目录

[4.1 什么是performance\_schema 1](#_Toc22486)

[4.2 performance\_schema使用快速入门 2](#_Toc22302)

[4.2.1　检查当前数据库版本是否支持 2](#_Toc13414)

[4.2.2　启用performance\_schema 3](#_Toc6878)

[4.2.3　performance\_schema表的分类 4](#_Toc32121)

[4.2.4　performance\_schema简单配置与使用 6](#_Toc8404)

## 4.1 什么是performance\_schema

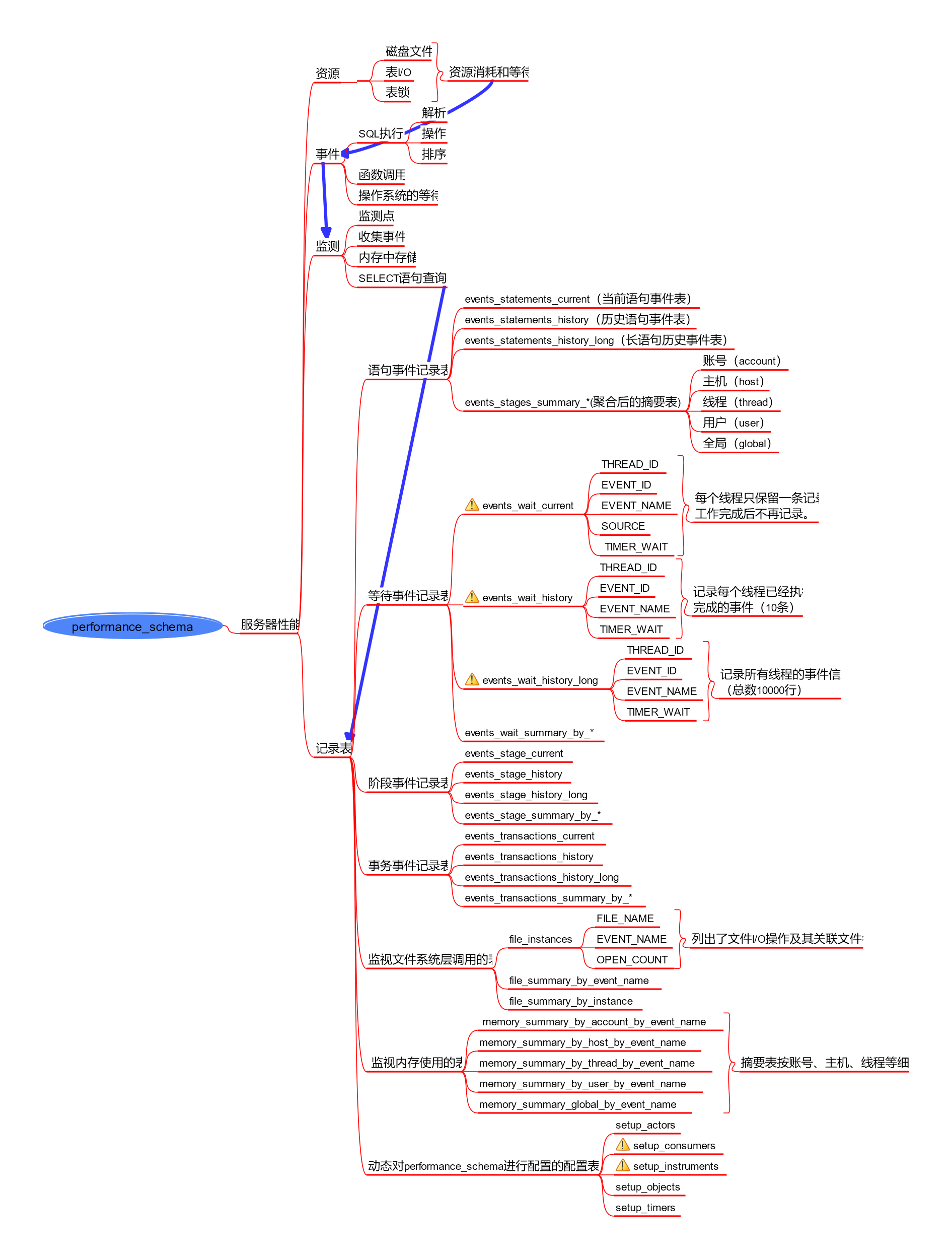
#MySQL的performance\_schema是运行在较低级别的用于监控MySQL Server运行过程中的**资源消耗、资源等待**等情况的一个功能特性。

