

30 | 答疑文章（二）：用动态的观点看加锁

2019-01-21 林晓斌

MySQL实战45讲

[进入课程 >](#)



讲述：林晓斌

时长 15:40 大小 14.36M



在第20和21篇文章中，我和你介绍了 InnoDB 的间隙锁、next-key lock，以及加锁规则。在这两篇文章的评论区，出现了很多高质量的留言。我觉得通过分析这些问题，可以帮助你加深对加锁规则的理解。

所以，我就从中挑选了几个有代表性的问题，构成了今天这篇答疑文章的主题，即：用动态的观点看加锁。

为了方便你理解，我们再一起复习一下加锁规则。这个规则中，包含了两个“原则”、两个“优化”和一个“bug”：

原则 1：加锁的基本单位是 next-key lock。希望你还记得，next-key lock 是前开后闭区间。


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候，next-key lock 退化为行锁。

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。

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。

接下来，我们的讨论还是基于下面这个表 t：


 复制代码

```
1 CREATE TABLE `t` (  
2   `id` int(11) NOT NULL,  
3   `c` int(11) DEFAULT NULL,  
4   `d` int(11) DEFAULT NULL,  
5   PRIMARY KEY (`id`),  
6   KEY `c` (`c`)  
7 ) ENGINE=InnoDB;  
8  
9 insert into t values(0,0,0),(5,5,5),  
10 (10,10,10),(15,15,15),(20,20,20),(25,25,25);
```

不等号条件里的等值查询

有同学对“等值查询”提出了疑问：等值查询和“遍历”有什么区别？为什么我们文章的例子里面，where 条件是不等号，这个过程里也有等值查询？

我们一起来看下这个例子，分析一下这条查询语句的加锁范围：

 复制代码

```
1 begin;  
2 select * from t where id>9 and id<12 order by id desc for update;
```

利用上面的加锁规则，我们知道这个语句的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？

我们说加锁单位是 next-key lock，都是前开后闭区间，但是这里用到了优化 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)。

但是，我们的查询语句中 where 条件是大于号和小于号，这里的“等值查询”又是从哪里来的呢？

要知道，加锁动作是发生在语句执行过程中的，所以你在分析加锁行为的时候，要从索引上的数据结构开始。这里，我再把这个过程拆解一下。

如图 1 所示，是这个表的索引 id 的示意图。



(0,0,0) (5,5,5) (10,10,10) (15,15,15) (20,20,20) (25,25,25)

图 1 索引 id 示意图


1. 首先这个查询语句的语义是 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树的搜索过程得到的，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=5 这一行，所以会加一个 next-key lock (0,5]。

也就是说，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，是 @发条橙子同学提出的问题：下面这个语句的加锁范围是什么？

 复制代码

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

这条查询语句里用的是 in，我们先来看这条语句的 explain 结果。

```
mysql> explain select id from t where c in(5,20,10) lock in share mode;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t | NULL | range | c | c | 5 | NULL | 3 | 100.00 | Using where; Using index |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

图 2 in 语句的 explain 结果

可以看到，这条 in 语句使用了索引 c 并且 rows=3，说明这三个值都是通过 B+ 树搜索定位的。

在查找 c=5 的时候，先锁住了 (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)。

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


通过这个分析，我们可以知道，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，再加 c=10 的记录锁，最后加 c=20 的记录锁。

你可能会说，这个加锁范围，不就是从 (5,25) 中去除掉 c=15 的行锁吗？为什么这么麻烦地分段说呢？

因为我要跟你强调这个过程：这些锁是“在执行过程中一个一个加的”，而不是一次性加上去的。

理解了加锁过程之后，我们就可以来分析下面例子中的死锁问题了。

如果同时有另外一个语句，是这么写的：

 复制代码

