mysql性能剖析

完成一项任务所花费的时间可以分为两部分：执行时间和等待时间。

分析查询。我们定义性能最有效的方法是响应时间。

# 通过pt-query-digest分析服务器负载

捕获mysql所有的慢查询可以配置：long\_query\_time=0

## 下载和安装

|  |
| --- |
| wget http://www.percona.com/get/pt-query-digest  chmod u+x pt-query-digest  mv pt-query-digest /usr/local/bin/ |

## pt-query-digest的使用

### pt-query-digest参数介绍

|  |
| --- |
| pt-query-digest --user=anemometer --password=123 --review h=172.16.129.128,D=slow\_query\_log,t=global\_query\_review \  --history h=172.16.129.128,D=slow\_query\_log,t=global\_query\_review\_history \  --no-report --limit=0% --filter=" \$event->{Bytes} = length(\$event->{arg}) and \$event->{hostname}=\"$HOSTNAME\"" \  /data/mysql/log/slow.log |

* –filter 对输入的慢查询按指定的字符串进行匹配过滤后再进行分析
* –limit限制输出结果百分比或数量，默认值是20,即将最慢的20条语句输出，如果是50%则按总响应时间占比从大到小排序，输出到总和达到50%位置截止。
* –host mysql服务器地址
* –user mysql用户名
* –password mysql用户密码
* --history 将分析结果保存到表中，分析结果比较详细，下次再使用--history时，如果存在相同的语句，且查询所在的时间区间和历史表中的不同，则会记录到数据表中，可以通过查询同--CHECKSUM来比较某类型查询的历史变化。
* --review 将分析结果保存到表中，这个分析只是对查询条件进行参数化，一个类型的查询一条记录，比较简单。当下次使用--review时，如果存在相同的语句分析，就不会记录到数据表中。
* --output 分析结果输出类型，值可以是report(标准分析报告)、slowlog(Mysql slow log)、json、json-anon，一般使用report，以便于阅读。
* --since 从什么时间开始分析，值为字符串，可以是指定的某个”yyyy-mm-dd [hh:mm:ss]”格式的时间点，也可以是简单的一个时间值：s(秒)、h(小时)、m(分钟)、d(天)，如12h就表示从12小时前开始统计。
* --until 截止时间，配合--since可以分析一段时间内的慢查询。

### 报告结果分析

**开始总的摘要信息*：***

|  |
| --- |
| --此工具执行日志分析时的所用时间、内存资源(rss物理内存占用大小，vsz虚拟内存占用大小)  # 1.1s user time, 90ms system time, 35.82M rss, 187.30M vsz  --分析时的系统时间  # Current date: Wed Jan 13 16:28:17 2016  --进行分析的主机名，非记录日志的数据库服务器  # Hostname: vdle-commerce53  --分析的日志文件名称  # Files: slow.log  --文件中总共的语句数量，唯一的语句数量(对语句进行了格式化)，QPS，并发数  # Overall: 4.78k total, 417 unique, 0.05 QPS, 0.18x concurrency \_\_\_\_\_\_\_\_\_\_  --记录日志的时间范围  # Time range: 2016-01-12 11:54:14 to 2016-01-13 16:25:40 |

|  |
| --- |
| # Attribute total min max avg 95% stddev median  # ============ ======= ======= ======= ======= ======= ======= =======  # Exec time 18823s 100ms 1526s 4s 3s 36s 163ms  # Lock time 14235s 0 429s 3s 75ms 28s 49us  # Rows sent 56.63M 0 914.41k 12.13k 12.50k 52.75k 793.42  # Rows examine 130.62M 0 3.64M 27.98k 130.04k 111.67k 5.73k  # Query size 6.60M 6 921.37k 1.41k 755.64 27.44k 72.65 |

* –Exec time：语句执行时间
* –Lock time：锁占有时间
* –Rows sent：发送到客户端的行数
* –Row examine：扫描的行数(SELECT语句)
* –Row affecte：发送改变的行数(UPDATE, DELETE, INSERT语句)
* –Bytes sent：发送多少bytes的查询结果集
* –Query size：查询语句的字符数

