目录

[第9章　sys系统库应用示例荟萃 1](#_Toc9691)

[9.1　查看慢SQL语句慢在哪里(session视图) 1](#_Toc11653)

[9.3　查看是否有MDL锁等待(schema\_table\_lock\_waits视图) 3](#_Toc6311)

[9.4　查看InnoDB缓冲池中的热点数据有哪些(innodb\_buffer\_stats\_by\_schema视图) 5](#_Toc32079)

[9.5　查看冗余索引(schema\_redundant\_indexes视图) 6](#_Toc6108)

[9.7 查询表的增、删、改、查数据量和I/O耗时统计信息(schema\_table\_statistics\_with\_buffer视图) 6](#_Toc32411)

[9.8　查看MySQL磁盘文件产生的磁盘流量与读写比例(io\_global\_by\_file\_by\_bytes视图) 7](#_Toc14170)

[9.9　查看哪些语句使用了全表扫描(statements\_with\_full\_table\_scans视图) 8](#_Toc21911)

[9.10　查看哪些语句使用了文件排序(statements\_with\_sorting视图) 9](#_Toc13094)

[9.11　查看哪些语句使用了临时表(statements\_with\_temp\_tables视图) 10](#_Toc31895)

## 第9章　sys系统库应用示例荟萃

重要：我们可以把sys系统库理解为一个探测性能信息的综合可自定义工具箱。要理解本章各种示例的基本思想，就是假设存在某个问题(这个需要我们创设对应的情景，你可以自由创设各种情景)导致性能，为了证实提出的假设，调取与该问题密切相关的sys工具(主要是视图)，看从中获得的数据是否支持假设(我们制造了问题，当然会看到视图数据支持这种假设)。

最终，这些示例均旨在说明本书前言提出的“方法论”：

“本书的性能优化方法论还是工业革命时代的机械思维，简而言之，就是寻找因果关系，大胆假设，小心求证。”

**而求证的基本工具就是sys视图。**

本章与第六章的关系：第六章从底层出发讲解性能问题排查的基本起点(performance\_schema)，本章讲解排查性能问题一些有用的sys\_schema工具和方法。

注：关于sys系统库的更详细解释，可参考本书的网站：

<https://github.com/xiaoboluo768/mysql-system-schema/wiki/sys-schema%E8%AF%A6%E8%A7%A3>

注：由于实验环境的未知原因，有些查询使用的是x$视图。正常情况下直接查询对应的视图即可。

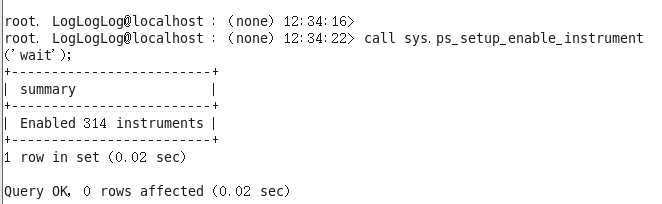
注：9.2和9.6两节不要求。

## 9.1　查看慢SQL语句慢在哪里(session视图)

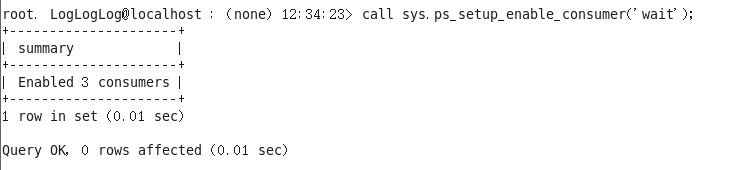
#如果我们频繁地在慢查询日志(注：关于什么是慢查询日志在9.4节补充解释)中发现某个语句执行缓慢，且在表结构、索引结构、统计信息中都无法找出原因时，则可以利用sys系统库中的撒手锏：sys.session视图(注：或者processlist视图，但session视图只显示前台有关线程)结合performance\_schema的等待事件来找出症结所在。

# 首先需要启用与等待事件相关的instruments和consumers，否则last\_wait字段值可能为NULL

mysql> call **sys.ps\_setup\_enable\_instrument**('wait');



mysql> call **sys.ps\_setup\_enable\_consumer**('wait');



