

# MySQL数据库优化

1. Mysql基础操作
2. 常用的Sql技巧
3. Sql语句优化
4. Mysql服务器优化

# 一、Mysql基础操作



1.1 mysql表复制

1.2 mysql索引

1.3 mysql视图

1.4 mysql内置函数

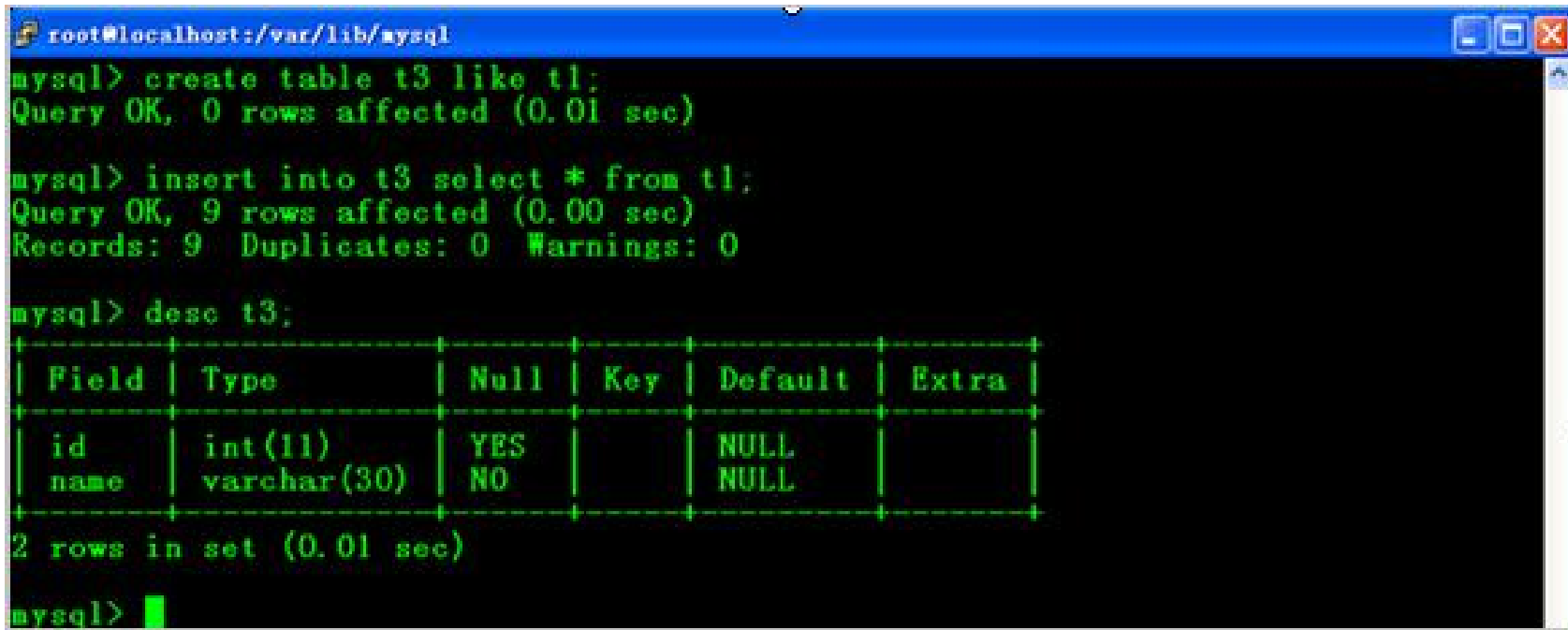
1.5 重排auto\_increment值

# 1.1 mysql表复制

复制表结构+复制表数据

```
mysql> create table t3 like t1;
```

```
mysql> insert into t3 select * from t1;
```



```
root@localhost:/var/lib/mysql
mysql> create table t3 like t1;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into t3 select * from t1;
Query OK, 9 rows affected (0.00 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> desc t3;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)       | YES  |     | NULL    |       |
| name  | varchar(30)   | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)

mysql>
```

