

19 | 为什么我只查一行的语句，也执行这么慢？

2018-12-26 林晓斌



一般情况下，如果我跟你说查询性能优化，你首先会想到一些复杂的语句，想到查询需要返回大量的数据。但有些情况下，“查一行”，也会执行得特别慢。今天，我就跟你聊聊这个有趣的话题，看看什么情况下，会出现这个现象。

需要说明的是，如果MySQL数据库本身就有很大的压力，导致数据库服务器CPU占用率很高或ioutil（IO利用率）很高，这种情况下所有语句的执行都有可能变慢，不属于我们今天的讨论范围。

为了便于描述，我还是构造一个表，基于这个表来说明今天的问题。这个表有两个字段id和c，并且我在里面插入了10万行记录。

```
mysql> CREATE TABLE `t` (  
  `id` int(11) NOT NULL,  
  `c` int(11) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB;  
  
delimiter ;;  
create procedure idata()  
begin  
  declare i int;  
  set i=1;  
  while(i<=100000) do  
    insert into t values(i,i);  
    set i=i+1;  
  end while;  
end;;  
delimiter ;  
  
call idata();
```

接下来，我会用几个不同的场景来举例，有些是前面的文章中我们已经介绍过的知识点，你看看能不能一眼看穿，来检验一下吧。

第一类：查询长时间不返回

如图1所示，在表t执行下面的SQL语句：

```
mysql> select * from t where id=1;
```

查询结果长时间不返回。

```
mysql> select * from t where id=1;
```

图1 查询长时间不返回

一般碰到这种情况的话，大概率是表t被锁住了。接下来分析原因的时候，一般都是首先执行一下show processlist命令，看看当前语句处于什么状态。

然后我们再针对每种状态，去分析它们产生的原因、如何复现，以及如何处理。

等MDL锁

如图2所示，就是使用show processlist命令查看Waiting for table metadata lock的示意图。

```
mysql> show processlist;
```

Id	User	Host	db	Command	Time	State	Info
5	root	localhost:61558	test	Query	0	init	show processlist
7	root	localhost:63852	test	Sleep	31		NULL
8	root	localhost:63870	test	Query	25	Waiting for table metadata lock	select * from t where id=1

3 rows in set (0.00 sec)

图2 Waiting for table metadata lock状态示意图

出现这个状态表示的是，现在有一个线程正在表t上请求或者持有MDL写锁，把select语句堵住了。

在第6篇文章[《全局锁和表锁：给表加个字段怎么有这么多阻碍？》](#)中，我给你介绍过一种复现方法。但需要说明的是，那个复现过程是基于MySQL 5.6版本的。而MySQL 5.7版本修改了MDL的加锁策略，所以就不能复现这个场景了。

不过，在MySQL 5.7版本下复现这个场景，也很容易。如图3所示，我给出了简单的复现步骤。

session A	session B
lock table t write;	
	select * from t where id=1;

图3 MySQL 5.7中Waiting for table metadata lock的复现步骤

session A 通过lock table命令持有表t的MDL写锁，而session B的查询需要获取MDL读锁。所以，session B进入等待状态。

这类问题的处理方式，就是找到谁持有MDL写锁，然后把它kill掉。

但是，由于在show processlist的结果里面，session A的Command列是“Sleep”，导致查找起来很不方便。不过有了performance_schema和sys系统库以后，就方便多了。（MySQL启动时需要设置performance_schema=on，相比于设置为off会有10%左右的性能损失）

通过查询sys.schema_table_lock_waits这张表，我们就可以直接找出造成阻塞的process id，把这个连接用kill 命令断开即可。

```
mysql> select blocking_pid from sys.schema_table_lock_waits;
+-----+
| blocking_pid |
+-----+
|          4 |
+-----+
```

图4 查获加表锁的线程id

等flush

接下来，我给你举另外一种查询被堵住的情况。

我在表t上，执行下面的SQL语句：

```
mysql> select * from information_schema.processlist where id=1;
```

