目录

[第5章 performance\_schema配置详解 1](#_Toc24251)

[5.2 启动时配置 1](#_Toc21331)

[5.2.1 启动选项 1](#_Toc2712)

[5.2.2　system variables 3](#_Toc32654)

[5.3 运行时配置 4](#_Toc647)

[5.3.1　performance\_timers表 4](#_Toc32121)

[5.3.2　setup\_timers表 5](#_Toc17596)

[5.3.3　setup\_consumers表 5](#_Toc15258)

[5.3.4　setup\_instruments表 6](#_Toc21054)

[5.3.5　setup\_actors表 9](#_Toc14605)

[5.3.6　setup\_objects表 10](#_Toc28445)

[5.3.7　threads表 11](#_Toc3424)

[第6章 performance\_schema应用示例荟萃 13](#_Toc12658)

[6.1　利用等待事件排查MySQL性能问题 13](#_Toc18464)

第5章 performance\_schema配置详解

#两个基本概念：

instruments：生产者，用于**采集MySQL中各种操作产生的事件信息**，对应配置表中的配置项，我们可以称之为事件采集配置项。

consumers：消费者，对应的消费者表用于**存储来自instruments采集的数据**，对应配置表中的配置项，我们可以称之为消费存储配置项。

注：5.1节“编译时配置”部分略

5.2 启动时配置

5.2.1 启动选项

**注：一般来说实验时(非生产环境)往往动态修改配置。也就是说5.2节只需有所了解即可。**

#查看启动选项。之所以叫作启动选项，是因为它们在mysqld启动时就需要通过命令行指定，或者**在my.cnf中指定**，启动之后通过show variables命令无法查看，因为它们不属于system variables）。

[root@localhost ~]# mysqld --verbose --help |grep performance-schema |grep -v '\-\-' |sed '1d' |sed '/[0-9]\+/d'

文本

中度可信度描述已自动生成

**注：sed(stream editor)命令用于处理文本行。以上命令的含义是：把含有performance-schema的选项过滤出来，不过“--** **performance-schema”的不要，然后删除第一行，然后含有数字的也不要，所以最后剩下就是一些TRUE/FALSE的开关，而这些开关都是跟消费者有关的。正因为：**

**performance-schema-consumer-events-waits-current FALSE**

**所以默认并不会收集每个线程的当前等待事件，就是说mysql启动后查对应地表是空表。**

**注：以下是上一章用到的结构图。每个启动选项performance-schema-comsumer-\*如果设置为true，会令对应的一些表格启动数据存储。例如：**

**performance\_schema\_consumer\_events\_statements\_current=TRUE会开启events\_statements\_current表的记录**

**performance\_schema\_consumer\_global\_instrumentation=TRUE会开启各种全局表的记录。**

图示

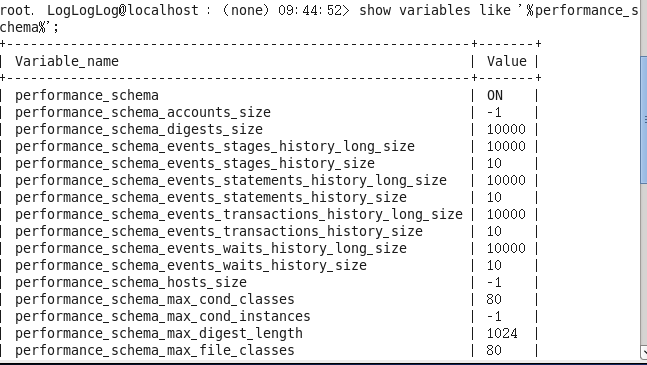
描述已自动生成

**注：关于performance\_schema\_instrument的配置从略。**

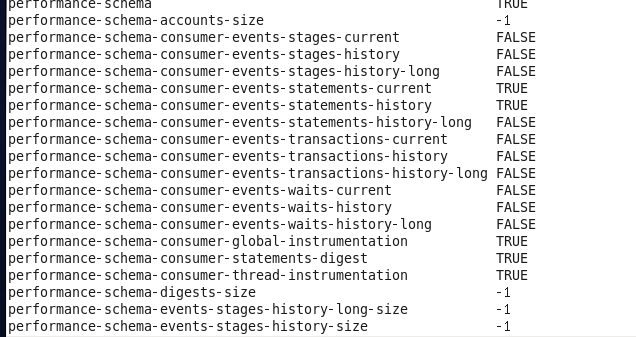
5.2.2　system variables

#与performance\_schema相关的system variables用于**限定consumers表的存储限制**，它们都是**只读**变量。

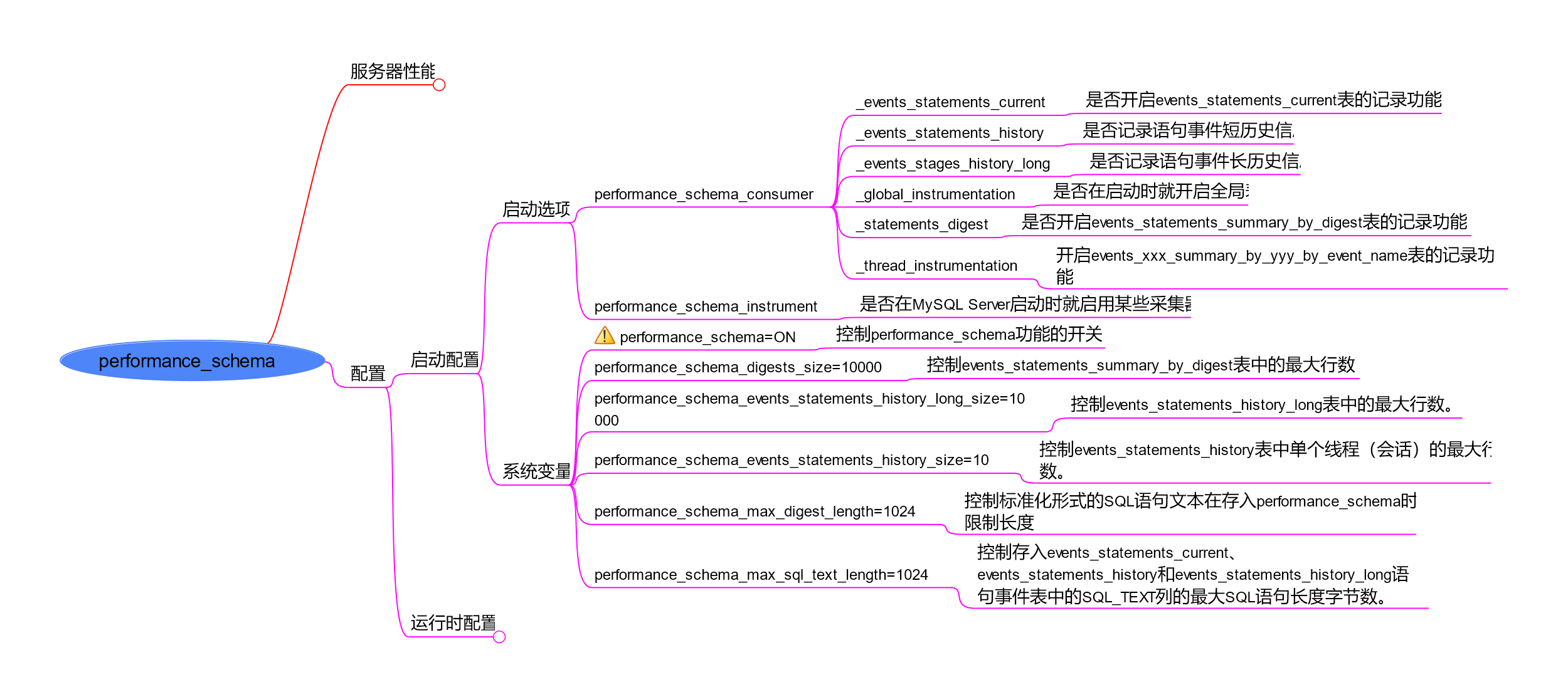
mysql> show variables like '%performance\_schema%';



