# All about locks: DML, DDL, foreign key, online operations

**Franck Pachot**
**dbi services**
**Switzerland**

**Keywords:**

Locks, Online operations,

## Introduction

Locking in Oracle is usually an easy topic: most of the time, you don't have to use explicit locks (the LOCK TABLE statements). The implicit locking done by DML (insert, update, delete, select for update) is transparent, and most of the time efficient. A query (select without for update) does not lock anything. Deadlocks are rare and *enqueue* wait events are not so frequent.

But when you need to go further, to understand a deadlock situation, to understand why a session is blocked, or to have an efficient referential integrity validation, then all that becomes more complex. The locking mechanism in Oracle is not simple to understand. There are several names for the same things, and those names can be misleading. Just a few examples: locks on rows are called *transaction locks*, but some table locks are called *row share* or *row exclusive*… even if they do not lock any rows. Those *row exclusive* locks are not so exclusive because they can be acquired by several sessions concurrently on the same resource. Those *row exclusive* locks can also be called *sub exclusive* locks, and there is a *share sub exclusive* lock mode as well…

Don't panic, we will take those terms one by one and you will understand everything. You will even be able to remember easily the lock compatibility matrix just because you will understand the meanings of lock modes.

## Lock Types

A lock is a mechanism used to serialize access to a resource. As concurrent sessions will wait for the resource, as in a queue, they are also called *enqueues*, and this is the term used in wait events to measure the time waited. Oracle uses locks to:

- protect data as tables, rows, index entries, … (DML Locks)
- protect metadata in the dictionary or in the shared pool (Data Dictionary Lock)
- or to protect internal objects in memory (Latches, Mutextes and other internal locks).

Here we will be talking about data only. Data locks are also called **DML locks** because they are used for DML (Data Manipulation Language), but they are also used by DDL (Data Definition Language) when it accesses data.

There are three **types** of **DML locks**:

- **Row level locks** are called *transaction locks* (**TX**) because, even if they are triggered by a concurrent DML on a row, the locked resource is the transaction. TX *enqueues* are not waiting for a row, but for the completion of the transaction that has updated the row.
  The TX lock is identified by the transaction id v$transaction
- **Table level locks** are called *table locks* (**TM**) and the locked resource is the database object (table, index, partition…). In addition to DML or DDL, they can be acquired explicitly with the LOCK TABLE statement.
  The TM locks are identified by an object_id (as in dba_objects)
- **User defined locks** (**UL**) resource is not an Oracle object but just a number that has a meaning only for the application. They are managed by the dbms_lock package.

Here we are talking about table locks (TM) only, but we will explain quickly TX locks in order to clear confusion between table level and row level locks.

**Lock Modes**

Basically, a resource can be locked in two modes: **Exclusive** (to prevent any concurrent access) or **Share** (to prevent only exclusive access). But Oracle has defined 6 modes (including the no lock mode) and each one has several names. *Table 1 - Lock mode names* shows the 6 lock modes, with their numbers and different names that are we can find in v$ views, in the documentation, in trace files, in OEM…

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1** | NL | | Null | | | | N |
| **2** | SS | RS | Row-S | Row Share | SubShare | Intended Share (IS) | L |
| **3** | SX | RX | Row-X | Row Exclusive | SubExclusive | Intended Exclusive (IX) | R |
| **4** | S | | Share | | | | S |
| **5** | SSX | SRX | S/Row-X | Share Row Exclusive | Share-SubExclusive | | C |
| **6** | X | | Exclusive | | | | X |

Table 1 - Lock mode names

And when we look at the wait events, from V$SESSION, or from the Blocking Sessions screen of Entreprise Manager, we don't have those names but a number:

| | P1TEXT | P1 | P1RAW |
|---|---|---|---|
| **TM** mode **1**: | name/mode | 1414332417 | 544D0001 |
| **TM** mode **2**: | name/mode | 1414332418 | 544D0002 |
| **TM** mode **3**: | name/mode | 1414332419 | 544D0003 |
| **TM** mode **4**: | name/mode | 1414332420 | 544D0004 |

| | | | |
|---|---|---|---|
| **TM** mode **5**: | name/mode | 1414332421 | `544D0005` |
| **TM** mode **6**: | name/mode | 1414332422 | `544D0006` |
| **TX** mode **4**: | name/mode | 1415053316 | `54580004` |
| **TX** mode **6**: | name/mode | 1415053318 | `54580006` |

Table 2 - enqueue wait event parameters

How to remember all those names and numbers without confusion? And guess the DML operations that has cause them? And the compatibility matrix that shows what is allowed and what is blocked by a lock (*Table 3 - Lock compatibility matrix*)?