注：这是系统初始时的状态，如果前面实验有改动过部分配置，结果会有所不同。

Select sleep(300);

图表, 文本

中度可信度描述已自动生成

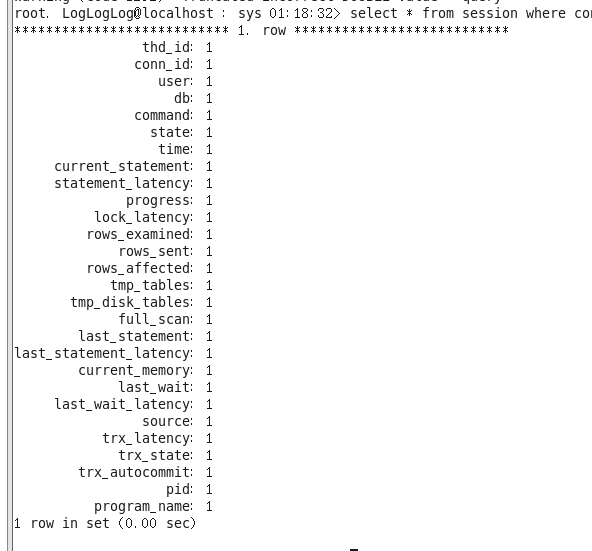
# 然后，使用**session视图**进行查询（这里只查询command为query的线程信息，代表正在执行查询）

注：这里需要补充的是，事先应通过另一个会话创建一个“**慢SQL语句**”(还有什么慢语句？)

图表, 文本

中度可信度描述已自动生成

mysql> select \* from session where command='query' and conn\_id!=connection\_id()\G



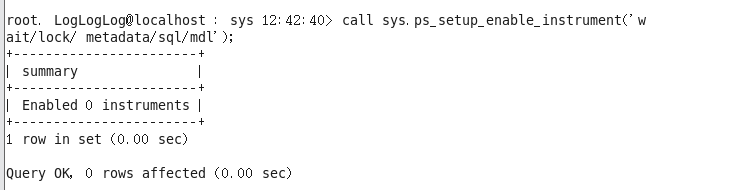
注：结果中各字段的含义见书中解释。其中last\_wait\_latency可以知道线程为最近等待事件的等待时间，但这里并没有开计时开关，所以只显示还在等待。

## 9.3　查看是否有MDL锁等待(schema\_table\_lock\_waits视图)

#通过**schema\_table\_lock\_waits视图**可以查看当前连接线程的MDL锁等待信息，显示哪些会话被MDL锁阻塞，是谁阻塞了这些会话。

# 首先需要启用与MDL锁等待事件相关的instruments

mysql> call sys.ps\_setup\_enable\_instrument('wait/lock/ metadata/sql/mdl');