```
1 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？

我们现在都知道间隙锁是不互锁的，但是这两条语句都会在索引 c 上的 c=5、10、20 这三行记录上加记录锁。

这里你需要注意一下，由于语句里面是 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。

也就是说，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。

关于死锁的信息，MySQL 只保留了最后一个死锁的现场，但这个现场还是不完备的。

有同学在评论区留言到，希望我能展开一下怎么看死锁。现在，我就来简单分析一下上面这个例子的死锁现场。

怎么看死锁？

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

```
2019-01-09 19:21:11 0x7feb98d47700
*** (1) TRANSACTION:
TRANSACTION 422127109356256, ACTIVE 0 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 4 lock struct(s), heap size 1136, 3 row lock(s)
MySQL thread id 98, OS thread handle 140649857836800, query id 119190 localhost 127.0.0.1 root Sending data
select id from t where c in(5,20,10) lock in share mode
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 24 page no 4 n bits 80 index c of table `test`.`t` trx id 422127109356256 lock mode S waiting
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 0000000a; asc    ;;
 1: len 4; hex 0000000a; asc    ;;

*** (2) TRANSACTION:
TRANSACTION 1315, ACTIVE 0 sec starting index read
mysql tables in use 1, locked 1
5 lock struct(s), heap size 1136, 7 row lock(s)
MySQL thread id 99, OS thread handle 140649858103040, query id 119189 localhost 127.0.0.1 root Sending data
select id from t where c in(5,20,10) order by c desc for update
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 24 page no 4 n bits 80 index c of table `test`.`t` trx id 1315 lock_mode X
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 0000000a; asc    ;;
 1: len 4; hex 0000000a; asc    ;;

Record lock, heap no 6 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 00000014; asc    ;;
 1: len 4; hex 00000014; asc    ;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 24 page no 4 n bits 80 index c of table `test`.`t` trx id 1315 lock_mode X waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 00000005; asc    ;;
 1: len 4; hex 00000005; asc    ;;

*** WE ROLL BACK TRANSACTION (1)
```

图 3 死锁现场

我们来看看这图中的几个关键信息。

1. 这个结果分成三部分：

(1) TRANSACTION，是第一个事务的信息；

(2) TRANSACTION，是第二个事务的信息；

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

2. 第一个事务的信息中：

WAITING FOR THIS LOCK TO BE GRANTED，表示的是这个事务在等待的锁信息；

index c of table `test`.`t`，说明在等的是表 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；

Record lock 说明这是一个记录锁；

n_fields 2 表示这个记录是两列，也就是字段 c 和主键字段 id；

0: len 4; hex 0000000a; asc ;; 是第一个字段，也就是 c。值是十六进制 a，也就是 10；

1: len 4; hex 0000000a; asc ;; 是第二个字段，也就是主键 id，值也是 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。

当然你是知道的，既然出现死锁了，就表示这个事务也占有别的锁，但是没有显示出来。别着急，我们从第二个事务的信息中推导出来。

3. 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哪些锁；

index c of table `test`.`t` 表示锁是在表 t 的索引 c 上；

hex 0000000a 和 hex 00000014 表示这个事务持有 c=10 和 c=20 这两个记录锁；

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

从上面这些信息中，我们就知道：

1. “lock in share mode” 的这条语句，持有 c=5 的记录锁，在等 c=10 的锁；
2. “for update” 这个语句，持有 c=20 和 c=10 的记录锁，在等 c=5 的记录锁。

因此导致了死锁。这里，我们可以得到两个结论：

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 这条语句占有的资源更多，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。

怎么看锁等待？

看完死锁，我们再来看一个锁等待的例子。

在第 21 篇文章的评论区，@Geek_9ca34e 同学做了一个有趣验证，我把复现步骤列出来：

session A	session B
begin; select * from t where id>10 and id<=15 for update;	
	delete from t where id=10; (Query OK) insert into t values(10,10,10); (blocked)

图 4 delete 导致间隙变化

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

现在我们一起看一下此时 show engine innodb status 的结果，看看能不能给我们一些提示。锁信息是在这个命令输出结果的 TRANSACTIONS 这一节。你可以在文稿中看到这张图片

```

---TRANSACTION 1311, ACTIVE 4 sec inserting
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 11, OS thread handle 140504341464832, query id 20700 localhost 127.0.0.1 root update
insert into t values(10,10,10)
----- TRX HAS BEEN WAITING 4 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 24 page no 3 n bits 80 index PRIMARY of table `test`.`t` trx id 1311 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 5 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
 0: len 4; hex 0000000f; asc      ;;
 1: len 6; hex 000000000513; asc      ;;
 2: len 7; hex b0000001250134; asc    % 4;;
 3: len 4; hex 0000000f; asc      ;;
 4: len 4; hex 8000000f; asc      ;;