It is possible: we will explain the meaning of **Share** and **Exclusive**, as well as the meaning of **Row** or **Sub** or **Intended**, and everything will be clear.

**Share / Exclusive**

An **exclusive lock** (X) disallows to share a resource: it prevents another session to acquire a share lock (S) or an exclusive lock (X) on the same resource.

A **share lock** (S) allows sharing a resource: multiple sessions can acquire a share lock (S) on the same resource. But it prevents another session to acquire an exclusive lock (X) on the same resource. When reading data, we usually want to prevent concurrent writes, so we need a share access on it.

Here we have the basic elements to understand and remember the compatibility matrix: For X and S locks, the matrix is: S/S are compatible but S/X, and X/X are not compatible.

The general idea behind X (exclusive) and S (share) is that

- When we need to **write** data, we acquire an **exclusive** lock.
- When we need to **read** the data and make sure that no one is writing concurrently, then we acquire a **share** lock. If someone is already updating the data (holding an X lock), then we must wait to see if his change is committed or not. Or we may choose not to wait and cancel our reading operation.

Note that on Oracle, locking in share mode is required only when reading the **current** version of data. A query (such as a `select` without the `for update clause`) can read a consistent version of data without any locks.

**Sub / Row**

We have seen that there are sometimes two names and two abbreviations for the same mode: Sub (S) and Row (R).

Let's focus on table locks. The resource locked by Share (S) and Exclusive (X) is a table. And a table is made of rows. Oracle has lock modes that can be acquired on a resource, but that concern only a subpart of the locked resource. If the resource is a table, then the subparts are rows.

For example, if we update one or more records in a table (insert, update, or delete), we are writing so we need an exclusive lock. But we are not writing on the whole table. We do not need to lock the whole table in exclusive mode. So we will acquire a lock that concerns only some rows. This is the **Row-X** lock (Row Exclusive). Similarly, if we want to acquire a lock for blocking reads (reading data and prevent concurrent modification on what we read), as the '`select for update`' did before version 9i, then rather than acquiring a Share lock on whole table we can acquire a Row-S (Row Share) lock.

We are at table level but we acquire **Row-S** and **Row-X** locks for row modification.

Here is the reason for the dual naming **Sub**/**Row**: in the case of a table lock, the subparts are **Rows**, so we can talk about Row-S (RS) et Row-X (RX). But in the general case, for a lock acquired for a **Sub**part of a resource, the name is **Sub-S** (Sub Share) and **Sub-X** (Sub Exclusive).

And even for a table lock, if the table is partitioned, then an exclusive lock X on a partition will acquire a **Sub-X** on the table as well (the subpart here is the partition, so there is no reason to use Row-X naming here).

In this document, we will continue by reasoning on table locks, and we will use **Row** naming rather than the **Sub** naming. But the actual reason is that I prefer to avoid the confusion since the abbreviation of Sub (S) is the same as the one for Share…

**Table level / Row level**

Here we have been talking about locks acquired at table level (TM locks) even if they concern subparts.

Besides that, the DML operations (INSERT, UPDATE, DELETE or SELECT FOR UPDATE) have to acquire locks at row level. Two sessions can concurrently modify rows in the same table if they do not touch the same row. And we have seen that this requires a TM-RX (Row-X) lock on the table. However, in addition to that **table level lock**, each session will also acquire a lock on the row itself: within the block, the row will have a lock flag, which is a pointer to the ITL entry, which identifies the transaction that has updated the row.

It is a **row level lock** but the resource that is locked is actually the transaction, which is the reason why it is a called a TX lock. A transaction sees that a row is locked by another transaction, and then it will request a lock on the transaction that locked the rows - waiting for end of that transaction.

That TX lock is always exclusive: Oracle did not implement share lock at the record level (Other RDBMS needs it for transaction isolation, but Oracle uses multi-versioning for that).

So, we have **table locks** that concern the whole table (TM locks in mode S and X) and we have **row level locks** (TX locks).

Then what is the reason for **table level row locks** (TM locks in mode Row-S and Row-X) ?

