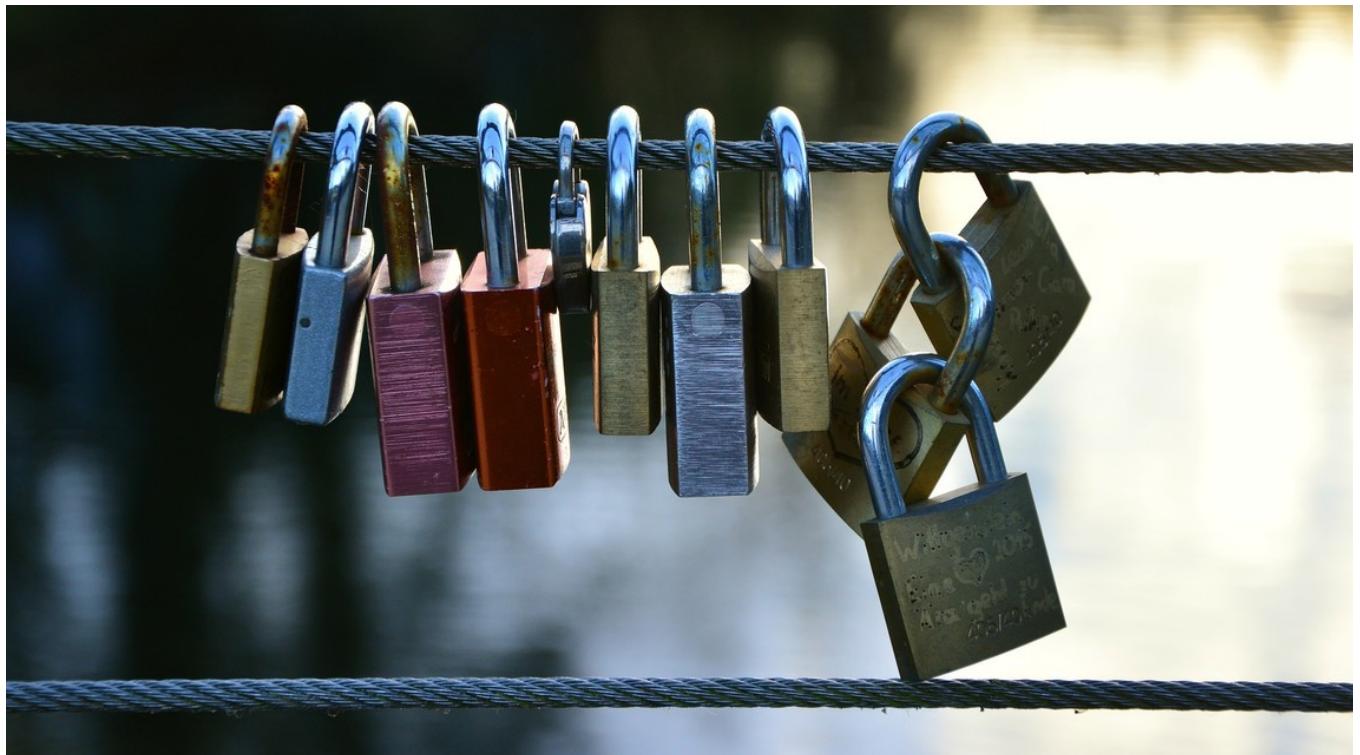


## 40 | insert语句的锁为什么这么多？

2019-02-13 林晓斌



在上一篇文章中，我提到MySQL对自增主键锁做了优化，尽量在申请到自增id以后，就释放自增锁。

因此，`insert`语句是一个很轻量的操作。不过，这个结论对于“普通的`insert`语句”才有效。也就是说，还有些`insert`语句是属于“特殊情况”的，在执行过程中需要给其他资源加锁，或者无法在申请到自增id以后就立马释放自增锁。

那么，今天这篇文章，我们就一起来聊聊这个话题。

### `insert ... select` 语句

我们先从昨天的问题说起吧。表t和t2的表结构、初始化数据语句如下，今天的例子我们还是针对这两个表展开。

```
CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `c` int(11) DEFAULT NULL,
  `d` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `c` (`c`)
) ENGINE=InnoDB;
```

```
insert into t values(null, 1,1);
insert into t values(null, 2,2);
insert into t values(null, 3,3);
insert into t values(null, 4,4);
```

```
create table t2 like t
```

