注：前面两次实验分析了I/O瓶颈和锁问题之后，本次实验主要关注跟实际业务有关的性能问题。因此，我们需要学习了解如何从performance\_schema中了解SQL语句以及事务的执行情况，并从中分析影响性能的具体原因。

**全局性提示：必须根据自己的实际结果对笔记进行重新整理。应结合所分析问题制造一些情境配合分析，如错误的SQL语句，长时间被阻塞的更新操作等。**

注意：6.3和6.4节实际是接着前面6.1和6.2节的环境继续做实验，因此实验结果自然反映了前面执行语句的情况。因此setup\_consumer和setup\_instrument两个表的设置也是延续之前的，如果没有声明就是按前面的设置(重启过mysql的要特别注意这一点)，而在后面一些实验中会额外声明增加的设置。

**6.3 查看最近的SQL语句执行信息**

**6.3.1　查看最近的TOP SQL语句**

**回顾：**

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**列出与当前分析有关的表：**

图形用户界面, 文本

描述已自动生成

#当前事件记录表和**语句事件**历史记录表可以查询数据库中最近执行的一些SQL语句，以及与语句相关的信息。

#以events\_**statements**\_history表为例，查询结果**按照语句完成时间倒序**排序列。

mysql> select THREAD\_ID, EVENT\_NAME, SOURCE,sys.format\_time(TIMER\_WAIT), sys.format\_time(LOCK\_TIME), SQL\_TEXT, CURRENT\_SCHEMA, MESSAGE\_TEXT, ROWS\_AFFECTED, ROWS\_SENT, ROWS\_EXAMINED **from events\_statements\_history** where CURRENT\_SCHEMA!='performance\_schema' **order by TIMER\_WAIT** desc limit 10\G

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

注：根据语句事件历史记录表查询最近执行的一些SQL语句。要看到效果就要创设一些问题情景。上面结果延续了6.2.3的执行结果（update语句由于表被锁而长时间阻塞）。结果按照语句完成时间倒序排序列。

#按照通常优化慢SQL语句的原则，**优先优化执行次数最多的语句**，然后是执行时间最长的语句。**以上查询结果并不是通常所说的TOP SQL语句**，我们可以使用events\_statements\_**summary\_by\_digest**表来查询经过**统计之后的TOP SQL语句**。

mysql> select SCHEMA\_NAME,DIGEST\_TEXT,**COUNT\_STAR**, **sys.format\_time**(SUM\_TIMER**\_WAIT**) as sum\_time, sys.format\_time(MIN\_TIMER\_WAIT) as min\_time,sys.format\_time(AVG\_TIMER\_WAIT) as avg\_time, sys.format\_time(MAX\_TIMER\_WAIT) as max\_time,sys.format\_time(SUM\_LOCK\_TIME) as sum\_lock\_time, SUM\_ROWS\_AFFECTED,SUM\_ROWS\_SENT,SUM\_ROWS\_EXAMINED **from events\_statements\_summary\_by\_digest** where SCHEMA\_NAME is not null **order by COUNT\_STAR** desc limit 10\G

文本

描述已自动生成

**注：这里为了突出效果，重复多次执行了以上查询语句。**

**6.3.2　查看最近执行失败的SQL语句**

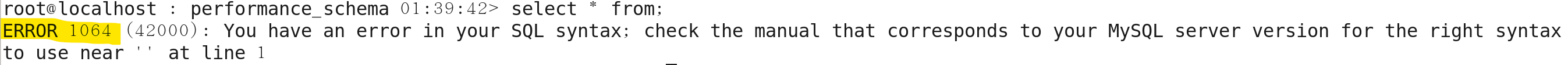
**注：该问题的背景是，有些程序执行过程的错误无法马上得知，需要反查SQL语句执行失败的记录。做实验时需要“制造错误”。**

#对于SQL语句的语法错误，**错误日志并不会记录**。

在performance\_schema的**语句事件记录表**中针对每一条语句的执行状态都记录了较为详细的信息(events\_statements\_表和events\_statements\_summary\_by\_digest表)

#使用events\_statements\_history\_long表或者events\_statements\_history表查询发生语法错误的SQL语句。**模拟一条语法错误的SQL语句：**

mysql> select \* from;



