MySQL
Day1

下载：mysql-5.7.17.tar

重新克隆新的虚拟机：

eth0 网卡:192.168.4.50-192.168.4.57

主机名称:mysql50-mysql57

案例一：安装部署 MySQL

准备工作(非必须的操作)：

关闭防火墙、关闭 SELinux、卸载 mariadb

```
systemctl stop mariadb
rm -rf /etc/my.cnf
rm -rf /var/lob/mysql
rpm -e --nodeps mariadb mariadb-server mariadb-devel
```

安装部署 MySQL

真机操作

```
[root@GYP-work~]# tar xf mysql-5.7.17.tar -C /var/ftp/mysql
[root@GYP-work~]# vim mysql.repo
[mysql]
name=mysql5.7
enabled=1
baseurl=ftp://192.168.4.254/mysql
gpgcheck=0

[root@GYP-work~]# for i in 192.168.4.{50..58};do
scp mysql.repo $i:/etc/yum.repos.d/
done

[root@GYP-work~]# for i in 192.168.4.{50..58};do
ssh $i "yum -y install mysql-community*"
done
```

启动服务

```
[root@mysql50~]# systemctl restart mysqld
[root@mysql50~]# systemctl enable mysqld
```

配置 MySQL 管理员密码(默认数据库管理员账号为 root)

```
[root@mysql50 ~]# grep "temporary password" /var/log/mysqld.log
[root@mysql50 ~]# mysql -uroot -p'rln/g-Da*7fI'
```

登陆服务端后必须马上修改密码，不然会报错：

```
mysql> show databases;
ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing
this statement.
```

策略参数	值	描述
validate_password_policy	0 或 LOW	长度
	1 或 MEDIUM(默认)	长度；数字、大写/小写,特殊符号
	2 或 STRONG	长度；数字、大写/小写、特殊符号；字典文件

```
mysql> set global validate_password_policy=0;
```

```
mysql> set global validate_password_length=6;
mysql> alter user user() identified by '123456';

vim /etc/my.cnf
validate_password_policy=0
validate_password_length=6

systemctl restart mysqld
```

初始化脚本 init_mysql.sh

```
#!/bin/bash
PASSWORD=`awk '/temporary password/{print $NF}' /var/log/mysqld.log`
mysql --connect-expired-password -uroot -p"$PASSWORD" -e "set global validate_password_policy=0"
mysql --connect-expired-password -uroot -p"$PASSWORD" -e "set global validate_password_length=6"
mysql --connect-expired-password -uroot -p"$PASSWORD" -e "alter user user() identified by '123456'"
sed -i '/\[mysqld\]/a validate_password_policy=0\nvalidate_password_length=6' /etc/my.cnf
systemctl restart mysqld
```

案例二：数据库基本管理

1.数据库操作基本流程

连接登陆数据库

创建数据库

创建数据表

插入数据记录

断开连接

连接 MySQL 数据库的命令语法格式：

mysq [-h 服务器 ip 或域名 -u 用户名 -p 密码 -P 端口号 数据库名]

注意事项：

操作指令不区分大小写(密码和变量除外)

每条 SQL 语句都以分号;结束

默认不支持 Tab 键补齐（可以自行下载安装 https://github.com/dbcli/mycli）工具实现自动补齐

\c 可以取消书写错误的命令

常用的 SQL 命令分类：

DDL 数据定义语言(create, alter, drop)

DML 数据操作语言(insert, update, delete)

DCL 数据控制语言(grant, revoke)

DTL 数据事物语言(commit, rollback, savepoint)

数据库基本操作

```
mysql> show databases;      //查看数据库
mysql> use mysql;           //切换数据库
mysql> select database();   //自己在哪个数据库
mysql> create database tts character set utf8;       //创建数据库，支持中文
```

```
mysql> drop database tts;                                    //删除数据库
```

提示：数据库命名规则

（数字、字母、下划线，不能纯数字；区分大小写；不能使用关键字或特殊符号）

创建数据表基本语法格式如下：

```
create table 数据库名称.数据表名称(
   字段名 1      数据类型(宽度)      约束条件,
   字段名 2      数据类型(宽度)      约束条件,
   ......
);
```