现在，我们一起来看看为什么在可重复读隔离级别下，`binlog_format=statement`时执行：

```
insert into t2(c,d) select c,d from t;
```

这个语句时，需要对表t的所有行和间隙加锁呢？

其实，这个问题我们需要考虑的还是日志和数据的一致性。我们看下这个执行序列：

session A	session B
insert into t values(-1, -1,-1);	insert into t2(c,d) select c,d from t;

图1 并发insert场景

实际的执行效果是，如果**session B**先执行，由于这个语句对表t主键索引加了 $(-\infty, 1]$ 这个**next-key lock**，会在语句执行完成后，才允许**session A**的**insert**语句执行。

但如果没锁的话，就可能出现**session B**的**insert**语句先执行，但是后写入**binlog**的情况。于是，在`binlog_format=statement`的情况下，**binlog**里面就记录了这样的语句序列：

```
insert into t values(-1,-1,-1);
insert into t2(c,d) select c,d from t;
```

这个语句到了备库执行，就会把id=1这一行也写到表t2中，出现主备不一致。

## insert 循环写入

当然了，执行`insert ...select` 的时候，对目标表也不是锁全表，而是只锁住需要访问的资源。

如果现在有这么一个需求：要往表t2中插入一行数据，这一行的c值是表t中c值的最大值加1。

此时，我们可以这么写这条SQL语句：

```
insert into t2(c,d) (select c+1, d from t force index(c) order by c desc limit 1);
```

这个语句的加锁范围，就是表t索引c上的(3,4]和(4,supremum]这两个next-key lock，以及主键索引上id=4这一行。

它的执行流程也比较简单，从表t中按照索引c倒序，扫描第一行，拿到结果写入到表t2中。

因此整条语句的扫描行数是1。

这个语句执行的慢查询日志（slow log），如下图所示：

```
# Query_time: 0.000732 Lock_time: 0.000356 Rows_sent: 0 Rows_examined: 1
SET timestamp=1548852517;
insert into t2(c,d) (select c+1, d from t force index(c) order by c desc limit 1);
```

图2 慢查询日志—将数据插入表t2

通过这个慢查询日志，我们看到`Rows_examined=1`，正好验证了执行这条语句的扫描行数为1。

那么，如果我们是要把这样的一行数据插入到表t中的话：

```
insert into t(c,d) (select c+1, d from t force index(c) order by c desc limit 1);
```

语句的执行流程是怎样的？扫描行数又是多少呢？

这时候，我们再看慢查询日志就会发现不对了。

```
# Query_time: 0.000478 Lock_time: 0.000128 Rows_sent: 0 Rows_examined: 5
SET timestamp=1548852287;
insert into t(c,d) (select c+1, d from t force index(c) order by c desc limit 1);
```

图3 慢查询日志—将数据插入表t

可以看到，这时候的`Rows_examined`的值是5。

我在前面的文章中提到过，希望你都能够学会用explain的结果来“脑补”整条语句的执行过程。今天，我们就来一起试试。

如图4所示就是这条语句的explain结果。

mysql> explain insert into t(c,d) (select c+1, d from t force index(c) order by c desc limit 1);											
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	INSERT	t	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	Using temporary
1	SIMPLE	t	NULL	index	NULL	c	5	NULL	1	100.00	

图4 explain结果

从Extra字段可以看到“Using temporary”字样，表示这个语句用到了临时表。也就是说，执行过程中，需要把表t的内容读出来，写入临时表。

图中rows显示的是1，我们不妨先对这个语句的执行流程做一个猜测：如果说的是把子查询的结果读出来（扫描1行），写入临时表，然后再从临时表读出来（扫描1行），写回表t中。那么，这个语句的扫描行数就应该是2，而不是5。

所以，这个猜测不对。实际上，Explain结果里的rows=1是因为受到了limit 1的影响。

从另一个角度考虑的话，我们可以看看InnoDB扫描了多少行。如图5所示，是在执行这个语句前后查看Innodb\_rows\_read的结果。

mysql> show status like '%Innodb_rows_read%';	
Variable_name	Value
Innodb_rows_read	2242
1 row in set (0.00 sec)	
mysql> insert into t(c,d) (select c+1, d from t force index(c) order by c desc limit 1);	Query OK, 1 row affected (0.00 sec)
Records: 1 Duplicates: 0 Warnings: 0	
mysql> show status like '%Innodb_rows_read%';	
Variable_name	Value
Innodb_rows_read	2246
1 row in set (0.00 sec)	