```

图 5 锁等待信息

我们来看几个关键信息。

1. index PRIMARY of table `test`.`t`，表示这个语句被锁住是因为表 t 主键上的某个锁。
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，你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. 那么这个 gap 是在哪个记录之前的呢？接下来的 0~4 这 5 行的内容就是这个记录的信息。
4. n_fields 5 也表示了，这一个记录有 5 列：

0: len 4; hex 0000000f; asc ;; 第一列是主键 id 字段，十六进制 f 就是 id=15。所以，这时我们就知道了，这个间隙就是 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; hex 000000000513; asc ;; 第二列是长度为 6 字节的事务 id，表示最后修改这一行的是 trx id 为 1299 的事务。

2: len 7; hex b0000001250134; asc % 4;; 第三列长度为 7 字节的回滚段信息。可以看到，这里的 acs 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，我们就知道了，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) 变成了一个 (5,15)。

说到这里，你可以联合起来再思考一下这两个现象之间的关联：

1. session A 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 篇文章的课后思考题，当我们执行 select * from t where c >= 15 and c <= 20 order by c desc lock in share mode; 向左扫描到 c=10 的时候，要把 (5, 10] 锁起来。

也就是说，所谓“间隙”，其实根本就是从“这个间隙右边的那个记录”定义的。

update 的例子

看过了 insert 和 delete 的加锁例子，我们再来看一个 update 语句的案例。在留言区中 @信信 同学做了这个试验：

session A	session B
begin; select c from t where c > 5 lock in share mode;	
	update t set c = 1 where c = 5; (Query OK) update t set c = 5 where c = 1; (blocked)

图 6 update 的例子

你可以自己分析一下，session A 的加锁范围是索引 c 上的 (5,10]、(10,15]、(15,20]、(20,25] 和 (25,supremum]。

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

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

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

按照我们上一节说的，索引 c 上 (5,10) 间隙是由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：

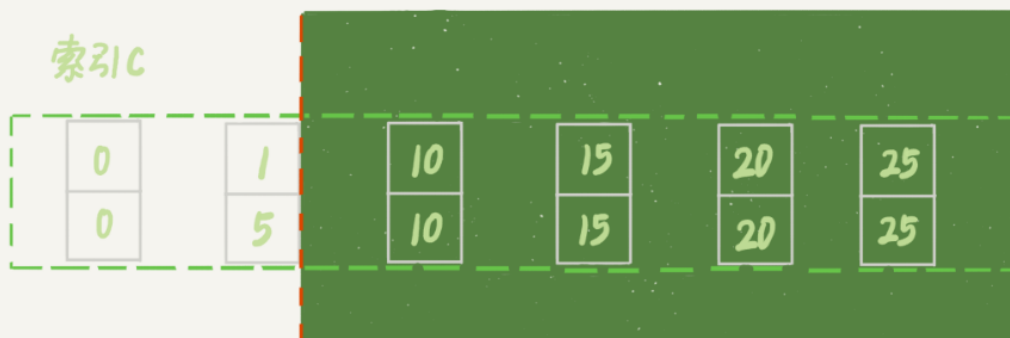


图 7 session B 修改后，session A 的加锁范围

好，接下来 session B 要执行 `update t set c = 5 where c = 1` 这个语句了，一样地可以拆成两步：

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的 (1,10) 中插入数据，所以就被堵住了。

小结

今天这篇文章，我用前面[第 20](#)和[第 21 篇](#)文章评论区的几个问题，再次跟你复习了加锁规则。并且，我和你重点说明了，分析加锁范围时，一定要配合语句执行逻辑来进行。

