# oracle死锁详解

## <http://blog.itpub.net/26736162/viewspace-2127247/>(本文来源于itpub)

## 1  ****死锁简介****

### 1.1  ****什么是死锁？****

所谓死锁，是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

Oracle对于“死锁”是要做处理的，而不是不闻不问。

|  |
| --- |
| [ZFLHRDB1:oracle]:/oracle>oerr ora 6000060, 00000, "deadlock detected while waiting for resource"  // \*Cause:  Transactions deadlocked one another while waiting for resources.  // \*Action: Look at the trace file to see the transactions and resources  //          involved. Retry if necessary. |

Cause: Your session and another session are waiting for are source locked by the other. This condition is known AS a deadlock. To resolve the deadlock, one or more statements were rolled back for the other session to continue work.

Action Either: l. Enter arollback statement and re—execute all statements since the last commit or 2. Wait until the lock is released, possibly a few minutes, and then re—execute the rolled back statements.

### 1.2  ****死锁的trace文件****

Oracle中产生死锁的时候会在alert告警日志文件中记录死锁的相关信息，无论单机还是RAC环境都有Deadlock这个关键词，而且当发生死锁时都会生成一个trace文件，这个文件名在alert文件中都有记载。由于在RAC环境中，是由LMD（Lock Manager Daemon）进程统一管理各个节点之间的锁资源的，所以，RAC环境中trace文件是由LMD进程来生成的。

在RAC环境中，告警日志的形式如下所示：

|  |
| --- |
| Mon Jun 20 10:10:56 2016  Global Enqueue Services **Deadlock detected**. More info in file  /u01/app/oracle/diag/rdbms/raclhr/raclhr2/trace/raclhr2\_lmd0\_19923170.trc. |

在单机环境中，告警日志的形式如下所示：

|  |
| --- |
| Mon Jun 20 12:10:56 2016  ORA-00060: **Deadlock detected**. More info in file/u01/app/oracle/diag/rdbms/dlhr/dlhr/trace/dlhr\_ora\_16973880.trc. |

通常来讲，对于单机环境，当有死锁发生后，在trace文件中会看到如下的日志信息：

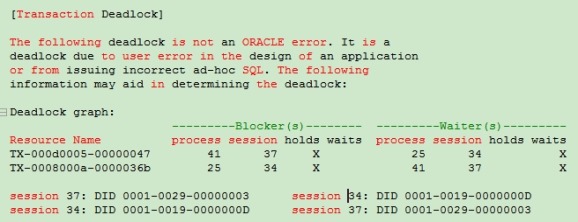


                                    图2-1 单机环境下的死锁

当看到trace文件时，需要确认一下产生锁的类型，是两行还是一行，是TX还是TM，如果只有一行那么说明是同一个SESSION，可能是自治事务引起的死锁。

对于RAC环境，当有死锁发生后，在trace文件中会看到如下的日志信息：

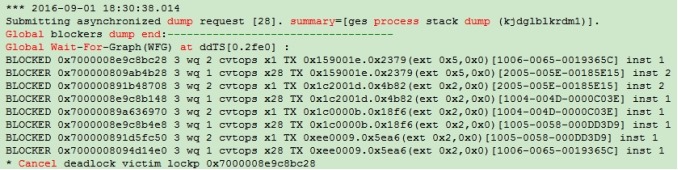


                                       图2-2 RAC环境下的死锁

### 1.3  ****死锁的检测时间****

死锁的监测时间是由隐含参数\_lm\_dd\_interval来控制的，在Oracle 11g中，隐含参数\_lm\_dd\_interval的值默认为10，而在Oracle 10g中该参数默认为60，单位为秒。

|  |
| --- |
| SYS@oraLHRDB2> SELECT A.INDX,    2         A.KSPPINM NAME,    3         A.KSPPDESC,    4         B.KSPPSTVL    5  FROM   X$KSPPI  A,    6         X$KSPPCV B    7  WHERE  A.INDX = B.INDX    8  AND LOWER(A.KSPPINM) LIKE  LOWER('%&PARAMETER%');  ENTER VALUE FOR PARAMETER: **\_lm\_dd\_interval**  OLD   8: AND LOWER(A.KSPPINM) LIKE  LOWER('%&PARAMETER%')  NEW   8: AND LOWER(A.KSPPINM) LIKE  LOWER('%\_LM\_DD\_INTERVAL%')          INDX NAME               KSPPDESC                       KSPPSTVL  ---------- ------------------ ------------------------------ --------------------         578 \_lm\_dd\_interval    dd time interval in seconds    10 |

可以看到该隐含参数的值为10。

## 2  ****死锁的分类****

有人的地方就有江湖，有资源阻塞的地方就可能有死锁。Oralce中最常见的死锁分为：行级死锁（Row-Level Deadlock）和块级死锁（Block-Level Deadlock），其中，行级死锁分为①主键、唯一索引的死锁（会话交叉插入相同的主键值），②外键未加索引，③表上的位图索引遭到并发更新，④常见事务引发的死锁（例如，两个表之间不同顺序相互更新操作引起的死锁；同一张表删除和更新之间引起的死锁），⑤自治事务引发的死锁。块级死锁主要指的是ITL（Interested Transaction List）死锁。

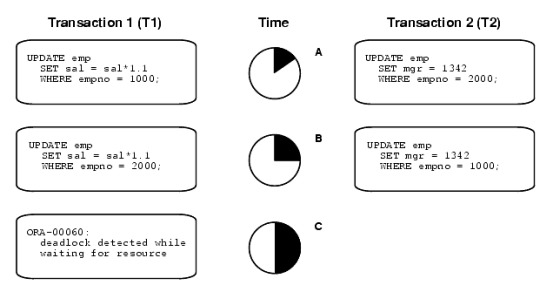
死锁分类图如下所示：



图2-3 死锁的分类图

### 2.1  ****行级死锁****

行级锁的发生如下图所示，在A时间，TRANSACRION1和TRANSCTION2分别锁住了它们要UPDATE的一行数据，没有任何问题。但每个TRANSACTION都没有终止。接下来在B时间，它们又试图UPDATE当前正被对方TRANSACTION锁住的行，因此双方都无法获得资源，此时就出现了死锁。之所以称之为死锁，是因为无论每个TRANSACTION等待多久，这种锁都不会被释放。



行级锁的死锁一般是由于应用逻辑设计的问题造成的，其解决方法是通过分析trace文件定位出造成死锁的SQL语句、被互相锁住资源的对象及其记录等信息，提供给应用开发人员进行分析，并修改特定或一系列表的更新（UPDATE）顺序。

以下模拟各种行级死锁的产生过程，版本都是11.2.0.4。

#### 2.1.1  ****主键、唯一索引的死锁（会话交叉插入相同的主键值）****

**主键的死锁其本质是唯一索引引起的死锁**，这个很容易模拟出来的，新建一张表，设置主键（或创建唯一索引）后插入一个值，然后不要COMMIT，另一个会话插入另一个值，也不要COMMIT，然后再把这两个插入的值互相交换一下，在两个会话中分别插入，死锁就会产生。

会话1，sid为156：

|  |
| --- |
| SYS@lhrdb> set sqlprompt "\_user'@'\_connect\_identifier S1> "  SYS@lhrdb S1> DROP TABLE T\_DEADLOCK\_PRIMARY\_LHR;    Table dropped.    ====>>>>> CREATE UNIQUE INDEX IDX\_TEST\_LHR ON T\_DEADLOCK\_PRIMARY\_LHR(ID);  SYS@lhrdb S1> CREATE TABLE T\_DEADLOCK\_PRIMARY\_LHR(ID NUMBER PRIMARY KEY);    Table created.    SYS@lhrdb S1> select userenv('sid') from dual;    USERENV('SID')  --------------             156    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(1);    1 row created.    SYS@lhrdb S1> |

会话2，sid为156：

|  |
| --- |
| SYS@lhrdb> set sqlprompt "\_user'@'\_connect\_identifier S2> "  SYS@lhrdb S2> select userenv('sid') from dual;    USERENV('SID')  --------------             191    SYS@lhrdb S2> INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(2);    1 row created.    SYS@lhrdb S2> INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(1);        ====>>>>> 产生了阻塞 |

SELECT NVL**(**A.SQL\_ID**,** A.PREV\_SQL\_ID**)** SQL\_ID**,**

       A.BLOCKING\_SESSION**,**

       A.SID**,**

       A.SERIAL#**,**

       A.LOGON\_TIME**,**

       A.EVENT

  FROM GV$SESSION A

 WHERE A.SID IN **(156,191)**

 ORDER BY A.LOGON\_TIME**;**

156阻塞了191会话，即会话1阻塞了会话2。

会话1再次插入数据：

|  |
| --- |
| SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(2);          ====>>>>> 产生了阻塞 |

此时，去会话2看的时候，已经报出了死锁的错误：

此时的阻塞已经发生了变化：

告警日志：

|  |
| --- |
| Fri Sep 23 09:03:11 2016  ORA-00060: Deadlock detected. More info in file /oracle/app/oracle/diag/rdbms/lhrdb/lhrdb/trace/lhrdb\_ora\_25886852.trc. |