A table can have millions of rows. When we need a share lock on the table (TM-Share), it would not be efficient to scan all the table rows in order to see if there are some rows that are locked. In the other way, acquiring an exclusive lock at table level when we need to update only few rows would be disastrous for concurrency and scalability.

This is where **Sub** table locks are used: a transaction that has the **intention** to modify some rows will first acquire a Row-X lock on the table, without having to check all rows. This **intention** explains the third name that we can find less frequently for Sub/Row locks: **Intended Share** (IS) and **Intended eXclusive** (IX).

Now we can understand that two sessions can acquire an RX lock on the same table, even if 'X' means 'exclusive'. Remember that we are talking about **table level locks**, even if some of them have the 'row' term in their names.

At this level two transactions can modify rows in the same table. Both can acquire a TM lock in Row-X mode (RX), because both can have the intention to modify rows.

**Read / Write**

We have seen that the general idea is that an exclusive lock (X) is acquired when you are writing and a share lock (S) is acquired when you are reading and want that read to prevent concurrent writes.

In the same way, a Sub eXclusive lock (RX or SX) is acquired when you have the intention to write a subpart, and a Sub Share lock (RS or SS) is acquired when you have the intention to do a blocking reads on a subpart.

But in Oracle, we have to go further:

Oracle performs non-blocking reads by default, meaning that you can read consistent data without having to block concurrent writes (See Tom Kyte link below). Those reads are known as 'query mode' or '**consistent** ' read. They do not lock the data because they do not have to read the **current** version of the blocks. If there is concurrent writes on the block, they build a previous version of it by applying undo information.

When you want to do a blocking read, you use a '`select for update`' which locks the row without updating it (event if it is often done with the goal to update it – as the name suggests). Until 9i, the '`select for update`' acquired a share lock (Row-S), which is still in the idea of reading. But from 9i, an exclusive lock is acquired: the `select for update` has the same locking behavior than an update. Besides that, at row level, the `select for update` lock has always been an exclusive TX lock.

So we can stay in the idea that **share** is for reading and **exclusive** is for writing. But then we must remember that in Oracle the non-blocking reads do not need to acquire any lock, and that a `select for udpate` is actually considered as writing. After all, it generates redo and undo, and it cannot be done on a read-only database.

The DDL (Data Definition Language) also acquires table locks (TM). An `alter table move`, for example, will put an X lock on the table: it writes into the table and must prevent blocking reads (S,RS) and writes (X,RX). A create index, however, does not modify the table, but it has to read data while there is no concurrent modifications during the index creation, in order to have the current version of data, thus it locks the table in S mode.

Referential integrity also acquires TM locks. For example, the common issue with unindexed foreign keys leads to S locks on child table when you issue a delete, or update on the key, on the parent table. This is because without an index, Oracle has no single lower level resource to lock in order to prevent a concurrent insert that can violate the referential integrity. When the foreign key columns are the leading columns in a regular index, then the first index entry with the parent value can be used as a single resource and locked with a row-level TX lock.

And what if referential integrity has an `on delete cascade` ? In addition to the S mode, there is the intention to update rows in the child table, as with Row-X (RX) mode. This is where the share row exclusive (SRX) occurs: S+RX=SRX.

**Table operations**

**SELECT,** without a FOR UPDATE, is a non-blocking read, so it does not acquire any lock in Oracle. You can even drop a table while another session is reading it.

Above, we have seen that a read must acquire a Share lock in order to prevent concurrent modifications. But there, there is no need for a lock because there cannot be any concurrent modification. In Oracle, a simple select do not read the current value of data, but a past image at the time of the beginning of the query (in read-commited isolation level) or at the time of the beginning of the transaction (in serializable or read-only isolation level). The past image cannot have concurrent modifications, so there is not lock.

**INSERT, UPDATE, DELETE**, have the intention to write on some rows, so they acquires a Row-X mode table lock (in addition to the row level TX locks that will be requested when a locked row is encountered).

**SELECT FOR UPDATE** is doing blocking reads on some rows, so it acquired a Row-S before 9i. But now it has the same behavior as an update and acquires a Row-X.

**DML with referential integrity** need additional locks. Row-S or Row-X are acquired on all tables linked by referential integrity.

In addition to that, a delete or an update of the referenced columns on the parent table will acquire a Share lock on the child if there is no index starting with the foreign key.