图5 查看 Innodb\_rows\_read变化

可以看到，这个语句执行前后，Innodb\_rows\_read的值增加了4。因为默认临时表是使用Memory引擎的，所以这4行查的都是表t，也就是说对表t做了全表扫描。

这样，我们就把整个执行过程理清楚了：

1. 创建临时表，表里有两个字段c和d。

2. 按照索引c扫描表t，依次取c=4、3、2、1，然后回表，读到c和d的值写入临时表。这时，Rows\_examined=4。
  3. 由于语义里面有limit 1，所以只取了临时表的第一行，再插入到表t中。这时，Rows\_examined的值加1，变成了5。
- 也就是说，这个语句会导致在表t上做全表扫描，并且会给索引c上的所有间隙都加上共享的next-key lock。所以，这个语句执行期间，其他事务不能在这个表上插入数据。
- 至于这个语句的执行为什么需要临时表，原因是这类一边遍历数据，一边更新数据的情况，如果读出来的数据直接写回原表，就可能在遍历过程中，读到刚刚插入的记录，新插入的记录如果参与计算逻辑，就跟语义不符。
- 由于实现上这个语句没有在子查询中就直接使用limit 1，从而导致了这个语句的执行需要遍历整个表t。它的优化方法也比较简单，就是用前面介绍的方法，先insert into到临时表temp\_t，这样就只需要扫描一行；然后再从表temp\_t里面取出这行数据插入表t1。
- 当然，由于这个语句涉及的数据量很小，你可以考虑使用内存临时表来做这个优化。使用内存临时表优化时，语句序列的写法如下：

```
create temporary table temp_t(c int,d int) engine=memory;
insert into temp_t (select c+1, d from t force index(c) order by c desc limit 1);
insert into t select * from temp_t;
drop table temp_t;
```

## insert 唯一键冲突

前面的两个例子是使用insert ...select的情况，接下来我要介绍的这个例子就是最常见的insert语句出现唯一键冲突的情况。

对于有唯一键的表，插入数据时出现唯一键冲突也是常见的情况了。我先给你举一个简单的唯一键冲突的例子。

session A	session B
insert into t values(10,10,10);	
begin; insert into t values(11,10,10); (Duplicate entry '10' for key 'c')	
	insert into t values(12,9,9); (blocked)

图6 唯一键冲突加锁

这个例子也是在可重复读（repeatable read）隔离级别下执行的。可以看到，**session B**要执行的**insert**语句进入了锁等待状态。

也就是说，**session A**执行的**insert**语句，发生唯一键冲突的时候，并不只是简单地报错返回，还在冲突的索引上加了锁。我们前面说过，一个**next-key lock**就是由它右边界值定义的。这时候，**session A**持有索引c上的(5,10]共享**next-key lock**（读锁）。

至于为什么要加这个读锁，其实我也没有找到合理的解释。从作用上来看，这样做可以避免这一行被别的事务删掉。

这里[官方文档](#)有一个描述错误，认为如果冲突的是主键索引，就加记录锁，唯一索引才加**next-key lock**。但实际上，这两类索引冲突加的都是**next-key lock**。

备注：这个bug，是我在写这篇文章查阅文档时发现的，已经[发给官方](#)并被verified了。

有同学在前面文章的评论区问到，在有多个唯一索引的表中并发插入数据时，会出现死锁。但是，由于他没有提供复现方法或者现场，我也无法做分析。所以，我建议你在评论区发问题的时候，尽量同时附上复现方法，或者现场信息，这样我才好和你一起分析问题。

这里，我就先和你分享一个经典的死锁场景，如果你还遇到过其他唯一键冲突导致的死锁场景，也欢迎给我留言。

	session A	session B	session C
T1	begin; insert into t values(null, 5,5);		
T2		insert into t values(null, 5,5);	insert into t values(null, 5,5);
T3	rollback;		(Deadlock found)

图7 唯一键冲突—死锁

在**session A**执行**rollback**语句回滚的时候，**session C**几乎同时发现死锁并返回。

这个死锁产生的逻辑是这样的：

1. 在T1时刻，启动**session A**，并执行**insert**语句，此时在索引c的c=5上加了记录锁。注意，这个索引是唯一索引，因此退化为记录锁（如果你的印象模糊了，可以回顾下[第21篇文章](#)介绍的加锁规则）。

