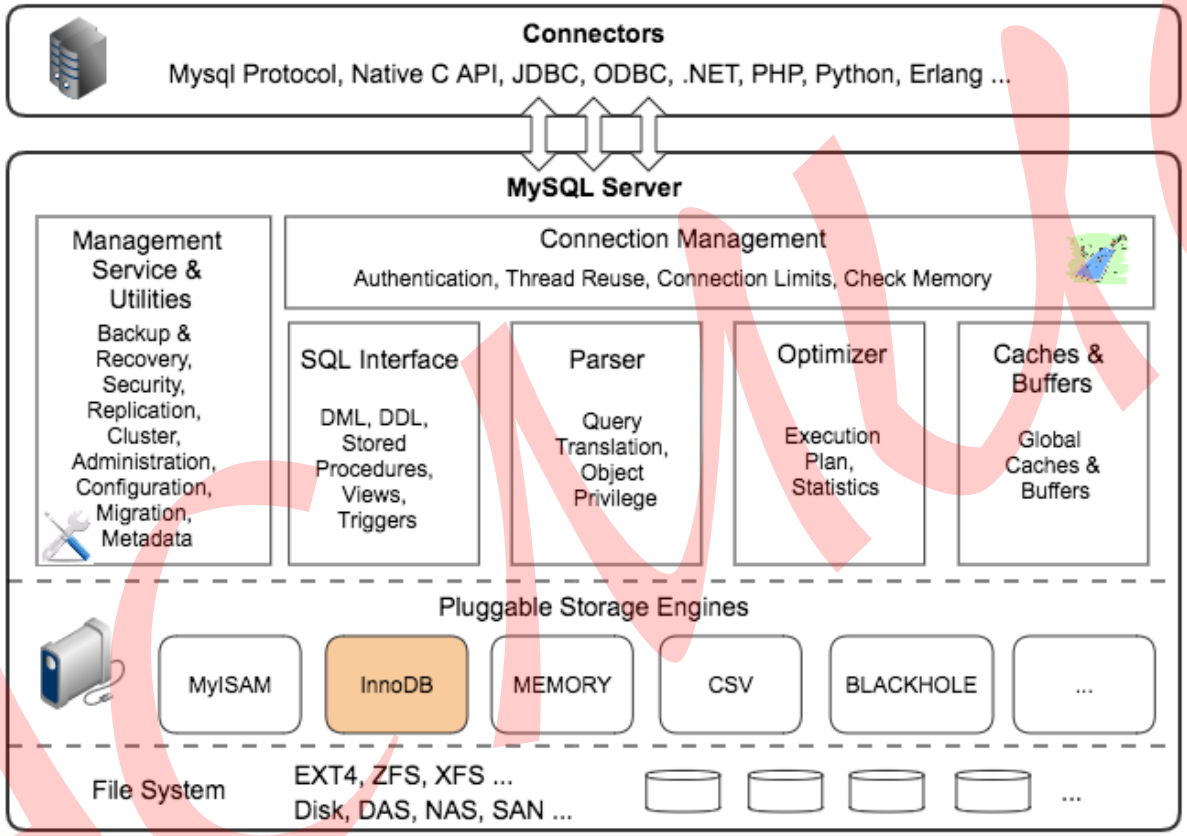
MySQL锁源码分析

MySQL的两层架构

- 服务层 VS存储引擎层
- 特色：通用化 VS专业化
- 职能：网络通讯、语法分析和对象管理 VS 数据管理


Connectors
Mysql Protocol, Native C API, JDBC, ODBC, .NET, PHP, Python, Erlang ...
MySQL Server
Management Service & Utilities
Backup & Recovery, Security, Replication, Cluster, Administration, Configuration, Migration, Metadata
Connection Management
Authentication, Thread Reuse, Connection Limits, Check Memory
SQL Interface
DML, DDL, Stored Procedures, Views, Triggers
Parser
Query Translation, Object Privilege
Optimizer
Execution Plan, Statistics
Caches & Buffers
Global Caches & Buffers
Pluggable Storage Engines
MyISAM
InnoDB
MEMORY
CSV
BLACKHOLE
...
File System
EXT4, ZFS, XFS ...
Disk, DAS, NAS, SAN ...
...

