# 21\_为什么我只改一行的语句，锁这么多

加锁规则有一下前提

1. MySQL版本 5.x系列 <=5.7.24 ,8.0系列<=8.0.13

**加锁规则**

1. **原则1：加锁基本单位是next-key lock,是前开后闭区间**
2. **原则2：查找过程中访问到的对象才会加锁**
3. **优化1：索引上的等值查询，给唯一索引加锁的时候，next-key lock退化为行锁**
4. **优化2：索引上的等值查询，向右遍历时且最后一个值不满足等值条件的时候，next-key lock 退化为间隙锁**
5. **一个bug:唯一索引上的范围查找会访问到不满足条件的第一个值为止**

建表和初始化语句

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);

**案例一 ：等值查询间隙锁**

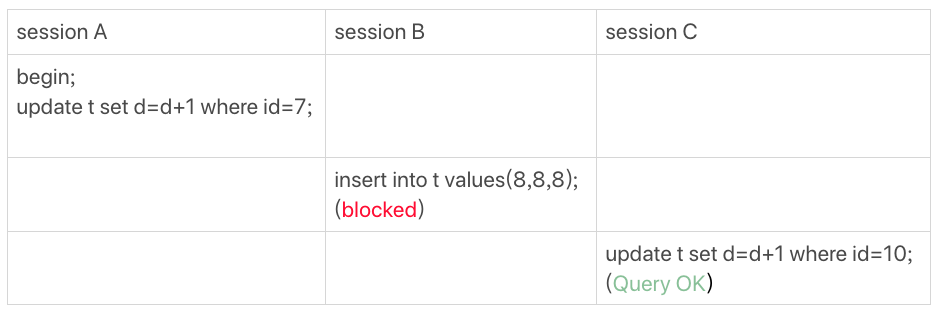


图1 等值查询的间隙锁

由于表t 没有id=7的记录，

1. 根据原则1，加锁单位是next-key lock，sessionA加锁范围就是（5，10];
2. 同时根据优化2，这是一个等值查询id=7,而id=10不满足查询条件，next-key lock退化为

间隙锁，因此最终加锁的范围(5,10);