这里，我先卖个关子。

你可以看一下图5。我查出来这个线程的状态是Waiting for table flush，你可以设想一下这是什么原因。

```
mysql> select * from information_schema.processlist where id=6;
+----+-----+-----+-----+-----+-----+-----+-----+
| ID | USER | HOST           | DB   | COMMAND | TIME | STATE                               | INFO                               |
+----+-----+-----+-----+-----+-----+-----+-----+
| 6  | root | localhost:47074 | test | Query   | 622  | Waiting for table flush             | select * from t where id=1 |
+----+-----+-----+-----+-----+-----+-----+-----+
```

图5 Waiting for table flush状态示意图

这个状态表示的是，现在有一个线程正要对表t做flush操作。MySQL里面对表做flush操作的方法，一般有以下两个：

```
flush tables t with read lock;
```

```
flush tables with read lock;
```

这两个flush语句，如果指定表t的话，代表的是只关闭表t；如果没有指定具体的表名，则表示关闭MySQL里所有打开的表。

但是正常这两个语句执行起来都很快，除非它们也被别的线程堵住了。

所以，出现Waiting for table flush状态的可能情况是：有一个flush tables命令被别的语句堵住

了，然后它又堵住了我们的select语句。

现在，我们一起来复现一下这种情况，复现步骤如图6所示：

session A	session B	session C
select sleep(1) from t;		
	flush tables t;	
		select * from t where id=1;

图6 Waiting for table flush的复现步骤

在session A中，我故意每行都调用一次sleep(1)，这样这个语句默认要执行10万秒，在这期间表t一直是被session A“打开”着。然后，session B的flush tables t命令再要去关闭表t，就需要等session A的查询结束。这样，session C要再次查询的话，就会被flush 命令堵住了。

图7是这个复现步骤的show processlist结果。这个例子的排查也很简单，你看到这个show processlist的结果，肯定就知道应该怎么做了。

```
mysql> show processlist;
```

Id	User	Host	db	Command	Time	State	Info
4	root	localhost:49548	test	Query	38	User sleep	select sleep(1) from t
5	root	localhost:49604	test	Query	35	Waiting for table flush	flush tables t
6	root	localhost:49634	test	Query	30	Waiting for table flush	select * from t where id=1
7	root	localhost:49726	test	Query	0	starting	show processlist

图 7 Waiting for table flush的show processlist 结果

等行锁

现在，经过了表级锁的考验，我们的select 语句终于来到引擎里了。

```
mysql> select * from t where id=1 lock in share mode;
```

上面这条语句的用法你也很熟悉了，我们在第8篇[《事务到底是隔离的还是不隔离的？》](#)文章介绍当前读时提到过。

由于访问id=1这个记录时要加读锁，如果这时候已经有一个事务在这行记录上持有一个写锁，我们的select语句就会被堵住。

复现步骤和现场如下：

session A	session B
begin; update t set c=c+1 where id=1;	
	select * from t where id=1 lock in share mode;

图 8 行锁复现

```
mysql> show processlist;
+-----+-----+-----+-----+-----+-----+-----+-----+
| Id | User | Host | db | Command | Time | State | Info |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 4 | root | localhost:65224 | test | Query | 0 | starting | show processlist |
| 8 | root | localhost:10354 | test | Query | 1 | statistics | select * from t where id=1 lock in share mode |
| 10 | root | localhost:11276 | test | Sleep | 52 | | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

图 9 行锁show processlist 现场

显然，**session A**启动了事务，占有写锁，还不提交，是导致**session B**被堵住的原因。

这个问题并不难分析，但问题是怎么查出是谁占着这个写锁。如果你用的是MySQL 5.7版本，可以通过sys.innodb_lock_waits 表查到。

查询方法是：

```
mysql> select * from t sys.innodb_lock_waits where locked_table='test'.t'\G
```