#performance\_schema通过监视Server的事件来实现监视其内部执行情况，**“事件”**就是在Server内部活动中所做的任何事情以及对应的时间消耗，利用这些信息来判断Server中的相关资源被消耗在哪里。

#收集到的事件数据被存储在performance\_schema数据库的表中。performance\_schema的表中数据不会持久化存储在磁盘中，而是保存在内存中，一旦服务器重启，这些数据就会丢失

#与information\_schema不同，**information\_schema**主要关注Server运行过程中的元数据(注：例如字段约束条件)信息。**sys schema**是一组视图、存储过程和函数，可以方便地访问performance\_schema收集的数据，同时检索的数据可读性也更高。

注：简单地说，performance\_schema就是关于性能的数据库，而informance\_schema就是关于数据的信息(元数据)数据库。以下是performance\_schema知识点归纳：

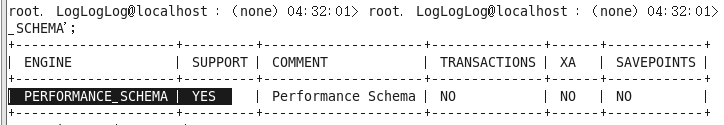
****

## 4.2 performance\_schema使用快速入门

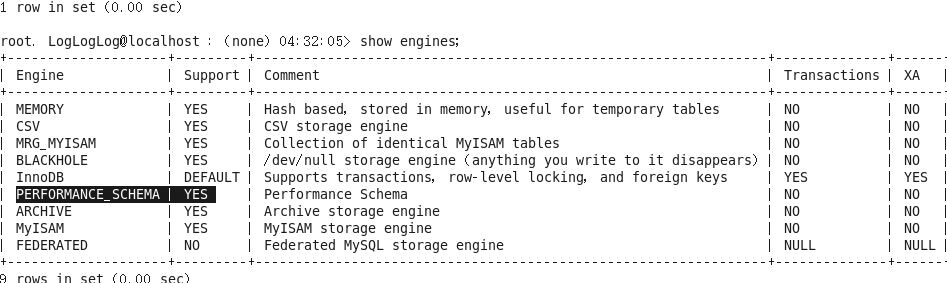
## **4.2.1　检查当前数据库版本是否支持**

#performance\_schema**被视为存储引擎**，如果该引擎可用，则应该在INFORMATION\_SCHEMA.ENGINES表或show engines语句的输出中可以看到它的Support字段值为YES

mysql> SELECT \* FROM INFORMATION\_SCHEMA.ENGINES WHERE ENGINE ='PERFORMANCE\_SCHEMA';



mysql> show engines;

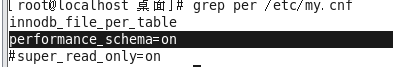


## 4.2.2　启用performance\_schema

# performance\_schema在MySQL 5.6及之前的版本中默认没有启用。如果要显式启用或关闭performance\_schema，则**需要使用参数performance\_schema=ON|OFF来设置**，并在my.cnf中进行配置

