链滴

两个 INSERT 发生死锁原因剖析

作者：centrexzj

原文链接：https://ld246.com/article/1536649176994

来源网站：链滴

开始之前，关于锁、死锁，我们要先统一下几点认知：

1. 死锁是由于多个事务相互持有其他事务所需要的锁，结果导致事务都无法继续，进而触发死锁检测其中某个事务会被回滚，释放相应的锁，其他事务得以正常继续；简言之，就是多个事务之间的锁等产生了回路，死循环了；

2. 死锁发生时，会立刻被检测到，并且回滚其中某个事务，而不会长时间阻塞、等待；

3. 从MySQL 5.7.15开始，新增选项 innodb_deadlock_detect，没记错的话应该是阿里团队率先实的。当它设置为 OFF 时（默认值是 ON），InnoDB会不检测死锁，在高并发场景（例如“秒杀”）务中特别有用，可以有效提高事务并发性能；

4. 在启用死锁检测时，InnoDB默认的最大检测深度为200，在上面提到的高并发高竞争场景下，在点数据上的锁等待队列可能很长，死锁检测代价很大。或者当等待队列中所有的行锁总数超过 100万，也会被认为认为发生死锁了，直接触发死锁检测处理机制；

5. InnoDB行锁等待超时默认为50秒，一般建议设置5-10秒就够了；

6. 有时候，可能会口误把 长时间的行锁等待 说成是 死锁，其实二者完全不一样，不要犯这种常识性误。

好了，正式开始今天的案例。

先看测试表：

```
mysql> show create table ld\G
*************************** 1. row ***************************
       Table: ld
Create Table: CREATE TABLE `ld` (
  `id` int(11) NOT NULL DEFAULT '0',
  `name` varchar(10) DEFAULT NULL,
  UNIQUE KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> select * from ld;
+----+-------+
| id | name  |
+----+-------+
|  1 | jerry |
+----+-------+
1 row in set (0.00 sec)
```

然后我们执行下面的测试：

session1	session2	session3
begin;delete from ld where id=1;		
	begin;insert into ld select 1,'dkey';	
		begin;insert into ld select 1,'mark';
commit;		
	Query OK, 1 row affected (11.82 sec)Records: 1 Duplicates: 0 Warnings: 0	ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction

这时候我们看下InnoDB STATUS的输出：

```
------------------------
LATEST DETECTED DEADLOCK
------------------------
2017-09-05 04:50:42 0x7f233f088700
*** (1) TRANSACTION:
TRANSACTION 5415714, ACTIVE 19 sec inserting
mysql tables in use 1, locked 1
LOCK WAIT 3 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 1430, OS thread handle 139789350385408, query id 9663235 localhost root
xecuting
insert into ld select 1,'dkey'
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 130 page no 3 n bits 72 index id of table `sbtest`.`ld` trx id 5415714 l
ck_mode X locks rec but not gap waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 32
 0: len 4; hex 80000001; asc     ;;
 1: len 6; hex 00000052a31d; asc     R  ;;
 2: len 7; hex 670000038517ea; asc g       ;;
 3: len 5; hex 6a65727279; asc jerry;;

*** (2) TRANSACTION:
TRANSACTION 5415715, ACTIVE 10 sec inserting
mysql tables in use 1, locked 1
3 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 1431, OS thread handle 139789358106368, query id 9663237 localhost root
xecuting
insert into ld select 1,'mark'
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 130 page no 3 n bits 72 index id of table `sbtest`.`ld` trx id 5415715 l
ck mode S (注意S锁)
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 32
 0: len 4; hex 80000001; asc     ;;
 1: len 6; hex 00000052a31d; asc     R  ;;
 2: len 7; hex 670000038517ea; asc g       ;;
 3: len 5; hex 6a65727279; asc jerry;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 130 page no 3 n bits 72 index id of table `sbtest`.`ld` trx id 5415715 l
ck_mode X locks rec but not gap waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info bits 32
 0: len 4; hex 80000001; asc     ;;
 1: len 6; hex 00000052a31d; asc     R  ;;
 2: len 7; hex 670000038517ea; asc g       ;;
 3: len 5; hex 6a65727279; asc jerry;;
```

从上面这个输出来看，我们看到的现场是两个 insert 请求发生了死锁。单纯看这2个SQL的话，应该产生锁等待才对，而不是死锁。

按照我们常规理解，session1未commit前，应该是持有id=1上的record lock(X)，而session2和session3则都在等待这个锁的释放。而实际上呢，肯定不是这样的，否则也不至于发生死锁了。

这次的案例其实在MySQL官方文档上已经解释过了，而且也给了演示案例（如本例）。文档中是这

说的：

INSERT sets an exclusive lock on the inserted row. This lock is an index-record lock, not a next key lock (that is, there is no gap lock) and does not prevent other sessions from inserting into he gap before the inserted row.