- 在T2时刻，session B要执行相同的insert语句，发现了唯一键冲突，加上读锁；同样地，session C也在索引c上，c=5这一个记录上，加了读锁。
- T3时刻，session A回滚。这时候，session B和session C都试图继续执行插入操作，都要加上写锁。两个session都要等待对方的行锁，所以就出现了死锁。

这个流程的状态变化图如下所示。

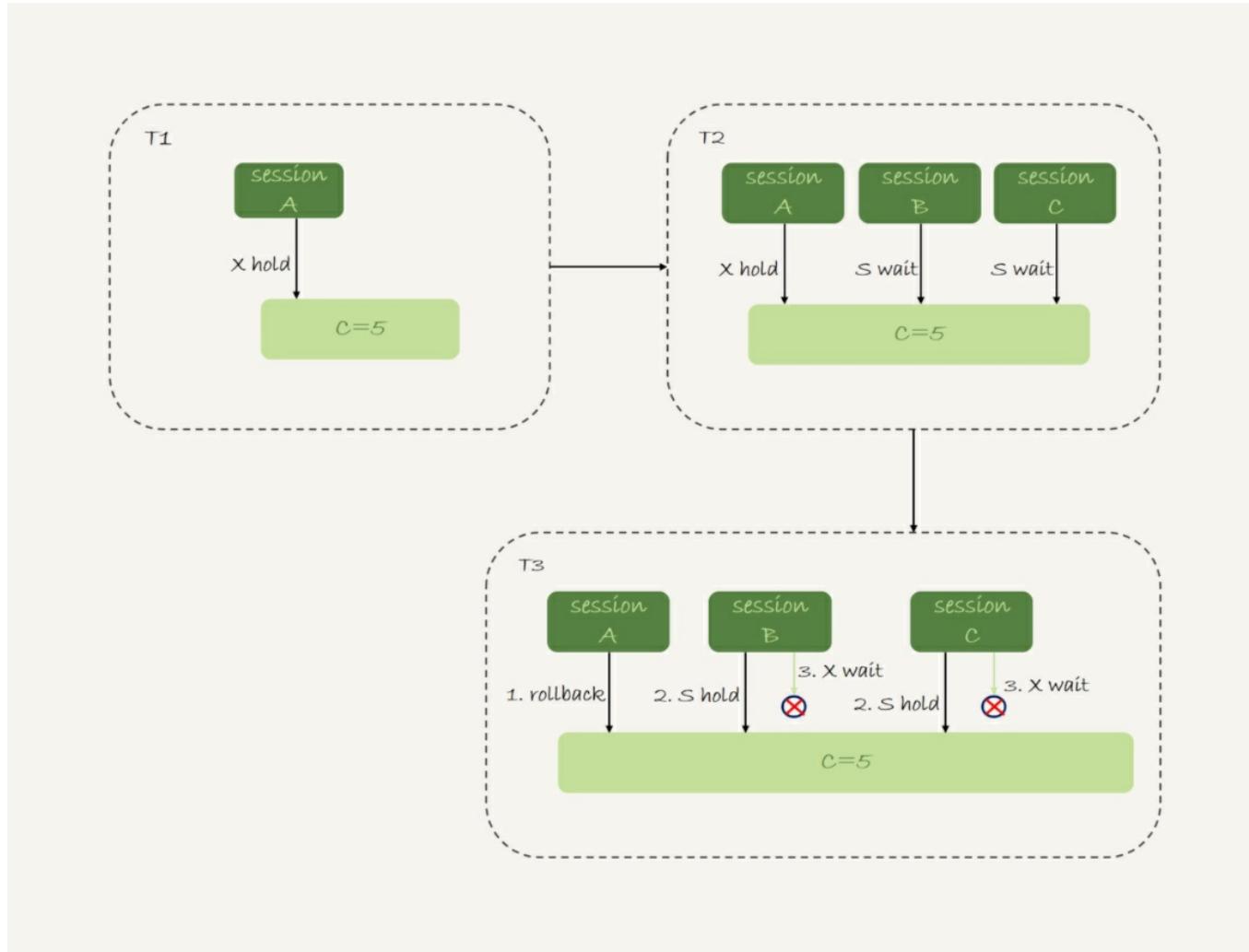


图8 状态变化图—死锁

### insert into ... on duplicate key update

上面这个例子是主键冲突后直接报错，如果是改写成

```
insert into t values(11,10,10) on duplicate key update d=100;
```

