30 | 答疑文章（二）：用动态的观点看加锁

**加锁规则 ： 两个“原则”和 两个 “优化”和一个“bug”**

原则1：加锁的基本单位next-key lock.是前开后闭区间的

原则2：查找过程中访问到的对象才会加锁

优化1：索引上的等值查询，给唯一索引加锁的时，next-key lock 退化为行锁

优化2：索引上的等值查询，向右遍历且最后一个不满足等值条件时，next-key lock 退化为间隙锁

一个bug:唯一索引的范围查找会访问到不满足条件的第一个值为止

创建表t

CREATE TABLE `t` (

`id` int(11) NOT NULL,

`c` int(11) DEFAULT NULL,

`d` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `c` (`c`)

) ENGINE=InnoDB;

insert into t values(0,0,0),(5,5,5),

(10,10,10),(15,15,15),(20,20,20),(25,25,25);

**不等号条件里的等值查询**

分析这个查询的加锁范围

begin;

select \* from t where id>9 and id<12 order by id desc for update;

利用加锁规则，加锁范围是主键索引上的(0,5]、(5,10]、和(10,15)，也就是说id=15这一行，并没有被加上行锁，为什么呢？

加锁的基本单位是next-key lock都是前开后闭区间，这里用到优化规则2，即索引上的等值查询，向右遍历的时候id=15不满足条件，所以next-key lock退化为了间隙锁(10,15)

