# mysql5.6之performance\_schema使用初探

- Mysql频道\_数据库 - 360sdn.com\_专业优秀的程序员网上知识家园

http://www.360sdn.com/mysql/2013/0515/127.html

**一：Performance Schema的开启**

在mysql中新增了一个性能优化的引擎,在mysql5.6.6之前Performance Schema是默认关闭的,在Mysql5.6.6之后Performance Schema默认是开启的。所以mysql5.6.6之前的版本要使用Performance Schema需要用手工开启,首先输入命令查看当前数据库中performance schema的开启情况。

首先在dos命令窗口输入以下命令：

[?](http://www.360sdn.com/mysql/2013/0515/127.html)

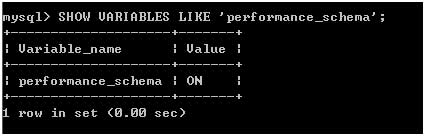
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | mysqld --verbose --help    performance-schema                                FALSE  performance-schema-events-waits-history-long-size 10000  performance-schema-events-waits-history-size      10  performance-schema-max-cond-classes               80  performance-schema-max-cond-instances             1000  performance-schema-max-file-classes               50  performance-schema-max-file-handles               32768  performance-schema-max-file-instances             10000  performance-schema-max-mutex-classes              200  performance-schema-max-mutex-instances            1000000  performance-schema-max-rwlock-classes             30  performance-schema-max-rwlock-instances           1000000  performance-schema-max-table-handles              100000  performance-schema-max-table-instances            50000  performance-schema-max-thread-classes             50  performance-schema-max-thread-instances           1000 |

然后在my.ini配置文件中输入以下内容,开启performance\_schema参数

[?](http://www.360sdn.com/mysql/2013/0515/127.html)

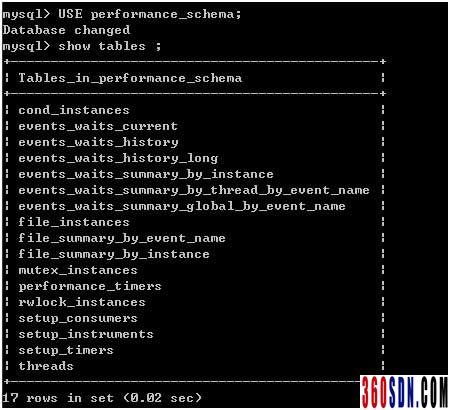
|  |  |
| --- | --- |
| 1  2 | [mysqld]  performance\_schema=on |

然后重启mysql服务.在用命令查看,如下:



**二：Performance Schema的深入了解**

在mysql的命令窗口输入图片中的命令,看看有什么,看以下命令截图：



这里的数据表分为几类：  
1、 setup table ： 设置表，配置监控选项。  
2、 current events table : 记录当前那些thread 正在发生什么事情。  
3、 history table 发生的各种事件的历史记录表  
4、 summary table 对各种事件的统计表  
5、 杂项表。

**三、setup 表的说明**

mysql\_Performance Schema\_setup 表 setup\_consumers 描述各种事件 setup\_instruments 描述这个数据库下的表名以及是否开启监控。 setup\_timers 描述 监控选项已经采样频率的时间间隔

[?](http://www.360sdn.com/mysql/2013/0515/127.html)

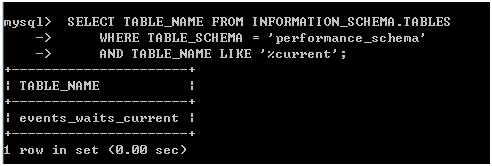
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | mysql> SELECT \* FROM setup\_timers;    +-----------+-------------+    | NAME      | TIMER\_NAME  |    +-----------+-------------+    | idle      | MICROSECOND |    | wait      | CYCLE       |    | stage     | NANOSECOND  |    | statement | NANOSECOND  |    +-----------+-------------+ |

这个要多说一点 目前 performance-schema 只支持 'wait' 时间的监控，代码树上 wait/ 下的函数都可以监控到。

**四、性能事件表**

[?](http://www.360sdn.com/mysql/2013/0515/127.html)

|  |  |
| --- | --- |
| 1  2  3 | SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES       WHERE TABLE\_SCHEMA = 'performance\_schema'       AND TABLE\_NAME LIKE '%current'; |

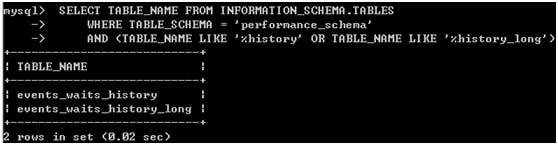


记录当前正在发生的等待事件，这个表是只读的表，不能update ，delete ，但是可以truncate

**五、性能历史表**

[?](http://www.360sdn.com/mysql/2013/0515/127.html)

|  |  |
| --- | --- |
| 1  2  3 | SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES       WHERE TABLE\_SCHEMA = 'performance\_schema'       AND (TABLE\_NAME LIKE '%history' OR TABLE\_NAME LIKE '%history\_long'); |



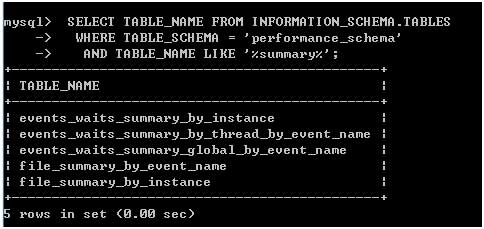
这些表与前面的性能表的结构是一致的， history 表只保留每个线程（thread） 的最近的10个事件， history\_long 记录最近的10000个事件。

新事件如表，如果旧表满了，就会丢弃旧的数据，标准的先进先出（FIFO) 这俩表也是只读表，只能truncate

**六、事件汇总表**

[?](http://www.360sdn.com/mysql/2013/0515/127.html)

|  |  |
| --- | --- |
| 1  2  3 | SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES    WHERE TABLE\_SCHEMA = 'performance\_schema'     AND TABLE\_NAME LIKE '%summary%'; |

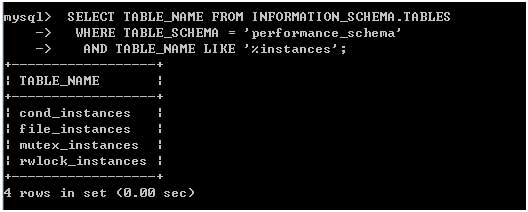


按照相关的标准对进行的事件统计表，events\_waits\_summary\_global\_by\_event\_name, 在mysql5.5.7 以前叫： EVENTS\_WAITS\_SUMMARY\_BY\_EVENT\_NAME表也是只读的，只能turcate

**七、性能优化实例（performance schema instance） 表**

[?](http://www.360sdn.com/mysql/2013/0515/127.html)

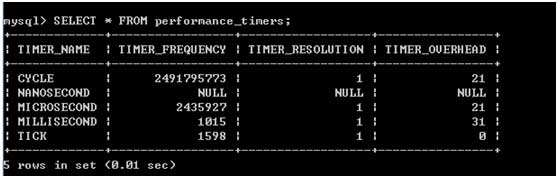
|  |  |
| --- | --- |
| 1  2  3 | SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES    WHERE TABLE\_SCHEMA = 'performance\_schema'     AND TABLE\_NAME LIKE '%instances'; |



记录各种等待事件涉及到的实例，主要是3类: cond (容器） mutex （互斥锁） ，rwlock （读写锁）这表是只读的。

**八、performance\_timers 表**

SELECT \* FROM performance\_timers;