所以session B 要往这个间隙里面插入id=8的记录会被锁住，但是sessionC 修改id=10这行是可以的

**案例二：非唯一索引等值锁**

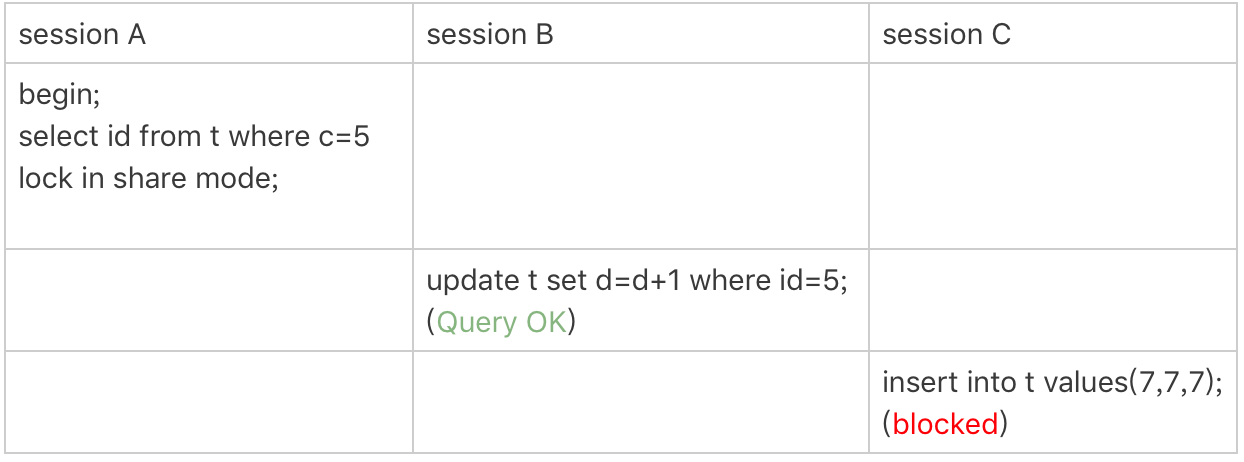


图2 只加在非唯一索引上的锁

Session A要给索引 c=5的这一行加锁

1. 根据原则1，加锁单元是next-key lock,因此会给(5,10]加上next-key lock
2. 要注意c是普通索引，因此仅访问c=5这一条记录是不能马上停下来的，需要向右遍历，查到c=10发放弃。根据原则2，访问到的都要加锁，因此要给(5,10]加next-key lock
3. 但是同时这个符合优化2，等值判断，向右遍历，最后一个值不满足c=5这个等值条件，因此退化成间隙锁（5,10）
4. 根据原则2，**只有访问到的对象才会加锁，**这个查询使用覆盖索引，并不需要访问主键索引，所以主键索引上没有加任何锁，这就是为什么sessionB的update可以执行完成

但session C要插入(7,7,7)的记录，就会被sessionA 的间隙锁（5,10）锁住

需要注意，lock in share mode只锁覆盖索引，但是如果是for update就不一样了，执行for update，系统会认为接下来更新数据，因此会顺便给主键索引上满足条件的行加上行锁。

**说明 ，**锁是加在索引上的；同时 如果用lock in share mode来给行加读锁避免数据被跟新的话，就必须的绕过覆盖索引的优化，在查询字段中加入索引中不存在的字段。比如，将sessionA的查询语句改成select d from t where c=5 lock in share mode

**案例三：主键索引范围锁**

下面两条查询语句，加锁范围相同吗？

mysql> select \* from t where id=10 for update;

mysql> select \* from t where id>=10 and id<11 for update;