```
mysql> select * from sys.innodb_lock_waits where locked_table='`test`.`t`'\G
***** 1. row *****
      wait_started: 2018-12-13 20:12:35
      wait_age: 00:00:08
      wait_age_secs: 8
      locked_table: `test`.`t`
      locked_index: PRIMARY
      locked_type: RECORD
      waiting_trx_id: 421668144410224
      waiting_trx_started: 2018-12-13 20:12:35
      waiting_trx_age: 00:00:08
      waiting_trx_rows_locked: 1
      waiting_trx_rows_modified: 0
      waiting_pid: 8
      waiting_query: select * from t where id=1 lock in share mode
      waiting_lock_id: 421668144410224:23:4:2
      waiting_lock_mode: S
      blocking_trx_id: 1101302
      blocking_pid: 4
      blocking_query: NULL
      blocking_lock_id: 1101302:23:4:2
      blocking_lock_mode: X
      blocking_trx_started: 2018-12-13 20:01:57
      blocking_trx_age: 00:10:46
      blocking_trx_rows_locked: 1
      blocking_trx_rows_modified: 1
      sql_kill_blocking_query: KILL QUERY 4
      sql_kill_blocking_connection: KILL 4
1 row in set, 3 warnings (0.00 sec)
```

图10 通过sys.innodb_lock_waits 查行锁

可以看到，这个信息很全，4号线程是造成堵塞的罪魁祸首。而干掉这个罪魁祸首的方式，就是KILL QUERY 4或KILL 4。

不过，这里不应该显示“KILL QUERY 4”。这个命令表示停止4号线程当前正在执行的语句，而这个方法其实是没有用的。因为占有行锁的是update语句，这个语句已经是之前执行完成了的，现在执行KILL QUERY，无法让这个事务去掉id=1上的行锁。

实际上，KILL 4才有效，也就是说直接断开这个连接。这里隐含的一个逻辑就是，连接被断开的时候，会自动回滚这个连接里面正在执行的线程，也就释放了id=1上的行锁。

第二类：查询慢

经过了重重封“锁”，我们再来看看一些查询慢的例子。

先来看一条你一定知道原因的SQL语句：

```
mysql> select * from t where c=50000 limit 1;
```


由于字段c上没有索引，这个语句只能走id主键顺序扫描，因此需要扫描5万行。

作为确认，你可以看一下慢查询日志。注意，这里为了把所有语句记录到slow log里，我在连接后先执行了 `set long_query_time=0`，将慢查询日志的时间阈值设置为0。

```
# Query_time: 0.011543 Lock_time: 0.000104 Rows_sent: 1 Rows_examined: 50000
SET timestamp=1544723147;
select * from t where c=50000 limit 1;
```

图11 全表扫描5万行的slow log

Rows_examined显示扫描了50000行。你可能会说，不是很慢呀，11.5毫秒就返回了，我们线上一一般都配置超过1秒才算慢查询。但你要记住：**坏查询不一定是慢查询**。我们这个例子里面只有10万行记录，数据量大起来的话，执行时间就线性涨上去了。

扫描行数多，所以执行慢，这个很好理解。

但是接下来，我们再看一个只扫描一行，但是执行很慢的语句。

如图12所示，是这个例子的slow log。可以看到，执行的语句是

```
mysql> select * from t where id=1;
```

虽然扫描行数是1，但执行时间却长达800毫秒。

```
# User@Host: root[root] @ localhost [127.0.0.1] Id: 5
# Query_time: 0.804400 Lock_time: 0.000205 Rows_sent: 1 Rows_examined: 1
SET timestamp=1544728393;
```

图12 扫描一行却执行得很慢

是不是有点奇怪呢，这些时间都花在哪里了？

如果我把这个slow log的截图再往下拉一点，你可以看到下一个语句，`select * from t where id=1 lock in share mode`，执行时扫描行数也是1行，执行时间是0.2毫秒。