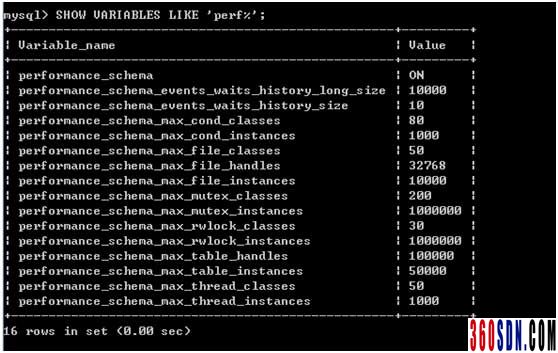


这个表式只读表，记录了事件采样频率的设定，我们前面说的setup\_timer 表的timer\_name 只能是这几个中一个。

**九、performance\_schema的各个系统参数**

[?](http://www.360sdn.com/mysql/2013/0515/127.html)

|  |  |
| --- | --- |
| 1 | SHOW VARIABLES LIKE 'perf%'; |

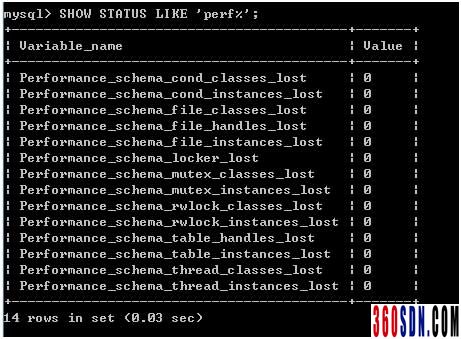


涉及到系统状态的参数：

[?](http://www.360sdn.com/mysql/2013/0515/127.html)

|  |  |
| --- | --- |
| 1 | SHOW STATUS LIKE 'perf%'; |

mysql\_performance\_schema系统状态的参数



# MySQL5.6 PERFORMANCE\_SCHEMA 说明

- jyzhou - 博客园

http://www.cnblogs.com/zhoujinyi/p/5236705.html

[MySQL5.6 PERFORMANCE\_SCHEMA 说明](http://www.cnblogs.com/zhoujinyi/p/5236705.html)

**背景：**

      MySQL 5.5开始新增一个数据库：PERFORMANCE\_SCHEMA，主要用于收集数据库服务器性能参数。并且库里表的存储引擎均为PERFORMANCE\_SCHEMA，而用户是不能创建存储引擎为PERFORMANCE\_SCHEMA的表。MySQL5.5默认是关闭的，需要手动开启，在配置文件里添加：

[mysqld]

performance\_schema=ON

查看是否开启：

mysql>show variables like 'performance\_schema';

+--------------------+-------+

| Variable\_name | Value |

+--------------------+-------+

| performance\_schema | **ON** |

+--------------------+-------+

从MySQL5.6开始，默认打开，本文就从MySQL5.6来说明，在数据库使用当中PERFORMANCE\_SCHEMA的一些比较常用的功能。具体的信息可以查看[官方文档](http://dev.mysql.com/doc/refman/5.6/en/performance-schema.html)。

**相关表信息：**

**一：配置（setup）表：**

[复制代码](javascript:void(0);)

zjy@performance\_schema **10**:**16**:**56**>show tables like '%setup%';

+----------------------------------------+

| Tables\_in\_performance\_schema (%setup%) |

+----------------------------------------+

| setup\_actors |

| setup\_consumers |

| setup\_instruments |

| setup\_objects |

| setup\_timers |

+----------------------------------------+

[复制代码](javascript:void(0);)

**1，setup\_actors**：配置用户纬度的监控，默认监控所有用户。

zjy@performance\_schema **10**:**19**:**11**>select \* from setup\_actors;

+------+------+------+

| HOST | USER | ROLE |

+------+------+------+

| % | % | % |

+------+------+------+

**2，setup\_consumers**：配置events的消费者类型，即收集的events写入到哪些统计表中。

[复制代码](javascript:void(0);)

zjy@: performance\_schema **10**:**23**:**35**>select \* from setup\_consumers;

+--------------------------------+---------+

| NAME | ENABLED |

+--------------------------------+---------+

| events\_stages\_current | NO |

| events\_stages\_history | NO |

| events\_stages\_history\_long | NO |

| events\_statements\_current | YES |

| events\_statements\_history | NO |

| events\_statements\_history\_long | NO |

| events\_waits\_current | NO |

| events\_waits\_history | NO |

| events\_waits\_history\_long | NO |

| global\_instrumentation | YES |

| thread\_instrumentation | YES |

| statements\_digest | YES |

+--------------------------------+---------+

[复制代码](javascript:void(0);)

这里需要说明的是需要查看哪个就更新其ENABLED列为YES。如：

zjy@performance\_schema **10**:**25**:**02**>update setup\_consumers set ENABLED='YES' where NAME in ('events\_stages\_current','events\_waits\_current');

Query OK, **2** rows affected (**0.00** sec)

更新完后立即生效，但是服务器重启之后又会变回默认值，要永久生效需要在配置文件里添加：

[复制代码](javascript:void(0);)

[mysqld]

#performance\_schema

performance\_schema\_consumer\_events\_waits\_current=on

performance\_schema\_consumer\_events\_stages\_current=on

performance\_schema\_consumer\_events\_statements\_current=on

performance\_schema\_consumer\_events\_waits\_history=on

performance\_schema\_consumer\_events\_stages\_history=on

performance\_schema\_consumer\_events\_statements\_history=on

[复制代码](javascript:void(0);)

即在这些表的前面加上：performance\_schema\_consumer\_xxx。表setup\_consumers里面的值有个层级关系：

**global\_instrumentation** > **thread\_instrumentation** = **statements\_digest** > events\_stages\_**current** = events\_statements\_current = events\_waits\_current > events\_stages\_**history** = events\_statements\_history = events\_waits\_history > events\_stages\_**history\_long** = events\_statements\_history\_long = events\_waits\_history\_long

只有上一层次的为YES，才会继续检查该本层为YES or NO。global\_instrumentation是最高级别consumer，如果它设置为NO，则所有的consumer都会忽略。其中history和history\_long存的是current表的历史记录条数，history表记录了每个线程最近等待的10个事件，而history\_long表则记录了最近所有线程产生的10000个事件，这里的10和10000都是可以配置的。这三个表表结构相同，history和history\_long表数据都来源于current表。长度通过控制参数：

[复制代码](javascript:void(0);)

zjy@performance\_schema **11**:**10**:**03**>show variables like 'performance\_schema%history%size';

+--------------------------------------------------------+-------+

| Variable\_name | Value |

+--------------------------------------------------------+-------+

| performance\_schema\_events\_stages\_history\_long\_size | **10000** |

| performance\_schema\_events\_stages\_history\_size | **10** |

| performance\_schema\_events\_statements\_history\_long\_size | **10000** |

| performance\_schema\_events\_statements\_history\_size | **10** |

| performance\_schema\_events\_waits\_history\_long\_size | **10000** |

| performance\_schema\_events\_waits\_history\_size | **10** |

+--------------------------------------------------------+-------+

[复制代码](javascript:void(0);)

**3，setup\_instruments**：配置具体的instrument，主要包含4大类：idle、stage/xxx、statement/xxx、wait/xxx：

[复制代码](javascript:void(0);)

zjy@performance\_schema **10**:**56**:**35**>select name,count(\*) from setup\_instruments group by LEFT(name,**5**);

+---------------------------------+----------+

| name | count(\*) |

+---------------------------------+----------+

| idle | **1** |

| stage/sql/After create | **111** |

| statement/sql/select | **179** |

| wait/synch/mutex/sql/PAGE::lock | **296** |

+---------------------------------+----------+

[复制代码](javascript:void(0);)

idle表示socket空闲的时间，stage类表示语句的每个执行阶段的统计，statement类统计语句维度的信息，wait类统计各种等待事件，比如IO，mutux，spin\_lock,condition等。

**4，setup\_objects**：配置监控对象，默认对mysql，performance\_schema和information\_schema中的表都不监控，而其它DB的所有表都监控。

[复制代码](javascript:void(0);)

zjy@performance\_schema **11**:**00**:**18**>select \* from setup\_objects;

+-------------+--------------------+-------------+---------+-------+

| OBJECT\_TYPE | OBJECT\_SCHEMA | OBJECT\_NAME | ENABLED | TIMED |

+-------------+--------------------+-------------+---------+-------+

| TABLE | mysql | % | NO | NO |

| TABLE | performance\_schema | % | NO | NO |

| TABLE | information\_schema | % | NO | NO |

| TABLE | % | % | **YES** | **YES** |

+-------------+--------------------+-------------+---------+-------+

[复制代码](javascript:void(0);)