# 1.2 mysql索引



## 1.ALTER TABLE用来创建普通索引、UNIQUE索引或PRIMARY KEY索引

ALTER TABLE table\_name ADD INDEX index\_name (column\_list)

ALTER TABLE table\_name ADD UNIQUE (column\_list)

ALTER TABLE table\_name ADD PRIMARY KEY (column\_list)

## 2.create index

CREATE INDEX index\_name ON table\_name (column\_list)

CREATE UNIQUE INDEX index\_name ON table\_name (column\_list)

## 3.drop index

DROP INDEX index\_name ON talbe\_name

## 4.alter table table drop

ALTER TABLE table\_name DROP INDEX index\_name

ALTER TABLE table\_name DROP PRIMARY KEY

# 1.3 mysql视图



创建视图:

```
mysql> create view v_t1 as select * from t1 where id>4  
and id<11;
```

Query OK, 0 rows affected (0.00 sec)

view视图的帮助信息:

```
mysql> ? view
```

ALTER VIEW

CREATE VIEW

DROP VIEW

查看视图:

```
mysql> show tables;
```

删除视图v\_t1:

```
mysql> drop view v_t1;
```

# 1.4 mysql内置函数



字符串函数:

CONCAT (string2 [,... ])	//连接字符串
LCASE (string2 )	//转换成小写
UCASE (string2 )	//转换成大写
LENGTH (string )	//string长度
LTRIM (string2 )	//去除前端空格
RTRIM (string2 )	//去除后端空格
REPEAT (string2 ,count )	//重复count次
REPLACE (str ,search_str ,replace_str )	//在str中用replace_str替换search_str
SUBSTRING (str , position [,length ])	//position开始,取length个字符
SPACE(count)	//生成count个空格

# 1.4 mysql内置函数



数学函数:

<code>BIN (decimal_number )</code>	<code>//十进制转二进制</code>
<code>CEILING (number2 )</code>	<code>//向上取整</code>
<code>FLOOR (number2 )</code>	<code>//向下取整</code>
<code>MAX(num1 ,num2)</code>	<code>//取最大值</code>
<code>MIN(num1,num2)</code>	<code>//取最小值</code>
<code>SQRT(number2)</code>	<code>//开平方</code>
<code>RAND()</code>	<code>//返回0-1内的随机值</code>



# 1.4 mysql内置函数



日期函数:

CURDATE()	//返回当前日期
CURTIME()	//返回当前时间
NOW()	//返回当前的日期时间
UNIX_TIMESTAMP(date)	//返回当前date的UNIX时间戳
FROM_UNIXTIME()	//返回UNIX时间戳的日期值
WEEK(date)	//返回日期date为一年中的第几周
YEAR(date)	//返回日期date的年份
DATEDIFF(expr,expr2) 数	//返回起始时间expr和结束时间expr2间天数

# 1.5 重排auto\_increment值



MYSQL数据库自动增长的ID如何恢复

清空表的时候。不能用

`delete from tablename;`

而是要用：

`truncate table tablename;`

这样auto\_increment 就恢复成1了

或者清空内容后直接用ALTER命令修改表:

`alter table tablename auto_increment = 1;`

## 二、常见SQL技巧和常见问题 云知梦

2.1 正则表达式的使用

2.2 巧用RAND()提取随机行

2.3 mysql中help的使用

## 2.1 正则表达式的使用

MySQL利用REGEXP命令提供给用户扩展的正则表达式功能,具体模式序列如下:

序列	序列说明	序列	序列说明
^	在字符串的开始处进行匹配	a?	匹配1个或零个a
\$	在字符串的末尾处进行匹配	a1 a2	匹配a1或a2
.	匹配任意单个字符,包括换行符	a(m)	匹配m个a
[...]	匹配出括号内德任意字符	a(m,)	匹配至少m个a
[^...]	匹配不出现括号内的任意字符	a(m,n)	匹配m到n个a
a*	匹配零个或多个a(包括空串)	a(,n)	匹配0到n个a
a+	匹配1个或多个(不包括空串)	(...)	将模式元素组成单一元素