Prior to inserting the row, a type of gap lock called an insert intention gap lock is set. This lock signals the intent to insert in such a way that multiple transactions inserting into the same in ex gap need not wait for each other if they are not inserting at the same position within the g p. Suppose that there are index records with values of 4 and 7. Separate transactions that att mpt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks rior to obtaining the exclusive lock on the inserted row, but do not block each other because he rows are nonconflicting.

If a duplicate-key error occurs, a shared lock on the duplicate index record is set. This use of a shared lock can result in deadlock should there be multiple sessions trying to insert the same ow if another session already has an exclusive lock. This can occur if another session deletes t e row.

核心内容是：当需要进行唯一性冲突检测时，需要先加一个S锁。

这样的话，上面案例的加锁过程就不是之前推测的那样，而是像下面这样了：

session1	**session2**	**session3**
begin;delete from ld where id=1;持有id=1的record lock(X)		
	begin;insert into ld select 1,’ dkey';需要判断唯一性，检测到冲突，请求id=1的next-key lock(S)被阻塞，等待ing	
		begin;insert into ld select 1, mark';需要判断唯一性，检测到冲突，请求id=1的next-key lock(S)被阻塞，等待ing
commit;提交，释放id=1上的锁		
	后面session3检测到死锁冲突后，session2才insert成功；Query K, 1 row affected (11.82 sec)Records: 1 Duplicates: 0 Warnings: 0	
	功获取id=1的next-key lock(S)；请求id=1的record lock(X)锁；触发死锁检测，失败、回滚；ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction	

下面是另一个类似的案例：

session1	**session2**
begin;select * from d where id = 1 lock in share mode;持有id=1上的record lock(S)	
-	begin;select * from d where id = 1 lock in share mode;持有id=1上的record lock(S)
delete from d where id = 1;请求id=1上的record lock(X)，被session2阻塞了，等待中	
-	delete from d where id = 1;请求id=1上的record lock(X)，检测到死锁，失败，回滚

通过上面这两个案例，其实想要告诉大家的是：发生死锁时，不能只看现场，还得分析过程，才能知真正的原因，死锁发生的原因也并不复杂，但是得能想办法还原过程。

下面提供一个更加诡异的死锁案例，这个死锁案例出现了S GAP锁，可能从来没有见过。

```
mysql> create table testunj1 (id1 int primary key,id2 int unique key,name varchar(20));
mysql> insert into testunj1 values(1,1,'gaopeng'),(10,10,'gaopeng'),(20,20,'gaopeng');
mysql> select * from testunj1;
+-----+------+---------+
| id1 | id2  | name    |
+-----+------+---------+
|   1 |    1 | gaopeng |
|  10 |   10 | gaopeng |
|  20 |   20 | gaopeng |
+-----+------+---------+
3 rows in set (0.01 sec)
```

死锁模拟：

session1	session2	session3
begin;insert into testunj1 values(17,17,’gaopeng’);insert into testunj1 values(15,15,’gaopng’);		
	begin;insert into testunj1 values(14,15,’gaopeng’);堵塞	
		begin;insert into ld select 1,mark';需要判断唯一性，检测到冲突，请求id=1的next-key lock(S)被阻塞，等待ing
commit;提交，释放id=1上的锁		
	后面session3检测到死锁冲突后，session2才insert成功；Query K, 1 row affected (11.82 sec)Records: 1 Duplicates: 0 Warnings: 0	
功获取id=1的next-key lock(S)；请求id=1的record lock(X)锁；触发死锁检测，失败、回滚；ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction		

这时候我们看下InnoDB STATUS的输出：

```
------------------------
LATEST DETECTED DEADLOCK
------------------------
2017-09-05 05:53:59 0x7f233f088700
*** (1) TRANSACTION:
TRANSACTION 5415743, ACTIVE 20 sec inserting
mysql tables in use 1, locked 1
LOCK WAIT 4 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 1
MySQL thread id 1430, OS thread handle 139789350385408, query id 9663296 localhost root
pdate
insert into testunj1 values(14,15,'gaopeng')
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 131 page no 4 n bits 72 index id2 of table `sbtest`.`testunj1` trx id 54
5743 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000014; asc     ;;
 1: len 4; hex 80000014; asc     ;;

*** (2) TRANSACTION:
TRANSACTION 5415744, ACTIVE 6 sec inserting
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 1
MySQL thread id 1431, OS thread handle 139789358106368, query id 9663298 localhost root
pdate
```

```
insert into testunj1 values(16,17,'gaopeng')
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 131 page no 4 n bits 72 index id2 of table `sbtest`.`testunj1` trx id 54
5744 lock mode S locks gap before rec
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000014; asc     ;;
 1: len 4; hex 80000014; asc     ;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 131 page no 4 n bits 72 index id2 of table `sbtest`.`testunj1` trx id 54
5744 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000014; asc     ;;
 1: len 4; hex 80000014; asc     ;;
```

可以看到lock mode S locks gap，完结.

转载至:《两个INSERT发生死锁原因剖析》