oracle 中锁的概念十分重要，对于其他关系型数据库更不用说，有的数据库发出 select 语句都要加锁，但是 oracle 有独立的 undo（可从 undo 里读取修改过的数据）所以不加锁。

下面介绍 oracle 里的几种锁模式：

| 锁代码 | 锁模式名称 | 锁模式缩写 | 锁模式别名 |
|---|---|---|---|
| 0 | none | none | none |
| 1 | null | null | null |
| 2 | ROW-S | SS | RS |
| 3 | ROW-X | SX | RX |
| 4 | Share | S | S |
| 5 | S/ROW-X | SSX | SRX |
| 6 | Exclusive | X | X |

锁代码说明：

0：none

1：null 空

2：Row-S 行共享(RS)：共享表锁 (row share)

3：Row-X 行专用(RX)：用于行的修改 (row exclusive)

4：Share 共享锁(S)：阻止其他 DML 操作(share)

5：S/Row-X 共享行专用(SRX)：阻止其他事务操作 (share row exclusive)

6：exclusive 专用(X)：排它锁，最高级锁，独立访问使用 (exclusive)

（数字越大锁级别越高，影响的操作越多）

TX 表示的是行级锁，TM 表示的是表级锁。

下面是关于 summary of table locks:

| Sql Statement | Mode of table lock | RS | RX | S |
|---|---|---|---|---|
| select * from table .... | none | Y | Y | Y |
| insert into table .... | RX | Y | Y | N |
| update table .... | RX | Y* | Y* | N |
| delete from table .... | RX | Y* | Y* | N |
| select ... from table for update of | RS | Y* | Y* | Y* |
| lock table table in row share mode | RS | Y | Y | Y |
| lock table table in row execlusive | RX | Y | Y | N |
| lock table table in share mode | S | Y | N | Y |

| | | | | |
|---|---|---|---|---|
| lock table table in share row execlusive mode | SRX | Y | N | N |
| lock table table in execlusive mode | X | N | N | N |

Y(YES)  N(NO)

Y*:if no conficting row locks are held by another transaction.otherwise,waits occur.(如果没有冲突行锁是被另一个事务持有。否则,等待出现)

对于锁的概念理解很重要，尤其在写应用程序的时候。

-bash-3.2$ lsb_release -a

Description: Enterprise Linux Enterprise Linux Server release 5.5 (Carthage)

Release: 5.5

Codename: Carthage

SQL> select * from v$version where rownum=1;

BANNER

---

Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production

SQL> show user;

USER 为 "HR"

SQL> create table t(id number,name varchar2(10));

表已创建。

SQL> insert into t values(1,'diy');

已创建 1 行。

SQL> insert into t values(2,'os');

已创建 1 行。

SQL> COMMIT;

提交完成。

block（阻塞）：

SQL> update t set name='d' where id=1;

已更新 1 行。

SQL> select distinct sid from v$mystat;

SID

---

21

在另一个 session 里：

SQL> select distinct sid from v$mystat;

SID

---

19

SQL> update t set name='o' where id=1;

此时会一直等待，知道第一个事务结束（commit 或 rollback),这是 block，不是死锁！

我们分析这个过程：

SQL> show user;

USER 为 "SYS"

SQL> select * from v$lock where sid in(19,21) order by sid;

ADDR KADDR SID TY ID1 ID2 LMODE REQUEST CTIME BLOCK

38687638 38687664 19 AE 100 0 4 0 810 0

38687ED4 38687F00 19 TX 262175 1888 0 6 230 0

004CA6B0 004CA6E0 19 TM 76714 0 3 0 230 0

004CA6B0 004CA6E0 21 TM 76714 0 3 0 240 0

37FBACC0 37FBAD00 21 TX 262175 1888 6 0 240 1（锁定了一个事务）

38687720 3868774C 21 AE 100 0 4 0 3730 0

已选择 6 行。

注意 AE：Edition Lock，是 11g 新增加的锁类型，这是一个会话锁，只要有会话就会有一个锁。

此时 session 号为 21 的先更新 id=1 这一行，获得了一个 TM（RX)锁，又获得了 TX(X)锁；

session 号为 22 的也获得了一个 TM 锁(RX)，但是和上面的 TM 兼容，所以此时没有阻塞，

但是由于行锁并不和上面的行锁兼容，所以没有获得行锁 X,从上面的 LMODE 可以看出。

我们可以通过下面两个视图分析数据库中被锁的对象：

SQL> select * from v$locked_object;

XIDUSN XIDSLOT XIDSQN OBJECT_ID SESSION_ID

ORACLE_USERNAME OS_USER_NAME PROCESS LOCKED_MODE

0 0 0 76714 19 HR oracle 6130 3

4 31 1888 76714 21 HR oracle 5022 3

SQL> SELECT * FROM DBA_OBJECTS WHERE OBJECT_ID=76714;

OWNER OBJECT_NAME SUBOBJECT_NAME OBJECT_ID DATA_OBJECT_ID

OBJECT_TYPE CREATED LAST_DDL_TIME TIMESTAMP STATUS T G S NAMESPACE EDITION_NAME

---

---

HR T 76714 76714 TABLE 22-4 月 -15 22-4 月

-15 2015-04-22:12:39:06 VALID N N N 1

我们在重建索引时，为了不影响系统性能，往往：

alter index index_name rebuild online;

但是我们为什么不：alter index index_name rebuild

下面简单操作示范：

SQL> create table ttt as select * from dba_objects;