**5，setup\_timers**：配置每种类型指令的统计时间单位。MICROSECOND表示统计单位是微妙，CYCLE表示统计单位是时钟周期，时间度量与CPU的主频有关，NANOSECOND表示统计单位是纳秒。但无论采用哪种度量单位，最终统计表中统计的时间都会装换到皮秒。（1秒＝1000000000000皮秒）

[复制代码](javascript:void(0);)

zjy@performance\_schema **11**:**05**:**12**>select \* from setup\_timers;

+-----------+-------------+

| NAME | TIMER\_NAME |

+-----------+-------------+

| idle | MICROSECOND |

| wait | CYCLE |

| stage | NANOSECOND |

| statement | NANOSECOND |

+-----------+-------------+

[复制代码](javascript:void(0);)

**二：instance表**

**1，cond\_instances**：条件等待对象实例

表中记录了系统中使用的条件变量的对象，**OBJECT\_INSTANCE\_BEGIN**为对象的内存地址。

**2，file\_instances**：文件实例

表中记录了系统中打开了文件的对象，包括ibdata文件，redo文件，binlog文件，用户的表文件等，**open\_count**显示当前文件打开的数目，如果重来没有打开过，不会出现在表中。

[复制代码](javascript:void(0);)

zjy@performance\_schema **11**:**20**:**04**>select \* from file\_instances limit **2**,**5**;

+---------------------------------+--------------------------------------+------------+

| FILE\_NAME | EVENT\_NAME | **OPEN\_COUNT** |

+---------------------------------+--------------------------------------+------------+

| /var/lib/mysql/mysql/plugin.frm | wait/io/file/sql/FRM | **0** |

| /var/lib/mysql/mysql/plugin.MYI | wait/io/file/myisam/kfile | **1** |

| /var/lib/mysql/mysql/plugin.MYD | wait/io/file/myisam/dfile | **1** |

| /var/lib/mysql/ibdata1 | wait/io/file/innodb/innodb\_data\_file | **2** |

| /var/lib/mysql/ib\_logfile0 | wait/io/file/innodb/innodb\_log\_file | **2** |

+---------------------------------+--------------------------------------+------------+

[复制代码](javascript:void(0);)

**3，mutex\_instances：**互斥同步对象实例

表中记录了系统中使用互斥量对象的所有记录，其中name为：wait/synch/mutex/\*。**LOCKED\_BY\_THREAD\_ID**显示哪个线程正持有mutex，若没有线程持有，则为NULL。

**4，rwlock\_instances：** 读写锁同步对象实例

表中记录了系统中使用读写锁对象的所有记录，其中name为 wait/synch/rwlock/\*。**WRITE\_LOCKED\_BY\_THREAD\_ID**为正在持有该对象的thread\_id，若没有线程持有，则为NULL。**READ\_LOCKED\_BY\_COUNT**为记录了同时有多少个读者持有读锁。（通过 events\_waits\_current 表可以知道，哪个线程在等待锁；通过rwlock\_instances知道哪个线程持有锁。rwlock\_instances的缺陷是，只能记录持有写锁的线程，对于读锁则无能为力）。

**5，socket\_instances：**活跃会话对象实例  
表中记录了thread\_id,socket\_id,ip和port，其它表可以通过thread\_id与socket\_instance进行关联，获取IP-PORT信息，能够与应用对接起来。  
event\_name主要包含3类：  
wait/io/socket/sql/server\_unix\_socket，服务端unix监听socket  
wait/io/socket/sql/server\_tcpip\_socket，服务端tcp监听socket  
wait/io/socket/sql/client\_connection，客户端socket

**三：Wait表**

**1，events\_waits\_current**：记录了当前线程等待的事件

**2，events\_waits\_history**：记录了每个线程最近等待的10个事件

**3，events\_waits\_history\_long**：记录了最近所有线程产生的10000个事件

表结构定义如下：

[复制代码](javascript:void(0);)

CREATE TABLE `events\_waits\_current` (

`THREAD\_ID` bigint(**20**) unsigned NOT NULL COMMENT '线程ID',

`EVENT\_ID` bigint(**20**) unsigned NOT NULL COMMENT '当前线程的事件ID，和THREAD\_ID确定唯一',

`END\_EVENT\_ID` bigint(**20**) unsigned DEFAULT NULL COMMENT '当事件开始时，这一列被设置为NULL。当事件结束时，再更新为当前的事件ID',

`EVENT\_NAME` varchar(**128**) NOT NULL COMMENT '事件名称',

`SOURCE` varchar(**64**) DEFAULT NULL COMMENT '该事件产生时的源码文件',

`TIMER\_START` bigint(**20**) unsigned DEFAULT NULL COMMENT '事件开始时间（皮秒）',

`TIMER\_END` bigint(**20**) unsigned DEFAULT NULL COMMENT '事件结束结束时间（皮秒）',

`TIMER\_WAIT` bigint(**20**) unsigned DEFAULT NULL COMMENT '事件等待时间（皮秒）',

`SPINS` int(**10**) unsigned DEFAULT NULL COMMENT '',

`OBJECT\_SCHEMA` varchar(**64**) DEFAULT NULL COMMENT '库名',

`OBJECT\_NAME` varchar(**512**) DEFAULT NULL COMMENT '文件名、表名、IP:SOCK值',

`OBJECT\_TYPE` varchar(**64**) DEFAULT NULL COMMENT 'FILE、TABLE、TEMPORARY TABLE',

`INDEX\_NAME` varchar(**64**) DEFAULT NULL COMMENT '索引名',

`OBJECT\_INSTANCE\_BEGIN` bigint(**20**) unsigned NOT NULL COMMENT '内存地址',

`NESTING\_EVENT\_ID` bigint(**20**) unsigned DEFAULT NULL COMMENT '该事件对应的父事件ID',

`NESTING\_EVENT\_TYPE` enum('STATEMENT','STAGE','WAIT') DEFAULT NULL COMMENT '父事件类型(STATEMENT, STAGE, WAIT)',

`OPERATION` varchar(**32**) NOT NULL COMMENT '操作类型（lock, read, write）',

`NUMBER\_OF\_BYTES` bigint(**20**) DEFAULT NULL COMMENT '',

`FLAGS` int(**10**) unsigned DEFAULT NULL COMMENT '标记'

) ENGINE=PERFORMANCE\_SCHEMA DEFAULT CHARSET=utf8

[复制代码](javascript:void(0);)

**四：Stage 表**

**1，events\_stages\_current**：记录了当前线程所处的执行阶段

**2，events\_stages\_history**：记录了当前线程所处的执行阶段10条历史记录

**3，events\_stages\_history\_long**：记录了当前线程所处的执行阶段10000条历史记录

表结构定义如下：

[复制代码](javascript:void(0);)

CREATE TABLE `events\_stages\_current` (

`THREAD\_ID` bigint(**20**) unsigned NOT NULL COMMENT '线程ID',

`EVENT\_ID` bigint(**20**) unsigned NOT NULL COMMENT '事件ID',

`END\_EVENT\_ID` bigint(**20**) unsigned DEFAULT NULL COMMENT '结束事件ID',

`EVENT\_NAME` varchar(**128**) NOT NULL COMMENT '事件名称',

`SOURCE` varchar(**64**) DEFAULT NULL COMMENT '源码位置',

`TIMER\_START` bigint(**20**) unsigned DEFAULT NULL COMMENT '事件开始时间（皮秒）',

`TIMER\_END` bigint(**20**) unsigned DEFAULT NULL COMMENT '事件结束结束时间（皮秒）',

`TIMER\_WAIT` bigint(**20**) unsigned DEFAULT NULL COMMENT '事件等待时间（皮秒）',

`NESTING\_EVENT\_ID` bigint(**20**) unsigned DEFAULT NULL COMMENT '该事件对应的父事件ID',

`NESTING\_EVENT\_TYPE` enum('STATEMENT','STAGE','WAIT') DEFAULT NULL COMMENT '父事件类型(STATEMENT, STAGE, WAIT)'

) ENGINE=PERFORMANCE\_SCHEMA DEFAULT CHARSET=utf8

[复制代码](javascript:void(0);)

