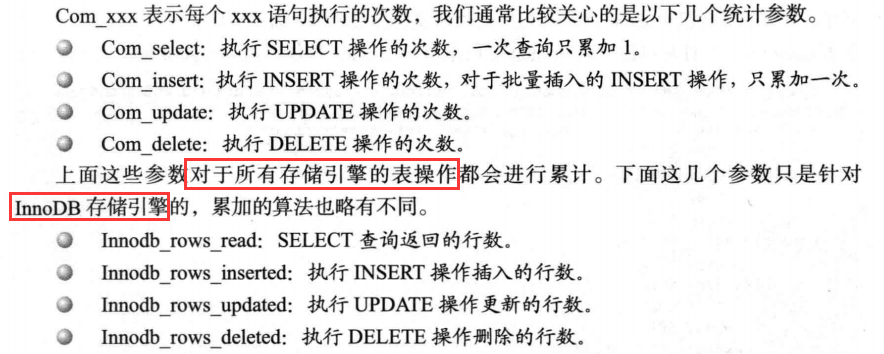
# 18.1.1 show status执行频率



# 2@MySQL 性能调优的10个方法 - 小米 - 博客频道 - CSDN.NET

http://blog.csdn.net/u011225629/article/details/47092911

[**MySQL**](http://lib.csdn.net/base/mysql) 应该是最流行了 WEB 后端[**数据库**](http://lib.csdn.net/base/mysql)。WEB 开发语言最近发展很快，[**PHP**](http://lib.csdn.net/base/php)， Ruby, [**Python**](http://lib.csdn.net/base/python), [**Java**](http://lib.csdn.net/base/javase) 各有特点，虽然 NOSQL 最近越來越多的被提到，但是相信大部分[**架构**](http://lib.csdn.net/base/architecture)师还是会选择 MYSQL 来做数据存储。

MYSQL 如此方便和稳定，以至于我们在开发 WEB 程序的时候很少想到它。即使想到优化也是程序级别的，比如，不要写过于消耗资源的 SQL 语句。但是除此之外，在整个系统上仍然有很多可以优化的地方。

1. 选择合适的存储引擎: InnoDB

除非你的数据表使用来做只读或者全文检索 (相信现在提到全文检索，没人会用 MYSQL 了)，你应该默认选择 InnoDB 。

你自己在[**测试**](http://lib.csdn.net/base/softwaretest)的时候可能会发现 MyISAM 比 InnoDB 速度快，这是因为： MyISAM 只缓存索引，而 InnoDB 缓存数据和索引，MyISAM 不支持事务。但是 如果你使用 innodb\_flush\_log\_at\_trx\_commit = 2 可以获得接近的读取性能 (相差百倍) 。

1.1 如何将现有的 MyISAM 数据库转换为 InnoDB:

mysql -u [USER\_NAME] -p -e "SHOW TABLES IN [DATABASE\_NAME];" | tail -n +2 | xargs -I '{}' echo "ALTER TABLE {} ENGINE=InnoDB;" > alter\_table.sql

perl -p -i -e 's/(search\_[a-z\_]+ ENGINE=)InnoDB//1MyISAM/g' alter\_table.sql

mysql -u [USER\_NAME] -p [DATABASE\_NAME] < alter\_table.sql

1.2 为每个表分别创建 InnoDB FILE：

innodb\_file\_per\_table=1

这样可以保证 ibdata1 文件不会过大，失去控制。尤其是在执行 mysqlcheck -o –all-databases 的时候。

2. 保证从内存中读取数据，讲数据保存在内存中

2.1 足够大的 innodb\_buffer\_pool\_size

推荐将数据完全保存在 innodb\_buffer\_pool\_size ，即按存储量规划 innodb\_buffer\_pool\_size 的容量。这样你可以完全从内存中读取数据，最大限度减少磁盘操作。

2.1.1 如何确定 innodb\_buffer\_pool\_size 足够大，数据是从内存读取而不是硬盘？

方法 1

mysql> SHOW GLOBAL STATUS LIKE 'innodb\_buffer\_pool\_pages\_%';

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

| Variable\_name | Value |

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

| Innodb\_buffer\_pool\_pages\_data | 129037 |

| Innodb\_buffer\_pool\_pages\_dirty | 362 |

| Innodb\_buffer\_pool\_pages\_flushed | 9998 |

| Innodb\_buffer\_pool\_pages\_free | 0 | !!!!!!!!

| Innodb\_buffer\_pool\_pages\_misc | 2035 |

| Innodb\_buffer\_pool\_pages\_total | 131072 |

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

6 rows in set (0.00 sec)

发现 Innodb\_buffer\_pool\_pages\_free 为 0，则说明 buffer pool 已经被用光，需要增大 innodb\_buffer\_pool\_size

InnoDB 的其他几个参数：

innodb\_additional\_mem\_pool\_size = 1/200 of buffer\_pool

innodb\_max\_dirty\_pages\_pct 80%

方法 2

或者用iostat -d -x -k 1 命令，查看硬盘的操作。

2.1.2 服务器上是否有足够内存用来规划

执行 echo 1 > /proc/sys/vm/drop\_caches 清除[**操作系统**](http://lib.csdn.net/base/operatingsystem)的文件缓存，可以看到真正的内存使用量。

2.2 数据预热

默认情况，只有某条数据被读取一次，才会缓存在 innodb\_buffer\_pool。所以，数据库刚刚启动，需要进行数据预热，将磁盘上的所有数据缓存到内存中。数据预热可以提高读取速度。

对于 InnoDB 数据库，可以用以下方法，进行数据预热:

1. 将以下脚本保存为 MakeSelectQueriesToLoad.sql

SELECT DISTINCT

CONCAT('SELECT ',ndxcollist,' FROM ',db,'.',tb,

' ORDER BY ',ndxcollist,';') SelectQueryToLoadCache

FROM

(

SELECT

engine,table\_schema db,table\_name tb,

index\_name,GROUP\_CONCAT(column\_name ORDER BY seq\_in\_index) ndxcollist

FROM

(

SELECT

B.engine,A.table\_schema,A.table\_name,

A.index\_name,A.column\_name,A.seq\_in\_index

FROM

information\_schema.statistics A INNER JOIN

(

SELECT engine,table\_schema,table\_name

FROM information\_schema.tables WHERE

engine='InnoDB'

) B USING (table\_schema,table\_name)

WHERE B.table\_schema NOT IN ('information\_schema','mysql')

ORDER BY table\_schema,table\_name,index\_name,seq\_in\_index

) A

GROUP BY table\_schema,table\_name,index\_name

) AA

ORDER BY db,tb

;

2. 执行

mysql -uroot -AN < /root/MakeSelectQueriesToLoad.sql > /root/SelectQueriesToLoad.sql

3. 每次重启数据库，或者整库备份前需要预热的时候执行：

mysql -uroot < /root/SelectQueriesToLoad.sql > /dev/null 2>&1

2.3 不要让数据存到 SWAP 中

如果是专用 MYSQL 服务器，可以禁用 SWAP，如果是共享服务器，确定 innodb\_buffer\_pool\_size 足够大。或者使用固定的内存空间做缓存，使用 memlock 指令。

3. 定期优化重建数据库

mysqlcheck -o –all-databases 会让 ibdata1 不断增大，真正的优化只有重建数据表结构：

CREATE TABLE mydb.mytablenew LIKE mydb.mytable;

INSERT INTO mydb.mytablenew SELECT \* FROM mydb.mytable;

ALTER TABLE mydb.mytable RENAME mydb.mytablezap;

ALTER TABLE mydb.mytablenew RENAME mydb.mytable;

DROP TABLE mydb.mytablezap;

4. 减少磁盘写入操作

4.1 使用足够大的写入缓存 innodb\_log\_file\_size

但是需要注意如果用 1G 的 innodb\_log\_file\_size ，假如服务器当机，需要 10 分钟来恢复。

推荐 innodb\_log\_file\_size 设置为 0.25 \* innodb\_buffer\_pool\_size

4.2 innodb\_flush\_log\_at\_trx\_commit

这个选项和写磁盘操作密切相关：

innodb\_flush\_log\_at\_trx\_commit = 1 则每次修改写入磁盘  
innodb\_flush\_log\_at\_trx\_commit = 0/2 每秒写入磁盘

如果你的应用不涉及很高的安全性 (金融系统)，或者基础架构足够安全，或者 事务都很小，都可以用 0 或者 2 来降低磁盘操作。

4.3 避免双写入缓冲

innodb\_flush\_method=O\_DIRECT

5. 提高磁盘读写速度

RAID0 尤其是在使用 EC2 这种虚拟磁盘 (EBS) 的时候，使用软 RAID0 非常重要。

6. 充分使用索引

6.1 查看现有表结构和索引

SHOW CREATE TABLE db1.tb1/G

6.2 添加必要的索引

索引是提高查询速度的唯一方法，比如[**搜索引擎**](http://lib.csdn.net/base/searchengine)用的倒排索引是一样的原理。

索引的添加需要根据查询来确定，比如通过慢查询日志或者查询日志,或者通过 EXPLAIN 命令分析查询。

ADD UNIQUE INDEX

ADD INDEX

6.2.1 比如，优化用户验证表：

添加索引

ALTER TABLE users ADD UNIQUE INDEX username\_ndx (username);

ALTER TABLE users ADD UNIQUE INDEX username\_password\_ndx (username,password);

每次重启服务器进行数据预热

echo “select username,password from users;” > /var/lib/mysql/upcache.sql

添加启动脚本到 my.cnf

[mysqld]

init-file=/var/lib/mysql/upcache.sql

6.2.2 使用自动加索引的框架或者自动拆分表结构的框架

比如，Rails 这样的框架，会自动添加索引，Drupal 这样的框架会自动拆分表结构。会在你开发的初期指明正确的方向。所以，经验不太丰富的人一开始就追求从 0 开始构建，实际是不好的做法。

