

# mysql死锁以及死锁日志分析

落学梦宇

## 死锁的概念

死锁：死锁一般是事务相互等待对方资源，最后形成环路造成的。

对于死锁，数据库处理方法：牺牲一个连接，保证另外一个连接成功执行。

发生死锁会返回ERROR：1213 错误提示，大部分的死锁InnoDB存储引擎本身可以侦测到，不需要人为进行干预。

注意：

InnoDB存储引擎并不会回滚大部分的错误异常，像阻塞章节里面的例子，但是死锁例外，发现死锁后，InnoDB存储引擎会马上回滚一个事务，会返回1213错误。

## 死锁的情形举例

eg1:

```
begin;
rows affected (0.00 sec)

delete from info_area where id=1;
rows affected (0.04 sec)

update info_users set mobile='18514656666' where mobile='18514656420';
row affected (14.50 sec)
1 Changed: 1 Warnings: 0

MariaDB > begin;
Query OK, 0 rows affected (0.00 sec)

MariaDB > update info_users set mobile='18514656666' where mobile='
Query OK, 1 row affected (0.03 sec)
Rows matched: 1 Changed: 1 Warnings: 0

MariaDB > delete from info_area where id=1;
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting
```

```
LATEST DETECTED DEADLOCK
=====
2017-05-18 15:34:00 7f148878a000
*** (1) TRANSACTION:
TRANSACTION 4463088, ACTIVE 46 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 17 lock struct(s), heap size 2096, 2885 row lock(s)
MySQL thread id 18, OS thread handle 5a7f73884c2b00, query id 24313 localhost root Searching rows for update
update info_users set mobile='18514656666' where mobile='18514656420'
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 32 page no 29155 n bits 824 index MOBILE of table info_users trx table locks 2 total table locks 2 trx id 4463088 lock_mode X wait
ing lock hold time 18 wait time before grant 0
*** (2) TRANSACTION:
TRANSACTION 4463089, ACTIVE 28 sec starting index read
mysql tables in use 1, locked 1
6 lock struct(s), heap size 1376, 4 row lock(s), undo log entries 1
MySQL thread id 23, OS thread handle 5a7f73884c2b00, query id 24314 localhost root updating
delete from info_area where id=1
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 32 page no 29155 n bits 824 index MOBILE of table info_users trx table locks 2 total table locks 2 trx id 4463089 lock_mode X lock
hold time 22 wait time before grant 0
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 32 page no 4 n bits 212 index GEN_CLUST_INDEX of table info_area trx table locks 2 total table locks 2 trx id 4463089 lock_mode X wait
ing lock hold time 0 wait time before grant 0
```

## 分析死锁日志：

### 第一部分

从日志里我们可以看到事务1当前正在执行update info\_users set mobile='18514656666' where mobile='18514656620'，该条语句正在申请表info\_users的索引IDX\_MOBILE的X锁，所以提示lock\_mode X waiting

第二部分：

然后日志的下半部分说明了事务2当前‘持有的锁’以及‘等待的锁’：

从日志的HOLDS THE LOCKS(S)块中我们可以看到事务2持有索引IDX\_MOBILE的X锁，并且是记录锁（Record Lock）。该锁是通过事务2在步骤2执行的update语句申请的。

从日志的WAITING FOR THIS LOCK TO BE GRANTED块中我们可以看到事务2正在申请持有表info\_area的索引GEN\_CLUST\_INDEX的X锁，该锁是delete from info\_area where id=1;语句申请的。

eg2:

步骤	事务1	事务2
1	begin;	begin;
2	update info_users set name='aaa' where id=1;	空
3	空	update info_users set name='bbb' where id=2;
4	update info_users set name='bbb' where id=2;	空

eg3:

步骤	事务1	事务2
1	begin;	begin;
2		DELETE from users where uid='bbb';执行成功
3	DELETE from users where uid='bbb';等待	空

```
>
>
> begin;
rows affected (0.00 sec)

DELETE from users where uid='bbb';
(40001): Deadlock found when trying to get lock; try restarting transaction
```

```
.MariaDB [ ] > begin;
Query OK, 0 rows affected (0.00 sec)

.MariaDB [ ] > DELETE from users where uid=
Query OK, 1 row affected (0.00 sec)

.MariaDB [ ] > insert INTO users VALUES(2,
Query OK, 1 row affected (0.00 sec)
```

分析死锁日志：

第一部分

从日志里我们可以看到事务1当前正在执行DELETE from users where uid='bbb';，该条语句正在申请索引UID的X锁，所以提示lock\_mode X waiting

第二部分：

然后日志的下半部分说明了事务2当前‘持有的锁’以及‘等待的锁’：

从日志的HOLDS THE LOCKS(S)块中我们可以看到事务2持有索引UID的X锁，并且是记录锁（Record Lock）。该锁是通过事务2在步骤2执行的delete语句申请的。

从日志的WAITING FOR THIS LOCK TO BE GRANTED块中我们可以看到事务2正在申请持有索引UID的S锁，该锁是insert INTO users VALUES(2,'bbb');语句申请的。insert语句在普通情况下是会申请X锁，但是这里出现了S锁。这是因为uid字段是一个索引，所以insert语句会在插入前进行一次duplicate key的检查，为了使这次检查成功，需要申请S锁防止其他事务对uid字段进行修改。

那么为什么该S锁会失败呢？这是对同一个字段的锁的申请是需要排队的。S锁前面还有一个未申请成功的X锁，所以S锁必须等待，所以形成了循环等待，死锁出现了。

通过阅读死锁日志，我们可以清楚地知道两个事务形成了怎样的循环等待，再加以分析，就可以逆向推断出循环等待的成因，也就是死锁形成的原因。