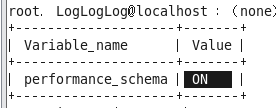
[mysqld]

performance\_schema = ON # 注意：该参数为只读参数，需要**在实例启动之前**设置才生效



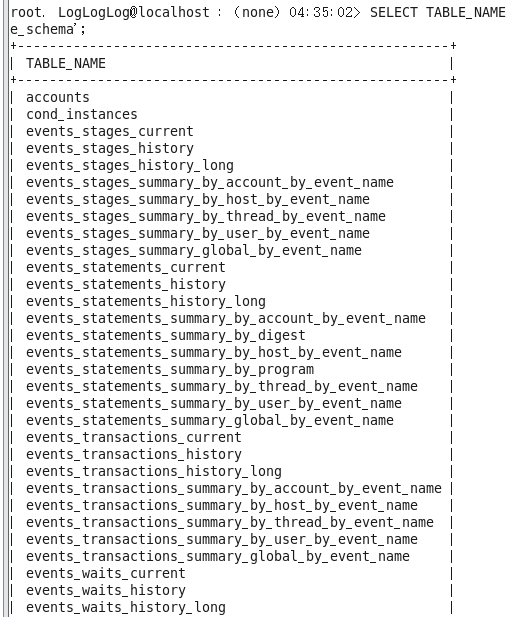
# mysqld启动之后，通过如下语句查看performance\_schema启用是否生效

mysql> show variables like 'performance\_schema';



#通过**INFORMATION\_SCHEMA.TABLES**表查询有哪些performance\_schema引擎表

mysql>SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_SCHEMA ='performance\_schema' and **engine**='performance\_schema';



注：INFORMATION\_SCHEMA.TABLES表记录了mysql server里面的表格元数据，即诸如表名称、使用何种存储引擎等信息，比如可以查询线程表的元数据，它是performance\_schema需要收集的基本信息之一：

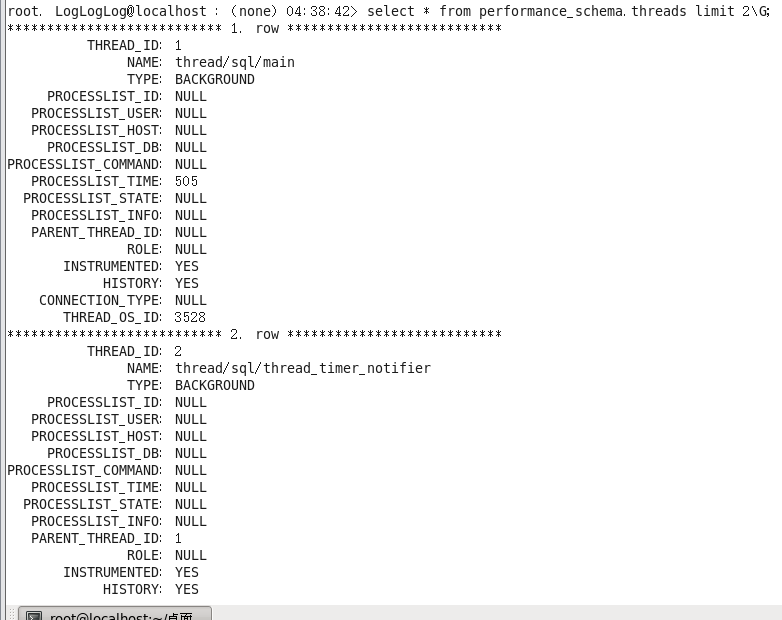
SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE table\_name='threads’ \G

文本, 信件

描述已自动生成

然后可以查询当前有哪些线程：

select \* from performance\_schema.threads limit 2\G;

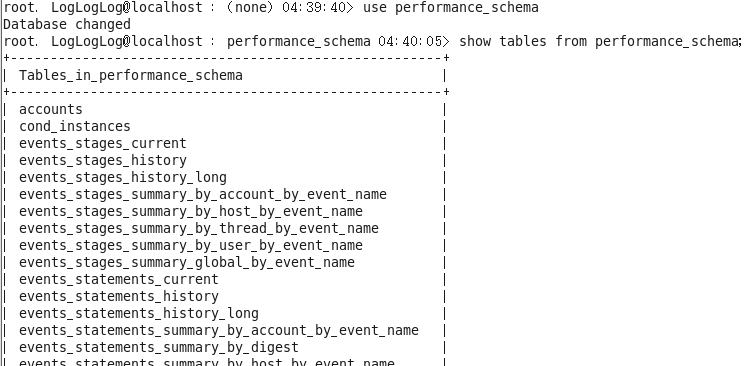


#使用**show tables**语句来查询有哪些performance\_schema引擎表

mysql> use performance\_schema

Database changed

mysql> show tables from performance\_schema;