7. 分析查询日志和慢查询日志

记录所有查询，这在用 ORM 系统或者生成查询语句的系统很有用。

log=/var/log/mysql.log

注意不要在生产环境用，否则会占满你的磁盘空间。

记录执行时间超过 1 秒的查询：

long\_query\_time=1

log-slow-queries=/var/log/mysql/log-slow-queries.log

8. 激进的方法，使用内存磁盘

现在基础设施的可靠性已经非常高了，比如 EC2 几乎不用担心服务器硬件当机。而且内存实在是便宜，很容易买到几十G内存的服务器，可以用内存磁盘，定期备份到磁盘。

将 MYSQL 目录迁移到 4G 的内存磁盘

mkdir -p /mnt/ramdisk

sudo mount -t tmpfs -o size=4000M tmpfs /mnt/ramdisk/

mv /var/lib/mysql /mnt/ramdisk/mysql

ln -s /tmp/ramdisk/mysql /var/lib/mysql

chown mysql:mysql mysql

9. 用 NOSQL 的方式使用 MYSQL

B-TREE 仍然是最高效的索引之一，所有 MYSQL 仍然不会过时。

用 HandlerSocket 跳过 MYSQL 的 SQL 解析层，MYSQL 就真正变成了 NOSQL。

10. 其他

* 单条查询最后增加 LIMIT 1，停止全表扫描。
* 将非”索引”数据分离，比如将大篇文章分离存储，不影响其他自动查询。
* 不用 MYSQL 内置的函数，因为内置函数不会建立查询缓存。
* PHP 的建立连接速度非常快，所有可以不用连接池，否则可能会造成超过连接数。当然不用连接池 PHP 程序也可能将
* 连接数占满比如用了 @ignore\_user\_abort(TRUE);
* 使用 IP 而不是域名做数据库路径，避免 DNS 解析问题

# 2@MySQL监控主要指标及采集方法

\_数据库技术\_Linux公社-Linux系统门户网站

http://www.linuxidc.com/Linux/2016-11/136788.htm

MySQL监控属于DB监控的模块之一，包括采集、展示、监控告警。本文主要介绍MySQL监控的主要指标和采集方法。

　　MySQL监控和Redis监控的逻辑类似，可参考文章《Redis监控主要指标及采集方法 <http://www.linuxidc.com/Linux/2016-11/136783.htm>》。

　　DBA前台添加MySQL监控时系统会调用自动调度平台接口将Mysql监控的加密账户密码和ip端口等信息发送至目标，同时发送采集Agent。

　　一、采集指标和命令

　　1、MySQL服务运行状态

约定所有MySQL服务都必须以ip1（内网ip）来绑定，每个机器只有一个ip1，可以有多个端口，即多个MySQL Server。采集程序读取ip端口信息文件来判断server是否存在。

sockParam=`ps aux | grep -P "mysqld.\*--port=${port}" | grep -oP " --socket.\*\.sock"`  # 空则获取不到该服务器端口mysql socket配置，请检查mysql配置是否正确  
MYSQL="/usr/local/mysql/bin/mysql -hlocalhost --port=${port} ${sockParam} -u${user} -p${password} "  
MYSQL\_ADMIN="/usr/local/mysql/bin/mysqladmin -hlocalhost --port=${port} ${sockParam} -u${user} -p${password} "  
curStatus=`${MYSQL} -e" show global status"`  # 空则是获取不到该服务器mysql状态，请检查mysql是否正常运行  
if [ -z "${curStatus}" ]  
then  
    portExists=0  
else  
    echo "${curStatus}" >> ${curFile}  
    portExists=1

　　2、连接数

${MYSQL\_ADMIN} processlist -v | wc -l

　　3、线程数

grep 'Threads\_connected' ${curFile} | awk '{print $2}'

　　4、慢查询数

grep 'Slow\_queries' ${curFile} | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值，等于最近1分钟的慢查询次数。上次数据保存在last.cache。

　　5、打开表数

grep 'Open\_tables' ${curFile} | awk -F ' ' '{print $2}'

　　6、每秒执行select数

grep 'Com\_select' ${curFile} | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值除以时间差，等于最近1分钟的执行数量。上次数据保存在last.cache。

　　7、每秒执行delete数

grep 'Com\_delete' ${curFile} | grep -v 'multi' | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值除以时间差，等于最近1分钟的执行数量。上次数据保存在last.cache。

　　8、每秒执行insert数

grep 'Com\_insert' ${curFile} | grep -v 'select' | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值除以时间差，等于最近1分钟的执行数量。上次数据保存在last.cache。

　　9、每秒执行update数

grep 'Com\_update' ${curFile} | grep -v 'multi' | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值除以时间差，等于最近1分钟的执行数量。上次数据保存在last.cache。

　　10、每秒钟执行replace数

grep 'Com\_replace' ${curFile} | grep -v 'select' | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值除以时间差，等于最近1分钟的执行数量。上次数据保存在last.cache。

　　11、每秒钟执行的 Innodb\_rows\_deleted

grep 'Innodb\_rows\_deleted' ${curFile} | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值除以时间差，等于最近1分钟的执行数量。上次数据保存在last.cache。

　　12、每秒钟执行的 Innodb\_rows\_inserted

grep 'Innodb\_rows\_inserted' ${curFile} | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值除以时间差，等于最近1分钟的执行数量。上次数据保存在last.cache。

　　13、每秒钟执行的 Innodb\_rows\_read

grep 'Innodb\_rows\_read' ${curFile} | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值除以时间差，等于最近1分钟的执行数量。上次数据保存在last.cache。

　　14、每秒钟执行的 Innodb\_rows\_updated

grep 'Innodb\_rows\_updated' ${curFile} | awk -F ' ' '{print $2}'

需要计算两次的慢查询次数得到差值除以时间差，等于最近1分钟的执行数量。上次数据保存在last.cache。

　　15、每秒钟执行的 innodb rows total

expr ${innodbRowsDeletedPS} + ${innodbRowsInsertedPS} + ${innodbRowsReadPS} + ${innodbRowsUpdatedPS}

等于前面四个Innodb\_rows\_\*执行次数的总和

　　16、每秒处理命令数 qps

expr ${mysqlSelectNumPS} + ${mysqlInsertNumPS} + ${mysqlUpdateNumPS} + ${mysqlDeleteNumPS} + ${mysqlReplaceNumPS}

等于前面五个mysql命令Com\_\*的数量总和

17、每秒接收字节数 KByte/s

grep 'Bytes\_received' ${curFile} | awk -F ' ' '{print $2}'

18、每秒发送字节数

grep 'Bytes\_sent' ${curFile} | awk -F ' ' '{print $2}'

19、可立即获得锁的次数

grep 'Table\_locks\_immediate' ${curFile} | awk -F ' ' '{print $2}'

　　20、不可立即获得锁的次数

grep 'Table\_locks\_waited' ${curFile} | awk -F ' ' '{print $2}'

　　21、一行锁定需等待时间

grep 'Innodb\_row\_lock\_waits' ${curFile} | awk -F ' ' '{print $2}'

　　22、 当前脏页数

grep 'Innodb\_buffer\_pool\_pages\_dirty' ${curFile} | awk -F ' ' '{print $2}'

　　23、要求清空的缓冲池页数

grep 'Innodb\_buffer\_pool\_pages\_flushed' ${curFile} | awk -F ' ' '{print $2}'

　　24、Innodb 写入日志字节数 KByte

grep 'Innodb\_os\_log\_written' ${curFile} | awk -F ' ' '{print $2}'

　　25、占用内存大小 MByte

pid=`ps aux | grep 'mysqld' | grep -Ev 'safe|grep' | awk '{print $2}' `  
mem=`cat /proc/${pid}/status | grep 'VmRSS' | awk '{print $2}'`  
mysqlMem=`echo "scale=2;${mem} / 1024" | bc`

除以1024得到MByte

　　26、handler socket每秒处理数

curHsTableLock=`grep 'Hs\_table\_lock' ${curFile} | awk '{print $2}'`  
preHsTableLock=`grep 'Hs\_table\_lock' ${preFile} | awk '{print $2}'`  
if [ -n "${curHsTableLock}" ]  
then  
    hsQPS=`echo "scale=0;(${curHsTableLock} - ${preHsTableLock}) / ${intervalTime}" | bc`  
else  
    hsQPS=0  
fi

　　27、主从同步和状态

#主从信息  
#是否为从服务器  
slave\_running=`grep 'Slave\_running' ${curFile} | awk '{print $2}'`  
if [ "${slave\_running}A" = "ONA" ]  
then  
    slaveRunning=1  
    slaveStatus=`${MYSQL} -e'show slave status\G'`  
    echo "${slaveStatus}" > ${slaveFile}  
      
    slaveIoRunning=`grep 'Slave\_IO\_Running' ${slaveFile} | awk -F ':' '{print $2}'`  
    slaveSqlRunning=`grep 'Slave\_SQL\_Running' ${slaveFile} | awk -F ':' '{print $2}'`

    if [ "${slaveIoRunning}A" == "NoA" -o "${slaveSqlRunning}A" == "NoA" ]  
    then  
        slaveRunning=3  
    fi  
      
    secondsBehindMaster=`grep 'Seconds\_Behind\_Master' ${slaveFile} | awk -F ':' '{print $2}'`  
    if [ "${secondsBehindMaster}A" = "NULLA" ]  
    then  
        secondsBehindMaster=8888  # 表示主从不同步  
    fi

    #是从库时 获取主库ip  
    master=`grep 'Master\_Host' ${slaveFile} | awk -F ':' '{print $2}'`  
    masterPort=`grep 'Master\_Port' ${slaveFile} | awk -F ':' '{print $2}'`  
else  
    master=""  
    masterPort=""  
    slaveRunning=0  
    secondsBehindMaster=10000  # 不用检测  
fi

注：Seconds\_Behind\_Master，该值作为判断主从延时的指标，那么它又是怎么得到这个值的呢，同时，它为什么又受到很多人 的质疑？

Seconds\_Behind\_Master是通过比较sql\_thread执行的event的timestamp和io\_thread复制好的 event的timestamp(简写为ts)进行比较，而得到的这么一个差值。我们都知道的relay-log和主库的bin-log里面的内容完全一样，在记录sql语句的同时会被记录上当时的ts，所以比较参考的值来自于binlog，其实主从没有必要与NTP进行同步，也就是说无需保证主从时钟的 一致。你也会发现，其实比较真正是发生在io\_thread与sql\_thread之间，而io\_thread才真正与主库有关联，于是，问题就出来了， 当主库I/O负载很大或是网络阻塞，io\_thread不能及时复制binlog（没有中断，也在复制），而sql\_thread一直都能跟上 io\_thread的脚本，这时Seconds\_Behind\_Master的值是0，也就是我们认为的无延时，但是，实际上不是，你懂得。这也就是为什 么大家要批判用这个参数来监控数据库是否发生延时不准的原因，但是这个值并不是总是不准，如果当io\_thread与master网络很好的情况下，那么 该值也是很有价值的。

之前，提到 Seconds\_Behind\_Master这个参数会有负值出现，我们已经知道该值是io\_thread的最近跟新的ts与sql\_thread执行到 的ts差值，前者始终是大于后者的，唯一的肯能就是某个event的ts发生了错误，比之前的小了，那么当这种情况发生时，负值出现就成为可能。

