Enqueue Internals

트랜잭션 처리를 위해 락은 필수적인 요소이며, 이로 인해 많은 기술자료에서 락에 대해 설명 하고 있다. 따라서, 본 기사에서는 다른 자료에서 언급되는 일반적인 사항들을 최대한 배제하고, enqueue의 내부 구조 및 처리 방식에 대해서 논의하도록 하겠다.

㈜엑셈 수석 컨설턴트 김 시 연 (siyeon@ex-em.com)

enqueue의 내부 구조 및 처리 방식을 알기 위해서는 enqueue 리소스, enqueue 락, enqueue 아키텍처에 대한 이해가 필요하다. 따라서 본 기사에는 다음과 같은 목차에 따라 enqueue를 설명한다.

1. enqueue 리소스란?
2. enqueue 락이란?
3. owners 리스트, waiters 리스트, converters 리스트란?
4. enqueue 아키텍처
5. foreign key와 TM 락 관계
6. foreign key를 이용한 converters 리스트 테스트

1. enqueue 리소스란?

enqueue 리소스란 enqueue 락을 위해 사용되는 데이터베이스 리소스이다. 오라클은 enqueue 리소스 관리를 위해 X$KSQRS(kernel service enqueue resource) 메모리 구조를 사용하며, V$RESOURCE 뷰를 통해 확인이 가능하다. 그렇다면, V$RESOURCE 뷰 쿼리와 내용을 확인해보자.

```
SQL> select view_definition from v$fixed_view_definition
where view_name='GV$RESOURCE';

VIEW_DEFINITION
----------------------------------------
select inst_id,addr,ksqrsidt,ksqrsid1,ksqrsid2
from   x$ksqrs
where  bitand( ksqrsflg , 2 ) !=0

SQL>  select * from v$resource;

ADDR                TYPE                ID1                 ID2
----------------    ----    --------------    ------------------
C00000001556AA10    TS                   3                   1
C00000001556DC08    TX             1114135                24534
C000000015579B38    MR                  10                   0
C00000001557D170    TM               66607                   0
...
```

위의 결과에 의하면, enqueue 리소스 구조는 락 유형과 두 개의 식별자로 구성된다. 두 개의 식별자는 각각 ID1, ID2이다. <TYPE-ID1-ID2>의 구성은 데이터베이스에서 유일한 값을 가지게 되며, 이것을 'enqueue identifier' 라고 한다. 또한, 각 enqueue 리소스에는 enqueue lock을 관리하기 위한 3개의 링크드 리스트(linked-list)를

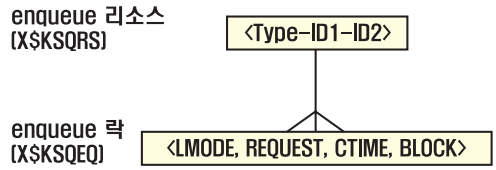
가지고 있다. 3개의 링크드 리스트는 1. Owners list 2. Waiters list 3. Converters list 이다. 그렇다면 enqueue lock 은 무엇이고, 3개의 링크드 리스트는 어떠한 용도로 사용되는지 확인해보자.

2. enqueue 락이란?

enqueue 락은 락 그 자체이다. 오라클은 락을 관리하기 위해 X$KSQEQ(kernel service enqueue object) 메모리 구조를 사용하며, V$ENQUEUE_LOCK 뷰를 통해 확인이 가능하다. 그렇다면 V$ENQUEUE_LOCK 뷰 쿼리와 내용을 확인해보자

```
SQL> select view_definition from v$fixed_view_definition
     where view_name='GV$ENQUEUE_LOCK';
VIEW_DEFINITION
----------------------------------------
select   s.inst_id,l.addr,l.ksqlkadr,s.ksusenum,r.ksqrsidt,
         r.ksqrsid1,r.ksqrsid2 ,
         l.ksqlkmod,l.ksqlkreq,l.ksqlkctim,l.ksqlklblk
from     x$ksqeq l,
         x$ksuse s,
         x$ksqrs r
where    l.ksqlkses=s.addr
and      bitand( l.kssobflg , 1 ) !=0
and      ( l.ksqlkmod!=0 OR l.ksqlkreq!=0 )
and      l.ksqlkres=r.addr

SQL> select * from v$enqueue_lock;
ADDR             KADDR            SID TYPE  ID1  ID2  LMODE REQUEST   CTIME BLOCK
------------     -------------- --- --- ---  --  ----  -----  -----  ----
C00000001550DAC0 C00000001550DAE0 164 TS     3    1     3       0  111257     0
C00000001550CBD0 C00000001550CBF0 165 RS    25    1     2       0 1111264     0
C00000001550CAA0 C00000001550CAC0 165 CF     0    0     2       0 1111267     0
C00000001550CA08 C00000001550CA28 165 XR     4    0     1       0 1111281     0
C00000001550CD00 C00000001550CD20 166 RT     1    0     6       0 1111264     0
C00000001550DA28 C00000001550DA48 167 MR    12    0     4       0  525644     0
...
```

위의 결과에 의하면, V$ENQUEUE_LOCK 뷰는 X$KSQEQ(kernel service enqueue object)와 X$KSUSE(kernel service session info), X$KSQRS(Kernel service enqueue resource) fixed 테이블로 구성되어있다. 이중에서 enqueue 락의 메모리 구조에 해당되는 X$KSQEQ 중에서 V$ENQUEUE_LOCK 뷰에서 보여지는 부분은 락 state object의 주소(ADDR), 락 주소(KADDR), 락 소유 모드(LMODE), 락 요청 모드(REQUEST), 락 소유 또는 요청 시점부터 현재까지의 경과시간(CTIME), 다른 락을 블로킹하는지의 여부를 나타내는 블로킹 플래그(BLOCK)이다. 정리하면, enqueue 락을 위한 주요 정보는 <LMODE-REQUEST-CTIME-BLOCK> 이라고 볼 수 있다. 또한, 락 정보를 위한 메모리 구조에는 TYPE, ID1, ID2 정보가 존재하지 않는다. 해당 정보는 enqueue 리소스를 위한 메모리 구조인 X$KSQRS와 조인(x$ksqeq.ksqlkres=x$ksqrs.addr)하여 V$ENQUEUE_LOCK에서 보여주는 것이다. 하나의 enqueue 리소스는 여러 개의 enqueue lock과 연결이 가능하다. 이해를 돕기 위해, 여기까지의 내용을 그림으로 표현하면 다음과 같다.