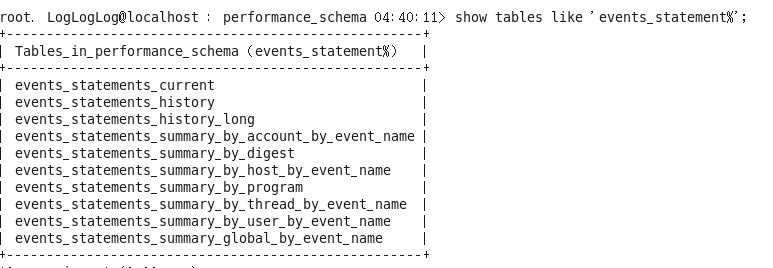


## 4.2.3　performance\_schema表的分类

#performance\_schema库下的表可以按照**监视的不同维度**进行分组(注：注意结合上面的思维导图理解)。

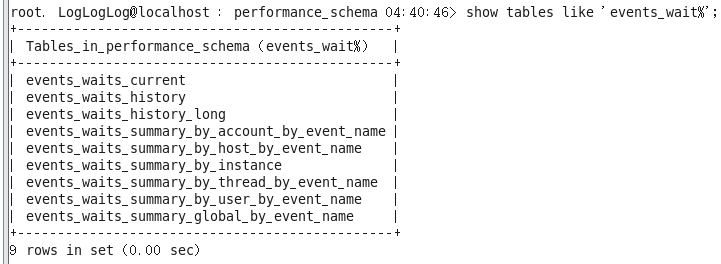
#语句事件记录表：记录语句事件信息的表

mysql> show tables like 'events\_statement%';



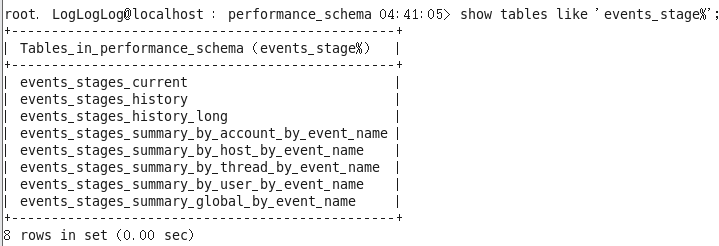
#等待事件记录表：与语句事件记录表类似。

mysql> show tables like 'events\_wait%';



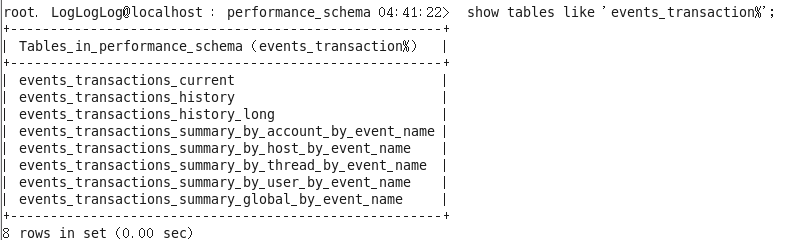
#阶段事件记录表：记录语句执行阶段事件的表，与语句事件记录表类似。

mysql> show tables like 'events\_stage%';



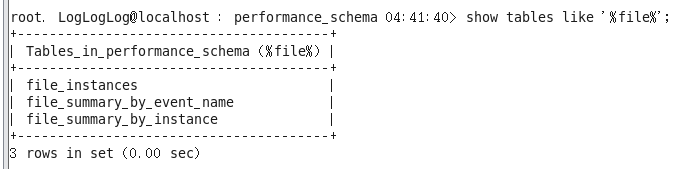
#事务事件记录表：记录与事务相关的事件的表，与语句事件记录表类似

mysql> show tables like 'events\_transaction%';