　　28、检测采集Agent心跳情况

**本文永久更新链接地址**：<http://www.linuxidc.com/Linux/2016-11/136788.htm>

# MySQL关键性能监控(QPS/TPS) - 星辰——博客园 - 博客园

http://www.cnblogs.com/chenty/p/5191777.html

原文链接：<http://www.cnblogs.com/chenty/p/5191777.html>

工作中尝尝会遇到各种数据库性能调优，除了查看某条SQL执行时间长短外，还需要对系统的整体处理能力有更全局的掌握。

QPS：Query per second，每秒查询量

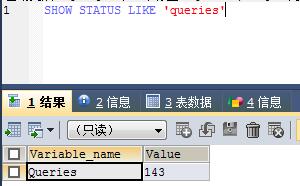
TPS：Transaction per second，每秒事物量

以上两个指标在实际应用中会经常被问到，作为一个项目领导者，必须时刻掌握这些重要指标，并根据相应趋势做出调整。

以下列出上述两个指标的具体算法：

QPS = Queries/ Seconds

Queries 是系统状态值--**总查询次数**，可以通过show status查询得出

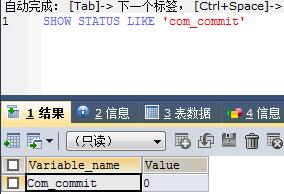


Seconds是监控的时间区间，单位秒

QPS = (Q2 - Q1)/10   间隔10秒查询两次并记录Q1，Q2，以此计算出QPS值

TPS = (Com\_commit + Com\_rollback) / Seconds

Com\_commit是系统状态值--事务提交数



Com\_rollback是系统状态值--事务回滚数



取时间差，计算规则同QPS

以上为手动计算，存在很多不便之处，后续可以考虑通过脚本自动计算。

分类: [mysql](http://www.cnblogs.com/chenty/category/787602.html)

# 2@mysql常用监控脚本命令整理\_Mysql\_脚本之家

http://www.jb51.net/article/37791.htm

#/bin/sh

#检测mysql server是否正常提供服务

mysqladmin -u sky -ppwd -h localhost ping

#获取mysql当前的几个状态值

mysqladmin -u sky -ppwd -h localhost status

#获取数据库当前的连接信息

mysqladmin -u sky -ppwd -h localhost processlist

#获取当前数据库的连接数

mysql -u root -p123456 -BNe "select host,count(host) from processlist group by host;" information\_schema

#显示mysql的uptime

mysql -e"SHOW STATUS LIKE '%uptime%'"|awk '/ptime/{ calc = $NF / 3600;print $(NF-1), calc"Hour" }'

#查看数据库的大小

mysql -u root -p123456-e 'select table\_schema,round(sum(data\_length+index\_length)/1024/1024,4) from information\_schema.tables group by table\_schema;'

sum(mysql\_info\_schema\_table\_size/1024/1024) by (schema)

#查看某个表的列信息

mysql -u <user> --password=<password> -e "SHOW COLUMNS FROM <table>" <database> | awk '{print $1}' | tr "\n" "," | sed 's/,$//g'

#执行mysql脚本

mysql -u user-name -p password < script.sql

#mysql dump数据导出

mysqldump -uroot -T/tmp/mysqldump test test\_outfile --fields-enclosed-by=\" --fields-terminated-by=,

#mysql数据导入

mysqlimport --user=name --password=pwd test --fields-enclosed-by=\" --fields-terminated-by=, /tmp/test\_outfile.txt

LOAD DATA INFILE '/tmp/test\_outfile.txt' INTO TABLE test\_outfile FIELDS TERMINATED BY '"' ENCLOSED BY ',';

#mysql进程监控

ps -ef | grep "mysqld\_safe" | grep -v "grep"

ps -ef | grep "mysqld" | grep -v "mysqld\_safe"| grep -v "grep"

#查看当前数据库的状态

mysql -u root -p123456 -e 'show status'

#mysqlcheck 工具程序可以检查(check),修 复( repair),分 析( analyze)和优化(optimize)MySQL Server 中的表

mysqlcheck -u root -p123456 --all-databases

#mysql qps查询 QPS = Questions(or Queries) / Seconds

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Questions"'

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Queries"'

#mysql Key Buffer 命中率 key\_buffer\_read\_hits = (1 - Key\_reads / Key\_read\_requests) \* 100% key\_buffer\_write\_hits= (1 - Key\_writes / Key\_write\_requests) \* 100%

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Key%"'

#mysql Innodb Buffer 命中率 innodb\_buffer\_read\_hits=(1-Innodb\_buffer\_pool\_reads/Innodb\_buffer\_pool\_read\_requests) \* 100%

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Innodb\_buffer\_pool\_read%"'

#mysql Query Cache 命中率 Query\_cache\_hits= (Qcache\_hits / (Qcache\_hits + Qcache\_inserts)) \* 100%

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Qcache%"'

#mysql Table Cache 状态量

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Open%"'

#mysql Thread Cache 命中率 Thread\_cache\_hits = (1 - Threads\_created / Connections) \* 100% 正常来说,Thread Cache 命中率要在 90% 以上才算比较合理。

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Thread%"'

#mysql 锁定状态:锁定状态包括表锁和行锁两种,我们可以通过系统状态变量获得锁定总次数,锁定造成其他线程等待的次数,以及锁定等待时间信息

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "%lock%"'

#mysql 复制延时量 在slave节点执行

mysql -u root -p123456 -e 'SHOW SLAVE STATUS'

#mysql Tmp table 状况 Tmp Table 的状况主要是用于监控 MySQL 使用临时表的量是否过多,是否有临时表过大而不得不从内存中换出到磁盘文件上

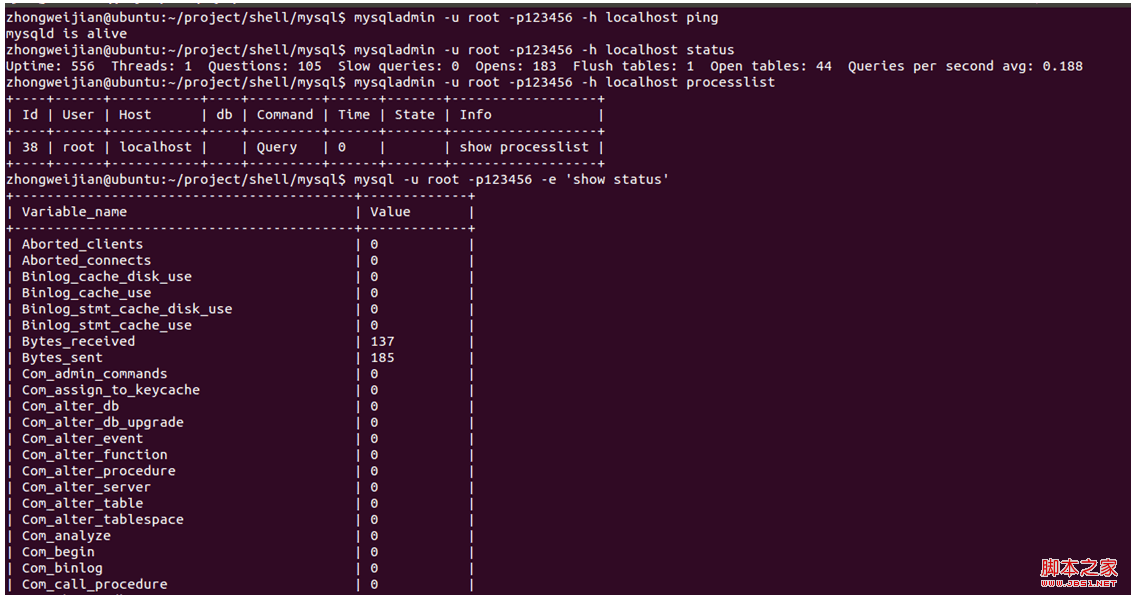
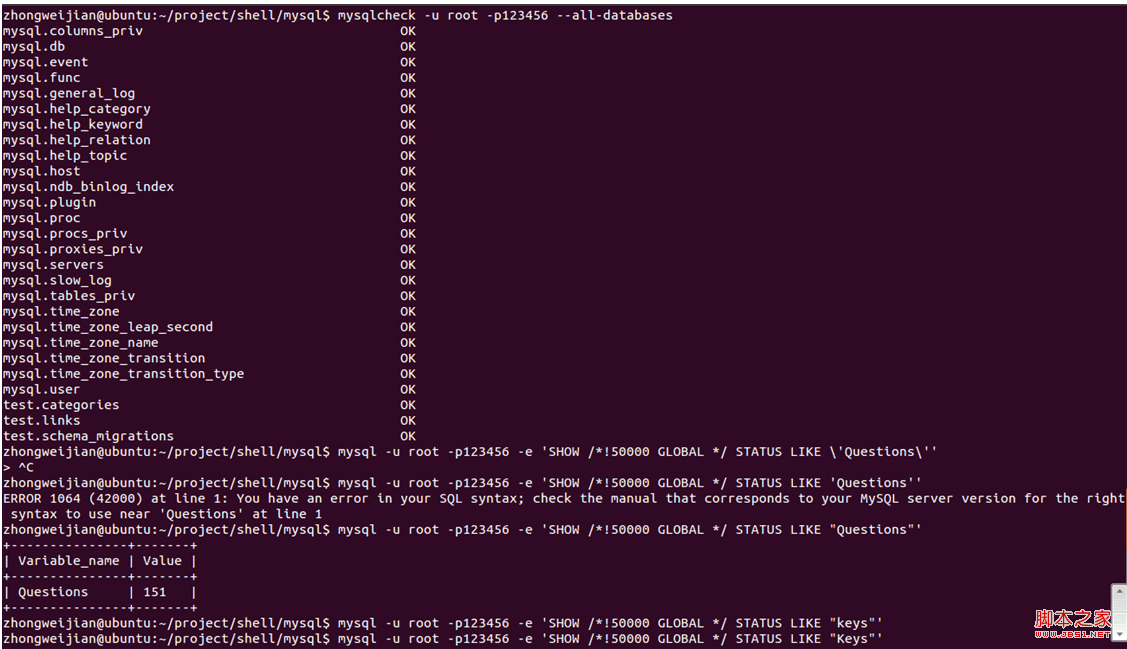
mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Created\_tmp%"'

#mysql Binlog Cache 使用状况:Binlog Cache 用于存放还未写入磁盘的 Binlog 信 息 。

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Binlog\_cache%"'

#mysql nnodb\_log\_waits 量:Innodb\_log\_waits 状态变量直接反应出 Innodb Log Buffer 空间不足造成等待的次数

mysql -u root -p123456 -e 'SHOW /\*!50000 GLOBAL \*/ STATUS LIKE "Innodb\_log\_waits'

[](http://files.jb51.net/file_images/article/201306/201306031600243.gif?20135316035)   
[](http://files.jb51.net/file_images/article/201306/201306031601324.gif?20135316146)