**查询分组统计结果**

|  |
| --- |
| # Profile  # Rank Query ID Response time Calls R/Call V/M Item  # ==== ================== =============== ===== ========= ===== ==========  # 1 0x8FF01CCA61099478 8400.8008 44.6% 33 254.5697 32.76 INSERT user  # 2 0x5F108E0C07546C37 2995.3410 15.9% 25 119.8136 12... UPDATE user  # 3 0x0D71DE723F0B2DFE 1525.6127 8.1% 1 1525.6127 0.00 SELECT UNION parent student school area parent login\_record student login\_record study\_everage\_new student parent order\_info assignment\_record\_detail student exam\_record\_detail exam\_record exam\_record\_detail exam\_record assignment\_record exam\_record assignment\_record study\_everage\_new student assignment\_record\_detail student exam\_record\_detail exam\_record exam\_record\_detail exam\_record assignment\_record exam\_record assignment\_record user friendship\_card  # 4 0x7887892BFFD1B101 997.0236 5.3% 5 199.4047 13... DELETE message\_mobile  # 5 0x3BD0F99AD93D65B0 888.8154 4.7% 7 126.9736 20... UPDATE com.hengtiansoft user  # 6 0x82CBC81A3E407391 561.5900 3.0% 6 93.5983 19... UPDATE user  # 7 0xB49526993E41455D 496.5757 2.6% 35 14.1879 12.97 SHOW TABLE STATUS  # 8 0x67A347A2812914DF 429.6014 2.3% 1358 0.3163 1.80 SELECT school  # 9 0x17B84F2D62CB9D9C 198.1246 1.1% 6 33.0208 15... UPDATE tgindex  # 10 0x813031B8BBC3B329 155.4107 0.8% 83 1.8724 8.49 COMMIT  # 11 0x3D53D0B1275A4E77 147.0607 0.8% 385 0.3820 7.29 SELECT k?\_qa\_all.question\_knowledge\_point\_?\_?  # 12 0x8CFCC5D65CBF5ED5 143.8177 0.8% 3 47.9392 1.24 SELECT INFORMATION\_SCHEMA.FILES INFORMATION\_SCHEMA.PARTITIONS |

* –Rank:分析的所有查询语句的排名，默认按查询时间降序排序，可以通过–order-by指定排序方式
* –Query ID：查询语句的哈希指纹，去掉了多余空格、和文本字符
* –Response time：响应时间，占所有响应时间的百分比
* –Calls：查询执行的次数
* –R/Call：每次执行的平均响应时间
* –V/M：响应时间Variance-to-mean的比率，参考：http://en.wikipedia.org/wiki/Index\_of\_dispersion
* –Item：查询语句
* –最后一行没有包括在报告中的查询合计统计信息，如使用了选项–limit和–outliers

**每个独立查询语句的分析**

|  |
| --- |
| --QPS：每秒查询数(queries per second)  --concurrency：该查询的近似并发值  --ID：16进制，查询语句的指纹，去掉了多余空格、和文本字符、转换成小写，使用--filter可以用来进行过滤(如：pt-query-digest mysql-slow.201407250000 --filter '$event->{fingerprint} && make\_checksum($event->{fingerprint}) eq "0793E2F7F5EBE1B1"' > slow2.txt)，必须移除0x  --at byte 1087859：查询语句在日志文件中的偏移量(byte offset)，不一定精确，根据偏移量在日志文件中查找语句(如tail -c +1087859 mysql-slow.201407250000 |head)  # Query 1: 0.00 QPS, 1.22x concurrency, ID 0x8FF01CCA61099478 at byte 1087859  # This item is included in the report because it matches --limit.  # Scores: V/M = 32.76  # Time range: 2016-01-12 16:28:52 to 18:23:50  # Attribute pct total min max avg 95% stddev median  # ============ === ======= ======= ======= ======= ======= ======= =======  --pct在整个日志文件中，执行语句占用百分比(0%)，总计执行了33次  # Count 0 33  # Exec time 44 8401s 104ms 405s 255s 346s 91s 258s  # Lock time 58 8382s 100us 404s 254s 346s 91s 258s  # Rows sent 0 0 0 0 0 0 0 0  # Rows examine 0 0 0 0 0 0 0 0  # Query size 0 18.32k 566 643 568.33 563.87 9.91 563.87  # String:  --数据库名  # Databases k12\_qa\_all  # Hosts 172.16.129.201  --执行语句的用户名  # Users root  --查询的执行时间分布情况图，可以使用选项--report-histogram进行定义  # Query\_time distribution  # 1us  # 10us  # 100us  # 1ms  # 10ms  # 100ms ######  # 1s  # 10s+ ################################################################  # Tables  --可以使用该语句查询表的统计信息，如大小  # SHOW TABLE STATUS FROM `k12\_qa\_all` LIKE 'user'\G  --可以使用该语句查看表的结构信息  # SHOW CREATE TABLE `k12\_qa\_all`.`user`\G  --可以使用该语句查看查询计划，如非select语句，工具会转换成类似的select语句，方便进行explain  /\* insert com.hengtiansoft.k12.common.security.domain.User \*/ insert into user (area\_id, avatar\_url, book\_coin, create\_by, create\_ts, data\_source, edu\_coin, email, first\_login, gender, host\_area, invite\_code, is\_open, is\_virtural, last\_login\_ts, mobile, parent\_id, password, point, qq\_coin, status, stcard\_security\_code, stcard\_wrong\_count, user\_id, user\_name, user\_type) values (null, null, null, null, null, null, null, null, null, null, null, '830410', null, null, null, '15700120000', null, '123456', null, null, 'I', null, null, '15700120000T', 'username', 'U')\G |