Id定义为int类型，这两个语句并不完全等价

在逻辑上，这两条查询肯定是等价的，但是它们的加锁规则不太一样，现在看看加锁效果

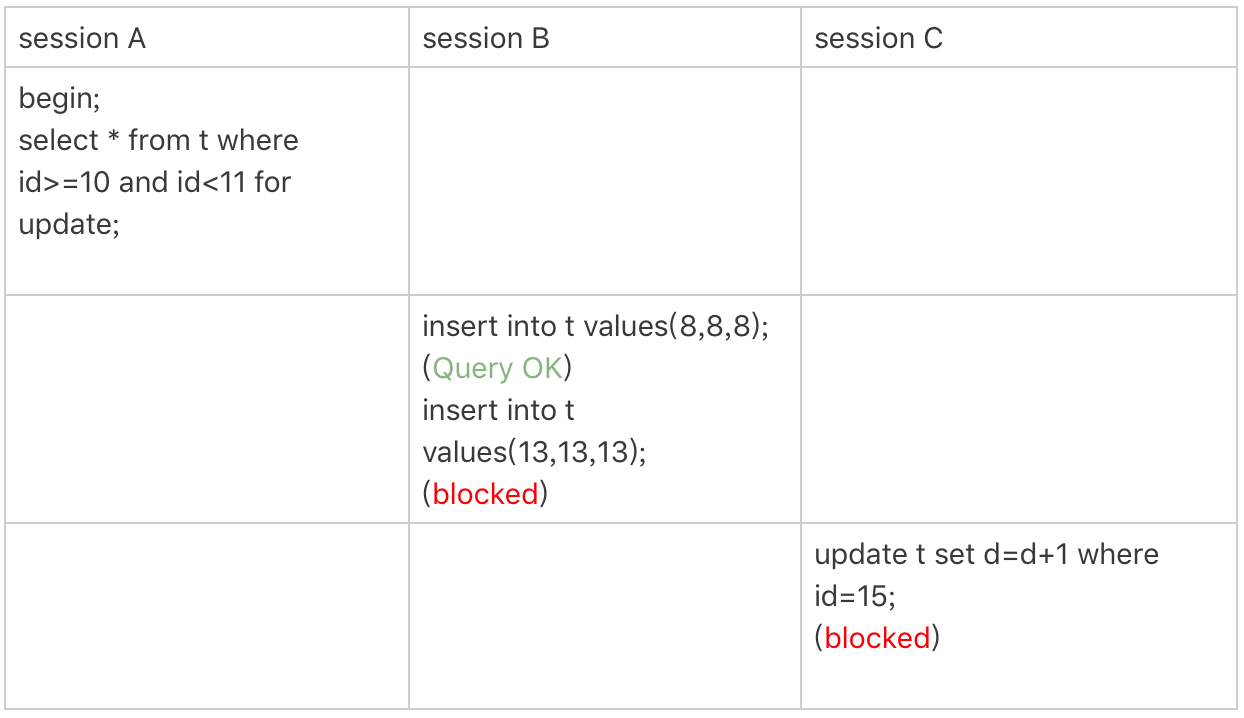


图3 主键上范围查询的锁

1.开始执行的时候，要找到第一个id=10的行，因此本该是next-key lock (5,10]。根据优化1，主键id上等值条件，退化成行锁，只加了id=10这一行的行锁。

2.范围查找就往后继续找，找到id=15这一行停下来，因此需要加next-key lock(10,15]

所以，sessionA 这时候锁的范围就是主键索引上，行锁id=10和next-key lock (10,15],这样，sessionB 和sessionC 结果就正确了