创建一个下图所示的数据库，数据库名称为 school，数据表名称为 student

学号	姓名	性别	手机号	通信地址
NSD131201	张三	男	13089468298	朝阳区劲松南路
NSD131202	韩梅梅	女	13722233333	海淀区北三环西路
NSD131203	王五	男	18023445678	丰台区兴隆中街

```
mysql> show character set;        //查看所有可用编码
mysql> create database school character set utf8;
mysql> create table student(学号 char(20),姓名 char(20),性别 char(5),手机号 int(11),通信地址 char(50));
mysql> desc school.student;       //查看表结构
+--------------+----------+------+-----+---------+-------+
| Field        | Type     | Null | Key | Default | Extra |
+--------------+----------+------+-----+---------+-------+
| 学号         | char(20) | YES  |     | NULL    |       |
| 姓名         | char(20) | YES  |     | NULL    |       |
| 性别         | char(5)  | YES  |     | NULL    |       |
| 手机号       | int(11)  | YES  |     | NULL    |       |
| 通信地址     | char(50) | YES  |     | NULL    |       |
+--------------+----------+------+-----+---------+-------+
mysql> show create database school;     //查看数据库信息
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| school   | CREATE DATABASE `school` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+-----------------------------------------------------------------+
```

插入数据的语法格式：insert into 数据库名称.数据库表名称 values(值列表);

```
mysql> insert into student values('NSD18100','葫芦娃','男',1383888888,'北海道');

mysql> select 姓名,手机号 from school.student;
mysql> select * from school.student;    //生产环境一定不能用这一条
+----------+----------+--------+-----------+--------------+
| 学号     | 姓名     | 性别   | 手机号    | 通信地址     |
```

```
+----------+----------+--------+------------+--------------+
| NSD18100 | 葫芦娃     | 男      | 1383888888 | 北海道        |
| NSD18101 | 蛇精       | 女      | 1383889999 | 上海          |
| NSD18102 | 爷爷       | 男      | 1387777777 | 北京          |
+----------+----------+--------+------------+--------------+
```

更新数据语法格式：update 数据库名称.数据表名称 set 字段=值[where 条件]

```
mysql> update student set 性别="女";
mysql> update student set 性别='男' where 姓名='葫芦娃';
```

删除数据

```
mysql> delete from school.student where 学号='NSD181003';
mysql> delete from school.student;
```

删除数据表

```
mysql> drop table school.student;
```

案例三：MySQL 数据类型

1.数字类型

类型	大小	范围(有符号)	范围(无符号)	用途
tinyint	1 字节	-128~127	0~255	微小整数
smallint	2 字节	-32768~32767	0~65535	小整数
mediumint	3 字节	-2^{23}~2^{23}-1	0~2^{24}-1	中整数
int	4 字节	-2^{31}~2^{31}-1	0~2^{32}-1	大整数
bigint	8 字节	-2^{63}~2^{63}-1	0~264-1	极大整数
float	4 字节			单精度浮点数(小数点)
double	8 字节			双精度浮点数(小数点)
decimal	Decimal(M,D)，其中 M 为有效位数，D 为小数位数，M 应大于 D，占用 M+2 字节			
unsigned	标记使用无符号存储			

```
mysql> create table school.num(id tinyint,age int(3),score float(4,2));
```

4 表示显示 4 位，小数位占 2 位。

```
mysql> insert into num values(125,30,95.8);
```

注意：()位数不是那么重要，只是为了显示，存储大小由数据类型决定。

2.字符类型

类型	描述
char(字符数)	固定长度，最大长度 255 字符，不够指定的字符数时自动在右边填补空格，超出指定字符数则无法写入。
varchar(字符数)	可变长度，根据实际数据大小分配存储空间，超出指定字符数则无法写入。
text/blob	字符数大于 65535 时使用

char(3)这是固定长度，就是 3 个字符长度

从性能上来说，char 速度远远高于 varchar

```
mysql> create table school.info(name char(4), email varchar(30));
```

描述信息用 varchar 更好

```
mysql> insert into school.info values('我','abc@163.com');
```

3.日期时间类型

datetime 日期时间类型，占 8 个字符

范围 1000-10-10 00:00:00:000000~9999-12-31 23:59:59:999999

如果不给该类型的数据赋值，则默认为 NULL

timestamp 日期时间类型，占 4 个字节

范围 1970-01-01 00:00:00:000000~2038-01-19 03:14:07.999999

如果不给该类型的数据赋值，则 mysql 自动为其分配当前的系统时间

时间格式：YYYYmmddhhmmss

data 日期类型，占用 4 个字节

范围 0001-01-01~9999-12-31

默认使用 4 位数字表示，当只有 2 位数字赋值时

01-69 自动识别为 2001-2069

70-99 自动识别为 1970~1999

year 年份类型，占用 1 个字节

范围 1901-2155

time 时间类型，占用 3 个字节

范围 HH:MM:SS

创建学员信息表：姓名、出生日期、入学年份、上课时间、下课时间

```
mysql> create table school.stuinfo(name char(5),bitrh datetime,start year,begin time,end time);
mysql> insert into stuinfo values('tom',20181010080700,2019,090000,180000);
mysql> insert into stuinfo values('bob',"2018-11-11 10:18:15",2019,090000,180000);
```