그렇다면 X$KSQEQ에는 모든 TYPE의 락 정보를 저장하고 있는 것일까? 결론부터 말하자면, 트랜잭션 락(TX) 및 DML 락(TM)은 enqueue 대기현상이 발생하지 않는다면 X$KSQEQ 메모리 구조에 저장되지 않는다. 오라클은 트랜잭션 락(TX) 정보를 저장하기 위해서는 X$KTCXB(kernel transaction control object-V$TRANSACTION_ENQUEUE 뷰의 base 테이블)를 사용하고, DML 락(TM) 정보를 저장하기 위해서는 X$KTADM(kernel transaction access definition dml lock) 메모리 구조를 사용한다. 테스트를 통해 확인해보자.

-- 테스트 시작

1) 테스트용 테이블 생성 및 테스트 데이터 입력

```
SQL> drop table enq1;

SQL> create table enq1 (id number, name char(10));

SQL> insert into enq1 values (1,'enq1');

SQL> insert into enq1 values (2,'enq2');

SQL> insert into enq1 values (3,'enq3');

SQL> commit;
```

2) delete 수행

```
SQL> delete enq1 where id=1;

1 row deleted
```

3) V$RESOURCE 및 V$ENQUEUE_LOCK 뷰 확인

```
SQL> select * from v$resource where type in ('TX','TM');

ADDR             TYPE                   ID1                  ID2
---------------- ---- -------------------- --------------------
C0000000155778D8 TM                   66620                    0
C0000000155860E0 TX                 1048604                21667
```

〈TM-66620-0〉〈TX-1048604,21667〉 2개의 리소스가 생성되었다.

```
SQL> select * from v$enqueue_lock where type in ('TX','TM');

no rows selected
```

2개의 리소스(〈TM-66620-0〉〈TX-1048604,21667〉)를 사용하는 enqueue 락이 존재하지 않는다.

4) V$TRANSACTION_ENQUEUE 뷰 확인

```
SQL> select * from v$transaction_enqueue;

ADDR             KADDR            SID TYPE       ID1      ID2      LMODE    REQUEST CTIME    BLOCK
---------------- ---------------- --- ------ ---------- ---------- ---------- ------- ---------- ----------
C0000000148285D0 C000000014828758  70 TX       1048604      21667          6       0        558          0
```

V$TRANSACTION_ENQUEUE 뷰에는 〈TX-1048604-21667〉 리소스를 사용하는 enqueue 락 〈6-0-558-0〉 이 1개 존재한다.

참조) V$TRANSACTION_ENQUEUE 뷰 쿼리

```
select  s.inst_id, l.ktcxbxba,l.ktcxblkp,s.ksusenum,r.ksqrsidt,
        r.ksqrsid1, r.ksqrsid2,l.ksqlkmod,l.ksqlkreq,l.ksqlktim,l.ksqlklblk
from    x$ktcxb l,
        x$ksuse s,
        x$ksqrs r
where   l.ksqlkses=s.addr
and     bitand( l.ksspaflg , 1 ) !=0
and     ( l.ksqlkmod!=0 OR l.ksqlkreq!=0 )
and     l.ksqlkres=r.addr
```

5) X$KTADM 확인

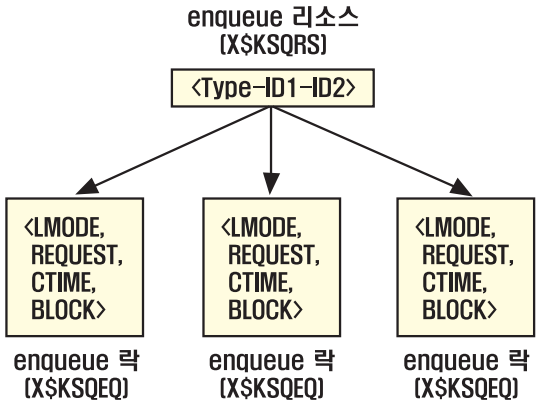
(아래의 쿼리는 V$LOCK 뷰 쿼리의 일부분이다. V$LOCK 뷰 쿼리의 전문은 Appendix를 참조하라)

```
SQL> select ksqlkmod  "LMODE",
            ksqlkreq  "REQUEST",
            ksqlktim "CTIME",
            ksqlklblk "BLOCK"
from   x$ktadm
where  bitand( kssobflg , 1 ) !=0
and  ( ksqlkmod!=0  OR ksqlkreq!=0 );

LMODE REQUEST  CTIME BLOCK
----- -------  ----- -----
    3       0  1308     0
```

〈TM-66620-0〉 리소스를 사용하는 enqueue lock 〈3-0-1308-0〉 이 1개 존재한다.

-- 테스트 완료

여기까지의 내용을 그림으로 표현하면 다음과 같다.



3. owners 리스트, waiters 리스트, converters 리스트란?

하나의 enqueue 리소스에는 복수개의 enqueue 락이 연결된다. 락 모드의 호환성 여부에 따라, enqueue 리소스를 공유해서 사용할 수도 있고, enqueue 리소스를 대기할 수도 있다. 오라클에서는 이 각각을 owners 리스트와 waiters 리스트로 관리하게 된다. 락 모드간의 호환성 여부는 아래의 표를 확인하기 바란다.

	N	SS	SX	S	SSX	X
N	O	O	O	O	O	O
SS	O	O	O	O	O	X
SX	O	O	O	X	X	X
S	O	O	X	O	X	X
SSX	O	O	X	X	X	X
X	O	X	X	X	X	X

그렇다면 테스트를 통해, owners 리스트와 waiters 리스트가 어떻게 관리되는지 확인해보자. 테스트의 편의성을 위해서 "lock table" 명령어를 사용한다.

-- 테스트 시작

1) 테스트용 테이블 생성 및 테스트 데이터 입력

```
SQL> drop table enq2;

SQL> create table enq2 (id number, name char(10));
```

2) 오브젝트 ID 확인

```
SQL> select object_id from dba_objects where object_name='ENQ2';

OBJECT_ID

----------

     66631
```

hex값으로 10447이다.

3) 2개의 세션에서 SX 모드로 lock table 수행

```
Session # 144

SQL> lock table enq2 in row exclusive mode;

Table(s) Locked.


Session # 148

SQL> lock table enq2 in row exclusive mode;

Table(s) Locked.
```

SX 모드간에는 호환성이 있으므로 enqueue 락 설정이 가능하다.