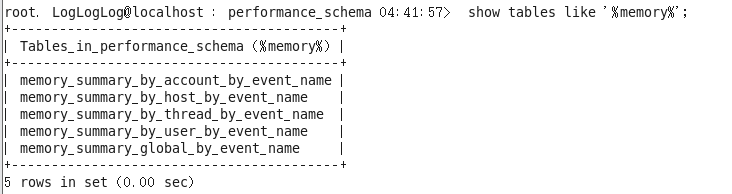
#监视文件系统层调用的表

mysql> show tables like '%file%';



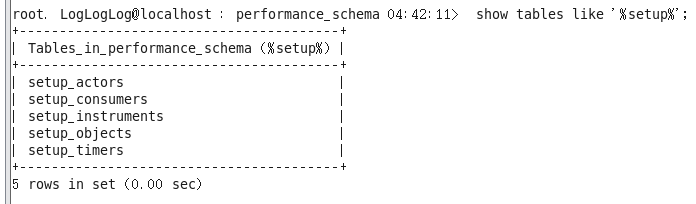
#监视内存使用的表：

mysql> show tables like '%memory%';



#动态对performance\_schema进行配置的配置表

mysql> show tables like '%setup%';



## 4.2.4　performance\_schema简单配置与使用

#如何performance\_schema里的表来提供性能事件数据呢？当数据库初始化完成并启动时，默认不会收集所有的事件。我们以配置监测等待事件数据为例进行说明。

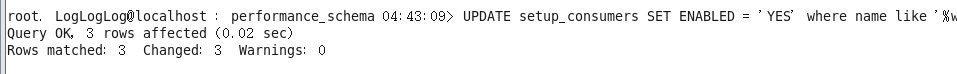
#打开**等待事件**的采集器配置项开关，需要修改**setup\_instruments**配置表中对应的采集器配置项。

mysql> UPDATE **setup\_instruments** SET ENABLED = 'YES', TIMED = 'YES' where name like 'wait%';



#打开**等待事件**的保存表配置项开关，修改**setup\_consumers**配置表中对应的配置项。

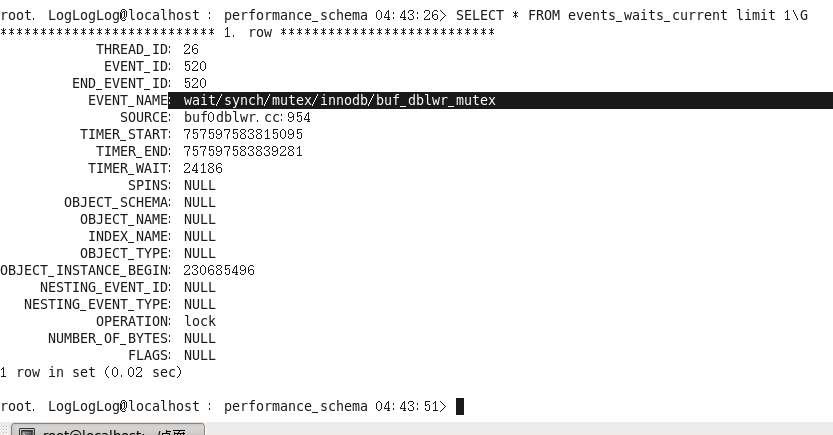
mysql> UPDATE **setup\_consumers** SET ENABLED = 'YES' where name like '%wait%';



