txgl

2015

Mysql总结

mysql

txgl

[键入公司地址]

[一、 安装mysql 4](#_Toc32125)

[(一) windows下安装mysql 4](#_Toc32175)

[1) 通用二进制包安装 4](#_Toc8646)

[2) 安装包文件安装 6](#_Toc14513)

[3) 源码编译安装 6](#_Toc22222)

[(二) linux下安装mysql 6](#_Toc20107)

[1) 通用二进制码安装 6](#_Toc16649)

[2) 源码编译安装 9](#_Toc11421)

[3) 系统对应的安装包安装 9](#_Toc28714)

[二、 mysql主从库配置 9](#_Toc25688)

[(一) 数据库设置 10](#_Toc5693)

[1) 主库my.ini/my.cnf设置： 10](#_Toc13428)

[2) 从库my.ini/my.cnf设置： 10](#_Toc24715)

[(二) 主库新增用户并赋权限 11](#_Toc8380)

[(三) 设置从库的主库信息 11](#_Toc18620)

[1) 进入主库：查看主库的master信息： 11](#_Toc13622)

[2) 进入从库：查看从库的slave信息： 12](#_Toc10185)

[3) 设置从库的主库信息 12](#_Toc14272)

[(四) 设置主从库常见错误 13](#_Toc25020)

[1) MySQL server UUIDs 错误 14](#_Toc25426)

[2) Could not find first log file name in binary log index file错误 14](#_Toc30813)

[三、 mysql 基本数据类型及其操作 14](#_Toc25219)

[(一) 基本数据类型 15](#_Toc1071)

[(二) 数据操作（转换） 15](#_Toc11012)

[(三) 表的创建、删除及结构修改 15](#_Toc20075)

[1) 创建表 16](#_Toc10800)

[2) 修改表 16](#_Toc23527)

[3) 复制表 17](#_Toc29409)

[4) 删除表 18](#_Toc30720)

[5) 特殊表：dual 18](#_Toc24365)

[6) 创建临时表 19](#_Toc2046)

[7) 表结构查询 19](#_Toc27825)

[四、 mysql赋权 20](#_Toc29336)

[(一) 查询、插入、更新、删除数据库中表数据。 21](#_Toc4245)

[(二) 创建表、索引、视图、存储过程、函数等权限。 21](#_Toc1141)

[(三) 普通 DBA权限。 21](#_Toc26748)

[(四) 高级 DBA权限。 22](#_Toc16867)

[(五) grant 权限，分别作用在多个层次上 22](#_Toc6121)

[(六) 查看 MySQL 用户权限 22](#_Toc22299)

[(七) 撤销已经赋予给 MySQL 用户权限的权限。 22](#_Toc24040)

[(八) grant、revoke 用户权限注意事项 23](#_Toc9160)

[五、 mysql增删改查排序等常见操作 23](#_Toc3330)

[(一) 查询 24](#_Toc944)

[1) 模糊查询和索引（索引与优化like查询） 24](#_Toc8130)

[(二) 排序 26](#_Toc17835)

[(三) 数据库触发联合索引的几个条件 26](#_Toc13898)

[六、 mysql常用内置函数 27](#_Toc2891)

[(一) 数学函数 28](#_Toc28122)

[(二) 字符串函数 28](#_Toc24299)

[(三) 日期、时间相关函数 29](#_Toc18732)

[(四) 条件判断函数 30](#_Toc31481)

[1) IF(expr,v1,v2)函数 30](#_Toc22804)

[2) CASE函数 30](#_Toc21679)

[(五) 系统信息函数 31](#_Toc3785)

[(六) 加密解密函数 31](#_Toc19879)

[(七) 其他函数 31](#_Toc11708)

[七、 mysqlc常用命令 32](#_Toc19171)

[(一) 输出当前所用的数据库： 33](#_Toc16621)

[(二) 输出版本号 33](#_Toc9538)

[八、 mysql慢查询 33](#_Toc22554)

[(一) 慢查询作用 34](#_Toc18045)

[(二) 如何开启慢查询 34](#_Toc13121)

[九、 导入数据 35](#_Toc7964)

[(一) 命令行导入 36](#_Toc30534)

[十、 mysql调优 38](#_Toc17669)

[(一) 查看MySQL运行情况 39](#_Toc7362)

[(二) 查看INNODB数据库引擎运行状态 39](#_Toc18322)

[(三) 查看当前正在进行的进程，对于有锁表等情况的排查很有用处 39](#_Toc4670)

[(四) 查看MySQL的配置参数 39](#_Toc27988)

[(五) 查看当前已经被打开的表列表 39](#_Toc5489)

[(六) SHOW STATUS;关键结果释义 39](#_Toc28908)

[(七) 提升性能的建议 40](#_Toc9014)

[(八) 查看MySQL服务器配置信息 42](#_Toc15772)

[(九) 查看MySQL服务器运行的各种状态值 42](#_Toc7244)

[十一、 48](#_Toc29060)

[十二、 mysql常见错误以及解决方法 48](#_Toc27951)

[(一) 远程连接10038错误 49](#_Toc23208)

# 安装mysql

注： my.ini/my.cnf编码：

my.ini/my.cnf文件的编码必须是英文编码（如windows中的ANSI），不能是UTF-8或GBK等。

## windows下安装mysql

windows下安装mysql可以分为两种方式：通用二进制包安装、安装包文件安装、源码编译安装；

* 注：此次的案例基于windows7实现
* 注：此次的安装包为：mysql-5.6.24-winx64.zip
* 注：安装目录：E:\mysql\mysql-5.6.24-winx64

### 通用二进制包安装

1. 修改配置文件my.ini

复制E:\mysql\mysql-5.6.24-winx64\my-default.ini，粘贴在同一个目录中，并将名称改为my.ini

同时将其内容修改为：

[mysqladmin]

user=root

[mysqld]

basedir=E:/mysql/mysql-5.6.24-winx64

datadir=E:/mysql/mysql-5.6.24-winx64/data

port=3307

socket=E:/mysql/mysql-5.6.24-winx64/temp/mysql.sock

tmpdir=E:/mysql/mysql-5.6.24-winx64/temp

pid-file=E:/mysql/mysql-5.6.24-winx64/temp/manager.pid

log-error=E:/mysql/mysql-5.6.24-winx64/temp/log.err

max\_allowed\_packet=16M

###如果为：bind-address = 127.0.0.1；则远程无法访问

bind-address=127.0.0.1

lower\_case\_table\_names=1

character-set-server=UTF8

collation-server=utf8\_general\_ci

[mysqld\_safe]

basedir=E:/mysql/mysql-5.6.24-winx64

datadir=E:/mysql/mysql-5.6.24-winx64/data

port=3307

socket=E:/mysql/mysql-5.6.24-winx64/temp/mysql.sock

tmpdir=E:/mysql/mysql-5.6.24-winx64/temp

pid-file=E:/mysql/mysql-5.6.24-winx64/temp/manager.pid

log-error=E:/mysql/mysql-5.6.24-winx64/temp/log.err

[client]

port=3307

socket=E:/mysql/mysql-5.6.24-winx64/temp/mysql.sock

default-character-set=UTF8

[manager]

port=3307

socket=E:/mysql/mysql-5.6.24-winx64/temp/mysql.sock

pid-file=E:/mysql/mysql-5.6.24-winx64/temp/manager.pid

sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES

1. 启动mysql

mysqld --defaults-file="E:/mysql/mysql-5.6.24-winx64/my.ini" --console --skip-grant-tables

注：mysql5.7版本及其以上，在启动mysql之前需要初始化，

mysqld –initialize：

根据my.ini自动创建指定的data，默认用户为root，密码处于指定的data目录下的DESKTOP-PH8IHCD.err（或类似的文件）中，比如

A temporary password is generated for root@localhost: qxcqh,7p\_INp

其中【qxcqh,7p\_INp】就是密码

mysqld --initialize-insecure：

与mysqld –initialize 不同之处是默认密码为空

1. 将mysql注册为服务

cmd进入安装目录的bin目录：E:/mysql/mysql-5.6.24-winx64/bin，执行如下命令

mysqld --install"mysql3307"--defaults-file="E:/mysql/mysql-5.6.24-winx64/my.ini"

* 注：如果报Install/Remove of the Service Denied；解决办法：打开cmd.exe程序的时候选择“用管理员身份打开”。
* 注：mysql3307：为注册的mysql服务名，带空格需用引号；
* 注：E:/mysql/mysql-5.6.24-winx64/my.ini为配置的文件路径，带空格需用引号；
* 注：删除服务：sc delete "mysql3307"

1. 启动服务

net start "mysql3307"

1. 关闭服务

net stop "mysql3307"

1. 删除服务

删除服务：sc delete "mysql3307"