If the referential integrity has an ON DELETE CASCADE, then the Share lock requested by the unindexed foreign keys become a Share Row Exclusive one as it adds the Row-X from the delete.

**A direct-path INSERT** has to prevent any concurrent modification, thus it acquires an X lock on the table. If it is inserted into a specific partition (naming the partition with the table name), then the table has a Sub-X lock for the intention to write on a subpart, and the partition has an X lock.

All those locks are acquired until the end of the transaction (commit or rollback) at the exception of the TM-Share of non-indexed foreign keys that is released faster.

**DDL** can acquire locks, for the whole operation or for only a short period (and then it can be considered as an **online** operation).

When you need to achieve repeatable reads (to avoid phantom reads – concurrent inserts that would change your result set) then you cannot rely on row-level locks (TX) for an obvious reason: you cannot lock a row that do not exists yet (there is no 'range lock' in Oracle). Then you need to lock the whole table and this is done with a Share lock. For example, if you create an index, or rebuild it without the online option, the DDL acquires a TM-Share lock on the table.

And when a DDL need exclusive write access, such an ALTER TABLE MOVE, a TM-X mode lock is acquired.

Those are just examples. There are many situations that we cannot explain here and that change with versions. But thinking about which the set of data that is written, and which one need to be read with blocking reads, will help you to understand what happens.

In addition to that, you can acquire table locks with the LOCK TABLE statement and following modes: ROW SHARE, ROW EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE

You think there are already too many synonyms for Row-S, Sub-S, etc.? Here is another one: you can acquire it with `LOCK TABLE IN SHARE UPDATE MODE...`

**Compatibility Matrix**

The whole reason for all those lock modes is to allow or disallow (or serialize) concurrent access on a resource. So we need to know which one are compatible or not.

The lock compatibility matrix shows that.

| Compatible locks? | Row-S (RS) | Row-X (RX) | Share (S) | S/Row-X (SRX) | Exclusive (X) |
|---|---|---|---|---|---|
| Row-S (RS) | yes | yes | yes | yes | no |
| Row-X (RX) | yes | yes | no | no | no |
| Share (S) | yes | no | yes | no | no |
| S/Row-X (SRX) | yes | no | no | no | no |
| Exclusive (X) | no | no | no | no | no |

Table 3 - Lock compatibility matrix

Did you ever read and try to remember that compatibility matrix? Now that we understand the meaning of each mode, it can be easier. Let's build it from the definitions we have seen above.

We already have seen the compatibility about X and S: by definition, the matrix shows that S/S are compatible but S/X and X/X are not.

About subparts, this is different. Modifying a few rows in a table does not prevent another session to modify some (other) rows in the same table. The row-by-row conflict is managed by the row level locks (TX) but here we are talking about table locks (TM). Therefore the matrix shows that RS/RS, RX/RX, RS/RX are compatible.

But among resources and sub-resources, there may be conflict, and this is the reason for Sub-locks: If the entire table has an exclusive lock (X) acquired by another session, then it is not possible to intend a row modification (RX). And if there is an exclusive lock for some rows (RX) then it prevents to acquire a share lock on the table (S). For that reason, the matrix shows that X/RX, X/RS and S/RX not compatible. But S/RS are both sharable, so they are compatible.

And then about Share Row eXclusive (SRX) which is actually a combination of S and RX: the table has a share lock, and a subset has an exclusive lock.

Because X or RX are not compatible with the S in SRX, then X/SRX and RX/SRX are not compatible.

And because S (as well as the S in SRX) is not compatible with the RX in SRX, then S/SRX and SRX/SRX are not compatible either. But there is no conflict with RS/SRX as RS is compatible with S as well as with RX.

**Dictionary Views**

This is enough theory. We will now check how to see locks in our Oracle instance.