其内容可以看到很经典的一段：

|  |
| --- |
| Deadlock graph:                         ---------Blocker(s)--------  ---------Waiter(s)---------  Resource Name          process session holds waits  process session holds waits  TX-0008000c-000008dc        38     191     X             29     156           S  TX-00030016-00000892        29     156     X             38     191           S    session 191: DID 0001-0026-00000115     session 156: DID 0001-001D-0000000D  session 156: DID 0001-001D-0000000D     session 191: DID 0001-0026-00000115    Rows waited on:    Session 191: no row    Session 156: no row |

这就是主键的死锁，模拟完毕。

此时，若是会话2执行提交后，会话1就会报错，违反唯一约束：

|  |
| --- |
| SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(2);      INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(2)  \*  ERROR at line 1:  ORA-00001: unique constraint (SYS.SYS\_C0011517) violated |

一、 **脚本**

|  |
| --- |
| 会话1：  DROP TABLE T\_DEADLOCK\_PRIMARY\_LHR;  CREATE TABLE T\_DEADLOCK\_PRIMARY\_LHR(ID NUMBER PRIMARY KEY);  --CREATE UNIQUE INDEX IDX\_TEST\_LHR ON T\_DEADLOCK\_PRIMARY\_LHR(ID);  select userenv('sid') from dual;  INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(1);    会话2：  select userenv('sid') from dual;  INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(2);  INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(1);      会话1：---死锁产生  INSERT INTO T\_DEADLOCK\_PRIMARY\_LHR VALUES(2);      SELECT NVL(A.SQL\_ID, A.PREV\_SQL\_ID) SQL\_ID,         A.BLOCKING\_SESSION,         A.SID,         A.SERIAL#,         A.LOGON\_TIME,         A.EVENT    FROM GV$SESSION A  WHERE A.SID IN (156,191)  ORDER BY A.LOGON\_TIME; |

#### 2.1.2  ****外键的死锁（外键未加索引）****

外键未加索引很容易导致死锁。在以下两种情况下，Oracle在修改父表后会对子表加一个全表锁：

1. 如果更新了父表的主键，由于外键上没有索引，所以子表会被锁住。

2. 如果删除了父表中的一行，由于外键上没有索引，整个子表也会被锁住。

总之，就是**更新或者删除父表的主键**，都会导致对其子表加一个全表锁。

如果父表存在删除记录或者更改外键列的情形，那么就需要在子表上为外键列创建索引。

除了全表锁外，在以下情况下，未加索引的外键也可能带来问题：

1. 如果有ON DELETE CASCADE，而且没有对子表加索引：例如，EMP是DEPT的子表，DELETE DEPTNO = 10应该CASCADE（级联）至EMP[4]。如果EMP中的DEPTNO没有索引，那么删除DEPT表中的每一行时都会对EMP做一个全表扫描。这个全表扫描可能是不必要的，而且如果从父表删除多行，父表中每删除一行就要扫描一次子表。

2. 从父表查询子表：再次考虑EMP/DEPT例子。利用DEPTNO查询EMP表是相当常见的。如果频繁地运行以下查询（例如，生成一个报告），你会发现没有索引会使查询速度变慢：

SELECT \* FROM DEPT, EMP

WHERE EMP.DEPTNO = DEPT.DEPTNO

AND DEPT.DEPTNO = :X;

那么，什么时候不需要对外键加索引呢？答案是，一般来说，当满足以下条件时不需要加索引：

     1.  没有从父表删除行。

     2.  没有更新父表的惟一键/主键值。

     3.  没有从父表联结子表。

如果满足上述全部3个条件，那你完全可以跳过索引，不需要对外键加索引。不过个人还是强烈建议对子表添加索引，既然已经创建了外键，就不在乎再多一个索引吧，因为一个索引所增加的代价比如死锁，与缺失这个索引所带来的问题相比，是微不足道的。

子表上为外键列建立索引，可以：

1）提高针对外键列的查询或改动性能

2）减小表级锁粒度，降低死锁发生的可能性

外键的死锁可以这样通俗的理解：有两个表A和B：A是父表，B是子表。如果没有在B表中的外键加上索引，那么A表在更新或者删除主键时，都会在表B上加一个全表锁。这是为什么呢？因为我们没有给外键加索引，在更新或者删除A表主键的时候，需要查看子表B中是否有对应的记录，以判断是否可以更新删除。那如何查找呢？当然只能在子表B中一条一条找了，因为我们没有加索引吗。既然要在子表B中一条一条地找，那就得把整个子表B都锁定了。由此就会导致以上一系列问题。

实验过程：

会话1首先建立子表和父表

|  |
| --- |
| SYS@lhrdb S1> CREATE TABLE T\_DEADLOCK\_P\_LHR (ID NUMBER PRIMARY KEY, NAME VARCHAR2(30)); --父表的主键    Table created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_P\_LHR VALUES (1, 'A');    1 row created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_P\_LHR VALUES (2, 'B');    1 row created.    SYS@lhrdb S1> CREATE TABLE T\_DEADLOCK\_F\_LHR (FID NUMBER, FNAME VARCHAR2(30), FOREIGN KEY (FID) REFERENCES T\_DEADLOCK\_P\_LHR);   --子表的外键    Table created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_F\_LHR VALUES (1, 'C');    1 row created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_F\_LHR VALUES (2, 'D');    1 row created.    SYS@lhrdb S1>  SYS@lhrdb S1> COMMIT;    Commit complete. |

会话1执行一个删除操作，这时候在子表和父表上都加了一个Row-X(SX)锁

|  |
| --- |
| SYS@lhrdb S1> delete from T\_DEADLOCK\_F\_LHR where FID=1;    1 row deleted.    SYS@lhrdb S1> delete from T\_DEADLOCK\_P\_LHR where id=1;    1 row deleted. |

查询会话1的锁信息：

SELECT LK.SID**,**

       DECODE**(**LK.TYPE**,**

              'TX'**,**

              'Transaction'**,**

              'TM'**,**

              'DML'**,**

              'UL'**,**

              'PL/SQL User Lock'**,**

              LK.TYPE**)** LOCK\_TYPE**,**

       DECODE**(**LK.LMODE**,**

**0,**

              'None'**,**

**1,**

              'Null'**,**

**2,**

              'Row-S (SS)'**,**

**3,**

              'Row-X (SX)'**,**

**4,**

              'Share'**,**

**5,**

              'S/Row-X (SSX)'**,**

**6,**

              'Exclusive'**,**

              TO\_CHAR**(**LK.LMODE**))** MODE\_HELD**,**

       DECODE**(**LK.REQUEST**,**

**0,**

              'None'**,**

**1,**

              'Null'**,**

**2,**

              'Row-S (SS)'**,**

**3,**

              'Row-X (SX)'**,**

**4,**

              'Share'**,**

**5,**

              'S/Row-X (SSX)'**,**

**6,**

              'Exclusive'**,**

              TO\_CHAR**(**LK.REQUEST**))** MODE\_REQUESTED**,**

       OB.OBJECT\_TYPE**,**

       OB.OBJECT\_NAME**,**

       LK.BLOCK**,**

       SE.LOCKWAIT

  FROM V$LOCK LK**,** DBA\_OBJECTS OB**,** V$SESSION SE

 WHERE LK.TYPE IN **(**'TM'**,** 'UL'**)**

   AND LK.SID **=** SE.SID

   AND LK.ID1 **=** OB.OBJECT\_ID**(+)**

 AND SE.SID IN **(156,191)**

 ORDER BY SID**;**

BLOCK为0表示没有阻塞其它的锁。

会话2：执行另一个删除操作，发现这时候第二个删除语句等待

|  |
| --- |
| SYS@lhrdb S2> delete from T\_DEADLOCK\_F\_LHR where FID=2;    1 row deleted.    SYS@lhrdb S2> delete from T\_DEADLOCK\_P\_LHR where id=2;          ====>>>>> 产生了阻塞 |

BLOCK为1表示阻塞了其它的锁。

会话1执行删除语句，死锁发生

|  |
| --- |
| SYS@lhrdb S1> delete from T\_DEADLOCK\_P\_LHR where id=1;        ====>>>>> 产生了阻塞，而会话2产生了死锁 |

|  |
| --- |
| SYS@lhrdb S2> delete from T\_DEADLOCK\_P\_LHR where id=2;        delete from T\_DEADLOCK\_P\_LHR where id=2              \*  ERROR at line 1:  ORA-00060: deadlock detected while waiting for resource |

告警日志：

|  |
| --- |
| Fri Sep 23 10:31:10 2016  ORA-00060: Deadlock detected. More info in file /oracle/app/oracle/diag/rdbms/lhrdb/lhrdb/trace/lhrdb\_ora\_25886852.trc. |

查看内容：

|  |
| --- |
| \*\*\* 2016-09-23 10:31:10.212  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  TM-00017731-00000000        38     191    SX   SSX       29     156    SX   SSX  TM-00017731-00000000        29     156    SX   SSX       38     191    SX   SSX    session 191: DID 0001-0026-00000115     session 156: DID 0001-001D-0000000D  session 156: DID 0001-001D-0000000D     session 191: DID 0001-0026-00000115    Rows waited on:    Session 191: no row    Session 156: no row |