```
# Query_time: 0.000258 Lock_time: 0.000132 Rows_sent: 1 Rows_examined: 1
SET timestamp=1544728398;
select * from t where id=1 lock in share mode;
```

图 13 加上lock in share mode的slow log

看上去是不是更奇怪了？按理说lock in share mode还要加锁，时间应该更长才对啊。

可能有的同学已经有答案了。如果你还没有答案的话，我再给你一个提示信息，图14是这两个

语句的执行输出结果。

```
mysql> select * from t where id=1;
+----+-----+
| id | c      |
+----+-----+
| 1  | 1      |
+----+-----+
1 row in set (0.81 sec)

mysql> select * from t where id=1 lock in share mode;
+----+-----+
| id | c      |
+----+-----+
| 1  | 1000001 |
+----+-----+
1 row in set (0.00 sec)
```

图14 两个语句的输出结果

第一个语句的查询结果里c=1，带lock in share mode的语句返回的是c=1000001。看到这里应该有更多的同学知道原因了。如果你还是没有头绪的话，也别着急。我先跟你说明一下复现步骤，再分析原因。

session A	session B
start transaction with consistent snapshot;	
	update t set c=c+1 where id=1; //执行100万次
select * from t where id=1;	
select * from t where id=1 lock in share mode;	

图15 复现步骤

你看到了，session A先用start transaction with consistent snapshot命令启动了一个事务，之后session B才开始执行update 语句。

session B执行完100万次update语句后，id=1这一行处于什么状态呢？你可以从图16中找到答案。

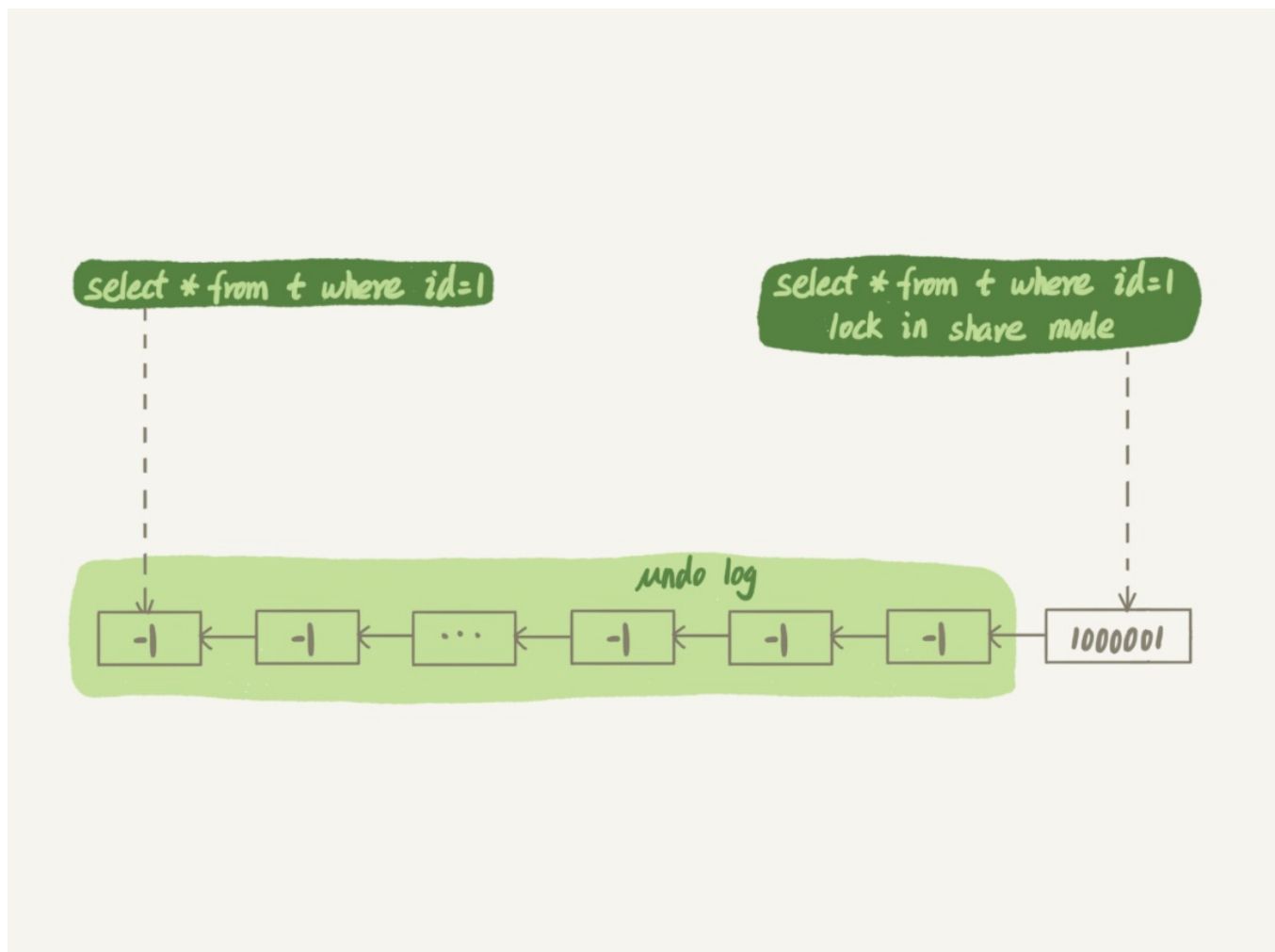


图16 id=1的数据状态

session B更新完100万次，生成了100万个回滚日志(undo log)。

带lock in share mode的SQL语句，是当前读，因此会直接读到1000001这个结果，所以速度很快；而select * from t where id=1这个语句，是一致性读，因此需要从1000001开始，依次执行undo log，执行了100万次以后，才将1这个结果返回。

注意，undo log里记录的其实是“把2改成1”，“把3改成2”这样的操作逻辑，画成减1的目的是方便你看图。

小结

今天我给你举了在一个简单的表上，执行“查一行”，可能会出现的被锁住和执行慢的例子。这其中涉及到了表锁、行锁和一致性读的概念。

在实际使用中，碰到的场景会更复杂。但大同小异，你可以按照我在文章中介绍的定位方法，来定位并解决问题。

最后，我给你留一个问题吧。

我们在举例加锁读的时候，用的是这个语句，select * from t where id=1 lock in share mode。由

于id上有索引，所以可以直接定位到id=1这一行，因此读锁也是只加在了这一行上。

但如果是下面的SQL语句，