# 使用Anemometer分析MySQL慢查询记录

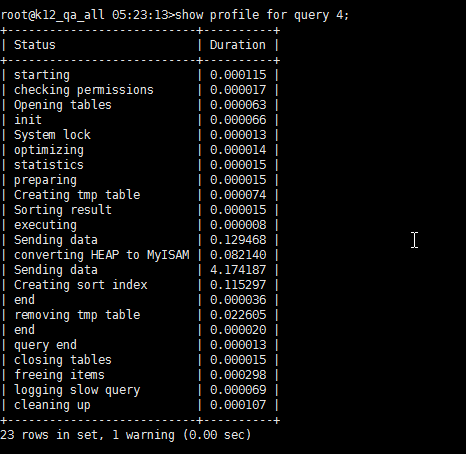
## 安装配置

|  |
| --- |
| #安装配置LANM环境  yum install -y php-bcmath.x86\_64 php.x86\_64 php-dba.x86\_64 php-cli.x86\_64 php-pdo.x86\_64 php-common.x86\_64 php-gd.x86\_64 php-mysql.x86\_64 php-devel  vi /etc/php.ini  date.timezone = Asia/Shanghai  #下载Anemometer  cd /usr/local/src  git clone https://github.com/box/Anemometer.git anemometer  cd anemometer  #初始化表结构  mysql -uroot -p123< mysql56-install.sql  mysql -uroot -p123< install.sql  mysql -uroot -p123 -e "grant ALL ON slow\_query\_log.\* to 'anemometer'@'%' IDENTIFIED BY '123';"  mysql -uroot -p123 -e "grant SELECT ON \*.\* to 'anemometer'@'%' IDENTIFIED BY '123';"  #安装percona的toolkit工具  yum install -y <http://www.percona.com/downloads/percona-release/redhat/0.1-3/percona-release-0.1-3.noarch.rpm>  yum install percona-toolkit -y  pt-query-digest --user=anemometer --password=123 --review h=172.16.129.128,D=slow\_query\_log,t=global\_query\_review --history h=172.16.129.128,D=slow\_query\_log,t=global\_query\_review\_history --no-report --limit=0% --filter=" \$event->{Bytes} = length(\$event->{arg}) and \$event->{hostname}=\"$HOSTNAME\"" /data/mysql/log/slow.log  #修改配置文件,配置anemometer  cp -r anemometer/ /var/www/html  cd /var/www/html/anemometer/conf/  cp sample.config.inc.php config.inc.php  vi config.inc.php  date.timezone = Asia/Shanghai  $conf['datasources']['172.16.129.128'] = array(  'host' => '172.16.129.128',  'port' => 3306,  'db' => 'slow\_query\_log',  'user' => 'anemometer',  'password' => '123',  'tables' => array(  'global\_query\_review' => 'fact',  'global\_query\_review\_history' => 'dimension'  ),  'source\_type' => 'slow\_query\_log'  );  #自动化处理  # vi /etc/logrotate.d/mysql  postrotate  pt-query-digest --user=anemometer --password=123 --review h=172.16.129.128,D=slow\_query\_log,t=global\_query\_review --history h=172.16.129.128,D=slow\_query\_log,t=global\_query\_review\_history --no-report --limit=0% --filter=" \$event->{Bytes} = length(\$event->{arg}) and \$event->{hostname}=\"$HOSTNAME\"" /data/mysql/log/slow.log  endscript |
| #模板  $conf['datasources']['localhost'] = array(  'host' => '172.16.129.128',  'port' => 3306,  'db' => 'slow\_query\_log',  'user' => 'anemometer',  'password' => '123',  'tables' => array(  'global\_query\_review' => 'fact',  'global\_query\_review\_history' => 'dimension'  ),  'source\_type' => 'slow\_query\_log'  );    $conf['plugins'] = array(    'visual\_explain' => '/usr/bin/pt-visual-explain',  'query\_advisor' => '/usr/bin/pt-query-advisor',    #... other lines    $conn['user'] = 'anemometer';  $conn['password'] = 'my.ttlsa.com';    return $conn;  }, |