Let's do a DML operation and see which information can be gathered from dictionary views.

```
SESSION1> select sid from v$mystat where rownum=1;
       SID
----------
        13
SESSION1> update test set dummy='Y';
1 row updated.
```

So I'm in session 13 and I updated one row in the TEST table**.**

**DBA_LOCKS** shows all locks:

```
SESSION1> select * from dba_locks where session_id=13;
```

```
SESSION_ID LOCK_TYPE    MODE_HELD            MODE_REQUESTED LOCK_ID1
LOCK_ID2 LAST_CONVERT BLOCKING_OTHERS
---------- ------------ -------------------- -------------- --------
-------- ------------ ---------------
        13 DML          Row-X (SX)           None           723764
0                   1 Not Blocking
        13 Transaction  Exclusive            None           524303
43037               1 Not Blocking
```

Our transaction is holding two locks:

- One transaction lock (TX) in exclusive mode that was acquired when our transaction started. LOCK_ID1 and LOCK_ID2 identifies the transaction (USN/SLOT/SQN are encoded into 2 integers)
- One table level lock (DML) in Row-X (SX) mode acquired by the update statement as it has the intention to modify rows. LOCK_ID1 is the object id of the table. LAST_CONVERT shows that the transaction started 1 second ago and that the UPDATE statement started 1 second ago as well. We will see the exact time when locks are acquired later.

**DBA_DML_LOCKS** shows TM locks only:

```
SESSION1> select * from dba_dml_locks where session_id=13;
```

```
SESSION_ID LOCK_TYPE    MODE_HELD            MODE_REQUESTED LOCK_ID1
LOCK_ID2 LAST_CONVERT BLOCKING_OTHERS
---------- ------------ -------------------- -------------- --------
-------- ------------ ---------------
        13 E_FRANCK TEST     Row-X (SX)     None
1 Not Blocking
```

Here the object_id has been converted into owner and object_name.If we check in DBA_OBJECTS, the OBJECT_ID for E_FRANCK.TEST is 723764 as shows the LOCK_ID1 as given by DBA_LOCKS.

**V$LOCKED_OBJECTS** shows TM locks as well:

```
SESSION1> select * from v$locked_object where session_id=13;

    XIDUSN    XIDSLOT     XIDSQN  OBJECT_ID SESSION_ID
ORACLE_USERNAME OS_USER_NAME  PROCESS LOCKED_MODE
---------- ---------- ---------- ---------- ----------
-- ------------ ------- -----------
         8         15      43037     723764         13 E_FRANCK
e_FRANCK      6674              3
```

Here we have additional information:

- The transaction identification (XIDUSN,XIDSLOT,XIDSQN). Look at lock ID1 and ID2 for TX lock:LOCK_ID1= XIDUSN*65536+XIDSLOT and LOCK_ID2=XIDSQN
  That means that the TX and TM locks we see are held by the same transaction.
- We have the Oracle username and OS username and process ID as well the lock mode uses the mode number.
- Mode 3 is for Row-X.

These are our session locks. Now we open another session and attempt to lock the TEST table in S mode:

```
SESSION1> select sid from v$mystat where rownum=1;

       SID
----------
        47

SESSION2> lock table test in share mode;
```

That second session (SID 47) is now waiting and we will open a third session to check those blocking locks.

```
SQL> select * from dba_locks where session_id=13;

SESSION_ID LOCK_TYPE    MODE_HELD      MODE_REQUESTED LOCK_ID1
LOCK_ID2 LAST_CONVERT BLOCKING_OTHERS
---------- ------------ -------------- -------------- -------- -----
--- ------------ -----------------
        13 DML          Row-X (SX)     None           723764   0
10 Blocking
        13 Transaction  Exclusive      None           524303   43037
10 Not Blocking
```

Our first session (SID 13) still have the same locks, but we know that the DML lock is blocking another session.

```
SESSION_ID LOCK_TYPE    MODE_HELD      MODE_REQUESTED LOCK_ID1
LOCK_ID2 LAST_CONVERT BLOCKING_OTHERS
---------- ------------ -------------- -------------- -------- -----
--- ------------ -----------------
        13 DML          Row-X (SX)     None           723764   0
10 Blocking
        13 Transaction  Exclusive      None           524303   43037
10 Not Blocking
```

Our second session (SID 47 that did the lock statement) has not acquired (held) any lock yet. It is waiting to acquire the Share mode lock on the table (object_id 723724)

**DBA_BLOCKERS** shows which sessions are blocking another session:

```
SESSION1> select * from dba_blockers;

HOLDING_SESSION
---------------
             13
```

**DBA_WAITERS** has more information about the waiting sessions

```
SESSION1> select * from dba_waiters;

WAITING_SESSION HOLDING_SESSION LOCK_TYPE    MODE_HELD
MODE_REQUESTED     LOCK_ID1   LOCK_ID2
--------------- ---------------- ------------ ---------------- ------
---------- ---------- ----------
             47               13 DML          Row-X (SX)       Share
723764          0
```

Second session (SID 47) that requested TM lock in Share mode on TEST table (object_id 723764 ) is waiting for session 13 that holds Row-X lock mode on that table.