整理出来的常用的shell脚本会放到<https://github.com/zhwj184/shell-work> 这,主要包括。

shell-work  
shell常用分析命令和脚本  
mysql监控脚本  
cookielog分析脚本  
线上java进程信息dump和机器信息dump脚本  
cpu监控相关  
dish 磁盘管理

[脚本之家打包下载地址](http://xiazai.jb51.net/201605/yuanma/shell-work(jb51.net).rar)

# SQL语句技巧\_索引的优化\_慢查询日志开启\_root密码的破解

- 漂定 - 博客园

http://www.cnblogs.com/ahwu/p/3669068.html

1.正则表达式的使用 regexp  
例:  
select name,email from t where email regexp '@163[.,]com$'  
使用like方式查询  
selct name,email from t where email like '%@163.com' or email like '%@163,com'

2.巧用rand()提取随机行  
select \* from t3 order by rand() limit 3;

3.利用group by 的with rollup  
使用group by 的with rollup子句可以检索出更多的分组聚合信息  
注:with rollup 不可以和 order by 同时使用  
select cname,pname,count(pname) from demo group by cname,pname with rollup;

4.用bit group functions 做统计  
在使用group by 语句时可以同时使用bit\_and,bit\_or 函数来完成统计工作.这两个函数的作用主要是做数值之间的逻辑运算.  
select id,bit\_or(kind) from order\_rab group by id  
对order\_rab表中的id分组时对kind做位与和或计算  
select id,bit\_and(kind) from order\_rab group by id

-------------------------------------------------------  
优化SQL语句的一般步骤

1.通过show status命令了解各种sql的执行频率  
show [session|global] status;  
其中: session(默认)表示当前连接  
global 表示自数据库启动至今  
show status;  
show global status;  
show status like 'Com\_%';  
show global status like 'Com\_%';

参数说明:  
com\_select 执行select操作的次数,一次查询只累计加1  
com\_update 执行update操作的次数  
com\_insert 执行insert操作的次数,对批量插入只算一次  
com\_delete 执行delete操作的次数

针对Innodb存储引擎的  
innodb\_row\_read 执行select操作的次数  
innodb\_row\_updated 执行update操作的次数  
innodb\_row\_inserted 执行insert操作的次数  
innodb\_row\_deleted 执行delete操作的次数

其它:  
connections 连接mysql的数量  
uptime 服务器已经工作的秒数  
slow\_queries 慢查询的次数   
mysql>show variables like '%long%'; 看到long\_query\_time的值就是默认慢查询记录的时间(秒)

2.定位执行效率较低的SQL语句 explain 或 desc  
explain select \* from table; 或  
desc select \* from table;  
字段说明:  
select\_type:SIMPLE表示select类型,  
type:range 表示表的连接类型,性能有好到差  
system(表仅一行)  
const()只一行匹配)  
eq\_ref 对于前面的每一行使用主键和唯一  
...

3.使用like的查询,后面如果是常量并且只有%号不在第一个字符,索引才可能会被使用  
explain select \* from table where name like '%3'; 索引不会被使用  
explain select \* from table where name like '3%'; 使用

4.or 的前台条件字段都要加索引字段才会生效,否则整体不生效

5.查看索引使用情况

show status like 'Handler\_read%';  
Handler\_read\_key 这个值代表了一个行被索引值读的次数,  
Handler\_read\_rnd\_next 值高,则意叶着查询运行的低效.并且应该建立索引补救

6.慢查询日志  
log\_slow\_queries=slow.log # 慢查询日志的路径  
long\_query\_time=5 # 记录的时间  
重启mysql服务

7.socket问题  
有时候登陆mysql提示不能用socket登陆,此时可以换成tcp方式去登陆,但是必须要在php去用之前把这事情解决好  
mysql -uroot -proot --protocol tcp -hlocalhost

8.root密码丢失破解  
1.停止服务 pkill mysqld  
#跳过授权表mysql.user和mysql.db这些表  
2.mysqld\_safe --skip-grant-tables --user=mysql &

3.mysql -uroot 回车

4.update user set password=password('root') where user='root' and host='localhost';  
5.pkill mysqld  
6.重启mysql服务  
mysqld\_safe --user=mysql &

# mysql状态分析之show global status

- 小佛爷 - 博客园

http://www.cnblogs.com/what-/p/6824377.html

[mysql状态分析之show global status](http://www.cnblogs.com/what-/p/6824377.html)

公司的nagios监控服务器长期对内网用[MySQL数据库](http://lib.csdn.net/base/mysql)发出ctritical报警，因为我将其他同事的手机短信报警也开通了，搞得整个系统组的同事都怨声载道(呵呵)这时候就需要根据其status对其Mysql数据库进行优化了，这时候可以等MySQL服务器稳定运行了一段时间后运行，根据服务器的“状态”进行优化。  
mysql> show global status;  
可以列出MySQL服务器运行各种状态值，我个人较喜欢的用法是show status like '查询值%';  
**一、慢查询**  
mysql> show variables like '%slow%';  
+------------------+-------+  
| Variable\_name | Value |  
+------------------+-------+  
| log\_slow\_queries | ON |  
| slow\_launch\_time | 2     |  
+------------------+-------+  
mysql> show global status like '%slow%';  
+---------------------+-------+  
| Variable\_name    | Value |  
+---------------------+-------+  
| Slow\_launch\_threads | 0     |  
| Slow\_queries        | 4148 |  
+---------------------+-------+  
打开慢查询日志可能会对系统性能有一点点影响，如果你的MySQL是主－从结构，可以考虑打开其中一台从服务器的慢查询日志，这样既可以监控慢查询，对系统性能影响又小，另mysql有自带的命令mysqldumpslow可进行查询，例下列命令可以查出访问次数最多的20个sql语句mysqldumpslow -s c -t 20 host-slow.log

**二、连接数**  
经常会遇见”MySQL: ERROR 1040: Too manyconnections”的情况，一种是访问量确实很高，MySQL服务器抗不住，这个时候就要考虑增加从服务器分散读压力，另外一种情况是MySQL配置文件中max\_connections值过小：  
mysql> show variables like 'max\_connections';  
+-----------------+-------+  
| Variable\_name | Value |  
+-----------------+-------+  
| max\_connections | 256 |  
+-----------------+-------+  
这台MySQL服务器最大连接数是256，然后查询一下服务器响应的最大连接数：  
mysql> show global status like 'Max\_used\_connections';  
+----------------------+-------+  
| Variable\_name        | Value |  
+----------------------+-------+  
| Max\_used\_connections | 245 |  
+----------------------+-------+  
MySQL服务器过去的最大连接数是245，没有达到服务器连接数上限256，应该没有出现1040错误，比较理想的设置是：  
Max\_used\_connections / max\_connections   \* 100% ≈ 85%  
最大连接数占上限连接数的85％左右，如果发现比例在10%以下，MySQL服务器连接数上限设置的过高了。

**三、Key\_buffer\_size**  
key\_buffer\_size是对MyISAM表性能影响最大的一个参数，下面一台以MyISAM为主要存储引擎服务器的配置：  
mysql> show variables like 'key\_buffer\_size';  
+-----------------+------------+  
| Variable\_name | Value    |  
+-----------------+------------+  
| key\_buffer\_size | 536870912 |  
+-----------------+------------+  
分配了512MB内存给key\_buffer\_size，我们再看一下key\_buffer\_size的使用情况：  
mysql> show global status like 'key\_read%';  
+------------------------+-------------+  
| Variable\_name       | Value    |  
+------------------------+-------------+  
| Key\_read\_requests    | 27813678764 |  
| Key\_reads              | 6798830     |  
+------------------------+-------------+  
一共有27813678764个索引读取请求，有6798830个请求在内存中没有找到直接从硬盘读取索引，计算索引未命中缓存的概率：  
key\_cache\_miss\_rate ＝ Key\_reads / Key\_read\_requests \* 100%  
比如上面的数据，key\_cache\_miss\_rate为0.0244%，4000个索引读取请求才有一个直接读硬盘，已经很BT 了，key\_cache\_miss\_rate在0.1%以下都很好（每1000个请求有一个直接读硬盘），如果key\_cache\_miss\_rate在 0.01%以下的话，key\_buffer\_size分配的过多，可以适当减少。  
MySQL服务器还提供了key\_blocks\_\*参数：  
mysql> show global status like 'key\_blocks\_u%';  
+------------------------+-------------+  
| Variable\_name       | Value    |  
+------------------------+-------------+  
| Key\_blocks\_unused    | 0           |  
| Key\_blocks\_used        | 413543    |  
+------------------------+-------------+  
Key\_blocks\_unused 表示未使用的缓存簇(blocks)数，Key\_blocks\_used表示曾经用到的最大的blocks数，比如这台服务器，所有的缓存都用到了，要么 增加key\_buffer\_size，要么就是过渡索引了，把缓存占满了。比较理想的设置：Key\_blocks\_used / (Key\_blocks\_unused + Key\_blocks\_used) \* 100% ≈ 80%

**四、临时表**  
mysql> show global status like 'created\_tmp%';  
+-------------------------+---------+  
| Variable\_name           | Value |  
+-------------------------+---------+  
| Created\_tmp\_disk\_tables | 21197 |  
| Created\_tmp\_files    | 58    |  
| Created\_tmp\_tables    | 1771587 |  
+-------------------------+---------+  
每次创建临时表，Created\_tmp\_tables增加，如果是在磁盘上创建临时表，Created\_tmp\_disk\_tables也增加,Created\_tmp\_files表示MySQL服务创建的临时文件文件数，比较理想的配置是：  
Created\_tmp\_disk\_tables / Created\_tmp\_tables \* 100% <= 25%  
比如上面的服务器Created\_tmp\_disk\_tables / Created\_tmp\_tables \* 100% ＝ 1.20%，应该相当好了。我们再看一下MySQL服务器对临时表的配置：  
mysql> show variables where Variable\_name in ('tmp\_table\_size', 'max\_heap\_table\_size');  
+---------------------+-----------+  
| Variable\_name    | Value     |  
+---------------------+-----------+  
| max\_heap\_table\_size | 268435456 |  
| tmp\_table\_size    | 536870912 |  
+---------------------+-----------+  
只有256MB以下的临时表才能全部放内存，超过的就会用到硬盘临时表。

**五、Open Table情况**  
mysql> show global status like 'open%tables%';  
+---------------+-------+  
| Variable\_name | Value |  
+---------------+-------+  
| Open\_tables | 919 |  
| Opened\_tables | 1951   |  
+---------------+-------+  
Open\_tables 表示打开表的数量，Opened\_tables表示打开过的表数量，如果Opened\_tables数量过大，说明配置中 table\_cache(5.1.3之后这个值叫做table\_open\_cache)值可能太小，我们查询一下服务器table\_cache值：  
mysql> show variables like 'table\_cache';  
+---------------+-------+  
| Variable\_name | Value |  
+---------------+-------+  
| table\_cache | 2048   |

