目录

[6.2 锁问题排查 1](#_Toc742)

[6.2.1　找出谁持有全局读锁 1](#_Toc16995)

[6.2.2　找出谁持有MDL锁 5](#_Toc18699)

[6.2.3　找出谁持有表级锁 8](#_Toc17964)

**6.2 锁问题排查**

**注：本节主要讨论的是这样一个问题：由于mysql服务器同一时间会建立众多会话，假如某个对象(整个数据库或某个表或某个数据行)被某个会话锁定但迟迟没有被解锁，导致其他会话无法执行有关操作，这时该如何找出哪个会话持有锁？**

**6.2.1　找出谁持有全局读锁**

#全局读锁通常是由flush table with read lock；这类语句添加的。

**注：解锁语句为unlock tables；主从架构下的备份会用到这种上锁手法。**

# MySQL 5.7版本开始提供了performance\_schema.metadata\_locks表，用来记录一些Server层的锁信息（包括全局读锁和MDL锁等）。

#下面通过一个示例来演示如何使用performance\_schema找出谁持有全局读锁。

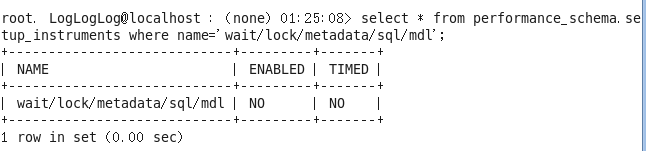
**注：注意实验之前查看MDL(metadata lock)记录是否已经开启：**

mysql> select \* from performance\_schema.setup\_instruments where name='wait/lock/metadata/sql/mdl';

图形用户界面, 文本, 应用程序

描述已自动生成

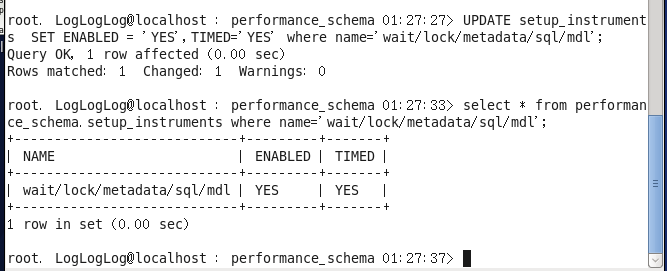
MDL记录未开启



**如果没有就要开启事件收集：**

mysql> UPDATE setup\_instruments SET ENABLED = 'YES',TIMED='YES' where name='wait/lock/metadata/sql/mdl';

开启事件收集



**相关介绍见5.3.4节介绍**

#全局读锁通常是由flush table with read lock；这类语句添加的。各种备份工具为了得到一致性备份，以及在具有主从复制架构的环境中做主备切换时常常使用这类语句。最难排查的一种情况，就是线上系统权限约束不规范，各种人员使用的数据库账号都具有RELOAD权限时(**注：执行flush语句**)，都可以对数据库加全局读锁。

#MySQL 5.7版本开始提供了performance\_schema.metadata\_locks表，用来记录一些Server层的锁信息（包括全局读锁和MDL锁等）。

**注：以下部分截图前后经过若干次操作不断补充，所以在一些细节上不一致。**

#开启**第一个**会话，执行加全局读锁的语句

mysql> flush table with read lock;



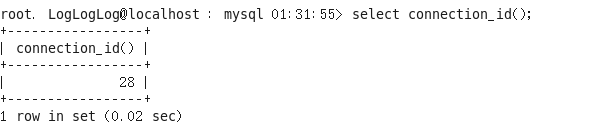


# 查询以下加锁线程的process id，以便与后续排查过程对应

mysql> select connection\_id();