使用正则表达式“\$”和“[...]”进行匹配:

```
mysql>select name,email from t where email REGEXP  
"@163[.,]com$"
```

使用like方式查询:

```
mysql>select name,email from t where email like  
"%@163.com"  
or email like "%@163,com"
```

## 2.2 巧用RAND( )提取随机行



- MySQL数据库中有一个随机函数rand( )是获取一个0—1之间的数，利用这个函数一起和order by能够把数据随机排序。

```
mysql>select * from stu order by rand( );
```

- 下面是通过limit随机抽取了3条数据样本。

```
mysql>select * from stu order by rand( ) limit 3;
```

## 2.3 mysql help使用



在mysql中那么多的命令如何才能记得住是个问题，这里有一个特别好的获得帮助的好方法，当然是在mysql>的提示下的操作：

- 1.? % 可以获得所有的mysql>里的命令，这个是最多的,那么这里的東西如何去进一步获得帮助呢？
- 2.? create
- 3.? opti% 因为记不住optimize的全称，这个时候可以用%来替代
- 4.? reg% 获得了记不住了的regex用法.
- 5.查看所有用? contents可以得到所有的帮助大纲,通过这个目录再用?继续往下细查.

# 三、SQL语句优化



- 3.1 优化SQL语句的一般步骤
- 3.2 索引优化
- 3.3 check与optimize使用方法
- 3.4 常用SQL的优化



## 3.1 优化SQL语句的一般步骤



3.1.1 通过show status命令了解各种SQL的执行频率。

格式：mysql> 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**

```
mysql> show status;
```

```
mysql> show global status;
```

```
mysql> show status like 'Com_%' ;
```

```
mysql> show global status like 'Com_%' ;
```

- 参数说明：
- Com\_XXX表示每个XXX语句执行的次数如：
  - 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
- 只针对于InnoDB存储引擎的。
  - InnoDB\_rows\_read 执行select操作的次数
  - InnoDB\_rows\_updated 执行update操作的次数
  - InnoDB\_rows\_inserted 执行insert操作的次数
  - InnoDB\_rows\_deleted 执行delete操作的次数
- 其他：
  - 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

### 3.1.2 定位执行效率较低的SQL语句

1)explain select \* from table where id=1000;

2)desc select \* from table where id=1000;

- 3.1.3 通过EXPLAIN分析较低效SQL的执行计划

```
mysql> explain select count(*) from stu where name like  
"a%" \G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: stu
```

```
type: range
```

```
possible_keys: name,ind_stu_name
```

```
key: name
```

```
key_len: 50
```

```
ref: NULL
```

```
rows: 8
```

```
Extra: Using where; Using index
```

```
1 row in set (0.00 sec)
```

- 每一列的简单解释
  - **id: 1**
  - **select\_type:** SIMPLE 表示select的类型，常见的取值有SIMPLE ( ) 简单表，即不使用表连接或者子查询)、PRIMARY (主查询，即外层的查询)、UNION (UNION中的第二个或者后面的查询语句)、SUBQUERY (子查询中的第一个SELECT) 等
  - **table: stu** 输出结果集的表
  - **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)、ref (同eq\_ref，但没有使用主键和唯一)、ref\_or\_null (同前面对null查询)、index\_merge (索引合并优化)、unique\_subquery (主键子查询)、index\_subquery (非主键子查询)、range (表中的范围查询)、index (都通过查询索引来得到数据)、all (通过全表扫描得到的数据)
  - **possible\_keys:** name, ind\_stu\_name 表查询时可能使用的索引。
  - **key: name** 表示实际使用的索引。
  - **key\_len: 50** 索引字段的长度
  - **ref: NULL**
  - **rows: 8** 扫描行的数量
  - **Extra:** Using where; Using index 执行情况的说明和描述