+---------------+-------+  
比较合适的值为：  
Open\_tables / Opened\_tables   \* 100% >= 85%  
Open\_tables / table\_cache \* 100% <= 95%

**六、线程使用情况**  
mysql> show global status like 'Thread%';  
+-------------------+-------+  
| Variable\_name     | Value |  
+-------------------+-------+  
| Threads\_cached | 46 |  
| Threads\_connected | 2     |  
| Threads\_created | 570 |  
| Threads\_running | 1     |  
+-------------------+-------+  
如果我们在MySQL服务器配置文件中设置了thread\_cache\_size，当客户端断开之后，服务器处理此客户的线程将会缓存起来以响应下一个客户 而不是销毁（前提是缓存数未达上限）。Threads\_created表示创建过的线程数，如果发现Threads\_created值过大的话，表明 MySQL服务器一直在创建线程，这也是比较耗资源，可以适当增加配置文件中thread\_cache\_size值，查询服务器 thread\_cache\_size配置：  
mysql> show variables like 'thread\_cache\_size';  
+-------------------+-------+  
| Variable\_name     | Value |  
+-------------------+-------+  
| thread\_cache\_size | 64 |  
+-------------------+-------+  
示例中的服务器还是挺健康的。

**七、查询缓存(query cache)**  
mysql> show global status like 'qcache%';  
+-------------------------+-----------+  
| Variable\_name           | Value     |  
+-------------------------+-----------+  
| Qcache\_free\_blocks    | 22756     |  
| Qcache\_free\_memory    | 76764704   |  
| Qcache\_hits          | 213028692 |  
| Qcache\_inserts       | 208894227 |  
| Qcache\_lowmem\_prunes | 4010916 |  
| Qcache\_not\_cached    | 13385031   |  
| Qcache\_queries\_in\_cache | 43560     |  
| Qcache\_total\_blocks     | 111212 |  
+-------------------------+-----------+  
MySQL查询缓存变量解释：  
Qcache\_free\_blocks：缓存中相邻内存块的个数。数目大说明可能有碎片。FLUSH QUERY CACHE会对缓存中的碎片进行整理，从而得到一个空闲块。  
Qcache\_free\_memory：缓存中的空闲内存。  
Qcache\_hits：每次查询在缓存中命中时就增大  
Qcache\_inserts：每次插入一个查询时就增大。命中次数除以插入次数就是不中比率。  
Qcache\_lowmem\_prunes： 缓存出现内存不足并且必须要进行清理以便为更多查询提供空间的次数。这个数字最好长时间来看；如果这个数字在不断增长，就表示可能碎片非常严重，或者内存 很少。（上面的 free\_blocks和free\_memory可以告诉您属于哪种情况）  
Qcache\_not\_cached：不适合进行缓存的查询的数量，通常是由于这些查询不是 SELECT 语句或者用了now()之类的函数。  
Qcache\_queries\_in\_cache：当前缓存的查询（和响应）的数量。  
Qcache\_total\_blocks：缓存中块的数量。

我们再查询一下服务器关于query\_cache的配置：  
mysql> show variables like 'query\_cache%';  
+------------------------------+-----------+  
| Variable\_name             | Value     |  
+------------------------------+-----------+  
| query\_cache\_limit          | 2097152 |  
| query\_cache\_min\_res\_unit     | 4096    |  
| query\_cache\_size          | 203423744 |  
| query\_cache\_type          | ON        |  
| query\_cache\_wlock\_invalidate | OFF    |  
+------------------------------+-----------+  
各字段的解释：  
query\_cache\_limit：超过此大小的查询将不缓存  
query\_cache\_min\_res\_unit：缓存块的最小大小  
query\_cache\_size：查询缓存大小  
query\_cache\_type：缓存类型，决定缓存什么样的查询，示例中表示不缓存 select sql\_no\_cache 查询  
query\_cache\_wlock\_invalidate：当有其他客户端正在对MyISAM表进行写操作时，如果查询在query cache中，是否返回cache结果还是等写操作完成再读表获取结果。  
query\_cache\_min\_res\_unit的配置是一柄”双刃剑”，默认是4KB，设置值大对[大数据](http://lib.csdn.net/base/hadoop)查询有好处，但如果你的查询都是小数据查询，就容易造成内存碎片和浪费。  
查询缓存碎片率 = Qcache\_free\_blocks / Qcache\_total\_blocks \* 100%  
如果查询缓存碎片率超过20%，可以用FLUSH QUERY CACHE整理缓存碎片，或者试试减小query\_cache\_min\_res\_unit，如果你的查询都是小数据量的话。  
查询缓存利用率 = (query\_cache\_size - Qcache\_free\_memory) / query\_cache\_size \* 100%  
查询缓存利用率在25%以下的话说明query\_cache\_size设置的过大，可适当减小；查询缓存利用率在80％以上而且Qcache\_lowmem\_prunes > 50的话说明query\_cache\_size可能有点小，要不就是碎片太多。  
查询缓存命中率 = (Qcache\_hits - Qcache\_inserts) / Qcache\_hits \* 100%  
示例服务器 查询缓存碎片率 ＝ 20.46％，查询缓存利用率 ＝ 62.26％，查询缓存命中率 ＝ 1.94％，命中率很差，可能写操作比较频繁吧，而且可能有些碎片。

**八、排序使用情况**  
mysql> show global status like 'sort%';  
+-------------------+------------+  
| Variable\_name     | Value    |  
+-------------------+------------+  
| Sort\_merge\_passes | 29       |  
| Sort\_range        | 37432840 |  
| Sort\_rows       | 9178691532 |  
| Sort\_scan       | 1860569 |  
+-------------------+------------+  
Sort\_merge\_passes 包括两步。MySQL 首先会尝试在内存中做排序，使用的内存大小由系统变量Sort\_buffer\_size 决定，如果它的大小不够把所有的记录都读到内存中，MySQL 就会把每次在内存中排序的结果存到临时文件中，等MySQL 找到所有记录之后，再把临时文件中的记录做一次排序。这再次排序就会增加 Sort\_merge\_passes。实际上，MySQL会用另一个临时文件来存再次排序的结果，所以通常会看到 Sort\_merge\_passes增加的数值是建临时文件数的两倍。因为用到了临时文件，所以速度可能会比较慢，增加 Sort\_buffer\_size 会减少Sort\_merge\_passes 和 创建临时文件的次数，但盲目的增加Sort\_buffer\_size 并不一定能提高速度

**九、文件打开数(open\_files)**  
mysql> show global status like 'open\_files';  
+---------------+-------+  
| Variable\_name | Value |  
+---------------+-------+  
| Open\_files | 1410   |  
+---------------+-------+  
mysql> show variables like 'open\_files\_limit';  
+------------------+-------+  
| Variable\_name | Value |  
+------------------+-------+  
| open\_files\_limit | 4590   |  
+------------------+-------+  
比较合适的设置：Open\_files / open\_files\_limit \* 100% <= 75％

**十、表锁情况**  
mysql> show global status like 'table\_locks%';  
+-----------------------+-----------+  
| Variable\_name       | Value     |  
+-----------------------+-----------+  
| Table\_locks\_immediate | 490206328 |  
| Table\_locks\_waited | 2084912 |  
+-----------------------+-----------+  
Table\_locks\_immediate 表示立即释放表锁数，Table\_locks\_waited表示需要等待的表锁数，如果Table\_locks\_immediate / Table\_locks\_waited >5000，最好采用InnoDB引擎，因为InnoDB是行锁而MyISAM是表锁，对于高并发写入的应用InnoDB效果会好些。示例中的服务 器Table\_locks\_immediate / Table\_locks\_waited ＝ 235，MyISAM就足够了。

**十一、表扫描情况**  
mysql> show global status like 'handler\_read%';  
+-----------------------+-------------+  
| Variable\_name       | Value    |  
+-----------------------+-------------+  
| Handler\_read\_first | 5803750     |  
| Handler\_read\_key    | 6049319850   |  
| Handler\_read\_next     | 94440908210 |  
| Handler\_read\_prev     | 34822001724 |  
| Handler\_read\_rnd    | 405482605 |  
| Handler\_read\_rnd\_next | 18912877839 |  
+-----------------------+-------------+  
mysql> show global status like 'com\_select';  
+---------------+-----------+  
| Variable\_name | Value     |  
+---------------+-----------+  
| Com\_select | 222693559 |  
+---------------+-----------+  
计算表扫描率：  
表扫描率 ＝ Handler\_read\_rnd\_next / Com\_select  
如果表扫描率超过4000，说明进行了太多表扫描，很有可能索引没有建好，增加read\_buffer\_size值会有一些好处，但最好不要超过8MB。

分类: [mysql](http://www.cnblogs.com/what-/category/998075.html)

# MySQL :: MySQL 5.7 Reference Manual :: 5.1.7 Server Status Variables

https://dev.mysql.com/doc/refman/5.7/en/server-status-variables.html#statvar\_Innodb\_buffer\_pool\_bytes\_dirty

# Redis监控主要指标及采集方法

\_数据库技术\_Linux公社-Linux系统门户网站

http://www.linuxidc.com/Linux/2016-11/136783.htm

公司的Redis业务很多，Redis监控自然也是DB监控的一大模块，包括采集、展示、监控告警。本文主要介绍Redis监控的主要指标和采集方法。

　　一、Redis监控系统逻辑

　　1、DBA通过前台页面添加redis监控，填写ip和端口，配置阈值、负责人等信息

　　2、前台调用自动调度平台接口将redis监控采集Agent发送到目标机器上的固定文件夹，并添加crond，每分钟运行（采集Agent包含采集程序和ip端口信息文件）

　　3、redis监控Agent采集相关指标通过http接口上报到mysql服务器（目标机器无法直接连接mysql）

　　4、前台读取mysql数据进行展示

　　5、解析程序每分钟通过读取配置信息和Agent上报的数据进行解析并发送告警（Rtx/Wechat/Sms）给指定负责人

　　二、采集指标和命令

　　1、redis服务进程 ip-port

约定所有redis服务都必须以ip1（内网ip）来绑定，每个机器只有一个ip1，可以有多个端口，即多个redis实例。采集程序读取ip端口信息文件来判断有多少个实例

ps aux | grep -E "redis-server.\*$port"

约定所有redis客户端安装标准路径是

REDISPATH\_CLI="/usr/local/redis/bin/redis-cli"