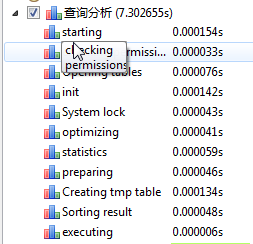
# 通过show profile剖析单条查询

## 使用mysql自带工具

|  |
| --- |
| *-- 开启查询剖析*  **set** profiling=1;  *-- 执行查询语句*  **SELECT** \*  **FROM** school  **GROUP** **BY** NAME  ,area\_id  , **LEVEL**  **HAVING** **COUNT**(\*) > 1;  *-- 查看查询剖析*  **show** profiles;  **show** profile **for** **query** 1; |



## 使用HeidiSQL客户端

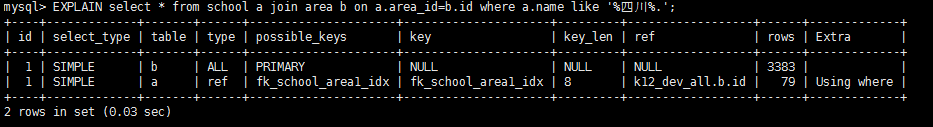


# MySQL性能分析：explain的使用

MySQL的EXPLAIN命令用于SQL语句的查询执行计划(QEP)。，大部分的性能问题可以通过此命令来简单的解决，Explain可以用来查看SQL语句的执行效 果，可以帮助选择更好的索引和优化查询语句，写出更好的优化语句。MySQL将查询分为简单查询和复杂查询。复杂类型分为简单子查询，所谓派生表(在from子句中的子查询)，以及union查询。

语法：explain [extended partitions] 查询语句或者更新语句

EXPLAIN select \* from school a join area b on a.area\_id=b.id where a.name like '%四川%.';



## explain列解析

### id列

id 列是在QEP 中展示的表的连续引用。标识SELECT所属，一般按照sql语句的顺序来编号。(**MySQL执行顺序：id越大，越开始执行。id一样，顺序执行。**)

### select\_type列

select\_type显示对应的查询是简单还是复杂查询。Simple意味着查询不包括子查询和UNION。如果查询包含任何复杂的子查询，则最外层部分标记为PRIMARY。最常见的值包括SIMPLE、PRIMARY、DERIVED 和UNION。其他可能的值还有UNION RESULT、DEPENDENT SUBQUERY、DEPENDENT UNION、UNCACHEABLE UNION 以及UNCACHEABLE QUERY。

SIMPLE：对于不包含子查询和其他复杂语法的简单查询。

PRIMARY：为更复杂的查询而创建的首要表(**也就是最外层的表**，最后执行)。

DERIVED：用来表示包含在FROM子句的子查询中的select,mysql会递归执并将结果放在一个临时表中。服务器称其为派生表。

SUBQUERY：包含在select列表中的子查询中的 select(不在from子句中)