文本

低可信度描述已自动生成



#开启**第二个**会话，执行可能对数据造成修改的任意语句

mysql> use sysbench\_test



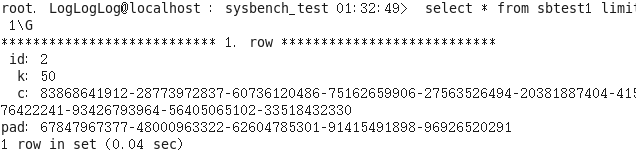


**注：实验室环境中测试数据库名为sysbench\_test**

mysql> select \* from sbtest1 limit 1\G

图形用户界面, 文本, 应用程序

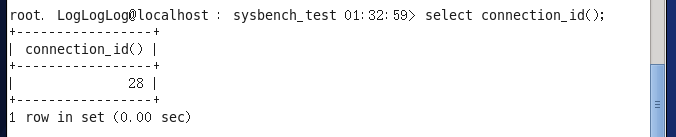
描述已自动生成



**注：仍然可以查询操作。但下面的更新操作被阻塞。**

mysql> select connection\_id();

文本

中度可信度描述已自动生成

mysql> update sbtest1 set pad='xxx' where id=21; # 操作被阻塞





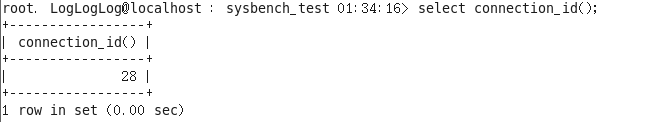
**注：根据实际情况操作，不一定有id=21的记录。上面查到的是id=2，所以这里就更新这条记录。**

#开启**第三个**会话，开始使用一些手段进行排查

mysql> select connection\_id();

图片包含 图表

描述已自动生成

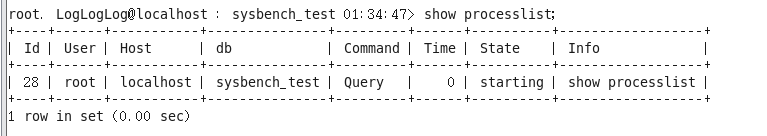


# 查询processlist信息，这里只能看到process id为5的线程State为Waiting for global read lock，表示正在等待全局读锁(**注：就是下图中“waiting for global read lock”**)

**mysql> show processlist;**

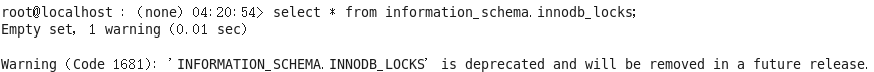
图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成



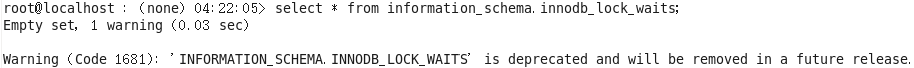
# 继续查询**information\_schema**.innodb\_locks、innodb\_lock\_waits、innodb\_trx表，发现三个表均为空

mysql> select \* from information\_schema.innodb\_locks;





mysql> select \* from information\_schema.innodb\_lock\_waits;