回滚会话建立外键列上的索引：

|  |
| --- |
| SYS@lhrdb S1> rollback;    Rollback complete.    SYS@lhrdb S1> create index ind\_p\_f\_fid on T\_DEADLOCK\_F\_LHR(fid);    Index created. |

重复上面的步骤会话1删除子表记录：

---会话1：

|  |
| --- |
| SYS@lhrdb S1> delete from T\_DEADLOCK\_F\_LHR where FID=1;    1 row deleted.    SYS@lhrdb S1> delete from T\_DEADLOCK\_P\_LHR where id=1;    1 row deleted.    SYS@lhrdb S1> |

---会话2：

|  |
| --- |
| SYS@lhrdb S2> delete from T\_DEADLOCK\_F\_LHR where FID=2;    1 row deleted.    SYS@lhrdb S2> delete from T\_DEADLOCK\_P\_LHR where id=2;    1 row deleted. |

所有的删除操作都可以成功执行，也没有阻塞的生成，重点就是在外键列上建立索引。

一、 **脚本**

|  |
| --- |
| --主表  DROP TABLE T\_DEADLOCK\_P\_LHR;  CREATE TABLE T\_DEADLOCK\_P\_LHR (ID NUMBER PRIMARY KEY, NAME VARCHAR2(30)); --父表的主键  INSERT INTO T\_DEADLOCK\_P\_LHR VALUES (1, 'A');  INSERT INTO T\_DEADLOCK\_P\_LHR VALUES (2, 'B');    --子表  DROP TABLE T\_DEADLOCK\_F\_LHR;  CREATE TABLE T\_DEADLOCK\_F\_LHR (FID NUMBER, FNAME VARCHAR2(30), FOREIGN KEY (FID) REFERENCES T\_DEADLOCK\_P\_LHR);  --子表的外键  INSERT INTO T\_DEADLOCK\_F\_LHR VALUES (1, 'C');  INSERT INTO T\_DEADLOCK\_F\_LHR VALUES (2, 'D');    COMMIT;      ---执行一个删除操作，这时候在子表和父表上都加了一个Row-S(SX)锁  delete from T\_DEADLOCK\_F\_LHR where FID=1;  delete from T\_DEADLOCK\_P\_LHR where id=1;  ---会话2：执行另一个删除操作，发现这时候第二个删除语句等待  delete from T\_DEADLOCK\_F\_LHR where FID=2;  delete from T\_DEADLOCK\_P\_LHR where id=2;  ---会话1：死锁发生  delete from T\_DEADLOCK\_P\_LHR where id=1;      ---回滚会话建立外键列上的索引：  create index ind\_p\_f\_fid on T\_DEADLOCK\_F\_LHR(fid);      --重复上面的步骤会话1删除子表记录：  ---会话1：  delete from T\_DEADLOCK\_F\_LHR where FID=1;  delete from T\_DEADLOCK\_P\_LHR where id=1;  ---会话2：执行另一个删除操作，发现这时候第二个删除语句等待  delete from T\_DEADLOCK\_F\_LHR where FID=2;  delete from T\_DEADLOCK\_P\_LHR where id=2; |

#### 2.1.3  ****位图(BITMAP)索引死锁****

表上的位图索引遭到并发更新也很容易产生死锁。在有位图索引存在的表上面，其实很容易就引发阻塞与死锁。这个阻塞不是发生在表上面，而是发生在索引上。因为位图索引锁定的范围远远比普通的b-tree索引锁定的范围大。

|  |
| --- |
| SYS@lhrdb S1> CREATE TABLE T\_DEADLOCK\_BITMAP\_LHR (ID NUMBER , NAME VARCHAR2(10));    Table created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (1,'A');    1 row created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (1,'B');    1 row created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (1,'C');    1 row created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (2,'A');    1 row created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (2,'B');    1 row created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (2,'C');    1 row created.    SYS@lhrdb S1> COMMIT;    Commit complete. |

--那么在ID列上建bitmap index的话，所有ID＝1的会放到一个位图中，所有ID=2的是另外一个位图，而在执行DML操作的时候，锁定的将是整个位图中的所有行，而不仅仅是DML涉及到的行。由于锁定的粒度变粗，bitmap index更容易导致死锁的发生。

会话1：此时所有ID=1的行都被锁定

|  |
| --- |
| SYS@lhrdb S1> CREATE BITMAP INDEX IDX\_DEADLOCK\_BITMAP\_LHR ON T\_DEADLOCK\_BITMAP\_LHR(ID);    Index created.    SYS@lhrdb S1> UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=3 WHERE ID=1 AND NAME='A';    1 row updated. |

会话2：此时所有ID=2的行都被锁定

|  |
| --- |
| SYS@lhrdb S2> UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=4 WHERE ID=2 AND NAME='A';    1 row updated. |

会话1：此时会话被阻塞

|  |
| --- |
| SYS@lhrdb S1> UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=4 WHERE ID=2 AND NAME='B';          ====>>>>> 产生了阻塞 |

会话2：会话被阻塞

|  |
| --- |
| SYS@lhrdb S2> UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=3 WHERE ID=1 AND NAME='B';        ====>>>>> 产生了阻塞 |

再回到SESSION 1，发现系统检测到了死锁的发生

|  |
| --- |
| SYS@lhrdb S1> UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=4 WHERE ID=2 AND NAME='B';  UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=4 WHERE ID=2 AND NAME='B'         \*  ERROR at line 1:  ORA-00060: deadlock detected while waiting for resource |

告警日志：

|  |
| --- |
| Fri Sep 23 11:20:21 2016  ORA-00060: Deadlock detected. More info in file /oracle/app/oracle/diag/rdbms/lhrdb/lhrdb/trace/lhrdb\_ora\_19726460.trc. |

内容：

|  |
| --- |
| \*\*\* 2016-09-23 11:26:51.264  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-0009000e-00000b0f        29     156     X             38     191           S  TX-00070001-00000b2c        38     191     X             29     156           S    session 156: DID 0001-001D-0000000D     session 191: DID 0001-0026-00000115  session 191: DID 0001-0026-00000115     session 156: DID 0001-001D-0000000D    Rows waited on:    Session 156: obj - rowid = 00017734 - AAAXc0AAAAAAAAAAAA    (dictionary objn - 96052, file - 0, block - 0, slot - 0)    Session 191: obj - rowid = 00017734 - AAAXc0AAAAAAAAAAAA    (dictionary objn - 96052, file - 0, block - 0, slot - 0) |

死锁发生的根本原因是对于资源的排他锁定顺序不一致。上面的试验中，session1对于bitmap index中的2个位图是先锁定ID=1的位图，然后请求ID=2的位图，而在此之前ID=2的位图已经被session2锁定。session2则先锁定ID=2的位图，然后请求ID=2的位图，而此前ID=1的位图已经被session1锁定。于是，session1等待session2释放ID=2的位图上的锁，session2等待session1释放ID=1的位图上的锁，死锁就发生了

而如果我们创建的是普通的B\*Tree index，重复上面的试验则不会出现任何的阻塞和死锁，这是因为锁定的只是DML操作涉及到的行，而不是所有ID相同的行。

一、 **脚本**

|  |
| --- |
| CREATE TABLE T\_DEADLOCK\_BITMAP\_LHR (ID NUMBER , NAME VARCHAR2(10));  INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (1,'A');  INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (1,'B');  INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (1,'C');  INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (2,'A');  INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (2,'B');  INSERT INTO T\_DEADLOCK\_BITMAP\_LHR VALUES (2,'C');  COMMIT;    CREATE BITMAP INDEX IDX\_DEADLOCK\_BITMAP\_LHR ON T\_DEADLOCK\_BITMAP\_LHR(ID);    --会话1：此时所有ID=1的行都被锁定  UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=3 WHERE ID=1 AND NAME='A';  --会话2：此时所有ID=2的行都被锁定  UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=4 WHERE ID=2 AND NAME='A';    --会话1：此时会话被阻塞  UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=4 WHERE ID=2 AND NAME='B';    --会话2：会话被阻塞,再回到SESSION 1，发现系统检测到了死锁的发生  UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=3 WHERE ID=1 AND NAME='B';    --会话1：  UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=4 WHERE ID=2 AND NAME='B';  UPDATE T\_DEADLOCK\_BITMAP\_LHR SET ID=4 WHERE ID=2 AND NAME='B'  \*  ERROR at line 1:  ORA-00060: deadlock detected while waiting for resource |

#### 2.1.4  ****常见事务引发的死锁****

如果你有两个会话，每个会话都持有另一个会话想要的资源，此时就会出现死锁（deadlock）。例如，如果我的数据库中有两个表A和B，每个表中都只有一行，就可以很容易地展示什么是死锁。我要做的只是打开两个会话（例如，两个SQL\*Plus会话）。在会话A中更新表A，并在会话B中更新表B。现在，如果我想在会话B中更新表A，就会阻塞。会话A已经锁定了这一行。这不是死锁；只是阻塞而已。因为会话A还有机会提交或回滚，这样会话B就能继续。如果我再回到会话A，试图更新表B，这就会导致一个死锁。要在这两个会话中选择一个作为“牺牲品”，让它的语句回滚。