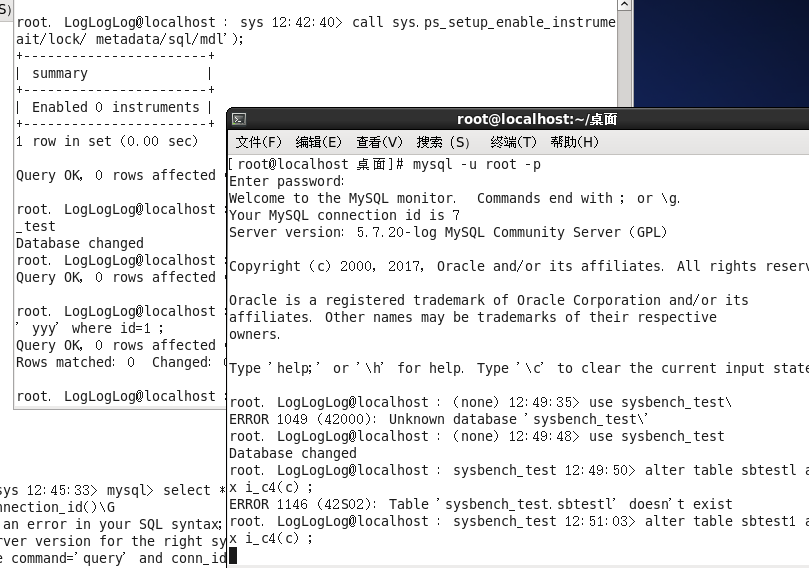


注：如果顺着前面的实验，这里已经设置好了。但是重启过服务器或机器的要注意打开开关。注意需要参考6.2.2节另起两个会话构造锁等待。

会话1：

会话2：

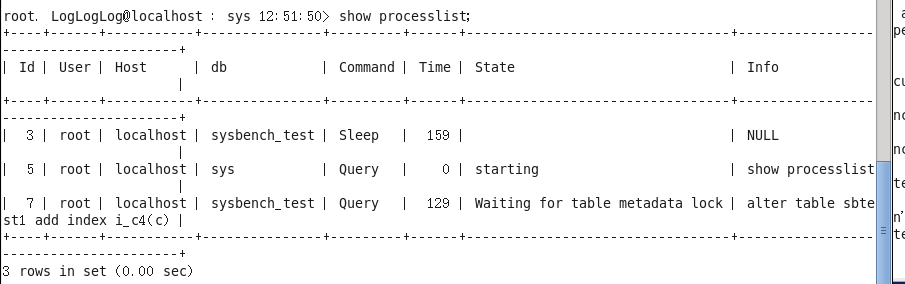
alter table sbtest1 add index i\_c4(c) ;



mysql> select \* from schema\_table\_lock\_waits\G



注：这里用x$**schema\_table\_lock\_waits**，waiting\_pid显示等待MDL锁的processlist\_id，blocking\_pid显示持有MDL锁的processlist\_id. 其余字段解释见教材。



注：以后的实践中我们当然可以直接使用schema\_table\_lock\_waits视图，但正如作者在书中许多地方反复强调的，我们必须搞清楚这些视图的来源和原理，从而真正在后面灵活地自己去排查问题。

#视图查询语句文本

SELECT g.object\_schema AS object\_schema,

g.object\_name AS object\_name,

pt.thread\_id AS waiting\_thread\_id,

pt.processlist\_id AS waiting\_pid,

sys.ps\_thread\_account(p.owner\_thread\_id) AS waiting\_account,

p.lock\_type AS waiting\_lock\_type,

p.lock\_duration AS waiting\_lock\_duration,

sys.format\_statement(pt.processlist\_info) AS waiting\_query,

pt.processlist\_time AS waiting\_query\_secs,

ps.rows\_affected AS waiting\_query\_rows\_affected,

ps.rows\_examined AS waiting\_query\_rows\_examined,

gt.thread\_id AS blocking\_thread\_id,

gt.processlist\_id AS blocking\_pid,

sys.ps\_thread\_account(g.owner\_thread\_id) AS blocking\_account,

g.lock\_type AS blocking\_lock\_type,

g.lock\_duration AS blocking\_lock\_duration,

CONCAT('KILL QUERY ', gt.processlist\_id) AS sql\_kill\_blocking\_query,

CONCAT('KILL ', gt.processlist\_id) AS sql\_kill\_blocking\_connection

FROM **performance\_schema.metadata\_locks** g

INNER JOIN **performance\_schema.metadata\_locks** p

ON g.object\_type = p.object\_type

AND g.object\_schema = p.object\_schema

AND g.object\_name = p.object\_name

AND g.lock\_status = 'GRANTED'

AND p.lock\_status = 'PENDING'

INNER JOIN **performance\_schema.threads** gt ON g.owner\_thread\_id = gt.thread\_id

INNER JOIN **performance\_schema.threads** pt ON p.owner\_thread\_id = pt.thread\_id

LEFT JOIN **performance\_schema.events\_statements\_current** gs ON g.owner\_thread\_id = gs.thread\_id