1. 登录mysql

cmd进入安装目录的bin目录：E:/mysql/mysql-5.6.24-winx64/bin，执行如下命令

mysql -u root -p

### 安装包文件安装

### 源码编译安装

## linux下安装mysql

linux安装mysql可以分为三种方式：通用二进制码安装、源码编译安装、系统对应的安装包安装；

* 注：此次的案例基于redhat6.4实现！

删除系统原有的mysql

查找已有的mysql：

rpm -q mysql

如果没有mysql，则使用

rpm -qa | grep -i mysql

强制卸载将要卸载的必须是上面查询到的mysql的全称包括版本号，此处是在无法正常卸载时不得已的操作，强制卸载，必须用参数--nodeps

rpm -e --nodeps 输出的包名（比如mysql-5.0.77-4.el5\_5.3）

### 通用二进制码安装

此次案例基于

* 安装包：mysql-5.6.24-linux-glibc2.5-i686.tar.gz；
* 目录：/app/mysql

步骤：

1. 复制安装包到安装目录：

cp mysql-5.6.24-linux-glibc2.5-i686.tar.gz/app/mysql

1. 进入安装路径并解压安装包：

cd /app/mysql

tar -zxvf mysql-5.6.24-linux-glibc2.5-i686.tar.gz

* 此时mysql的主路径为：/app/mysql/mysql-5.6.24-linux-glibc2.5-i686

1. 进入主路径，并复制support-files/my-default.cnf到主路径下改其名为my\_install.cnf，并更改其内容

cp support-files/my-default.cnf/app/mysql/mysql-5.6.24-linux-glibc2.5-i686

mv my-default.cnfmy\_install.cnf

vi my\_install.cnf

将其内容修改为以下方框中的内容；

vi命令：

1. 输入“i”：表示编辑；
2. ctrl+c：停止编辑，并开始接收vi命令；
3. ":wq "：保存并退出；
4. ":q!"：退出但不保存；

[mysqladmin]

user=root

[mysqld]

basedir=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686

datadir=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/data

port=3307

socket=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/tmp/mysql.sock

tmpdir=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/tmp

pid-file=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/temp/manager.pid

log-error=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/temp/log.err

max\_allowed\_packet=16M

###如果为：bind-address = 127.0.0.1；则远程无法访问

bind-address=0.0.0.0

lower\_case\_table\_names=1

character-set-server=UTF8

collation-server=utf8\_general\_ci

[mysqld\_safe]

basedir=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686

datadir=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/data

port=3307

socket=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/temp/mysql.sock

tmpdir=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/temp

pid-file=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/temp/manager.pid

log-error=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/temp/log.err

[client]

port=3307

socket=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/temp/mysql.sock

default-character-set=UTF8

[manager]

port=3307

socket=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/temp/mysql.sock

pid-file=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/temp/manager.pid

sql\_mode=NO\_ENGINE\_SUBSTITUTION,STRICT\_TRANS\_TABLES

1. 创建temp目录，然后赋权限并调整用户为mysql用户组下的mysql用户

mkdir temp

chmod 755 temp

chown -R mysql:mysql temp

如果用户mysql不存在，则新建

groupadd mysql

useradd -r -g mysql mysql

可以在/etc/group文件中查看用户组：cat /etc/group

删除用户用户组：

groupdel mysql

userdel -r mysql

"-r"：表示创建mysql用户的目录/home/mysql；

"-g"：表示用户组；

1. 安装mysql

执行命令：

scripts/mysql\_install\_db --user=mysql --basedir=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686 --datadir=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/data --port=3307 --socket=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/tmp/mysql.sock --tmpdir=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/tmp --pid-file=/app/mysql/mysql-5.6.24-linux-glic2.5-i686/tmp/manager.pid --log-error=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/tmp/tmp.err

或者

scripts/mysql\_install\_db --defaults-file=my\_install.cnf --user=mysql

1. 启动mysql

此时将my\_install.cnf名称修改为my.cnf(如果mysql安装后自动生成了此文件，将自动生成的my.cnf改为其他名称)

./bin/mysqld\_safe --defaults-file=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/my.cnf --user=mysql &

* 其中--defaulsts-file参数和--user参数的顺序不可更改，否者会报错【\*\*mysqld: unknown variable 'defaults-file'\*\*\*\*】；
* 启动命令中的“&”表示后台执行，也就是说退出此shell，mysql依然正常运行；

1. 修改linux的环境变量，以便于在shell中执行mysql的相关命令

vi /etc/profile

在/etc/profile中追加以下内容

#mysql

MYSQL\_HOME=/app/mysql/mysql-5.6.24-linux-glibc2.5-i686

PATH=$PATH:$MYSQL\_HOME/bin:MYSQL\_HOME/lib

export PATH MYSQL\_HOME

source /etc/profile

1. 关闭mysql

[root@localhost ~]#mysqladmin -uroot -p888888 shutdown

1. 首次登录mysql

在shell中直接输入mysql，默认以root+空密码登录；

[root@localhost ~]# mysql

[root@localhost ~]#use mysql;

[root@localhost ~]# select \* from user;

1. 修改root密码

修改root密码为123456

[root@localhost ~]#mysqladmin -u root password 123456

登录

[root@localhost ~]#mysql -u root -p

或者

修改root密码为123456

[root@localhost ~]#mysql

[root@localhost ~]#use mysql

[root@localhost ~]#UPDATE user SET Password=PASSWORD('123456') where USER='root';

登录

[root@localhost ~]#mysql -u root -p

或者（高版本的mysql只有此种方法可用）

修改root密码为123456

[root@localhost ~]#mysql

[root@localhost ~]#use mysql

[root@localhost ~]# set password for 用户名@localhost = password('新密码');

登录

[root@localhost ~]#mysql -u root -p

然后输入密码即可登录；

* mysql登录：mysql -u用户 -p密码 -hip

例如：mysql -uroot -pmysqlpassword -h127.0.0.1

1. mysql远程登录

GRANT ALL ON \*.\* TO root@'%' IDENTIFIED BY 'root' WITH GRANT OPTION;

### 源码编译安装

### 系统对应的安装包安装

# mysql主从库配置

以下部分以windows 10 64bit 系统+ mysql-5.6.29-winx64为例；

## 数据库设置

注：

1. 修改主从库my.ini/my.cnf中的server-id，保证两者不同； 否者会报Slave IO异常（Slave\_IO\_Running: No）；
2. xxx/data/auto.cnf中的server-uuid，保证两者不同否者会报Slave IO异常（Slave\_IO\_Running: No）；

### 主库my.ini/my.cnf设置：

[mysqld]

###主从同步设置master（start）

server-id=1

#要生成的二进制日记文件名称

log-bin=E:/meiCai/mysql/mysql-5.6.29-winx64-3307-master/temp/mysql-master-bin

#sync\_binlog=1

#binlog-do-db要同步的数据库

binlog-do-db=test

#不同步的数据库,如果指定了binlog-do-db这里应该可以不用指定的

binlog-ignore-db=mysql

###主从同步设置master（end）

### 从库my.ini/my.cnf设置：

[mysqld]

###主从同步设置slave（start）

server-id=2

#master-host=127.0.0.1

#master-user=backup

#master-password=888888

#master-port=3307

#master-connect-retry=60

#sync\_binlog=1

##同步的数据库

replicate-do-db=test

##被忽略的数据库

replicate-ignore-db=mysql

#要生成的二进制日记文件名称

log-bin=E:/meiCai/mysql/mysql-5.6.29-winx64-3308-slave/temp/mysql-slave-bin

relay\_log=mysql-relay-bin

log\_slave\_updates=1

read\_only=1

###主从同步设置slave（end）

## 主库新增用户并赋权限

~~GRANT REPLICATION SLAVE,RELOAD,SUPER,FILE ON \*.\* TO 'backup'@'127.0.0.1' IDENTIFIED BY '888888';~~

~~GRANT REPLICATION SLAVE,RELOAD,SUPER,FILE ON \*.\* TO 'backup'@'localhost' IDENTIFIED BY '888888';~~

GRANT REPLICATION SLAVE,RELOAD,SUPER,FILE ON \*.\* TO 'backup'@'%' IDENTIFIED BY '888888';

grant all on \*.\* to backup@'%'IDENTIFIED BY '888888';

flush privileges;

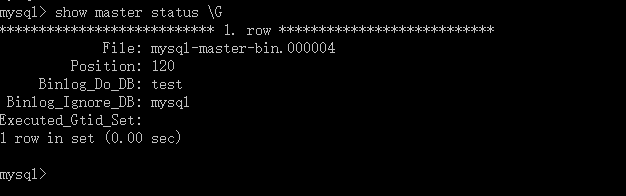
## 设置从库的主库信息

### 进入主库：查看主库的master信息：

输入命令：

mysql> show master status \G

结果如下所示：



其中

1. File对应my.ini/my.cnf中的log-bin设置，其后的数字代表着快照版本号，可以使用flush logs命令来更新，每执行一次快照号加1；
2. Binlog\_Do\_DB对应my.ini/my.cnf中的binlog-do-db设置；
3. Binlog\_Ignore\_DB对应my.ini/my.cnf中的binlog-ignore-db设置；