mysql> select \* from information\_schema.innodb\_trx\G

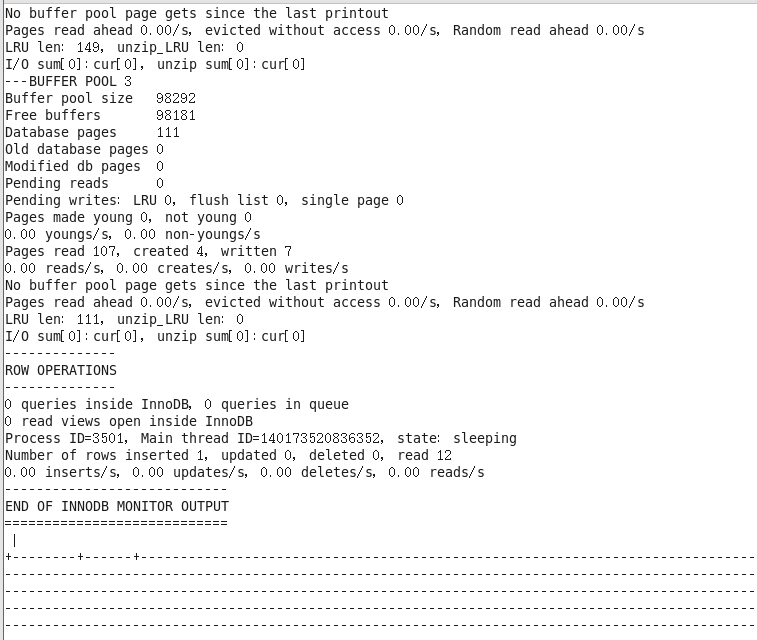




**注：查询的是information\_schema，不是performance\_schema，information\_schema存放的是元数据的内容，但这里却没有相关信息。**

# 再使用show engine innodb status;查看（这里只需要看TRANSACTION段落即可），仍然**无任何有用**的锁信息

mysql> show engine innodb status;



**注：显示内容基本都与锁无关，结果略。**

# 开启**第四个**会话。通过performance\_schema.metadata\_locks表来排查谁持有全局读锁，全局读锁在该表中通常**记录着同一个会话的OBJECT\_TYPE为global和commit、LOCK\_TYPE都为SHARED的两把显式锁**

mysql> select \* from performance\_schema.**metadata\_locks** where OWNER\_THREAD\_ID!=sys.ps\_thread\_id(connection\_id())\G

文本, 信件

描述已自动生成



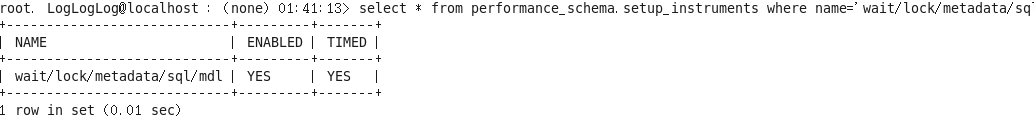
做到此处发现只有空集 但是已经开启了wait/lock/metadata/sql/mdl

未找到出错原因

**注：如果做到这里发现只有一个空集，很可能是因为一开始没有开启wait/lock/metadata/sql/mdl：**

图片包含 文本

描述已自动生成



**要按前面的介绍设置开启。可是因为前面已经加了全局锁，所以只能解锁(见本节最后)，重新来过。**

**上面语句查询与当前会话(第4个会话)对应线程无关(即第1-3会话)的锁信息，当前对应有：**

**表格

中度可信度描述已自动生成**

**可知当前会话的线程号为48。从上面的信息来看，45号线程(第1个会话)拿到了锁（LOCK\_STATUS为GRANTED），而46号线程(第2个会话)等待(LOCK\_STATUS为PENDING)。**

# 查看process id为4和5的线程各自对应的**内部线程ID**是多少

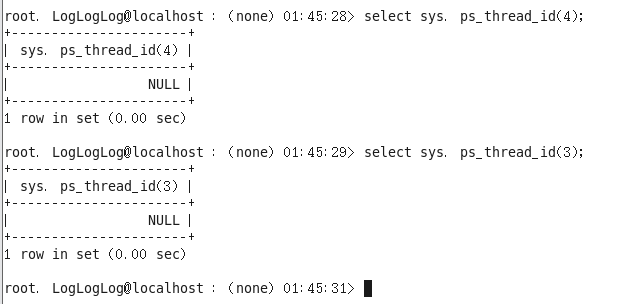
mysql> select sys.ps\_thread\_id(4);

mysql> select sys.ps\_thread\_id(5);