查询语句where条件是大于和小于号，这里的“等值查询”是从哪里来的呢？

加锁的动作是发证在执行的过程中的，分析加锁时，要从索引上的数据结构开始，过程拆解

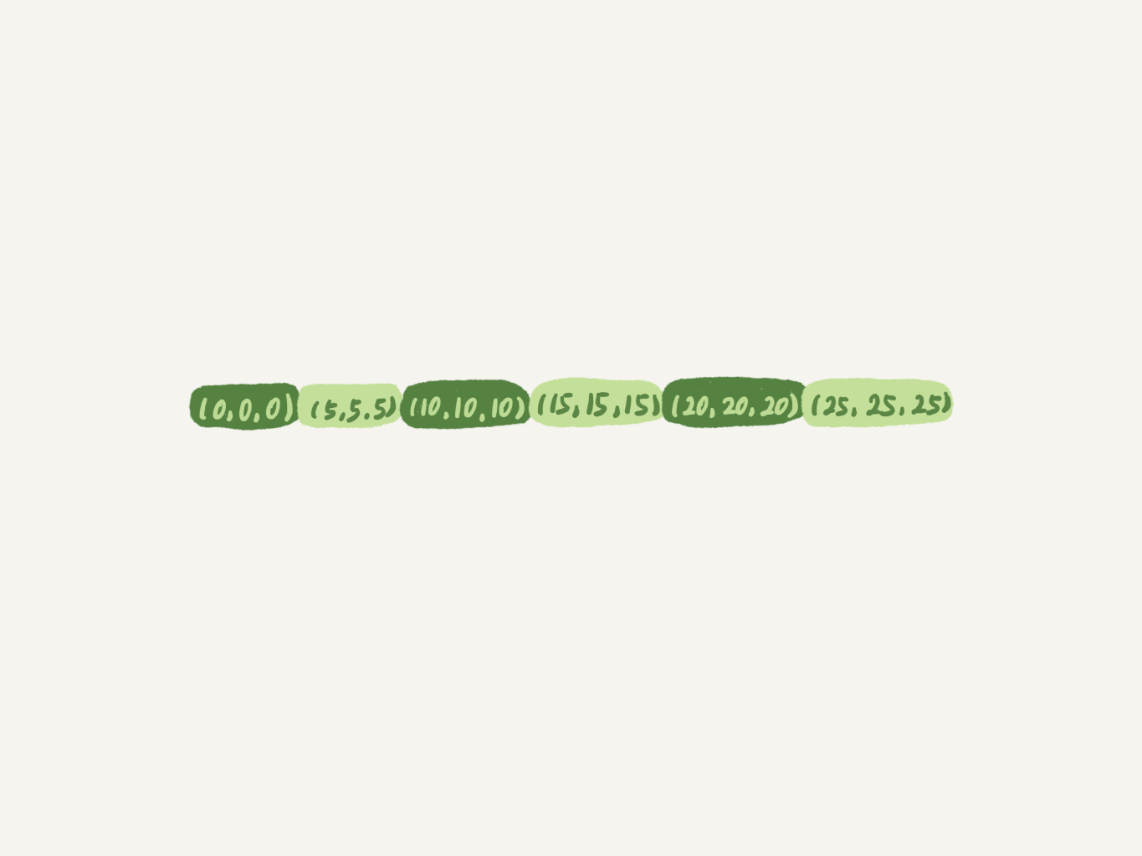
索引id 示意图

图 1 索引id示意图

1. 首先这个查询语句 order by id desc,要拿到满足条件的所有行，优化器必须先找到“第一个id<12的值”
2. 这个过程是通过索引数的搜索过程得到的，在引擎内部，其实是要找到id=12的这个值，只是最终没有找到，但找到了(10,15)这个间隙
3. 然后向左遍历，在遍历过程中，就不是等值查询了，会扫描到id=15这一行，所以会加一个next-key lock (0,5]

在执行过程中，通过树搜索的方式定位记录的时候，用的是“等值查询”的方法

**等值查询的过程**

与上面这个例子对应的，下面这个语句的加速范围是什么？

begin;

select id from t where c in(5,20,10) lock in share mode;

这条语句里用的是 in, 查看explain结果

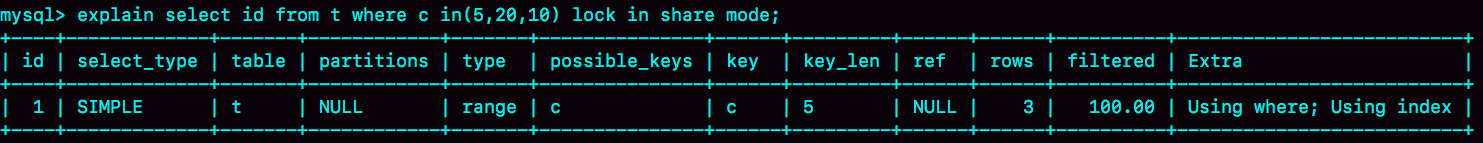


图 2 in 语句的explain 结果

这条in语句使用了索引c 并且rows=3,说明这个三个值都是通过B+树搜索定位的

在查找c-5时，先锁住(0,5]，但是因为c不是唯一索引，为了确认还有没有别的记录c=5，就要向右遍历，找到c=10才确认没有了，这个过程满足优化2，所以加了间隙锁(5,10)

同样，执行c=10这个逻辑时，加锁范围是(5,10]和(10,15),执行c=20这个逻辑的时候，加锁的范围(15,20]和(20,25)

这条语句在索引c上加的三个记录锁的顺序是：先加c=5的记录锁，在加c=10的记录锁，最后加c=20的记录锁

这个加锁范围，不就是从(5,25)中去掉c=15的行锁吗？为什么这么麻烦的分段说？

因为：这些锁是“执行过程中一个个加的”，而不是一次性加上去的，理解了这个加速过程，就可以分析下面的例子中死锁问题了

如果同时有另外一个语句，是这么写的

select id from t where c in(5,20,10) order by c desc for update;

此时的加锁范围，又是什么呢？

间隙锁是不互锁的，但是这两条语句都会在索引c上的c=5,10,20这三行记录上加锁记录

这里需要注意，由于语句是order by c desc ，这三个记录锁的加锁顺序，是先锁c=20,然后c=10,最后是c=5.

也就是，这两条语句要加锁相同的资源，但是加锁顺序相反，当这两条语句并发执行的时候，就可能出现死锁

**怎么看死锁？**

图3是在出现死锁后，执行show engine innodb status 命令得到部分输出，这个命令会输出很多信息，有一节LATESTEDTECTED DEADLOCK 这是记录的最后一次死锁信息