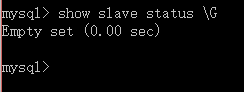
### 进入从库：查看从库的slave信息：

输入命令：

mysql>show slave status \G

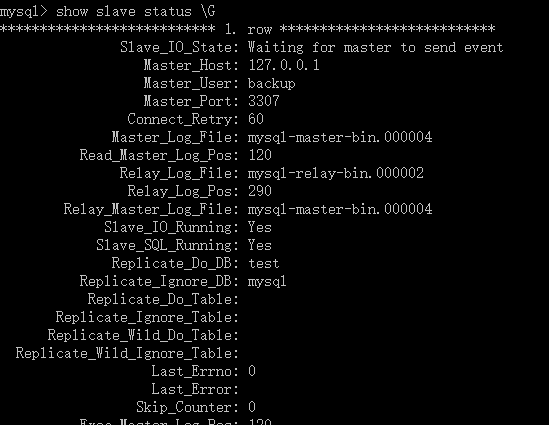
如果slave未配置：

结果显示如下：



如果已经配置：

结果显示如下：



### 设置从库的主库信息

如果未设置

则使用如下命令：

CHANGE MASTER TO

MASTER\_HOST='127.0.0.1',

MASTER\_PORT=3307,

MASTER\_USER='backup',

MASTER\_PASSWORD='888888',

MASTER\_LOG\_FILE='mysql-bin.000011',

MASTER\_LOG\_POS=120;

建立连接主库的管道；

如果执行此命令报错，可以使用

reset slave;

或

stop slave;

重置或停止slave；

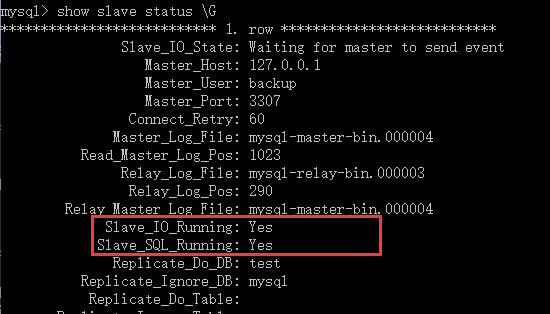
然后启动slave

start slave;

然后输入命令查看

mysql>show slave status \G

如果显示



则表示主从连接成功；否者根据之前的配置查看是否有出入，若无出入，到log.err文件中查看具体的错误信息；

## 设置主从库常见错误

### MySQL server UUIDs 错误

日志中显示

[ERROR] Fatal error: The slave I/O thread stops because master and slave have equal MySQL server UUIDs; these UUIDs must be different for replication to work.

解决方法：

在主从库中执行：show variables like ‘server\_id ';

查看server\_id ，如果两个值一样，则将my.ini/my.cnf中的server-id的值改为不同的值即可；如果一样，则查看xxxxx/data/auto.cnf文件，里面记录了数据库的uuid，每个库的uuid应该是不一样的，内容大致如下：

[auto]

server-uuid=6dcee5be-8cdb-11e2-9408-90e2ba2e2ea6

将主从库的server-uuid改为不同的值即可；

### Could not find first log file name in binary log index file错误

日志中显示

[ERROR] Error reading packet from server: Could not find first log file name in binary log index file ( server\_errno=1236)

[ERROR] Slave I/O: Got fatal error 1236 from master when reading data from binary log: 'Could not find first log file name in binary log index file', Error\_code: 1236

解决方法：

此问题说明从库在查找主库的日志时失败；

在主库中输入命令：

mysql> show master status \G

查找file信息

然后将从库的主库信息中的master\_log\_file修改为主库的file信息

执行命令如下所示：

CHANGE MASTER TO

MASTER\_LOG\_FILE='mysql-bin.000011';

# mysql 基本数据类型及其操作

## 基本数据类型

mysql提供了五种整型： tinyint、smallint、mediumint、int和bigint。int为integer的缩写。这些类型在可表示的取值范围上是不同的。 整数列可定义为unsigned从而禁用负值；这使列的取值范围为0以上。各种类型的存储量需求也是不同的。取值范围较大的类型所需的存储量较大。

mysql 提供三种浮点类型： float、double和decimal。与整型不同，浮点类型不能是unsigned的，其取值范围也与整型不同，这种不同不仅在于这些类型有最大 值，而且还有最小非零值。最小值提供了相应类型精度的一种度量，这对于记录科学数据来说是非常重要的（当然，也有负的最大和最小值）。

|  |  |  |
| --- | --- | --- |
| 类型 | 存储长度 | 类型说明 |
| tinyint[(m)] | 1字节 | 有符号值：-128 到127（- 2^7 到2^7 - 1）  无符号值：0到255（0 到2^8 - 1） |
| smallint[(m)] | 2字节 | 有符号值：-32768 到32767（- 2^15 到2^15 - 1）  无符号值：0到65535（0 到2^1 6 - 1） |
| mediumint[(m)] | 3字节 | 有符号值：-8388608 到8388607（- 2^23 到2^23 - 1 ）  无符号值：0到16777215（0 到2^24 - 1） |
| int[(m)] | 4字节 | 有符号值：-2147683648 到2147683647（- 2^31 到2^31- 1）  无符号值：0到4294967295（0 到2^32 - 1） |
| bigint[(m)] | 8字节 | 有符号值：-9223372036854775808 到9223373036854775807（- 2^63到2^63-1）  无符号值：0到18446744073709551615（0到2^64 – 1） 8个字节 |
| float[(m, d)] | 4字节 | 最小非零值：±1.175494351e - 38 |
| double[(m, d)] | 8字节 | 最小非零值：±2.2250738585072014e - 308 |
| decimal[(m, d)] |  | 可变；其值的范围依赖于m 和d  m字节（mysql < 3.23）  m+2字节（mysql > 3.23 ） |

## 数据操作（转换）

int 转化为string类型：SELECT CONCAT(1,'');

string转化为int类型：SELECT CAST(' 1' AS SIGNED);

## 表的创建、删除及结构修改

### 创建表

CREATE TABLE IF NOT EXISTS`NewTable` (

`peopleid` smallint(6) NOT NULL AUTO\_INCREMENT,

`firstname` char(50) NOT NULL,

`lastname` char(50) NOT NULL,

`age` smallint(6) NOT NULL,

`townid` smallint(6) NOT NULL,

PRIMARY KEY (`peopleid`),

UNIQUE KEY `unique\_fname\_lname`(`firstname`,`lastname`),

KEY `fname\_lname\_age` (`firstname`,`lastname`,`age`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

IF NOT EXISTS：表示如果不存在，则创建，否者无任何操作

其中peopleid是主键,以firstname和lastname两列建立了一个唯一索引,以firstname,lastname,age三列建立了一个普通索引

### 修改表

1. 修改表名

alter table table\_name rename table\_new\_name;

1. 增加列[add 列名]

alter table 表名 add 列名 列类型 列参数【加的列在表的最后面】

例：alter table test add username char(20) not null default '';

alter table test add birth date not null default '0000-00-00';

alter table 表名 add 列名 列类型 列参数 after 某列【把新列加在某列后面】

例：alter table test add gender char(1) not null default '' after username;

alter table 表名 add 列名 列类型 列参数 first【把新列加在最前面】

例：alter table test add pid int not null default 0 first;

1. 删除列[drop 列名]

alter table 表名 drop 列名

例：alter table test drop pid;

1. 修改列[modife 列名]

alter table 表名 modify 列名 新类型 新参数【修改列类型】

例：alter table test modify gender char(4) not null default '';

alter table 表名 change 旧列名 新列名 新类型 新参数【修改列名和列类型】

例：alter table test change pid uid int unsigned not null default 0;

### 复制表

1. 复制表结构(只复制表的列名以及列类型)以及数据

CREATE TABLE 新表 SELECT \* FROM 旧表;

也可以明确地为一个已生成的列指定类型

CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;

用SELECT语句创建的列附在表的右侧，而不是覆盖在表上

如：

mysql> SELECT \* FROM foo;

+---+

| n |

+---+

| 1 |

+---+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;

mysql> SELECT \* FROM bar;

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

| m | n |

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

| NULL | 1 |

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

1. 只复制表结构(只复制表的列名以及列类型)

CREATE TABLE 新表 SELECT \* FROM 旧表 WHERE 1=2

即:让WHERE条件不成立

1. 完全复制表结构(包含列名、列类型、主键、索引等等信息，但那是并不复制数据)

根据其它表的定义（包括在原表中定义的所有的列属性和索引），使用LIKE创建一个空表:

(低版本的mysql不支持，mysql4.0.25 不支持，mysql5已经支持了)

CREATE TABLE 新表 LIKE 旧表

1. 复制旧表数据到新表（两表结构一样）

INSERT INTO 新表 SELECT \* FROM 旧表

1. 复制旧表数据到新表（两表结构不一样）

INSERT INTO 新表(字段1,字段2,…….) SELECT 字段1, 字段2, …… FROM 旧表

### 删除表

DROP TABLE tbl\_name;