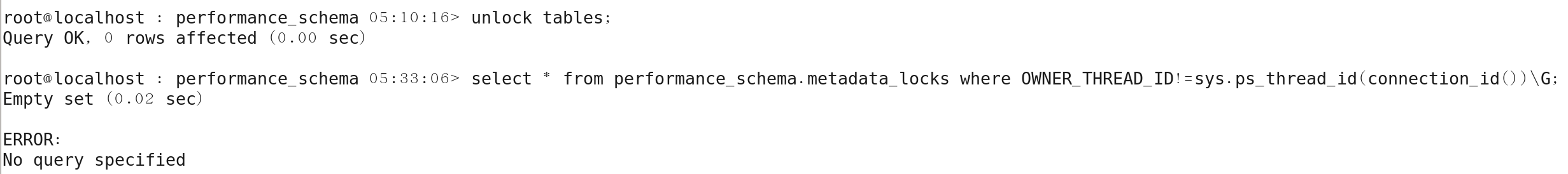
表格

中度可信度描述已自动生成

**注：根据实际我们这里是查看process id为3和4的线程，也正好印证了前面注释的说法。**



**注：实验完成后解锁：**





**之前被阻塞的线程可以发现会被执行：**

图形用户界面, 文本

中度可信度描述已自动生成



**6.2.2　找出谁持有MDL锁**

**注：MDL锁(meta data lock)，更改或定义数据表时会加上MDL锁。**

#我们可能经常会遇到被阻塞等待MDL锁的情况。例如：使用show processlist；语句查看线程信息时可能会发现State字段值为“Waiting for table metadata lock”**(注：6.2.1介绍的全局读锁就会显示“waiting for global read lock”)。**

# MDL锁记录对应的instruments为wait/lock/metadata/sql/mdl，默认未启用**（注：务必检查，方法见6.2.1节开始）**；对应的consumers为performance\_schema.metadata\_locks，在setup\_consumers中只受全局配置项global\_instrumentation控制，默认已启用

# 应该如何去排查是谁持有了MDL锁没有释放呢？下面尝试进行MDL锁的等待场景模拟。**会话1**，显式开启一个事务，并执行一条update语句更新sbtest1表**不提交**

mysql> use sbtest



**注：实验环境使用的数据库名为：sysbench\_test**

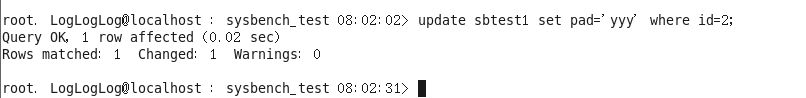
mysql> begin;

CREATE TABLE sbtest1 (id VARCHAR(50),pad VARCHAR(30) NOT NULL,name VARCHAR(50));

mysql> update sbtest1 set pad='yyy' where id=1;

文本

中度可信度描述已自动生成



**注：注意要确认表中是否有该id的记录，按实际操作。现在事务还没有提交。虽然已经Changed。**

# **会话2，**对sbtest1表执行DDL语句添加一个普通索引

mysql> use sbtest1

Database changed

mysql> alter table sbtest1 add index i\_c(c); # 被阻塞



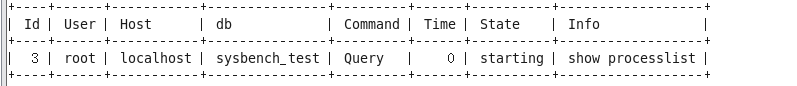


#**会话3**，使用show processlist; 语句查询线程信息，可以发现update语句正在等待MDL锁（**Waiting for table metadata lock**）。

mysql> show processlist;

图示

中度可信度描述已自动生成



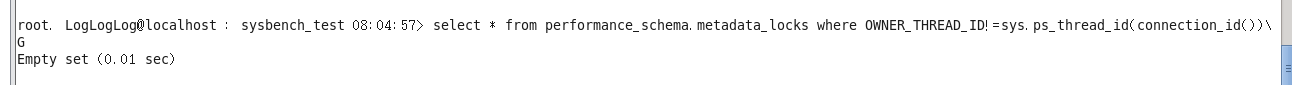
**注：可见到connection\_id为4的会话等待MDL锁。**

#通过查询performance\_schema.metadata\_locks表得知MDL锁信息

mysql> select \* from performance\_schema.metadata\_locks where OWNER\_THREAD\_ID!=sys.ps\_thread\_id(connection\_id())\G