想要更新表B的会话A还阻塞着，Oracle不会回滚整个事务。只会回滚与死锁有关的某条语句。会话B仍然锁定着表B中的行，而会话A还在耐心地等待这一行可用。收到死锁消息后，会话B必须决定将表B上未执行的工作提交还是回滚，或者继续走另一条路，以后再提交。一旦这个会话执行提交或回滚，另一个阻塞的会话就会继续，好像什么也没有发生过一样。

一、 **模拟一：两个表之间不同顺序相互更新操作引起的死锁**

1、创建两个简单的表A和B，每个表中仅仅包含一个字段id。

|  |
| --- |
| [ZFZHLHRDB2:oracle]:/oracle>ORACLE\_SID=raclhr2  [ZFZHLHRDB2:oracle]:/oracle>sqlplus / as sysdba    SQL\*Plus: Release 11.2.0.4.0 Production on Mon Jun 20 09:40:24 2016    Copyright (c) 1982, 2013, Oracle.  All rights reserved.      Connected to:  Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production  With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,  Data Mining and Real Application Testing options    SYS@raclhr2> select \* from v$version;    BANNER  --------------------------------------------------------------------------------  Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production  PL/SQL Release 11.2.0.4.0 - Production  CORE    11.2.0.4.0      Production  TNS for IBM/AIX RISC System/6000: Version 11.2.0.4.0 - Production  NLSRTL Version 11.2.0.4.0 - Production    SYS@raclhr2> show parameter cluster    NAME                                 TYPE        VALUE  ------------------------------------ ----------- ------------------------------  cluster\_database                     boolean     TRUE  cluster\_database\_instances           integer     2  cluster\_interconnects                string    SYS@raclhr2> create table A (id int);    Table created.    SYS@raclhr2> create table B (id int);    Table created. |

2、每张表中仅初始化一条数据

|  |
| --- |
| SYS@raclhr2> insert into A values (1);    1 row created.    SYS@raclhr2> insert into B values (2);    1 row created.    SYS@raclhr2> commit;    Commit complete. |

3、在第一个会话session1中更新表A中的记录“1”为“10000”，**不提交；**在第二个会话session2中更新表B中的记录“2”为“20000”，**不提交**

session1的情况：

|  |
| --- |
| SYS@raclhr2> insert into A values (1);    1 row created.    SYS@raclhr2> insert into B values (2);    1 row created.    SYS@raclhr2> commit;    Commit complete. |

|  |
| --- |
| SYS@raclhr2> SELECT a.SID,    2         b.SERIAL# ,    3         c.SPID    4  FROM   v$mystat  a,    5         v$session b ,    6         v$process c    7  WHERE  a.SID = b.SID    8  and b.PADDR=c.ADDR    9  AND    rownum = 1;           SID    SERIAL# SPID  ---------- ---------- ------------------------         133          3 20906088    SYS@raclhr2>  update A set id = 10000 where id = 1;    1 row updated. |

session2的情况：

|  |
| --- |
| SYS@raclhr2> SELECT a.SID,    2         b.SERIAL# ,    3         c.SPID    4  FROM   v$mystat  a,    5         v$session b ,    6         v$process c    7  WHERE  a.SID = b.SID    8  and b.PADDR=c.ADDR    9  AND    rownum = 1;           SID    SERIAL# SPID  ---------- ---------- ------------------------         195         21 11010172    SYS@raclhr2> update B set id = 20000 where id = 2;    1 row updated.    SYS@raclhr2> |

4、此时，没有任何问题发生。OK，现在注意一下下面的现象，我们再回到会话session1中，更新表B的记录，此时出现了会话阻塞，更新hang住不能继续。

|  |
| --- |
| SYS@raclhr2> update B set id = 10000 where id = 2; |

这里出现了“锁等待”（“阻塞”）的现象，原因很简单，因为在session2中已经对这条数据执行过update操作没有提交表示已经对该行加了行级锁。

|  |
| --- |
| SYS@raclhr2> set line 9999  SYS@raclhr2> SELECT A.INST\_ID,    2         A.SID,    3         A.SERIAL#,    4         A.SQL\_ID,    5         A.BLOCKING\_INSTANCE,    6         A.BLOCKING\_SESSION,    7         A.EVENT    8    FROM gv$session a    9   WHERE a.USERNAME IS NOT NULL  10     and a.STATUS = 'ACTIVE'  11     and a.BLOCKING\_SESSION IS NOT NULL ;       INST\_ID        SID    SERIAL# SQL\_ID        BLOCKING\_INSTANCE BLOCKING\_SESSION EVENT  ---------- ---------- ---------- ------------- ----------------- ---------------- ------------------------------           2        133          3 6k793mj0duubw                 2              195 enq: TX - row lock contention |

我们可以通过v$session视图看到，实例2的195阻塞了实例2的133会话，即本实验中的session2阻塞了session1。

6、接下来再执行一条SQL后死锁就会产生了：在session2中，更新表A的记录

|  |
| --- |
| SYS@raclhr2> update A set id = 10000 where id = 1; |

这里还是长时间的等待，但是这里发生了死锁，这个时候我们去第一个会话session1中看一下，原先一直在等待的SQL语句报了如下的错误：

|  |
| --- |
| SYS@raclhr2> update B set id = 10000 where id = 2;  update B set id = 10000 where id = 2         \*  ERROR at line 1:  ORA-00060: deadlock detected while waiting for resource |

若此时查询v$session视图可以看到实例2的133阻塞了实例2的195会话，即本实验中的session1阻塞了session2，和刚刚的阻塞情况相反，说明oracle做了自动处理：

|  |
| --- |
| SYS@raclhr2> set line 9999  SYS@raclhr2> SELECT A.INST\_ID,    2         A.SID,    3         A.SERIAL#,    4         A.SQL\_ID,    5         A.BLOCKING\_INSTANCE,    6         A.BLOCKING\_SESSION,    7         A.EVENT    8    FROM gv$session a    9   WHERE a.USERNAME IS NOT NULL  10     and a.STATUS = 'ACTIVE'  11      and a.BLOCKING\_SESSION IS NOT NULL ;       INST\_ID        SID    SERIAL# SQL\_ID        BLOCKING\_INSTANCE BLOCKING\_SESSION EVENT  ---------- ---------- ---------- ------------- ----------------- ---------------- ------------------------------           2        195         21 5q7t3877fdu3n                 2              133 enq: TX - row lock contention |

更进一步：查看一下alert警告日志文件发现有如下的记录：

|  |
| --- |
| Mon Jun 20 10:10:56 2016  Global Enqueue Services Deadlock detected. More info in file  /u01/app/oracle/diag/rdbms/raclhr/raclhr2/trace/raclhr2\_lmd0\_19923170.trc. |

若是单机环境，报警日志为：

|  |
| --- |
| ORA-00060: Deadlock detected. More info in file /oracle/app/oracle/diag/rdbms/dlhr/dlhr/trace/dlhr\_ora\_16973880.trc. |

可以看到该文件是由lmd进程生成的，为rac的特有进程，完成CacheFusion的作用，再进一步：看看系统自动生成的trace文件中记录了什么：

|  |
| --- |
| user session for deadlock lock 0x70001001569c378    sid: 133 ser: 3 audsid: 4294967295 user: 0/SYS      flags: (0x41) USR/- flags\_idl: (0x1) BSY/-/-/-/-/-      flags2: (0x40009) -/-/INC    pid: 60 O/S info: user: oracle, term: UNKNOWN, ospid: 20906088      image: oracle@ZFZHLHRDB2 (TNS V1-V3)    client details:      O/S info: user: oracle, term: pts/0, ospid: 16122014      machine: ZFZHLHRDB2 program: sqlplus@ZFZHLHRDB2 (TNS V1-V3)      application name: sqlplus@ZFZHLHRDB2 (TNS V1-V3), hash value=346557658    current SQL:    update B set id = 10000 where id = 2  DUMP LOCAL BLOCKER: initiate state dump for DEADLOCK    possible owner[60.20906088] on resource TX-00140013-0000072D      user session for deadlock lock 0x700010015138660    sid: 195 ser: 21 audsid: 4294967295 user: 0/SYS      flags: (0x41) USR/- flags\_idl: (0x1) BSY/-/-/-/-/-      flags2: (0x40009) -/-/INC    pid: 46 O/S info: user: oracle, term: UNKNOWN, ospid: 11010172      image: oracle@ZFZHLHRDB2 (TNS V1-V3)    client details:      O/S info: user: oracle, term: pts/1, ospid: 16646154      machine: ZFZHLHRDB2 program: sqlplus@ZFZHLHRDB2 (TNS V1-V3)      application name: sqlplus@ZFZHLHRDB2 (TNS V1-V3), hash value=346557658    current SQL:    update A set id = 10000 where id = 1  DUMP LOCAL BLOCKER: initiate state dump for DEADLOCK    possible owner[46.11010172] on resource TX-000B0018-00000416 |

若是单机环境比较明显：

|  |
| --- |
| [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-000d0005-00000047        41      37     X             25      34           X  TX-0008000a-0000036b        25      34     X             41      37           X    session 37: DID 0001-0029-00000003session 34: DID 0001-0019-0000000D  session 34: DID 0001-0019-0000000Dsession 37: DID 0001-0029-00000003    Rows waited on:    Session 37: obj - rowid = 00015FE7 - AAAV/nAABAAAXeBAAA    (dictionary objn - 90087, file - 1, block - 96129, slot - 0)    Session 34: obj - rowid = 00015FE6 - AAAV/mAABAAAXZ5AAA    (dictionary objn - 90086, file - 1, block - 95865, slot - 0) |

注意trace文件中的一行如下提示信息，说明一般情况下都是应用和人为的，和Oracle同学没有关系：

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.