4.字符类型

枚举类型（选择类型）

enum(值 1，值 2，值 3)　　单选项

set(值 1，值 2，值 3)　　多选项

```
mysql>  create  table  school.tea(name  char(5),gender  enum('boy','girl'),interest
set('film','book','music','football'));
mysql> insert into school.tea values('tom','boy','book,film');
mysql> insert into school.tea values('harry','man','nan');
ERROR 1265 (01000): Data truncated for column 'gender' at row 1
```

Day2

案例一：约束条件

1.常用约束条件：

条件约束	功能描述
null	允许为空，默认设置
not null	不允许为空
key	索引类型
default	设置默认值，缺省为 NULL

```
mysql> desc info;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | char(4)     | YES  |     | NULL    |       |
| email | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

```
mysql> create table school.restrict(name char(5) not null,gender enum("male","female") not
null default "male",age int(3) not null default 21,interest set("book","movie","eat"));
mysql> desc school.restrict;
+----------+--------------------------+------+-----+---------+-------+
| Field    | Type                     | Null | Key | Default | Extra |
+----------+--------------------------+------+-----+---------+-------+
| name     | char(5)                  | NO   |     | NULL    |       |
| gender   | enum('male','female')    | NO   |     | male    |       |
| age      | int(3)                   | NO   |     | 21      |       |
| interest | set('book','movie','eat') | YES  |     | NULL    |       |
+----------+--------------------------+------+-----+---------+-------+
```

```
mysql> insert into school.restrict(name) values("tom");
mysql> select * from school.restrict;
+------+--------+-----+----------+
| name | gender | age | interest |
+------+--------+-----+----------+
| tom  | male   |  21 | NULL     |
+------+--------+-----+----------+
```

```
mysql> insert into school.restrict(name) values("");
mysql> select * from school.restrict;
+------+--------+-----+----------+-------+
| name | gender | age | interest | email |
+------+--------+-----+----------+-------+
| tom  | male   |  21 | NULL     | NULL  |
```

```
|      | male   |  21 | NULL     | NULL  |
+------+--------+-----+----------+-------+
```

//NULL 表示不能不写，但如果写了""也算写了。空白不算空

案例二：修改表结构

基本用法：

```
alter table 表名 执行动作;
```

执行动作	功能描述
add	添加字段
modify	修改字段类型
change	修改自定名称
drop	删除字段
rename	修改表名称

1.add 添加字段

```
alter table 表名 add 字段名称 类型(宽度) 约束条件;
mysql> alter table school.restrict add email varchar(30);
mysql> desc school.restrict;
+----------+--------------------------+------+-----+---------+-------+
| Field    | Type                     | Null | Key | Default | Extra |
+----------+--------------------------+------+-----+---------+-------+
| name     | char(5)                  | NO   |     | NULL    |       |
| gender   | enum('male','female')    | NO   |     | male    |       |
| age      | int(3)                   | NO   |     | 21      |       |
| interest | set('book','movie','eat') | YES  |     | NULL    |       |
| email    | varchar(30)              | YES  |     | NULL    |       |
+----------+--------------------------+------+-----+---------+-------+
```

//默认添加的字段在表的最后。所有已经添加的数据，email 列都是 NULL。

通过 after 字段名可以将新添加的字段放到某个字段后面，或者 first 直接放到第一列

```
mysql> alter table school.restrict add phone varchar(12) not null after name;
//添加新字段 phone，放到 name 字段的后面。
mysql> desc school.restrict;
+----------+--------------------------+------+-----+---------+-------+
| Field    | Type                     | Null | Key | Default | Extra |
+----------+--------------------------+------+-----+---------+-------+
| name     | char(5)                  | NO   |     | NULL    |       |
| phone    | varchar(12)              | NO   |     | NULL    |       |
| gender   | enum('male','female')    | NO   |     | male    |       |
| age      | int(3)                   | NO   |     | 21      |       |
| interest | set('book','movie','eat') | YES  |     | NULL    |       |
| email    | varchar(30)              | YES  |     | NULL    |       |
+----------+--------------------------+------+-----+---------+-------+
mysql> select * from school.restrict;
+------+-------+--------+-----+----------+-------+
| name | phone | gender | age | interest | email |
+------+-------+--------+-----+----------+-------+
| tom  |       | male   |  21 | NULL     | NULL  |
```

```
|      |       | male   |  21 | NULL     | NULL  |
+------+-------+--------+-----+----------+-------+
```

加这一列不能为空，但之前又没有数据，只能添加空白

```
mysql> alter table school.restrict add addr varchar(30) not null default 'hangzhou' first;
//添加新字段 addr，放到所有字段前面
```

2 modify 修改字段