元数据锁和InnoDB锁

- 元数据锁
  - 服务层
  - 数据库对象锁
- InnoDB锁
  - 存储引擎层
  - 表锁/行锁

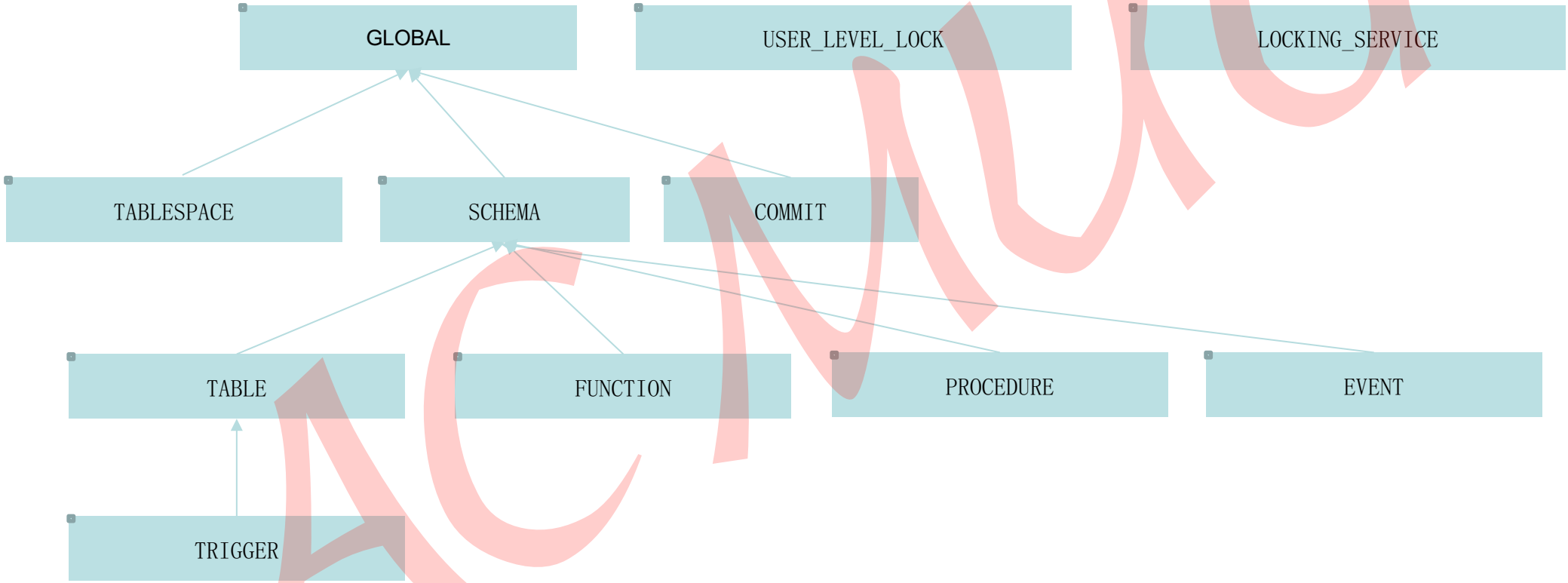
元数据锁

- 类型
- 关系
- 申请与释放
- 源码
  - 源文件
  - 函数

元数据锁-类型

- GLOBAL
- TABLESPACE
- SCHEMA
- TABLE
- FUNCTION
- PROCEDURE
- TRIGGER
- COMMIT
- USER_LEVEL_LOCK
- LOCKING_SERVICE

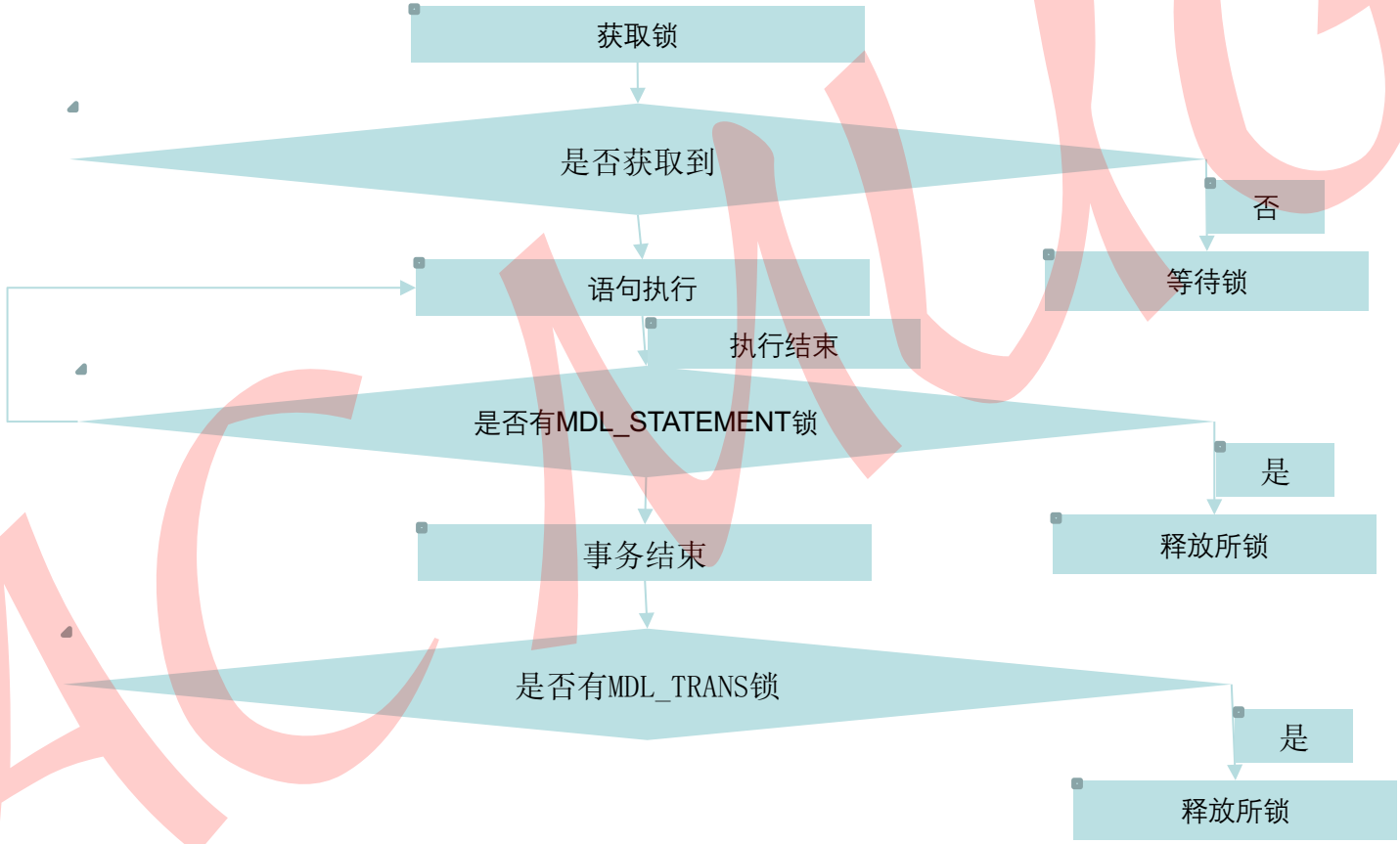
元数据锁-关系


GLOBAL
USER_LEVEL_LOCK
LOCKING_SERVICE
TABLESPACE
SCHEMA
COMMIT
TABLE
FUNCTION
PROCEDURE
EVENT
TRIGGER

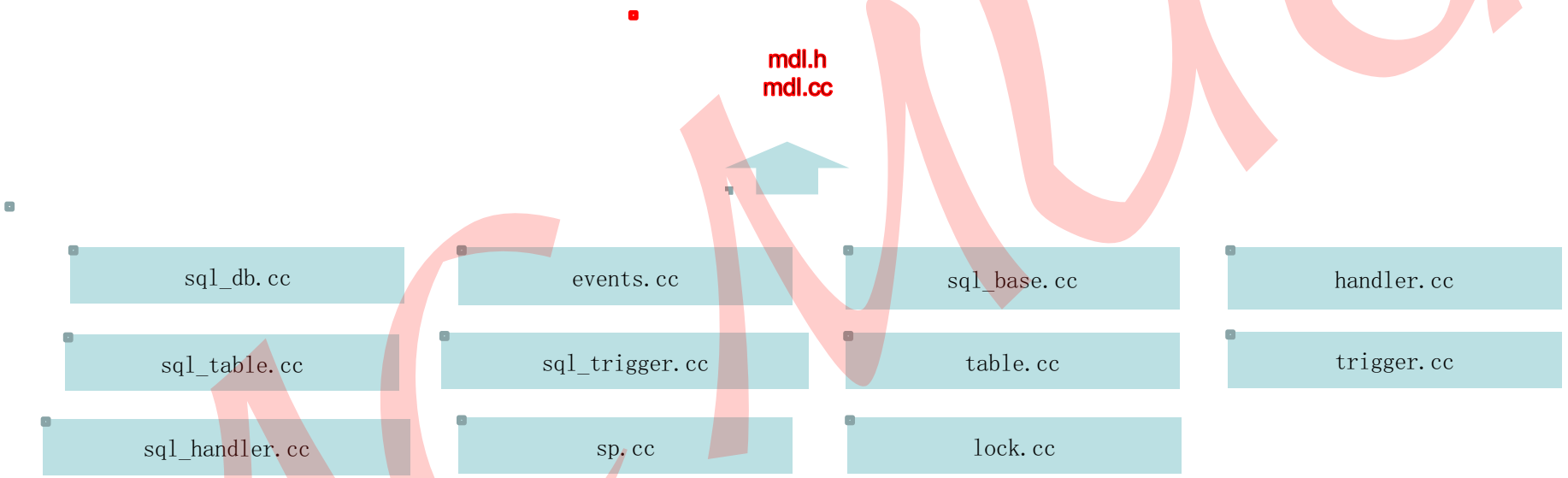
元数据锁-申请和释放1

- 释放类型
  - MDL_STATEMENT
  - MDL_TRANSACTION
  - MDL_EXPLICIT

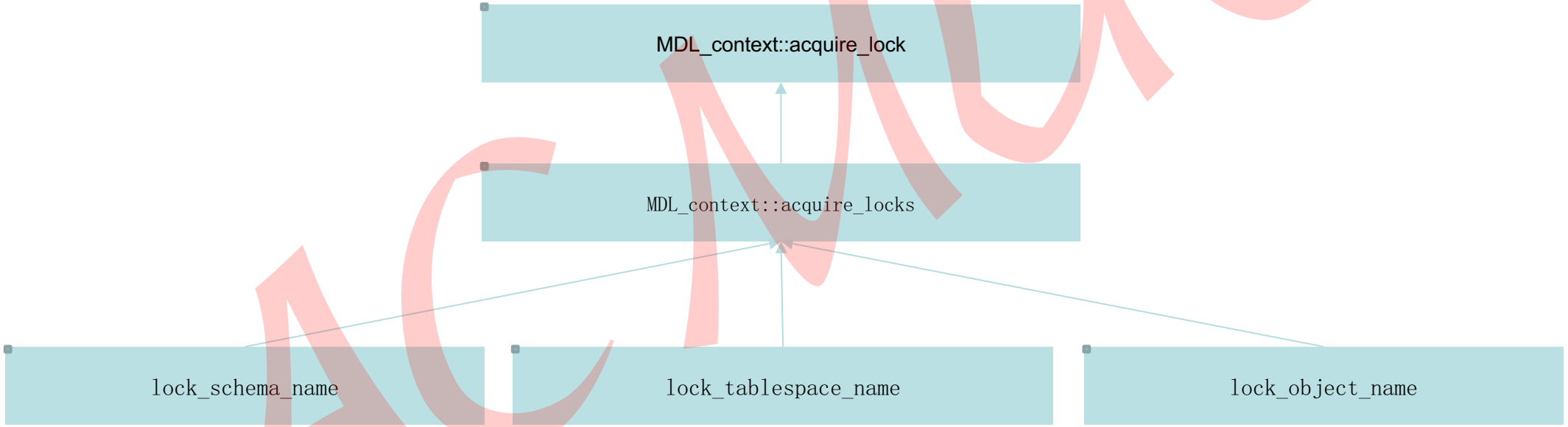
元数据锁-申请和释放2


获取锁
是否获取到
否
等待锁
语句执行
执行结束
是否有MDL_STATEMENT锁
是
释放所锁
事务结束
是否有MDL_TRANS锁
是
释放所锁

元数据锁-源文件

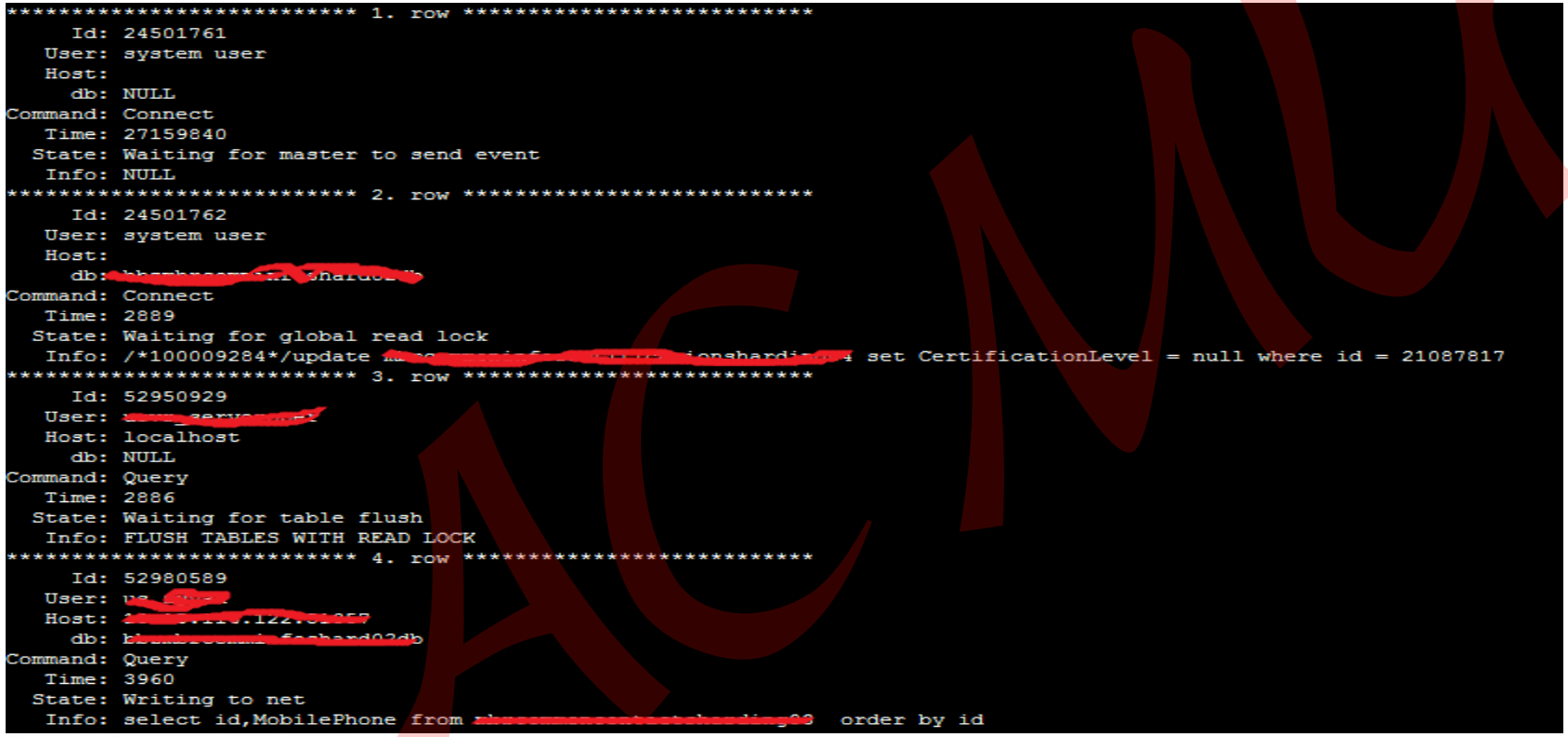


元数据锁-主要函数



案例1-现象

- 从机备份过程中有160左右的连接被阻塞，情况如下



