo..”

Incorrect

Myth 9: Changing database to noarchivelog mode ~~disables~~ redo generation completely.

- Changing the database mode to noarchivelog mode, does not alter the way redo is generated.

Case	# of rows	Structure	Redo size	Insert mode
#11	1001	Archivelog mode/Regular table	45,520	Conventional
#12	1001	Arc hivelog mode/Regular table	4680	Direct
#13	1001	Noarchivelog mode/regular table	45,520	Conventional
#14	1001	Noarchivelog mode/regular table	4680	Direct

Script:redo_myth_09.sql

“undo_retention should not be increased, since that can increase redo size..”

Incorrect

Myth 10: undo_retention set to non-zero value generates ~~more~~ redo.

- Undo_retention determines, for how long undo records shouldn't be overwritten. Value of this parameter does not change the redo generation.
- This test case be disproved with various values for undo_retention parameter.

Case	# of rows	Undo_Retention	Redo size	Insert mode
#1	101	90	4532	Conventional
#2	101	300	4532	Conventional
#3	101	3000	4532	Conventional

Script:redo_myth_10.sql

“updating a column with same value does not actually update it, since Oracle compares pre-update and post-update values..

Incorrect

Myth 11: updating a column to same value

```
update redo_internals_tbl set varchar2_column =  
varchar2_column;
```

```
CHANGE #3 TYP:0 CLS:36 AFN:3 DBA:0x00c0943f OBJ:4294967295 SCN:0x0000.0075c131 SEQ: 1 OP:5.1
```

```
...  
KDO undo record:
```

```
KTB Redo
```

```
op: 0x03 ver: 0x01
```

```
compat bit: 4 (post-11) padding: 0
```

```
op: Z
```

```
KDO Op code: URP row dependencies Disabled
```

```
 xtype: XAxtype KDO_KDOM2 flags: 0x00000080 bdba: 0x01005620 hdba: 0x0100560c
```

```
itli: 2 ispac: 0 maxfr: 4858
```

```
tabn: 0 slot: 0(0x0) flag: 0x2c lock: 0 ckix: 32
```

```
ncol: 2 nnew: 1 size: 0
```

```
Vector content:
```

```
col 1: [20] 31 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
```

```
...
```

```
KTB Redo
```

```
op: 0x03 ver: 0x01
```

```
compat bit: 4 (post-11) padding: 0
```

```
op: Z
```

```
KDO Op code: URP row dependencies Disabled
```

```
 xtype: XRxtype KDO_KDOM2 flags: 0x00000080 bdba: 0x01005620 hdba: 0x0100560c
```

```
itli: 2 ispac: 0 maxfr: 4858
```

```
tabn: 0 slot: 0(0x0) flag: 0x2c lock: 0 ckix: 32
```

```
ncol: 2 nnew: 1 size: 0
```

```
Vector content:
```

```
col 1: [20] 31 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
```

Same value in
Both undo &
Redo records

```
Script:redo_myth_11.sql
```

“updating a column updates every column in that row,
and so you can update columns even if it is
unnecessary..”

Incorrect

Myth 12: updating all columns..

```
CHANGE #3 TYP:0 CLS:36 AFN:3 DBA:0x00c0943f OBJ:4294967295 SCN:0x0000.0075c131 SEQ: 1 OP:5.1
```

```
...
```

```
KDO undo record:
```

```
KTB Redo
```

```
op: 0x03 ver: 0x01
```

```
compat bit: 4 (post-11) padding: 0
```

```
op: Z
```

```
KDO Op code: URP row dependencies Disabled
```

```
 xtype: XAxtype KDO_KDOM2 flags: 0x00000080 bdba: 0x01005620 hdba: 0x0100560c
```

```
itli: 2 ispac: 0 maxfr: 4858
```

```
tabn: 0 slot: 0(0x0) flag: 0x2c lock: 0 ckix: 32
```

```
ncol: 2 nnew: 1 size: 0
```

```
Vector content:
```

```
col 1: [20] 31 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
```

```
...
```

```
KTB Redo
```

```
op: 0x03 ver: 0x01
```

```
compat bit: 4 (post-11) padding: 0
```

```
op: Z
```

```
KDO Op code: URP row dependencies Disabled
```

```
 xtype: XRxtype KDO_KDOM2 flags: 0x00000080 bdba: 0x01005620 hdba: 0x0100560c
```

```
itli: 2 ispac: 0 maxfr: 4858
```

```
tabn: 0 slot: 0(0x0) flag: 0x2c lock: 0 ckix: 32
```

```
ncol: 2 nnew: 1 size: 0
```

```
Vector content:
```

```
col 1: [20] 31 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
```

Updated varchar2_column,
And redo shows update
For that column

```
Script:redo_myth_12.sql
```

References

- Oracle support site. Metalink.oracle.com. Various documents
- Internal's guru Steve Adam's website

www.ixora.com.au

- Jonathan Lewis' website

www.jlcomp.daemon.co.uk

- Julian Dyke's website

www.julian-dyke.com

- 'Oracle8i Internal Services for Waits, Latches, Locks, and Memory'
by Steve Adams

- Tom Kyte's website

Asktom.oracle.com