#通过查询events\_waits\_current表来得知，该表中每个线程只包含一行数据，用于显示每个线程的最新监视事件（正在做的事情）。

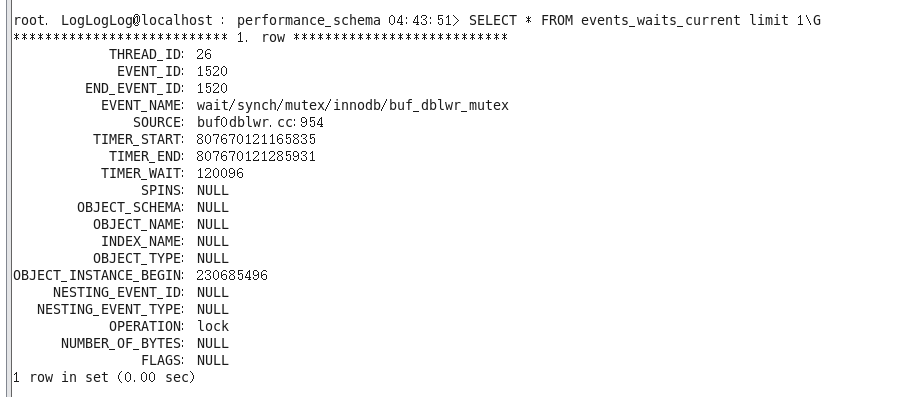
mysql> SELECT \* FROM events\_waits\_current limit 1\G

注：在执行上面这条语句之前，通过如下操作事件开关务必检查是否打开。如果没有打开事件开关（这里比较简单粗暴全部打开了），那么是不会有对应事件收集。一旦打开，马上开始收集事件。关于setup\_consumers和setup\_instruments的详细解释，留到后面介绍。



# 该事件信息表示**线程ID为4**（注：以实际为准）的线程正在等待InnoDB存储引擎的log\_sys\_mutex锁（注：同样以实际为准），这是InnoDB存储引擎的一个互斥锁，等待时间为65664皮秒（\*\_ID列(注：THREAD\_ID, EVENT\_ID, END\_EVENT\_ID))表示事件来自哪个线程、事件编号是多少；EVENT\_NAME表示检测到的具体内容；SOURCE表示这个检测代码在哪个源文件中以及行号；计时器字段TIMER\_START、TIMER\_END、TIMER\_WAIT分别表示该事件的开始时间、结束时间和总的花费时间，如果该事件正在运行而没有结束，那么TIMER\_END和TIMER\_WAIT的值显示为NULL。注：计时器统计的值是近似值，并不是完全精确的）

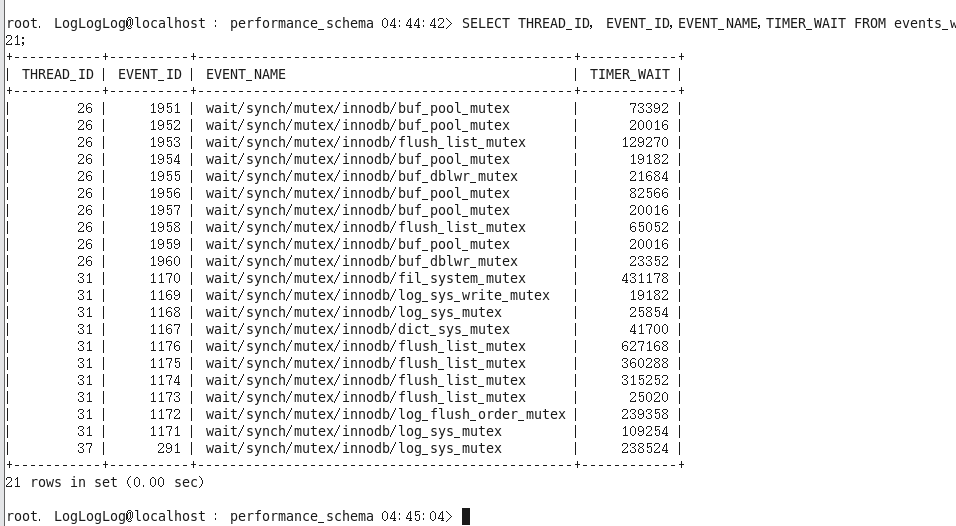
注：以上是书中对查询结果的解释。这里得到另一个查询结果（比较常见）。



因为1皮秒等于一万亿分之一（即10的12次方分之一）秒，所以把时间结果换算成更大单位其实意义不大。在皮秒精度内就可以比较等待时间的长短了。按照书里的解释，同样是一个互斥锁。编号为29的线程为此等待了35793皮秒。这一等待就算是一次事件，编号为1360。我们可以结合SOURCE(buf0dblwr.cc文件第954行)、TIMER\_START等信息继续追溯。