案例1-分析

- 该连接操作耗时最长，断定连接积累的触发

```
*************************** 4. row ***************************
     Id: 52980589
   User: [redacted]
   Host: [redacted]
     db: [redacted]
Command: Query
   Time: 3960
  State: Writing to net
   Info: select id,MobilePhone from [redacted] order by id
```

- 后续的连接所处状态为“Waiting for global read lock”

```
*************************** 89. row ***************************
     Id: 52982814
   User: [redacted]
   Host: localhost
     db: [redacted]
Command: Query
   Time: 1286
  State: Waiting for global read lock
   Info: insert into processList(ID,USER,HOST,DB,COMMAND,TIME,STATE,INFO)
and != 'Sleep'  and USER not in ('usvr_replication','us_pf_zhu'))
*************************** 90. row ***************************
```

- 判断为元数据锁出现等待

案例1-调试

- 复制环境，设断点于函数”MDL_context::acquire_lock”
- 执行flush tables with read lock，定位到对全局锁的锁定
- 其堆栈电梯如下

```
#0  MDL_context::acquire_lock (this=0x7f1ae4000c08,
    mdl_request=0x7f1b2d74ab90, lock_wait_timeout=31536000)
    at /home/jiangyx/ops/mysql/mysql-5.7.18/sql/mdl.cc:3562
#1  0x000000000173633a in Global_read_lock::lock_global_read_lock (
    this=0x7f1ae4002790, thd=0x7f1ae4000b70)
    at /home/jiangyx/ops/mysql/mysql-5.7.18/sql/lock.cc:1115
#2  0x00000000015b4a93 in reload_acl_and_cache (thd=0x7f1ae4000b70,
    options=16388, tables=0x0, write_to_binlog=0x7f1b2d74badc)
    at /home/jiangyx/ops/mysql/mysql-5.7.18/sql/sql_reload.cc:221
```

案例1-代码分析

- 跟踪堆栈到发起锁的代码

```
    if (thd->global_read_lock.lock_global_read_lock(thd))
  return 1;                                 // Killed
    if (close_cached_tables(thd, tables,
                            ((options & REFRESH_FAST) ?  FALSE : TRUE),
                            thd->variables.lock_wait_timeout))
    {
      /*
        NOTE: my_error() has been already called by reopen_tables() withi
        close_cached_tables().
      */
      result= 1;
    }

    if (thd->global_read_lock.make_global_read_lock_block_commit(thd)) //
led
    {
      /* Don't leave things in a half-locked state */
      thd->global_read_lock.unlock_global_read_lock(thd);
      return 1;
    }
```

- 在上全局锁后，MySQL试图关闭打开的表，但由于如下操作，导致无法关闭，因此flush及其他操作被阻塞

```
*************************** 4. row ***************************
     Id: 52980589
   User: [redacted]
   Host: [redacted]
     db: [redacted]
Command: Query
   Time: 3960
  State: Writing to net
```