的话，就会给索引c上[5,10]加一个排他的next-key lock（写锁）。

**insert into ...on duplicate key update** 这个语义的逻辑是，插入一行数据，如果碰到唯一键约束，就执行后面的更新语句。

注意，如果有多个列违反了唯一性约束，就会按照索引的顺序，修改跟第一个索引冲突的行。

现在表t里面已经有了(1,1,1)和(2,2,2)这两行，我们再来看看下面这个语句执行的效果：

```
mysql> insert into t values(2,1, 100) on duplicate key update d=100;
Query OK, 2 rows affected (0.00 sec)
mysql> select * from t where id<=2;
+---+---+---+
| id | c   | d   |
+---+---+---+
| 1  | 1   | 1   |
| 2  | 2   | 100 |
+---+---+---+
2 rows in set (0.00 sec)
```

图9 两个唯一键同时冲突

可以看到，主键id是先判断的，MySQL认为这个语句跟id=2这一行冲突，所以修改的是id=2的行。

需要注意的是，执行这条语句的affected rows返回的是2，很容易造成误解。实际上，真正更新的只有一行，只是在代码实现上，insert和update都认为自己成功了，update计数加了1，insert计数也加了1。

## 小结

今天这篇文章，我和你介绍了几种特殊情况下的insert语句。

insert ...select是很常见的在两个表之间拷贝数据的方法。你需要注意，在可重复读隔离级别下，这个语句会给select的表里扫描到的记录和间隙加读锁。

而如果insert和select的对象是同一个表，则有可能会造成循环写入。这种情况下，我们需要引入用户临时表来做优化。

insert语句如果出现唯一键冲突，会在冲突的唯一值上加共享的next-key lock(S锁)。因此，碰到由于唯一键约束导致报错后，要尽快提交或回滚事务，避免加锁时间过长。

最后，我给你留一个问题吧。

你平时在两个表之间拷贝数据用的是什么方法，有什么注意事项吗？在你的应用场景里，这个方法，相较于其他方法的优势是什么呢？

你可以把你的经验和分析写在评论区，我会在下一篇文章的末尾选取有趣的评论来和你一起分析。感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。

## 上期间题时间

我们已经在文章中回答了上期问题。

有同学提到，如果在`insert ...select`执行期间有其他线程操作原表，会导致逻辑错误。其实，这是不会的，如果不加锁，就是快照读。

一条语句执行期间，它的一致性视图是不会修改的，所以即使有其他事务修改了原表的数据，也不会影响这条语句看到的数据。

评论区留言点赞板：

@长杰 同学回答得非常准确。

The image is a promotional graphic for a MySQL course. It features a portrait of a man with glasses and a black shirt, standing with his arms crossed. To his left is the title 'MySQL 实战 45讲' and a subtitle '从原理到实战，丁奇带你搞懂 MySQL'.

林晓斌 网名丁奇  
前阿里资深技术专家

新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

精选留言



huolang

8

老师，死锁的例子，关于sessionA拿到的c=5的记录锁，sessionB和sessionC发现唯一键冲突会加上读锁我有几个疑惑：

1. sessionA拿到的c=5的记录锁是写锁吗？
2. 为什么sessionB和sessionC发现唯一键冲突会加上读锁？
3. 如果sessionA拿到c=5的记录所是写锁，那为什么sessionB和sessionC还能加c=5的读锁，写锁和读锁不应该是互斥的吗？
4. sessionA还没有提交，为什么sessionB和sessionC能发现唯一键冲突？

2019-02-13

## 作者回复

1. 是的
2. 这个我觉得是为了防止这个记录再被删除（不过这个理由不是很硬，我还没有找到其他解释
3. 互斥的，所以这两个语句都在等待。注意**next-key lock**是由间隙锁和记录锁组成的哦， 间隙锁加成功了的。好问题。
4. 还没有提交，但是这个记录已经作为最新记录写进去了，复习一下08篇哈

2019-02-14



老杨同志

4

课后问题：

我用的最多还是**insert into select**。如果数量比较大，会加上**limit 100,000**这种。并且看看后面的**select**条件是否走索引。缺点是会锁**select**的表。方法二：导出成**excel**，然后拼**sql**成**insert into values(),(),()**的形式。方法3，写类似淘宝调动的定时任务，任务的逻辑是查询**100**条记录，然后多个线程分到几个任务执行，比如是个线程，每个线程**10**条记录，插入后，在查询新的**10**条记录处理。