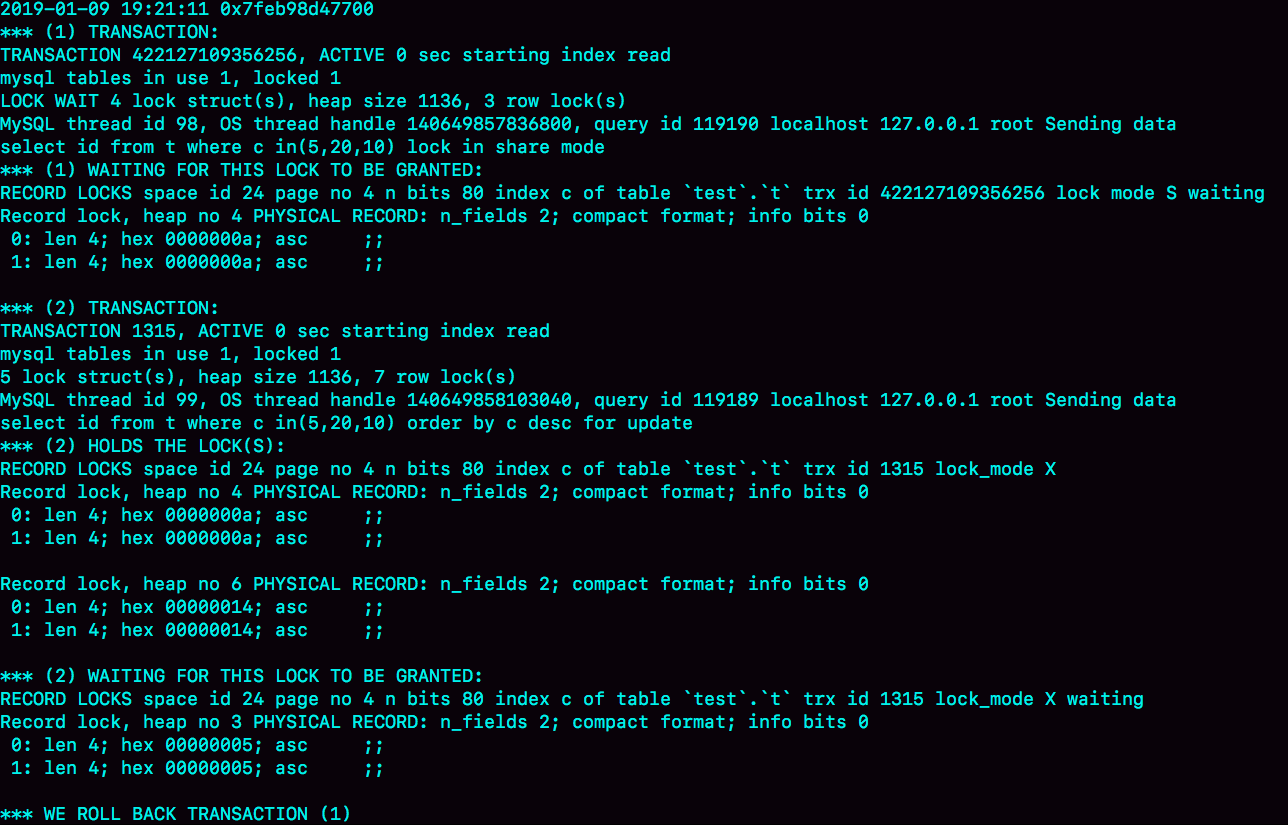


图3 死锁现场

图中几个关键信息

1. 这个结果分三部分
2. TRANSACTION 是第一个事务信息
3. TRANAACTION 是第二个事务信息

WE ROLL BACK TRANSACTION(1),是最终的处理结果，表示回滚了第一个事务

1. 第一个事务中的信息

·WAITING FOR THIS LOCK TO BE GRANTED表示是这个事务在等待的锁信息

·Index c of table ‘test’ t，说明在等待的是表t的索引 c上面的锁

·Lock mode S waiting表示这个语句要自己加一个读锁，当前的状态是等待中

·Recored lock说明这是一个记录锁

·n\_fileds 2表示这个记录是两列，也就是字段c和主键id

·0:len 4;hex 0000000a;axc；是第二个字段，也就是主键id,也是10

·这两行里面的asc表示的是，接下来要打印出值里面的“可打印字符”，但10不是可打印字符，因此就显示空格

·第一个事务信息就只显示出了等锁的状态，在等待（c=10,id=10）这一行的锁

·既然出现死锁，就表示别的事务也占了锁，

1. 第二个事务显示的信息要多一些