文本

描述已自动生成



**注：上面结果表明，45号线程因为事务未提交所以占有了sbtest1表的共享写锁。然后继续看performance\_schema.metadata\_locks表的剩余结果，会发现有等待锁的现象：**

**文本

低可信度描述已自动生成**

**可以知道46等的就是45占有的锁。**

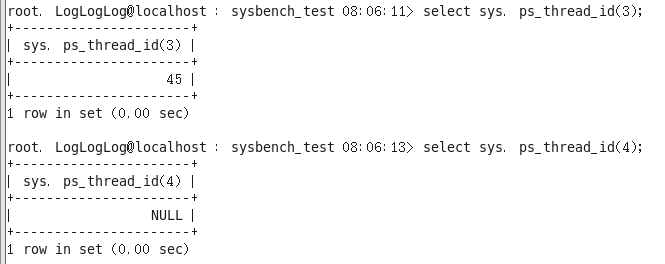
**于是，遍历会话ID反查：**

**图片包含 图表

描述已自动生成**

图片包含 文本

描述已自动生成



**而可以发现45号线程对应的正是会话3，46号线程对应会话4。这时可以继续排查了解为何会话3一直不提交事务了。会话3一直在sleep。**# 查询**information\_schema**.innodb\_trx表，确认该线程是否存在一个没有提交的事务。

mysql> select \* from information\_schema.innodb\_trx\G

. 文本

描述已自动生成



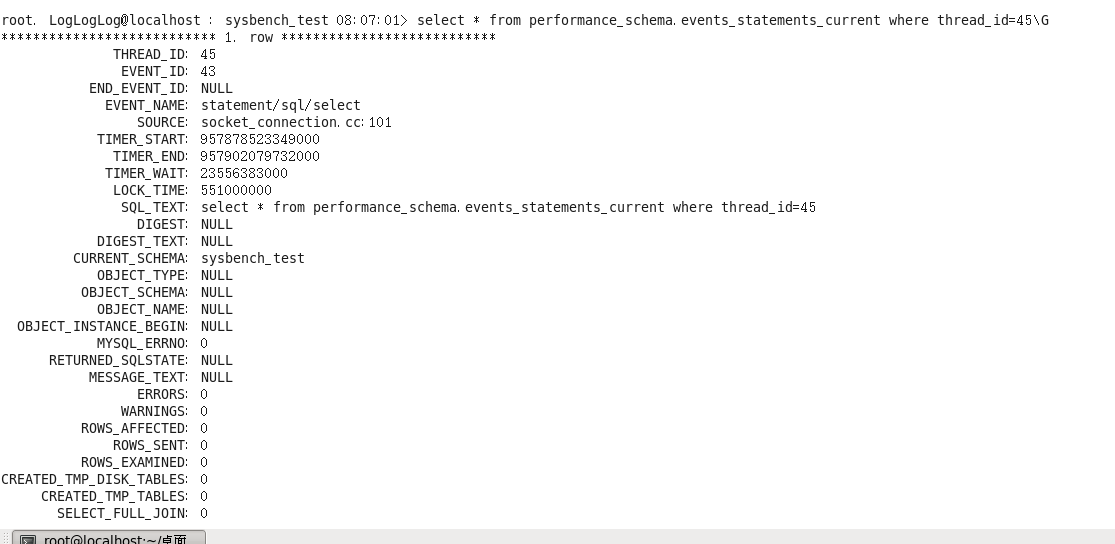
#从我们掌握的所有数据信息来看，虽然知道是线程的事务没有提交导致的发生MDL锁等待，但是并不知道线程正在做什么事情。我们当然可以杀掉线程，让等待线程继续往下执行，但是不知道占锁线程在执行什么语句，就无法找到相关的开发人员进行**优化**。