```
begin;
select * from t where c=5 for update;
commit;
```

这个语句序列是怎么加锁的呢？加的锁又是什么时候释放呢？

你可以把你的观点和验证方法写在留言区里，我会在下一篇文章的末尾给出我的参考答案。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

在上一篇文章最后，我留给你的问题是，希望你可以分享一下之前碰到过的、与文章中类似的场景。

@封建的风 提到一个有趣的场景，值得一说。我把他的问题重写一下，表结构如下：

```
mysql> CREATE TABLE `table_a` (
  `id` int(11) NOT NULL,
  `b` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `b` (`b`)
) ENGINE=InnoDB;
```

假设现在表里面，有100万行数据，其中有10万行数据的b的值是'1234567890'，假设现在执行语句是这么写的：

```
mysql> select * from table_a where b='1234567890abcd';
```

这时候，MySQL会怎么执行呢？

最理想的情况是，MySQL看到字段b定义的是varchar(10)，那肯定返回空呀。可惜，MySQL并没有这么做。

那要不，就是把'1234567890abcd'拿到索引里面去做匹配，肯定也没能够快速判断出索引树b上并没有这个值，也很快就能返回空结果。

但实际上，MySQL也不是这么做的。

这条SQL语句的执行很慢，流程是这样的：

1. 在传给引擎执行的时候，做了字符截断。因为引擎里面这个行只定义了长度是10，所以只截了前10个字节，就是'1234567890'进去做匹配；
2. 这样满足条件的数据有10万行；
3. 因为是select *， 所以要做10万次回表；
4. 但是每次回表以后查出整行，到server层一判断，b的值都不是'1234567890abcd'；
5. 返回结果是空。

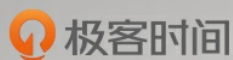
这个例子，是我们文章内容的一个很好的补充。虽然执行过程中可能经过函数操作，但是最终在拿到结果后，server层还是要做一轮判断的。

评论区留言点赞板：

@赖阿甘 提到了等号顺序问题，时间上MySQL优化器执行过程中，where 条件部分，a=b和b=a的写法是一样的。

@沙漠里的骆驼 提到了一个常见的问题。相同的模板语句，但是匹配行数不同，语句执行时间相差很大。这种情况，在语句里面有order by这样的操作时会更明显。

@Justin 回答了我们正文中的问题，如果id的类型是整数，传入的参数类型是字符串的时候，可以用上索引。



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。



某、人

👍 39

最近几张干货越来越多了,很实用,收获不少.先回答今天的问题

版本5.7.13

rc模式下:

session 1:

begin;

select * from t where c=5 for update;

session 2:

delete from t where c=10 --等待

session 3:

insert into t values(100001,8) --成功

session 1:

commit

session 2:事务执行成功

rr模式下:

begin;

select * from t where c=5 for update;

session 2:

delete from t where c=10 --等待

session 3:

insert into t values(100001,8) --等待

session 1:

commit

session 2:事务执行成功

session 3: 事务执行成功

从上面这两个简单的例子,可以大概看出上锁的流程.

不管是rr模式还是rc模式,这条语句都会先在server层对表加上MDL S锁,然后进入到引擎层。