二、 **模拟二：同一张表删除和更新之间引起的死锁**

造成死锁的原因就是多个线程或进程对同一个资源的争抢或相互依赖。这里列举一个对同一个资源的争抢造成死锁的实例。

|  |
| --- |
| SYS@lhrdb S1> CREATE  TABLE T\_DEADLOCK\_DU\_LHR (    ID  NUMBER,  test  VARCHAR(10) )  ;    Table created.    SYS@lhrdb S1> INSERT  INTO  T\_DEADLOCK\_DU\_LHR  VALUES(1,'test1');    1 row created.    SYS@lhrdb S1> INSERT  INTO  T\_DEADLOCK\_DU\_LHR  VALUES(2,'test2');    1 row created.    SYS@lhrdb S1> COMMIT;    Commit complete.    SYS@lhrdb S1> SELECT \* FROM T\_DEADLOCK\_DU\_LHR;            ID TEST  ---------- ----------           1 test1           2 test2 |

会话1更新第一条记录：

|  |
| --- |
| SYS@lhrdb S1> UPDATE T\_DEADLOCK\_DU\_LHR T SET T.TEST=100 WHERE ID=1;    1 row updated. |

会话2删除第二条记录：

|  |
| --- |
| SYS@lhrdb S2> DELETE FROM  T\_DEADLOCK\_DU\_LHR T  WHERE ID=2;    1 row deleted. |

接下来会话1更新第二条记录，这是就产生了阻塞：

|  |
| --- |
| SYS@lhrdb S1> UPDATE T\_DEADLOCK\_DU\_LHR T SET T.TEST=100 WHERE ID=2;        ====>>>>> 产生了阻塞，会话2阻塞了会话1 |

会话2删除第一条记录：

|  |
| --- |
| SYS@lhrdb S2> DELETE FROM  T\_DEADLOCK\_DU\_LHR T  WHERE ID=1;        ====>>>>> 产生了阻塞，此时会话1产生死锁 |

查看会话1：

|  |
| --- |
| SYS@lhrdb S1> UPDATE T\_DEADLOCK\_DU\_LHR T SET T.TEST=100 WHERE ID=2;            UPDATE T\_DEADLOCK\_DU\_LHR T SET T.TEST=100 WHERE ID=2         \*  ERROR at line 1:  ORA-00060: deadlock detected while waiting for resource      SYS@lhrdb S1> SYS@lhrdb S1> SYS@lhrdb S1> SYS@lhrdb S1> SYS@lhrdb S1> SYS@lhrdb S1> |

告警日志：

|  |
| --- |
| Fri Sep 23 15:10:55 2016  ORA-00060: Deadlock detected. More info in file /oracle/app/oracle/diag/rdbms/lhrdb/lhrdb/trace/lhrdb\_ora\_19726460.trc. |

内容：

|  |
| --- |
| \*\*\* 2016-09-23 15:10:55.326  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-00090008-00000b0c        29     156     X             38     191           X  TX-000a0007-00000d28        38     191     X             29     156           X    session 156: DID 0001-001D-0000000D     session 191: DID 0001-0026-00000115  session 191: DID 0001-0026-00000115     session 156: DID 0001-001D-0000000D    Rows waited on:    Session 156: obj - rowid = 0001773F - AAAXc/AABAABc2RAAB    (dictionary objn - 96063, file - 1, block - 380305, slot - 1)    Session 191: obj - rowid = 0001773F - AAAXc/AABAABc2RAAA    (dictionary objn - 96063, file - 1, block - 380305, slot - 0) |

#### 2.1.5  ****自治事务引发的死锁****

一般来说构成死锁至少需要两个会话，而自治事务是一个会话可能引发死锁。

自治事务死锁情景:存储过程INSERT表A,然后INSERT表B;其中INSERT表A触发TRIGGER T,T也INSERT表B,T是自治事务(AT),AT试图获取对B的锁,结果B已经被主事务所HOLD,这里会报出来ORA-00060 – 等待资源时检查到死锁.

解决方法:去掉了T中的PRAGMA AUTONOMOUS\_TRANSACTION声明,保持和存储过程事务一致.

一、 **模拟一：更新**

在主事务中如果更新了部分记录，这时若自治事务更新同样的记录，就会造成死锁，下面通过一个简单的例子模拟了这个错误的产生：

|  |
| --- |
| SYS@lhrdb S1> CREATE TABLE T\_DEADLOCK\_AT2\_LHR(ID NUMBER, NAME VARCHAR2(30));    Table created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_AT2\_LHR SELECT ROWNUM, TNAME FROM TAB WHERE ROWNUM<=4;    4 rows created.    SYS@lhrdb S1> COMMIT;    Commit complete.    SYS@lhrdb S1> CREATE OR REPLACE PROCEDURE PRO\_TESTAT\_LHR AS    2    PRAGMA AUTONOMOUS\_TRANSACTION;    3  BEGIN    4    UPDATE T\_DEADLOCK\_AT2\_LHR SET NAME = NAME WHERE ID = 1;    5    COMMIT;    6  END;    7  /    Procedure created.      SYS@lhrdb S1> UPDATE T\_DEADLOCK\_AT2\_LHR SET NAME = NAME WHERE ID = 1;    1 row updated.    SYS@lhrdb S1> EXEC PRO\_TESTAT\_LHR;    BEGIN PRO\_TESTAT\_LHR; END;    \*  ERROR at line 1:  ORA-00060: deadlock detected while waiting for resource  ORA-06512: at "SYS.PRO\_TESTAT\_LHR", line 4  ORA-06512: at line 1      SYS@lhrdb S1> SYS@lhrdb S1> |

在使用自治事务的时候要避免当前事务锁定的记录和自治事务中锁定的记录相互冲突。

告警日志：

|  |
| --- |
| Fri Sep 23 14:03:10 2016  ORA-00060: Deadlock detected. More info in file /oracle/app/oracle/diag/rdbms/lhrdb/lhrdb/trace/lhrdb\_ora\_19726460.trc. |

内容：

|  |
| --- |
| \*\*\* 2016-09-23 14:10:34.974  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-000a0015-00000d25        38     191     X             38     191           X    session 191: DID 0001-0026-00000115     session 191: DID 0001-0026-00000115    Rows waited on:    Session 191: obj - rowid = 0001773A - AAAXc6AABAABc2BAAA    (dictionary objn - 96058, file - 1, block - 380289, slot - 0)    ----- Information for the OTHER waiting sessions -----  ----- End of information for the OTHER waiting sessions -----    Information for THIS session:    ----- Current SQL Statement for this session (sql\_id=3w3thujdh1y3a) -----  UPDATE T\_DEADLOCK\_AT2\_LHR SET NAME = NAME WHERE ID = 1  ----- PL/SQL Stack -----  ----- PL/SQL Call Stack -----    object      line  object    handle    number  name  700010052c9cf40         4  procedure SYS.PRO\_TESTAT\_LHR  700010052cb7588         1  anonymous block |

1、 **脚本**

|  |
| --- |
| CREATE TABLE T\_DEADLOCK\_AT2\_LHR(ID NUMBER, NAME VARCHAR2(30));  INSERT INTO T\_DEADLOCK\_AT2\_LHR SELECT ROWNUM, TNAME FROM TAB WHERE ROWNUM<=4;  COMMIT;    CREATE OR REPLACE PROCEDURE PRO\_TESTAT\_LHR AS    PRAGMA AUTONOMOUS\_TRANSACTION;  BEGIN    UPDATE T\_DEADLOCK\_AT2\_LHR SET NAME = NAME WHERE ID = 1;    COMMIT;  END;  /    UPDATE T\_DEADLOCK\_AT2\_LHR SET NAME = NAME WHERE ID = 1;  exec PRO\_TESTAT\_LHR;        ERROR AT LINE 1:  ORA-00060: DEADLOCK DETECTED WHILE WAITING FOR RESOURCE  ORA-06512: AT "SYS.PRO\_TESTAT\_LHR", LINE 4  ORA-06512: AT LINE 3 |

二、 **模拟二：插入**

主事务和自治事务插入的是同一个主键值也会引起死锁。

|  |
| --- |
| SYS@lhrdb S1> CREATE TABLE T\_DEADLOCK\_AT\_LHR(X INT  PRIMARY KEY,Y INT);    Table created.    SYS@lhrdb S1> CREATE OR REPLACE PROCEDURE PRO\_AUTO\_PROC\_LHR(P\_X INT,P\_Y INT) AS    2  PRAGMA AUTONOMOUS\_TRANSACTION;    3  BEGIN    4    INSERT INTO T\_DEADLOCK\_AT\_LHR(X,Y) VALUES(P\_X,P\_Y);    5    COMMIT;    6  END;    7  /    Procedure created.    SYS@lhrdb S1> INSERT INTO T\_DEADLOCK\_AT\_LHR(X,Y) VALUES(1,1);    1 row created.    SYS@lhrdb S1> EXEC PRO\_AUTO\_PROC\_LHR(1,2);  BEGIN PRO\_AUTO\_PROC\_LHR(1,2); END;    \*  ERROR at line 1:  ORA-00060: deadlock detected while waiting for resource  ORA-06512: at "SYS.PRO\_AUTO\_PROC\_LHR", line 4  ORA-06512: at line 1 |