**五：Statement 表**

**1，events\_statements\_current**：通过 thread\_id+event\_id可以唯一确定一条记录。Statments表只记录最顶层的请求，SQL语句或是COMMAND，每条语句一行。event\_name形式为statement/sql/\*，或statement/com/\*

**2，events\_statements\_history**

**3，events\_statements\_history\_long**

表结构定义如下：

[复制代码](javascript:void(0);)

CREATE TABLE `events\_statements\_current` (

`THREAD\_ID` bigint(**20**) unsigned NOT NULL COMMENT '线程ID',

`EVENT\_ID` bigint(**20**) unsigned NOT NULL COMMENT '事件ID',

`END\_EVENT\_ID` bigint(**20**) unsigned DEFAULT NULL COMMENT '结束事件ID',

`EVENT\_NAME` varchar(**128**) NOT NULL COMMENT '事件名称',

`SOURCE` varchar(**64**) DEFAULT NULL COMMENT '源码位置',

`TIMER\_START` bigint(**20**) unsigned DEFAULT NULL COMMENT '事件开始时间（皮秒）',

`TIMER\_END` bigint(**20**) unsigned DEFAULT NULL COMMENT '事件结束结束时间（皮秒）',

`TIMER\_WAIT` bigint(**20**) unsigned DEFAULT NULL COMMENT '事件等待时间（皮秒）',

`LOCK\_TIME` bigint(**20**) unsigned NOT NULL COMMENT '锁时间',

`SQL\_TEXT` longtext COMMENT '记录SQL语句',

`DIGEST` varchar(**32**) DEFAULT NULL COMMENT '对SQL\_TEXT做MD5产生的32位字符串',

`DIGEST\_TEXT` longtext COMMENT '将语句中值部分用问号代替，用于SQL语句归类',

`CURRENT\_SCHEMA` varchar(**64**) DEFAULT NULL COMMENT '默认的数据库名',

`OBJECT\_TYPE` varchar(**64**) DEFAULT NULL COMMENT '保留字段',

`OBJECT\_SCHEMA` varchar(**64**) DEFAULT NULL COMMENT '保留字段',

`OBJECT\_NAME` varchar(**64**) DEFAULT NULL COMMENT '保留字段',

`OBJECT\_INSTANCE\_BEGIN` bigint(**20**) unsigned DEFAULT NULL COMMENT '内存地址',

`MYSQL\_ERRNO` int(**11**) DEFAULT NULL COMMENT '',

`RETURNED\_SQLSTATE` varchar(**5**) DEFAULT NULL COMMENT '',

`MESSAGE\_TEXT` varchar(**128**) DEFAULT NULL COMMENT '信息',

`ERRORS` bigint(**20**) unsigned NOT NULL COMMENT '错误数目',

`WARNINGS` bigint(**20**) unsigned NOT NULL COMMENT '警告数目',

`ROWS\_AFFECTED` bigint(**20**) unsigned NOT NULL COMMENT '影响的数目',

`ROWS\_SENT` bigint(**20**) unsigned NOT NULL COMMENT '返回的记录数',

`ROWS\_EXAMINED` bigint(**20**) unsigned NOT NULL COMMENT '读取扫描的记录数目',

`CREATED\_TMP\_DISK\_TABLES` bigint(**20**) unsigned NOT NULL COMMENT '创建磁盘临时表数目',

`CREATED\_TMP\_TABLES` bigint(**20**) unsigned NOT NULL COMMENT '创建临时表数目',

`SELECT\_FULL\_JOIN` bigint(**20**) unsigned NOT NULL COMMENT 'join时，第一个表为全表扫描的数目',

`SELECT\_FULL\_RANGE\_JOIN` bigint(**20**) unsigned NOT NULL COMMENT '引用表采用range方式扫描的数目',

`SELECT\_RANGE` bigint(**20**) unsigned NOT NULL COMMENT 'join时，第一个表采用range方式扫描的数目',

`SELECT\_RANGE\_CHECK` bigint(**20**) unsigned NOT NULL COMMENT '',

`SELECT\_SCAN` bigint(**20**) unsigned NOT NULL COMMENT 'join时，第一个表位全表扫描的数目',

`SORT\_MERGE\_PASSES` bigint(**20**) unsigned NOT NULL COMMENT '',

`SORT\_RANGE` bigint(**20**) unsigned NOT NULL COMMENT '范围排序数目',

`SORT\_ROWS` bigint(**20**) unsigned NOT NULL COMMENT '排序的记录数目',

`SORT\_SCAN` bigint(**20**) unsigned NOT NULL COMMENT '全表排序数目',

`NO\_INDEX\_USED` bigint(**20**) unsigned NOT NULL COMMENT '没有使用索引数目',

`NO\_GOOD\_INDEX\_USED` bigint(**20**) unsigned NOT NULL COMMENT '',

`NESTING\_EVENT\_ID` bigint(**20**) unsigned DEFAULT NULL COMMENT '该事件对应的父事件ID',

`NESTING\_EVENT\_TYPE` enum('STATEMENT','STAGE','WAIT') DEFAULT NULL COMMENT '父事件类型(STATEMENT, STAGE, WAIT)'

) ENGINE=PERFORMANCE\_SCHEMA DEFAULT CHARSET=utf8

[复制代码](javascript:void(0);)

**六：Connection 表**

**1，users**：记录用户连接数信息

**2，hosts**：记录了主机连接数信息

**3，accounts**：记录了用户主机连接数信息

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

zjy@performance\_schema **12**:**03**:**27**>select \* from users;

+------------------+---------------------+-------------------+

| USER | CURRENT\_CONNECTIONS | TOTAL\_CONNECTIONS |

+------------------+---------------------+-------------------+

| debian-sys-maint | **0** | **36** |

| zjy | **1** | **22285** |

| dchat\_php | **0** | **37864** |

| dxyslave | **2** | **9** |

| nagios | **0** | **10770** |

| dchat\_data | **140** | **2233023** |

| NULL | **0** | **15866** |

| dchat\_api | **160** | **2754212** |

| mha\_data | **1** | **36** |

| backup | **0** | **15** |

| cacti | **0** | **4312** |

| kol | **10** | **172414** |

+------------------+---------------------+-------------------+

**12** rows in set (**0.00** sec)

zjy@performance\_schema **12**:**03**:**34**>select \* from hosts;

+-----------------+---------------------+-------------------+

| HOST | CURRENT\_CONNECTIONS | TOTAL\_CONNECTIONS |

+-----------------+---------------------+-------------------+

| **192.168**.**100.218** | **150** | **2499422** |

| **192.168**.**100.240** | **10** | **172429** |

| **192.168**.**100.139** | **0** | **698** |

| **192.168**.**100.21** | **0** | **2** |

| **192.168**.**100.220** | **150** | **2526136** |

| **192.168**.**100.25** | **1** | **7** |

| NULL | **0** | **15867** |

| **192.168**.**100.241** | **0** | **21558** |

| **192.168**.**100.191** | **1** | **34** |

| localhost | **0** | **10807** |

| **192.168**.**100.118** | **1** | **2** |

| **192.168**.**100.251** | **0** | **4312** |

| **192.168**.**100.23** | **1** | **31** |

| **192.168**.**100.193** | **0** | **15** |

+-----------------+---------------------+-------------------+

**14** rows in set (**0.01** sec)

zjy@performance\_schema **12**:**05**:**21**>select \* from accounts;

