加锁规则

两个原则

- 加锁的基本单位是next-key lock。前开后闭区间，而且是向左加锁，例如id 有 0、5、10，如果扫到了10，那么加锁的区间是(5,10]
- 查找过程中访问的对象才会加锁

两个优化

- 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
- 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（等值向右查询指<= ）

一个bug

- 唯一索引上的范围查询会访问到不满足条件的第一个值为止

例子

前提

```
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);
```

等值查询间隙锁

session A	session B	session C
begin; update t set d=d+1 where id=7;		
	insert into t values(8,8,8); (blocked)	
		update t set d=d+1 where id=10; (Query OK)

解释

- 根据原则1，加锁单位是next-key lock，session A 加锁范围是(5,10]
- 根据优化2，这是一个等值查询，而id=10不满足查询条件，next-key lock退化为间隙锁，因此最终加锁范围是(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 session C 修改 id=10 这行是可以的。

非唯一索引等值锁

session A	session B	session C
begin; select id from t where c=5 lock in share mode;		
	update t set d=d+1 where id=5; (Query OK)	
		insert into t values(7,7,7); (blocked)

解释

- 根据原则1，加锁单位是next-key lock，因此会给(0.5]加上next-key lock
- c是普通索引，因此仅访问c=5这一条记录不能马上停下来，需要向右遍历，查询到c=10才放弃。根据原则2，访问到的都要加锁，因此要给(5,10]加上next-key lock。
- 但是同时这个符号优化2，等值判断，向右遍历，最后一个不满足c=5这个等值条件，因此退化为间隙锁(5,10)。
- 根据原则2，只有访问到的对象才会加锁，这个查询用的是覆盖索引，并不需要访问主键索引，所以主键索引上没有加任何的锁。这也是为什么session B的update语句能够执行完成
- 但 session C 要插入一个 (7,7,7) 的记录，就会被 session A 的间隙锁 (5,10) 锁住。

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 时，系统会认为你接下来要更新数据，因此会顺便给主键索引上满足条件的行加上行锁。

锁是加在索引上的；同时，它给我们的指导是，如果你要用 lock in share mode 来给行加读锁避免数据被更新的话，就必须得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

主键索引范围锁

session A	session B	session C
begin; select * from t where id>=10 and id<11 for update;		
	insert into t values(8,8,8); (Query OK) insert into t values(13,13,13); (blocked)	
		update t set d=d+1 where id=15; (blocked)

解释

- 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上的等值查询，退化为行锁，只加了id=10的这一行的行锁。
- 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]
- 那么session A的锁的范围就是在主键索引上，行锁id=10以及next-key lock(10,15]。

首次 session A 定位查找 id=10 的行的时候，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。

非唯一性索引范围锁

session A	session B	session C
begin; select * from t where c>=10 and c<11 for update;		
	insert into t values(8,8,8); (blocked)	
		update t set d=d+1 where c=15; (blocked)

解释

这次 session A 用字段 c 来判断，加锁规则跟案例三唯一的不同是：在第一次用 c=10 定位记录的时候，索引 c 上加了 (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，也就是说不会蜕变为行锁，因此最终 sesion A 加的锁是，索引 c 上的 (5,10] 和 (10,15] 这两个 next-key lock。

唯一索引范围锁bug

session A	session B	session C
begin; select * from t where id>10 and id<=15 for update;		
	update t set d=d+1 where id=20; (blocked)	
		insert into t values(16,16,16); (blocked)

解释

- session A是一个范围查询，按照原则的话，应该是在索引id上只加(10,15]这个next-key lock，并且因为id是唯一键，所以循环判断到id=15行就应该停止
- 但是实现上，InnoDB会3往前扫描到第一个不满足条件的行为止，也就是id=20。而且由于这个是范围扫描，因此索引id上的(15,20]这个next-key lock也会被锁上
- 所以你看到了，session B 要更新 id=20 这一行，是会被锁住的。同样地，session C 要插入 id=16 的一行，也会被锁住。

mysql 8.0.18被修复了

倒序查询加锁分析

session A	session B
begin; select * from t where c>=15 and c<=20 order by c desc lock in share mode;	
	insert into t values(6,6,6); (blocked)

解释

- 首先select *，说明查询需要回表，那么就会在id和c的索引上分别加锁
- C上的索引加锁分析
  - order by desc所以是从右向左的搜索在索引上，第一个要定位的是索引 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]
  - 在索引 c 上向左遍历，要扫描到 c=10 才停下来（向左遍历查询到第一个不满足c=15的值为止），所以 next-key lock 会加到 (5,10]，这正是阻塞 session B 的 insert 语句的原因
  - 综上所述，C的加锁范围是(5,25)
- id主键索引加锁分析
  - 由于是 select * 所以需要回表，id 上的对应行也会被加锁。 但是c=10这行并不满足条件，只是在索引c上被访问了，加了锁，不满足条件就丢弃了，不会回表，所以id上没有被访问，也不会加锁。 即c上有20,15,10加锁，id上只有20,15加锁。