4) V$RESOURCE, V$LOCK 뷰 내용 확인

```
SQL> select * from v$resource where type='TM';

ADDR             TYPE                 ID1                  ID2

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

C000000035567BD0 TM                   66631                    0
```

〈TM-66631-0〉 enqueue 리소스가 1개 생성되었다.

```
SQL> select * from v$lock where type='TM';
```

```
ADDR             KADDR              SID   TYPE   ID1   ID2 LMODE REQUEST   CTIME BLOCK
-------------    ---------------    ---   ----   ----  --- ----- -------   ----- -----
C0000000447A00A8 C0000000447A00D0   144     TM 66631     0     3       0     145     0
C0000000447A01A8 C0000000447A01D0   148     TM 66631     0     3       0     142     0
```

〈TM-66631-0〉 enqueue 리소스를 사용하는 enqueue 락이 2개 생성되었다.

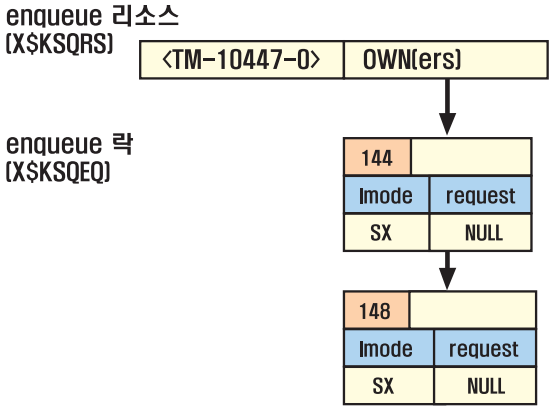
5) enqueue 덤프를 통한 현재 상태 분석

```
SQL> conn / as sysdba

SQL> oradebug setmypid

SQL> oradebug dump enqueues 10
```

```
res             identification       NUL SS  SX   S  SSX  X   md    link
             owners                        converters                    waiters
-------------------------------------------------------------------------------------------------------
c0000000355567bd0  TM-00010447-00000000 U  0   0   2   0    0   0    8     [c0000000355875a0,c0000000355875a0]
[c0000000347a01e0,c0000000347a00e0] [c0000000035567c00,c0000000035567c00] [c0000000035567bf0,c0000000035567bf0]
  lock          que  owner          session             hold wait ser link
-------------------------------------------------------------------------------------------------------
c0000000347a01d0  OWN  c0000000353febd0 c0000000353febd0 (148) SX   NLCK  15 [c0000000347a00e0,c0000000035567be0]
c0000000347a00d0  OWN  c0000000353f9a50 c0000000353f9a50 (144) SX   NLCK  18 [c0000000035567be0,c0000000347a01e0]
```

enqueue 리소스 〈TM-00010447-00000000〉 (TM 타입의 ID1은 OBJECT_ID이며 ENQ2의 OBJECT_ID는 66631 (0x10447)이다) 에 대해서 2개의 enqueue 락이 OWN(ers) 리스트에 존재한다. 각 enqueue 락은 SX 모드로 enqueue 리소스를 사용중이다.

여기까지의 테스트 결과를 그림으로 표현하면 다음과 같다.



6) X 모드로 lock table 설정

```
Session #149

SQL> lock table enq2 in exclusive mode;

-- enqueue waiting
```

SX 모드와 X 모드는 호환성이 없으므로 enqueue 대기가 발생한다.

7) V$RESOURCE, V$LOCK 뷰 내용 확인

```
SQL> select * from v$resource where type='TM';
```

```
ADDR                 TYPE                      ID1              ID2
------------------   ----   ------------------  --------------
C000000045567BD0     TM                     66631               0
```

기존의 내용과 동일하다.

```
SQL> select * from v$lock where type='TM';
```

```
ADDR             KADDR              SID  TYPE   ID1  ID2 LMODE  REQUEST  CTIME  BLOCK
----------       -------------      ---  ----   --  ---  ---   -------  -----  -----
C0000000347A00A8 C0000000347A00D0   144  TM   66631    0     3        0    247      1
C0000000347A01A8 C0000000347A01D0   148  TM   66631    0     3        0    244      1
C0000000347A02A8 C0000000347A02D0   149  TM   66631    0     0        6     60      0
```

〈TM-66631-0〉 enqueue 리소스를 사용하는 enqueue 락이 1개 추가 생성되었다. ENQ2 테이블에 EXCLUSIVE로 lock table을 수행한 149번 세션은 enqueue 대기를 하고 있다.

8) enqueue 덤프를 통한 현재 상태 분석

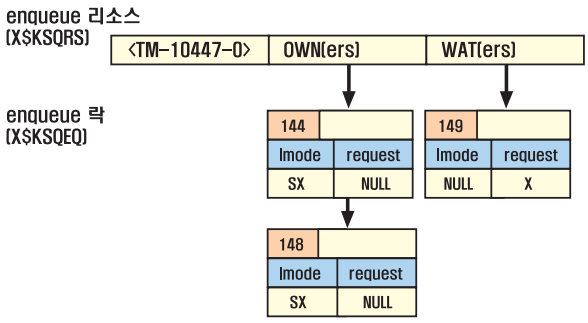
```
SQL> conn / as sysdba

SQL> oradebug setmypid

SQL> oradebug dump enqueues 10
```

```
res          identification  NUL SS SX  S SSX  X md  link
         owners                converters              waiters
```

```
-------------------------------------------------------------------------------
c0000000355567bd0  TM-00010447-00000000 U   0    0    2    0    0    0   8  [c0000000355875a0,c0000000355875a0]
[c0000000347a01e0,c0000000347a00e0] [c000000035567c00,c000000035567c00] [c0000000347a02e0,c0000000347a02e0]
lock             que owner           session              hold wait ser link
-------------------------------------------------------------------------------
c0000000347a01d0 OWN c0000000353febd0 c0000000353febd0 (148)   SX NLCK  15 [c0000000347a00e0,c000000035567be0]
c0000000347a00d0 OWN c0000000353f9a50 c0000000353f9a50 (144)   SX NLCK  18 [c0000000035567be0,c0000000347a01e0]
c0000000347a02d0 WAT c000000035400030 c000000035400030 (149) NLCK    X   4 [c0000000035567bf0,c0000000035567bf0]
```

enqueue 리소스에 대해서 2개의 enqueue 락이 OWN(ers) 리스트에 존재하며, 1개의 enqueue 락이 WAT(ers) 리스트에 존재한다. WAT(ers) 리스트에 존재하는 enqueue 락은 enqueue 리소스를 X 모드로 요청하고 있다.

여기까지의 테스트 결과를 그림으로 표현하면 다음과 같다.