+------------------+-----------------+---------------------+-------------------+

| USER | HOST | CURRENT\_CONNECTIONS | TOTAL\_CONNECTIONS |

+------------------+-----------------+---------------------+-------------------+

| cacti | **192.168**.**100.251** | **0** | **4313** |

| debian-sys-maint | localhost | **0** | **36** |

| backup | **192.168**.**100.193** | **0** | **15** |

| dchat\_api | **192.168**.**100.220** | **80** | **1382585** |

| dchat\_php | **192.168**.**100.220** | **0** | **20292** |

| zjy | **192.168**.**100.139** | **0** | **698** |

| zjy | **192.168**.**100.241** | **0** | **21558** |

| mha\_data | **192.168**.**100.191** | **1** | **34** |

| dxyslave | **192.168**.**100.118** | **1** | **2** |

| kol | **192.168**.**100.240** | **10** | **172431** |

| dxyslave | **192.168**.**100.25** | **1** | **7** |

| dchat\_data | **192.168**.**100.218** | **70** | **1109974** |

| zjy | **192.168**.**100.23** | **1** | **31** |

| dchat\_php | **192.168**.**100.218** | **0** | **17572** |

| dchat\_data | **192.168**.**100.220** | **70** | **1123306** |

| NULL | NULL | **0** | **15868** |

| mha\_data | **192.168**.**100.21** | **0** | **2** |

| dchat\_api | **192.168**.**100.218** | **80** | **1371918** |

| nagios | localhost | **0** | **10771** |

+------------------+-----------------+---------------------+-------------------+

View Code

**七：Summary 表： Summary表聚集了各个维度的统计信息包括表维度，索引维度，会话维度，语句维度和锁维度的统计信息**

**1，events\_waits\_summary\_global\_by\_event\_name**：按等待事件类型聚合，每个事件一条记录

[复制代码](javascript:void(0);)

CREATE TABLE `events\_waits\_summary\_global\_by\_event\_name` (

`EVENT\_NAME` varchar(**128**) NOT NULL COMMENT '事件名称',

`COUNT\_STAR` bigint(**20**) unsigned NOT NULL COMMENT '事件计数',

`SUM\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '总的等待时间',

`MIN\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '最小等待时间',

`AVG\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '平均等待时间',

`MAX\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '最大等待时间'

) ENGINE=PERFORMANCE\_SCHEMA DEFAULT CHARSET=utf8

[复制代码](javascript:void(0);)

**2，events\_waits\_summary\_by\_instance**：按等待事件对象聚合，同一种等待事件，可能有多个实例，每个实例有不同的内存地址，因此  
event\_name+object\_instance\_begin唯一确定一条记录。

[复制代码](javascript:void(0);)

CREATE TABLE `events\_waits\_summary\_by\_instance` (

`EVENT\_NAME` varchar(**128**) NOT NULL COMMENT '事件名称',

`OBJECT\_INSTANCE\_BEGIN` bigint(**20**) unsigned NOT NULL COMMENT '内存地址',

`COUNT\_STAR` bigint(**20**) unsigned NOT NULL COMMENT '事件计数',

`SUM\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '总的等待时间',

`MIN\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '最小等待时间',

`AVG\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '平均等待时间',

`MAX\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '最大等待时间'

) ENGINE=PERFORMANCE\_SCHEMA DEFAULT CHARSET=utf8

[复制代码](javascript:void(0);)

**3，events\_waits\_summary\_by\_thread\_by\_event\_name**：按每个线程和事件来统计，thread\_id+event\_name唯一确定一条记录。

[复制代码](javascript:void(0);)

CREATE TABLE `events\_waits\_summary\_by\_thread\_by\_event\_name` (

`THREAD\_ID` bigint(**20**) unsigned NOT NULL COMMENT '线程ID',

`EVENT\_NAME` varchar(**128**) NOT NULL COMMENT '事件名称',

`COUNT\_STAR` bigint(**20**) unsigned NOT NULL COMMENT '事件计数',

`SUM\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '总的等待时间',

`MIN\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '最小等待时间',

`AVG\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '平均等待时间',

`MAX\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '最大等待时间'

) ENGINE=PERFORMANCE\_SCHEMA DEFAULT CHARSET=utf8

[复制代码](javascript:void(0);)

**4，events\_stages\_summary\_global\_by\_event\_name**：按事件阶段类型聚合，每个事件一条记录，表结构同上。

**5，events\_stages\_summary\_by\_thread\_by\_event\_name**：按每个线程和事件来阶段统计，表结构同上。

**6，events\_statements\_summary\_by\_digest**：按照事件的语句进行聚合。

[复制代码](javascript:void(0);)

CREATE TABLE `events\_statements\_summary\_by\_digest` (

`SCHEMA\_NAME` varchar(**64**) DEFAULT NULL COMMENT '库名',

`DIGEST` varchar(**32**) DEFAULT NULL COMMENT '对SQL\_TEXT做MD5产生的32位字符串。如果为consumer表中没有打开statement\_digest选项，则为NULL',

`DIGEST\_TEXT` longtext COMMENT '将语句中值部分用问号代替，用于SQL语句归类。如果为consumer表中没有打开statement\_digest选项，则为NULL。',

`COUNT\_STAR` bigint(**20**) unsigned NOT NULL COMMENT '事件计数',

`SUM\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '总的等待时间',

`MIN\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '最小等待时间',

`AVG\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '平均等待时间',

`MAX\_TIMER\_WAIT` bigint(**20**) unsigned NOT NULL COMMENT '最大等待时间',

`SUM\_LOCK\_TIME` bigint(**20**) unsigned NOT NULL COMMENT '锁时间总时长',

`SUM\_ERRORS` bigint(**20**) unsigned NOT NULL COMMENT '错误数的总',

`SUM\_WARNINGS` bigint(**20**) unsigned NOT NULL COMMENT '警告的总数',

`SUM\_ROWS\_AFFECTED` bigint(**20**) unsigned NOT NULL COMMENT '影响的总数目',

`SUM\_ROWS\_SENT` bigint(**20**) unsigned NOT NULL COMMENT '返回总数目',

`SUM\_ROWS\_EXAMINED` bigint(**20**) unsigned NOT NULL COMMENT '总的扫描的数目',

`SUM\_CREATED\_TMP\_DISK\_TABLES` bigint(**20**) unsigned NOT NULL COMMENT '创建磁盘临时表的总数目',

`SUM\_CREATED\_TMP\_TABLES` bigint(**20**) unsigned NOT NULL COMMENT '创建临时表的总数目',

`SUM\_SELECT\_FULL\_JOIN` bigint(**20**) unsigned NOT NULL COMMENT '第一个表全表扫描的总数目',

`SUM\_SELECT\_FULL\_RANGE\_JOIN` bigint(**20**) unsigned NOT NULL COMMENT '总的采用range方式扫描的数目',

`SUM\_SELECT\_RANGE` bigint(**20**) unsigned NOT NULL COMMENT '第一个表采用range方式扫描的总数目',

`SUM\_SELECT\_RANGE\_CHECK` bigint(**20**) unsigned NOT NULL COMMENT '',

`SUM\_SELECT\_SCAN` bigint(**20**) unsigned NOT NULL COMMENT '第一个表位全表扫描的总数目',

`SUM\_SORT\_MERGE\_PASSES` bigint(**20**) unsigned NOT NULL COMMENT '',

`SUM\_SORT\_RANGE` bigint(**20**) unsigned NOT NULL COMMENT '范围排序总数',

`SUM\_SORT\_ROWS` bigint(**20**) unsigned NOT NULL COMMENT '排序的记录总数目',

`SUM\_SORT\_SCAN` bigint(**20**) unsigned NOT NULL COMMENT '第一个表排序扫描总数目',

`SUM\_NO\_INDEX\_USED` bigint(**20**) unsigned NOT NULL COMMENT '没有使用索引总数',

`SUM\_NO\_GOOD\_INDEX\_USED` bigint(**20**) unsigned NOT NULL COMMENT '',

`FIRST\_SEEN` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00' COMMENT '第一次执行时间',

`LAST\_SEEN` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00' COMMENT '最后一次执行时间'

) ENGINE=PERFORMANCE\_SCHEMA DEFAULT CHARSET=utf8

[复制代码](javascript:void(0);)

**7，events\_statements\_summary\_global\_by\_event\_name**：按照事件的语句进行聚合。表结构同上。

**8，events\_statements\_summary\_by\_thread\_by\_event\_name**：按照线程和事件的语句进行聚合，表结构同上。

**9，file\_summary\_by\_instance**：按事件类型统计（**物理IO维度**）

**10，file\_summary\_by\_event\_name**：具体文件统计（**物理IO维度**）

9和10一起说明：

统计IO操作：COUNT\_STAR，SUM\_TIMER\_WAIT,MIN\_TIMER\_WAIT,AVG\_TIMER\_WAIT,MAX\_TIMER\_WAIT

统计读      ：COUNT\_READ,SUM\_TIMER\_READ,MIN\_TIMER\_READ,AVG\_TIMER\_READ,MAX\_TIMER\_READ, SUM\_NUMBER\_OF\_BYTES\_READ

统计写      ：COUNT\_WRITE,SUM\_TIMER\_WRITE,MIN\_TIMER\_WRITE,AVG\_TIMER\_WRITE,MAX\_TIMER\_WRITE, SUM\_NUMBER\_OF\_BYTES\_WRITE