告警日志：

|  |
| --- |
| Fri Sep 23 13:49:06 2016  ORA-00060: Deadlock detected. More info in file /oracle/app/oracle/diag/rdbms/lhrdb/lhrdb/trace/lhrdb\_ora\_19726460.trc. |

内容：

|  |
| --- |
| \*\*\* 2016-09-23 13:49:06.546  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-000a000b-00000d28        29     156     X             29     156           S    session 156: DID 0001-001D-0000000D     session 156: DID 0001-001D-0000000D    Rows waited on:    Session 156: no row    ----- Information for the OTHER waiting sessions -----  ----- End of information for the OTHER waiting sessions -----    Information for THIS session:    ----- Current SQL Statement for this session (sql\_id=4fv0tmjrzv28u) -----  INSERT INTO T\_DEADLOCK\_AT\_LHR(X,Y) VALUES(:B2 ,:B1 )  ----- PL/SQL Stack -----  ----- PL/SQL Call Stack -----    object      line  object    handle    number  name  70001005a9a0580         4  procedure SYS.PRO\_AUTO\_PROC\_LHR  700010052d18b90         1  anonymous block |

可以看到，等待的和持有锁的是同一个会话，根据trace信息记录的对象，发现问题是自治事务导致的。

1、 **脚本**

|  |
| --- |
| CREATE TABLE T\_DEADLOCK\_AT\_LHR(X INT  PRIMARY KEY,Y INT);      CREATE OR REPLACE PROCEDURE PRO\_AUTO\_PROC\_LHR(P\_X INT,P\_Y INT) AS  PRAGMA AUTONOMOUS\_TRANSACTION;  BEGIN    INSERT INTO T\_DEADLOCK\_AT\_LHR(X,Y) VALUES(P\_X,P\_Y);    COMMIT;  END;  /      INSERT INTO T\_DEADLOCK\_AT\_LHR(X,Y) VALUES(1,1);        SQL> EXEC PRO\_AUTO\_PROC\_LHR(1,2);  BEGIN AUTO\_PROC(1,2); END;    \*  ERROR AT LINE 1:  ORA-00060: DEADLOCK DETECTED WHILE WAITING FOR RESOURCE  ORA-06512: AT "BOCNET.AUTO\_PROC", LINE 4  ORA-06512: AT LINE 1 |

### 2.2  ****块级死锁****

块级死锁其实指的就是ITL死锁。

#### 2.2.1  ****ITL简介****

ITL（Interested Transaction List）是Oracle数据块内部的一个组成部分，用来记录该块所有发生的事务，有的时候也叫ITL槽位。如果一个事务一直没有提交，那么，这个事务将一直占用一个ITL槽位，ITL里面记录了事务信息、回滚段的入口和事务类型等等。如果这个事务已经提交，那么，ITL槽位中还保存有这个事务提交时候的SCN号。ITL的个数受表的存储参数INITRANS控制，在一个块内部，默认分配了2个ITL的个数，如果这个块内还有空闲空间，那么Oracle是可以利用这些空闲空间再分配ITL。如果没有了空闲空间，那么，这个块因为不能分配新的ITL，所以，就可能发生ITL等待。如果在并发量特别大的系统中，那么最好分配足够的ITL个数，或者设置足够的PCTFREE，保证ITL能扩展，但是PCTFREE有可能是被行数据给消耗掉的，例如UPDATE，所以，也有可能导致块内部的空间不够而导致ITL等待，出现了ITL等待就可能导致ITL死锁。

一、 **ITL结构**

如果DUMP一个块（命令：alter system dump datafile X block XXX;），那么在DUMP文件中就可以看到ITL信息：

|  |
| --- |
| Itl            Xid                       Uba         Flag   Lck        Scn/Fsc  0x01   0x0008.002.000009e9  0x00c0108b.04ac.24  --U-    3  fsc 0x0000.00752951  0x02   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000 |

**1) Itl：**ITL事务槽编号，ITL事务槽号的流水编号

**2) Xid**：代表对应的事务id（transac[X]tion identified），在回滚段事务表中有一条记录和这个事务对应。Xid由三列使用十六进制编码的数字列表示，分别是：Undo Segment Number +Transaction Table Slot Number+ Wrap,即由**undo段号+undo槽号+undo槽号的覆盖次数**三部分组成，即**usn.slot.sqn，**这里0x0008.002.000009e9转换为10进制为8.2.2537，从下边的查询出的结果是相对应的。

|  |
| --- |
| SYS@lhrdb> select xidusn,xidslot,xidsqn,ubafil,ubablk,ubasqn,ubarec from v$transaction;      XIDUSN    XIDSLOT     XIDSQN     UBAFIL     UBABLK     UBASQN     UBAREC  ---------- ---------- ---------- ---------- ---------- ---------- ----------           8          2       2537         3        4235       1196          36 |

**3) Uba**：(Undo Block Address)，该事务对应的回滚段地址，记录了最近一次的该记录的前镜像（修改前的值）。Uba组成：**Undo块地址(undo文件号和数据块号)+回滚序列号+回滚记录号**。多版本一致读是Oracle保证读操作不会被事务阻塞的重要特性。当Server Process需要查询一个正在被事务修改，但是尚未提交的数据时，就根据ITL上的uba定位到对应Undo前镜像数据位置。这里的Uba为：0x00c0108b.04ac.24，其中00c0108b（16进制）=0000 0000 1100 0000 0001 0000 1000 1011（2进制，共32位，前10位代表文件号，后22位代表数据块号）=文件号为3，块号为4235：（10进制）；04ac（16进制）=1196（10进制）；24（16进制）=36（10进制）。这个结果和v$transaction查询出来的结果一致。

|  |
| --- |
| SELECT UBAFIL 回滚段文件号,UBABLK 数据块号,UBASQN 回滚序列号,UBAREC 回滚记录号 FROM v$transaction ;  --查看UBA |

**4) Flag**：事务标志位，即当前事务槽的状态信息。这个标志位就记录了这个事务的操作状态，各个标志的含义分别是：

|  |  |
| --- | --- |
| **标识** | **简介** |
| ---- | 事务是活动的，未提交，或者在块清除前提交事务 |
| C--- | transaction has been committed and locks cleaned out   --事务已经提交，锁已经被清除（提交） |
| -B-- | this undo record contains the undo for this ITL entry |
| --U- | transaction committed (maybe long ago); SCN is an upper bound  --事务已经提交，但是锁还没有清除（快速提交） |
| ---T | transaction was still active at block cleanout SCN   --块清除的SCN被记录时，该事务仍然是活动的，块上如果有已经提交的事务，那么在clean ount的时候，块会被进行清除，但是这个块里面的事务不会被清除。 |
| C-U- | 块被延迟清除，回滚段的信息已经改写，SCN显示为最小的SCN，需要由由回滚段重新生成，例如在提交以前，数据块已经flush到数据文件上。 |

**5) Lck**：表示这个事务所影响的行数，锁住了几行数据，对应有几个行锁。我们看到01号事物槽Lck为3，因为该事物槽中的事物Flag为U，证明该事物已经提交，但是锁还没有清除。再比如对于下边这个ITL：

|  |
| --- |
| Itl            Xid                       Uba         Flag    Lck        Scn/Fsc  0x01   0x0006.020.00000271  0x00800205.0257.13  C---     0         scn 0x0000.001732c4  0x02   0x0008.006.00000279  0x00800351.0278.15  ----     1         fsc 0x0000.00000000 |

我们看到01号事物槽Lck为0，因为该事物槽中的事物Flag为C，证明该事物已经提交，锁也被清楚掉了，该事物槽可以被重用了。02号事物槽Lck为1，是因为我对第一行做了一个更新，并且没有提交，Flag为“----”说明该事物是活动的。

**6) Scn/Fsc**：Commit SCN或者快速提交（Fast Commit Fsc）的SCN。 Scn=SCN of commited TX; Fsc=Free space credit(bytes)每条记录中的行级锁对应Itl条目lb，对应于Itl列表中的序号，即那个事务在该记录上产生的锁。一个事物只有在提交之后才会在ITL事物槽中记录SCN。

二、 **ITL个数**

ITL的个数，受参数INITRANS控制，最大ITL个数受MAXTRANS控制（11g已废弃MAXTRANS），在一个块内部，默认分配了2个ITL的个数。ITL是block级的概念，一个ITL占用块46B的空间，参数INITRANS意味着块中除去block header外一部分存储空间无法被记录使用（46B\*INITRANS），当块中还有一定的FREE SPACE时，ORACLE可以使用FREE SPACE构建ITL供事务使用，如果这个块内还有空闲空间，那么Oracle是可以利用这些空闲空间并再分配ITL。如果没有了空闲空间（free space），那么，这个块因为不能分配新的ITL，所以就可能发生ITL等待，即enq: TX - allocate ITL entry等待事件。**注意：10g以后MAXTRANS参数被废弃，默认最大支持255个并发。**