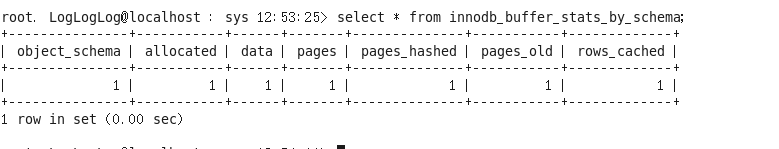
LEFT JOIN **performance\_schema.events\_statements\_current** ps ON p.owner\_thread\_id = ps.thread\_id

WHERE g.object\_type = 'TABLE';

## 9.4　查看InnoDB缓冲池中的热点数据有哪些(innodb\_buffer\_stats\_by\_schema视图)

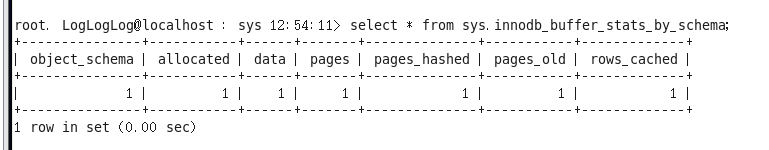
#使用**innodb\_buffer\_stats\_by\_schema视图**可以按照schema(注：数据库)分组查询InnoDB缓冲池的统计信息。

mysql> select \* from innodb\_buffer\_stats\_by\_schema;



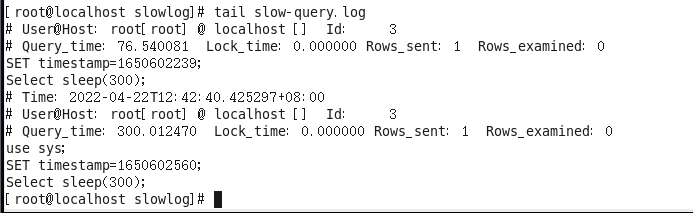
注：查询需要较多时间。为突出效果，在执行以上查询之前，执行了6.1节的压测命令，InnoDB缓冲池中有sysbench\_test库的内容,由此可以知道现在查询的“热点”，这些自然是影响性能的“大户”。

**注：既然对innodb\_buffer\_stats\_by\_schema视图的查询本身就慢，可以借此了解何为“慢查询”。执行了上一个语句后可以去查查慢查询日志（以下是执行了另一次查询）：**



**如果的确够慢（慢的定义后面再谈），可以查到慢查询日志(slowlog)的尾部最新新增的一条记录(也可以用编辑器打开查看)：**

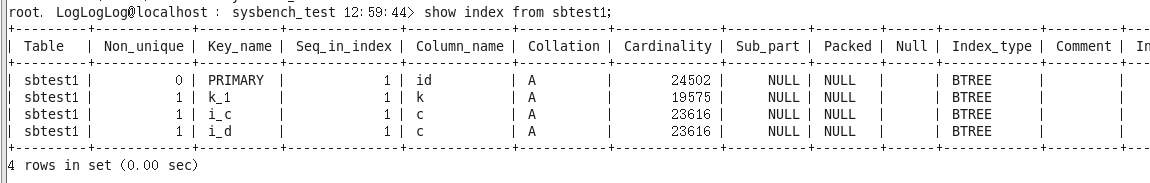
**/home/mysql/data/mysqldata1**



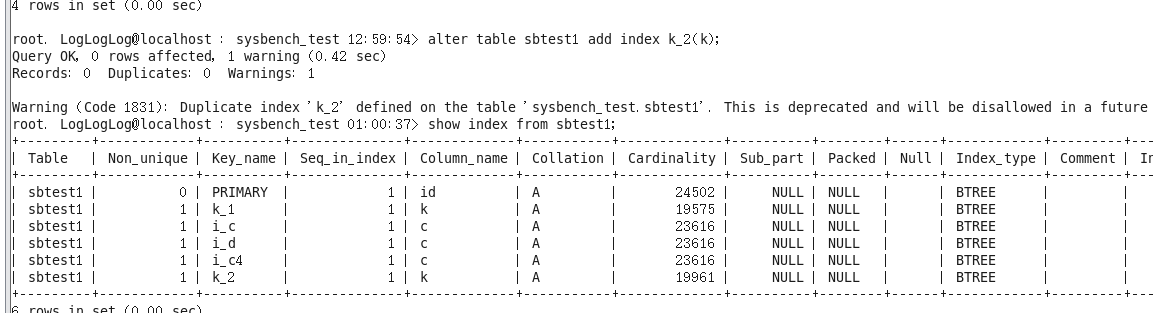
## 9.5　查看冗余索引(schema\_redundant\_indexes视图)