**首次sessionA定位查找id=10的行时候，是当做等值查询来判断的，而向右扫描到id=15的时候，用的是范围查询判断**

**案例四：非唯一索引范围锁**

案例中的查询语句where部分用的是字段c

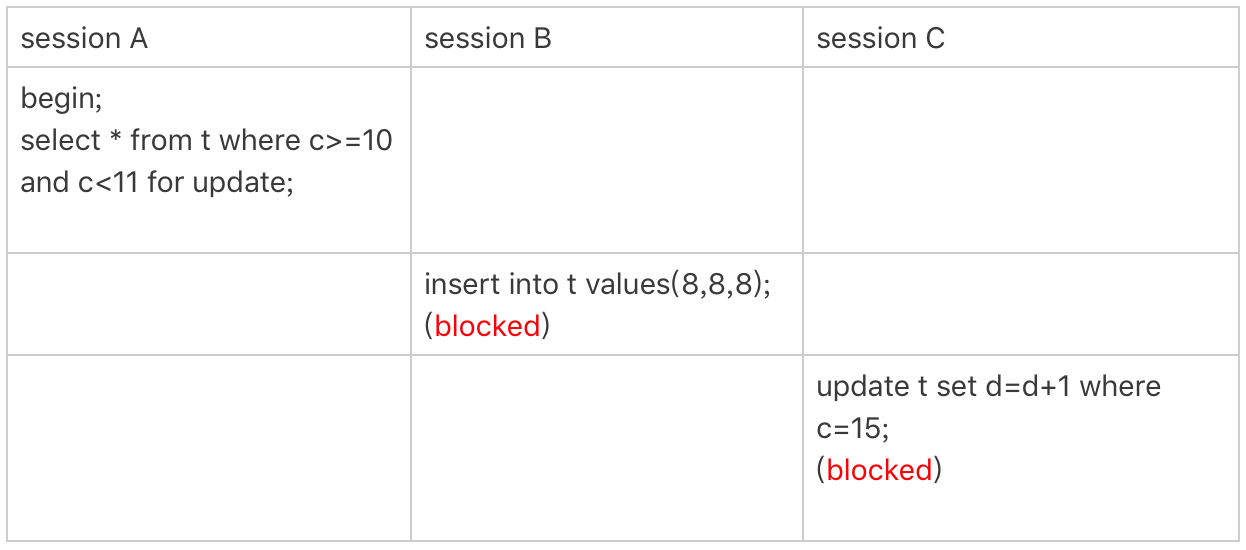


图4 非唯一索引范围锁

Session A 用字段c来判断，加锁规则跟案例三唯一不同的是：在第一次用c=10定位记录的时候，索引c加上了(5,10]这个next-key lock 后，由于索引c是非唯一索引，没有优化规则，也就是说不会蜕变成行锁，因此最终session A加的锁是索引c上的(5,10]和(10,15]这两个next-key lock