2019-02-13

## 作者回复

0

2019-02-14



sonic

3

你好，

我想问下文章中关于为什么需要创建临时表有这一句话：

如果读出来的数据直接写回原表，就可能在遍历过程中，读到刚刚插入的记录，新插入的记录如果参与计算逻辑，就跟语义不符。

我的疑问是：既然隔离级别是可重复读，照理来说新插入的记录应该不会参与计算逻辑呀。

2019-02-14

## 作者回复

可重复读隔离级别下，事务是可以看到自己刚刚修改的数据的，好问题

2019-02-16



滔滔

2

老师，之前提到的一个有趣的问题"A、B两个用户，如果互相喜欢，则成为好友。设计上是有两张表，一个是**like**表，一个是**friend**表，**like**表有**user\_id**、**liker\_id**两个字段，我设置为复合唯一索引即**uk\_user\_id\_liker\_id**。语句执行顺序是这样的：

以A喜欢B为例：

1、先查询对方有没有喜欢自己（B有没有喜欢A）

`select * from like where user_id = B and liker_id = A`

2、如果有，则成为好友

`insert into friend`

3、没有，则只是喜欢关系

`insert into like`"，这个问题中如果把**select**语句改成"当前读"，则当出现A,B两个人同时喜欢对方

的情况下，是不是会出现由于"当前读"加的gap锁导致后面insert语句阻塞，从而发生死锁？

2019-02-13

### 作者回复

好问题

这种情况下一般是造成锁等待，不会造成死锁吧！

2019-02-14



夹心面包

2

1 关于insert造成死锁的情况,我之前做过测试,事务1并非只有insert,delete和update都可能造成死锁问题,核心还是插入唯一值冲突导致的.我们线上的处理办法是 1 去掉唯一值检测 2减少重复值的插入 3降低并发线程数量

2 关于数据拷贝大表我建议采用pt-archiver,这个工具能自动控制频率和速度,效果很不错,提议在低高峰期进行数据操作

2019-02-13

### 作者回复

[], 这两点都是很有用的建议

2019-02-13



王伯轩

1

老师你好,去年双11碰到了dbcrash掉的情况.至今没有找到答案,心里渗得慌.老师帮忙分析下. 我是一个开发,关于db的知识更多是在应用和基本原理上面,实在是找不到原因. 我也搜了一些资料 感觉像是mysql的bug,不过在其buglist中没有找到完全一致的, 当然也可能是我们业务也许导致库的压力大的原因.

应用端看到的现象是db没有响应, 应用需要访问db的线程全部僵死.db表现是hang住, 当时的诊断日志如下, 表面表现为一直获取不到latch锁(被一个insert线程持有不释放) <https://note.youdao.com/ynoteshare1/index.html?id=1771445db3ff1e08cbdd8328ea6765a7&type=note#/> 隔离级别是rr

同样的crash双11当天后面又出现了一次(哭死),

都是重启数据库解决的,

后面应用层面做了一样优化,没有再crash过, 优化主要如下:

1.减小读压力, 去除一些不必要的查询,

2.优化前, 有并发事务写和查询同一条数据记录, 即事务a执行insert尚未提交, 事务b就来查询(快照读), 优化后保证查询时insert事务已经提交

2019-02-19

### 作者回复

这就是压力太大了。. 一般伴随着ioutil很大, 语句执行特别慢, 别的语句就被堵着等锁, 等超时就自己crash

2019-02-19



nhanzhena-好客旅游网

1



循环插入数据，然后拿着刚刚插入的主键id，更新数据。请问怎么提高这个情况的效率

2019-02-15

| 作者回复

insert以后

select last\_insert\_id;

再update，

只能这么做啦

如果要快一些，可能可以考虑减少交互，比如写成存储过程

2019-02-16



伟仔\_Hoo

0

老师，看到您的回复，当select c+1, d from t force index(c) order by c desc limit 1;这条语句单独执行是会在c索引上加[4,sup]这个next key lock，于是我进行了尝试

sessionA:

begin;

select c+1, d from t3 force index(c) order by c desc limit 1;

sessionB:

insert into t3 values(5, 5, 5);

结果是，sessionB插入成功，是不是我哪里理解错了？我的版本是5.7.23

2019-03-15

| 作者回复

session A的select语句没有加 for update 或者 lock in share mode ?