统计其他IO事件，比如create，delete，open，close等：COUNT\_MISC,SUM\_TIMER\_MISC,MIN\_TIMER\_MISC,AVG\_TIMER\_MISC,MAX\_TIMER\_MISC

**11，table\_io\_waits\_summary\_by\_table**：根据wait/io/table/sql/handler，聚合每个表的I/O操作（**逻辑IO纬度**）

统计IO操作：COUNT\_STAR,SUM\_TIMER\_WAIT,MIN\_TIMER\_WAIT,AVG\_TIMER\_WAIT,MAX\_TIMER\_WAIT

统计读      ：COUNT\_READ,SUM\_TIMER\_READ,MIN\_TIMER\_READ,AVG\_TIMER\_READ,MAX\_TIMER\_READ

              ：COUNT\_FETCH,SUM\_TIMER\_FETCH,MIN\_TIMER\_FETCH,AVG\_TIMER\_FETCH, MAX\_TIMER\_FETCH

统计写      ：COUNT\_WRITE,SUM\_TIMER\_WRITE,MIN\_TIMER\_WRITE,AVG\_TIMER\_WRITE,MAX\_TIMER\_WRITE

INSERT统计，相应的还有DELETE和UPDATE统计：COUNT\_INSERT,SUM\_TIMER\_INSERT,MIN\_TIMER\_INSERT,AVG\_TIMER\_INSERT,MAX\_TIMER\_INSERT

**12，table\_io\_waits\_summary\_by\_index\_usage：**与table\_io\_waits\_summary\_by\_table类似，按索引维度统计

**13，table\_lock\_waits\_summary\_by\_table**：聚合了表锁等待事件，包括internal lock 和 external lock

internal lock通过SQL层函数thr\_lock调用，OPERATION值为：  
read normal、read with shared locks、read high priority、read no insert、write allow write、write concurrent insert、write delayed、write low priority、write normal  
external lock则通过接口函数handler::external\_lock调用存储引擎层，OPERATION列的值为：read external、write external

**14，Connection Summaries表**：account、user、host

events\_waits\_summary\_by\_account\_by\_event\_name  
events\_waits\_summary\_by\_user\_by\_event\_name  
events\_waits\_summary\_by\_host\_by\_event\_name   
events\_stages\_summary\_by\_account\_by\_event\_name  
events\_stages\_summary\_by\_user\_by\_event\_name  
events\_stages\_summary\_by\_host\_by\_event\_name   
events\_statements\_summary\_by\_account\_by\_event\_name  
events\_statements\_summary\_by\_user\_by\_event\_name  
events\_statements\_summary\_by\_host\_by\_event\_name

**15，socket\_summary\_by\_instance、socket\_summary\_by\_event\_name**：socket聚合统计表。

**八：其他相关表**

**1，performance\_timers**：系统支持的统计时间单位

**2，threads**：监视服务端的当前运行的线程

**统计应用：**

**关于SQL维度的统计信息主要集中在events\_statements\_summary\_by\_digest表中，通过将SQL语句抽象出digest，可以统计某类SQL语句在各个维度的统计信息**

**1，哪个SQL执行最多：**

[复制代码](javascript:void(0);)

zjy@performance\_schema **11**:**36**:**22**>**SELECT SCHEMA\_NAME,DIGEST\_TEXT,COUNT\_STAR,SUM\_ROWS\_SENT,SUM\_ROWS\_EXAMINED,FIRST\_SEEN,LAST\_SEEN FROM events\_statements\_summary\_by\_digest ORDER BY COUNT\_STAR desc LIMIT 1\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **1**. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**SCHEMA\_NAME**: dchat

**DIGEST\_TEXT**: SELECT ...

**COUNT\_STAR**: **1161210102**

SUM\_ROWS\_SENT: **1161207842**

SUM\_ROWS\_EXAMINED: **0**

**FIRST\_SEEN**: **2016**-**02**-**17** **00**:**36**:**46**

**LAST\_SEEN**: **2016**-**03**-**07** **11**:**36**:**29**

[复制代码](javascript:void(0);)

各个字段的注释可以看上面的表结构说明：从2月17号到3月7号该SQL执行了1161210102次。

**2，哪个SQL平均响应时间最多：**

[复制代码](javascript:void(0);)

zjy@performance\_schema **11**:**36**:**28**>**SELECT SCHEMA\_NAME,DIGEST\_TEXT,COUNT\_STAR,AVG\_TIMER\_WAIT,SUM\_ROWS\_SENT,SUM\_ROWS\_EXAMINED,FIRST\_SEEN,LAST\_SEEN FROM events\_statements\_summary\_by\_digest ORDER BY AVG\_TIMER\_WAIT desc LIMIT 1\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **1**. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**SCHEMA\_NAME**: dchat

**DIGEST\_TEXT**: SELECT ...

COUNT\_STAR: **1**

**AVG\_TIMER\_WAIT**: **273238183964000**

SUM\_ROWS\_SENT: **50208**

SUM\_ROWS\_EXAMINED: **5565651**

**FIRST\_SEEN**: **2016**-**02**-**22** **13**:**27**:**33**

**LAST\_SEEN**: **2016**-**02**-**22** **13**:**27**:**33**

[复制代码](javascript:void(0);)

各个字段的注释可以看上面的表结构说明：从2月17号到3月7号该SQL平均响应时间273238183964000皮秒（1000000000000皮秒=1秒）

**3，哪个SQL扫描的行数最多：**

SUM\_ROWS\_EXAMINED

**4，哪个SQL使用的临时表最多：**

SUM\_CREATED\_TMP\_DISK\_TABLES、SUM\_CREATED\_TMP\_TABLES

**5，哪个SQL返回的结果集最多：**

SUM\_ROWS\_SENT

**6，哪个SQL排序数最多：**

SUM\_SORT\_ROWS

通过上述指标我们可以间接获得某类SQL的逻辑IO(SUM\_ROWS\_EXAMINED)，CPU消耗(SUM\_SORT\_ROWS)，网络带宽(SUM\_ROWS\_SENT)的对比。

通过**file\_summary\_by\_instance**表，可以获得系统运行到现在，哪个文件(表)物理IO最多，这可能意味着这个表经常需要访问磁盘IO。

**7，哪个表、文件逻辑IO最多（热数据）：**

[复制代码](javascript:void(0);)

zjy@performance\_schema **12**:**16**:**18**>**SELECT FILE\_NAME,EVENT\_NAME,COUNT\_READ,SUM\_NUMBER\_OF\_BYTES\_READ,COUNT\_WRITE,SUM\_NUMBER\_OF\_BYTES\_WRITE FROM file\_summary\_by\_instance ORDER BY SUM\_NUMBER\_OF\_BYTES\_READ+SUM\_NUMBER\_OF\_BYTES\_WRITE DESC LIMIT 2\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **1**. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FILE\_NAME: /var/lib/mysql/**ibdata1 #文件**

EVENT\_NAME: wait/io/file/innodb/innodb\_data\_file

COUNT\_READ: **544**

SUM\_NUMBER\_OF\_BYTES\_READ: **10977280**

COUNT\_WRITE: **3700729**

SUM\_NUMBER\_OF\_BYTES\_WRITE: **1433734217728**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **2**. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FILE\_NAME: /var/lib/mysql/dchat/**fans.ibd #表**

EVENT\_NAME: wait/io/file/innodb/innodb\_data\_file

COUNT\_READ: **9370680**

SUM\_NUMBER\_OF\_BYTES\_READ: **153529188352**

COUNT\_WRITE: **67576376**

SUM\_NUMBER\_OF\_BYTES\_WRITE: **1107815432192**

[复制代码](javascript:void(0);)