所以sessionB要插入(8,8,8)的这个语句是会被堵住

扫描到c=15才停止扫描，是合理的，因为InnoDB要扫描到c=15，才知道不需要继续往后找

**案例五：唯一索引范围锁bug**

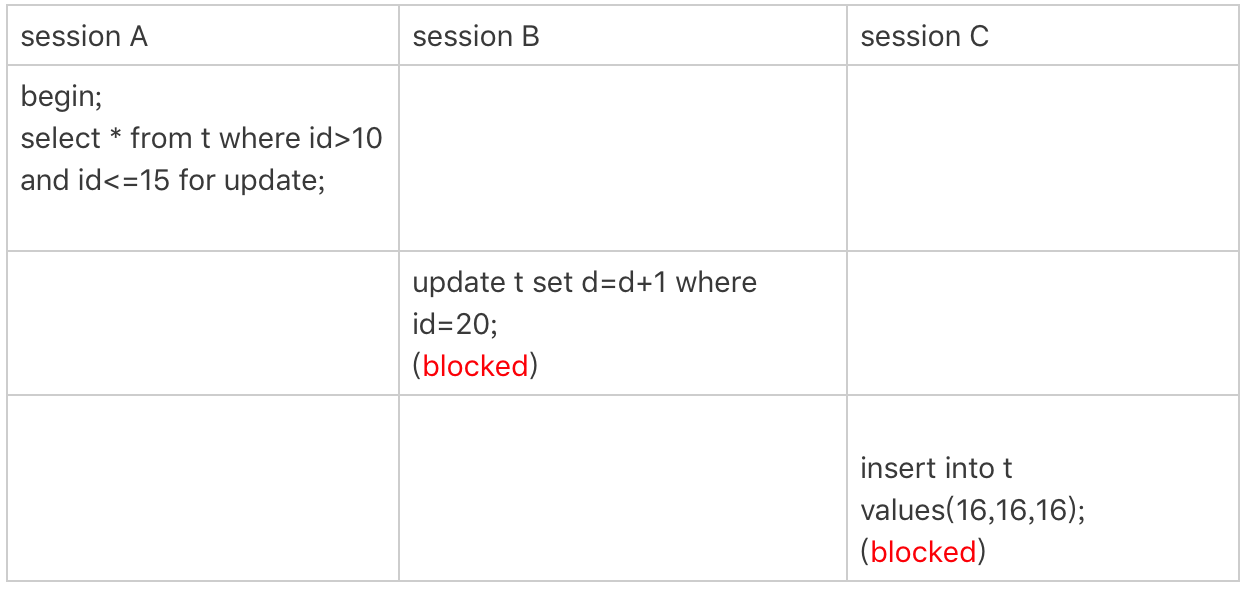


图 5 唯一索引范围锁bug

Session A是一个范围查询，按照原则1的话，应该是索引id上加(10,15]next-key lock,并且因为id是唯一键，所以循环判断id=15这一行就应该停止了。

innoDB会往前扫描的第一个不满足条件为止，也就是id=20,而且由于这是个范围扫描，因此索引id上的(15.20]这个next-key lock也会被锁住

SessionB更新id=20这一行，会被锁住，同样的，sessionC要插入id=16的一行，也被锁住

这里锁住id=20这一行，其实是没有必要的，因为扫描到id=15,就可以确定不用往后找了，但实现上还是这么做了。认为是bug

**案例六：非唯一索引上存在“等值”的例子**

插入一条记录

mysql> insert into t values(30,10,30);