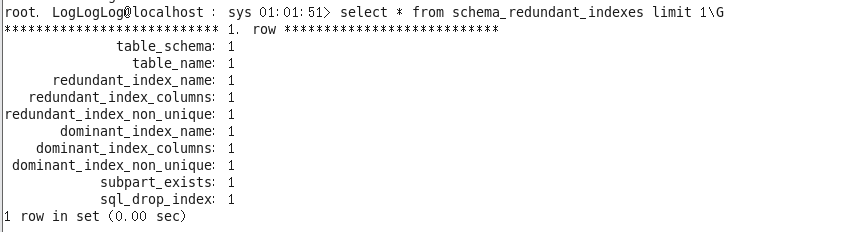
注：冗余的索引自然会影响性能。首先要“制造冗余”。先查看有什么索引：

mysql> select \* from schema\_redundant\_indexes limit 1\G



然后增加冗余的索引：

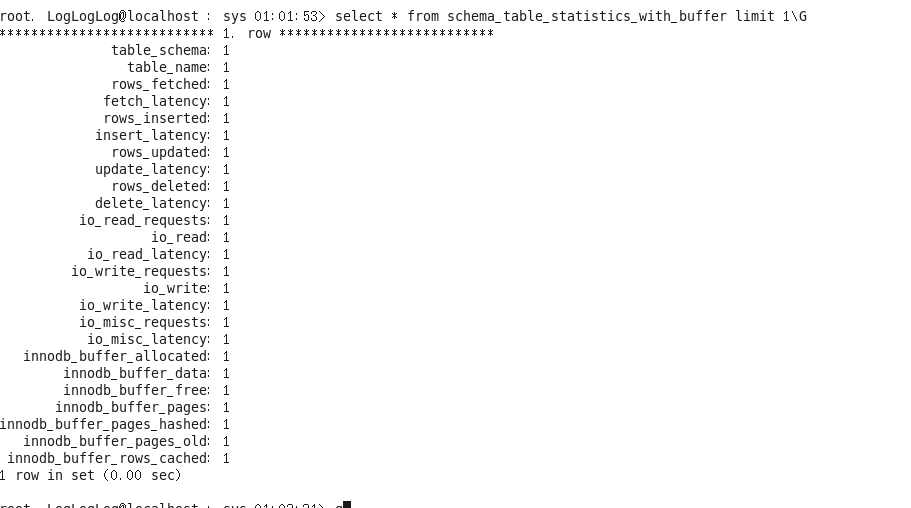


我们可以查看到对应的冗余索引信息，还会提示你怎样把它删掉：  


## 9.7 查询表的增、删、改、查数据量和I/O耗时统计信息(schema\_table\_statistics\_with\_buffer视图)

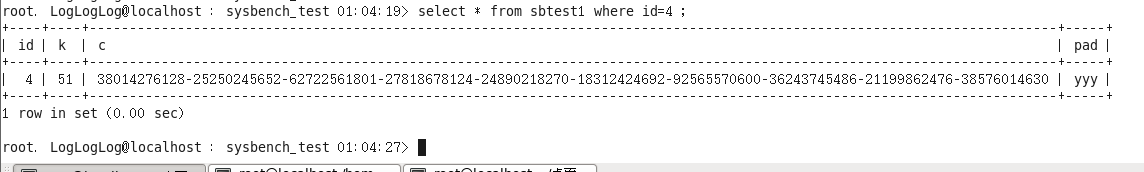
#查询表的**增、删、改、查数据量**，I/O耗时，以及在InnoDB缓冲池中占用情况等统计信息