如果在并发量特别大的系统中，最好分配足够的ITL个数，其实它并浪费不了太多的空间，或者，设置足够的PCTFREE，保证ITL能扩展，但是PCTFREE有可能是被行数据给消耗掉的，如UPDATE，所以，也有可能导致块内部的空间不够而导致ITL等待。

**对于表（数据块）来说，INITRANS这个参数的默认值是1。对于索引（索引块）来说，这个参数默认值是2。**

ITL等待表现出的等待事件为“TX - allocate ITL entry”，根据MOS（Troubleshooting waits for 'enq: TX - allocate ITL entry' (Doc ID 1472175.1)提供的解决办法，需要修改一些参数，SQL如下，这里假设用户名为TLHR，表名为TLHRBOKBAL，表上的索引名为PK\_TLHRBOKBAL：

|  |
| --- |
| ALTER TABLE TLHR.TLHRBOKBAL PCTFREE 20  INITRANS 16;  ALTER TABLE TLHR.TLHRBOKBAL MOVE NOLOGGING PARALLEL 12;  ALTER TABLE TLHR.TLHRBOKBAL LOGGING NOPARALLEL;  ALTER INDEX TLHR.PK\_TLHRBOKBAL  REBUILD PCTFREE 20 INITRANS 16 NOLOGGING PARALLEL 12;  ALTER INDEX TLHR.PK\_TLHRBOKBAL LOGGING NOPARALLEL; |

无MOS权限的朋友可以去http://blog.itpub.net/26736162/viewspace-2124531/阅读。

#### 2.2.2  ****ITL引起的死锁案例处理****

由ITL不足引发的块级死锁的一个处理案例可以参考我的BLOG:http://blog.itpub.net/26736162/viewspace-2124771/、http://blog.itpub.net/26736162/viewspace-2124735/。

#### 2.2.3  ****ITL死锁的模拟****

我们首先创建一张表T\_ITL\_LHR，这里指定PCTFREE为0，INITRANS为1，就是为了观察到ITL的真实等待情况，然后我们给这些块内插入数据，把块填满，让它不能有空间分配。

|  |
| --- |
| SYS@lhrdb21> SELECT \* FROM V$VERSION;    BANNER  --------------------------------------------------------------------------------  Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production  PL/SQL Release 11.2.0.4.0 - Production  CORE    11.2.0.4.0      Production  TNS for IBM/AIX RISC System/6000: Version 11.2.0.4.0 - Production  NLSRTL Version 11.2.0.4.0 - Production    SYS@lhrdb21> SHOW PARAMETER CLUSTER    NAME                                 TYPE        VALUE  ------------------------------------ ----------- ------------------------------  cluster\_database                     boolean     TRUE  cluster\_database\_instances           integer     2  cluster\_interconnects                string    SYS@lhrdb21> CREATE TABLE T\_ITL\_LHR(A INT) PCTFREE 0 INITRANS 1;    Table created.    SYS@lhrdb21> BEGIN    2    FOR I IN 1 .. 2000 LOOP    3      INSERT INTO T\_ITL\_LHR VALUES (I);    4    END LOOP;    5  END;    6  /    PL/SQL procedure successfully completed.    SYS@lhrdb21> COMMIT;    Commit complete. |

我们检查数据填充的情况：

|  |
| --- |
| SYS@lhrdb21> SELECT F, B, COUNT(\*)    2    FROM (SELECT DBMS\_ROWID.ROWID\_RELATIVE\_FNO(ROWID) F,    3                 DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID) B    4            FROM T\_ITL\_LHR)    5   GROUP BY F, B    6   ORDER BY F,B;             F          B      COUNT(\*)  ---------- ---------- ----------           1      94953        734           1      94954        734           1      94955        532 |

可以发现，这2000条数据分布在3个块内部，其中有2个块(94953和94954)填满了，一个块（94955）是半满的。因为有2个ITL槽位，我们需要拿2个满的数据块，4个进程来模拟ITL死锁：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验步骤** | **会话** | **SID** | **要更新的块号** | **要更新的行号** | **是否有阻塞** |
| 步骤一 | 1 | 19 | 94953  94953 | 1 | N |
| 2 | 79 | 2 | N |  |  |
| 3 | 78 | 94954  94954 | 1 | N |  |
| 4 | 139 | 2 | N |  |  |

**会话1：**

|  |
| --- |
| SYS@lhrdb21> SELECT USERENV('SID') FROM DUAL;    USERENV('SID')  --------------              19    SYS@lhrdb21> UPDATE T\_ITL\_LHR SET A=A    2   WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94953    3  AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=1;    1 row updated. |

**会话2：**

|  |
| --- |
| SYS@lhrdb21> SELECT USERENV('SID') FROM DUAL;    USERENV('SID')  --------------              79    SYS@lhrdb21> UPDATE T\_ITL\_LHR SET A=A    2   WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94953    3  AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=2;    1 row updated. |

**会话3：**

|  |
| --- |
| SYS@lhrdb21> SELECT USERENV('SID') FROM DUAL;    USERENV('SID')  --------------              78    SYS@lhrdb21> UPDATE T\_ITL\_LHR SET A=A    2   WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94954    3  AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=1;    1 row updated. |

**会话4：**

|  |
| --- |
| SYS@lhrdb21> SELECT USERENV('SID') FROM DUAL;    USERENV('SID')  --------------             139    SYS@lhrdb21> UPDATE T\_ITL\_LHR SET A=A    2   WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94954    3  AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=2;    1 row updated. |

这个时候系统不存在阻塞，

SELECT NVL**(**A.SQL\_ID**,** A.PREV\_SQL\_ID**)** SQL\_ID**,**

       A.BLOCKING\_SESSION**,**

       A.SID**,**

       A.SERIAL#**,**

       A.LOGON\_TIME**,**

       A.EVENT

  FROM GV$SESSION A

 WHERE A.SID IN **(19,** **79,78,139)**

 ORDER BY A.LOGON\_TIME**;**

以上4个进程把2个不同块的4个ITL槽位给消耗光了，现在的情况，就是让他们互相锁住，达成死锁条件，回到会话1，更新块94954，注意，以上4个操作，包括以下的操作，更新的根本不是同一行数据，主要是为了防止出现的是行锁等待。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验步骤** | **会话** | **SID** | **要更新的块号** | **要更新的行号** | **是否有阻塞** |
| 步骤一 | 1 | 19 | 94953  94953 | 1 | N |
| 2 | 79 | 2 | N |  |  |
| 3 | 78 | 94954  94954 | 1 | N |  |
| 4 | 139 | 2 | N |  |  |
| 步骤二 | 1 | 19 | 94954 | 3 | Y |
| 3 | 78 | 94953 | 3 | Y |  |

**会话1：**

UPDATE T\_ITL\_LHR SET A=A

WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94954

AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=3;

会话1出现了等待。

**会话3：**

UPDATE T\_ITL\_LHR SET A=A

WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94953

AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=3;

会话3发现出现了等待。

我们查询阻塞的具体情况：

SELECT NVL**(**A.SQL\_ID**,** A.PREV\_SQL\_ID**)** SQL\_ID**,**

       A.BLOCKING\_SESSION**,**

       A.SID**,**

       A.SERIAL#**,**

       A.LOGON\_TIME**,**

       A.EVENT

  FROM GV$SESSION A

 WHERE A.SID IN **(19,** **79,78,139)**

 ORDER BY A.LOGON\_TIME**;**

可以看到，会话1被会话4阻塞了，会话3被会话2阻塞了。

注意，如果是9i，在这里就报死锁了，但是在10g里面，这个时候，死锁是不会发生的，因为这里的会话1还可以等待会话4释放资源，会话3还可以等待会话2释放资源，只要会话2与会话4释放了资源，整个环境又活了，那么我们需要把这两个进程也塞住。

出现的是行锁等待。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验步骤** | **会话** | **SID** | **要更新的块号** | **要更新的行号** | **是否有阻塞** |
| 步骤一 | 1 | 19 | 94953  94953 | 1 | N |
| 2 | 79 | 2 | N |  |  |
| 3 | 78 | 94954  94954 | 1 | N |  |
| 4 | 139 | 2 | N |  |  |
| 步骤二 | 1 | 19 | 94954 | 3 | Y |
| 3 | 78 | 94953 | 3 | Y |  |
| 步骤三 | 2 | 79 | 94954 | 4 | Y |
| 4 | 139 | 94953 | 4 | Y |  |

**会话2，注意，我们也不是更新的同一行数据：**

UPDATE T\_ITL\_LHR SET A=A

WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94954

AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=4;

会话2出现了等待，具体阻塞情况：

我做了几次实验，会话2执行完SQL后，会话3到这里就报出了死锁，但有的时候并没有产生死锁，应该跟系统的阻塞顺序有关，若没有产生死锁，我们可以继续会话4的操作。

|  |
| --- |
| SYS@lhrdb21> UPDATE T\_ITL\_LHR SET A=A    2   WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94953    3  AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=3;      UPDATE T\_ITL\_LHR SET A=A         \*  ERROR at line 1:  ORA-00060: deadlock detected while waiting for resource |

**会话4，注意，我们也不是更新的同一行数据：**

UPDATE T\_ITL\_LHR SET A=A

WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94953

AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=4;