**注：如果使用5.2.1节那条命令的前半部分，会得到如下结果(局部)：**



**这正好说明正如书中指出，这些系统变量都是在启动时就设置好的了。**

****

5.3 运行时配置

#如何**根据自己的需求**来灵活地开关performance\_schema中的采集信息呢？（比如：在默认配置下很多配置项并未开启，我们可能需要即时修改配置；再如：在高并发场景中，有大量的线程连接到MySQL，执行各种SQL语句产生大量的事件信息，而我们只想查看某一个会话产生的事件信息时，也可能需要即时修改配置）

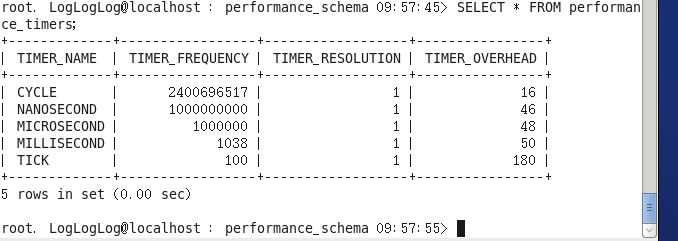
#配置表中的配置项之间存在着关联关系:

图示

低可信度描述已自动生成

5.3.1　performance\_timers表

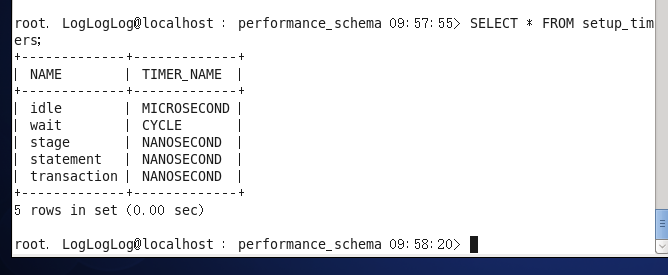
mysql> SELECT \* FROM performance\_timers;



5.3.2　setup\_timers表

#记录了Server中有哪些可用的事件计时器

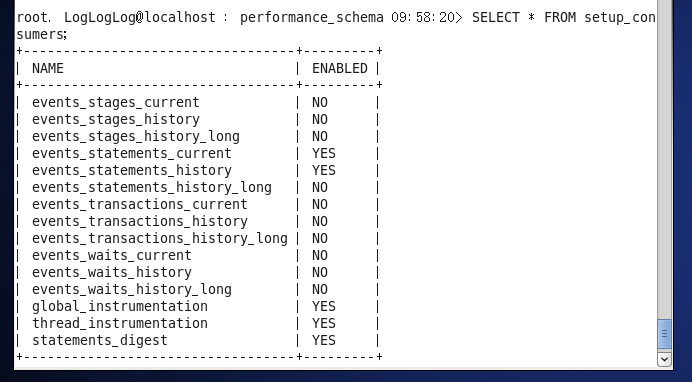
mysql> SELECT \* FROM setup\_timers;



5.3.3　setup\_consumers表

#列出了consumers可配置列表项

mysql> SELECT \* FROM setup\_consumers;

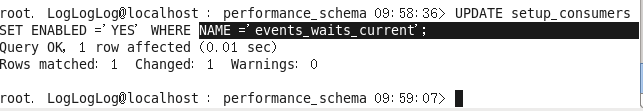


# 打开events\_waits\_current表的当前等待事件记录功能，where条件 ENABLED ='YES' 即表示打开对应的记录表功能

mysql> UPDATE setup\_consumers SET ENABLED ='YES' WHERE NAME ='events\_waits\_current';

图形用户界面, 文本, 应用程序

描述已自动生成

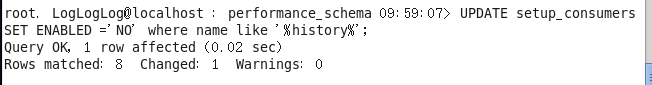


# 关闭历史事件记录功能

mysql> UPDATE setup\_consumers SET ENABLED ='NO' where name like '%history%';

图片包含 表格

描述已自动生成



**注：补充清晰的图5-2** **，setup\_consumers表中的consumers配置项优先级顺序。对于该图的理解不妨对照5.2.1节启动选项，看看有哪些是必须启动的(优先级高)，而有哪些其实并没有启动(优先级低，依赖于高级选项先启动)**

**文本

描述已自动生成**

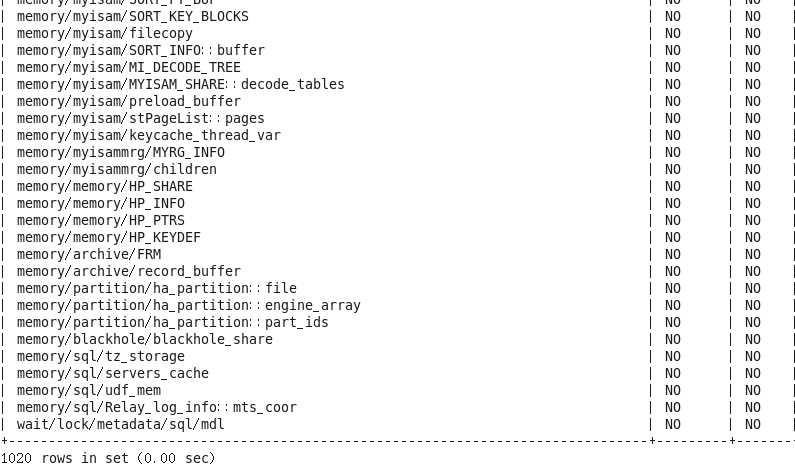
**对书中有关setup\_consumers的内容以及前面的内容小结如下：**

****

5.3.4　setup\_instruments表

# setup\_instruments表记录哪些事件支持被收集。

mysql> SELECT \* FROM setup\_instruments;



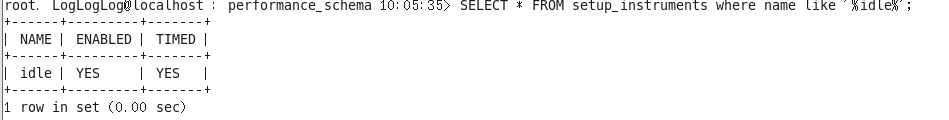
*注：图5-3(setup\_instruments表中的instruments名称层级结构)的高清版：*