如果发现不是标准路径会有 No such file or directory 的提示，则采集程序会在每天早上10点发送Rtx弹窗给DBA（DBA也可以根据前台页面图表展示来判断该采集器有无上报数据，无上报数据则可以上机查看采集器日志）

采集器使用下面的命令来采集redis实例的所有信息，然后根据信息筛选出需要的指标上报到mysql

$REDISPATH\_CLI -h ${agentIp} -p ${port} info > ${tmpFile} 2>&1

如果redis实例没有绑定在ip1或者127.0.0.1上面则会提示 Connection refused

　　2、连接客户数

grep "connected\_clients:" ${tmpFile} | awk -F ":" '{print $2}'

　　3、阻塞连接数

grep "blocked\_clients:" ${tmpFile} | awk -F ":" '{print $2}'

　　4、redis占用内存，单位Byte转成MB

grep "used\_memory:" ${tmpFile} | awk -F ":" '{print $2}' | awk '{printf "%.2f",$1/1024/1024}'

　　5、内存峰值，单位Byte转成MB

grep "used\_memory\_peak:" ${tmpFile} | awk -F ":" '{print $2}' | awk '{printf "%.2f",$1/1024/1024}'

　　6、主从角色

grep "role:" ${tmpFile} | awk -F ":" '{print $2}' # master（主），slave（从）

　　7、master\_link\_status

grep "master\_link\_status:" ${tmpFile} | awk -F ":" '{print $2}' # up down

down:**Master已经不可访问了，Slave依然运行良好，并且保留有AOF与RDB文件**

　　8、执行命令总数和qps

grep "total\_commands\_processed:" ${tmpFile} | awk -F ":" '{print $2}'

计算qps需要计算两次 total\_commands\_processed，然后除以时间差。逻辑是第一分钟将total\_commands\_processed的值和当时采集该值的时间保存到last.cache中，第二分钟采集的时候获取值和时间，和上次相减得到两个差值相除即可。

　　9、上报时间

reportTime=`date +'%Y-%m-%d %H:%M:00'`

时间的秒数是0，方便前台画图展示

　　三、解析告警

分析上报的指标是否异常，根据reportTime判断Agent心跳情况，将状态和异常信息存入mysql，发送告警给相关负责人。数据库分表如下：

　　redisMonitor库：配置表redisConf、当前状态redisCurrentStatus、异常信息表redisDisplay

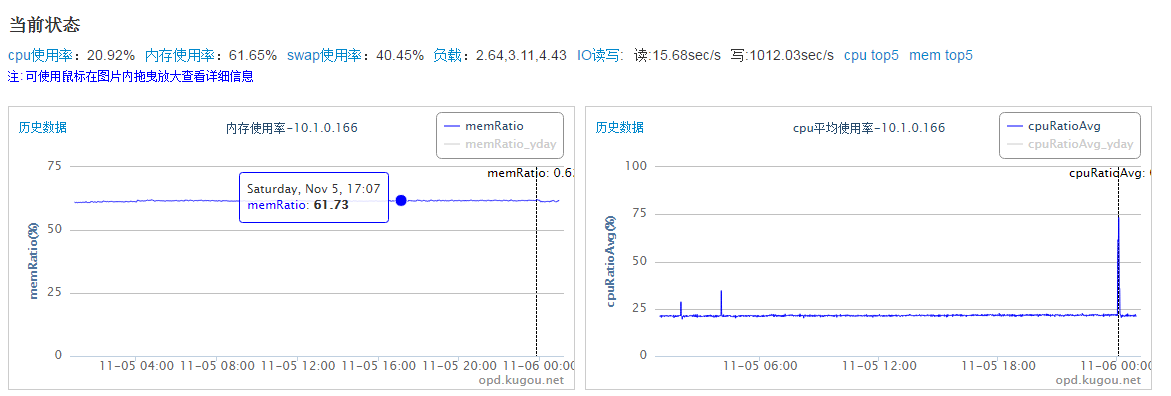
　　历史库按月分库，按照ip分表，保存redis采集器上报数据，Ex: redisStatus167872716

　　四、前台展示

　　1、异常页面



　　2、状态页面



**下面关于Redis的文章您也可能喜欢，不妨参考下：**

[**Ubuntu**](http://www.linuxidc.com/topicnews.aspx?tid=2)**14.04下Redis安装及简单测试** <http://www.linuxidc.com/Linux/2014-05/101544.htm>

Redis主从复制基本配置 <http://www.linuxidc.com/Linux/2015-03/115610.htm>

Redis集群明细文档 <http://www.linuxidc.com/Linux/2013-09/90118.htm>

Ubuntu 16.04环境中安装PHP7.0 Redis扩展 <http://www.linuxidc.com/Linux/2016-09/135631.htm>

[CentOS](http://www.linuxidc.com/topicnews.aspx?tid=14) 7.0 安装Redis 3.2.1详细过程和使用常见问题 <http://www.linuxidc.com/Linux/2016-09/135071.htm>

Ubuntu 16.04环境中安装PHP7.0 Redis扩展 <http://www.linuxidc.com/Linux/2016-09/135631.htm>

Ubuntu 15.10下Redis集群部署文档 <http://www.linuxidc.com/Linux/2016-06/132340.htm>

Redis实战 中文PDF <http://www.linuxidc.com/Linux/2016-04/129932.htm>

**Redis 的详细介绍**：[请点这里](http://www.linuxidc.com/Linux/2012-04/59269.htm)   
**Redis 的下载地址**：[请点这里](http://www.linuxidc.com/down.aspx?id=627)

# MySQL 内存和CPU优化相关的参数 - 艾森豪威迩 - 博客园

http://www.cnblogs.com/xiaoit/p/4551676.html

mysql> SHOW GLOBAL STATUS LIKE 'innodb%read%';

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

| Variable\_name                         | Value   |

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

| Innodb\_buffer\_pool\_read\_ahead\_rnd     | 0       |

| Innodb\_buffer\_pool\_read\_ahead         | 0       |

| Innodb\_buffer\_pool\_read\_ahead\_evicted | 0       |

| Innodb\_buffer\_pool\_read\_requests      | 612     |

| Innodb\_buffer\_pool\_reads              | 179     |

| Innodb\_data\_pending\_reads             | 0       |

| Innodb\_data\_read                      | 5115904 |

| Innodb\_data\_reads                     | 189     |

| Innodb\_pages\_read                     | 178     |

| Innodb\_rows\_read                      | 0       |

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

10 rows in set (0.00 sec)

#多核CPU

innodb\_read\_io\_threads

innodb\_write\_io\_threads

#内存

缓冲池命中率=

Innodb\_buffer\_pool\_read\_requests/(

Innodb\_buffer\_pool\_read\_requests+Innodb\_buffer\_pool\_read\_ahead+Innodb\_buffer\_pool\_reads)

平均每次读取字节数

Innodb\_data\_read/Innodb\_data\_reads

Innodb\_data\_reads:表示从物理磁盘读取页的次数

Innodb\_buffer\_pool\_read\_ahead：预读的次数

Innodb\_buffer\_pool\_read\_ahead\_evicted：预读的页，但是没有被读取就从缓冲池中被替换的页的数量，一般用来判断预读的频率

Innodb\_buffer\_pool\_read\_requests：从缓冲池中读取页的次数

Innodb\_data\_read：一共读取的字节数

Innodb\_data\_reads：发起读取请求的次数，每次读取可能需要读取多个页

# 中国移动MySQL数据库优化经验 - 推酷

http://www.tuicool.com/articles/veeAf2b

时间 2016-09-07 09:31:49  [程序师](http://www.tuicool.com/sites/qaUfmy)

原文  [http://www.techug.com/china-mobile-mysql-optimization-tips](http://www.techug.com/china-mobile-mysql-optimization-tips?utm_source=tuicool&utm_medium=referral)

主题 [MySQL](http://www.tuicool.com/topics/11030000)

开源数据库MySQL比较容易碰到性能瓶颈，为此经常需要对MySQL数据库进行优 化，而MySQL数据库优化需要运维DBA与相关开发共同参与，其中MySQL参数及服务器配置优化主要由运维DBA完成，开发则需要从数据类型优化，索引优化，SQL优化三个角度考虑MySQL数据库优化问题。

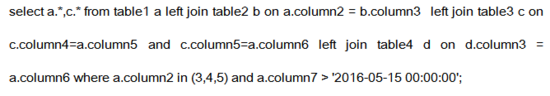
本次分享将从开发角度，看如何实现MySQL数据库优化。

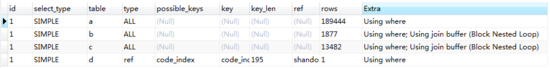
本次分享大纲：

* 一个例子
* 数据类型优化
* 索引优化
* SQL优化

[***程序员高效开发利器：编程水杯***](http://begeek.cn/post/programming-mug.html)

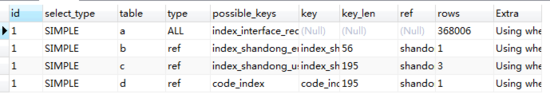
一、一个例子





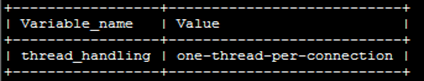
数据库需要处理的行数： 189444\*1877\*13482~~~479亿

如果在关联字段上加上合适的索引：



数据库需要处理的行数：368006\*1\*3\*1~~~110万

MySQL通常是一个请求对应一个线程，其thread\_handling是one-thread-per-connection，因此一条sql请求只能利用一个CPU



通过加索引，数据库需要处理的行数下降了4个数量级，第一种情况下等待半小时不一定能跑出结果，但第二种情况可以在秒级范围内拿到需要的结果。从该例子可以看出，MySQL数据库优化非常重要，一条不合理的SQL就可能导致服务异常。

开发需要掌握查看MySQL执行计划及profile工具：

1. EXPLAIN  SELECT ……
2. EXPLAIN EXTENDED SELECT ……
3. profile工具

SET profiling = 1;

show profiles;

— 显示最近发送的mysql服务的sql语句

show profile;

— 显示最近的单个SQL语句的详细过程信息

show profile all for query 61;

— 显示所有相关信息

二、数据类型优化

选择数据类型的步骤：

* Step1：确定合适的大类型，如数字、字符串、时间等；
* Step2：选择具体类型，相同大类型的不同子类型数据的存储长度，范围，允许的精度不同，有时候也有一些特殊的行为和属性。

普遍适用的原则：

* 使用小而简单的合适的数据类型；
* 对于可变长字符串VARCHAR，只分配真正需要的空间；
* 小心使用ENUM;
* 尽量使用整型定义标识列；
* 使用相同数据类型存储相似或者相关的值，尤其是关联条件中使用的列。