新插入的这一行c=10,现在表里面有两个c=10的行，这个时候索引c上的间隙是什么状态呢？由于非唯一索引包含主键的值，所以是不可能存在“相同”的两行。

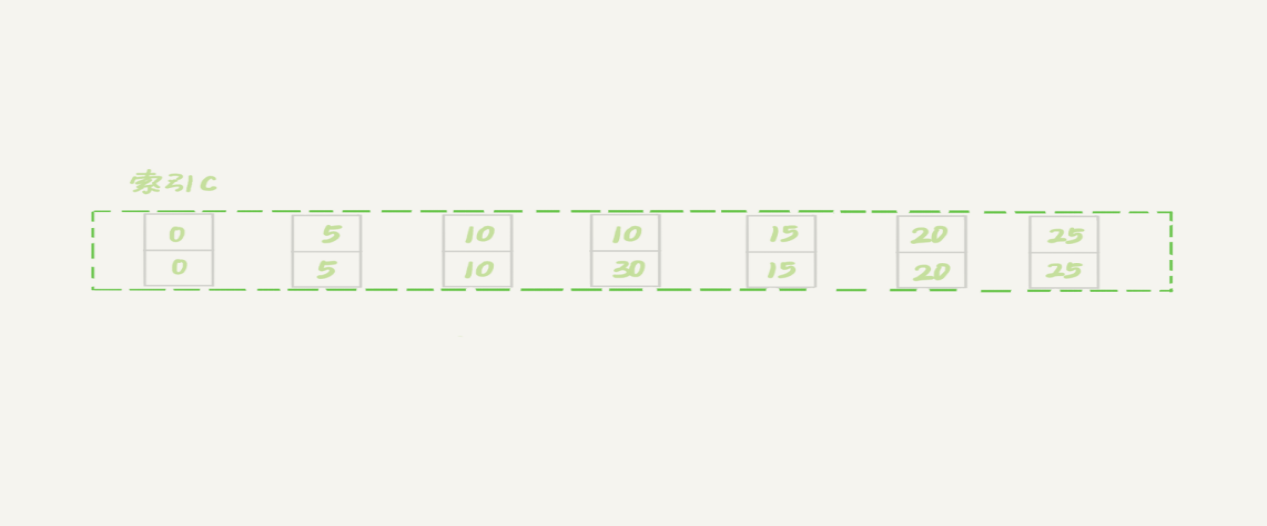


图 6 非唯一索引等值的例子

虽然有两个c=10，但是它们的主键id是不同的，（分别是10和30）因此这两个c=10的记录之间也是有间隙的

图中索引c=10上的主键id,为了跟间隙锁的开区间形式进行区别，用c=10,id=30,这样的形式表示这一行。

用delete语句来验证，注意delete语句加锁的逻辑，其实跟select...for update是类似的

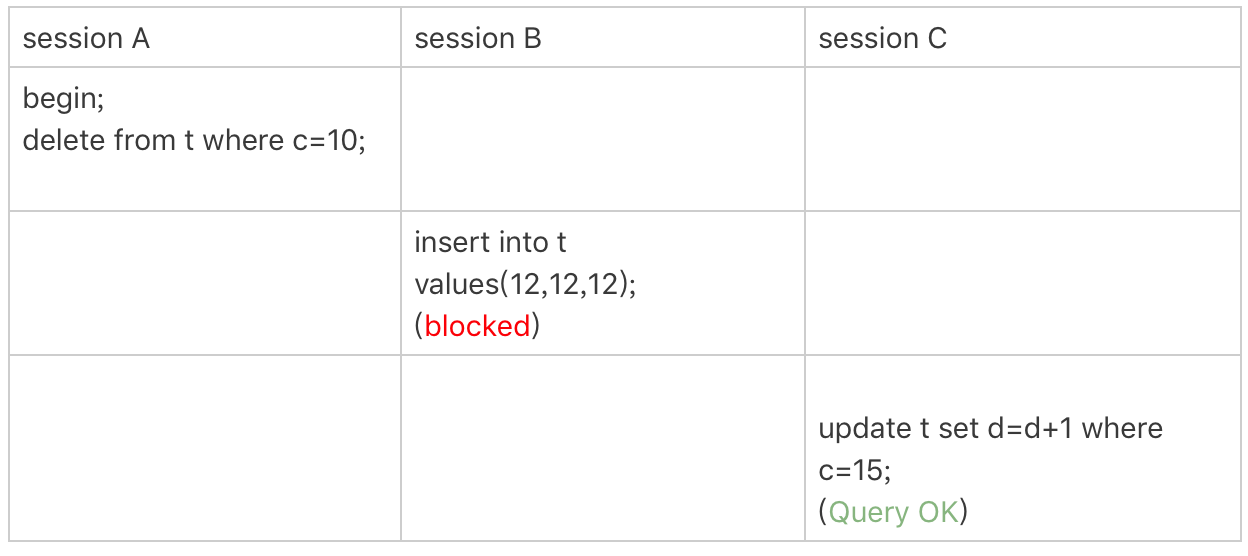


图7 delete 示例

Session A遍历的时候，想访问第一个c=10的记录，同样的，根据原则1，这里边加的（c=5,id=5）到c=10,id=10这个next-key lock,循环才结束，根据优化2，这是一个等值查询，向后查询不满足条件的行，所以退化成(c=10,id=10)到(c=15，id=15)的间隙锁。