#查询events\_statements\_history表中**错误号为1064的记录**，开启另一个会话（会话2）

mysql> use performance\_schema

Database changed

mysql> select THREAD\_ID,EVENT\_NAME,SOURCE,sys.format\_time(TIMER\_WAIT) as exec\_time,sys. format\_time (LOCK\_TIME) as lock\_time,SQL\_TEXT, CURRENT\_SCHEMA,MESSAGE\_TEXT,ROWS\_AFFECTED, ROWS\_SENT,ROWS\_EXAMINED,MYSQL\_ERRNO **from events\_statements\_history** where MYSQL\_ERRNO =1064\G

图形用户界面, 文本, 电子邮件

描述已自动生成

#可能你不知道错误号是多少，可以查询发生**错误次数不为0**的语句记录，在里边找到MESSAGE\_TEXT字段，提示信息为语法错误的就是它了。

mysql > select THREAD\_ID, EVENT\_NAME,SOURCE,sys.format\_time(TIMER\_WAIT) as exec\_time, sys.format\_time(LOCK\_TIME) as lock\_time,SQL\_TEXT, CURRENT\_SCHEMA,MESSAGE\_TEXT,ROWS\_AFFECTED, ROWS\_SENT,ROWS\_EXAMINED,MYSQL\_ERRNO,**errors from events\_statements\_history** where errors> 0\G

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

#使用**events\_statements\_summary\_by\_digest表**查询发生语句执行**错误的SQL语句记录**

mysql> select \* ;

ERROR 1096 (HY000): No tables used

mysql> select \* from sbtest4 where id between 100 and 2000 and xx=1;

ERROR 1054 (42S22): Unknown column 'xx' in 'where clause'

mysql> select SCHEMA\_NAME, DIGEST\_TEXT,COUNT\_STAR, sys.format\_time(AVG\_TIMER\_WAIT) as avg\_time,sys.format\_time(MAX\_TIMER\_WAIT) as max\_time,sys.format\_time(SUM\_LOCK\_TIME) as sum\_lock\_time,SUM\_ERRORS,FIRST\_SEEN,LAST\_SEEN **from events\_statements\_summary\_by\_digest** where SUM\_ERRORS!=0\G

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**注：与书中不同的是，为了查看最近的错误（之前还有很多其他错误），这里按时间远近排序。错误的语句实际被执行了多次。**

#查询**具体的错误信息**（例如：具体的错误代码、具体的错误提示信息以及具体的错误SQL语句文本等）

mysql> select THREAD\_ID, EVENT\_NAME,SOURCE,sys.format\_time(TIMER\_WAIT) as exec\_time, sys.format\_time(LOCK\_TIME) as lock\_time,SQL\_TEXT, CURRENT\_SCHEMA,MESSAGE\_TEXT,ROWS\_AFFECTED, ROWS\_SENT,ROWS\_EXAMINED,MYSQL\_ERRNO **from events\_statements\_history** where MYSQL\_ERRNO!=0\G

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

**注：这里试验发现，只能返回1行最近结果，再次访问结果集为空。**

**6.4　查看SQL语句执行阶段和进度信息**

#通过具有可预估工作量的阶段事件进行记录与计算，就可以得到一条语句执行的阶段信息和进度信息。

**注：默认并没有配置阶段事件，因此下面(共有三个会话)并没有出现“progress”列。**

mysql>show processlist;

表格

描述已自动生成

**6.4.1　查看SQL语句执行阶段信息**

**回顾相关事件表：**

**文本

中度可信度描述已自动生成**

**#配置启用阶段事件**

**注：注意会话的切换才能保证结果输出正确**

**会话1：**

mysql> use performance\_schema

Database changed

mysql> update **setup\_instruments** set enabled='yes',timed='yes' where **name like 'stage/%**';

Query OK, **120 rows affected** (0.00 sec)

Rows matched: **129 Changed**: 120 Warnings: 0

mysql> update **setup\_consumers** set enabled='yes' where **name like '%stage%';**

Query OK, 3 rows affected (0.00 sec)