mysql> select \* from schema\_table\_statistics\_with\_buffer limit 1\G



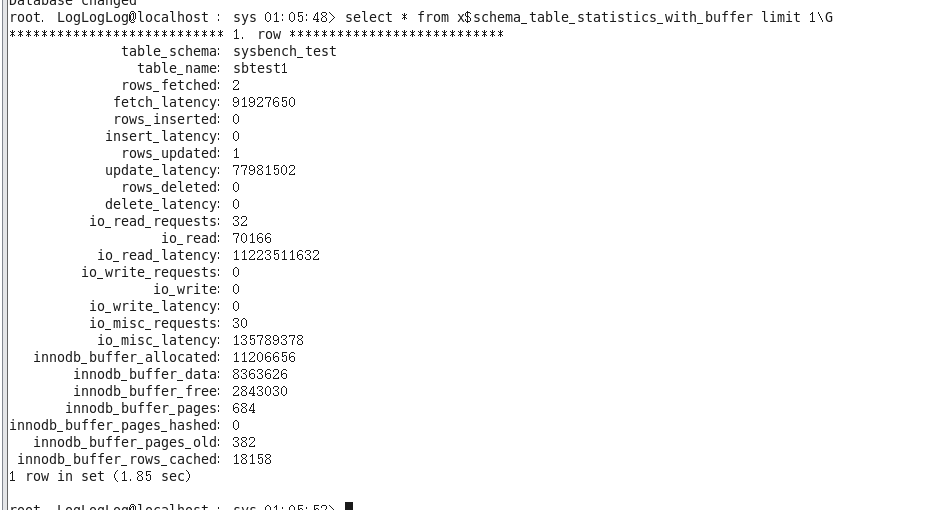
注：如果前面没有增删改数据表的内容，这里没有与io\_write有关的统计数据。因此，要给出增删操作，例如：

select \* from sbtest1 where id=4 ;



然后就能看到统计数据：

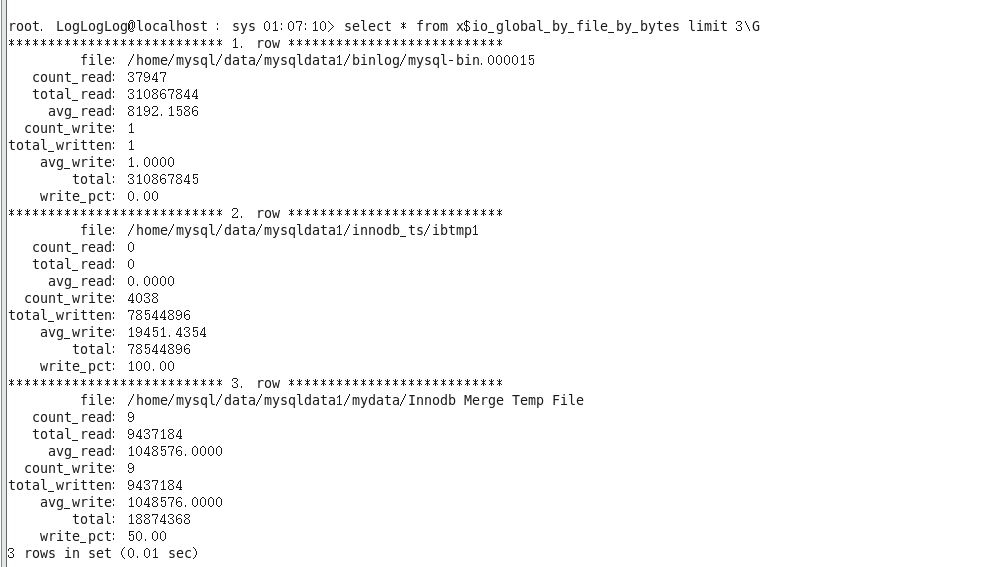
select \* from x$schema\_table\_statistics\_with\_buffer limit 1\G