2019-03-16



猫小妖的尾巴

0

老师，我们的业务中有用到insert ...on duplicate key update导致死锁的情况，表是有唯一索引，DBA那边的解释是有唯一索引的insert需要两把锁，事务1先申请X锁成功，然后申请S锁，但是事务2正在申请X锁，与事务1的S锁冲突，系统决定回滚事务2，然后我就改成先查询存在直接update不存在再用原来的逻辑，不过我感觉还是不太明白，你可以解释一下吗

2019-03-10



涵涵妈 lillian

0

老师，能帮忙看下这个死锁记录吗？对于duplicate key插入有什么阻止的好方法？**LATEST DETECTED DEADLOCK**

---

190222 8:37:45

\*\*\* (1) TRANSACTION:

TRANSACTION 16FEC1AE, ACTIVE 0 sec inserting

mysql tables in use 1, locked 1

LOCK WAIT 6 lock struct(s), heap size 1248, 3 row lock(s)

MySQL thread id 169973, OS thread handle 0x2ba0fa040700, query id 41915315 10.45.133.181

W59FFHKU

INSERT INTO resource (

Id

, Name

, Date

, User

) VALUES (99127, 'RS\_2098185e367d11e9878202a98a7af318', '', 'JR')

\*\*\* (1) WAITING FOR THIS LOCK TO BE GRANTED:

RECORD LOCKS space id 78 page no 71 n bits 160 index `PRIMARY` of table `resource` trx id 16FEC1AE lock\_mode X insert intention waiting

Record lock, heap no 1 PHYSICAL RECORD: n\_fields 1; compact format; info bits 0

0: len 8; hex 73757072656d756d; asc supremum;;

\*\*\* (2) TRANSACTION:

TRANSACTION 16FEC1AF, ACTIVE 0 sec inserting

mysql tables in use 1, locked 1

6 lock struct(s), heap size 1248, 3 row lock(s)

MySQL thread id 169996, OS thread handle 0x2ba0ffec2700, query id 41915317 10.45.133.181

W59FFHKU

INSERT INTO resource (

Id

, Name

, Date

, User

) VALUES (99125, 'RS\_2098b778367d11e9878202a98a7af318', '', 'JR')

\*\*\* (2) HOLDS THE LOCK(S):

RECORD LOCKS space id 78 page no 71 n bits 160 index `PRIMARY` of table `resource` trx id 16FEC1AF lock mode S

Record lock, heap no 1 PHYSICAL RECORD: n\_fields 1; compact format; info bits 0

0: len 8; hex 73757072656d756d; asc supremum;;

\*\*\* (2) WAITING FOR THIS LOCK TO BE GRANTED:

RECORD LOCKS space id 78 page no 71 n bits 160 index `PRIMARY` of table `resource` trx id 16FEC1AF lock\_mode X insert intention waiting

Record lock, heap no 1 PHYSICAL RECORD: n\_fields 1; compact format; info bits 0

0: len 8; hex 73757072656d756d; asc supremum;;

\*\*\* WE ROLL BACK TRANSACTION (2)

2019-03-10



涵涵妈 lillian

0

老师，重复主键插入冲突是否推荐insert ignore方法？

2019-03-09

| 作者回复

这个取决于业务需求，如果是明确会存在这样的情况，并且可以忽略，是可以这么用的

2019-03-09



轻松的鱼

0

老师好，想请教一下死锁的例子中：

1. 在 session A rollback 前, session B/C 都因为唯一性冲突申请了 S Next-key lock, 但是被 session A 的 X but not gap lock 阻塞;

2. 在 session A rollback 后, session B/C 顺利获得 S Next-key lock, 并且都要继续进行插入, 这时候我认为是因为插入意向锁 (LOCK\_INSERT\_INTENTION) 导致的死锁, 因为插入意向锁会被 gap lock 阻塞, 造成了相互等待。还没有进入到记录 X lock。

不知道我分析的对不对?

2019-03-06



张永志

0

对主键插入加读锁的个人理解, 两个会话insert同样记录, 在没有提交情况下, insert主键加读锁是为了避免第一个会话回滚后, 第二个会话可以正常执行; 第一个会话提交后, 第二个会话再报错。

2019-02-28

| 作者回复

是为了实现这个目的, 是吧!