- 索引是数据库优化中最常见也是最重要的手段之一，通过索引通常可以帮助用户解决大多数的SQL性能问题。
- 3.2.1 索引的存储分类
- MyISAM存储引擎的表的数据和索引是自动分开存储的，各自是独立的一个文件；InnoDB存储引擎的表的数据和索引是存储在同一个表空间里面，但可以有多多个文件组成。
- MySQL目前不支持函数索引，但是能对列的前面某一部分进行索引，例如name字段，可以只取name的前4个字符进行索引，这个特性可以大大缩小索引文件的大小，用户在设计表结构的时候也可以对文本列根据此特性进行灵活设计。

```
mysql> create index ind_company2_name on  
company2(name(4));
```

其中**company**表名 **ind\_company2\_name**索引名

## 3.2.2 MySQL如何使用索引

**索引用于快速找出在某个列中有一特定值的行。对相关列使用索引是提高SELECT操作性能的最佳途径。**

### 1、使用索引

**(1)对于创建的多列索引，只要查询的条件中用到最左边的列，索引一般就会被使用。如下创建一个复合索引。**

```
mysql>create index ind_sales2_com_mon  
onsales2(company_id,moneys);
```

**然后按company\_id进行查询，发现使用到了复合索引**

```
mysql>explain select * from sales2 where company_id=2006\G
```

**使用下面的查询就没有使用到复合索引。**

```
mysql>explain select * from sales2 where moneys=1\G
```

**(2) 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，如下：**

```
mysql> explain select * from company2 where name like "%3"\G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: company2
```

```
type: ALL
```

```
possible_keys: NULL
```

```
key: NULL
```

```
key_len: NULL
```

```
ref: NULL
```

```
rows: 1000
```

```
Extra: Using where
```

```
1 row in set (0.00 sec)
```



如下这个使用到了索引，而下面例子能够使用索引，区别就在于“%”的位置不同，上面的例子是吧“%”放在了第一位，而下面的例子则没有

```
mysql> explain select * from company2 where name like "3%" \G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: company2
```

```
type: range
```

```
possible_keys: ind_company2_name
```

```
key: ind_company2_name
```

```
key_len: 11
```

```
ref: NULL
```

```
rows: 103
```

```
Extra: Using where
```

```
1 row in set (0.00 sec)
```

(3)如果对大的文本进行搜索，使用全文索引而不使用 like “%...%” .

(4)如果列名是索引，使用column\_name is null将使用索引。如下

```
mysql> explain select * from company2 where name is null\G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: company2
```

```
type: ref
```

```
possible_keys: ind_company2_name
```

```
key: ind_company2_name
```

```
key_len: 11
```

```
ref: const
```

```
rows: 1
```

```
Extra: Using where
```

```
1 row in set (0.00 sec)
```

## 2、存在索引但不使用索引

(1)如果MySQL估计使用索引比全表扫描更慢，则不使用索引。例如如果列key\_part1均匀分布在1到100之间，查询时使用索引就不是很好

```
mysql>select * from table_name where key_part1>1 and  
key_part<90;
```

(2)如果使用MEMORY/HEAP表并且where条件中不使用“=”进行索引列，那么不会用到索引。Heap表只有在“=”的条件下会使用索引。

(3)用or分割开的条件，如果or前的条件中的列有索引，而后面的列中没有索引，那么涉及的索引都不会被用到。

```
mysql>show index from sales\G
```

```
***** 1. row *****
```

```
... ..
```

```
key_name: ind_sales_year
```

```
seq_in_index : 1
```

```
Column_name: year
```

```
... ..
```

从上面可以发现只有year列上面有索引。来看如下的执行计划。

```
mysql> explain select * from sales where year=2001 or  
country= 'China' \G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: sales
```

```
type: ALL
```

```
possible_keys: ind_sales_year
```

```
key: NULL
```

```
key_len: NULL
```

```
ref: NULL
```

```
rows: 12
```

```
Extra: Using where
```

```
1 row in set (0.00 sec)
```

(4)如果不是索引列的第一部分，如下例子:可见虽然在money上面建有复合索引，但是由于money不是索引的第一列，那么在查询中这个索引也不会被MySQL采用。

```
mysql> explain select * from sales2 where moneys=1 \G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: sales2
```

```
type: ALL
```

```
possible_keys: NULL
```

```
key: NULL
```

```
key_len: NULL
```

```
ref: NULL
```

```
rows: 1000
```

```
Extra: Using where
```

```
1 row in set (0.00 sec)
```