## 9.8　查看MySQL磁盘文件产生的磁盘流量与读写比例(io\_global\_by\_file\_by\_bytes视图)

#可以按照文件路径+名称分组（磁盘文件名）查看全局的I/O读写字节数、读写文件I/O事件数量统计信息，在默认情况下按照总I/O读写字节数进行降序排列。

mysql> select \* from x$io\_global\_by\_file\_by\_bytes limit 3\G



注：从实验9.1到这里总共有效地更新了sbtest表4次(count\_write)。可以根据结果想想自己前面做了什么操作与这里的结果对应起来。第一项是关于临时表的I/O，可结合9.11

注：以下是另一个系统的另一个结果(只有查询)：

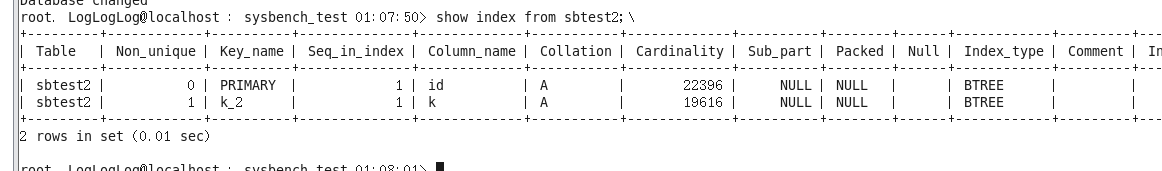
日历

描述已自动生成

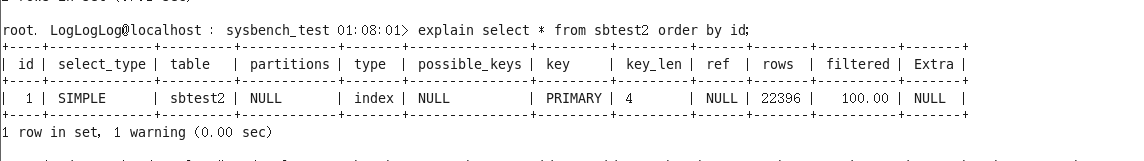
============9.9-9.11查看的语句均为可能影响性能的SQL语句=========

## 9.9　查看哪些语句使用了全表扫描(statements\_with\_full\_table\_scans视图)

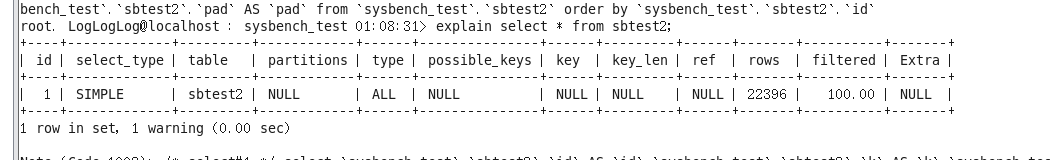
注：首先，这是sysbench\_test库中sbtest2的索引情况：



对根据主键id查询的解释：



如果type字段为ALL，则该语句要扫描全表：



注：以下语句应先执行使用全表扫描的语句，然后再查询statements\_with\_full\_table\_scans视图。

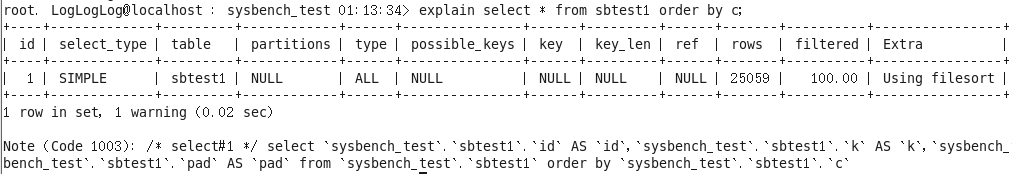
mysql> select \* from x$statements\_with\_full\_table\_scans order by last\_seen desc limit 2\G



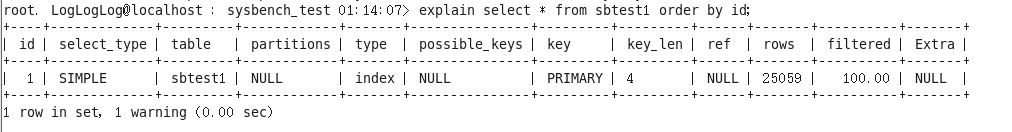
注：这里首先查询了sysbench\_test.sbtest1全表，然后按查询时间反查x$statements\_with\_full\_table\_scans。