表已创建。

SQL> select count(*) from dba_objects;

COUNT(*)

---

72746

SQL> create index index_id on TTT(OBJECT_ID);

索引已创建。

SQL> set autotrace traceonly;

SQL> SELECT * FROM TTT;

已选择 72746 行。

执行计划

---

Plan hash value: 774701505

```
--------------------------------------------------------------------------------
| Id  | Operation   | Name | Rows  | Bytes | Cost (%CPU)| Time |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  | | 51569 |    10M|   283   (1)| 00:00:04 |
|   1 |  TABLE ACCESS FULL| TTT  | 51569 |    10M|   283   (1)| 00:00:04 |
--------------------------------------------------------------------------------

Note
-----
   - dynamic sampling used for this statement (level=2)
```

统计信息

```
----------------------------------------------------------
308  recursive calls
 0  db block gets
       5909  consistent gets
       1035  physical reads
 0  redo size
    8067725  bytes sent via SQL*Net to client
      53755  bytes received via SQL*Net from client
       4851  SQL*Net roundtrips to/from client
 0  sorts (memory)
 0  sorts (disk)
      72746  rows processed

SQL> alter index index_id rebuild;
```

索引已更改。

```
SQL> SELECT * FROM TTT;
```

已选择 72746 行。

执行计划

---

Plan hash value: 774701505

---

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

---

| 0 | SELECT STATEMENT | | 51569 | 10M| 283 (1)| 00:00:04 |

| 1 | TABLE ACCESS FULL| TTT | 51569 | 10M| 283 (1)| 00:00:04 |

---

Note

---

- dynamic sampling used for this statement (level=2)

统计信息

---

36 recursive calls

0 db block gets

5886 consistent gets

0 physical reads

0 redo size

8067725 bytes sent via SQL*Net to client

53755 bytes received via SQL*Net from client

4851 SQL*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

72746 rows processed

SQL> alter index index_id rebuild online;

索引已更改。

SQL> SELECT * FROM TTT;

已选择 72746 行。

执行计划

---

Plan hash value: 774701505

---

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

---

| 0 | SELECT STATEMENT | | 51569 | 10M| 283 (1)| 00:00:04 |

| 1 | TABLE ACCESS FULL| TTT | 51569 | 10M| 283 (1)| 00:00:04 |

---

Note

---

- dynamic sampling used for this statement (level=2)

统计信息

---

5 recursive calls

0 db block gets

5879 consistent gets

0 physical reads

0 redo size

8067725 bytes sent via SQL*Net to client

53755 bytes received via SQL*Net from client

4851 SQL*Net roundtrips to/from client

0 sorts (memory)

0 sorts (disk)

72746 rows processed

两种扫描方式都是全表扫描，都会发生排序（sort 操作）但是 rebulid online 操作比 rebulid 性能更好，从逻辑读次数可知。

rebulid 操作会阻塞 dml 操作，而 online 操作不会（online 操作降低了锁级别）

deadlock:

```
SQL>show user;
USER 为 "HR"
SQL> select * from t;
ID NAME
---------- ----------
1 diy
2 os
SQL> SELECT DISTINCT SID FROM V$MYSTAT;
SID
----------
21
SQL> update t set name='d' where id=1;
已更新 1 行。
```

```
SQL> SHOW USER;
USER 为 "HR"
SQL> select * from t;
ID NAME
---------- ----------
1 diy
2 os
SQL> SELECT DISTINCT SID FROM V$MYSTAT;
SID
----------
19
SQL> update t set name='s' where id=2;
已更新 1 行。
```

```
SQL> SHOW USER;
USER 为 "HR"
SQL> select distinct sid from v$mystat;
  SID
----------
21
SQL> update t set name='y' where id=2;
```

update t set name='y' where id=2

*

第 1 行出现错误:

ORA-00060: 等待资源时检测到死锁

SQL> show user;

USER 为 "HR"

SQL> select distinct sid from v$mystat;

SID

---

19

SQL> update t set name='s' where id=1;

上述顺序按操作顺序排列。

告警日志里（alert)有警告：

Wed Apr 22 14:43:05 2015

ORA-00060: Deadlock detected. More info in file /u01/app/oracle/diag/rdbms/orcl3939/orcl3939/trace/orcl3939_ora_5024.trc.

查看转储文件：

.....

.....

*** 2015-04-22 14:43:04.053

DEADLOCK DETECTED ( ORA-00060 )

[Transaction Deadlock]

The following deadlock is not an ORACLE error. It is a

deadlock due to user error in the design of an application

or from issuing incorrect ad-hoc SQL. The following

information may aid in determining the deadlock:

Deadlock graph:

| | ---------Blocker(s)-------- | | | | ---------Waiter(s)--------- | | | |
|---|---|---|---|---|---|---|---|---|
| Resource Name | process | session | holds | waits | process | session | holds | waits |
| TX-00030014-000009b7 | 24 | 21 | X | | 32 | 19 | | X |
| TX-0006000d-00000a27 | 32 | 19 | X | | 24 | 21 | | X |

session 21: DID 0001-0018-00000019 session 19: DID 0001-0020-00000014

session 19: DID 0001-0020-00000014 session 21: DID 0001-0018-00000019

Rows waited on:

  Session 21: obj - rowid = 00012BAA - AAASuqAAEAAABuvAAB

.....

.....

上面的内容是不是太详细了！哪个 session，rowid 都告诉我们了！可以更加深入研究死锁。