9) 144번 세션에서 SHARE 모드로 lock table 수행

현재, 144번 세션은 현재 <TM-10447-0> 리소스에 대해 SX 모드로 enqueue 락을 획득하여 사용 중이다. 이때, shared mode로 lock table을 수행하면 enqueue 대기현상이 발생한다.

```
SQL> lock table enq2 in share mode;

-- enqueue waiting
```

10) V$RESOURCE, V$LOCK 뷰 내용 확인

```
SQL> select * from v$resource where type='TM';

ADDR             KADDR            SID TYPEID1 ID2  LMODE   REQUEST  CTIME BLOCK
---------------- ---------------- --- ------- ---- ------- -------- ----- -----
C0000000347A00A8 C0000000347A00D0 144 TM66631   0      3        5    37     1
C0000000347A01A8 C0000000347A01D0 148 TM66631   0      3        0    34     1
C0000000347A02A8 C0000000347A02D0 149 TM66631   0      0        6    31     0
```

11) enqueue 덤프를 통한 현재 상태 분석

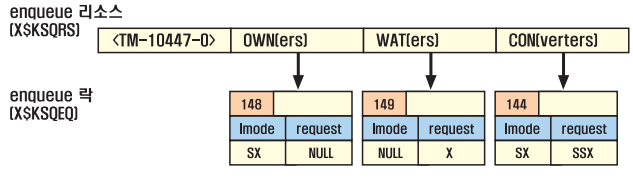
```
SQL> conn / as sysdba

SQL> oradebug setmypid

SQL> oradebug dump enqueues 10
```

```
res            identification    NUL  SS SX   S SSX  X md   link
          owners                    converters                   waiters
-------------------------------------------------------------------------------
c0000000035572728  TM-00010447-00000000 U   0    0    2    0    0    0   8  [c0000000355875a0,c0000000355875a0]
[c0000000347a01e0,c0000000347a01e0]  [c0000000347a00e0,c0000000347a00e0]  [c0000000347a02e0,c0000000347a02e0]
lock             que owner           session              hold wait ser link
-------------------------------------------------------------------------------
c0000000347a01d0 OWN c0000000353febd0 c0000000353febd0 (148)   SX NLCK  15 [c0000000035572738,c0000000035572738]
c0000000347a00d0 CON c0000000353f9a50 c0000000353f9a50 (144)   SX  SSX  18 [c0000000035572758,c0000000035572758]
c0000000347a02d0 WAT c000000035400030 c000000035400030 (149) NLCK    X   4 [c0000000035572748,c0000000035572748]
```

enqueue 리소스에 대해서 1개의 enqueue 락이 OWN(ers) 리스트에 존재하며, 1개의 enqueue 락이 WAT(ers) 리스트에 존재하며, 1개의 enqueue 락이 CON(verters) 리스트에 존재한다. CON(verters) 리스트에 존재하는 enqueue 락은 기존에 SX 모드였으며, SSX 모드로 락 모드를 변경하려고 한다..

즉, SX 모드로 락을 소유한 상태에서 S 모드를 추가로 요청하게 되면, SX 모드의 락을 릴리즈하고 S 모드의 락을 요청하는 것이 아니라, 기존의 SX 모드의 락을 SSX 모드의 락으로 변경(Converter)하는 방식으로 동작하게 된다

여기까지의 테스트 결과를 그림으로 표현하면 다음과 같다.



12) 148번 세션에서 롤백 수행 후 V$RESOURCE, V$LOCK 뷰 내용 확인

```
SQL>select * from v$lock where type='TM'

ADDR             KADDR             SID  TYPE    ID1  ID2 LMODE REQUEST CTIME BLOCK
--------------   ---------------   ---  ---  ----  ---  ----  -----  ----  ----
C0000000347A00A8 C0000000347A00D0 144   TM 66631    0     5       0    36     1
C0000000347A02A8 C0000000347A02D0 149   TM 66631    0     0       6   160     0
```

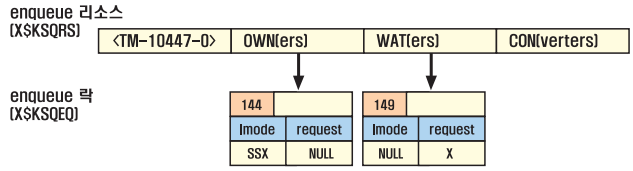
144번 세션이 SSX(5) 모드로 락을 획득하였다.

13) enqueue 덤프를 통한 현재 상태 분석

```
SQL> conn / as sysdba
SQL> oradebug setmypid
SQL> oradebug dump enqueues 10

res          identification  NUL SS SX  S  SSX   X  md link
       owners                converters              waiters
-------------------------------------------------------------------------------
c000000035572728  TM-00010447-00000000 U   0   0   0   0    1   0   20 [c00000003558 75a0,c0000000355875a0]
[c0000000347a00e0,c0000000347a00e0]   [c0000000355727 58,c0000000355727 58]   [c0000000347a02e0,c0000000347a02e0]
lock           que owner          session              hold wait ser  link
-------------------------------------------------------------------------------
c0000000347a00d0 OWN c0000000353f9a50 c0000000353f9a50 (144)  SSX NLCK  18 [c0000000355727 38,c0000000355727 38]
c0000000347a02d0 WAT c000000035400030 c000000035400030 (149) NLCK   X   4 [c0000000355727 48,c0000000355727 48]
```

enqueue 리소스에 대해서 1개의 enqueue 락이 OWN(ers) 리스트에 존재하며, 1개의 enqueue 락이 WAT(ers) 리스트에 존재한다. CON(veters) 리스트에 위치하였던 enqueue 락이 OWN(ers) 리스트로 이동한 것을 알 수 있다. 즉, CON(veters) 리스트에 존재하는 enqueue 락에 대한 처리가 WAT(ers) 리스트에 존재하는 enqueue 락에 대한 처리보다 우선시 된다. 조금만 생각해보면, 이것은 당연한 처리 순서인 것을 알 수 있다.

여기까지의 테스트 결과를 그림으로 표현하면 다음과 같다.