#借助**performance\_schema**.events\_statements\_current表来查询某个线程**正在执行**或者说最后一次执行完成的语句事件信息

mysql> select \* from performance\_schema.events\_statements\_current where thread\_id=45\G

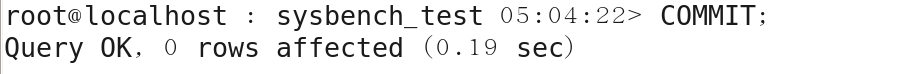
文本

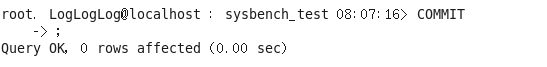
低可信度描述已自动生成



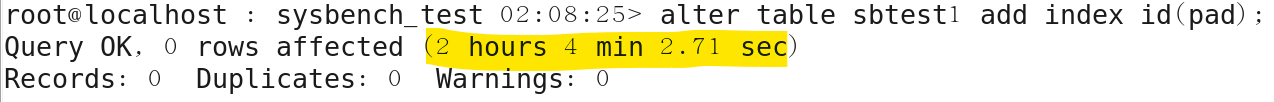
#通过SQL\_TEXT字段可以清晰地看到该线程正在执行的SQL语句是什么。如果是生产环境，那么现在你可以去**找相关的开发人员进行交涉**，下次碰到类似的语句必须及时提交，避免再次发生类似的问题。

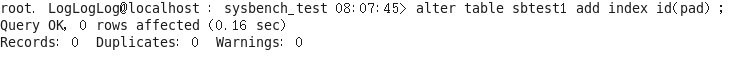
**补充：最后在会话1(即connection\_id=3的会话)中提交事务：**





**然后在会话2(即connection\_id=4的会话)中见到一直被阻塞的线程可以继续执行。**

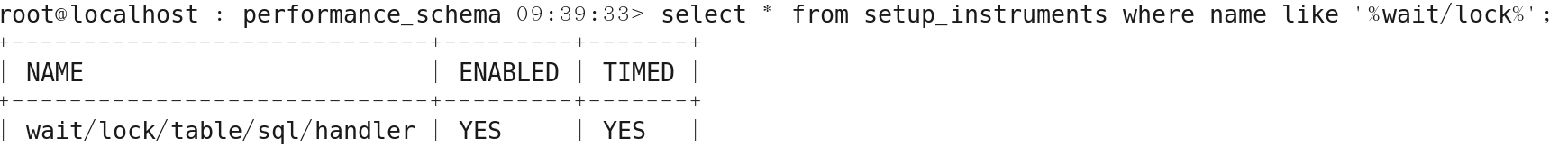




**6.2.3　找出谁持有表级锁**

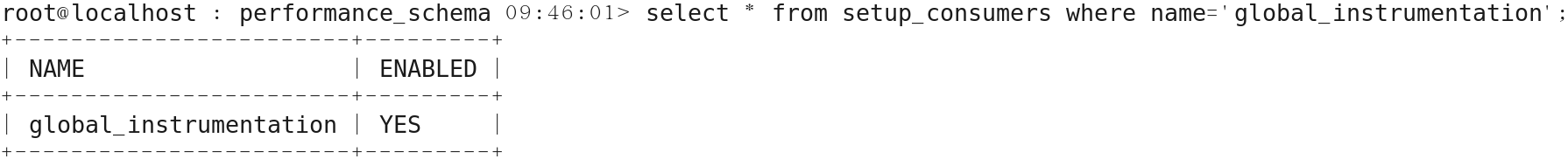
**注：顾名思义表级锁就是对整个表上锁。**

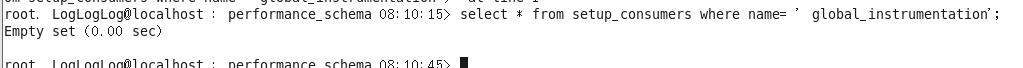
**注：注意要按6.2.2前期准备MDL(metadata lock)记录是否已经开启。下面补充书中内容**：在默认情况下，只需要设置系统配置参数performance\_schema=ON。表级锁对应的instruments（wait/lock/table/sql/handler）默认已启用：