Rows matched: 3 Changed: 3 Warnings: 0

**会话2：**

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

图形用户界面, 文本, 应用程序

描述已自动生成

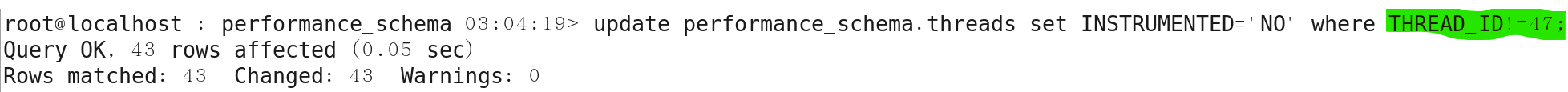
**会话1：**

# **先关闭其他线程的事件记录功能**，使用前面步骤查询到的thread\_id

mysql> update performance\_schema**.threads** set **INSTRUMENTED='NO'** where THREAD\_ID!=~~119~~;

Query OK, 101 rows affected (0.00 sec)

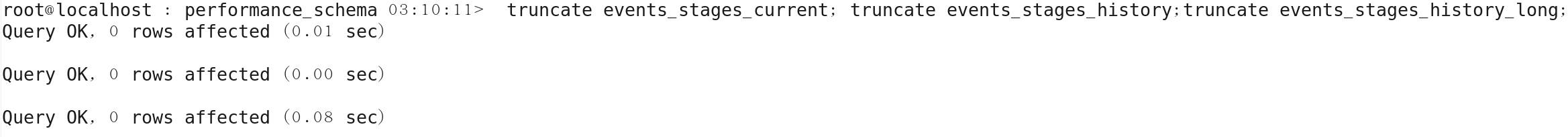
Rows matched: 101 Changed: 101 Warnings: 0



**注：不执行这个操作会因为会话1本身操作的影响而导致无法有效查看会话2的SQL语句阶段信息。**

# **清空阶段事件的三个表**，避免干扰

mysql> truncate events\_stages\_current; truncate events\_stages\_history;truncate events\_stages\_history\_long;



**注：源代码有问题，此处已是修正后执行的结果。**

**回到会话2：**执行DML语句

**注：书中所指DML语句（Data Manipulation Language），即insert、delete、udpate 和select 等。**

root@localhost : sbtest 06:06:37> select count(\*) from sbtest.sbtest4 where id between 100 and 200;

文本

中度可信度描述已自动生成

**注：这里已有多次反复试验，所以选择不同的数据表进行实验.**

**注意：下面再回到会话1执行如下语句，所得结果反映了执行会话2(线程号47)中SQL查询的过程(EVENT\_NAME)及其所需时间(exec\_time)**

#可以清晰地看到一条select语句的执行全过程

mysql> select THREAD\_ID,EVENT\_NAME,SOURCE,sys.format\_time(TIMER\_WAIT) as exec\_time,WORK\_COMPLETED, WORK\_ESTIMATED **from events\_stages\_history\_long;**

表格

描述已自动生成

**注：可见到47线程执行sql语句时，一个完整阶段：**

表格

描述已自动生成

**可见到占时最长的是打开表操作(opening tables)。各阶段的含义可参考：**

<https://blog.csdn.net/diligent203/article/details/87627497>

**显然耗时最长的阶段就是启发我们优化SQL语句的依据。**

**延申案例阅读(MySQL大量线程处于Opening tables的问题分析)：**

<https://www.cnblogs.com/CtripDBA/p/10304856.html>

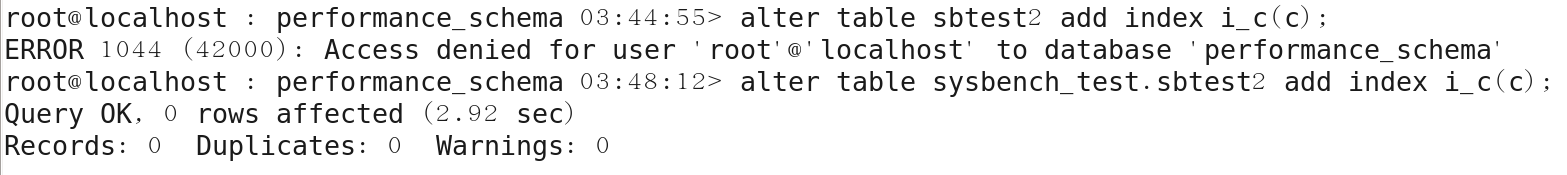
#通过以上查询数据可以清晰地看到**一条select语句的执行全过程**，以及每一个过程的时间开销等信息，那DDL语句的执行阶段又是怎样的呢？先对之前旧的信息进行清理，避免干扰。

mysql> truncate events\_stages\_current;truncate events\_stages\_history;truncate events\_stages\_history\_long;