InnoDB锁

- 事务隔离级
- 锁类型
- 锁级别
- 间隙锁
- 源码简介
- 源码改造

事务隔离级1

- 脏读
- 不可重复读
- 幻读

事务隔离级2

事务隔离级\现象	脏读	不可重复读	幻读
读未提交	O	O	O
读提交	X	O	O
可重复读	X	X	O
串行化	X	X	X

InnoDB锁类型

- 表锁
- 行锁
- 间隙锁

InnoDB锁级别

- 意向共享锁（IS）
- 意向排它锁（IX）
- 共享锁（S）
- 排它锁（X）
- 自增锁（AI）

InnoDB锁兼容性

	IS	IX	S	X	AI
IS	O	O	O	X	O
IX	O	O	X	X	O
S	O	X	O	X	X
X	X	X	X	X	X
AI	O	O	X	X	O

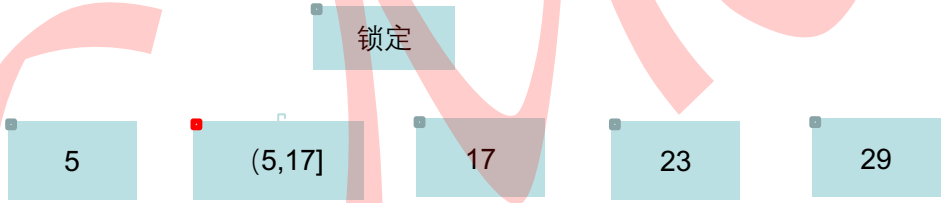
```
static const byte lock_compatibility_matrix[5][5] = {
 /**         IS     IX      S      X      AI */
 /* IS */ {  TRUE,  TRUE,  TRUE,  FALSE,  TRUE},
 /* IX */ {  TRUE,  TRUE,  FALSE, FALSE,  TRUE},
 /* S  */ {  TRUE,  FALSE, TRUE,  FALSE,  FALSE},
 /* X  */ {  FALSE, FALSE, FALSE, FALSE,  FALSE},
 /* AI */ {  TRUE,  TRUE,  FALSE, FALSE,  FALSE}
};
```

InnoDB间隙锁1

- create table t_lock(f1 int auto_increment, f2 int ,PRIMARY KEY (`f1`), key idx_f2(f2));
- insert into t_lock(f2) values (5);
- insert into t_lock(f2) values (17);
- insert into t_lock(f2) values (23);
- insert into t_lock(f2) values (29);

InnoDB间隙锁2

- begin;select * from t_lock where f2=16 for update
- 左开右闭的锁定区间



InnoDB间隙锁3

- RR隔离级下间隙锁的主要信息

```
(gdb) p lock[0]
$44 = {
  trx = 0x7ff3ed0ae8c0,
  trx_locks = {
    prev = 0x0,
    next = 0x0
  },
  index = 0x7ff398025f40,
  hash = 0x0,
  un_member = {
    tab_lock = {
      table = 0x400000037,
      locks = {
        prev = 0x48,
        next = 0x0
      }
    },
    rec_lock = {
      space = 55,
      page_no = 4,
      n_bits = 72
    }
  },
  type_mode = 547
}
(gdb) p lock->index ->name
$45 = {
  m_name = 0x7ff398026240 "idx_f2"
}
```

源码

核心代码

lock0lock.h/cc

lock0priv.h

lock0types.h

lock0prdt.h

应用代码

trx0trx.cc

trx0sys.cc

trx0rec.cc

死锁源码改造

- 为什么改进
- 改进的结果

原始输出

```
------------------------
LATEST DETECTED DEADLOCK
------------------------
2018-03-06 18:48:08 0x7f6d4d09c700
*** (1) TRANSACTION:
TRANSACTION 11536, ACTIVE 23 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 4 lock struct(s), heap size 1160, 3 row lock(s)
MySQL thread id 3, OS thread handle 140107420915456, query id 26 localhost root
updating
update t1 set f2='xxx' where f1=10
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 53 page no 4 n bits 72 index f1 of table `del_tdb1`.`t1` t
rx id 11536 lock_mode X locks rec but not gap waiting
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 000000000302; asc       ;;

*** (2) TRANSACTION:
TRANSACTION 11537, ACTIVE 16 sec starting index read
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1160, 3 row lock(s)
MySQL thread id 4, OS thread handle 140107420649216, query id 27 localhost root
updating
update t1 set f2='xxx' where f1=5
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 53 page no 4 n bits 72 index f1 of table `del_tdb1`.`t1` t
rx id 11537 lock_mode X locks rec but not gap
Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 000000000302; asc       ;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 53 page no 4 n bits 72 index f1 of table `del_tdb1`.`t1` t
rx id 11537 lock_mode X locks rec but not gap waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000005; asc     ;;
 1: len 6; hex 000000000301; asc       ;;

*** WE ROLL BACK TRANSACTION (2)
```