rc模式下,由于数据量不大只有10W。通过实验可以证明session 1上来就把该表的所有行都锁住了。

导致其他事务要对该表的所有现有记录做更新,是阻塞状态。为什么insert又能成功?

说明rc模式下for update语句没有上gap锁,所以不阻塞insert对范围加插入意向锁,所以更新成功。

session 1commit后,session 2执行成功。表明所有行的x锁是在事务提交完成以后才释放。

rr模式下,session 1和session 2与rc模式下都一样,说明rr模式下也对所有行上了X锁。

唯一的区别是insert也等待了,是因为rr模式下对没有索引的更新,聚簇索引上的所有记录,都被加上了X锁。其次,聚簇索引每条记录间的间隙(GAP),也同时被加上了GAP锁。由于gap锁阻

塞了insert要加的插入意向锁,导致insert也处于等待状态。只有当session 1 commit完成以后。session 1上的所有锁才会释放,S2,S3执行成功

由于例子中的数据量还比较小,如果数据量达到千万级别,就比较直观的能看出,上锁是逐行上锁的一个过程.扫描一条上一条,直到所有行扫描完,rc模式下对所有行上x锁。rr模式下不仅对所有行上X锁,还对所有区间上gap锁.直到事务提交或者回滚完成后,上的锁才会被释放。

2018-12-26

作者回复

分析得非常好。

两个模式下，各增加一个session 4 : update t set c=100 where id=10看看哦

基本就全了

2018-12-26



薛畅

18

回来老师的问题：

在 Read Committed 隔离级别下，会锁上聚簇索引中的所有记录；

在 Repeatable Read 隔离级别下，会锁上聚簇索引中的所有记录，并且会锁上聚簇索引内的所有 GAP；

在上面两个隔离级别的情况下，如果设置了 innodb_locks_unsafe_for_binlog 开启 semi-consistent read 的话，对于不满足查询条件的记录，MySQL 会提前放锁，不过加锁的过程是不可避免的。

2018-12-26



似水流年

16

请问老师，为什么select blocking_pid from sys.schema_table_lock_waits;查不到mdl锁的进程id，显示为空。

2018-12-28



沙漠里的骆驼

11

@高枕

这里有些资料提供给你参考：

1. 何登成的技术博客: 加锁分析 <http://hedengcheng.com/?p=771>

2. 锁的常见种类: <http://www.aneasystone.com/archives/2017/11/solving-dead-locks-two.html>

2018-12-26



老杨同志

5

愉快的做一下思考题

begin;

select * from t where c=5 for update;

commit;

历史知识的结论是，innodb先锁全表的所有行，返回server层，判断c是否等于5，然后释放c!=5的行锁。

验证方法:

事务A执行 锁住一行c!=5的记录 比如id=3 c=3

`select * from t where id = 3 for update` 或者 `update t set c=4 where id =3`

然后启动新事务B执行上面的语句`select * from t where c=5 for update;` 看看有没有被阻塞。

用于判断事务B的语句会不会试图锁不满足条件的记录。

然后把事务A和事务B的执行顺序对调一下, 也就是先执行B在执行A。看看有没有阻塞, 判断在事务B加锁成功的情况下会不会释放不满足查询条件记录的行锁。

2018-12-26

作者回复

思路清晰

隔离级别再愉快地改成RR试试

2018-12-26



小确幸

5

问一下: 索引扫描与全表扫描, 有什么异同点?

2018-12-26

作者回复

一般说全表扫描默认是值“扫描主键索引”

2018-12-26



蠢蠢欲动的腹肌

4

老师, 您好

我的mysql版本5.7.24, 尝试的时候发现了如下问题

锁住了表T