对应的consumers为performance\_schema.table\_handles，在setup\_consumers中只受全局配置项global\_instrumentation控制：





**# 会话1，加表级锁**

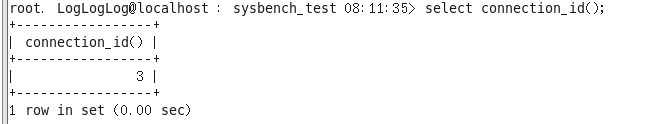
mysql> use sysbench\_test

Database changed

mysql> select connection\_id();

文本

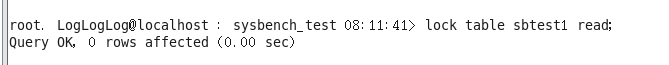
描述已自动生成

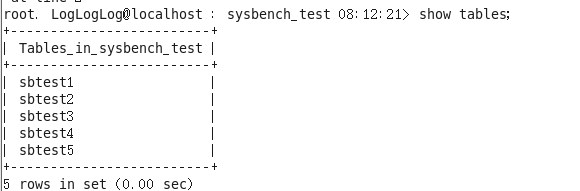


mysql> lock **table** sbtest1 read;

表格

低可信度描述已自动生成





**# 会话2，对该表执行update更新**

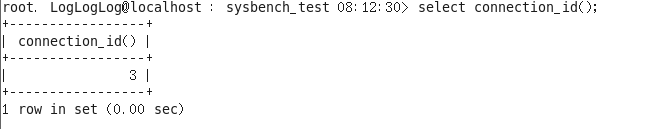
mysql> use sbtest

Database changed

mysql> select connection\_id();

文本

低可信度描述已自动生成



mysql> update sbtest1 set pad='xxx' where id=1; # 被阻塞

表格

描述已自动生成

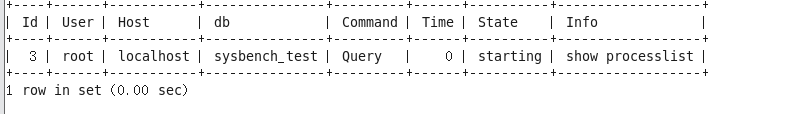


**补充：开启第三个会话（会话3）**

mysql> show processlist;

图片包含 文本

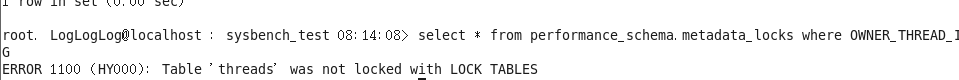
描述已自动生成



mysql> select \* from performance\_schema.metadata\_locks where OWNER\_THREAD\_ID!=sys.ps\_thread\_id(connection\_id())\G

文本

描述已自动生成



**注：SHARED\_READ\_ONLY可以共享读但锁后不能写。实际生产环境会有大量这样的锁信息而难以排查**

mysql> select \* from information\_schema.INNODB\_TRX;

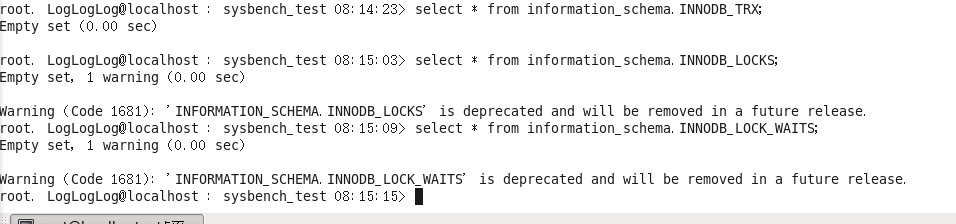
Empty set (0.00 sec)

mysql> select \* from information\_schema.INNODB\_LOCKS;