在我看来，每个想认真了解 MySQL 原理的同学，应该都要能够做到：通过 explain 的结果，就能够脑补出一个 SQL 语句的执行流程。达到这样的程度，才算是索引组织表、索引、锁的概念有了比较清晰的认识。你同样也可以用这个方法，来验证自己对这些知识点的掌握程度。

在分析这些加锁规则的过程中，我也顺便跟你介绍了怎么看 `show engine innodb status` 输出结果中的事务信息和死锁信息，希望这些内容对你以后分析现场能有所帮助。

老规矩，即便是答疑文章，我也还是要留一个课后问题给你的。

上面我们提到一个很重要的点：所谓“间隙”，其实根本就是从“这个间隙右边的那个记录”定义的。

那么，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呢？

你可以把你关于分析和验证方法写在留言区，我会在下一篇文章的末尾和你讨论这个问题。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。

上期问题时间

我在上一篇文章最后留给的问题，是分享一下你关于业务监控的处理经验。

在这篇文章的评论区，很多同学都分享了不错的经验。这里，我就选择几个比较典型的留言，和你分享吧：

@老杨同志 回答得很详细。他的主要思路就是关于服务状态和服务质量的监控。其中，服务状态的监控，一般都可以用外部系统来实现；而服务的质量的监控，就要通过接口的响应时间来统计。

@Ryoma 同学，提到服务中使用了 healthCheck 来检测，其实跟我们文中提到的 select 1 的模式类似。

@强哥 同学，按照监控的对象，将监控分成了基础监控、服务监控和业务监控，并分享了每种监控需要关注的对象。

这些都是很好的经验，你也可以根据具体的业务场景借鉴适合自己的方案。




MySQL 实战 45 讲

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。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 29 | 如何判断一个数据库是不是出问题了？

下一篇 31 | 误删数据后除了跑路，还能怎么办？



Ryoma 置顶

2019-01-22

👍 8

删除数据，导致锁扩大的描述：“因此，我们就知道了，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) 变成了一个 (5,15)。”

我觉得这个提到的(5, 10) 和 (10, 15)两个间隙会让人有点误解，实际上在删除之前间隙锁只有一个(10, 15)，删除了数据之后，导致间隙锁左侧扩张成了5，间隙锁成为了(5, 15)。

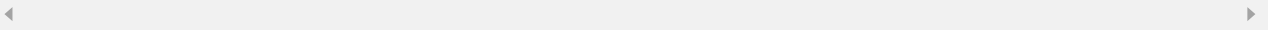
展开 ▾

作者回复: 嗯 所以我这里特别小心地没有写 “锁” 这个字。

间隙 (5,10)、(10,15) 是客观存在的。

你提得也很对，“锁”是执行过程中才加的，是一个动态的概念。

这个问题也能够让大家更了解我们标题的意思，置顶了哈 📌



令狐少侠 置顶

2019-01-22

👍 7

有个问题想确认下，在死锁日志里，lock_mode X waiting是间隙锁+行锁，lock_mode X locks rec but not gap这种加but not gap才是行锁？

老师你后面能说下group by的原理吗，我看目录里面没有

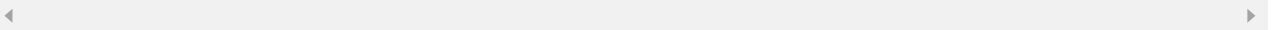
展开 ▾

作者回复: 对，好问题

lock_mode X waiting表示next-key lock；

lock_mode X locks rec but not gap是只有行锁；

还有一种 “locks gap before rec”，就是只有间隙锁；



IceGeek17

2019-02-11

👍 2

老师，新年好，有几个问题：

问题一：

对于文中的第一个例子（不等号条件里的等值查询），当试图去找 “第一个id<12的值”的时候，用的还是从左往右的遍历（因为用到了优化2），也就是说，当去找第一个等值的时