UNION：在UNION总第二个或随后的SELECT会被标记为UNION。第一个SELECT被标记就好像它以部分外查询来执行。例如过UNION左边被FROM包含，则它的第一个SELECT标记为DERIVED

UNION RESULT：用来从UNION的匿名临时表检索的结果SELECT被标记为UNION RESULT

**注意**：除了这些值，SUBQUERY，UNION还可以被标记为DEPENDENT和UNCACHEABLE。DEPENDENT 意味着SELECT依赖于外层查询发现的数据；UNCACHEABLE意味着SELECT中的某些性能阻止结果被缓存于一个Item\_  
cache中。

### table列

标识这一列对应行正在访问哪个表的名字或者别名（sql中定义的查询表或者别名）。当有FROM子句的时候，对应的临时表为<devivedN>N标识的子查询的id；当有UNION时候，UNION RESULT的table列会包含参与UNION的id列表。

### type列

关联类型决定MySQL如何查询表中的行。依次从差到优：

ALL：全表扫描，通常意味着MySQL必须扫描整张表，从头到尾，去找到需要的行。（除使用limit，或者extra列中显示Usingdistinct/not exists）

index：**按索引次序进行全表扫描**，有点是避免了排序，缺点是承担按索引次序读取整个表的开销。这通常意味着如果按随机次序访问行，开销会很大。如果在Extra列中有using index，说明正在使用覆盖索引，它只扫描索引的数据而不是按索引次序的每一行。(例如：select id from t)

range：**范围扫描就是有限制的索引扫描**，开始于索引里的某一点，返回匹配这个值域的行。不用遍历所有的全部索引，显而易见的范围扫描有between或者where子句带有>查询，or,in()等。

ref：叫做**索引查找**，它返回所有匹配匹配某个单个值的行。此类索引访问只有当使用非唯一性索引或者唯一性索引的非唯一性前缀时才会发生。叫做ref是因为索引要跟某个参考值相比较。ref\_or\_null是ref的一个变体，表示mysql必须在初次查找结果中进行第二次查找以找出NULL条目。

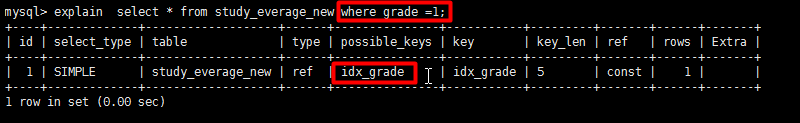
eq\_ref：使用这种索引查找，mysql最多只返回一条符合条件的记录。**这种方法可以在mysql使用主键或者唯一性索引查找时出现**。它性能非常好，它无须估计匹配行的范围或者找到行后继续查找。

const,system：当mysql能对查询的某部分进行优化并将其转化成一个常量时，他就会使用这些访问类型。例如，通过某行的主键放在where子句中来获取此行的主键，mysql就会把这个查询转换成一个常量。然后就可以高效的将表从联接执行中移除。(例如：select \* from (select \* from b where id=1 ) a;)

NULL：这种访问方式表示mysql能在优化阶段分解查询语句，在执行阶段甚至用不着在访问表或者索引。例如从索引列中选取最小值可以通过单独查找索引来完成，不需要在执行时访问表。

### possible\_keys

这一列显示了查询可以使用哪些索引，这是基于查询访问的列和使用的比较操作符来判断的。**NULL表示理论上没有使用索引。**



### key列

这一列显示MySQL决定采用哪种索引来优化对该表的访问，如果该索引没有出现在possible\_keys中，那么选用它可能出于另一种原因。例如，它可能选择了覆盖索引。**NULL 表示索引实效。**