或者是

DROP TABLE IF EXISTS tbl\_name;

### 特殊表：dual

在mysql里也存在和oracle里类似的dual虚拟表：官方声明纯粹是为了满足select ... from...这一习惯问题，mysql会忽略对该表的引用。

你可千万注意了：

select \* from dual; mysql会出错——1096：没有使用到表；而oracle 会返回列：“du”，其值为“X”——行为和只有一行记录的普通表是一样的！

select express　[from dual]; mysql总是作为返回该表达式值的普通select语句执行，返回一行记录的结果集，from dual 对mysql来说根本就是摆设！而oracle里该句必须有from dual；否则报错！

select express　from dual where 0=2; mysq 和 oracle的行为一致：该句就如同你认为的正常表那样——会先计算where的条件，再行计算express；这里的where条件会决定expres是否会返回！

综上：

oracle的dual具有代码美感，风格统一，约定成俗！

不得不说的在sqlserver里，select 有mysql 和 oracle 结合的特点：

select \* 　　sqlserver报和mysql类似的错误——你必须指定表。这种完全就算是病句吧，估计谁也不会认的！

select express 　　sqlserver 类似于 mysql ，有效语句，返回一行记录的结果集

select express where 0=2　　sqlserver的特色：不需要from子句，考虑where条件，决定是否返回结果集

例：

select '1' as tt from DUAL;

select '1' as tt;

select '1' as tt from dual where 0=2;

### 创建临时表

CREATE TEMPORARY TABLE temp\_person (

number INT(11),

name VARCHAR(255),

birthday DATE

);

在创建表格时，您可以使用TEMPORARY关键词。只有在当前连接情况下，TEMPORARY表才是可见的。当连接关闭时，TEMPORARY表被自动取消。这意味着两个不同的连接可以使用相同的临时表名称，同时两个临时表不会互相冲突，也不与原有的同名的非临时表冲突。（原有的表被隐藏，直到临时表被取消时为止。）您必须拥有CREATE TEMPORARY TABLES权限，才能创建临时表。

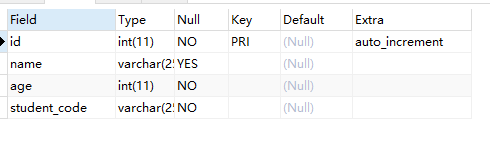
### 表结构查询

1. desc 表名【查询所有列】

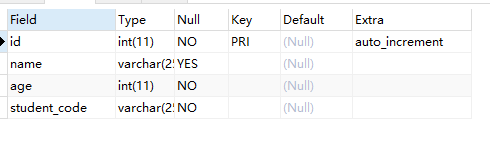
或者show columns from 表名【效果和desc一样】

或者show full columns from 表名

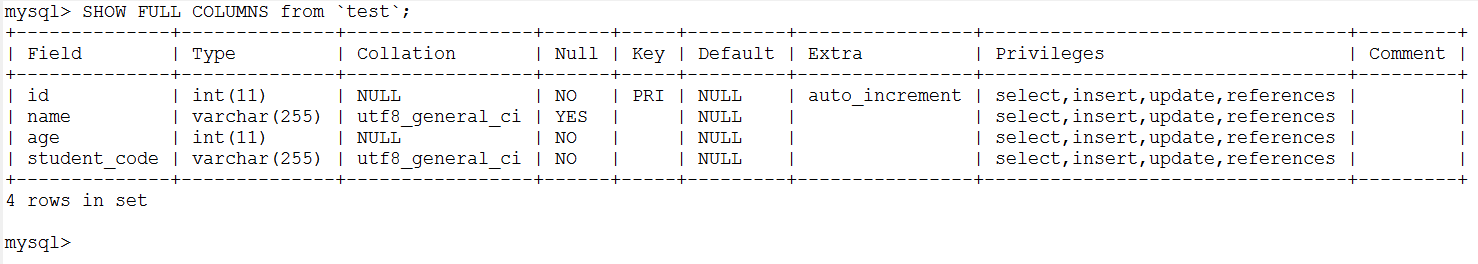
例: desc test;



例: show columns from test;

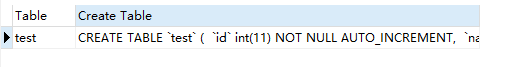


例: show full columns from test;



1. show create table 表名【查看表的创建代码】

例: show create table test;



其中Create Table部分如下所示：

CREATE TABLE `test` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) CHARACTER SET utf8 DEFAULT NULL,

`age` int(11) NOT NULL,

`student\_code` varchar(255) CHARACTER SET utf8 NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=latin1

# mysql赋权

注：赋权之后，使用flush privileges以便使得用户权限立即生效;

1. 赋权语句：grant 权限 on 数据库对象 to 用户 [with grant option]
2. 权限代表:

* select,insert,update,delete,create,drop,index,alter,grant,references,reload,shutdown,process,file等14个权限，可以用all或者all primileges 代替所有的权限

1. 数据库对象：

* 表、视图、索引、存储过程、函数等等

1. 当“数据库对象”称被\*.\*代替，表示赋予用户操作服务器上所有数据库所有表的权限。
2. 用户地址可以是localhost，也可以是ip地址、机器名字、域名。也可以用'%'表示从任何地址连接。
3. ‘连接口令’不能为空，否则创建失败。

## 查询、插入、更新、删除数据库中表数据。

1. grant 普通数据用户，查询、插入、更新、删除 数据库中所有表数据的权利

grant select, insert, update, delete on testdb.\* to username@'%'

## 创建表、索引、视图、存储过程、函数等权限。

1. grant 数据库开发人员，创建表、索引、视图、存储过程、函数等权限

grant 创建、修改、删除 MySQL 数据表结构权限。

grant create , alter, dropon testdb.\* to username @'192.168.0.%';

1. grant 操作 MySQL 外键权限。

grant references on testdb.\* to username @'192.168.0.%';

1. grant 操作 MySQL 临时表权限。

grant create temporary tables on testdb.\* to username @'192.168.0.%';

1. grant 操作 MySQL 索引权限。

grant index on testdb.\* to username @'192.168.0.%';

1. grant 操作 MySQL 视图、查看视图源代码 权限。

grant create view on testdb.\* to username @'192.168.0.%';

grant show view on testdb.\* to username @'192.168.0.%';

1. grant 操作 MySQL 存储过程、函数 权限。

grant create routine on testdb.\* to username @'192.168.0.%';

grant alter routine on testdb.\* to username @'192.168.0.%';

grant execute on testdb.\* to username @'192.168.0.%';

## 普通 DBA权限。

1. grant 普通 DBA 管理某个 MySQL 数据库的权限

grant all privileges on testdb to username @'localhost'

其中，关键字 “privileges” 可以省略。

## 高级 DBA权限。

1. grant 高级 DBA 管理 MySQL 中所有数据库的权限。

grant all on \*.\* to username @'localhost'

## grant 权限，分别作用在多个层次上

1. grant 作用在整个 MySQL 服务器上：

grant select on \*.\* to username @localhost;

* 可以查询 MySQL 中所有数据库中的表。

grant all on \*.\* to username @localhost;

* 可以管理 MySQL 中的所有数据库

1. grant 作用在单个数据库上：

grant select on testdb.\* to username @localhost;

* 可以查询 testdb 中的表。

1. grant 作用在单个数据表上：

grant select, insert, update, delete on testdb.orders to username @localhost;

1. grant 作用在表中的列上：

grant select(id, se, rank) on testdb.apache\_log to username @localhost;

1. grant 作用在存储过程、函数上：

grant execute on procedure testdb.pr\_add to username @'localhost'

grant execute on function testdb.fn\_add to username @'localhost'

## 查看 MySQL 用户权限

1. 查看当前用户（自己）权限：

* show grants;

1. 查看其他 MySQL 用户权限：

* show grants for username @localhost;

## 撤销已经赋予给 MySQL 用户权限的权限。

1. revoke 跟 grant 的语法差不多，只需要把关键字 “to” 换成 “from” 即可：

grant all on \*.\* to username @localhost;

revoke all on \*.\* from username @localhost;

## grant、revoke 用户权限注意事项

1. grant, revoke 用户权限后，该用户只有重新连接 MySQL 数据库，权限才能生效。

* 重新连接，或则flush privilege;

1. 如果想让授权的用户，也可以将这些权限 grant 给其他用户，需要选项 grant option

* grant select on testdb.\* to username @localhost with grant option;
* 这个特性一般用不到。实际中，数据库权限最好由 DBA 来统一管理。

# mysql增删改查排序等常见操作

## 查询

select \* from tableName where xxxxx group by xxxx order by xxx havin xxxx;