**会话2**：

**注：书中所指DDL语句（Data Definition Languages）就是数据定义语句，即create、drop、alter等。**

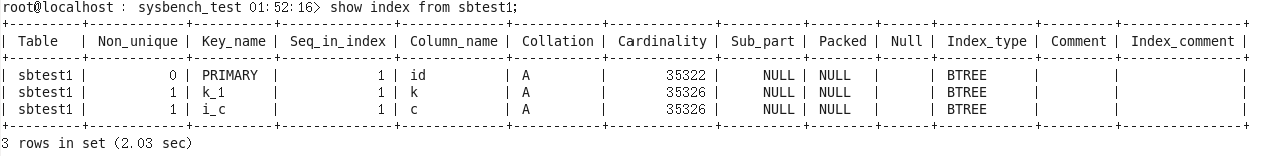
mysql> alter table sbtest2 add index i\_c(c);



**注：如果要重复实验，就要删除上面这个索引再来，或者新建一个**



**查询表的索引方法如下：**



**补充：sbtest表的定义**

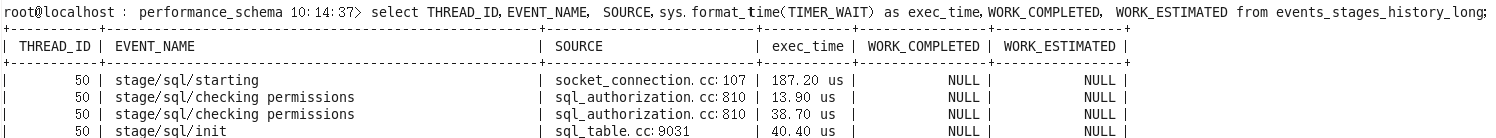
表格

中度可信度描述已自动生成

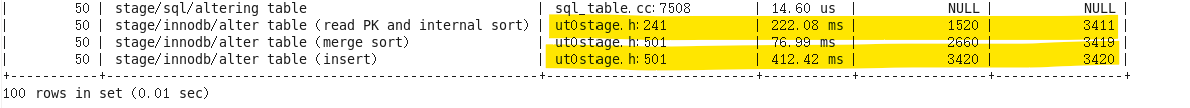
**会话1**中查询阶段事件信息，（此时DDL语句并**未执行完成(注：大概有一两秒时间，要非常抓紧如下实验)**，从**最后一行**记录信息中可以看到，**WORK\_COMPLETED和WORK\_ESTIMATED字段值不为NULL**，**表示该阶段事件是一个可以度量的事件**）。

mysql> select THREAD\_ID,EVENT\_NAME, SOURCE,sys.format\_time(TIMER\_WAIT) as exec\_time,WORK\_COMPLETED, WORK\_ESTIMATED **from events\_stages\_history\_long;**

**注:这是一开始的(另一次实验得到的信息，所以线程ID有所不同)，**



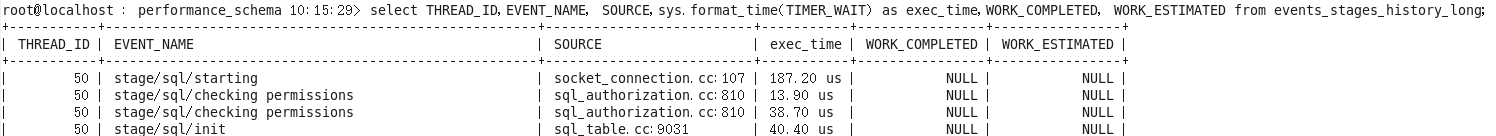
**然后是最后的：**



#等到DDL语句执行完成之后，我们**再次查看**阶段事件信息（会话1）。

mysql> select THREAD\_ID,EVENT\_NAME,SOURCE,sys.format\_time(TIMER\_WAIT) as exec\_time,WORK\_COMPLETED, WORK\_ESTIMATED **from events\_stages\_history\_long;**

**注：这是一开始的，结果一样（表继续延长，其中内容略）**



**然后是最后的：**

**文本

中度可信度描述已自动生成**

#通过以上查询数据可以清晰地看到一条alter语句添加索引的执行全过程，以及每一个过程的时间开销等信息，**执行时间最长**的是stage/innodb/alter table（merge sort），其次是stage/innodb/altertable（read PK and internal sort），说明本示例中创建索引的主要时间开销在内部的排序合并操作和数据排序上。