-- 테스트 완료

4. enqueue 아키텍처

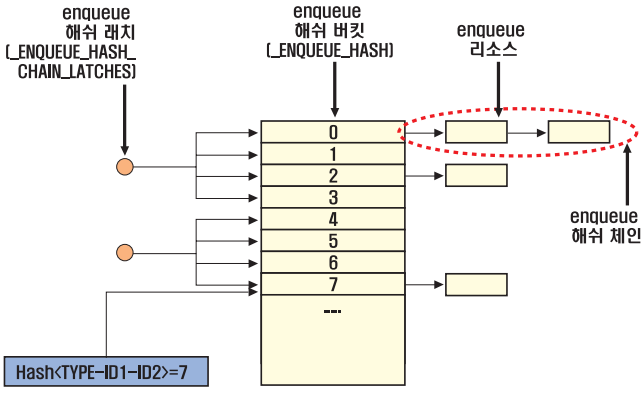
내부적으로 enqueue 아키텍처는 버퍼 캐시 아키텍처와 매우 유사하다. 주요 컴포넌트로는 enqueue 해쉬 체인, enqueue 해쉬 체인 래치, enqueue 해쉬 테이블 및 enqueue 리소스 등이다. enqueue 해쉬 체인 래치, enqueue 해쉬 테이블, enqueue 해쉬 체인간의 관계는 아래와 같다.

enqueue 해쉬 체인 래치 <-(1:m)-> enqueue 해쉬 버켓 <- (1:1) -> enqueue 해쉬 체인

enqueue 리소스는 'enqueue identifier' (<TYPE-ID1-ID2>)를 이용하여 해싱한 값을 통해, 적절한 enqueue 해쉬 테이블에 해쉬 되고 적절한 enqueue 해쉬 체인 위에 위치하게 된다.

enqueue 해쉬 체인 래치는 enqueue 해쉬 테이블과 해쉬 체인을 보호한다. enqueue 해쉬 체인 래치의 기본설정 값은 CPU_COUNT 와 동일하며, _ENQUEUE_HASH_CHAIN_LATCHES 파라미터에 의해 조정이 가능하다.

enqueue 해쉬 테이블의 기본 설정 길이는 SESSIONS 파라미터에 의해 결정되며, 계산 공식은 (SESSIONS - 10) * 2 + 55 이다. enqueue 해쉬 테이블의 길이는 _ENQUEUE_HASH 파라미터를 이용하여 조정이 가능하다. enqueue 아키텍처에 대한 개략도는 다음과 같다.



5. Foreign key와 TM 락 관계

인덱스를 생성하지 않은 foreign key 칼럼은 TM 락 경합을 발생시키는 대표적인 원인이다. 일부 문서에서, 오라클 9i 부터는 인덱스를 생성하지 않는 foreign key 칼럼이 TM 락 문제를 유발시키지 않는다고 되어 있지만, 여전히 TM 락 문제는 발생하게 된다. 따라서 이번 섹션에서는 오퍼레이션 유형별 테스트를 통해, 어떠한 경우에 TM 락 경합이 발생하는지를 확인해보도록 하겠다. 테스트는 oracle 10g Release 2 에서 수행하였다.

-- 테스트 시작

1) 테이블 및 Constraint 생성

```
SQL> drop table child;

SQL> drop table parent;

SQL> create table parent (id number, name char(10));

SQL> alter table parent add constraint parent_pk primary key (id);


SQL> create table child (id number, name char(10), parent_id number);

SQL> alter table child add constraint child_fk foreign key (parent_id)
     references parent(id);
```

2) 테스트 데이터 입력

```
SQL> insert into parent values (1, 'p1');

SQL> insert into parent values (2, 'p2');

SQL> insert into parent values (3, 'p3');


SQL> insert into child values (1, 'c1', 1);

SQL> insert into child values (2, 'c2', 2);

SQL> commit;
```

3) parent 테이블에 INSERT

```
SQL> insert into parent values(4,'m4');

SQL> @chk_lock       -- Appendix 참조

     SID OBJECT_NAM TYPE    ID1       ID2     LMODE   REQUEST     BLOCK
-------- ---------- --- ------  -------  -------  -------  -------
     144 CHILD      TM    64470        0        2        0        0
     144 PARENT     TM    64468        0        3        0        0
```

4) parent 테이블에 DELETE

```
SQL> delete parent where id=3;

SQL> @chk_lock

     SID OBJECT_NAM TYPE     ID1       ID2     LMODE   REQUEST     BLOCK
-------- ---------- --- ------  -------  -------  -------  -------
     144 PARENT     TM     64468        0        3        0        0
```

5) parent 테이블에 UPDATE

```
SQL> update parent set id=4 where id=3;

SQL> @chk_lock

      SID OBJECT_NAM  TYPE     ID1     ID2    LMODE   REQUEST   BLOCK
--------- ---------  ------ ----  ------  ------  -------  ------
      144 PARENT      TM     64468       0        3        0        0
```

PARENT 테이블에 INSERT 오퍼레이션을 수행할 경우 CHILD 테이블에 TM 락을 RS(2) 모드로 설정하며, PARENT 테이블에 DELETE/UPDATE 오퍼레이션을 수행할 경우

CHILD 테이블에는 TM락을 설정하지 않는다.

6) child 테이블에 INSERT

```
SQL> insert into child values (3, 'c1', 3);

SQL> @chk_lock

      SID OBJECT_NAM TYPE     ID1     ID2   LMODE   REQUEST   BLOCK
--------- --------- ------ ---- ------ ------ ------- ------
      144 CHILD      TM     64470      0       3         0       0
      144 PARENT     TM     64468      0       2         0       0
```

7) child 테이블에 DELETE

```
SQL> delete child where id=2;

SQL> @chk_lock

      SID OBJECT_NAM TYPE     ID1     ID2   LMODE   REQUEST   BLOCK
--------- --------- ------ ---- ------ ------ ------- ------
      144 CHILD      TM     64470      0       3         0       0
      144 PARENT     TM     64468      0       2         0       0
```

8) child 테이블에 UPDATE

```
SQL> update child set id=1 where id=2;

SQL> @chk_lock

      SID OBJECT_NAM TYPE     ID1     ID2   LMODE   REQUEST   BLOCK
--------- --------- ------ ---- ------ ------ ------- ------
      144 CHILD      TM     64470      0       3         0       0
```

CHILD 테이블에 INSERT/DELETE 오퍼레이션을 수행할 경우 PARENT 테이블에 TM락을 RS(2) 모드로 설정하며, CHILD 테이블에 UPDATE 오퍼레이션을 수행할 경우 PARENT 테이블에는 TM락을 설정하지 않는다. 3~8 테스트의 내용을 표로 정리하면 다음과 같다.