## 9.10　查看哪些语句使用了文件排序(statements\_with\_sorting视图)

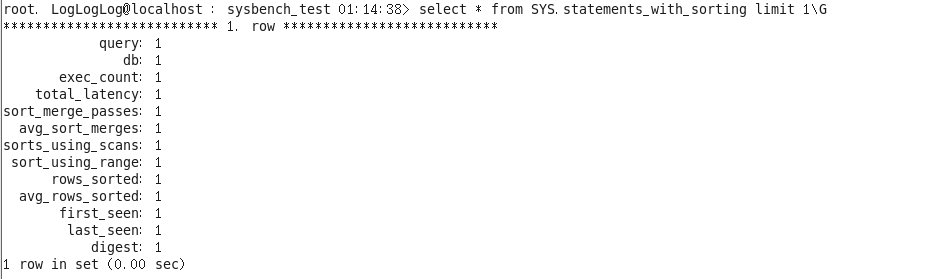
注：要先执行一些使用了文件排序的SQL语句。通过explain命令获取查询语句的执行计划(其余查询结果相同)：



可以知道以上的SQL语句使用了文件排序(具体情况需核实)，对比使用利用有序索引(sbtest1表id是索引)获取有序数据(非文件排序)：



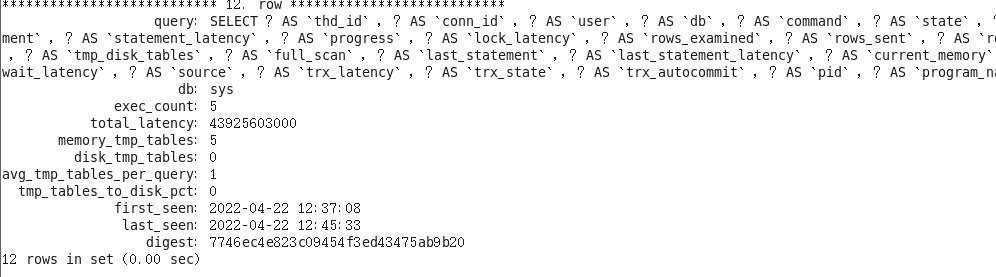
mysql> select \* from SYS.statements\_with\_sorting limit 1\G



## 9.11　查看哪些语句使用了临时表(statements\_with\_temp\_tables视图)

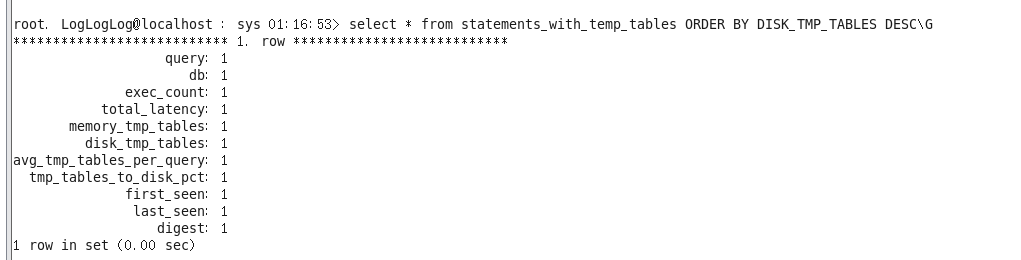
注：临时表用于存放临时数据，当关闭会话连接时会自动删除表。

mysql> select \* from x$statements\_with\_temp\_tables order by last\_seen DESC\G



注：show tables也会建立临时表，可以先执行该语句再查询statements\_with\_temp\_tables视图。注意这里反查了最新的临时表，当然也可以查谁使用了最多的临时表（意味着会对性能有影响），可以见到之前压测实验的记录：

select \* from statements\_with\_temp\_tables ORDER BY DISK\_TMP\_TABLES DESC\G



**推荐阅读案例：MySQL 临时表空间数据过多导致磁盘空间不足的问题排查，**

**https://zhuanlan.zhihu.com/p/266779699**