**注：显然以上结果并不完全跟书中的结果一致，请自行分析自己的实验结果。**

**注：6.4.2节查看SQL语句执行进度信息留到后续章节再讨论。**

**6.5　查看最近的事务执行信息**

**回顾：与事务有关的相关事件表：**

**文本

描述已自动生成**

#借助performance\_schema的events\_transactions\_\*表来查看与事务相关的记录，在这些表中详细记录了是否有事务被回滚、活跃（长时间**未提交的事务**也属于活跃事务）或已提交等信息。

#下面分别模拟几种事务情况，并查看事务事件记录表。

**会话1**：首先需要进行配置启用，事务事件默认并未启用。

mysql> update setup\_instruments set enabled='yes',timed='yes' where name like 'transaction';

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> update setup\_consumers set enabled='yes' where name like '%transaction%';

Query OK, 3 rows affected (0.00 sec)

Rows matched: 3 Changed: 3 Warnings: 0

**会话1**：执行清理，避免其他事务干扰

mysql> truncate events\_transactions\_current; truncate events\_transactions\_history; truncate events\_transactions\_history\_long;

Query OK, 0 rows affected (0.00 sec)

......

**会话2**：开启一个新会话用于执行事务，并**模拟事务回滚**。

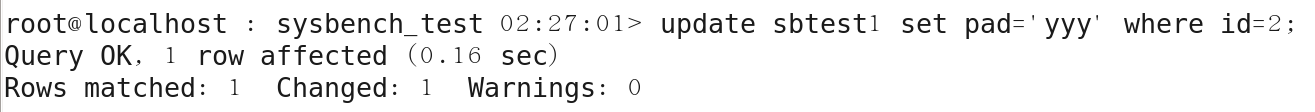
mysql> use sbtest

Database changed

mysql> begin;

Query OK, 0 rows affected (0.00 sec)

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



**会话1**：从events\_transactions\_current表中查询活跃事务

mysql> select THREAD\_ID, EVENT\_NAME,STATE,TRX\_ID, GTID,SOURCE,TIMER\_WAIT,ACCESS\_MODE, ISOLATION\_LEVEL,AUTOCOMMIT,NESTING\_EVENT\_ID,NESTING\_EVENT\_TYPE from events\_transactions\_**current**\G

文本

描述已自动生成

**会话2**：回滚事务，被回滚完成的事务不再活跃。

mysql> rollback;

Query OK, 0 rows affected (0.01 sec)

**会话1**：查询事务事件历史记录表events\_transactions**\_history\_long**

mysql> select THREAD\_ID, EVENT\_NAME,STATE,TRX\_ID, GTID,SOURCE,TIMER\_WAIT,ACCESS\_MODE, ISOLATION\_LEVEL,AUTOCOMMIT,NESTING\_EVENT\_ID,NESTING\_EVENT\_TYPE from events\_transactions\_history\_long\G

文本

描述已自动生成

**会话2**：模拟事务正常提交

mysql> begin;

Query OK, 0 rows affected (0.00 sec)

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

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> commit;

Query OK, 0 rows affected (0.01 sec)

**会话1**：看到事务已经提交成功。

mysql> select THREAD\_ID, EVENT\_NAME,STATE,TRX\_ID, GTID,SOURCE,TIMER\_WAIT,ACCESS\_MODE, ISOLATION\_LEVEL,AUTOCOMMIT,NESTING\_EVENT\_ID,NESTING\_EVENT\_TYPE from events\_transactions\_current\G

文本

描述已自动生成

#提示：如果一个事务长时间未提交（长时间处于ACTIVE状态），对于这种情况，虽然从events\_transactions\_current表中可以查询到未提交的事务事件信息，但是并不能很直观地看到**事务是什么时间点开始的**，我们可以借助information\_schema.innodb\_trx表来进行辅助判断。

**注：借助information\_schema.innodb\_trx辅助判断仍活跃(ACTIVE)事务的开始时间。注意这里需要重新利用会话2激活一个未提交的事务:**

**应用程序

低可信度描述已自动生成**

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

文本, 信件, 电子邮件

描述已自动生成