FORCE INDEX -- 强逼使用某个索引

USE INDEX --使用某个索引

IGNORE INDEX -- 忽略某个索引

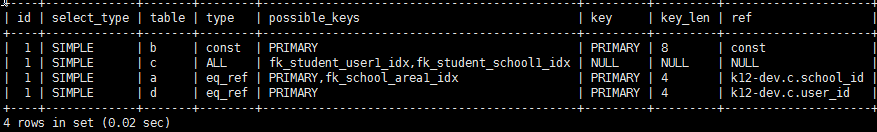
### key\_len列

**mysql在索引中使用的字节数**。key\_len显示了索引字段可能的最大长度，而不是表总数据使用的实际字节数。Key\_len是通过查找表定义而被计算出来的，而不是表中的数据。

### ref列

这一列显示了之前的表在key列记录的索引查找值**所引用的列或者常量**。(**索引引用的列或者常量**)

|  |
| --- |
| **EXPLAIN**  **SELECT** \*  **FROM** school a  **JOIN** **AREA** b **ON** a.area\_id=b.id **AND** a.area\_id=3306  **JOIN** student c **ON** c.school\_id=a.id  **JOIN** **USER** d **ON** d.id =c.user\_id; |

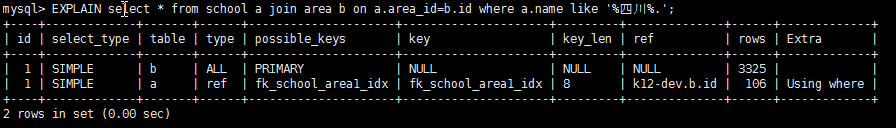


### rows列

MySQL估计为了找到所需的行而要读取的行数。(MySQL认为要检索行数)

### filtered列

filtered 列给出了一个百分比的值，这个百分比值和rows 列的值一起使用，可以估计出那些将要和QEP 中的前一个表进行连 接的行的数目。前一个表就是指id 列的值比当前表的id 小的表。这一列只有在EXPLAIN EXTENDED 语句中才会出现。



### partitions列

partitions 列代表给定表所使用的分区。这一列只会在EXPLAIN

PARTITIONS 语句中出现。

### Extra列

"**Using index**"：此值**表示MySQL将使用覆盖索引**，以避免访问表。不要把覆盖索引和index访问类型弄混了。

"**Using where**"：这意味着MySQL服务器将在**存储引擎检索行后在进行过滤**。许多WHERE条件里涉及索引的列，当它读取索引的时，就能被存储引擎检验，因此不是所有的where子句的查询都会显示"Using index"。有时"Using where"的出现就是一个暗示：查询可收益于不同的索引。

"**Using temporary**"：意味着MySQL在对查询排序时会使用的一个临时表。

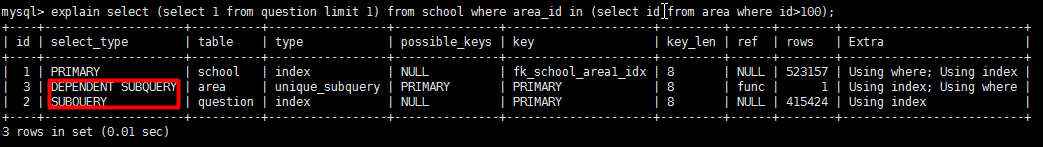
"**Using filesort**"：意味着MySQL会对结果使用一个外部索引排序，而不是按索引次序从表里读取行。可以通过选择合适的索引来改进性能，用索引来为查询结果排序。

"**Range checked for each recofe(index map:N**"：这个值意味着没有好用的索引，新的索引将在联接的每一行上重新评估。N使显示possible\_keys列中素引的位图，并且是冗余的。

## 复杂查询分析举例

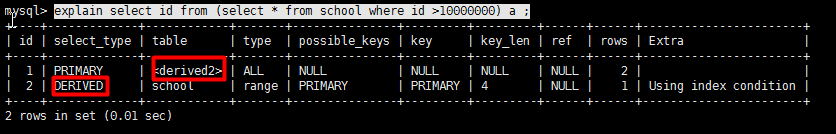
### 简单子查询

explain select (select 1 from question limit 1) from school where area\_id in (select id from area where id>100);

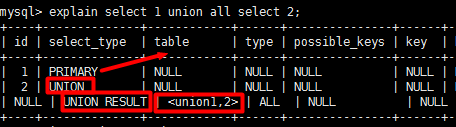


### From子句的子查询

explain select id from (select \* from school where id >10000000) a ;



### Union查询



注意：诊断间歇性问题。有时候我们单独诊断单条查询语句的时候，发现执行没有问题，并且波动比较大。这个时候可能是因为服务器性能下降，可以通过show grobal status或者show processlist来查看系统状态