候（通过树搜索去定位记录的时候），即使order by desc，但用的还是向右遍历，当找...
展开

作者回复: 1. 对的

2. 对的

3. “因为扫到了c=10，所以会加next-key lock (5,10]”，对的。
第二个“如果”，实现上并不是这样的，所以没法回答😞



2019-01-22

2

老师好：

`select * from t where c >= 15 and c <= 20 order by c desc for update;`

为什么这种c=20就是用来查数据的就不是向右遍历

`select * from t where c >= 15 and c <= 20` 这种就是向右遍历

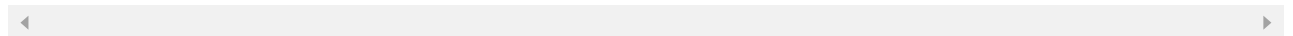
怎么去判断合适是查找数据，何时又是遍历呢，是因为第一个有order by desc，然...

展开

作者回复: 索引搜索就是“找到第一个值，然后向左或向右遍历”，
order by desc 就是要用最大的值来找第一个；
order by 就是要用做小的值来找第一个；

“所以只需要[20,25)来判断已经是最后一个20就可以了是吧”，
你描述的意思是对的，但是在MySQL里面不建议写这样的前闭后开区间哈，容易造成误解。
可以描述为：

“取第一个id=20后，向右遍历(25,25)这个间隙” ^_^



Jason_鹏
2019-01-22

2

最后一个update的例子，为没有加(0, 5)的间隙呢？我理解应该是先拿c = 5去b+树搜索，按照间隙最右原则，应该会加(0, 5]的间隙，然后c = 5不满足大于5条件，根据优化2原则退化(0, 5)的间隙，我是这样理解的

展开

作者回复: 根据 $c > 5$ 查到的第一个记录是 $c = 10$ ，因此不会加 $(0, 5]$ 这个next-key lock。

你提醒得对，我应该多说明这句，我加到文稿中啦👍



长杰

2019-01-22

👍 1

老师，还是select * from t where $c \geq 15$ and $c \leq 20$ order by c desc in share mode与select * from t where $id > 10$ and $id \leq 15$ for update的问题，为何select * from t where $id > 10$ and $id \leq 15$ for update不能解释为：根据 $id = 15$ 来查数据，加锁 $(15, 20]$ 的时候，可以使用优化2，

这个等值查询是根据什么规则来定的？如果select * from t where $id > 10$ and $id \leq 15$...

展开

作者回复: 1. 代码实现上，传入的就是 $id > 10$ 里面的这个10

2. 可以的，不过因为id是主键，而且 $id = 15$ 这一行存在，我觉得用优化1解释更好哦



Justin

2019-01-22

👍 1

想咨询一下 普通索引 如果索引中包括的元素都相同 在索引中顺序是怎么排解的呢 是按主键排列的吗 比如(name ,age) 索引 name age都一样 那索引中会按照主键排序吗？

作者回复: 会的



库淘淘

2019-01-21

👍 1

对于问题 我理解是这样

session 1 :

delete from t;

begin; select * from t for update;

session 2:...

展开

作者回复: 发现了👍



老杨同志

2019-01-21

👍 1

先说结论：空表锁 $(-\text{supernum}, \text{supernum}]$, 老师提到过mysql的正无穷是supernum, 在没有数据的情况下, next-key lock 应该是supernum前面的间隙加 supernum的行锁。但是前开后闭的区间, 前面的值是什么我也不知道, 就写了一个 $-\text{supernum}$ 。

稍微验证一下

session 1) ...

展开

作者回复: 赞

show engine innodb status 有惊喜😄



唯她命

2019-04-03

👍

老师, update语句 mysql在执行过程中, 都是先拆成 插入 和 删除的吗? 不是直接修改?

作者回复: 修改索引值都会修改位置的😄



小卡向前冲

2019-03-22

👍

老师你好, 我的问题跟@长杰的第二问类似。对于sql: `select * from t where id > 10 and id <= 15 for update` 加上 `order by id desc`。它的加锁顺序是不是: $(10, 15]$, $(15, 20]$ (bug), $(5, 10]$?