- (5)如果like是以%开始，可见虽然在name上面建有索引，但是由于where条件中like的值的“%”在第一位了，那么MySQL也会采用这个索引。

```
mysql> explain select * from company2 where name  
like '%3' \G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: company2
```

```
type: ALL
```

```
possible_keys: NULL
```

```
key: NULL
```

```
key_len: NULL
```

```
ref: NULL
```

```
rows: 1000
```

```
Extra: Using where
```

```
1 row in set (0.00 sec)
```

- (6)如果列类型是字符串，但在查询时把一个数值型常量赋值给了一个字符型的列名name，那么虽然在name列上有索引，但是也没有

```
mysql> explain select * from company2 where name  
name=294\G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: company2
```

```
type: ALL
```

```
possible_keys: ind_company2_name
```

```
key: NULL
```

```
key_len: NULL
```

```
ref: NULL
```

```
rows: 1000
```

```
Extra: Using where
```

```
1 row in set (0.00 sec)
```

- 而下面的sql语句就可以正确使用索引。

```
mysql> explain select * from company2 where name  
name= '294' \G
```

```
***** 1. row *****  
    id: 1  
  select_type: SIMPLE  
    table: company2  
     type: ref  
possible_keys: ind_company2_name  
      key: ind_company2_name  
   key_len: 23  
     ref: const  
    rows: 1  
  Extra: Using where  
1 row in set (0.00 sec)
```



### • 3.2.3 查看索引使用情况

- 如果索引正在工作，Handler\_read\_key的值将很高，这个值代表了一个行被索引值读的次数。
- Handler\_read\_rnd\_next的值高则意味着查询运行低效，并且应该建立索引补救。

```
mysql> show status like 'Handler_read%';
```

Variable_name	Value
Handler_read_first	0
Handler_read_key	5
Handler_read_next	0
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	2055

```
6 rows in set (0.00 sec)
```

## 3.3 两个简单实用的优化方法

- 分析表的语法如下:(检查一个或多个表是否有错误)

```
mysql> CHECK TABLE tbl_name[,tbl_name] ... [option] ...
```

option =

{ QUICK | FAST | MEDIUM | EXTENDED | CHANGED }

```
mysql> check table sales;
```

+	-----	+	-----	+	-----	+	-----	+
	Table		Op		Msg_type		Msg_text	
+	-----	+	-----	+	-----	+	-----	+
	sakila.sales		check		status		OK	
+	-----	+	-----	+	-----	+	-----	+

```
1 row in set (0.01 sec)
```

- 

- **优化表的语法格式：**

**OPTIMIZE [LOCAL | NO\_WRITE\_TO\_BINLOG] TABLE tbl\_name  
[,tbl\_name]**

- **如果已经删除了表的一大部分，或者如果已经对含有可变长度行的表进行了很多的改动，则需要做定期优化。这个命令可以将表中的空间碎片进行合并，但是此命令只对MyISAM、BDB和InnoDB表起作用。**

```
mysql> optimize table sales;
```

Table	Op	Msg_type	Msg_text
sakila.sales	optimize	status	OK

```
1 row in set (0.05 sec)
```

### 3.4 优化嵌套查询

下面是采用嵌套查询的效果（可以使用更有效的链接查询（Join）替代）

。

```
mysql> explain select * from sales2 where company_id not  
in(select id from company2)\G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: sales2
```

```
type: ALL
```

```
possible_keys: NULL
```

```
key: NULL
```

```
key_len: NULL
```

```
ref: NULL
```

```
rows: 1000
```

```
Extra: Using where
```

```
1 row in set (0.00 sec)
```

第1/2页

● \*\*\*\*\* 2. row \*\*\*\*\*

id: 2

select\_type: SIMPLE

table: company2

type: index\_subquery

possible\_keys: ind\_company2\_id

key: ind\_company2\_id

key\_len: 5

ref: func

rows: 2

Extra: Using index

1 row in set (0.00 sec)