**文本

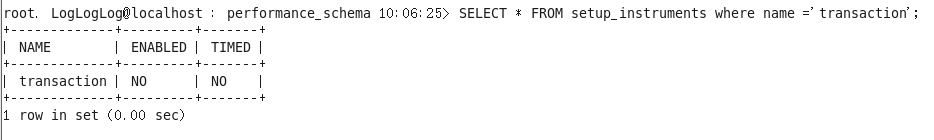
描述已自动生成**

**总共有1020行，这里只显示10行。总共有如下几类。**

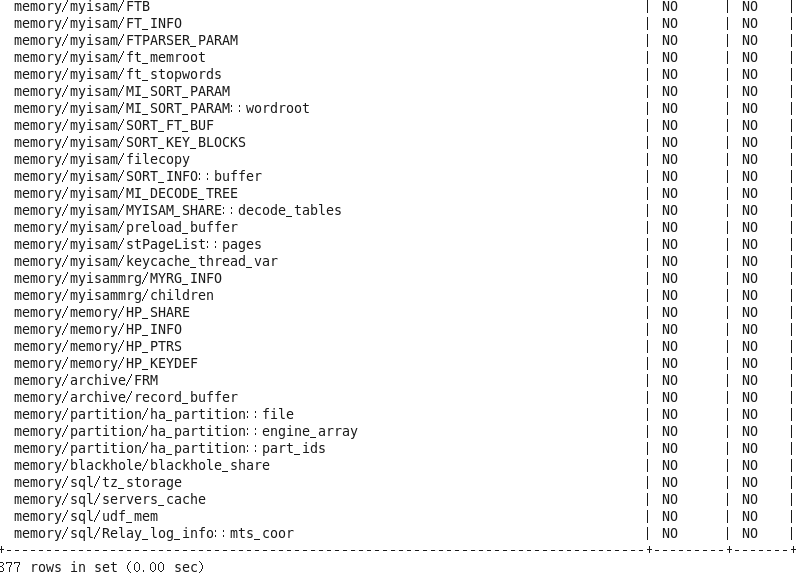
**1. idel（1个）**



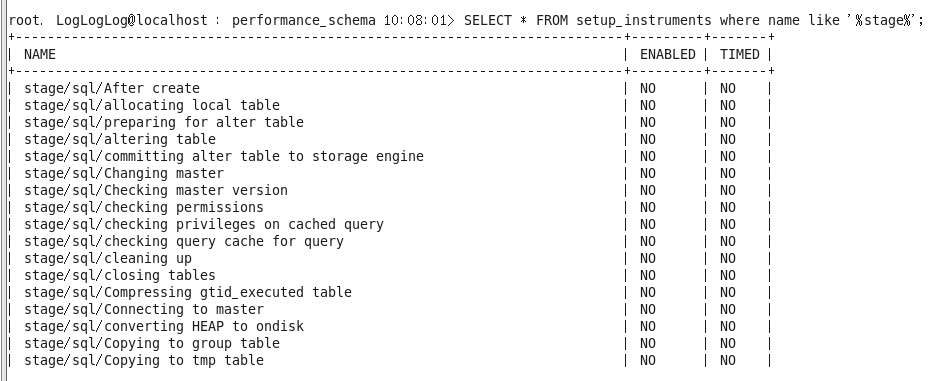
**2. transaction（1个）**



**3. memory(377个)**



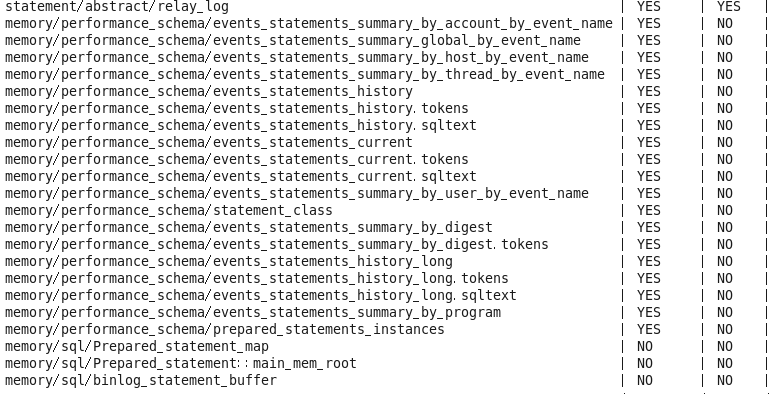
**4. stage(129个)**



**比如修改表所涉及的若干阶段都会触发事件收集：  
文本

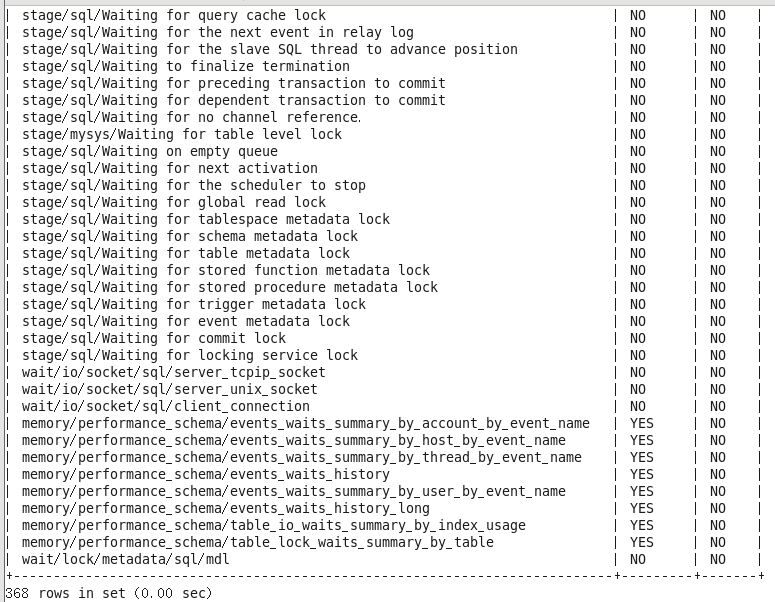
中度可信度描述已自动生成**

**5. statement(193个)**



**每一种类型的SQL语句、命令(cmd)等会触发事件收集。**

**6. wait(319个)**

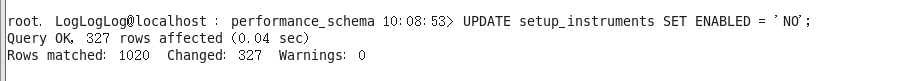


**各种同步等待触发事件收集。**

**注：以下是一些常见收集开关设置手法：**

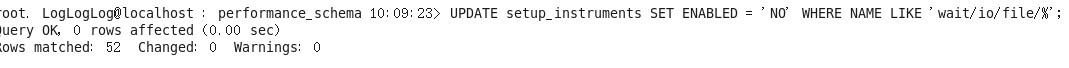
# 禁用所有的instruments，修改之后，生效的instruments修改会立即产生影响，即立即关闭收集功能

mysql> UPDATE setup\_instruments SET ENABLED = 'NO';



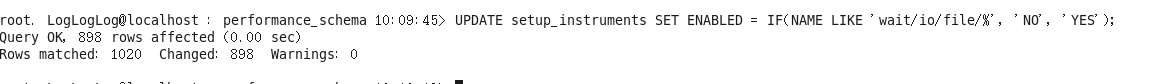
# 禁用所有文件类instruments，使用NAME字段结合like模糊匹配

mysql> UPDATE setup\_instruments SET ENABLED = 'NO' WHERE NAME LIKE 'wait/io/file/%';



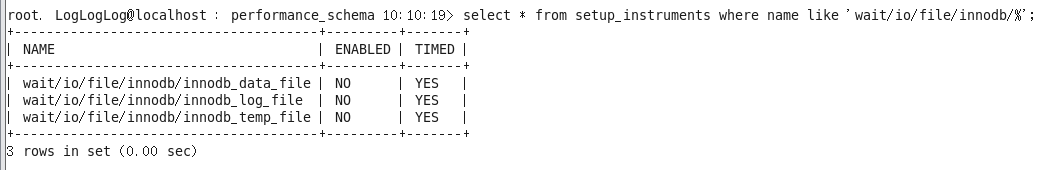
# 仅禁用文件类instruments，启用所有其他instruments，使用NAME字段结合if函数，LIKE模糊匹配到就改为NO，没有匹配到就改为YES

mysql> UPDATE setup\_instruments SET ENABLED = IF(NAME LIKE 'wait/io/file/%', 'NO', 'YES');