·“HOLDS THE LOCK(S)”用来显示这个事务持有那些锁

·index c of table ‘text’ t表示锁是在表t的索引c上

·hex000000a 和hex 00000014表示这个事务持有c=10和c=20这两个记录的锁

·WAITING FOR THIS LOCK TO BE GRANTED,表示在等（c=5,id=5）这个记录锁

从上面的信息中

1. “lock in share mode”的这条语句，持有c=5的记录锁，在等待c=10的锁
2. “for update”这个语句，持有c=10和c=10的记录锁，在等待c=5的锁

因此导致了死锁，这里可以得到两个结论

1. 由于锁是一个个加的，要避免死锁，对同一组资源，要按照尽量相同的顺序访问
2. 在发生死锁的时刻，for update这条语句占用资源更多，回滚成本更大，所以InnoDB选择了回滚成本更小的lock in share mode语句，来回滚。

**怎么看待死锁**

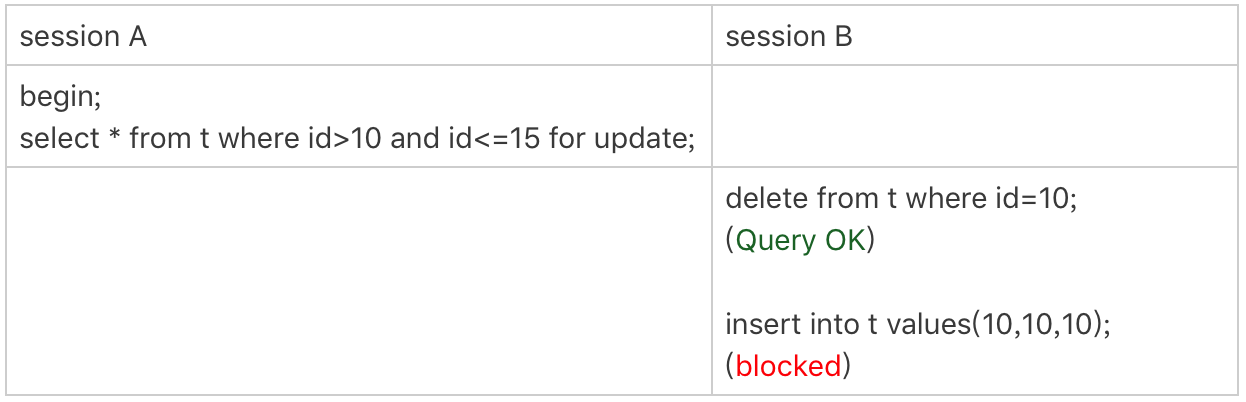


图4 delete 导致间隙变化

由于sessionA并没有锁住c=10这个记录，所以sessionB 删除id=10这一行是可以的，但是之后，sessionB再想insert id=10这一行回去就不行了