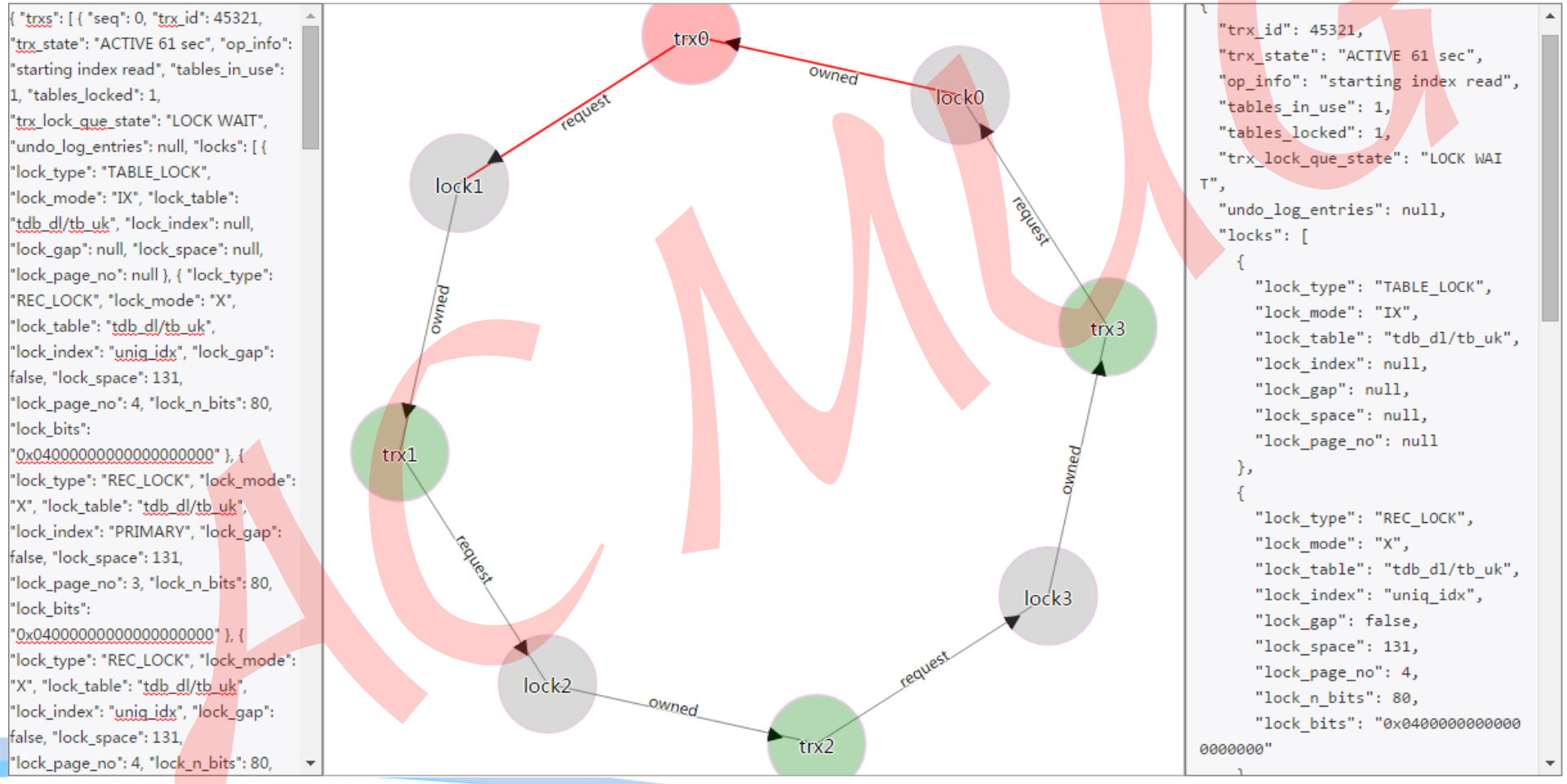
改造工作

- 格式化输出
- 可视化展示

格式化输出—json

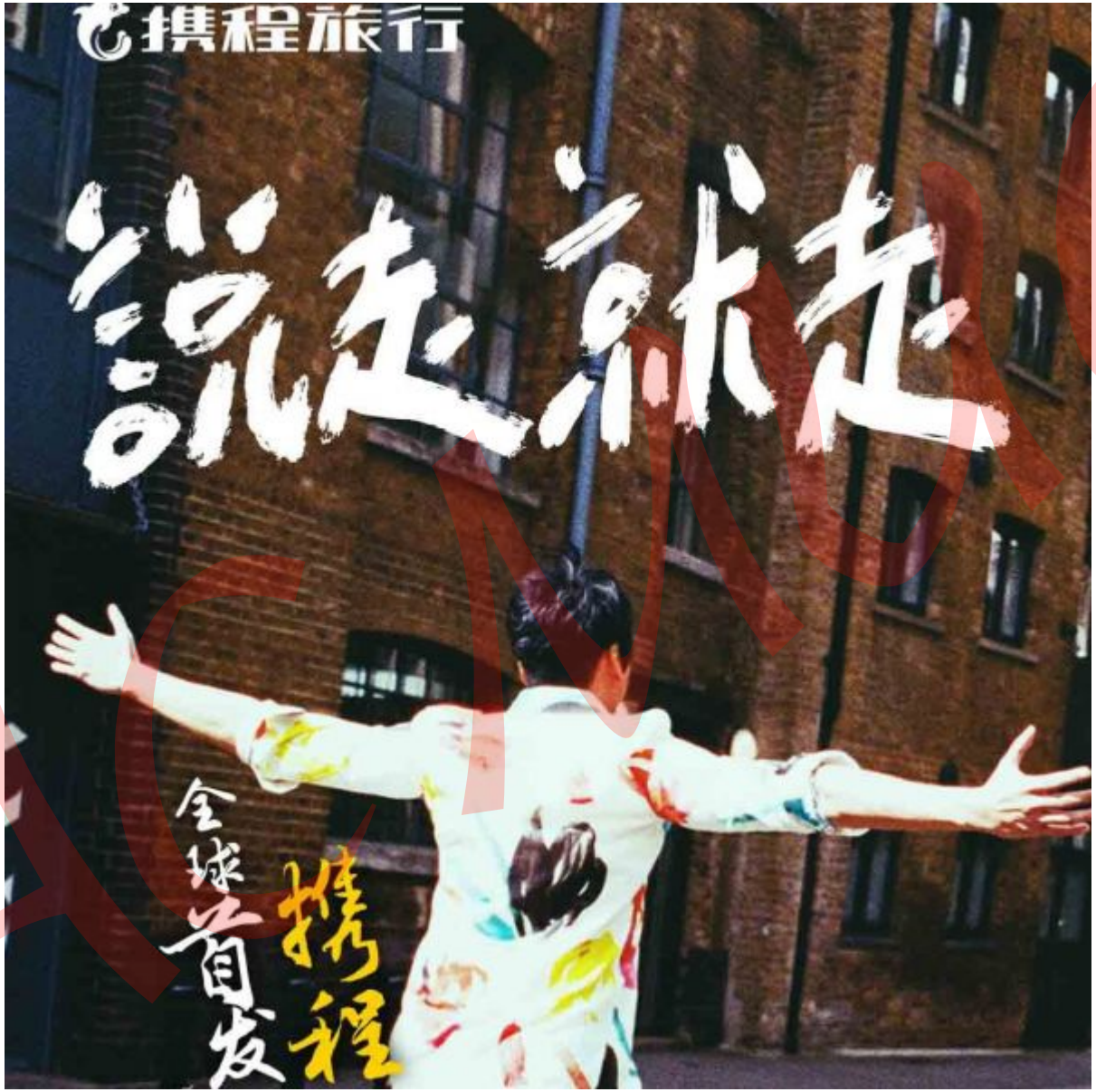

```
"trxs": [
    {
        "seq": 0,
        "trx_id": 45321,
        "trx_state": "ACTIVE 61 sec",
        "op_info": "starting index read",
        "tables_in_use": 1,
        "tables_locked": 1,
        "trx_lock_que_state": "LOCK WAIT",
        "undo_log_entries": null,
        "locks": [
            {
                "lock_type": "TABLE_LOCK",
                "lock_mode": "IX",
                "lock_table": "tdb_dl/tb_uk",
                "lock_index": null,
                "lock_gap": null,
                "lock_space": null,
                "lock_page_no": null
            },
            {
                "lock_type": "REC_LOCK",
```

改进后可视化输出


{ "trxs": [{ "seq": 0, "trx_id": 45321,
"trx_state": "ACTIVE 61 sec", "op_info":
"starting index read", "tables_in_use":
1, "tables_locked": 1,
"trx_lock_que_state": "LOCK WAIT",
"undo_log_entries": null, "locks": [{
"lock_type": "TABLE_LOCK",
"lock_mode": "IX", "lock_table":
"tdb_dl/tb_uk", "lock_index": null,
"lock_gap": null, "lock_space": null,
"lock_page_no": null }, { "lock_type":
"REC_LOCK", "lock_mode": "X",
"lock_table": "tdb_dl/tb_uk",
"lock_index": "uniq_idx", "lock_gap":