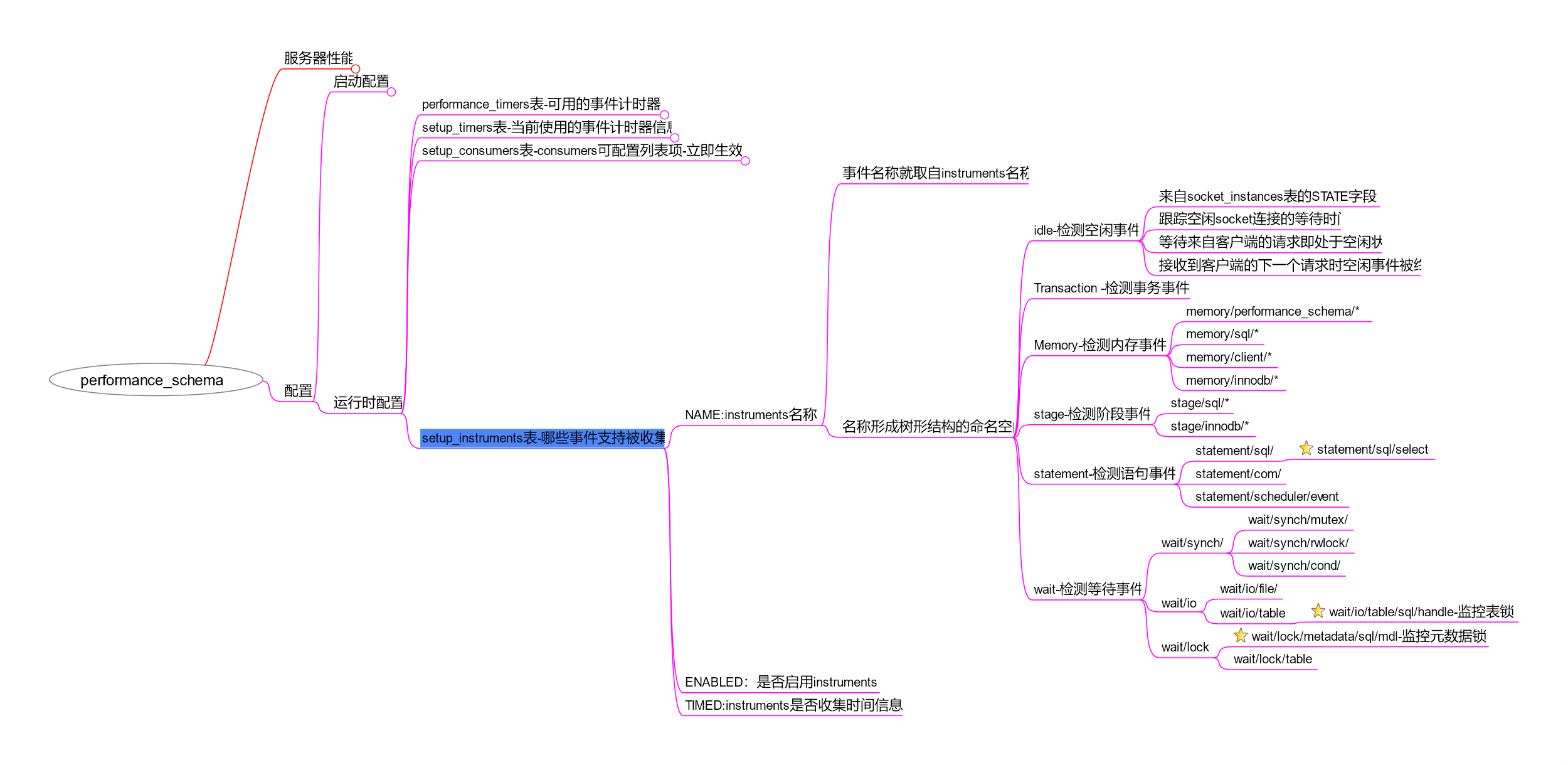


#查找与InnoDB存储引擎文件相关的instruments，可以使用如下语句：

mysql> select \* from setup\_instruments where name like 'wait/io/file/innodb/%';



**注：其余内容类似，略。小结：**

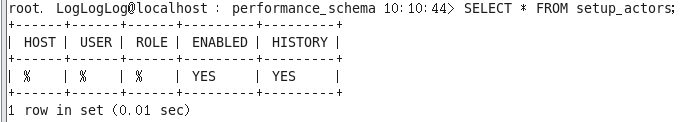
****

5.3.5　setup\_actors表

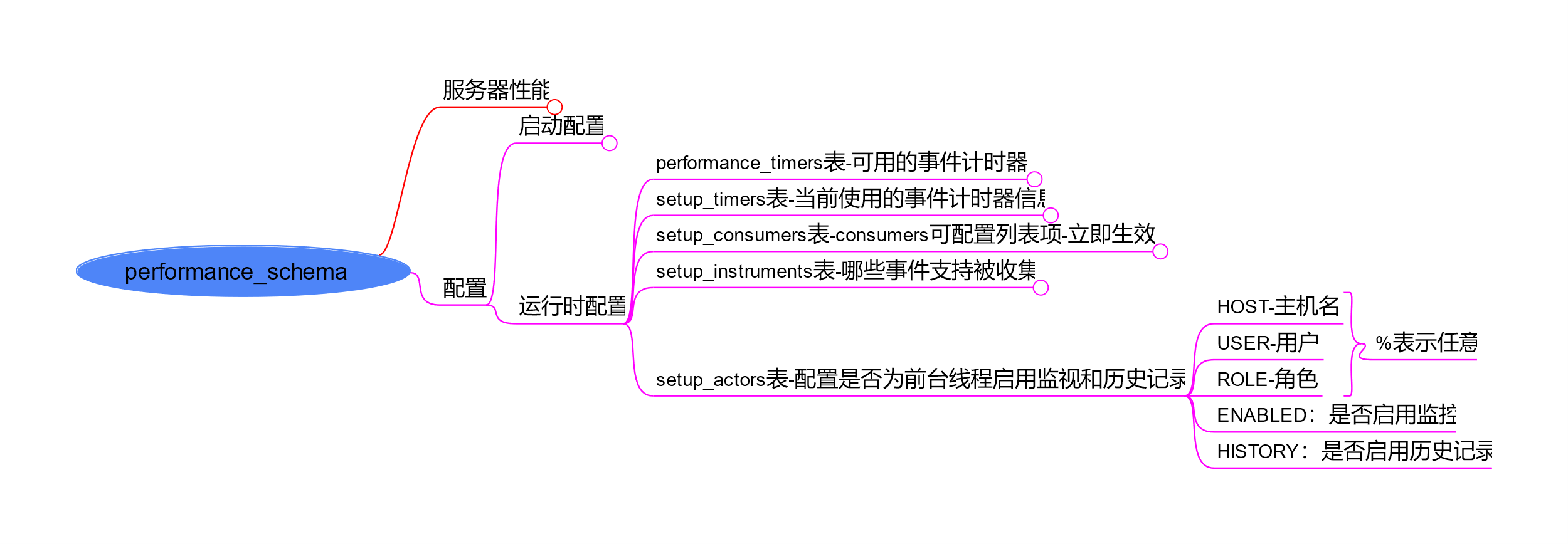
#setup\_actors表用于配置是否为新的前台Server线程（与客户端连接相关联的线程）启用监视和历史事件日志记录。

#setup\_actors表的初始内容是匹配任何用户和主机，因此对于所有的前台线程，默认启用监视和历史事件收集功能

mysql> SELECT \* FROM setup\_actors;



**注：这是最基本的默认设置，即对新建的前台客户端线程一律建立监控和历史日志记录。**

****

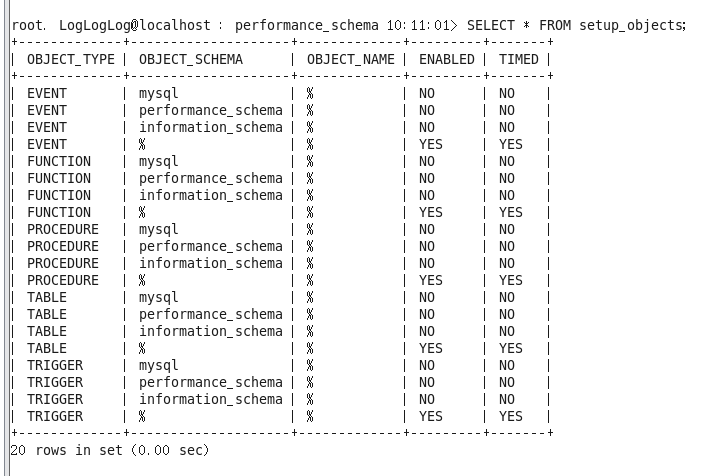
**注：其余内容略。**

5.3.6　setup\_objects表

#setup\_objects表控制performance\_schema是否监控特定对象。默认配置中开启监控的对象**不包含mysql、information\_schema和performance\_schema数据库中的所有表**。

#当performance\_schema在setup\_objects表中进行匹配检测时，会尝试**首先找到最具体（最精确）的匹配项**。例如，在匹配db1.t1表时，它会从setup\_objects表中先查找“db1”和“t1”的匹配项，然后查找“db1”和“%”，最后查找“%”和“%”。

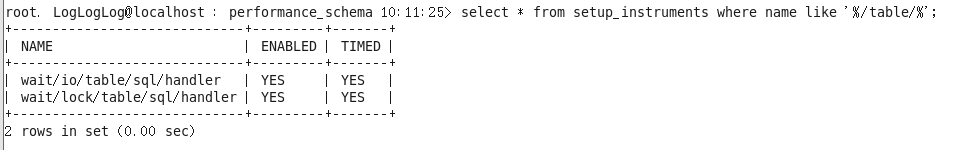
mysql> SELECT \* FROM setup\_objects;