	OPERATION					
	PARENT INSERT	PARENT DELETE	PARENT UPDATE	CHILD INSERT	CHILD DELETE	CHILD UPDATE
PARENT	RX	RX	RX	RS	RS	
CHILD	RS			RX	RX	RX

9) parent 테이블에 INSERT 후 child 테이블에 INSERT

```
Session #144

SQL> insert into parent values (4,'p4');

Session #145

SQL> insert into child values (3,'c3',3);


SQL> @chk_lock

      SID OBJECT_NAM TYPE     ID1     ID2   LMODE   REQUEST   BLOCK
--------- --------- ------ ---- ------ ------ ------- ------
      144 CHILD      TM     64470      0       2         0       0
      144 PARENT     TM     64468      0       3         0       0
      145 PARENT     TM     64468      0       2         0       0
      145 CHILD      TM     64470      0       3         0       0
```

10) parent 테이블에 INSERT 후 child 테이블에 DELETE

```
Session #144

SQL> insert into parent values (4,'p4');

Session #145

SQL> delete child where id=2;


SQL> @chk_lock


      SID OBJECT_NAM TYPE     ID1     ID2   LMODE   REQUEST   BLOCK
--------- --------- ------ ---- ------ ------ ------- ------
      144 CHILD      TM     64470      0       2         0       0
      144 PARENT     TM     64468      0       3         0       0
      145 PARENT     TM     64468      0       2         0       0
      145 CHILD      TM     64470      0       3         0       0
```

11) parent 테이블에 INSERT 후 child 테이블에 UPDATE

```
Session #144

SQL> insert into parent values (4,'p4');

Session #145

SQL> update child set id=1 where id=2;


SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2    LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  --------
      144  CHILD       TM     64470       0       2         0         0
      144  PARENT      TM     64468       0       3         0         0
      145  CHILD       TM     64470       0       3         0         0
```

12) parent 테이블에 DELETE 후 child 테이블에 INSERT

```
Session #144

SQL> delete parent where id=3;

Session #145

SQL> insert into child values (2,'c2',2);


SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2    LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  --------
      144  PARENT      TM     64468       0       3         0         0
      145  CHILD       TM     64470       0       3         0         0
      145  PARENT      TM     64468       0       2         0         0
```

13) parent 테이블에 DELETE 후 child 테이블에 DELETE

```
Session #144

SQL> delete parent where id=3;

Session #145

SQL> delete child where id=2;


SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2    LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  --------
      144  PARENT      TM     64468       0       3         0         0
      145  CHILD       TM     64470       0       3         0         0
      145  PARENT      TM     64468       0       2         0         0
```

14) parent 테이블에 DELETE 후 child 테이블에 UPDATE

```
Session #144

SQL> delete parent where id=3;

Session #145

SQL> update child set id=1 where id=2;


SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2    LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  --------
      144  PARENT      TM     64468       0       3         0         0
      145  CHILD       TM     64470       0       3         0         0
```

15) parent 테이블에 UPDATE 후 child 테이블에 INSERT

```
Session #144

SQL> update parent set id=4 where id=3;

Session #145

SQL> insert into child values (2,'c2',2);


SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2    LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  --------
      144  PARENT      TM     64468       0       3         0         0
      145  CHILD       TM     64470       0       3         0         0
      145  PARENT      TM     64468       0       2         0         0
```

16) parent 테이블에 UPDATE 후 child 테이블에 DELETE

```
Session #144
```

```
SQL> update parent set id=4 where id=3;

Session #145

SQL> delete child where id=2;


SQL> @chk_lock
      SID OBJECT_NAM TYPE    ID1     ID2   LMODE   REQUEST    BLOCK
--------- ---------- ----- ------ ------ ------- -------- -------
      144 PARENT     TM    64468      0       3         0       0
      145 CHILD      TM    64470      0       3         0       0
      145 PARENT     TM    64468      0       2         0       0
```

17) parent 테이블에 UPDATE 후 child 테이블에 UPDATE

```
Session #144

SQL> update parent set id=4 where id=3;

Session #145

SQL> update child set id=1 where id=2;


SQL> @chk_lock
      SID OBJECT_NAM TYPE    ID1     ID2   LMODE   REQUEST    BLOCK
--------- ---------- ----- ------ ------ ------- -------- -------
      144 PARENT     TM    64468      0       3         0       0
      145 CHILD      TM    64470      0       3         0       0
```

9~17번까지의 9개의 테스트는 PARENT 테이블에 DML 을 먼저 수행한 후 CHILD 테이블에 DML 을 수행하였을 때 각 테이블에 어떠한 모드로 락을 설정하는지를 확인해보기 위함이었다. 9개의 테스트 모두 TM 락 경합은 발생하지 않는다. 이를 표로 정리하면 다음과 같다.

	OPERATION																	
	PARENT INSERT → CHILD INSERT		PARENT INSERT → CHILD DELETE		PARENT INSERT → CHILD UPDATE		PARENT DELETE → CHILD INSERT		PARENT DELETE → CHILD DELETE		PARENT DELETE → CHILD UPDATE		PARENT UPDATE → CHILD INSERT		PARENT UPDATE → CHILD DELETE		PARENT UPDATE → CHILD UPDATE	
PARENT	RX	RS	RX	RS	RX	RS	RX	RS	RX	RS	RX		RX	RS	RX	RS	RX	
CHILD	RS	RX	RS	RX		RX		RX		RX		RX		RX		RX		RX
TM락 경합여부	No		No		No		No		No		No		No		No		No	

18) child 테이블에 INSERT 후 parent 테이블에 INSERT

```
Session #144

SQL> insert into child values (3,'c3',3);

Session #145

SQL> insert into parent values (5,'p5');


SQL> @chk_lock
      SID OBJECT_NAM TYPE    ID1     ID2   LMODE   REQUEST    BLOCK
--------- ---------- ----- ------ ------ ------- -------- -------
      144 CHILD      TM    64470      0       3         0       0
      144 PARENT     TM    64468      0       2         0       0
      145 PARENT     TM    64468      0       3         0       0
      145 CHILD      TM    64470      0       2         0       0
```

19) child 테이블에 INSERT 후 parent 테이블에 DELETE

```
Session #144

SQL> insert into child values (3,'c3',3);

Session #145

SQL> delete parent where id=4;


SQL> @chk_lock
      SID OBJECT_NAM TYPE    ID1     ID2   LMODE   REQUEST    BLOCK
--------- ---------- ----- ------ ------ ------- -------- -------
      144 CHILD      TM    64470      0       3         0       1
      144 PARENT     TM    64468      0       2         0       0
      145 PARENT     TM    64468      0       3         0       0
      145 CHILD      TM    64470      0       0         4       0
```