**8，哪个索引使用最多：**

[复制代码](javascript:void(0);)

zjy@performance\_schema **12**:**18**:**42**>**SELECT OBJECT\_NAME, INDEX\_NAME, COUNT\_FETCH, COUNT\_INSERT, COUNT\_UPDATE, COUNT\_DELETE FROM table\_io\_waits\_summary\_by\_index\_usage ORDER BY SUM\_TIMER\_WAIT DESC limit 1;**

+-------------+------------+-------------+--------------+--------------+--------------+

| OBJECT\_NAME | INDEX\_NAME | COUNT\_FETCH | COUNT\_INSERT | COUNT\_UPDATE | COUNT\_DELETE |

+-------------+------------+-------------+--------------+--------------+--------------+

| **fans** | **PRIMARY** | **29002695158** | **0** | **296373434** | **0** |

+-------------+------------+-------------+--------------+--------------+--------------+

**1** row in set (**0.29** sec)

[复制代码](javascript:void(0);)

通过**table\_io\_waits\_summary\_by\_index\_usage**表，可以获得系统运行到现在，哪个表的具体哪个索引(包括主键索引，二级索引)使用最多。

**9，哪个索引没有使用过：**

zjy@performance\_schema **12**:**23**:**22**>**SELECT OBJECT\_SCHEMA, OBJECT\_NAME, INDEX\_NAME FROM table\_io\_waits\_summary\_by\_index\_usage WHERE INDEX\_NAME IS NOT NULL AND COUNT\_STAR = 0 AND OBJECT\_SCHEMA <> 'mysql' ORDER BY OBJECT\_SCHEMA,OBJECT\_NAME;**

**10，哪个等待事件消耗的时间最多：**

zjy@performance\_schema **12**:**25**:**22**>**SELECT EVENT\_NAME, COUNT\_STAR, SUM\_TIMER\_WAIT, AVG\_TIMER\_WAIT FROM events\_waits\_summary\_global\_by\_event\_name WHERE event\_name != 'idle' ORDER BY SUM\_TIMER\_WAIT DESC LIMIT 1;**

**11，类似profiling功能：**

分析具体某条SQL，该SQL在执行各个阶段的时间消耗，通过events\_statements\_xxx表和events\_stages\_xxx表，就可以达到目的。两个表通过event\_id与nesting\_event\_id关联，stages表的nesting\_event\_id为对应statements表的event\_id；针对每个stage可能出现的锁等待，一个stage会对应一个或多个wait，通过stage\_xxx表的event\_id字段与waits\_xxx表的nesting\_event\_id进行关联。如：

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

比如分析包含count(\*)的某条SQL语句，具体如下：

SELECT

EVENT\_ID,

sql\_text

FROM events\_statements\_history

WHERE sql\_text LIKE '%count(\*)%';

+----------+--------------------------------------+

| EVENT\_ID | sql\_text |

+----------+--------------------------------------+

| **1690** | select count(\*) from chuck.test\_slow |

+----------+--------------------------------------+

首先得到了语句的event\_id为1690，通过查找events\_stages\_xxx中nesting\_event\_id为1690的记录，可以达到目的。

a.查看每个阶段的时间消耗：

SELECT

event\_id,

EVENT\_NAME,

SOURCE,

TIMER\_END - TIMER\_START

FROM events\_stages\_history\_long

WHERE NESTING\_EVENT\_ID = **1690**;

+----------+--------------------------------+----------------------+-----------------------+

| event\_id | EVENT\_NAME | SOURCE | TIMER\_END-TIMER\_START |

+----------+--------------------------------+----------------------+-----------------------+

| **1691** | stage/sql/init | mysqld.cc:**990** | **316945000** |

| **1693** | stage/sql/checking permissions | sql\_parse.cc:**5776** | **26774000** |

| **1695** | stage/sql/Opening tables | sql\_base.cc:**4970** | **41436934000** |

| **2638** | stage/sql/init | sql\_select.cc:**1050** | **85757000** |

| **2639** | stage/sql/System lock | lock.cc:**303** | **40017000** |

| **2643** | stage/sql/optimizing | sql\_optimizer.cc:**138** | **38562000** |

| **2644** | stage/sql/statistics | sql\_optimizer.cc:**362** | **52845000** |

| **2645** | stage/sql/preparing | sql\_optimizer.cc:**485** | **53196000** |

| **2646** | stage/sql/executing | sql\_executor.cc:**112** | **3153000** |

| **2647** | stage/sql/Sending data | sql\_executor.cc:**192** | **7369072089000** |

| **4304138** | stage/sql/end | sql\_select.cc:**1105** | **19920000** |

| **4304139** | stage/sql/query end | sql\_parse.cc:**5463** | **44721000** |

| **4304145** | stage/sql/closing tables | sql\_parse.cc:**5524** | **61723000** |

| **4304152** | stage/sql/freeing items | sql\_parse.cc:**6838** | **455678000** |

| **4304155** | stage/sql/logging slow query | sql\_parse.cc:**2258** | **83348000** |

| **4304159** | stage/sql/cleaning up | sql\_parse.cc:**2163** | **4433000** |

+----------+--------------------------------+----------------------+-----------------------+

通过间接关联，我们能分析得到SQL语句在每个阶段的时间消耗，时间单位以皮秒表示。这里展示的结果很类似profiling功能，有了performance schema，就不再需要profiling这个功能了。另外需要注意的是，由于默认情况下events\_stages\_history表中只为每个连接记录了最近10条记录，为了确保获取所有记录，需要访问events\_stages\_history\_long表

b.查看某个阶段的锁等待情况

针对每个stage可能出现的锁等待，一个stage会对应一个或多个wait，events\_waits\_history\_long这个表容易爆满[默认阀值10000]。由于select count(\*)需要IO(逻辑IO或者物理IO)，所以在stage/sql/Sending data阶段会有io等待的统计。通过stage\_xxx表的event\_id字段与waits\_xxx表的nesting\_event\_id进行关联。

SELECT

event\_id,

event\_name,

source,

timer\_wait,

object\_name,

index\_name,

operation,

nesting\_event\_id

FROM events\_waits\_history\_long

WHERE nesting\_event\_id = **2647**;

+----------+---------------------------+-----------------+------------+-------------+------------+-----------+------------------+

| event\_id | event\_name | source | timer\_wait | object\_name | index\_name | operation | nesting\_event\_id |

+----------+---------------------------+-----------------+------------+-------------+------------+-----------+------------------+

| **190607** | wait/io/table/sql/handler | handler.cc:**2842** | **1845888** | test\_slow | idx\_c1 | fetch | **2647** |

| **190608** | wait/io/table/sql/handler | handler.cc:**2842** | **1955328** | test\_slow | idx\_c1 | fetch | **2647** |

| **190609** | wait/io/table/sql/handler | handler.cc:**2842** | **1929792** | test\_slow | idx\_c1 | fetch | **2647** |

| **190610** | wait/io/table/sql/handler | handler.cc:**2842** | **1869600** | test\_slow | idx\_c1 | fetch | **2647** |

| **190611** | wait/io/table/sql/handler | handler.cc:**2842** | **1922496** | test\_slow | idx\_c1 | fetch | **2647** |

+----------+---------------------------+-----------------+------------+-------------+------------+-----------+------------------+

通过上面的实验，我们知道了statement,stage,wait的三级结构，通过nesting\_event\_id进行关联，它表示某个事件的父event\_id。

(**2**).模拟innodb行锁等待的例子

会话A执行语句update test\_icp set y=y+**1** where x=**1**(x为primary key)，不commit；会话B执行同样的语句update test\_icp set y=y+**1** where x=**1**，会话B堵塞，并最终报错。通过连接连接查询events\_statements\_history\_long和events\_stages\_history\_long，可以看到在updating阶段花了大约60s的时间。这主要因为实例上的innodb\_lock\_wait\_timeout设置为60，等待60s后超时报错了。

SELECT

statement.EVENT\_ID,

stages.event\_id,

statement.sql\_text,

stages.event\_name,

stages.timer\_wait

FROM events\_statements\_history\_long statement

join events\_stages\_history\_long stages

on statement.event\_id=stages.nesting\_event\_id

WHERE statement.sql\_text = 'update test\_icp set y=y+1 where x=1';