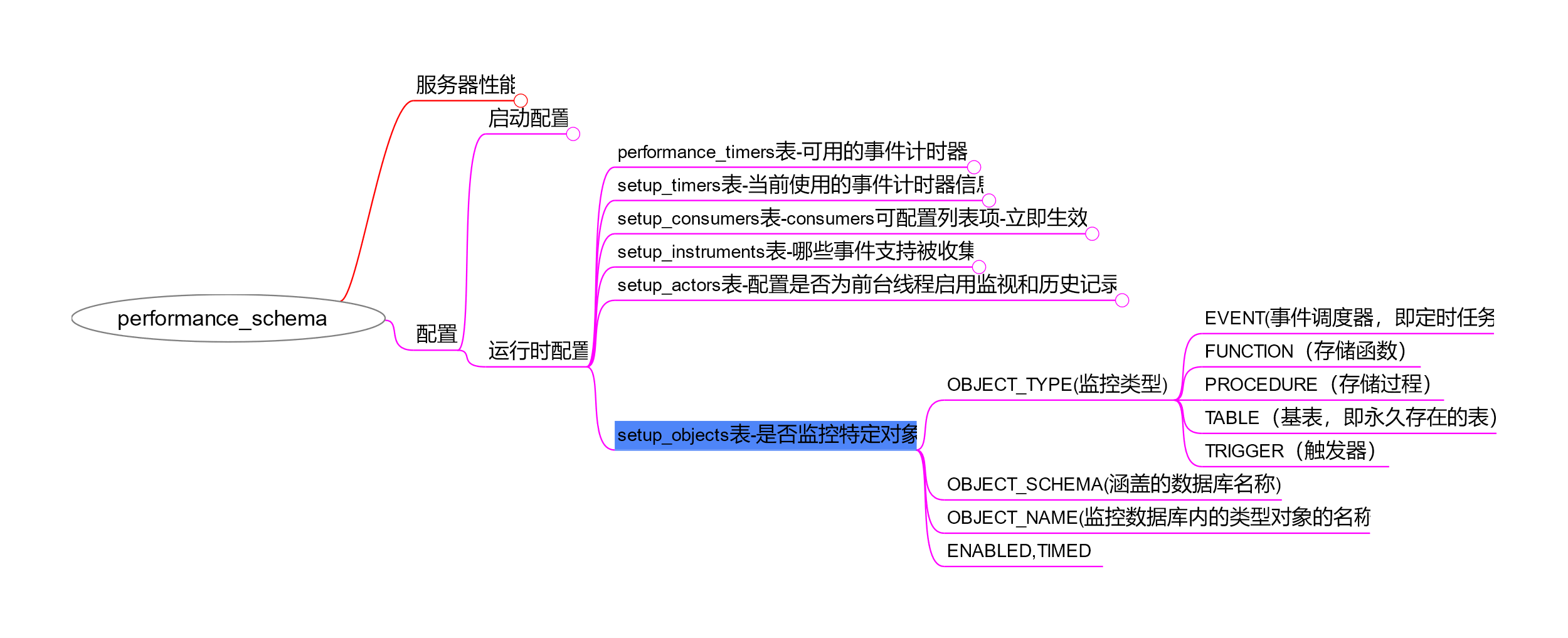
#对于与表对象相关的事件，instruments是否生效**需要将setup\_objects和setup\_instruments两个表中的配置项相结合**，以确定是否启用instruments以及计时器功能。

# setup\_instruments表

mysql> select \* from setup\_instruments where name like '%/table/%';



**注：这里与表锁和表io有关的事件收集已经开启。按默认除mysql、information\_schema和performance\_schema数据库中的所有表不开启监控，其他数据库都开启监控（匹配setup\_objects中的“%”）。所以上面两项事件配置项都会被收集。**