### 模糊查询和索引（索引与优化like查询）

1. 模糊查询：

* %：表示零个或多个字符。

　　可以匹配任意类型和任意长度的字符，有些情况下若是中文，请使用两个百分号（%%）表示。

　　比如：select \* from flow\_user where username like '%王%';

　　将会把flow\_user这张表里面，列名username中还有“王”的记录全部查询出来。

　　另外，如果需要找到flow\_user这张表里面，字段username中既有“唐”，又有“英”的记录，可以使用and条件，

　　select \* from flow\_user where username like '%英%' and username like '%唐%';

　　则可以查询出来所有包含“英”和“唐”的所有内容，“英”和“唐”所在的前后位置无所谓。

　　若是用select \* from flow\_user where username like '%英%唐%';

　　可以查出来包含“英唐”的内容，但是查不出来“唐英”的内容。

* \_:表示任意单个字符。匹配单个任意字符，它常用来限制表达式的字符长度：

　　select \* from flow\_user where username like '\_英\_';

　　只能找到“王英琨”这样username为三个字且中间一个字是“英”的内容。

　　再比如：select \* from flow\_user where username like '英\_\_';

　　只能找到“英雄点”这样username为三个字且第一个字是“英”的内容。

* []：表示括号内所列字符中的一个（类似正则表达式）。指定一个字符、字符串或范围，要求所匹配的对象为他们中的任一个。

　　select \* from flow\_user where username LIKE'[王李张]飞';

　　将找出“王飞”“李飞”“张飞”（而不是“张王李飞”）。

　　如[]内有一系列字符（01234，abcde之类的）则可略写为“0-4”，“a-e”:

　　select \* from flow\_user where username like '老[0-9]';

　　将找出“老1”、“老2”、……、“老9”；

　　Oracle 10g以上的版本用法为：

　　select \* from flow\_user where regexp\_like(username, '[张王李]飞');

* [^]：表示不在括号所列之内的单个字符。其取之和[]相同，但它所要求匹配对象为指定字符以外的任一个字符。

　　select \* from flow\_user where username LIKE'[^王李张]飞';

　　将找出不是“王飞”“李飞”“张飞“的”赵飞“、”吴飞“等。

　　注：oracle like 不支持正则，你可以使用支持like的正则regexp\_like

* 查询内容包含通配符时：

　　由于通配符的缘故，导致查询特殊字符“%”、“\_”、“[”的语句无法正常实现，把特殊字符用“[]”括起来便可以正常查询。

　　function sqlencode(str)

　　str=replace(str,"[","[[]") '此句一定要在最前

　　str=replace(str,"\_","[\_]")

　　str=replace(str,"%","[%]")

　　sqlencode=str

　　end function

* 模糊查询如何走索引：
* like %keyword 索引失效，使用全表扫描。但可以通过翻转函数+like前模糊查询+建立翻转函数索引=走翻转函数索引，不走全表扫描。

select id from table1 t where t.id like '%123'

在执行的时候，执行计划显示，消耗值，io值，cpu值均非常大，原因是like后面前模糊查询导致索引失效，进行全表扫描

解决方法：这种只有前模糊的sql可以改造如下写法

select id from table1 t where reverse (t.id) like reverse('%123')

使用翻转函数+like前模糊查询+建立翻转函数索引=走翻转函数索引，不走全扫描。有效降低消耗值，io值，cpu值这三个指标，尤其是io值的降低。

* like keyword% 索引有效。
* like %keyword% 索引失效，也无法使用反向索引。

select id from table1 t where t.id like '%123%'

可以改为：

select id from table1 t where locate('123' , t.id) > 0;

或者

select id from table1 t where position('123' in t.id) > 0;

或者

select id from table1 t where instr ( t.id , '123' ) > 0;

## 排序

1. order by 列名
2. order by field(列名, value1,…………)

mysql自有特性：

mysql order by field 按指定字段排序

SELECT \* FROM t\_goods\_class

WHERE 1=1 AND f\_leaf = 0

and parent\_id in (1211,1136,439345,1031)

AND type = 1

ORDER BY field(parent\_id,1211,1136,439345,1031), SEQ ASC

## 数据库触发联合索引的几个条件

本文主要总结查询语句触发联合索引（索引定义中至少包含两个索引列）的几种条件。

示例如下。首先创建表：

CREATE TABLE E (e1 INT, e2 VARCHAR(9), e3 INT, PRIMARY KEY(e1, e3));

这样就建立了一个联合索引：e1,e3

触发联合索引是有条件的：

1、使用联合索引的全部索引键，可触发索引的使用。

例如：SELECT E.\* FROM E WHERE E.e1=1 AND E.e3=2

2、使用联合索引的前缀部分索引键，如“key\_part\_1 <op>常量”，可触发索引的使用。

例如：SELECT E.\* FROM E WHERE E.e1=1

3、使用部分索引键，但不是联合索引的前缀部分，如“key\_part\_2 <op>常量”，不可触发索引的使用。

例如：SELECT E.\* FROM E WHERE E.e3=1

4、使用联合索引的全部索引键，但索引键不是AND操作，不可触发索引的使用。

例如：SELECT E.\* FROM E WHERE E.e3=2 OR E.e1=1

# mysql常用内置函数

## 数学函数

|  |  |
| --- | --- |
| ABS(x) | 返回x的绝对值 |
| CEIL(x)或CEILING(x) | 返回大于或等于x的最小整数（向上取整） |
| FLOOR(x) | 返回小于或等于x的最大整数（向下取整） |
| RAND() | 返回0~1的随机数 |
| RAND(x) | 返回0~1的随机数，x值相同时返回的随机数相同 |
| SIGN(x) | 返回x的符号，x是负数、0、正数分别返回-1、0、1 |
| PI() | 返回圆周率 |
| TRUNCATE(x,y) | 返回数值x保留到小数点后y位的值 |
| ROUND(x) | 返回离x最近的整数（四舍五入） |
| ROUND(x,y) | 保留x小数点后y位的值，但截断时要四舍五入 |
| POW(x,y)或POWER(x,y) | 返回x的y次方 |
| SQRT(x) | 返回x的平方根 |
| EXP(x) | 返回e的x次方 |
| MOD(x | 返回x 除以y以后的余数 |
| LOG(x) | 返回自然对数（以e为底的对数） |
| LOG10(x) | 返回以10为底的对数 |
| RADIANS(x) | 讲角度转换为弧度 |
| DEGREES(x) | 讲弧度转换为角度 |
| SIN(x) | 求正弦值 |
| ASIN(x) | 求反正弦值 |
| COS(x) | 求余弦值 |
| ACOS(x) | 求反余弦值 |
| TAN(x) | 求正切值 |
| ATAN(x)或ATAN(x,y) | 求反正切值 |
| COT(x) | 求余切值 |

## 字符串函数

|  |  |
| --- | --- |
| CHAR\_LENGTH(s) | 返回字符串s的字符数 |
| LENGTH(s) | 返回字符串s的长度 |
| CONCAT(s1,s2,.....) | 将字符串s1,s2等多个字符串合并为一个字符串 |
| CONCAT\_WS(x,s1,s2,.....) | 同CONCAT(s1,s2,.....)，以'x'为分割连接字符串 |
| INSERT(s1,x,len,s2) | 将字符串s2替换s1的x位置开始长度为len的字符串 |
| UPPER(s),UCASE(s) | 讲字符串s的所有字符都变成大写字母 |
| LOWER(s),LCASE(s) | 讲字符串s的所有字符都变成小写字母 |
| LEFT(s,n) | 返回字符串s的前n个字符 |
| RIGHT(s,n) | 返回字符串s的后n个字符 |
| LPAD(s1,len,s2) | 字符串s2来填充s1的开始处，使字符串长度达到len |
| RPAD(s1,len,s2) | 字符串s2来填充s1的结尾处，使字符串长度达到len |
| LTRIM(s) | 去掉字符串s开始处的空格 |
| RTRIM(s) | 去掉字符串s结尾处的空格 |
| TRIM(s) | 去掉字符串s开始处和结尾处的空格 |
| TRIM(s1 FROM s) | 去掉字符串s中开始处和结尾处的字符串s1 |
| REPEAT(s,n) | 将字符串s重复n次 |
| SPACE(n) | 返回n个空格 |
| REPLACE(s,s1,s2) | 用字符串s2代替字符串s中的字符串s1 |
| STRCMP(s1,s2) | 比较字符串s1和s2 |
| SUBSTRING(s,n,len) | 获取从字符串s中的第n个位置开始长度为len的字符串 |
| MID(s,n,len) | 同SUBSTRING(s,n,len) |
| ATE(s1,s),POSTTION(s1 IN s),INSTR(s,s1) | 从字符串s中获取s1的开始位置 |
| REVERSE(s) | 将字符串s的顺序反过来 |
| ELT(n,s1,s2...) | 返回第n个字符串 |
| FIELD(s,s1,s2...) | 返回第一个与字符串s匹配的字符串的位置 |
| FIND\_IN\_SET(s1,s2) | 返回在字符串s2中与s1匹配的字符串的位置 |
| MAKE\_SET(x,s1,s2...) | 按x的二进制数从s1,s2......sn中选取字符串 |