第2/2页

## 下面是使用更有效的链接查询 ( Join )

```
mysql> explain select * from sales2 left join company2 on  
sales2.company_id = company2.id where sales2.company_id  
is null\G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: sales2
```

```
type: ALL
```

```
possible_keys: ind_sales2_companyid_moneys
```

```
key: ind_sales2_companyid_moneys
```

```
key_len: 5
```

```
ref: count
```

```
rows: 1
```

```
Extra: Using where
```

```
1 row in set (0.00 sec)
```

第1/2页

● \*\*\*\*\* 2. row \*\*\*\*\*

id: 2  
select\_type: SIMPLE  
table: company2  
type: index\_subquery  
possible\_keys: ind\_company2\_id  
key: ind\_company2\_id  
key\_len: 5  
ref: func  
rows: 1  
Extra:

1 row in set (0.00 sec)

第2/2页

- 从执行计划中可以明显看出查询扫描的记录范围和使用索引的情况都有了很大的改善。连接（JOIN）子所以更有效率一些，是因为MySQL不需要再内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作。

# 四、Mysql服务器优化



4.1 四种字符集问题

4.2 slow log慢查询日志问题

4.3 root密码丢失



# 4.1 字符集设置



```
[client]
#password      = your_password
port           = 3306
socket         = /var/lib/mysql/mysql.sock
default-character-set=utf8
[mysqld]
port           = 3306
socket         = /var/lib/mysql/mysql.sock
character-set-server=utf8
collation-server=utf8_general_ci
```

# 4.1 字符集效果



```
mysql> \s
-----
mysql Ver 14.12 Distrib 5.0.77, for redhat-linux-gnu (i686) using readline 5.1

Connection id:          4
Current database:
Current user:           root@localhost
SSL:                    Not in use
Current pager:          stdout
Using outfile:          ''
Using delimiter:        ;
Server version:         5.0.77-log Source distribution
Protocol version:       10
Connection:             Localhost via UNIX socket
Server characterset:    utf8
Db      characterset:    utf8
Client characterset:    utf8
Conn.  characterset:    utf8
UNIX socket:            /var/lib/mysql/mysql.sock
Uptime:                 3 min 56 sec

Threads: 1  Questions: 14  Slow queries: 0  Opens: 12  Flush tables: 1  Open tables: 6  Queries p
er second avg: 0.059
-----
```

## 4.2 慢查询日志



### 1. 有关慢查询

开户和设置慢查询时间:

```
vi /etc/my.cnf
```

```
log_slow_queries=slow.log
```

```
long_query_time=5
```

## 4.2 慢查询日志

查看设置后是否生效

```
mysql> show variables like "%quer%";
```

```
mysql> show variables like "%quer%";
```

Variable_name	Value
ft_query_expansion_limit	20
have_query_cache	YES
log_queries_not_using_indexes	OFF
log_slow_queries	ON
long_query_time	5
query_alloc_block_size	8192
query_cache_limit	1048576
query_cache_min_res_unit	4096
query_cache_size	0
query_cache_type	ON
query_cache_wlock_invalidate	OFF
query_prealloc_size	8192

```
12 rows in set (0.01 sec)
```

## 4.2 慢查询日志

慢查询次数:

```
mysql> show global status like  
"%quer%";
```

```
mysql> show global status like "%quer%";
```

Variable_name	Value
Last_query_cost	0.000000
Qcache_queries_in_cache	0
Queries	5
Slow_queries	0

```
4 rows in set (0.00 sec)
```

## 4.3 root密码丢失



### root密码丢失破解

1. service mysqld stop

2. mysqld\_safe --skip-grant-tables --user=mysql &  
//跳过授权表mysql.user和mysql.db这些表

3. mysql -uroot

4. set password=password("wei");

//用这一条语句结果报错,就是因为加了--skip-grant-tables

4. mysql> update user set password=password("wei") where  
user='root' and host='localhost';

5. mysql> set password for  
root@localhost=password("wei");

6. mysql> set password=password("wei");  
//和第五步一样,都可能成功修改密码

云知梦，只为有梦想的人