****

**注：其余内容略。**

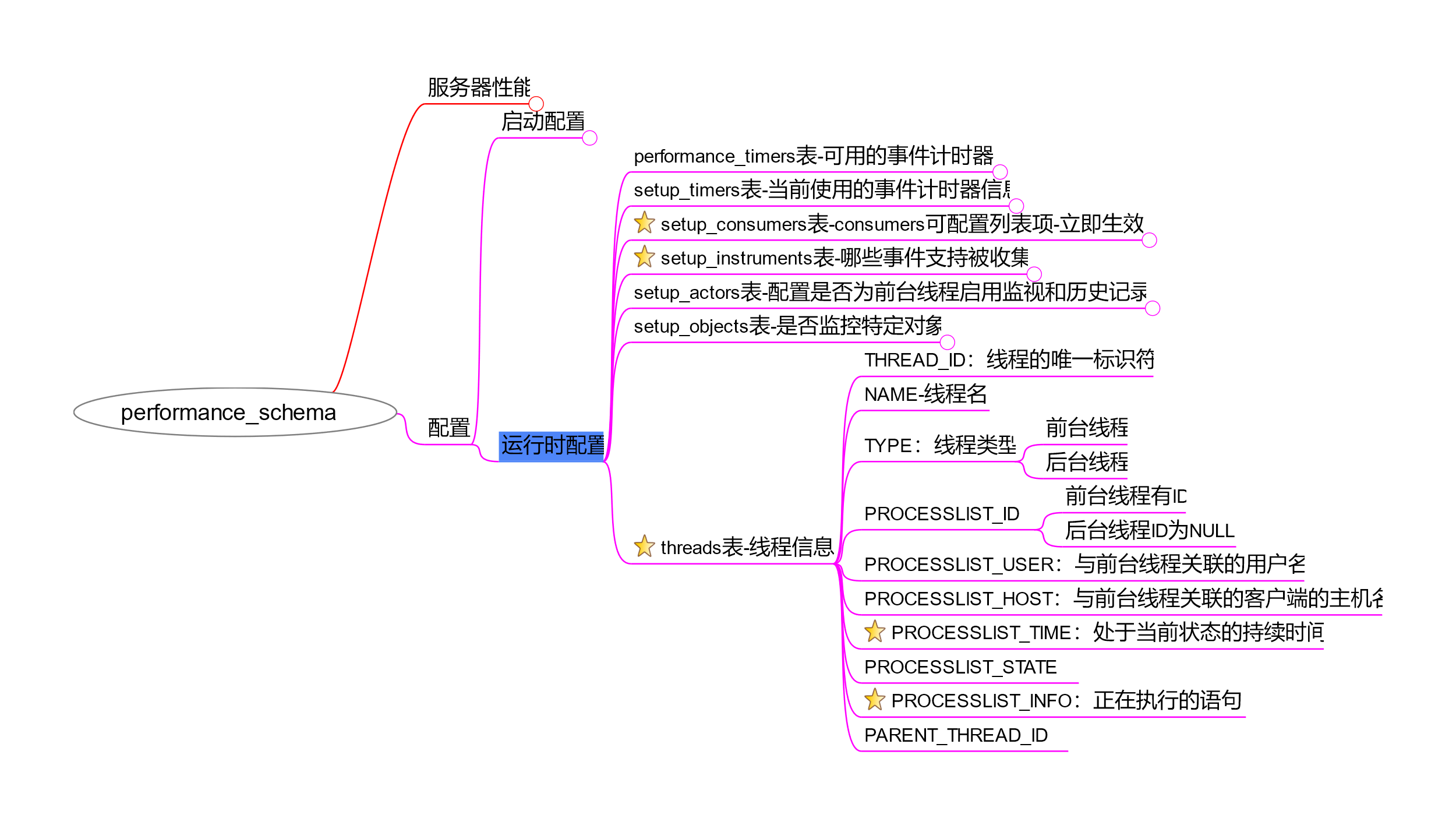
5.3.7　threads表

# threads表对于每个Server线程都会生成一行线程相关信息，当某个线程结束时，会从threads表中删除对应行。

mysql> select \* from threads where TYPE='FOREGROUND' limit 2\G



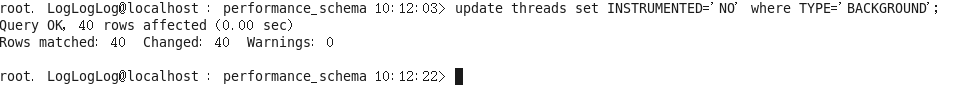
**注：**



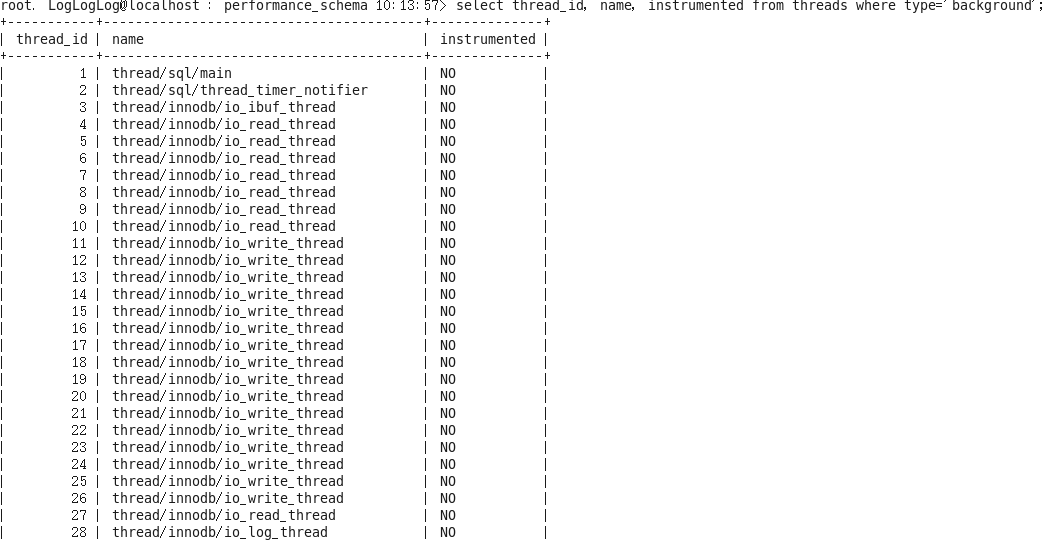
**注：以下是一些对threads表常见的设置手法：**

# 关闭所有后台线程的事件采集功能

mysql> update threads set INSTRUMENTED='NO' where TYPE='BACKGROUND';

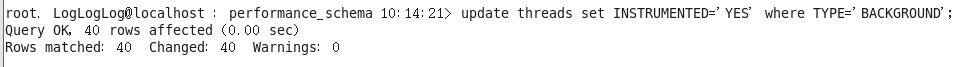


**注：可以去看看有哪些后台线程(即mysql源码中的函数名)被收集信息，比如第一个就是主线程。**



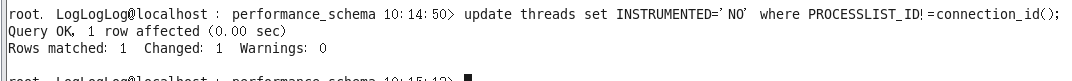
# 开启所有后台线程的事件采集功能

mysql> update threads set INSTRUMENTED='YES' where TYPE='BACKGROUND';

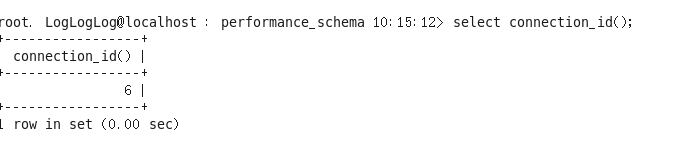


# 关闭除当前连接之外的所有前台线程的事件采集功能

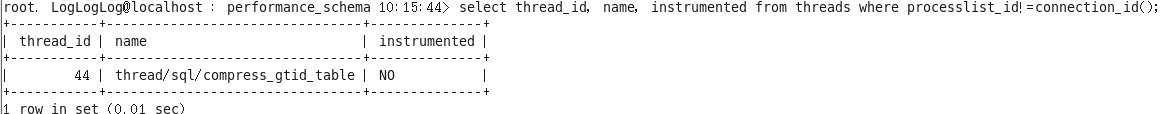
mysql> update threads set INSTRUMENTED='NO' where PROCESSLIST\_ID!=connection\_id();



**注：当前其实只有两个前台线程。当前操作的是3号会话。**

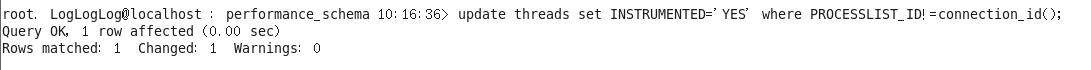


**另一个进程就是前面介绍的负责压缩GTID表的线程：**



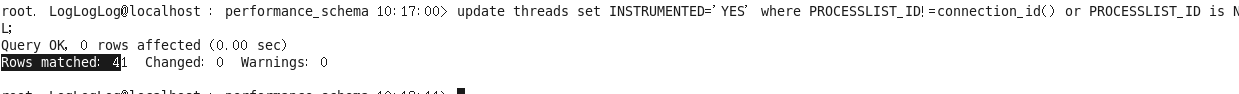
# 开启除当前连接之外的所有前台线程的事件采集功能

mysql> update threads set INSTRUMENTED='YES' where PROCESSLIST\_ID!=connection\_id();



# 当然，如果要连**后台线程**一起操作，请带上条件**PROCESSLIST\_ID is NULL**

update threads set INSTRUMENTED='YES' where PROCESSLIST\_ID!=connection\_id() or PROCESSLIST\_ID is NULL;



提示：关于文中提到的参数的详细解释，可参考本书下载资源中的“附录C”。

**注：即：**<https://github.com/xiaoboluo768/qianjinliangfang/wiki/%E9%99%84%E5%BD%95C-MySQL%E5%B8%B8%E7%94%A8%E9%85%8D%E7%BD%AE%E5%8F%98%E9%87%8F%E5%92%8C%E7%8A%B6%E6%80%81%E5%8F%98%E9%87%8F%E8%AF%A6%E8%A7%A3>

**这些内容可在需要时查阅。**

========================================================

到此为止，我们已基本了解了performance\_schema的基本构成和工作原理。后面将通过若干练习学习如何利用performance\_schema解决mysql的性能问题。

========================================================

**第6章 performance\_schema应用示例荟萃**

## 6.1　利用等待事件排查MySQL性能问题

#通常，在生产服务器上线之前，我们都会对数据库服务器的硬件进行I/O基准测试，对数据库进行增、删、改、查的基准测试(注：用于测定系统现有的性能指标，便于环境条件发生变化时进行对比)，建立基线参考数据，以便为日后的服务器扩容或架构升级提供数据支撑。

注：简单来说，我们可以通过对比既有指标思考现在系统是否真的慢了。至于原因是什么还需要进一步排查。

#以使用sysbench（0.5.x版本）(注：我们用的是1.0版本，我们将结合第44章的内容讨论)基准测试工具压测MySQL数据库为例，介绍如何使用performance\_schema的等待事件来排查数据库性能瓶颈。

注：该节实验需要使用sysbench数据库压测工具。这里先结合第44章的内容介绍sysbench的安装。在后面对应的地方介绍sysbench的基本使用。

sysbench官网：<https://github.com/akopytov/sysbench>，已介绍了下面的安装方式(但不可避免地，需要自行调整)