## 日期、时间相关函数

|  |  |
| --- | --- |
| CURDATE(),CURRENT\_DATE() | 返回当前日期 |
| CURTIME(),CURRENT\_TIME() | 返回当前时间 |
| NOW(),CURRENT\_TIMESTAMP() |  |
| LOCALTIME(),SYSDATE() |  |
| LOCALTIMESTAMP() | 返回当前日期和时间 |
| UNIX\_TIMESTAMP() | 以UNIX时间戳的形式返回当前时间 |
| UNIX\_TIMESTAMP(d) | 将时间d以UNIX时间戳的形式返回 |
| FROM\_UNIXTIME(d) | 把UNIX时间戳的时间转换为普通格式的时间 |
| UTC\_DATE() | 返回UTC(国际协调时间)日期 |
| UTC\_TIME() | 返回UTC时间 |
| MONTH(d) | 返回日期d中的月份值，范围是1~12 |
| MONTHNAME(d) | 返回日期d中的月份名称，如january |
| DAYNAME(d) | 返回日期d是星期几，如Monday |
| DAYOFWEEK(d) | 返回日期d是星期几，1表示星期日，2表示星期2 |
| WEEKDAY(d) | 返回日期d是星期几，0表示星期一,1表示星期2 |
| WEEK(d) | 计算日期d是本年的第几个星期，范围是0-53 |
| WEEKOFYEAR(d) | 计算日期d是本年的第几个星期，范围是1-53 |
| DAYOFYEAR(d) | 计算日期d是本年的第几天 |
| DAYOFMONTH(d) | 计算日期d是本月的第几天 |
| YEAR(d) | 返回日期d中的年份值 |
| QUARTER(d) | 返回日期d是第几季度，范围1-4 |
| HOUR(t) | 返回时间t中的小时值 |
| MINUTE(t) | 返回时间t中的分钟值 |
| SECOND(t) | 返回时间t中的秒钟值 |
| EXTRACT(type FROM d) | 从日期d中获取指定的值，type指定返回的值，如YEAR,HOUR等 |
| TIME\_TO\_SEC(t) | 将时间t转换为秒 |
| SEC\_TO\_TIME(s) | 将以秒为单位的时间s转换为时分秒的格式 |
| TO\_DAYS(d) | 计算日期d到0000年1月1日的天数 |
| FROM\_DAYS(n) | 计算从0000年1月1日开始n天后的日期 |
| DATEDIFF(d1,d2) | 计算日期d1到d2之间相隔的天数 |
| ADDDATE(d,n) | 计算开始日期d加上n天的日期 |
| ADDDATE(d, INTERVAL expr type) | 计算起始日期d加上一个时间段后的日期 |
| SUBDATE(d,n) | 计算起始日期d减去n天的日期 |
| SUBDATE(d, INTERVAL expr type) | 计算起始日期d减去一个时间段后的日期 |
| ADDTIME(t,n) | 计算起始时间t加上n秒的时间 |
| SUBTIME(t,n) | 计算起始时间t减去n秒的时间 |
| DATE\_FORMAT(d,f) | 按照表达式f的要求显示日期d |
| TIME\_FORMAT(t,f) | 按照表达式f的要求显示时间t |
| GET\_FORMAT(type,s) | 根据字符串s获取type类型数据的显示格式 |

## 条件判断函数

### IF(expr,v1,v2)函数

IF(expr,v1,v2)函数中，如果表达式expr成立，返回结果v1,否则，返回结果v2。

### CASE函数

CASE WHEN expr1 THEN v1 [WHEN expr2 THEN v2...][ELSE vn] END

## 系统信息函数

|  |  |
| --- | --- |
| VERSION() | 返回数据库的版本号 |
| CONNECTION\_ID() | 返回服务器的连接数，也就是到现在为止mysql服务的连接次数 |
| DATABASE(),SCHEMA() | 返回当前数据库名 |
| USER() | 返回当前用户的名称 |
| CHARSET(str) | 返回字符串str的字符集 |
| COLLATION(str) | 返回字符串str的字符排列方式 |
| LAST\_INSERT\_ID() | 返回最后生成的auto\_increment值 |

## 加密解密函数

|  |  |
| --- | --- |
| PASSWORD(str) | 对字符串str进行加密 |
| MD5(str) | 对字符串str进行加密 |
| ENCODE(str,pswd\_str) | 使用字符串pswd\_str来加密字符串str,加密结果是一个二进制数，必须使用BLOB类型来保持它 |
| DECODE(crypt\_str,pswd\_str) | 解密函数，使用字符串pswd\_str来为crypt\_str解密 |

## 其他函数

|  |  |
| --- | --- |
| FORMAT(x,n) | 格式化函数，可以讲数字x进行格式化，将x保留到小数点后n位，这个过程需要进行四舍五入。 |
| ASCII(s) | 返回字符串s的第一个字符的ASSCII码 |
| BIN(x) | 返回x的二进制编码 |
| HEX(x) | 返回x的十六进制编码 |
| OCT(x) | 返回x的八进制编码 |
| CONV(x,f1,f2) | 将x从f1进制数变成f2进制数 |
| INET\_ATON(IP) | 将IP地址转换为数字表示，IP值需要加上引号 |
| INET\_NTOA(n) | 可以将数字n转换成IP的形式 |
| GET\_LOCT(name,time) | 加锁函数，定义一个名称为name、持续时间长度为time秒的锁，如果锁定成功，返回1，如果尝试超时，返回0，如果遇到错误，返回NULL. |
| RELEASE\_LOCK(name) | 解除名称为name的锁，如果解锁成功，返回1，如果尝试超时，返回0，如果解锁失败，返回NULL。 |
| IS\_FREE\_LOCK(name) | 判断是否使用名为name的锁，如果使用，返回0，否则返回1. |
| CONVERT(s USING cs) | 将字符串s的字符集变成cs |
| CAST(x AS type),CONVERT(x,type) | 这两个函数将x变成type类型，这两个函数只对BINARY,CHAR,DATE,DATETIME,TIME,SIGNED INTEGER,UNSIGNED INTEGER这些类型起作用，但这两种方法只是改变了输出值得数据类型，并没有改变表中字段的类型。 |

# mysqlc常用命令

## 输出当前所用的数据库：

select database();

或则

show tables;

可以推测出

或则

在控制台输入

status

从查询结果中可以看出

## 输出版本号

select version();

或则

在控制台输入

status

从查询结果中可以看出

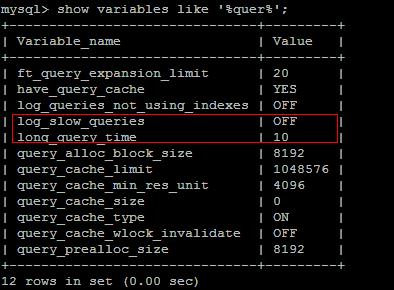
# mysql慢查询

## 慢查询作用

它能记录下所有执行超过long\_query\_time时间的SQL语句, 帮你找到执行慢的SQL, 方便我们对这些SQL进行优化.

## 如何开启慢查询

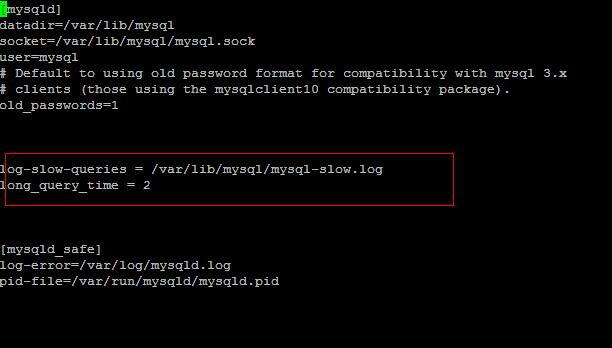
首先我们先查看MYSQL服务器的慢查询状态是否开启.执行如下命令:



我们可以看到当前log\_slow\_queries状态为OFF, 说明当前并没有开启慢查询.