2019-02-28



Mr.Strive.Z.H.L

0

老师您好:

关于文中的锁描述有所疑惑。

文中出现过 共享的next-key锁 和 排他的next-key锁。

我们知道next-key是由 gap lock 和 行锁组成的。

我一直以来的认知是 gap lock都是s锁, 没有x锁。

而行锁有s锁和x锁。

比如 select.....lock in share mode, 行锁是s  
锁。

比如select.....for update, 行锁就是x锁。

但是gap lock 始终是s锁。

文中直接描述next-key lock是排他的, 总让我认为gap lock和行锁都是x锁。

不知道我理解得对不对?

2019-02-27

| 作者回复

是这样的, gap lock是无所谓S还是X的。

但是record lock 有。

## Gap lock + 排他的record 就称作 排他的next-key lock 吧

2019-02-27



滔滔

0

老师，`select c+1, d from t force index(c) order by c desc limit 1;`这条语句如果单独执行，是会对表t进行全表加锁，还是只加`(3,4],(4,sup]`这两个next key锁。还有一个问题，这里为什么要加`force index(c)`，不加会是怎样的效果呢？

2019-02-24

| 作者回复

`(4,sup]`

以防优化器不走索引，影响我们结论（比如数据量比较小的时候）

2019-02-25



发条橙子。

0

老师，年后过来狂补课程了哈哈，看到老师的bug留言已经被fix掉准备在最新版本发布了呢。

这里我有一个疑问，我之前以为只有更新的时候才会加锁，参考前面的文章，innodb要先扫描表中数据，被扫描到的行要加锁。

或者我们执行 `select` 的时候手动加上 排他锁 或者 共享锁，也会锁住。

这里老师讲到如果索引唯一键冲突，innodb为了做处理加了 `next_key lock (S)` 这个可以理解。

`insert .. select` 也是因为有 `select` 索引会加锁 也可以理解

问题：

图7那个死锁的案例，`session A` 的时候 只是执行了 `insert` 语句，执行 `insert` 的时候也没有`select`之类的，为什么也会在索引c上加个锁，是什么时候加的呢？？？是 `insert` 语句有索引的话都会给索引加锁么？？

2019-02-23

| 作者回复

不是都会，是在要写入的时候，发现有主键冲突，才会加上这个next-key lock的锁

2019-02-23



滔滔

0

老师，有个问题`insert into ... on duplicate key update`语句在发生冲突的时候是先加next key读锁，然后在执行后面的`update`语句时再给冲突记录加上写锁，从而把之前加的next key读锁变成了写锁，是这样的吗？

2019-02-21

### 作者回复

不是，发现冲突直接加的就是写锁

2019-02-24



王伯轩

0

内存锁 大大计划讲下么,实际中碰到内存锁被持有后一直不释放导致db直接crash掉

2019-02-18

### 作者回复

这个系列里没讲到了

这种我碰到比较多的是io压力特别大，导致有的事务执行不下去，但是占着锁

然后其他事务就拿不到锁，有一个600计时，超过就crash了

2019-02-18



信信

0

老师好，文中提到： `insert into t2(c,d) (select c+1, d from t force index(c) order by c desc limit 1` )的加锁范围是表 t 索引 c 上的 `(4,supremum]` 这个 next-key lock 和主键索引上 `id=4` 这一行。

可是如果我把表t的id为3这行先删除，再执行这个`insert...select`，那么别的会话执行`insert into t values(3,3,3)`会被阻塞，这说明4之前也是有间隙锁的？

另外，`select c+1, d from t force index(c) order by c desc limit 1 for update` 是不是不能用作等值查询那样分析？因为如果算等值查询，根据优化1是没有间隙锁的。

2019-02-17

### 作者回复

你说的对，这里其实是“向左扫描”，加锁范围应该是`(3,4]` 和 `(4, supremum]`。

0

2019-02-17



Justin

0

插入意向锁的gal lock和next key lock中的 gaplock互斥吗？

2019-02-15

### 作者回复

额，

这里我们要澄清一下哈

只有一个gap lock，就是 `next key lock = gap lock + record lock;`

我们说一个`insert`语句如果要插入一个间隙，而这个间隙上有`gap lock`的话，`insert`语句会被堵住，这个被堵住的效果，实现机制上是用插入意向锁和`gap lock`相互作用来实现的。

`gap lock`并不属于插入意向锁的一部分，就没有“插入意向锁的gal lock”这个概念哈

2019-02-16