20) child 테이블에 INSERT 후 parent 테이블에 UPDATE

```
Session #144

SQL> insert into child values (3,'c3',3);

Session #145

SQL> update parent set id=5 where id=4;
```

```
SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2   LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  -------
      144  CHILD       TM     64470       0       3         0        1
      144  PARENT      TM     64468       0       2         0        0
      145  PARENT      TM     64468       0       3         0        0
      145  CHILD       TM     64470       0       0         4        0
```

21) child 테이블에 DELETE 후 parent 테이블에 INSERT

```
Session #144

SQL> delete child where id=2;

Session #145

SQL> insert into parent values (5,'p5');


SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2   LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  -------
      144  CHILD       TM     64470       0       3         0        0
      144  PARENT      TM     64468       0       2         0        0
      145  PARENT      TM     64468       0       3         0        0
      145  CHILD       TM     64470       0       2         0        0
```

22) child 테이블에 DELETE 후 parent 테이블에 DELETE

```
Session #144

SQL> delete child where id=2;

Session #145

SQL> delete parent where id=4;


SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2   LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  -------
      144  CHILD       TM     64470       0       3         0        1
      144  PARENT      TM     64468       0       2         0        0
      145  PARENT      TM     64468       0       3         0        0
      145  CHILD       TM     64470       0       0         4        0
```

23) child 테이블에 DELETE 후 parent 테이블에 UPDATE

```
Session #144

SQL> delete child where id=2;

Session #145

SQL> update parent set id=5 where id=4;


SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2   LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  -------
      144  CHILD       TM     64470       0       3         0        1
      144  PARENT      TM     64468       0       2         0        0
      145  PARENT      TM     64468       0       3         0        0
      145  CHILD       TM     64470       0       0         4        0
```

24) child 테이블에 UPDATE 후 parent 테이블에 INSERT

```
Session #144

SQL> update child set id=1 where id=2;

Session #145

SQL> insert into parent values (5,'p5');


SQL> @chk_lock

      SID  OBJECT_NAM  TYPE    ID1     ID2   LMODE   REQUEST    BLOCK
---------  ----------  -----  -----  ------  ------  --------  -------
      144  CHILD       TM     64470       0       3         0        0
      145  PARENT      TM     64468       0       3         0        0
      145  CHILD       TM     64470       0       2         0        0
```

25) child 테이블에 UPDATE 후 parent 테이블에 DELETE

```
Session #144

SQL> update child set id=1 where id=2;

Session #145

SQL> delete parent where id=4;


SQL> @chk_lock
```

```
       SID OBJECT_NAM TYPE    ID1     ID2    LMODE  REQUEST    BLOCK
---------- ---------- ----- ------ ------ -------- -------- -------
       144 CHILD      TM    64470      0        3        0       1
       145 PARENT     TM    64468      0        3        0       0
       145 CHILD      TM    64470      0        0        4       0
```

26) child 테이블에 UPDATE 후 parent 테이블에 UPDATE

```
Session #144
SQL> update child set id=1 where id=2;
Session #145
SQL> update parent set id=5 where id=4;

SQL> @chk_lock
       SID OBJECT_NAM TYPE    ID1     ID2    LMODE  REQUEST    BLOCK
---------- ---------- ----- ------ ------ -------- -------- -------
       144 CHILD      TM    64470      0        3        0       1
       145 PARENT     TM    64468      0        3        0       0
       145 CHILD      TM    64470      0        0        4       0
-- 테스트 완료
```

18~26번까지의 9개의 테스트는 CHILD 테이블에 DML 을 먼저 수행한 후 PARENT 테이블에 DML 을 수행하였을 때 각 테이블에 어떠한 모드로 락을 설정하는지를 확인해보기 위함이었다. 9개의 테스트 중 일부는 TM 락 경합이 발생하고, 일부는 TM 락 경합이 발생하지 않는다. 이를 표로 정리하면 다음과 같다.

	OPERATION																	
	CHILD INSERT → PARENT INSERT		CHILD INSERT → PARENT DELETE		CHILD INSERT → PARENT UPDATE		CHILD DELETE → PARENT INSERT		CHILD DELETE → PARENT DELETE		CHILD DELETE → PARENT UPDATE		CHILD UPDATE → PARENT INSERT		CHILD UPDATE → PARENT DELETE		CHILD UPDATE → PARENT UPDATE	
CHILD	RX	RS	RX	S	RX	S	RX	RS	RX	RS	RX	S	RX	RS	RX	S	RX	S
PARENT	RS	RX	RS	RX	RS	RX	RS	RX	RS	RX	RS	RX		RX		RX		RX
TM락 경합여부	N		Y		Y		N		Y		Y		N		Y		Y	

6. Foreign key를 이용한 Converters 리스트 테스트

앞선 테스트를 통해 parent 와 child 테이블에 대한 DML 순서에 따른 TM Lock 경합 발생 여부에 대해 살펴보았다. 이번 섹션에서는 Converters 리스트가 어떻게 활용되는지 테스트를 통해 알아보자.

```
Session #144
SQL> insert into child value (3,'c3',3);
```

이 시점의 V$LOCK을 조회해보면 child 테이블에 대해서는 RX(3) 모드로 Parent 테이블에 대해서는 RS(2) 모드로 TM 락을 소유하게 된다.

```
SQL> @chk_lock
       SID OBJECT_NAM TYPE    ID1     ID2    LMODE  REQUEST    BLOCK
---------- ---------- ----- ------ ------ -------- -------- -------
       144 PARENT     TM    64468      0        2        0       0
       144 CHILD      TM    64470      0        3        0       0

Session #145
SQL> insert into child value (3,'c3',3);
```

이 시점의 V$LOCK을 조회해보면 144, 145 세션 각각 child 테이블에 대해서는 RX(3) mode로 Parent 테이블에 대해서는 RS(2) 모드로 TM 락을 소유하게 된다.

```
SQL> @chk_lock
       SID OBJECT_NAM TYPE    ID1     ID2    LMODE  REQUEST    BLOCK
---------- ---------- ----- ------ ------ -------- -------- -------
       144 CHILD      TM    64470      0        3        0       0
       144 PARENT     TM    64468      0        2        0       0
       145 PARENT     TM    64468      0        2        0       0
       145 CHILD      TM    64470      0        3        0       0

Session #144
SQL> delete parent where id=4;
```

이때, 144 세션에서 parent 테이블에 대한 DELETE 작업을 하게 될 경우, 144 세션은 child 테이블에 대해 S(4) 모드의 TM 락이 필요하게 된다. 그런데 이미 144 세션은 child 테이블에 대해 RX(3) 모드의 TM 락을 소유하고 있다. 이러한 경우 144 세션은 기존의 RX(3) 모드의 TM 락을 릴리즈한 후에 다시 S(4) 모드로 TM 락을 요청할 수 없으므로, Lock conversion을 수행하게 된다.