这个delete语句在索引c上加锁范围，就是下图中蓝色区域覆盖部分

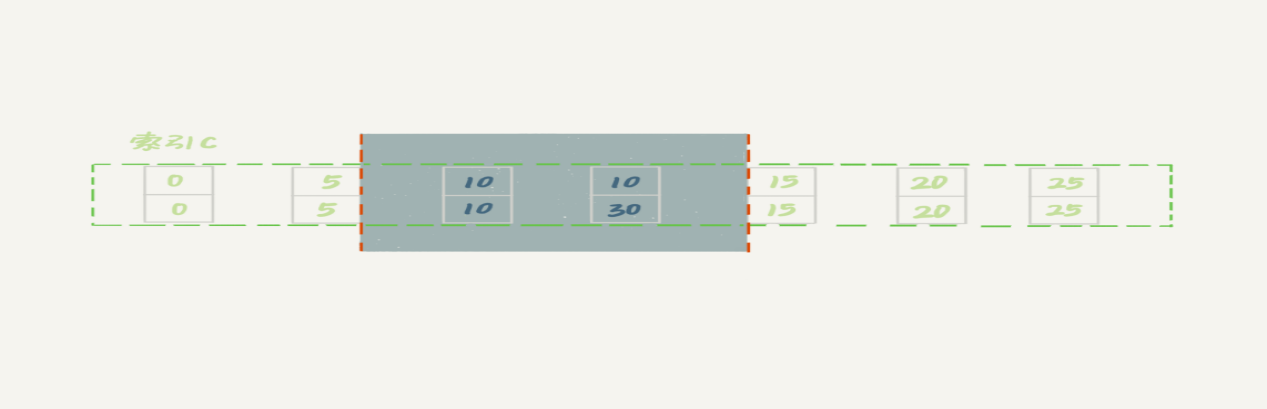


图 8 delete 加锁效果示例

蓝色区域都虚线，表示开区间，即(c=5,id=5)和(c=15,id=15)这两行都没有行锁

**案例七：limit语句加锁**

例子6也有一个对照案例，场景如下

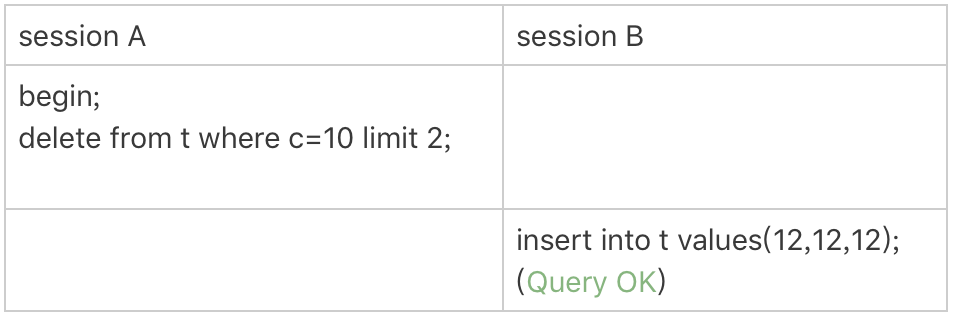


图 9 limit语句加锁

SessionA的delete语句加了limit2,表t里c=10的记录其实只有两条，因此不加limit2,删除的效果都是一样的，但是加锁的效果却不同，sessionB的insert语句执行通过了，跟案例六的结果不同。

案例七的delete语句明确加了limit2的限制，因此遍历到（c=10,id=30）这一行之后，满足条件的行已经有两条，循环结束了。

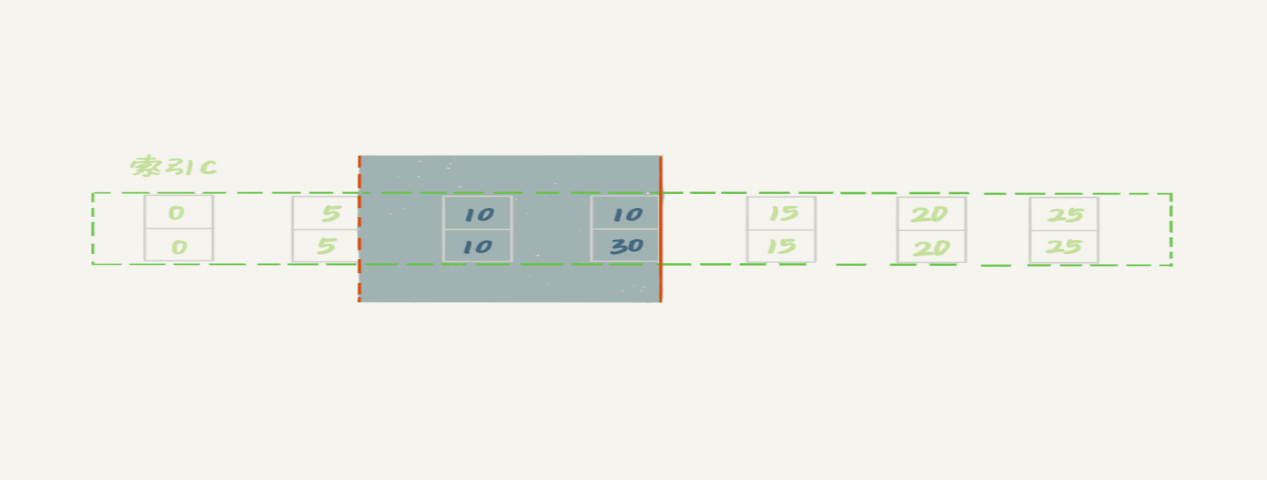
因此，索引c上加锁的范围就变成了从(c=5,id=5)到(c=10,id=30)这个前开后闭区间，如下图