核心原则：具体问题具体分析。一些特定的业务场景并不适合套用普遍使用的原则。

>>>>

使用小而简单的合适的数据类型：

* Case1：如果只需要存0-200，tinyint unsigned更好。因为更小的数据类型所需的磁盘，内存和CPU缓存更少，处理时需要的CPU周期也更少。
* Case2：用INT代替varchar(15)来存储IP地址。因为字符集和校对规则（排序规则）使字符比较比整型比较更复杂。
* Case3：使用MySQL内建的类型（date, time, datetime等）而不是字符串来存储日期和时间。
* Case4：用char存储密码的MD5值，因为密码的MD5是一个定长的值。

>>>>

对于可变长字符串VARCHAR，只分配真正需要的空间：

使用VARCHAR(4)和VARCHAR(200)存储‘ZYHY’的空间开销是一样的，但使用更短的列VARCHAR(4)有如下优势：

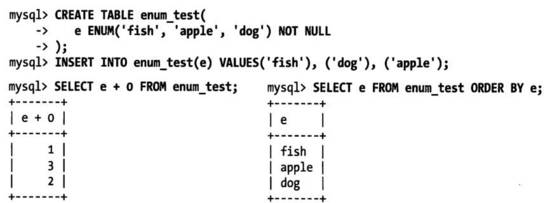
因为MySQL通常会分配固定大小的内存块来保存内部值，所以更长的列会消耗更多的内存，在使用内存临时表进行排序或者操作时会特别糟糕，利用磁盘临时表进行排序时也同样糟糕。

所以，建议只分配真正需要的空间。

>>> **>**

小心使用ENUM

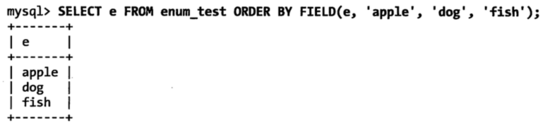
MySQL 在存储ENUM枚举时非常紧凑，会根据列表值的数量压缩到一个或者两个字节中。MySQL在内部会将每个值在列表中的位置保存为整数，并且在表的.frm 文件中保存“数字-字符串”映射关系的“查找表”。枚举字段是按照内部存储的整数而不是定义的字符串进行排序。



从上图中的select e + 0 from enum\_test;的结果可以看出，MySQL在内部会将每个值在列表中的位置保存为整数，可以与整数进行算术运算。

从上图中的select e from enum\_test order by e;的结果可以看出，排序结果与建表时的顺序一致，如果需要按字符创的字母顺序排序，则需要通过额外的方法来处理，比如：

1. 按照需要的顺序来定义枚举列；
2. 在查询中使用FIELD()函数显示地指定排序顺序，但这会导致MySQL无法利用索引消除排序。



与VARCHAR相比，ENUM优势与劣势：

1. 优势：数据紧凑，存储的是整数，占用空间小，作为关联字段时，效率比varchar类型高很多；
2. 劣 势：字符串列表是固定的，添加或者删除字符串必须使用ALTER TABLE,如果添加的字符串不在列表末尾，则需要重建整个表完成修改。由于ENUM保存为整数，必须进行查找才能转换为字符串，在需要转换为字符串时有 一些开销。在一些特定情况下，把varchar列和枚举列进行关联可能比varchar自关联更慢。

>>>>

尽量使用整型定义标识列

* 因为整形数据的执行计算和比较都很快；
* 不建议使用UUID等随机字符串作为标识列，因为随机字符串会任意分布在很大的空间，导致INSERT和SELECT语句变得很慢。

>>>>

使用相同数据类型存储相似或者相关的值，尤其是关联条件中使用的列

* 因为混用不同的数据类型可能导致性能问题，在关联条件中会有数据类型转换的资源消耗；
* 在比较操作时隐形类型转换可能导致很难发现的错误。

>> **>>**

关于整数类型指定宽度的一个解释

MySQL可以为整数类型指定宽度，如INT(11),但对大多数应用来说，这并没有什么意义：它不会限制值的合法范围，只是规定了MySQL的一些交互工具（例如MySQL命令行客户端）用来显示字符的个数。对于存储和计算来说，INT(1) 和INT(20)是相同的。

**>>>>**

关于实数类型

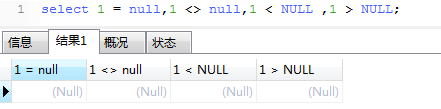
1. MySQL既支持精确类型（decimal, numeric），也支持不精确类型(float, double)。
2. 可以使用DECIMAL存储比BIGINT还大的整数。
3. CPU不支持对DECIMAL的直接计算，而是MySQL服务器自身对DECIMAL进行高精度计算。而CPU直接支持原生浮点运算，所以，浮点运算明显更快。
4. 可以考虑使用BIGINT代替DECIMAL，将需要存储的值根据小数的位数乘以相应的倍数即可，如精确到0.01，则把所有值乘以100存储到BIGINT中，这样可以同时避免浮点存储计算不精确和DECIMAL精确计算代价高的问题。

**>>>>**

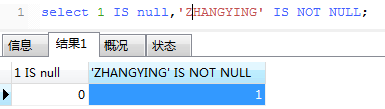
关于NULL的定义:

a missing unknown value, means “not having a value.”

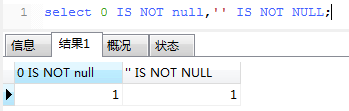
与NULL的任何数学运算的结果还是NULL



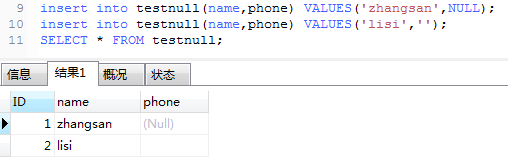
判断值是否等于NULL，不能简单用=，而要用IS NULL/ IS NOT NULL



0和空字符串都不是NULL:



NULL与空字符串的区别



上图中分别insert了一个NULL和一个空字符创，其表达的意义不一样：

* INSERT a NULL:不知道这个人有没有电话号码；
* INSERT a ‘’: 确定这个人没有电话号码；
* COUNT(table.column), MIN(), and SUM() 会忽略NULL ，count(\*)会计算包含NULL的所有行

三、索引优化

>>>>

索引类型

按数据存储方式分类：

* 聚簇索引：数据行实际上存放在索引的叶子（leaf page）页中。即数据行和相邻的键值紧凑地存储在一起。
* 二级索引（非聚簇索引）：二级索引的叶子节点包含了引用行的主键列（它不指向行的物理位置，而是行的主键值）。二级索引需要两次索引查找，而不是一次。（对于InnoDb，自适应哈希索引能够减少这样的重复工作）

按索引的数据结构分类：

* B-TREE索引
* 哈希索引
* 空间数据索引（R-TREE)
* 全文索引

InnoDB主键索引结构：

在InnoDB中，表数据文件本身就是按B+Tree组织的一个索引结构，这棵树的叶节点data域保存了完整的数据记录。这个索引的key是数据表的主键，因此InnoDB表数据文件本身就是主索引。

InnoDB非主键索引：

InnoDB的辅助索引data域存储相应的记录值及该记录对应的主键的值而不是地址。

>>>>

索引策略

* 经常与其他表进行关联的表，在关联字段上应该建立索引；
* 经常出现在Where子句中的字段，特别是大表的字段，应该建立索引；
* 频繁进行数据操作的表，不要建立太多的索引，数据的插入，更新和删除会对索引产生影响，太多的索引会导致插入更新删除操作缓慢；
* 索引应该建在选择性高的字段上Cardinality/rows尽可能等于1。Show index命令查看Cardinality（索引列去重后的行数）。
* 索引应该建在小字段上，整数字段尤其适合，对于大的文本字段甚至超长字段，不要建索引，或者建立前缀索引， 如create index 索引名 on 表名(列名1 (指定长度)，……)
* 删除无用的索引，如重复索引，不必要的冗余索引；
* 针对组合索引，设计合理的索引列顺序

下面介绍一些与索引相关的概念。

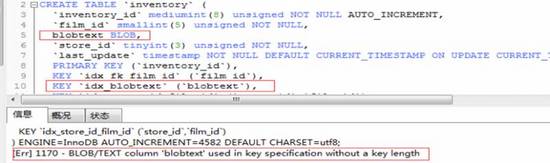
前缀索引：索引开始的部分字符，以节约索引空间，提高索引效率。

风险：会降低索引的选择性。

对于BLOB，text或者很长的varchar类型的列，必须使用前缀索引。

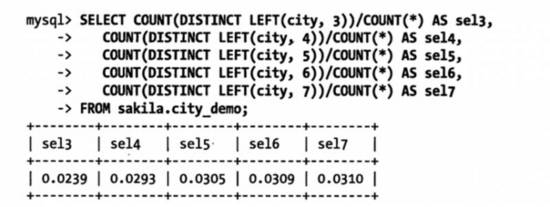
否则会报错：

[Err] 1170 – BLOB/TEXT column ‘blobtext’ used in key specification without a key length



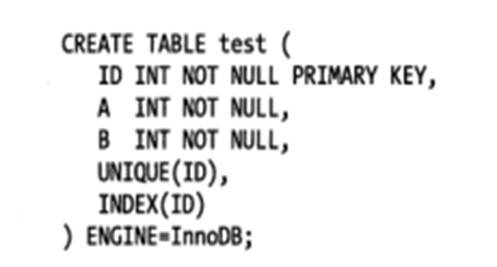
前缀索引的长度有一个权衡点：选择足够长的前缀以保证较高 的选择性，同时又不能太长。

那么如何计算不同前缀长度的选择性：



查询显示当前缀长度到达7的时候，再增加前缀长度，选择性提升的幅度已经很小。

重复索引：指在相同列上按照相同顺序创建相同类型的索引。 （SQL摘抄自《高性能MySQL》）



相当于建了三个重复索引。

MySQL需要单独维护重复索引，优化器在优化查询的时候也需要逐个进行考虑，因此 重复索引会影响性能。

冗余索引：

* Case1: 如创建了索引（A,B），再创建索引（A），则产生了冗余索引，因为索引（A）只是索引（A,B）的前缀索引。
* Case2: 索引（A），再创建索引（A,ID），其中ID是主键，对于InnoDB来说主键列已经包含在二级索引中了，所以这也是冗余。

什么时候需要冗余索引？

当扩展已有的索引会导致其变得太大，从而影响其他使用该索引的查询性能。

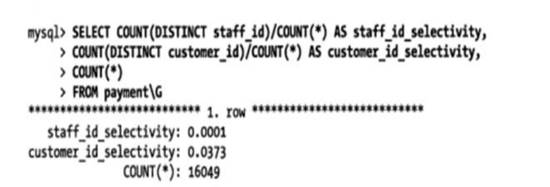
比如，在整数列上有一个索引，现在需要增多一个VARCHAR列来扩展该索引，此时，如果使用整数列与varchar列的组合索引比单独使用整数列的索引的效率要慢很多，因此，此时可以考虑冗余索引，以满足不同场景下的query需求。