开启慢查询非常简单, 操作如下:

Linux下找到mysql的配置文件my.ini, 在mysqld下方加入慢查询的配置语句(注意:一定要在[mysqld]下的下方加入)



log-slow-queries: 代表MYSQL慢查询的日志存储目录, 此目录文件一定要有写权限；

Windows下需要写绝对路径，如：log-slow-queries="C:/Program Files/MySQL/MySQL Server 5.5/log/mysql-slow.log"

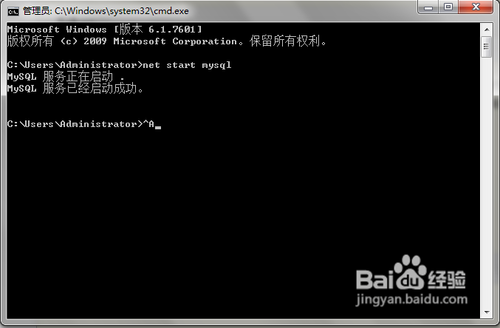
long\_query\_time: 最长执行时间. (如图, MSYQL将记录下所有执行时间超过2条的SQL语句, 此处为测试时间, 时间不应太小最好在5-10秒之内, 当然可以根据自己的标准而定);

配置好以后重新启动一个MYSQL服务

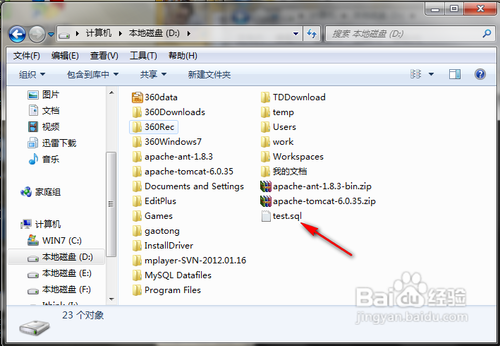
# 导入数据

## 命令行导入

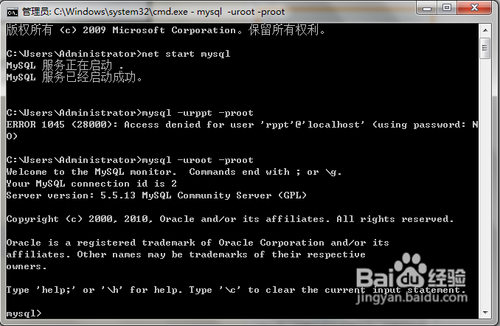
首先我们使用MySQL提供的命令行界面来导入数据库，确保自己的电脑中安装了MySQL数据库，我们可以通过命令行来确认是否安装了MySQL数据库，当然，第一步是打开Mysql的数据库服务，我们使用命令行来打开:如下图所示:



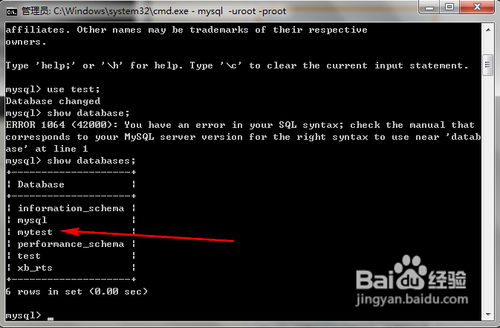
启动MySQL后，我们找到需要用到的脚本文件，也就是数据库文件，当然，我们首先得建立一个数据库，这样才可以导入脚本，如下图所示:



我们在将脚本拷到本地磁盘的根目录，这样方便进入找到脚本，这里以D盘来说明，使用test.sql:接着我们来到命令行，使用SOURCE d:/test.sql;来导入数据库,先进入mysql，如下图所示:



首先要在数据库中建立好数据库，然后导入脚本，所以先建立一个数据库哦，不要脚本是不知道你要往哪个数据库中导入脚本的，如下图所示:

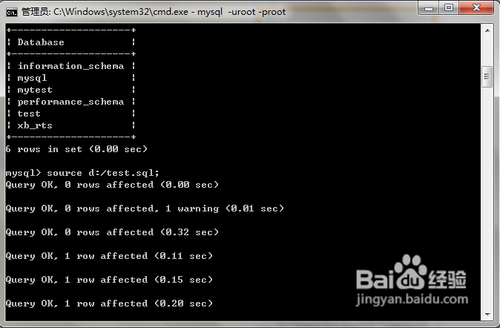


然后就可以输入导入.sql文件命令：

mysql> USE 数据库名;

mysql> SOURCE d:/test.sql;

如下图所示:



看到上面的画面，说明mysql数据库已经导入成功了哦!

# mysql调优

## 查看MySQL运行情况

SHOW STATUS;

## 查看INNODB数据库引擎运行状态

SHOW ENGINE INNODB STATUS;

## 查看当前正在进行的进程，对于有锁表等情况的排查很有用处

SHOW PROCESSLIST; 默认显示前100条

SHOW FULL PROCESSLIST; 显示所有

## 查看MySQL的配置参数

SHOW VARIABLES;

## 查看当前已经被打开的表列表

SHOW OPEN TABLES;

注意：一般情况下，打开MySQL的慢查询记录同样有利于排查，如何开启慢查询，请点击。

## SHOW STATUS;关键结果释义

|  |  |
| --- | --- |
| Aborted\_clients | 由于客户没有正确关闭连接已经死掉，已经放弃的连接数量。 |
| Aborted\_connects | 尝试已经失败的MySQL服务器的连接的次数。 |
| Connections | 试图连接MySQL服务器的次数 |
| Created\_tmp\_tables | 当执行语句时，已经被创造了的隐含临时表的数量。 |
| Delayed\_insert\_threads | 正在使用的延迟插入处理器线程的数量。 |
| Delayed\_writes | 用INSERT DELAYED写入的行数。 |
| Delayed\_errors | 用INSERT DELAYED写入的发生某些错误(可能重复键值)的行数。 |
| Flush\_commands | 执行FLUSH命令的次数。 |
| Handler\_delete | 请求从一张表中删除行的次数。 |
| Handler\_read\_first | 请求读入表中第一行的次数。 |
| Handler\_read\_key | 请求数字基于键读行。 |
| Handler\_read\_next | 请求读入基于一个键的一行的次数。 |
| Handler\_read\_rnd | 请求读入基于一个固定位置的一行的次数。 |
| Handler\_update | 请求更新表中一行的次数。 |
| Handler\_write | 请求向表中插入一行的次数。 |
| Key\_blocks\_used | 用于关键字缓存的块的数量。 |
| Key\_read\_requests | 请求从缓存读入一个键值的次数。 |
| Key\_reads | 从磁盘物理读入一个键值的次数。 |
| Key\_write\_requests | 请求将一个关键字块写入缓存次数。 |
| Key\_writes | 将一个键值块物理写入磁盘的次数。 |
| Max\_used\_connections | 同时使用的连接的最大数目。 |
| Not\_flushed\_key\_blocks | 在键缓存中已经改变但是还没被清空到磁盘上的键块。 |
| Not\_flushed\_delayed\_rows | 在INSERT DELAY队列中等待写入的行的数量。 |
| Open\_tables | 打开表的数量。 |
| Open\_files | 打开文件的数量。 |
| Open\_streams | 打开流的数量(主要用于日志记载） |
| Opened\_tables | 已经打开的表的数量。 |
| Questions | 发往服务器的查询的数量。 |
| Slow\_queries | 要花超过long\_query\_time时间的查询数量。 |
| Threads\_connected | 当前打开的连接的数量。 |
| Threads\_running | 不在睡眠的线程数量。 |
| Uptime | 服务器工作了多少秒。 |

## 提升性能的建议

1.如果opened\_tables太大,应该把my.cnf中的table\_cache变大

2.如果Key\_reads太大,则应该把my.cnf中key\_buffer\_size变大.可以用Key\_reads/Key\_read\_requests计算出cache失败率

3.如果Handler\_read\_rnd太大,则你写的SQL语句里很多查询都是要扫描整个表,而没有发挥索引的键的作用

4.如果Threads\_created太大,就要增加my.cnf中thread\_cache\_size的值.可以用Threads\_created/Connections计算cache命中率

5.如果Created\_tmp\_disk\_tables太大,就要增加my.cnf中tmp\_table\_size的值,用基于内存的临时表代替基于磁盘的

附2 SHOW PROCESSLIST;结果释义

id列:一个标识，你要kill 一个语句的时候很有用。

user列: 显示当前用户，如果不是root，这个命令就只显示你权限范围内的sql语句。

host列:显示这个语句是从哪个ip 的哪个端口上发出的。可用来追踪出问题语句的用户。

db列:显示这个进程目前连接的是哪个数据库。

command列:显示当前连接的执行的命令，一般就是休眠（sleep），查询（query），连接（connect）。

time列:此这个状态持续的时间，单位是秒。

state列:显示使用当前连接的sql语句的状态，很重要的列，后续会有所有的状态的描述，请注意，state只是语句执行中的某一个状态，一个sql语句，已查询为例，可能需要经过copying to tmp table，Sorting result，Sending data等状态才可以完成。

info列:显示这个sql语句，因为长度有限，所以长的sql语句就显示不全，但是一个判断问题语句的重要依据。

state列具体释义：

Checking table 正在检查数据表（这是自动的）。

Closing tables 正在将表中修改的数据刷新到磁盘中，同时正在关闭已经用完的表。这是一个很快的操作，如果不是这样的话，就应该确认磁盘空间是否已经满了或者磁盘是否正处于重负中。