```
mysql> lock table T write;
```

```
Query OK, 0 rows affected (0.00 sec)
```

另一个terminal查询时被阻塞, 但是查不到blocking_pid, 这是什么情况呢

```
mysql> select blocking_pid from sys.schema_table_lock_waits;
```

```
Empty set (0.00 sec)
```

ps:发现查询schema_table_lock_waits表与lock table的语句不能放在一个terminal执行, 否则会报

```
Table 'schema_table_lock_waits' was not locked with LOCK TABLES
```

自行尝试的同学要注意下, 老师有空的话也可以帮看看为什么。。。

2018-12-28



尘封

4

课后问题: d这一列不存在, 但是还是要加MDL锁, 释放时间应该是事务提交时。

2018-12-26

作者回复

抱歉, 是要写成where c=5, 发起堪误了

2018-12-26



Tony Du

👍 3

对于课后问题，`select * from t where c=5 for update`,

当级别为RR时，因为字段c上没有索引，会扫主键索引，这时会把表中的记录都加上X锁。同时，因为对于innodb来说，当级别为RR时，是可以解决幻读的，此时对于每条记录的间隙还要加上GAP锁。也就是说，表上每一条记录和每一个间隙都锁上了。

当级别为RC时，因为字段c上没有索引，会扫主键索引，这时会把表中的记录都加上X锁。

另外，之前看过相关文章，MySQL在实际实现中有些优化措施，比如当RC时，在MySQL server过滤条件，发现不满足后，会把不满足条件的记录释放锁（这里就是把 `c!=5` 的记录释放锁），这里会违背两阶段的约束。当然，之前每条记录的加锁操作还是不能省略的。

还有，对于semi consistent read开启的情况下，也会提前释放锁。

2018-12-27



张永志

👍 3

RC隔离级别下，对非索引字段更新，有个锁全表记录的过程，不符合条件的会及时释放行锁，不必等事务结束时释放；而直接用索引列更新，只会锁索引查找值和行。

2018-12-27



尘封

👍 3

老师，有没有遇到过select语句一直处于killed状态的情况？

2018-12-26

作者回复

有，这个是在后面的文章中会用到的例子

2018-12-26



浩涛

👍 2

老师，我们公司今年打算迁数据库，也就是oracle迁到mysql，目前要决定选用哪个版本和分支，不知道要怎么分析，您有什么好的建议吗？

2019-03-11

作者回复

目前5.7是比较主流的，就用最新GA版本吧，记得做全量回归测试

2019-03-13



xm

👍 2

@Elvis

Elvis

👍

0

老师，最近项目mysql遇到一个难题，表数据每天230万，一条语句的查询条件有1-40个，最坏情况下40，请问老师有没有好的建议，非常感谢

2018-12-27

作者回复

这个是索引最头大的问题之一了... 如果有明显的某类组合查询是最多的还好, 否则确实很麻烦。

我觉得应对这种大数据量的多条件的查询的话换种思路, 将mysql复杂的组合查询条件导入到es中作为key, 主键id作为value, 复杂的查询经过es后得到主键id, 之后走mysql会好很多, 目前公司是这样做的, 老师觉得怎么样?

2018-12-28

作者回复

嗯, 这个是一种思路, 见过这么干的

2018-12-28



小李子

2

老师, 为什么session B 执行了 `select in share mode`, 在等行锁的时候, session C 执行 `select * from sys.innodb_lock_waits where locked_table='test`.`t`'` 会报这个错
[Err] 1356 - View 'sys.innodb_lock_waits' references invalid table(s) or column(s) or function(s) or definer/invoke of view lack rights to use them, 而超时之后, 又可以查了? 另外, \G 参数会报语法错误?

2018-12-27



张永志

2

RR隔离级别下, 为保证binlog记录顺序, 非索引更新会锁住全表记录, 且事务结束前不会对不符合条件记录有逐步释放的过程。

2018-12-27

作者回复

准确

2018-12-27



信信

2

老师你好, 图3上方提到MySQL 5.7 版本修改了 MDL 的加锁策略, 不能复现第六章的场景。但我认为只要仍然满足: DML操作加MDL读锁, DDL操作加MDL写锁, 并且事务提交才释放锁, 那么就可以复现啊。。。所以5.7到底是改了什么导致无法复现的呢?

2018-12-27



某、人

2

老师我请教一个问题:

flush tables中close table的意思是说的把open_tables里的表全部关闭掉?下次如果有关于某张表的操作

又把frm file缓存进Open_table_definitions,把表名缓存到open_tables,还是open_table只是一个计数?

不是特别明白flush table和打开表是个什么流程

2018-12-26

作者回复

Flush tables是会关掉表，然后下次请求重新读表信息的

第一次打开表其实就是open_table_definitions，包括读表信息一类的

之后再查询就是拷贝一个对象，加一个计数这样的

2018-12-26



陈旭

👍 2

老师，最近遇到了一个问题，看您有什么建议。

业务场景是这样的：

- 1.开启事务
- 2.在表a插入一条记录
- 3.在表b更新一条记录
- 4.在表c更新一条记录
- 5.提交事务

看程序日志所有sql都没问题（没看数据库日志），但是结果是2的那条插入了，3和4都没更新，这个问题有哪几种情况？

2018-12-26

作者回复

这是被别的并发事务又改回去了吗？

要么是update的值跟原值相同

要么是update条件没有匹配到行

额，最好给一下每个语句执行后的affected rows，还有binlog里的日志内容，才好分析

2018-12-26



godtrue

👍 1

课前思考

1：为啥只查一行的语句，也执行这么慢？

查的慢，基本上就是索引使用的问题，和查一行还是N行(N不是巨大)，没有必然联系。查一行慢，猜测没有走索引查询，且数据量比较大。

课后思考

1：阅后发现自己的无知，只查询一行的语句，也比较慢，原因从大到小可分为三种情况？

第一MySQL数据库本身被堵住了，比如：系统或网络资源不够

第二SQL语句被堵住了，比如：表锁，行锁等，导致存储引擎不执行对应的SQL语句

第三确实是索引使用不当，没有走索引

第四是表中数据的特点导致的，走了索引，但回表次数庞大

感谢老师的分享，真是醍醐灌顶呀！

2019-07-31



残天噬魂

👍 1

为啥我开启了性能监控，按照案例1执行加锁，select * from sys.schema_table_lock_waits;这个没有结果啊

2019-07-09