# 首先下载 Sysbench源码。运行

按部就班安装Sysbench

wget https://github.com/akopytov/sysbench/archive/1.0.20.tar.gz

# 然后将文件重命名并解压缩。



mv 1.0.20.tar.gz sysbench.tar.gz

tar -zxvf sysbench.tar.gz

# 然后安装依赖

sudo yum install -y automake libtoolpkg-config libmysqlclient-dev

# 然后安装编译必须的文件，并编译

cd sysbench-1.0.20/./autogen.sh

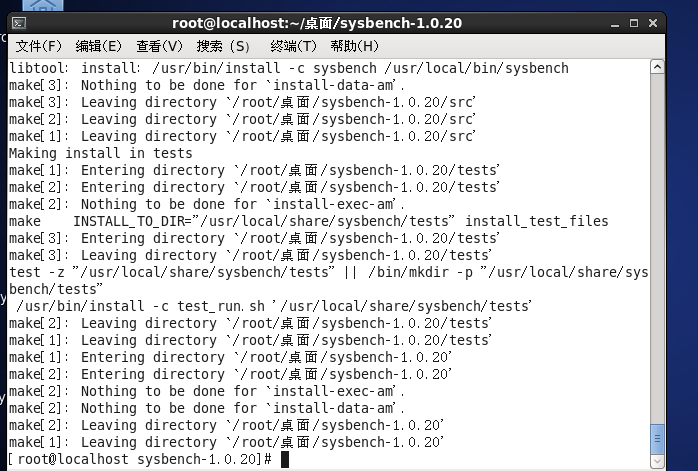
./configure

sudo make -j

sudo make install

安装成功



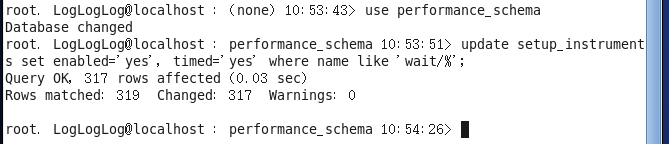


出现以下错误，使用符号连接方法解决



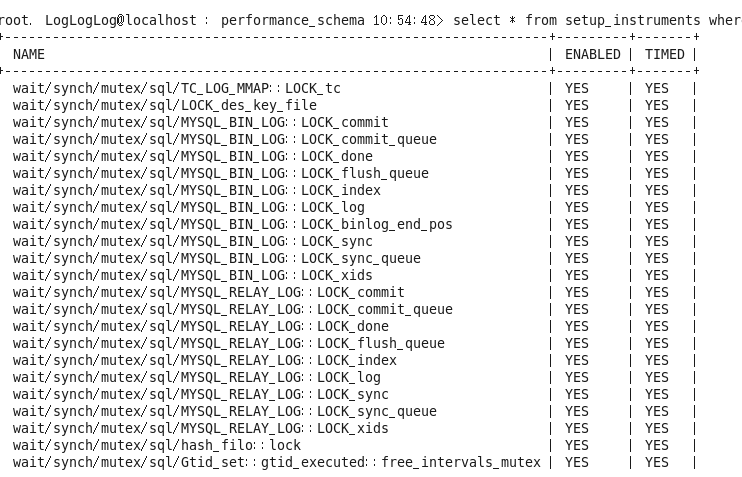
mysql> use performance\_schema

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



# 查看修改结果，enabled和timed字段为yes即表示当前instruments已经启用（但此时采集器并不会立即采集事件数据，需要保存这些等待事件的表，**当consumers启用之后才会开始采集**）

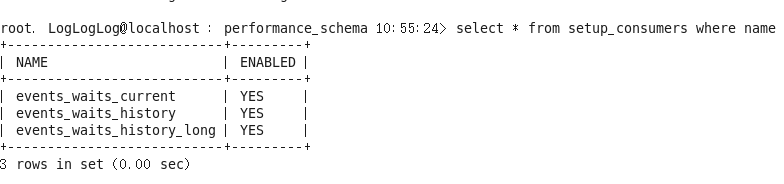
mysql> select \* from setup\_instruments where name like 'wait/%';



# 启用等待事件的consumers

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

mysql> select \* from setup\_consumers where name like '%wait%';



========================================================

然后增加测试数据库：



然后执行如下命令准备(注意根据实际环境调整命令中的参数，比如密码等)：

**注：以上为旧版命令。新版使用如下命令：**

[root@localhost ~]#sysbench /usr/share/sysbench/oltp\_read\_write.lua --tables=5 **--table\_size=5000000** --mysql-user=root --mysql-password=logloglog --mysql-host=localhost --mysql-port=3306 --mysql-db=sysbench\_test **--threads=16** --time=30 --report-interval=3 run

图片包含 日历

描述已自动生成

**注：多数选项都可以根据英文单词来理解，更详细的解释第44章(496-497页)。可根据实际灵活调整。最后一个“run”是要执行的内部命令，参考44.3.4节。**

**以下是书中的对它实验结果的解释，可以拿来参考实际做出来的结果。**

#从sysbench的输出结果中可以看到，在16个并发线程的OLTP**(注：online transactional processing，在线事务处理)**压力下，TPS不到100，且延迟时间超过600ms，说明存在严重的性能瓶颈（或者在MySQL内部发生了严重的互斥等待，或者硬件设备的性能严重不足）。现在，我们先使用操作系统命令查看硬件负载情况。

**此外第44章(504页)有解释“中间统计结果”，其中：**

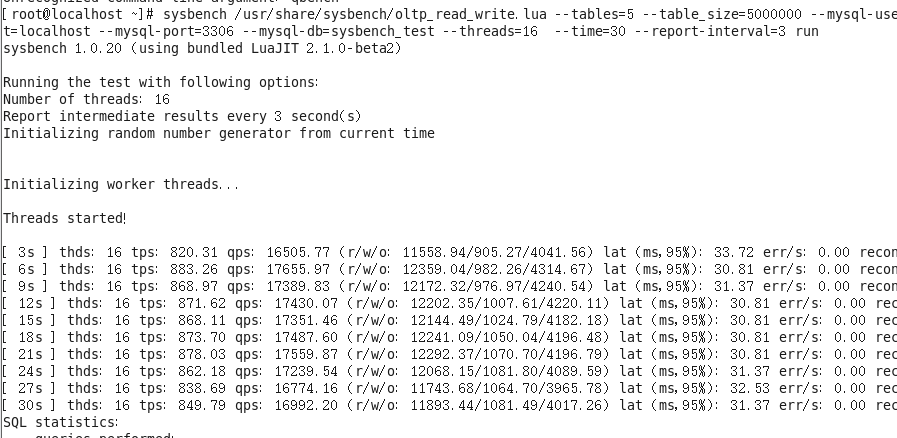
tps: 表示在report-interval时间间隔内的每秒事务数。

qps: 表示在report-interval时间间隔内的每秒查询数。

（r/w/o: \*/\*/\*）：表示在report-interval时间间隔内的每秒读/写/其他请求数，用于补充说明qps。

lat（ms,95%）：表示在report-interval时间间隔内的请求95%的延迟时间在\*\*ms以下。

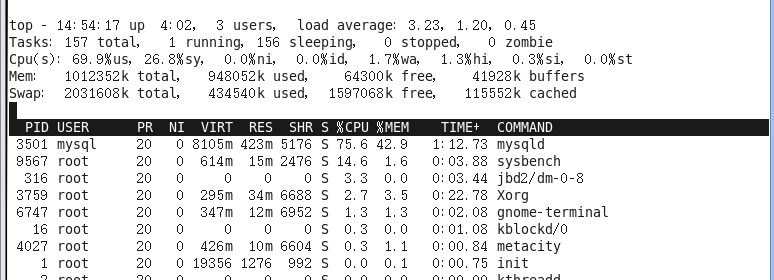
**不过要看到书中所描述的“严重的互斥等待”，不能用固态硬盘来做实验。所以把虚拟机迁移到某个机械硬盘上（甚至U盘上），并且增加并发线程数和表大小，并且延长了测试时间以便于观察：**

to

**这时延迟已经大幅度增加。**

# 通过top命令查看到CPU资源绝大部分都消耗在了%wa上，说明I/O设备性能出现严重不足

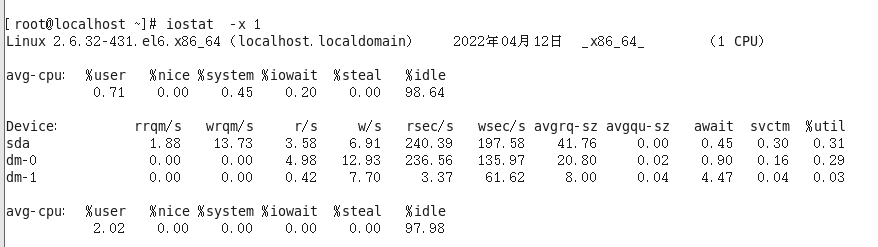
[root@localhost ~]# top



**注：已经可以看到CPU等待时间比例的大幅度升高(平常这个数字为0)。不过因为硬盘还是比较快，所以还是没有书中所描述的效果。**

# 使用iostat命令查看磁盘负载，通过%util列可以看到，磁盘处于100%满负载状态

**注：用iostat -x 1刷新看%util列。Iostat有关介绍可参考41.6节(第455页)**



# 为了方便查询等待事件统计，我们可以先创建一个视图(**sys.test\_waits**)，用于实时统计当前等待事件（非历史数据）

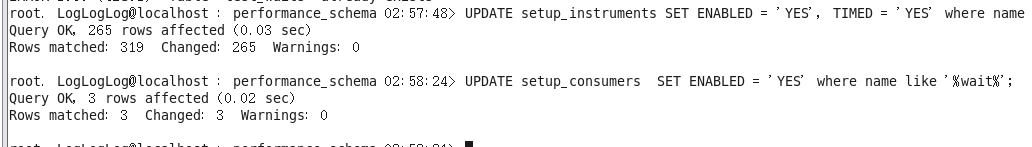
mysql> create view **sys.test\_waits** as select sum(TIMER\_WAIT) as TIMER\_WAIT,sum(NUMBER\_OF\_BYTES) as NUMBER\_OF\_BYTES, EVENT\_NAME,OPERATION from **events\_waits\_current** where **EVENT\_NAME!='idle'** group by EVENT\_NAME,OPERATION;