此时，show engine innodb status结果，看看提示，锁信息是在这个命令输出结果的TRANSACTIONS这一节

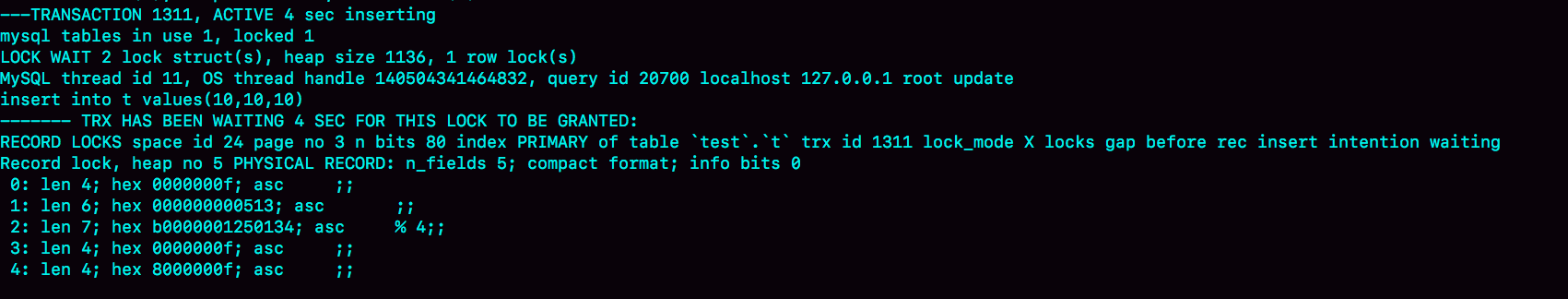


图 5 锁等待信息

几个关键信息

1.index PRIMARY of table ‘test’.t 表示这个语句被锁住是因为表t主键上的某锁

2.lock\_mode X locks gap before rec insert intention waiting 这里有几个信息

·insert intention表示当前线程准备插入一个记录，这是一个插入意向锁。为了便于理解，可以认为它就是这个插入动作本身

·gap before rec 表示这是一个间隙锁，而不是记录锁

3.这个gap实在那个记录之前呢？接下来的0-4这5行的内容就是这个记录的信息

4.n\_fileds 5表示这个记录有5列

·0：len 4 ;hex0000000f ;asc ;;第一列是主键id字段，十六进制f就是id=15,所以，这个间隙锁就是id=15之前的，因为id=10已经不存在了，它表示的就是（5,15）。

·1：len 6;hex00000513;asc;;第二列长度是6字节的事务id,表示最后修改该这一行的是trx\_id为1299的事务

·2：len 7:hex 0000001250134;asc % 4;;第三列长度为7字节的回滚段信息，这里的asc后面有显示内容（%和4）这是因为刚好这个字节是可打印字符

·后面两列是c和d的值，都是15

因此，由于delete操作把id=10这一行删掉了，原来的两个间隙（5,10）、（10,15）就变成了一个(5,15)

可以联合起来思考一下这两个现象之间的关联

1. sessionA执行完select语句后，什么都没做，但它加锁的范围突然“变大”了
2. 第21篇课后思考题，当执行select \* from t where c>=15 and c<=20 order by c desc lock in share mode，想左扫描到c=10时，要把(5,10]锁起来。

也就是，所谓“间隙”其实根本就是由“这个间隙右边的那个记录”定义的

**UPDATE例子**

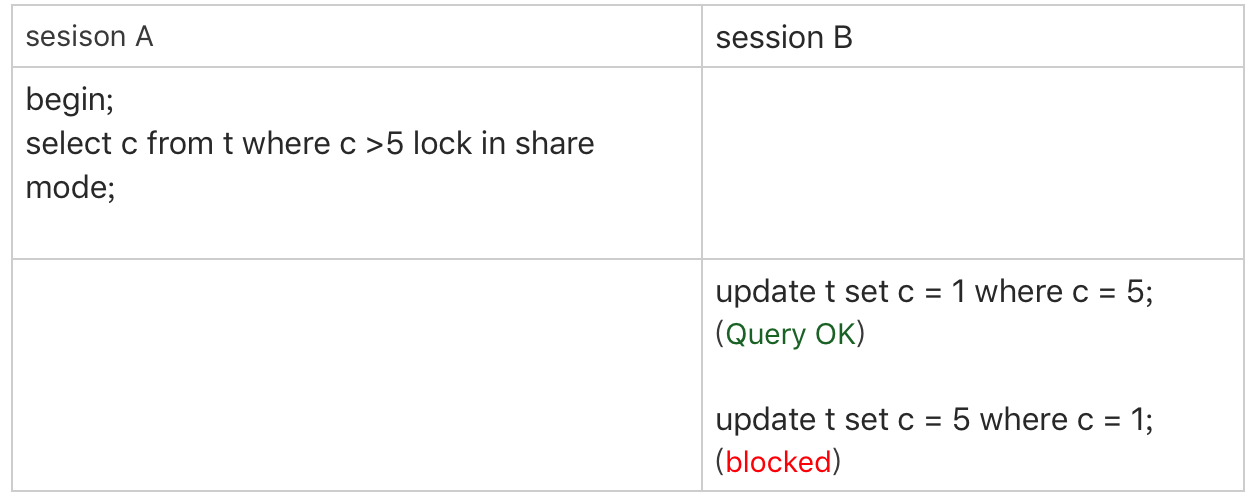


图 6 update 的例子

Session A加锁范围是索引c上的(5,10]、(10,15]、(15,20]、(20,25]和(25,supernum]