**UTLLOCKT.SQL** (script provided in ORACLE_HOME/rdbms/admin) shows it in formatted way so that we can see easily the waiters hierarchy:

```
SQL>@ ?/rdbms/admin/utllockt.sql
```

```
WAITING_SESSION   LOCK_TYPE         MODE_REQUESTED MODE_HELD
LOCK_ID1          LOCK_ID2
----------------- ----------------- -------------- -------------- --
-------------- -----------------
13                None
   47             DML               Share          Row-X (SX)
723764            0
```

**V$SESSION_WAITS** shows all current wait events. We are interested in enqueues:

```
SQL> select * from v$session_wait where event like 'enq%';

       SID EVENT                P1TEXT     P1RAW            P2TEXT
P2  SECONDS_IN_WAIT
---------- -------------------- ---------- ---------------- --------
-- ----------  ---------------
        47 enq: TM - contention name|mode  00000000544D0004 object #
723764              802
```

Here we see that second session session (SID 47) is currently waiting on a TM lock (enqueue). It has been waiting for 802 seconds (I ran that query a few minutes later). P2 is the object_id (object #) and P1 is the lock type and mode (name|mode) encoded in hexadecimal:544Dis the ascii code for 'TM' and4is the lock mode (Share). So we can have all information from waits events: session 47 is waiting for 802 seconds to acquire a TM lock in mode S on table TEST.

This is where the Table 2 - enqueue wait event parameters is useful.

**V$SESSION_EVENT** shows wait event statistics cumulated for the sessions.

```
SQL> select * from v$session_event where sid=47 and event like
'enq%';

       SID EVENT                          TOTAL_WAITS TOTAL_TIMEOUTS
TIME_WAITED AVERAGE_WAIT   MAX_WAIT
---------- ---------------------------- ----------- --------------
----------- ------------ ----------
        47 enq: TM - contention                  296            281
86490        292.2        295
```

We see our TM lock on which the session has been waiting 86490 centiseconds (865 seconds, I ran that query one minute after the previous one). I know that there were only one request for that lock, but 296 enqueue wait have been accounted for it. This is because waiting for a lock is not done with only one wait. The wait times out after 3 seconds (that's why you see the average wait time about 300 centiseconds, and that you have a number of timeouts that reaches the number if waits). This is the way it works: after 3 seconds waiting, the process takes control again, checks if it is not in a deadlock situation, and then waits another 3 seconds.

**Row level locks again, intention and transactions**

We have seen that from DBA_LOCKS the TM lock and the TX lock arrived at the same time because we started our transaction with the update statement. In truth, the TM lock was acquired before the TX lock because:

- TM lock is at table level, it is acquired as soon as the update is executed and before any rows have been read.
- TX lock is at row level. Even if the resource is the transaction, it is acquired when modifying a row

Let's prove that. We run the update again, but now we update no rows:

```
SESSION1> select sid from v$mystat where rownum=1;
       SID
----------
        18

SESSION1> update test set dummy='Y' where 0 = 1;
0 row updated.
```

So I'm in session 18 and I ran an update that has not updated any rows.

```
SESSION1> select * from dba_locks where session_id=18;

SESSION_ID LOCK_TYPE    MODE_HELD            MODE_REQUESTED LOCK_ID1
LOCK_ID2 LAST_CONVERT BLOCKING_OTHERS
---------- ------------ -------------------- -------------- --------
-------- ------------ ---------------
        18 DML          Row-X (SX)           None           723764
0                   1 Not Blocking
```

Even if we touched no rows, the Row-X was acquired by the update execution just because of our intention to update rows. But we actually update no rows so there is no TX lock yet. We have no TX lock yet, but we know we are in a transaction because our TM lock is related with a transaction (and will be released only at commit or rollback). Any doubt? Let's check V$LOCKED_OBJECT which has the transaction identification (USN/SLOT/SQN) for my TM lock.

```
SESSION1> select * from v$locked_object where session_id=13;

    XIDUSN    XIDSLOT     XIDSQN  OBJECT_ID SESSION_ID
ORACLE_USERNAME OS_USER_NAME  PROCESS LOCKED_MODE
---------- ---------- ---------- ---------- ---------- -------------
-- ------------- ------- -----------
         0          0          0     723764         18 E_FRANCK
e_FRANCK      11210              3
```