**注：首先应当参考4.2.4，开启io等待事件的收集和记录功能：**

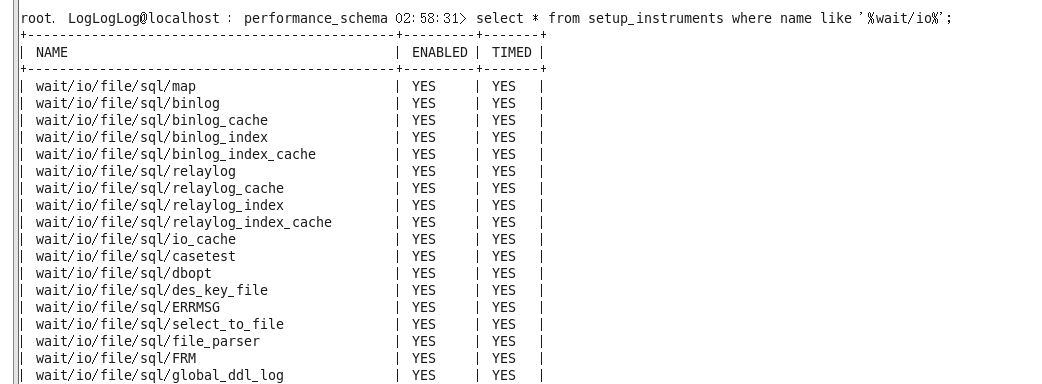
mysql>UPDATE setup\_instruments SET ENABLED = 'YES', TIMED = 'YES' where name like 'wait%';

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

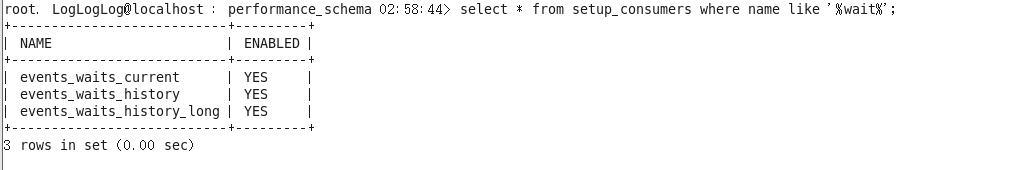


**然后可以查看到相关的设置已经完成：**

mysql> select \* from setup\_instruments where name like '%wait/io%';



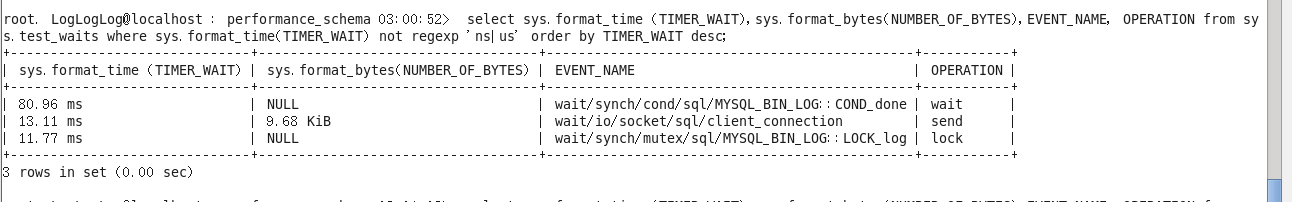
mysql > select \* from setup\_consumers where name like '%wait%';



# 使用上面创建的视图(**sys.test\_waits**)进行查询，并对查询结果进行降序排列。从下面的查询结果中可以看到，时间开销排名前5的有4个**都是与I/O相关的等待事件，**剩下1个是**与binlog相关的互斥等待事件。**

mysql> select sys.format\_time (TIMER\_WAIT),sys.format\_bytes(NUMBER\_OF\_BYTES),EVENT\_NAME, OPERATION **from sys.test\_waits** where sys.format\_time(TIMER\_WAIT) not regexp 'ns|us' order by TIMER\_WAIT desc;

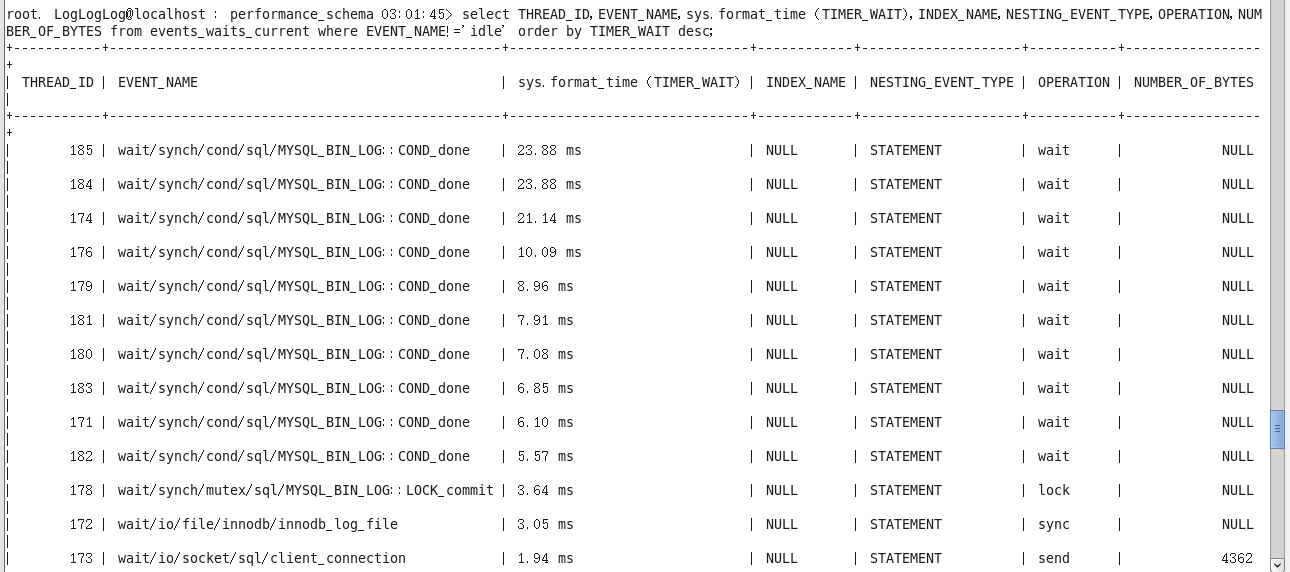
**注：首先要开启压测，然后再查看视图，否则视图为空。可以反复多次查看视图直到压测结束。**



# 当然，你也可以直接查询events\_waits\_current表（返回的数据行数可能比较多，且对查询结果没有做分组聚合，是逐行的事件记录数据）

mysql> select THREAD\_ID,EVENT\_NAME,sys.format\_time (TIMER\_WAIT),INDEX\_NAME,NESTING\_EVENT\_TYPE,OPERATION,NUMBER\_OF\_BYTES from events\_waits\_current where EVENT\_NAME!='idle' order by TIMER\_WAIT desc;

**注：也是要一边压测一边查询：**



**由于压测时插入大量的随机数据，而这些操作全部都要记录在二进制日志binlog中，所以会消耗大量的时间在等待日志更新操作上。**