+----------+----------+-------------------------------------+--------------------------------+----------------+

| EVENT\_ID | event\_id | sql\_text | event\_name | timer\_wait |

+----------+----------+-------------------------------------+--------------------------------+----------------+

| **5816** | **5817** | update test\_icp set y=y+**1** where x=**1** | stage/sql/init | **195543000** |

| **5816** | **5819** | update test\_icp set y=y+**1** where x=**1** | stage/sql/checking permissions | **22730000** |

| **5816** | **5821** | update test\_icp set y=y+**1** where x=**1** | stage/sql/Opening tables | **66079000** |

| **5816** | **5827** | update test\_icp set y=y+**1** where x=**1** | stage/sql/init | **89116000** |

| **5816** | **5828** | update test\_icp set y=y+**1** where x=**1** | stage/sql/System lock | **218744000** |

| **5816** | **5832** | update test\_icp set y=y+**1** where x=**1** | stage/sql/updating | **6001362045000** |

| **5816** | **5968** | update test\_icp set y=y+**1** where x=**1** | stage/sql/end | **10435000** |

| **5816** | **5969** | update test\_icp set y=y+**1** where x=**1** | stage/sql/query end | **85979000** |

| **5816** | **5983** | update test\_icp set y=y+**1** where x=**1** | stage/sql/closing tables | **56562000** |

| **5816** | **5990** | update test\_icp set y=y+**1** where x=**1** | stage/sql/freeing items | **83563000** |

| **5816** | **5992** | update test\_icp set y=y+**1** where x=**1** | stage/sql/cleaning up | **4589000** |

+----------+----------+-------------------------------------+--------------------------------+----------------+

查看wait事件：

SELECT

event\_id,

event\_name,

source,

timer\_wait,

object\_name,

index\_name,

operation,

nesting\_event\_id

FROM events\_waits\_history\_long

WHERE nesting\_event\_id = **5832**;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **1**. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

event\_id: **5832**

event\_name: wait/io/table/sql/handler

source: handler.cc:**2782**

timer\_wait: **6005946156624**

object\_name: test\_icp

index\_name: PRIMARY

operation: fetch

从结果来看，waits表中记录了一个fetch等待事件，但并没有更细的innodb行锁等待事件统计。

(**3**).模拟MDL锁等待的例子

会话A执行一个大查询select count(\*) from test\_slow，会话B执行表结构变更alter table test\_slow modify c2 varchar(**152**);通过如下语句可以得到alter语句的执行过程，重点关注“stage/sql/Waiting for table metadata lock”阶段。

SELECT

statement.EVENT\_ID,

stages.event\_id,

statement.sql\_text,

stages.event\_name as stage\_name,

stages.timer\_wait as stage\_time

FROM events\_statements\_history\_long statement

left join events\_stages\_history\_long stages

on statement.event\_id=stages.nesting\_event\_id

WHERE statement.sql\_text = 'alter table test\_slow modify c2 varchar(152)';

+-----------+-----------+----------------------------------------------+----------------------------------------------------+---------------+

| EVENT\_ID | event\_id | sql\_text | stage\_name | stage\_time |

+-----------+-----------+----------------------------------------------+----------------------------------------------------+---------------+

| **326526744** | **326526745** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/init | **216662000** |

| **326526744** | **326526747** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/checking permissions | **18183000** |

| **326526744** | **326526748** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/checking permissions | **10294000** |

| **326526744** | **326526750** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/init | **4783000** |

| **326526744** | **326526751** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/Opening tables | **140172000** |

| **326526744** | **326526760** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/setup | **157643000** |

| **326526744** | **326526769** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/creating table | **8723217000** |

| **326526744** | **326526803** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/After create | **257332000** |

| **326526744** | **326526832** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/Waiting for table metadata lock | **1000181831000** |

| **326526744** | **326526835** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/After create | **33483000** |

| **326526744** | **326526838** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/Waiting for table metadata lock | **1000091810000** |

| **326526744** | **326526841** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/After create | **17187000** |

| **326526744** | **326526844** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/Waiting for table metadata lock | **1000126464000** |

| **326526744** | **326526847** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/After create | **27472000** |

| **326526744** | **326526850** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/Waiting for table metadata lock | **561996133000** |

| **326526744** | **326526853** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/After create | **124876000** |

| **326526744** | **326526877** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/System lock | **30659000** |

| **326526744** | **326526881** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/preparing for alter table | **40246000** |

| **326526744** | **326526889** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/altering table | **36628000** |

| **326526744** | **326528280** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/end | **43824000** |

| **326526744** | **326528281** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/query end | **112557000** |

| **326526744** | **326528299** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/closing tables | **27707000** |

| **326526744** | **326528305** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/freeing items | **201614000** |

| **326526744** | **326528308** | alter table test\_slow modify c2 varchar(**152**) | stage/sql/cleaning up | **3584000** |

+-----------+-----------+----------------------------------------------+----------------------------------------------------+---------------+

从结果可以看到，出现了多次stage/sql/Waiting for table metadata lock阶段，并且间隔1s，说明每隔1s钟会重试判断。找一个该阶段的event\_id,通过nesting\_event\_id关联，确定到底在等待哪个wait事件。

SELECT

event\_id,

event\_name,

source,

timer\_wait,

object\_name,

index\_name,

operation,

nesting\_event\_id

FROM events\_waits\_history\_long

WHERE nesting\_event\_id = **326526850**;

+-----------+---------------------------------------------------+------------------+--------------+-------------+------------+------------+------------------+

| event\_id | event\_name | source | timer\_wait | object\_name | index\_name | operation | nesting\_event\_id |

+-----------+---------------------------------------------------+------------------+--------------+-------------+------------+------------+------------------+

| **326526851** | wait/synch/cond/sql/MDL\_context::COND\_wait\_status | mdl.cc:**1327** | **562417991328** | NULL | NULL | timed\_wait | **326526850** |

| **326526852** | wait/synch/mutex/mysys/my\_thread\_var::mutex | sql\_class.h:**3481** | **733248** | NULL | NULL | lock | **326526850** |

+-----------+---------------------------------------------------+------------------+--------------+-------------+------------+------------+------------------+

通过结果可以知道，产生阻塞的是条件变量MDL\_context::COND\_wait\_status，并且显示了代码的位置。

View Code

**总结：**

本文通过对Performance Schema数据库的介绍，主要用于收集数据库服务器性能参数：①提供进程等待的详细信息，包括锁、互斥变量、文件信息；②保存历史的事件汇总信息，为提供MySQL服务器性能做出详细的判断；③对于新增和删除监控事件点都非常容易，并可以改变mysql服务器的监控周期，例如（CYCLE、MICROSECOND）。通过该库得到数据库运行的统计信息，更好分析定位问题和完善监控信息。类似的监控还有：

[复制代码](javascript:void(0);)

打开标准的innodb监控：

CREATE TABLE innodb\_monitor (a INT) ENGINE=INNODB;

打开innodb的锁监控：

CREATE TABLE innodb\_lock\_monitor (a INT) ENGINE=INNODB;

打开innodb表空间监控：

CREATE TABLE innodb\_tablespace\_monitor (a INT) ENGINE=INNODB;

打开innodb表监控：

CREATE TABLE innodb\_table\_monitor (a INT) ENGINE=INNODB;

[复制代码](javascript:void(0);)

**参考文章：**

<https://dev.mysql.com/doc/refman/5.6/en/performance-schema.html>

<http://www.cnblogs.com/cchust/p/5022148.html>

<http://www.cnblogs.com/cchust/p/5057498.html>

<http://www.cnblogs.com/cchust/p/5061131.html>

[http://mysqllover.com/?p=522](http://mysqllover.com/?p=522&utm_source=tuicool&utm_medium=referral)

~~~~~~~~~~~~~~~ 万物之中,希望至美 ~~~~~~~~~~~~~~~

分类: [MySQL](http://www.cnblogs.com/zhoujinyi/category/421066.html),[随笔](http://www.cnblogs.com/zhoujinyi/category/414098.html)

标签: [performance\_schema、performance schema、监控](http://www.cnblogs.com/zhoujinyi/tag/performance_schema%E3%80%81performance%20schema%E3%80%81%E7%9B%91%E6%8E%A7/)