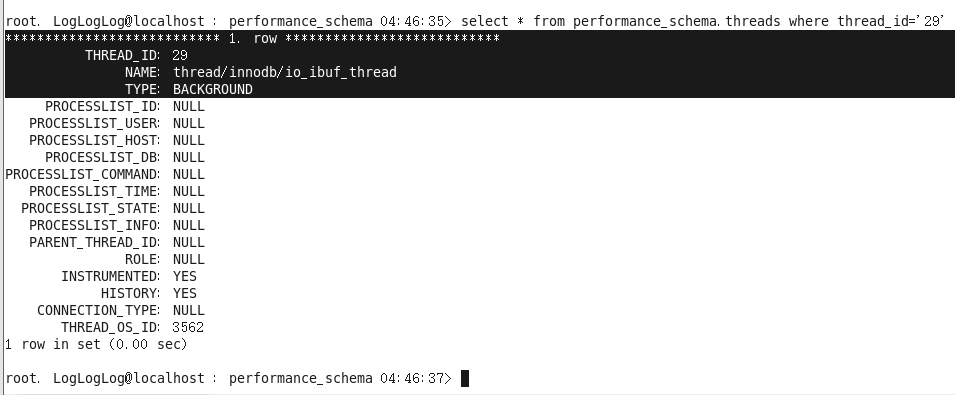
# **\*\_current表中每个线程只保留一条记录**，且一旦线程完成工作，该表中就不会再记录该线程的事件信息了。\*\_history表中记录每个线程已经执行完成的事件信息，但每个线程的事件信息只记录10条，再多就会被覆盖掉。\*\_history\_long表中记录所有线程的事件信息，但总记录数量是10000行，超过会被覆盖掉。

mysql> SELECT THREAD\_ID, EVENT\_ID,EVENT\_NAME,TIMER\_WAIT FROM events\_waits\_history ORDER BY THREAD\_ID limit 21;