会话4发现出现了等待。

虽然，以上的每个更新语句，更新的都不是同一个数据行，但是，的确，所有的进程都被阻塞住了，那么，死锁的条件也达到了，等待一会（**这个时间有个隐含参数来控制的：\_lm\_dd\_interval**），我们可以看到，会话2出现提示，死锁：

|  |
| --- |
| SYS@lhrdb21> UPDATE T\_ITL\_LHR SET A=A    2   WHERE DBMS\_ROWID.ROWID\_BLOCK\_NUMBER(ROWID)=94954    3  AND DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID)=4;    UPDATE T\_ITL\_LHR SET A=A         \*  ERROR at line 1:  ORA-00060: deadlock detected while waiting for resource |

报出死锁之后的阻塞情况：

我们可以在会话2上继续执行步骤三中的SQL，依然会产生死锁。生成死锁后，在告警日志中有下边的语句：

|  |
| --- |
| Fri Sep 09 17:56:55 2016  Global Enqueue Services Deadlock detected. More info in file  /oracle/app/oracle/diag/rdbms/lhrdb2/lhrdb21/trace/lhrdb21\_lmd0\_17039368.trc. |

其中的内容有非常经典的一段Global Wait-For-Graph(WFG)：

|  |
| --- |
| \*\*\* 2016-09-09 17:48:22.216  Submitting asynchronized dump request [1c]. summary=[ges process stack dump (kjdglblkrdm1)].  Global blockers dump end:-----------------------------------  Global Wait-For-Graph(WFG) at ddTS[0.395] :  BLOCKED 0x700010063d59b90 3 wq 2 cvtops x1001 TX 0x7000b.0xa67(ext 0x2,0x0)[1002-0029-00008387] inst 1  BLOCKER 0x700010063c6d268 3 wq 1 cvtops x28 TX 0x7000b.0xa67(ext 0x2,0x0)[1002-002D-00003742] inst 1  BLOCKED 0x700010063d5adc8 3 wq 2 cvtops x1 TX 0x30021.0x848(ext 0x2,0x0)[1002-002D-00003742] inst 1  BLOCKER 0x700010063d5a4b8 3 wq 1 cvtops x28 TX 0x30021.0x848(ext 0x2,0x0)[1002-0029-00008387] inst 1 |

该实验过程可能有点复杂，小麦苗画了个图来说明整个实验过程：

[oracle ora-60 deadlock发生在多个会话的情况](http://blog.itpub.net/28539951/viewspace-2139925/)

 ora-60错误可以发生在两个会话之间，即两个会话之间互相持有一个对方等待的资源（A deadlock occurs when a session (A) wants a resource held by another session (B) , but that session also wants a resource held by the first session (A). ）。如果几个会话之间的资源争用是环形，这也是死锁，并且oracle同样会监测到并处理。

**以下演示：**

DB Version:11.2.0.4

**建测试表：**

create table scott.tb\_60

(

  num NUMBER,

  txt VARCHAR2(10)

);

insert into scott.tb\_60 values ( 1, 'First' );

insert into scott.tb\_60 values ( 2, 'Second' );

insert into scott.tb\_60 values ( 3, 'Third' );

commit;

**会话1：**

更新num=1的记录

update scott.tb\_60

set txt='a'

where num=1;

**会话2：**

更新num=2的记录

update scott.tb\_60

set txt='b'

where num=2;

更新num=1的记录，此时会话2等待会话1提交或回滚来释放num=1上的tx锁

update scott.tb\_60

set txt='a'

where num=1;

**会话3：**

更新num=3的记录

update scott.tb\_60

set txt='c'

where num=3;

更新num=2的记录，此时会话3等待会话2提交或回滚来释放num=2上的tx锁

update scott.tb\_60

set txt='b'

where num=2;

**会话1：**

更新num=3的记录，此时会话1等待会话3提交或回滚来释放num=3上的tx锁，死锁产生

update scott.tb\_60

set txt='c'

where num=3;

报错信息：

ORA-00060: deadlock detected while waiting for resource

数据库alert.log中记录错误

ORA-00060: Deadlock detected. More info in file /u01/app/oracle/diag/rdbms/ct6601z3/ct6601z3/trace/ct6601z3\_ora\_6528.trc.

对应的trace file中记录死锁的相关信息

**Deadlock graph:**

                       ---------Blocker(s)--------  ---------Waiter(s)---------

Resource Name          process session holds waits  process session holds waits

TX-000a0005-00000c72        27     197     X             28       8           X

TX-0001001a-00000ca7        28       8     X             26     132           X

TX-00030001-00000d4e        26     132     X             27     197           X

session 197: DID 0001-001B-00000004     session 8: DID 0001-001C-00000002

session 8: DID 0001-001C-00000002       session 132: DID 0001-001A-00000004

session 132: DID 0001-001A-00000004     session 197: DID 0001-001B-00000004

**Rows waited on:**

  Session 197: obj - rowid = 00038F6D - AAA49tAAEAAABlkAAA

  (dictionary objn - 233325, file - 4, block - 6500, slot - 0)

  Session 8: obj - rowid = 00038F6D - AAA49tAAEAAABlkAAB

  (dictionary objn - 233325, file - 4, block - 6500, slot - 1)

  Session 132: obj - rowid = 00038F6D - AAA49tAAEAAABlkAAC

  (dictionary objn - 233325, file - 4, block - 6500, slot - 2)

...

[RAC全局死锁检测时间](http://blog.itpub.net/26736162/viewspace-2125366/)

对于单实例数据库而言，死锁的检测在秒级完成，而RAC环境则死锁的检测时间默认达到了1分钟。

对于单实例环境如果出现了死锁，那么马上其中一个进程就被中止，用户可以快速的得到错误返回。而对于RAC而言，死锁的检测并不是实时完成，而是需要60秒左右的时间。

会话1执行：

SQL> create table t\_deadlock (id number primary key, name varchar2(30));

Table created.

Elapsed: 00:00:00.12  
SQL> insert into t\_deadlock values (1, 'a');

1 row created.

Elapsed: 00:00:00.00  
SQL> insert into t\_deadlock values (2, 'b');

1 row created.

Elapsed: 00:00:00.00  
SQL> commit;

Commit complete.

Elapsed: 00:00:00.00  
SQL> update t\_deadlock set name = 'a1' where id = 1;

1 row updated.

Elapsed: 00:00:00.00

会话2执行：

SQL> set timing on  
SQL> update t\_deadlock set name = 'b2' where id = 2;

1 row updated.

Elapsed: 00:00:00.00  
SQL> update t\_deadlock set name = 'a2' where id = 1;

此时，会话2等待会话1的最终操作，下面会话1更新被会话2锁定的行，引发死锁：

SQL> update t\_deadlock set name = 'b1' where id = 2;  
update t\_deadlock set name = 'b1' where id = 2  
\*  
ERROR at line 1:  
ORA-00060: deadlock detected while waiting for resource

Elapsed: 00:01:00.12

可以看到，死锁的超时检测为1分钟。

而这个死锁的检测时间是可以调整的，Oracle通过隐含参数\_lm\_dd\_interval控制：

SQL> conn / as sysdba  
Connected.  
SQL> alter system set "\_lm\_dd\_interval" = 30 scope = spfile;

System altered.

SQL> shutdown immediate  
Database closed.  
Database dismounted.  
ORACLE instance shut down.  
SQL> startup  
ORACLE instance started.

Total System Global Area 281018368 bytes  
Fixed Size 2095672 bytes  
Variable Size 104859080 bytes  
Database Buffers 167772160 bytes  
Redo Buffers 6291456 bytes  
Database mounted.  
Database opened.

再次测试死锁的检测时间，会话1：

SQL> update t\_deadlock set name = 'a1' where id = 1;

1 row updated.

SQL> set timing on

会话2执行更新：

SQL> set timing on  
SQL> update t\_deadlock set name = 'b2' where id = 2;

1 row updated.

Elapsed: 00:00:00.02  
SQL> update t\_deadlock set name = 'a2' where id = 1;

会话1执行更新引发死锁：

SQL> update t\_deadlock set name = 'b1' where id = 2;

大约30秒后，会话2报错ORA-60：

update t\_deadlock set name = 'a2' where id = 1  
\*  
ERROR at line 1:  
ORA-00060: deadlock detected while waiting for resource

Elapsed: 00:00:30.27

在10.2.0.2版本上，Oracle存在一个bug，允许这个参数设置为0，在10.2.0.3以后，这个bug被修正，如果设置为0后，则数据库无法正常启动：

[oracle@node1 ~]$ sqlplus / as sysdba

SQL\*Plus: Release 10.2.0.5.0 - Production on Mon Jun 4 07:54:09 2012

Copyright (c) 1982, 2010, Oracle. All Rights Reserved.

Connected to an idle instance.

SQL> startup  
ORA-00067: invalid value 0 for parameter \_lm\_dd\_interval; must be at least 1

最后修改隐含参数是Oracle不推荐的，而且修改这个参数势必会影响RAC的正常工作方式导致LDM进程的繁忙度增加，而且可能影响RAC环境的稳定性和可用性。

如果确实对于前台的死锁检查时间要求较高，建议在测试环境中详细测试后再部署到产品环境中。

在11g中，该参数默认值为10