Empty set, 1 warning (0.00 sec)

mysql> select \* from information\_schema.INNODB\_LOCK\_WAITS;

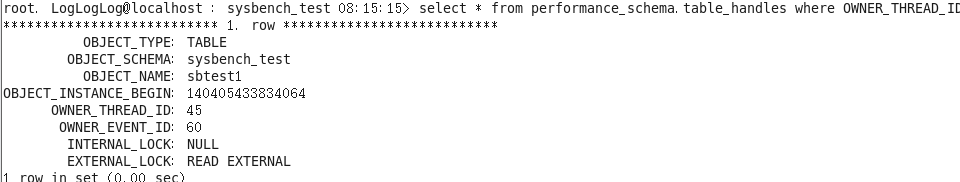
Empty set, 1 warning (0.00 sec)



mysql> select \* from performance\_schema.**table\_handles** where OWNER\_THREAD\_ID!=0\G

图形用户界面, 文本, 应用程序

描述已自动生成



**注：通过performance\_schema.table\_handles查询谁持有某个表。**

mysql> select \* from performance\_schema.events\_statements\_current where thread\_id=113\G

图形用户界面, 文本, 应用程序

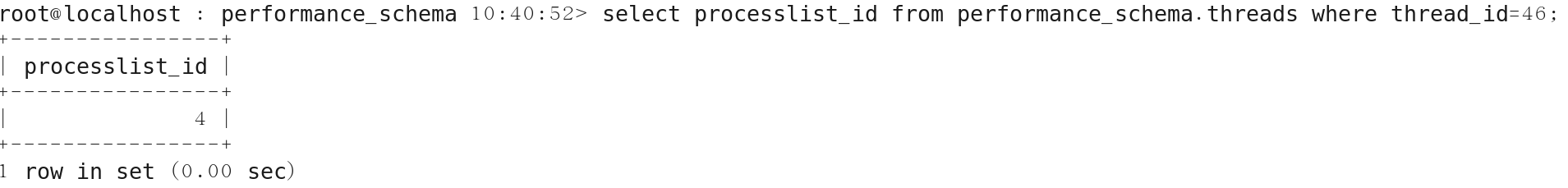
描述已自动生成



**注：上图操作中对sbtest1加锁后又查询（见DIGEST\_TEXT）**

# 如何知道内部ID为113的线程对应的process id是多少呢？可以通过performance\_schema.threads表查询

mysql> select processlist\_id from performance\_schema.threads where thread\_id=113;



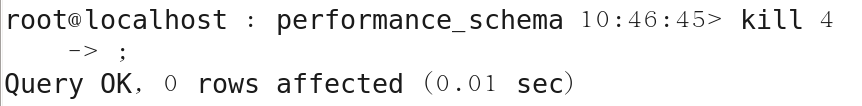


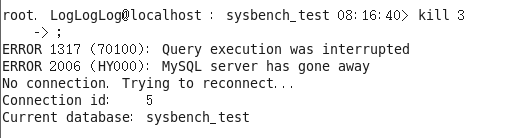
**注：这样就知道了内部ID为46的线程对应的process id=4**

# 执行kill

mysql> kill 18;

Query OK, 0 rows affected (0.00 sec)





**注：这样就知道了内部ID为46的线程对应的process id=4。杀掉后回到会话2能发现阻塞已经解除。**

mysql> show processlist;

......

3 rows in set (0.00 sec)

# 返回执行update语句的会话2，语句已经执行成功

mysql> update sbtest1 set pad='xxx' where id=1;

Query OK, 0 rows affected (7 min 50.23 sec)

Rows matched: 0 Changed: 0 Warnings: 0