注：编号为29的线程在历史记录里还有不少等待事件，都跟缓冲读写的互斥锁有关。其实还可以进一步查查这个编号29的线程是什么：

select \* from performance\_schema.threads where thread\_id='29' \G



可知这个线程叫做“page\_cleaner\_thread”，就是把缓冲池里面的改动过的页面写回到磁盘中。

图示

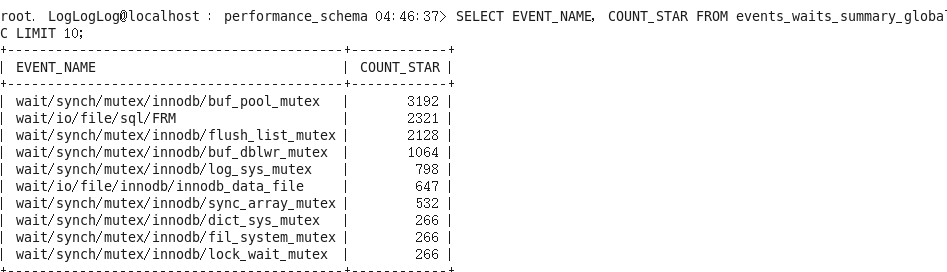
描述已自动生成

图片来源以及更详细的解答：

<https://www.cnblogs.com/geaozhang/p/7225340.html>

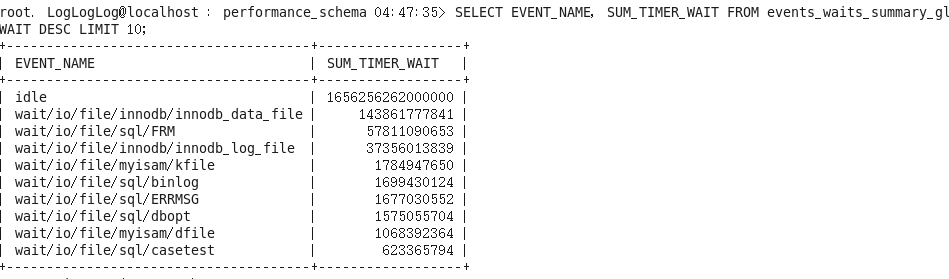
#**summary表提供所有事件的汇总信息**。该组中的表以不同的方式汇总事件数据（如：按用户、按主机、按线程等汇总）。例如：要查看哪些instruments占用的时间最多，则可以通过对events\_waits\_summary\_global\_by\_event\_name表的COUNT\_STAR或SUM\_TIMER\_WAIT列进行查询（这两列是对**事件的记录数**执行COUNT（\*）、事件记录的TIMER\_WAIT列执行SUM（TIMER\_WAIT）统计而来的）。

mysql> SELECT EVENT\_NAME, COUNT\_STAR FROM events\_waits\_summary\_global\_by\_event\_name ORDER BY COUNT\_STAR **DESC LIMIT 10**;



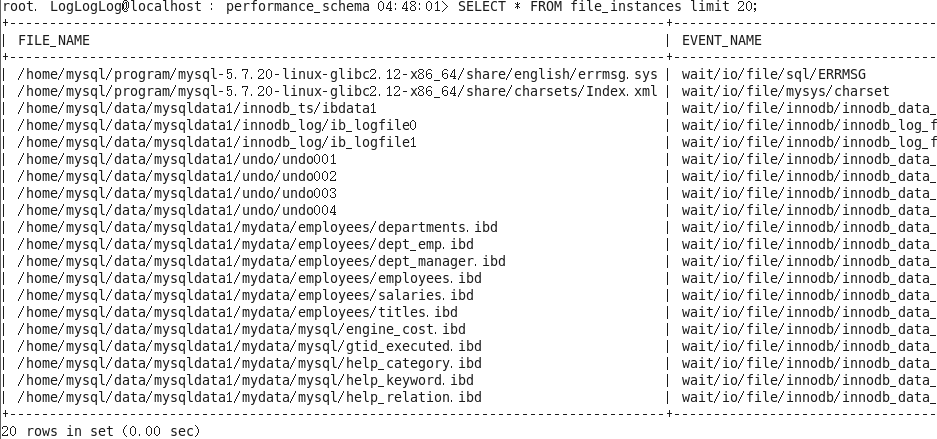
注：这里列出了发生次数排前十的等待事件，基本都是一些互斥锁(mutex)等待事件，其中也包括前面介绍的“线程29”所等待的事件。

mysql> SELECT EVENT\_NAME, SUM\_TIMER\_WAIT FROM events\_waits\_summary\_global\_by\_event\_name **ORDER BY SUM\_TIMER\_WAIT** DESC LIMIT 10;



# **instance表记录了哪些类型的对象会被检测。**这些对象在被Server使用时，在该表中将会产生一条事件记录。例如，file\_instances表列出了文件I/O操作及其关联文件名。

mysql> SELECT \* FROM file\_instances limit 20;



注：结合上面两条SQL语句，可以知道等待事件中主要是关于innodb有关的数据文件I/O。

注：书中最后提到：

#我们大多数时候并不会直接使用performance\_schema来查询性能数据，而是使用sys schema下的视图，为什么不直接学习sys schema呢？那你知道sys schema中的数据是从哪里来的吗？**sys schema中的数据实际上主要是从performance\_schema、information\_schema中获取的**，所以要想玩转sys schema，全面了解performance\_schema非常有必要。

实际也是对本章一开始关于performance\_schema，information\_schema，sys schema三者联系及区别的再次强调，即：

performance\_schema收集性能数据，information\_schema收集元数据，sys通过视图的形式把information\_schema 和performance\_schema结合起来呈现。