즉, 기존의 RX+S=SRX(5) 모드의 TM락을 획득하려고 하는 것이다. 아래의 V$LOCK 뷰의 결과를 확인해보자. 이렇게 lock conversion이 필요한 경우에는 WAT(ers) 리스트가 아닌 CON(verters) 리스트에서 대기 세션을 관리하게 된다.

```
SQL> @chk_lock
```

SID	OBJECT_NAM	TYPE	ID1	ID2	LMODE	REQUEST	BLOCK
144	CHILD	TM	64470	0	3	5	0
144	PARENT	TM	64468	0	3	0	0
145	PARENT	TM	64468	0	2	0	0
145	CHILD	TM	64470	0	3	0	1

요약

enqueue internal을 완벽히 이해하기 위해서는 더 많은 내용들에 대한 학습이 필요할 것이다. 하지만, 앞서 다루었던, enqueue 리소스, enqueue 락, enqueue 아키텍처에 대한 이해가 가장 먼저 선행되어야 할 것이라는 생각이 든다. 이번 기사에서 다루지 못한 enqueue 리소스 및 enqueue 락 획득/반납 처리 절차 및 TX 락 처리 절차에 대해서는 현재 기획중인 'Oracle Internals in 10g' 를 통해 기술하도록 하겠다.

참고 문헌

1. "OWI를 활용한 오라클 진단 & 튜닝" "㈜엑셈 역"

Appendix

1) v$lock 뷰 정의

```
-----------------------------------------------------
-- GV$LOCK 정의
-----------------------------------------------------
SELECT s.inst_id ,
       l.laddr                 "ADDR",
       l.kaddr                 "KADDR",
       s.ksusenum     "SID",
       r.ksqrsidt     "TYPE",
       r.ksqrsid1     "ID1",
       r.ksqrsid2     "ID2",
       l.lmode                 "LMODE",
       l.request      "REQUEST",
       l.ctime ,      "CTIME",
       decode( l.lmode , 0 , 0 , l.block ) "BLOCK"
FROM   v$_lock l ,
       x$ksuse s ,
       x$ksqrs r
WHERE  l.saddr=s.addr
AND    l.raddr=r.addr

-----------------------------------------------------
-- GV$_LOCK 정의
-----------------------------------------------------
SELECT USERENV( 'Instance' ) ,
       laddr ,
       kaddr ,
       saddr ,
       raddr ,
       lmode ,
       request ,
       ctime ,
       BLOCK
FROM   v$_lock1
UNION  ALL
SELECT inst_id ,
       addr ,
       ksqlkadr ,
       ksqlkses ,
       ksqlkres ,
       ksqlkmod ,
       ksqlkreq ,
       ksqlkctim ,
       ksqlklblk
FROM   x$ktadm
WHERE  bitand( kssobflg , 1 ) !=0
AND  ( ksqlkmod!=0  OR ksqlkreq!=0 )
UNION  ALL
```

```
SELECT inst_id ,
      addr ,
      ksqlkadr ,
      ksqlkses ,
      ksqlkres ,
      ksqlkmod ,
      ksqlkreq ,
      ksqlkctim ,
      ksqlklblk
FROM   x$ktatrfil
WHERE  bitand( kssobflg , 1 ) !=0
AND  ( ksqlkmod!=0 OR ksqlkreq!=0 )
UNION  ALL
SELECT inst_id ,
      addr ,
      ksqlkadr ,
      ksqlkses ,
      ksqlkres ,
      ksqlkmod ,
      ksqlkreq ,
      ksqlkctim ,
      ksqlklblk
FROM   x$ktatrfsl
WHERE  bitand( kssobflg , 1 ) !=0
AND  ( ksqlkmod!=0 OR ksqlkreq!=0 )
UNION  ALL
SELECT inst_id ,
      addr ,
      ksqlkadr ,
      ksqlkses ,
      ksqlkres ,
      ksqlkmod ,
      ksqlkreq ,
      ksqlkctim ,
ksqlklblk
FROM   x$ktatl
WHERE  bitand( kssobflg , 1 ) !=0
AND  ( ksqlkmod!=0 OR ksqlkreq!=0 )
UNION  ALL
SELECT inst_id ,
      addr ,
      ksqlkadr ,
      ksqlkses ,
      ksqlkres ,
      ksqlkmod ,
      ksqlkreq ,
      ksqlkctim ,
      ksqlklblk
FROM   x$ktstusc
WHERE  bitand( kssobflg , 1 ) !=0
AND  ( ksqlkmod!=0 OR ksqlkreq!=0 )
UNION  ALL
SELECT inst_id ,
      addr ,
      ksqlkadr ,
      ksqlkses ,
      ksqlkres ,
      ksqlkmod ,
      ksqlkreq ,
      ksqlkctim ,
      ksqlklblk
FROM   x$ktstuss
WHERE  bitand( kssobflg , 1 ) !=0
AND  ( ksqlkmod!=0 OR ksqlkreq!=0 )
UNION  ALL
SELECT inst_id ,
      addr ,
      ksqlkadr ,
      ksqlkses ,
      ksqlkres ,
      ksqlkmod ,
      ksqlkreq ,
      ksqlkctim ,
      ksqlklblk
FROM   x$ktstusg
WHERE  bitand( kssobflg , 1 ) !=0
AND  ( ksqlkmod!=0 OR ksqlkreq!=0 )
UNION  ALL
SELECT inst_id ,
      ktcxbxba ,
      ktcxblkp ,
      ksqlkses ,
      ksqlkres ,
      ksqlkmod ,
      ksqlkreq ,
      ksqlkctim ,
      ksqlklblk
FROM   x$ktcxb
WHERE  bitand( ksspaflg , 1 ) !=0
AND  ( ksqlkmod!=0 OR ksqlkreq!=0 )
```

2) chk_lock.sql

```
select  a.sid,  b.object_name,  a.type,  a.id1,  a.id2,  a.lmode,
a.request, a.block
from   v$lock a, dba_objects b
where  a.sid in (&sid.....)
and    a.type='TM'
and    a.id1=b.object_id(+)
order by sid
```