注意：根据c> 5查到的第一个记录是c=10,因此不回加(0,5]这个next-key lock

之后session B 的第一个update语句，要把c=5改成c=1可以理解为两步

1. 插入(c=1,id=5)这个纪录
2. 删除(c=5,id=5)这个纪录

按照上一节说的，索引c上(5,10)间隙是有这个间隙右边的记录，也就是c=10定义的，所以通过这个操作，session A的加锁范围变成了图7所示的样子

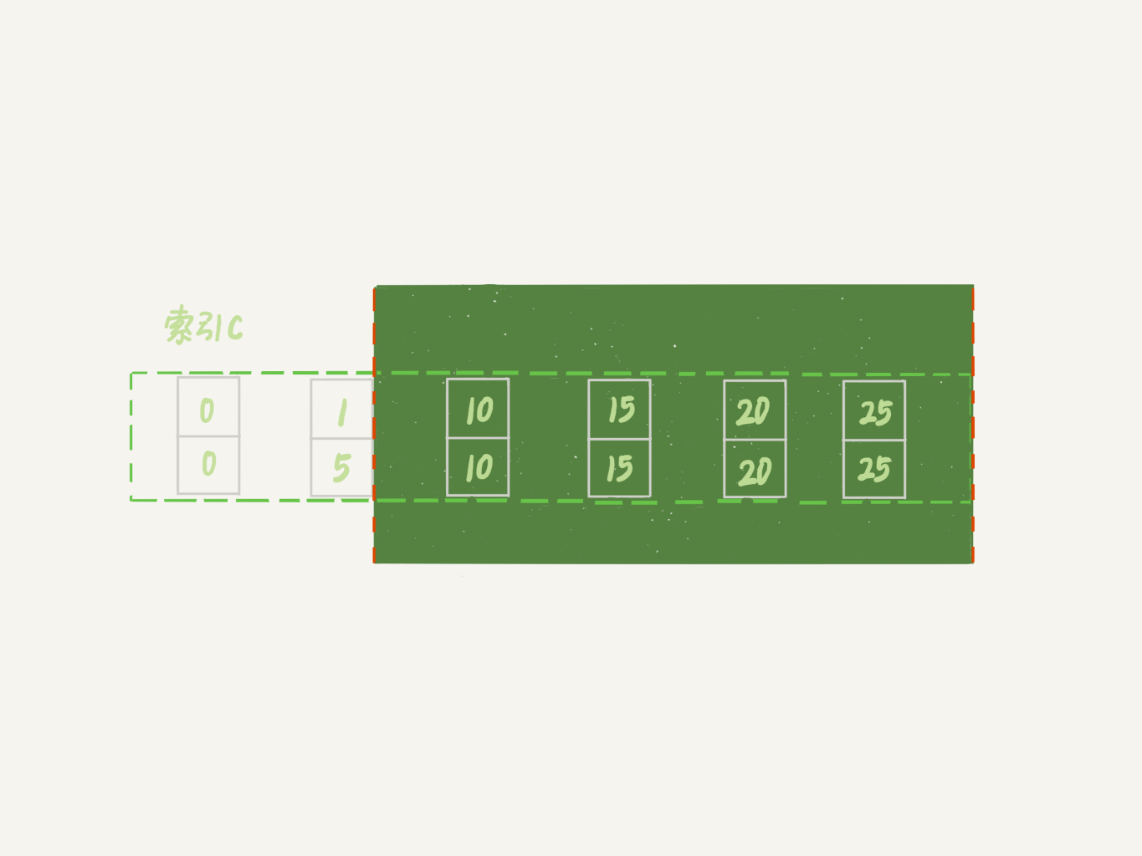


图 7 sessionB 修改后，sessionA的加锁范围

接下来sessionB要执行update t set c = 5 where c =1 这个语句，一样可以拆成2步

1. 插入(c=5,id=5)这个记录
2. 删除(c=1,id=5)这个记录

第一步试图在已经加了间隙锁(1,10)中插入数据，所以就被堵住了

思考题

一个空表有间隙吗？这个间隙是由谁定义的？你怎么验证这个结论？