false, "lock_space": 131,
"lock_page_no": 4, "lock_n_bits": 80,
"lock_bits":
"0x04000000000000000000" }, {
"lock_type": "REC_LOCK", "lock_mode":
"X", "lock_table": "tdb_dl/tb_uk",
"lock_index": "PRIMARY", "lock_gap":
false, "lock_space": 131,
"lock_page_no": 3, "lock_n_bits": 80,
"lock_bits":
"0x04000000000000000000" }, {
"lock_type": "REC_LOCK", "lock_mode":
"X", "lock_table": "tdb_dl/tb_uk",
"lock_index": "uniq_idx", "lock_gap":
false, "lock_space": 131,
"lock_page_no": 4, "lock_n_bits": 80,
trx0
owned
lock0
request
lock1
request
owned
trx3
trx1
request
owned
lock3
lock2
owned
request
trx2
{
"trx_id": 45321,
"trx_state": "ACTIVE 61 sec",
"op_info": "starting index read",
"tables_in_use": 1,
"tables_locked": 1,
"trx_lock_que_state": "LOCK WAI
T",
"undo_log_entries": null,
"locks": [
{
"lock_type": "TABLE_LOCK",
"lock_mode": "IX",
"lock_table": "tdb_dl/tb_uk",
"lock_index": null,
"lock_gap": null,
"lock_space": null,
"lock_page_no": null
},
{
"lock_type": "REC_LOCK",
"lock_mode": "X",
"lock_table": "tdb_dl/tb_uk",
"lock_index": "uniq_idx",
"lock_gap": false,
"lock_space": 131,
"lock_page_no": 4,
"lock_n_bits": 80,
"lock_bits": "0x0400000000000
0000000"

如何学习MySQL源码

- 由功能学习到源码学习
- 观摩他人的修改
- 由源码到功能的学习


携程旅行
说走就走
全球首发
携程