我不太明白为什么说, 这里要用优化1, 获得行锁? 还是说, 先获取id=15的行锁, 然后在加 $(10, 15]$, $(5, 10]$

展开



w

2019-03-15

👍

有个很奇怪的现象

session a: begin; select * from cd where c <= 10 for update;

session b: select * from cd where c = 15 for update;

此时session b会被block

session a稍作修改...

展开 ▾



Long

2019-01-28



感觉这篇文章以及前面加锁的文章，提升了自己的认知。还有，谢谢老师讲解了日志的对应细节.....还愿了

作者回复: 😊 👍



滔滔

2019-01-23



老师，有个疑问，select * from t where c >= 15 and c <= 20 order by c desc lock in share mode; 向左扫描到 c=10 的时候，为什么要把 (5, 10] 锁起来？不锁也不会出现幻读或者逻辑上的不一致吧 😊

作者回复: 会加锁，insert into t values (6,6,6) 被堵住了



尘封

2019-01-23



老师，咨询个问题，本来想在后面分区表的文章问，发现大纲里没有分区表这一讲。

1，timestamp类型为什么不支持分区？

2，前面的文章讲过分区不要太多，这个多了会怎么样？比如一个表一千多个分区

谢谢

展开 ▾

作者回复: 会讲的哈~

新春快乐~





堕落天使

2019-01-22



老师，您好：

我执行 “explain select id from t where c in(5,20,10) lock in share mode;” 时，显示的rows对应的值是4。为什么啊？

我的mysql版本是：5.7.23-0ubuntu0.16.04.1，具体sql语句如下：

```
mysql> select * from t;...
```

展开 ∨

作者回复: 你这个例子里面有两行c=10



Ivan

2019-01-22



```
Jan 17 23:52:27 prod-mysql-01 kernel: [ pid ] uid tgid total_vm rss cpu oom_adj  
oom_score_adj name
```

```
Jan 17 23:52:27 prod-mysql-01 kernel: [125254] 0 125254 27087 5 0 0 0  
mysqld_safe
```

```
Jan 17 23:52:27 prod-mysql-01 kernel: [126004] 498 126004 24974389 22439356 ...
```

展开 ∨

作者回复: 好问题，第33篇会说到哈

你可以在mysql客户端参数增加 --quick 再试试



PengfeiWan...

2019-01-22



老师，您好：

对文中以下语句感到有困惑：

我们说加锁单位是 next-key lock，都是前开后闭区，但是这里用到了优化 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)。...

展开 ∨

作者回复: 主要是这里这一行不存在。。

如果能够明确找到一行锁住的话，使用优化1就更准确些



ServerCode...

2019-01-22



林老师我有个问题想请教一下，描述如下，望给予指点，先谢谢了！

环境：虚拟机，CPU 4核，内存8G，系统CentOS7.4，MySQL版本5.6.40

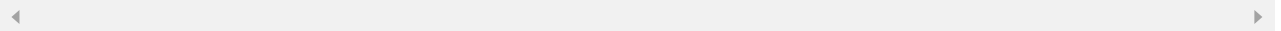
数据库配置：

bulk_insert_buffer_size = 256M...

展开 ∨

作者回复: 可以用到多核呀，你是怎么得到“时间主要耗费在了网络读取上。”这个结论的？

另外，把这三个文件先拷贝到数据库本地，然后本地执行load看看什么效果？



慕塔

2019-01-22



是这样的 假设只有一主一从 1)是集群只有一个sysbench实例，产生的数据流通过中间件，主机分全部写，和30%的读，另外70%的读全部分给从机。2)有两个sysbench，一个读写加压到主机，另一个只有加压到从机。主从复制之间通过binlog。问题在1)的QPS累加与2)QPS累加 意义一样吗 1)的一条事务有读写，而2)的情况，主机与1)一样，从机的读事务与主机里的读不一样吧 😊

展开 ∨

作者回复: 我觉得这两个对比不太公平^_^

1) 的测试可能会出现中间件瓶颈，

a)网络环节中间增加了一跳；

b) 如果是小查询，可能proxy先打到瓶颈

2)的测试结论一般会比1)好些

但是有这个架构，你肯定是从中间件访问数据库的，所以应该以1的测试结果为准