Connect Out 复制从服务器正在连接主服务器。

Copying to tmp table on disk 由于临时结果集大于 tmp\_table\_size，正在将临时表从内存存储转为磁盘存储以此节省内存。

Creating tmp table 正在创建临时表以存放部分查询结果。

deleting from main table 服务器正在执行多表删除中的第一部分，刚删除第一个表。

deleting from reference tables 服务器正在执行多表删除中的第二部分，正在删除其他表的记录。

Flushing tables 正在执行 FLUSH TABLES，等待其他线程关闭数据表。

Killed 发送了一个kill请求给某线程，那么这个线程将会检查kill标志位，同时会放弃下一个kill请求。MySQL会在每次的主循环中检查kill标志位，不过有些情况下该线程可能会过一小段才能死掉。如果该线程程被其他线程锁住了，那么kill请求会在锁释放时马上生效。

Locked 被其他查询锁住了。

Sending data 正在处理 SELECT 查询的记录，同时正在把结果发送给客户端。

Sorting for group 正在为 GROUP BY 做排序。

Sorting for order 正在为 ORDER BY 做排序。

Opening tables 这个过程应该会很快，除非受到其他因素的干扰。例如，在执 ALTER TABLE 或 LOCK TABLE 语句行完以前，数据表无法被其他线程打开。 正尝试打开一个表。

Removing duplicates 正在执行一个 SELECT DISTINCT 方式的查询，但是MySQL无法在前一个阶段优化掉那些重复的记录。因此，MySQL需要再次去掉重复的记录，然后再把结果发送给客户端。

Reopen table 获得了对一个表的锁，但是必须在表结构修改之后才能获得这个锁。已经释放锁，关闭数据表，正尝试重新打开数据表。

Repair by sorting 修复指令正在排序以创建索引。

Repair with keycache 修复指令正在利用索引缓存一个一个地创建新索引。它会比 Repair by sorting 慢些。

Searching rows for update 正在讲符合条件的记录找出来以备更新。它必须在 UPDATE 要修改相关的记录之前就完成了。

Sleeping 正在等待客户端发送新请求.

System lock 正在等待取得一个外部的系统锁。如果当前没有运行多个 mysqld 服务器同时请求同一个表，那么可以通过增加 –skip-external-locking参数来禁止外部系统锁。

Upgrading lock INSERT DELAYED 正在尝试取得一个锁表以插入新记录。

Updating 正在搜索匹配的记录，并且修改它们。

User Lock 正在等待 GET\_LOCK()。

Waiting for tables 该线程得到通知，数据表结构已经被修改了，需要重新打开数据表以取得新的结构。然后，为了能的重新打开数据表，必须等到所有其他线程关闭这个表。

以下几种情况下会产生这个通知：FLUSH TABLES tbl\_name, ALTERTABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE, 或 OPTIMIZE TABLE。

waiting for handler insert INSERT DELAYED 已经处理完了所有待处理的插入操作，正在等待新的请求。 大部分状态对应很快的操作，只要有一个线程保持同一个状态好几秒钟，那么可能是有问题发生了，需要检查一下。

还有其它的状态没在上面中列出来，不过它们大部分只是在查看服务器是否有存在错误是才用得着。

## 查看MySQL服务器配置信息

show variables;

## 查看MySQL服务器运行的各种状态值

show global status;

3, 慢查询

show variables like '%slow%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| log\_slow\_queries | OFF |
| slow\_launch\_time | 2 |

mysql> show global status like '%slow%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Slow\_launch\_threads | 0 |
| Slow\_queries | 279 |

配置中关闭了记录慢查询（最好是打开，方便优化，开启慢查询），超过2秒即为慢查询，一共有279条慢查询

4, 连接数

mysql> show variables like 'max\_connections';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| max\_connections | 500 |
|  |  |

mysql> show global status like 'max\_used\_connections';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Max\_used\_connections | 98 |
|  |  |

设置的最大连接数是500，而响应的连接数是498

max\_used\_connections / max\_connections \* 100% = 99.6% （理想值 ≈ 85%）

5, key\_buffer\_size key\_buffer\_size是对MyISAM表性能影响最大的一个参数, 不过数据库中多为Innodb

mysql> show variables like 'key\_buffer\_size';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| key\_buffer\_size | 67108864 |
|  |  |

mysql> show global statuslike 'key\_read%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Key\_read\_requests | 25629497 |
| Key\_reads | 66071 |

一共有25629497个索引读取请求，有66071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% =0.27% 需要适当加大key\_buffer\_size

mysql> show global status like 'key\_blocks\_u%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Key\_blocks\_unused | 10285 |
| Key\_blocks\_used | 47705 |

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 Key\_blocks\_used / (Key\_blocks\_unused + Key\_blocks\_used) \* 100% ≈ 18% （理想值 ≈ 80%）

6， 临时表

mysql> show global status like 'created\_tmp%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Created\_tmp\_disk\_tables | 4184337 |
| Created\_tmp\_files | 4124 |
| Created\_tmp\_tables | 4215028 |

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： Created\_tmp\_disk\_tables / Created\_tmp\_tables \* 100% ＝ 99% （理想值<= 25%）

mysql> show variables where Variable\_name in ('tmp\_table\_size', 'max\_heap\_table\_size');

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| max\_heap\_table\_size | 134217728 |
| tmp\_table\_size | 134217728 |

需要增加tmp\_table\_size

7,open table 的情况

mysql> show global status like 'open%tables%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Open\_tables | 1024 |
| Opened\_tables | 1465 |

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 | 1024 |
|  |  |

Open\_tables / Opened\_tables \* 100% =69% 理想值 （>= 85%） Open\_tables / table\_cache \* 100% = 100% 理想值 (<= 95%)

8, 进程使用情况

mysql> show global status like 'Thread%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Threads\_cached | 31 |
| Threads\_connected | 239 |
| Threads\_created | 2914 |
| Threads\_running | 4 |

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 | 32 |
|  |  |

9, 查询缓存(query cache)

mysql> show global status like 'qcache%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Qcache\_free\_blocks | 2226 |
| Qcache\_free\_memory | 10794944 |
| Qcache\_hits | 5385458 |
| Qcache\_inserts | 1806301 |
| Qcache\_lowmem\_prunes | 433101 |
| Qcache\_not\_cached | 4429464 |
| Qcache\_queries\_in\_cache | 7168 |
| Qcache\_total\_blocks | 16820 |

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 | 33554432 |
| query\_cache\_min\_res\_unit | 4096 |
| query\_cache\_size | 33554432 |
| 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。

10,排序使用情况

mysql> show global status like 'sort%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Sort\_merge\_passes | 2136 |
| Sort\_range | 81888 |
| Sort\_rows | 35918141 |
| Sort\_scan | 55269 |
|  |  |

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，见 How fast can you sort data with MySQL?（引自http://qroom.blogspot.com/2007/09/mysql-select-sort.html）

另外，增加read\_rnd\_buffer\_size(3.2.3是record\_rnd\_buffer\_size)的值对排序的操作也有一点的好处，参见：http://www.mysqlperformanceblog.com/2007/07/24/what-exactly-is- read\_rnd\_buffer\_size/

11.文件打开数(open\_files)

mysql> show global status like 'open\_files';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Open\_files | 821 |
|  |  |
|  |  |
|  |  |
|  |  |

mysql> show variables like 'open\_files\_limit';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| open\_files\_limit | 65535 |
|  |  |
|  |  |
|  |  |

比较合适的设置：Open\_files / open\_files\_limit \* 100% <= 75％

正常

12。 表锁情况

mysql> show global status like 'table\_locks%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Table\_locks\_immediate | 4257944 |
| Table\_locks\_waited | 25182 |
|  |  |
|  |  |

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.

13. 表扫描情况

mysql> show global status like 'handler\_read%';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Handler\_read\_first | 108763 |
| Handler\_read\_key | 92813521 |
| Handler\_read\_next | 486650793 |
| Handler\_read\_prev | 688726 |
| Handler\_read\_rnd | 9321362 |
| Handler\_read\_rnd\_next | 153086384 |

各字段解释参见:

http://hi.baidu.com/thinkinginlamp/blog/item/31690cd7c4bc5cdaa144df9c.html，

调出服务器完成的查询请求次数：

mysql> show global status like 'com\_select';

|  |  |
| --- | --- |
| **Variable\_name** | **Value** |
| Com\_select | 2693147 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

计算表扫描率：

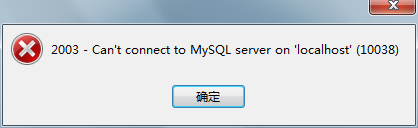
表扫描率 ＝ Handler\_read\_rnd\_next / Com\_select

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。

# 

# mysql常见错误以及解决方法

## 远程连接10038错误



解决方法：

查一下你的MYSQL用户表里, 是否允许远程连接

1. 授权

mysql>grant all privileges on \*.\* to 'root'@'%' identified by 'youpassword' with grant option;

mysql>flush privileges;

1. 修改/app/mysql/mysql-5.6.24-linux-glibc2.5-i686/my.conf

找到bind-address = 127.0.0.1这一行

改为bind-address = 0.0.0.0即可