Our transaction has no USN/SLOT/SQL because it has no entry in the undo segment transaction table yet (because no row modification happened yet). And if we check V$TRANSACTION, there is no rows

for our transaction. However, V$SESSION has a transaction address in TADDR for our session. What does that mean? We are in a transaction (no doubt about that, we did an update and that is done within a transaction). Our transaction has an address, but it has no entry in the transaction table. So, the TX lock resource is a transaction table entry rather than a transaction, and V$TRANSACTION shows transaction entries rather than transactions.

We did not write anything, but our intention to write to TEST table is marked with the Row-X lock and will remain until the end of our transaction.

**From Enterprise Manager**

Locked sessions are easily shown in Entreprise manager from their enqueue wait event.

They are shown in red in the Top Activity chart (Figure 1 – OEM Performance / Top Activity)

And the equivalent to utllockt.sql is in the Blocking Sessions screen (Figure 2 – OEM 12c Performance - Blocking Sessions). However, this is where it is good to know the decimal values for P1 that we have in Table 2 - enqueue wait event parameters, so that we can easily recognize TM-Share as 1414332420.


Figure 1 – OEM Performance / Top Activity

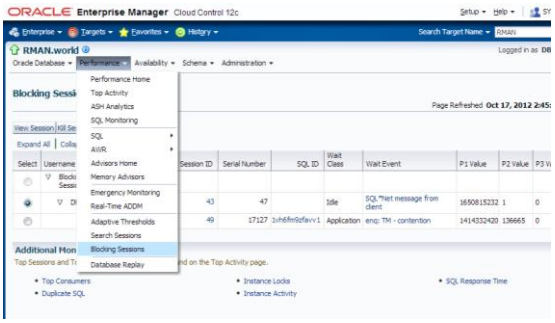


Figure 2 – OEM 12c Performance - Blocking Sessions

**Deadlock graph**

When locks are in a deadlock situation, Oracle will kill one session and dump the information in the session tracefile.

Here is an example where each session was waiting on the other:

```
Deadlock graph:
                       ---------Blocker(s)--------  ---------Waiter(s)---------
Resource Name          process session holds waits  process session holds waits
TM-000215da-00000000         49     172     SX             38      43           S
TM-000215d9-00000000         38      43     SX             49     172           S
```

So, we have two resource names with the lock type, the lock ID1 and the lock ID2

The lock type is a table locks (TM)

The object_id is in hexadecimal here (215da and 215d9), we have to convert it and retreive the table name from DBA_OBJECTS

Here is how to read the deadlock graph:

So on TABLE1 (215da) the session 172 holds a Row-X (SX) lock and the session 43 is waiting to aquire a Share (S) lock.

And on TABLE2 (215d9) the session 43 holds a Row-X (SX) lock and the session 172 is waiting to aquire a Share (S) lock.

From the trace file, we have more information about the requested lock. Because sessions were waiting on it we can know which statement was executing. But we don't have any information about the locks that was aquired before, except the table name and the lock mode.

**Event 10704 'trace enqueues'**

When we want to see all locks acquired and released by a session, we can set the event 10704. The in the tracefile we will be interrested by lines such as:

```
ksqgtl *** TM-00016ae5-00000000 mode=4 flags=0x401 timeout=21474836
***
```

and

```
ksqrcl: TM,16ae5,0
```

The first one is written when a table lock (TM) lock is acquired for object id 16ae5 in Share mode (4)

The second one is written when the lock is released.

**References**


- Oracle Database Concepts – DML locks
  http://download.oracle.com/docs/cd/E11882_01/server.112/e10713/consist.htm#CNCPT1340
- Thomas Kyte - Consistent Reads
  http://asktom.oracle.com/pls/apex/f?p=100:11:0::::P11_QUESTION_ID:27330770500351
- Kyle Hailey - Enqueue waits: Locks
  http://www.perfvision.com/papers/09_enqueues.ppt
- Oracle Comments in DBMS_LOCK package definition
  ORACLE_HOME/rdbms/admin/dbmslock.sql


**Contact:**


*Franck Pachot*
*franck.pachot@dbi-services.com*


ORACLE ACE

ORACLE Certified Master
Oracle Database 11g
Administrator