索引列顺序：

在多列B-tree索引中，索引列的顺序意味着索引首先按照最左列进行排序，其次是第二列，…

建议将选择性最高的列放在索引最左列。

如何确定选择性更高的字段： （SQL摘抄自《高性能MySQL》）



发现customer\_id的选择性更高。

索引列顺序建议为(customer\_id, staff\_id)。

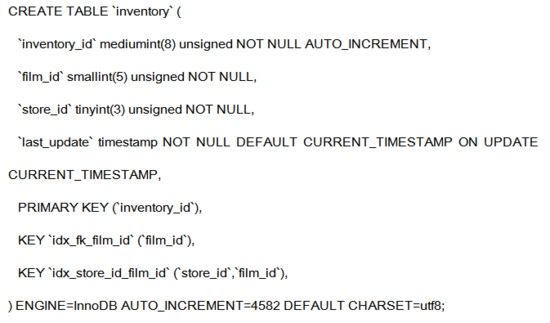
覆盖索引

索引包含（或者说覆盖）所有需要查询字段的值。

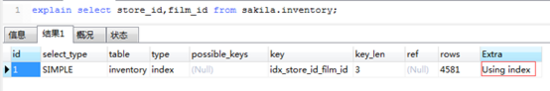
优势：

* 只需要读取索引，就可以访问到数据
* 索引按照列值顺序存储，顺序查询比随机io要快。

案例：



当发起一个被索引覆盖的查询时，在explain的extra列可以看到“Using index”的信息。



不能使用索引的场景

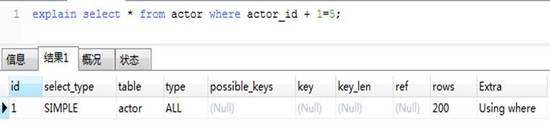
在一些场景下，索引不能生效，比如：

* 使用LIKE或者REGEXP时，以%开头，即“%\*\*\*”时；
* 在字段使用函数时；
* 在join时条件字段类型不一致时；
* 在组合索引里使用非第一个索引时；
* 使用!=以及<>不等于时；
* 索引列不独立时。

四、SQL优化

Where **子句中使用独立的列：**

查询中列如果不是独立的，则不会使用索引。



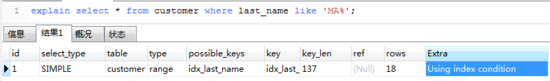
关联查询优化：

* 确保ON或者USING子句的列上有索引。一般只需要在关联顺序中的第二个表的相应列上创建索引。
* 关联字段类型保持一致。

LIKE **匹配优化：**

如果 LIKE 的参数是非通配字符开始的固定字符串，MySQL在做LIKE比较时也可能用到索引。

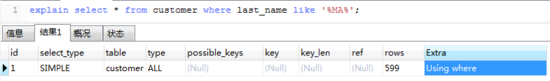
select \* from customer where last\_name like ‘MA%’;



Extra信息中显示使用了索引。

like后面使用通配符开始的字符串则不会使用索引

select \* from customer where last\_name like ‘%MA%’;



rows列显示599行，也就是customer表的总行数，因此没利用到索引。

避免SQL中出现不必要的类型转换：

select \* from charge\_record where phone=13990055761;



select \* from charge\_record where phone=‘13990055761’;



Select **指定列来代替select \*：**

* 在某些情况下 select \*  要比select 指定列 需要浪费更多的资源
* 如果某些列中含有text等类型，select 指定列可以减少网络传输缓冲区的使用
* 如果SQL中含有order by ,并且排序不能利用上已用的索引那么，额外的字段会占用更多的sort\_buffer\_size .
* Select指定列可以方便使用覆盖索引。

比如下面这个例子，使用到了覆盖索引。



子查询优化：

* MySQL5.6前，子查询大多时候会先遍历outer table，对于其返回的每一条记录都执行一次subquery，而且子查询没有任何索引，导致子查询相较于关联查询要慢很多（解决方案：表连接代替子查询）；
* MySQL5.6 后，对子查询进行了大幅度的优化，将子查询结果存入临时表，使得子查询只执行一次，而且优化器还会给子查询产生的派生表添加索引，使得子查询性能得到了强劲的优化。

曾经的“绝对真理”：子查询比关联查询慢很多。——不再成立。

通过子查询优化可以减少多个查询多次对数据进行访问。

但也有时候，子查询可能比关联查询还要快。

>>>>

GROUP BY优化：

表的标识列分组比其他列分组的效率高。

SELECT actor.first\_name, actor.last\_name, count(\*) FROM film\_actor INNER JOIN actor USING (actor\_id) GROUP BY actor.first\_name, actor.last\_name;

http://img1.tuicool.com/YN3MZvq.jpg!web

优化后：

SELECT actor.first\_name, actor.last\_name,count(\*) FROM film\_actor

INNER JOIN actor USING (actor\_id) GROUP BY actor.actor\_id ;

http://img2.tuicool.com/zAzue2Q.jpg!web

因为actor.actor\_id是主键，分组效率会提升。

使用GROUP BY子句时，结果集会自动按照分组的字段进行排序，GROUP BY子句中可以直接使用DESC或者ASC关键字，使得分组的结果集按需要的方向排序。

So：如果没有排序需求，可以加ORDER BY NULL,让MySQL不再进行文件排序，从而提高查询效率。

>>>>

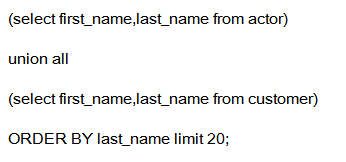
UNION优化：

除非需要消除重复的行，否则一定要使用union all，因为没有ALL关键字，MySQL会给临时表加上DISTINCT选项，使得对整个临时表做代价很高的唯一性检查。

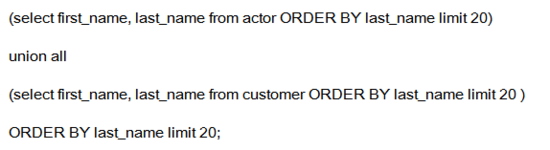
由于union产生的临时表无法使用优化器的优化策略，所以可以直接将WHERE, ORDER BY, LIMIT等子句冗余的写一份到各个子查询中。

案例：





如果把ORDER BY, LIMIT等子句冗余写一份到各个子查询中。



则排序的基数会有效的得到降低，从而提高效率。

**参考文献** ： 《高性能MySQL》

# syshacklog/innodb调优.md at master · syshack/syshacklog

https://github.com/syshack/syshacklog/blob/master/innodb%E8%B0%83%E4%BC%98.md

mysql调优（innodb引擎）

对性能影响最大的2个参数：

innodb\_buffer\_pool\_size

InnoDB最重要的设置，这个参数确定了要预留多少内存来缓存表数据和索引,对InnoDB性能有决定性的影响，在内存允许的情况下设置比InnoDB tablespaces大10%的内存大小。建议：尽可能大

innodb\_flush\_logs\_at\_trx\_commit 经常配合 sync\_binlog设置

可选项为0，1，2，安全级别为 0 < 2 < 1

默认为1，每进行一次事务，就刷新一次日志。

为2时，在每个事务提交时，日志缓冲被写到文件，但不对日志文件做到磁盘操作的刷新。

为0时，日志缓冲每秒一次地被写到日志文件，并且对日志文件做到磁盘操作的刷新。任何mysqld进程的崩溃会删除崩溃前最后一秒的事务。

sync\_binlog=N

N>0 — 每向二进制日志文件写入N条SQL或N个事务后，则把二进制日志文件的数据刷新到磁盘上；

N=0 — 不主动刷新二进制日志文件的数据到磁盘上，而是由操作系统决定；

推荐配置组合： N=1,1 安全性非常高，磁盘IO写能力足够支持业务，比如充值消费系统；

N=1,0 安全性高，磁盘IO写能力支持业务不富余，允许备库落后或无复制；

N=2,0或2 安全性一般，允许丢失一点事务日志，复制架构的延迟也能接受；

N=0,0 安全性差，磁盘IO写能力有限，无复制或允许复制延迟稍微长点能接受，例如：日志性登记业务；

内存相关

innodb\_additional\_mem\_pool\_size

一般设置到16M就可以了，如果需要增加mysql错误日志会记录。

IO/文件相关参数

innodb\_file\_per\_table

启用单表空间，减少共享表空间维护成本，减少空闲磁盘空间释放的压力。大数据量情况下会有性能上的提升，为此建议大家使用独立表空间代替共享表空间的方式；

innodb\_flush\_method

设置InnoDB同步IO的方式：

1.Default – 使用fsync（）

2.O\_SYNC 以sync模式打开文件，通常比较慢。

3.O\_DIRECT，在Linux上使用Direct IO。可以显著提高速度，特别是在RAID系统上。避免额外的数据复制和double buffering（mysql buffering 和OS buffering）。

innodb\_max\_dirty\_pages\_pct

控制Innodb的脏页在缓冲中在那个百分比之下，值在范围1-100,默认为90，90性能最好，不过重启恢复时间最长。

缓存相关

binlog\_cache\_size

一般场景2-4M，可以通过binlog\_cache\_use 以及 binlog\_cache\_disk\_use来分析设置的binlog\_cache\_size是否足够，是否有大量的binlog\_cache由于内存大小不够而使用临时文件（binlog\_cache\_disk\_use）来缓存了。

bulk\_insert\_buffer\_size:

如果经常性的需要批量插入大量数据，可以适当调大，原则上不超过32M。

日志相关参数

innodb\_log\_buffer\_size

此参数确定些日志文件所用的内存大小，以M为单位。缓冲区更大能提高性能，但意外的故障将会丢失数据.开发者建议设置为1－8M之间

innodb\_log\_files\_in\_group

为提高性能，MySQL可以以循环方式将日志文件写到多个文件。推荐设置为3

innodb\_log\_file\_size

确定数据日志文件的大小，以M为单位，更大的设置可以提高性能，但也会增加恢复故障数据库所需的时间,2G内为佳。

可以使用：show status like 'Innodb\_os\_log\_written'; select sleep(60); show status like 'Innodb\_os\_log\_written';

查看每分钟写多大，然后确定此值的大小。

连接相关

max\_connect\_errors

默认值为10，也即mysqld线程没重新启动过，一台物理服务器只要连接异常中断累计超过10次，就再也无法连接上mysqld服务，为此建议大家设置此值至少大于等于10W；

wait\_timeout

与服务器端无交互状态的连接超时时间。一般建议设置为：172800（48小时），维护连接建议客户端来做，用完及时关闭连接，避免长连接。