图 10 带limit2的加锁效果

(c=10,id=30)之后的间隙并没有在加锁的范围里，因此insert语句插入c=12是可以执行成功的

**对实践的指导意义，在删除数据的时候尽量用limit,**这样不仅可以控制删除数据的条数，让操作更安全，还可以减小加锁的范围。

**案例八：一个死锁的例子**

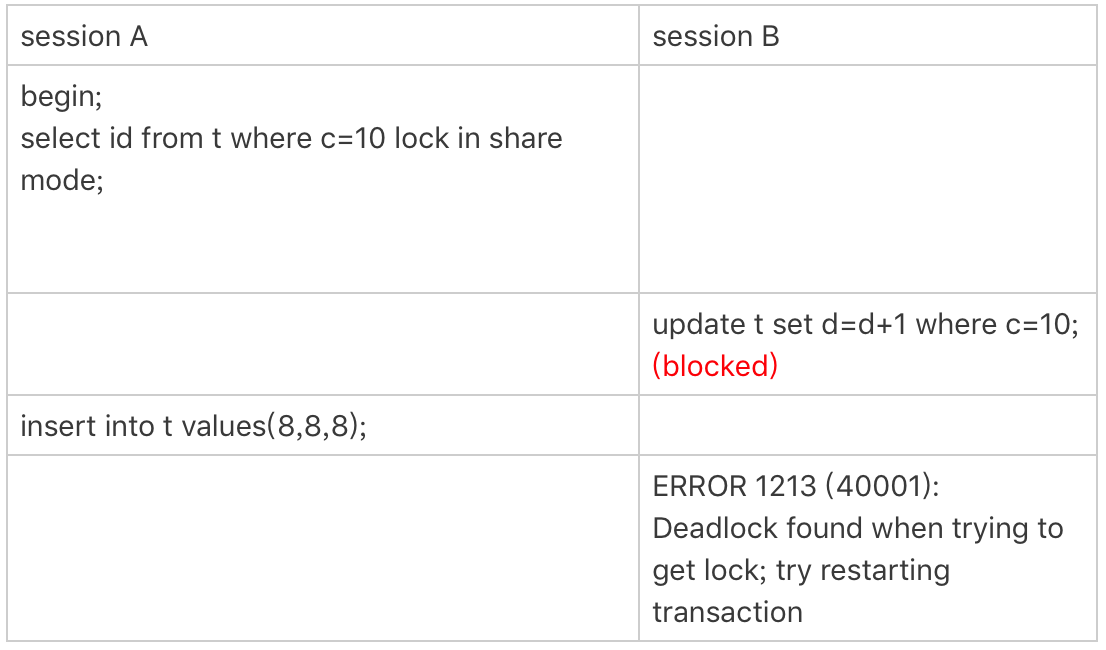
目的是说明：next-key lock 实际上是间隙锁和行锁加起来的结果。 

图 11 案例八的操作序列

1.session A 启动事务后执行查询语句 lock in share mode，在索引c上加了next-key lock(5,10]和间隙锁(10,15)

2.session B 的update语句也要在索引c上加next-key lock(5,10]，进入锁等待

3.然后sessionA 要在插入(8,8,8)这一行，被sessionB的间隙锁锁住，由于出现死锁，InnoDB让sessionB回滚

疑问？sessionB的next-key lock不是还没申请成功吗？

其实是这样的，sessionB的“加next-key lock(5,10]”操纵，实际上分两步，先是加（5,10）的间隙锁，加锁成功，然后加c=10的行锁，这时候才被锁住

**也就是说，在分析加锁规则的时候可以用next-key lock来分析，具体执行的时候，要分成间隙锁和行锁两阶段来执行的**

**思考题**

还是开头的初始化表t，里面有6条记录，图12的语句序列中，为什么sessionB的insert操作，会被锁住？



图 12 锁分析思考题

1. 由于order by c desc ,第一要定位的是索引c上“最右边的”c=20的行，所以会加上间隙锁（20,25）和next-key lock(15,20]
2. 在索引c上向左遍历，要扫描到c=10才停下来，所以next-key lock会加到（5,10]，这个正是阻塞sessionB的insert语句的原因
3. 在扫描过程中，c=20,c=15,c=10这三个值都存在，由于select \* ，所以会在主键id上加三个锁

因此sessionA的select语句锁的范围就是

1. 索引c上(5,25)
2. 主键索引上id=15、20两个行锁