```
alter table 表名 modify 字段名称 类型(宽度) 约束条件;
mysql> alter table school.restrict modify addr varchar(50) default 'shanghai' after name;
mysql> desc school.restrict;
+----------+--------------------------+------+-----+----------+-------+
| Field    | Type                     | Null | Key | Default  | Extra |
+----------+--------------------------+------+-----+----------+-------+
| name     | char(5)                  | NO   |     | NULL     |       |
| addr     | varchar(50)              | YES  |     | shanghai |       |
| phone    | varchar(12)              | NO   |     | NULL     |       |
| gender   | enum('male','female')    | NO   |     | male     |       |
| age      | int(3)                   | NO   |     | 21       |       |
| interest | set('book','movie','eat') | YES  |     | NULL     |       |
| email    | varchar(30)              | YES  |     | NULL     |       |
+----------+--------------------------+------+-----+----------+-------+
```

3. change 修改字段内容

```
alter table 表名 change 字段名 类型(宽度) 约束条件;
mysql> alter table school.restrict change name myname varchar(10);
//把数据表中的 name 字段重命名为 myname，同时修改了数据类型的长度。
```

4.drop 删除字段名称

```
alter table 表名 drop 字段名称
mysql> alter table school.restrict drop interest;
//删除数据表中的 interest 字段
```

5.rename 修改字段名称

```
mysql> alter table school.restrict rename school.rest;
```

案例三：MySQL 键值

什么是索引：就是对数据表中的若干字段进行排序的方法，类似于对一本书做目录，有了目录就可以快速定位数据的具体位置。

索引的优点：

通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性

可以加快数据的检索速度

索引的缺点

当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，会降低数据库写的速度

索引需要占额外的物理空间

键值的类型

INDEX	普通索引	
UNIQUE	唯一索引	
FULLTEXT	全文索引	一般用第三方软件来实现
PRIMARY KEY	主键	
FOREIGN KEY	外键	

1.INDEX 普通索引

使用说明:

一个表中可以有多个 INDEX 字段

字段的值允许有重复，也可以赋值 NULL

经常把做查询条件的字段设置为 INDEX 字段

INDEX 字段 KEY 标志是 MUL

```
mysql>  create  table  school.info(id  int(6)  not  null,  name  varchar(5),sex
enum('male','female'),age tinyint(3) default 1, index(id),index(name));
mysql> desc info;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| id    | int(6)               | NO   | MUL | NULL    |       |
| name  | varchar(5)           | YES  | MUL | NULL    |       |
| sex   | enum('male','female') | YES  |     | NULL    |       |
| age   | tinyint(3)           | YES  |     | 1       |       |
+-------+----------------------+------+-----+---------+-------+
```

在已有的数据表中创建和删除索引。

1.创建索引: create index 索引名称 on 数据表(字段列表)

```
mysql> create index age on school.info(age);
mysql> create index nianling on school.info(age);
```

提示：可以创建多个索引，索引与字段名称也可以不一样

2.删除索引: drop index 索引名称 on 数据表

```
mysql> drop index name on school.info;
```

查看索引信息:

```
mysql> show index from school.info\G;
*************************** 1. row ***************************
        Table: info
   Non_unique: 1
     Key_name: id          索引名
 Seq_in_index: 1
  Column_name: id          id 的索引
    Collation: A
  Cardinality: 0
     Sub_part: NULL
       Packed: NULL
         Null:
```

```
   Index_type: BTREE         索引原理是 btree
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: info
   Non_unique: 1
     Key_name: age
 Seq_in_index: 1
  Column_name: age
    Collation: A
  Cardinality: 0
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 3. row ***************************
        Table: info
   Non_unique: 1
     Key_name: nianling
 Seq_in_index: 1
  Column_name: age
    Collation: A
  Cardinality: 0
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
3 rows in set (0.00 sec)
```

2. primary key 主键索引

注意事项

一个表中只能有一个 primary key 字段

对应的字段值不允许有重复，且不允许赋 NULL 值

如果有多个字段都作为 PRIMARY KEY，称为复合主键，必须一起创建。

主键字段的 key 标志是 PRI

通常与 AUTO_INCREMENT 连用，自动增加

经常把表中能够唯一标识记录的字段设置为主键字段[记录编号字段]

```
mysql> create table school.student(stu_id char(9), name char(5), primary key(stu_id));
mysql> desc school.student;
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| stu_id | char(9) | NO   | PRI | NULL    |       |
| name   | char(5) | YES  |     | NULL    |       |
+--------+---------+------+-----+---------+-------+
```

提示：stu_id 字段自动被设置为不能为 NULL， key 下面有 Pri 标记(主键标记)

```
mysql> insert into school.student values(null,'tom');
```

```
ERROR 1048 (23000): Column 'stu_id' cannot be null
mysql> insert into school.student values('0001','jerry');
mysql> insert into school.student values('0001','tom');
ERROR 1062 (23000): Duplicate entry '0001' for key 'PRIMARY'
```

创建主键索引的另一种方法：

```
mysql> create table school.student2(id char(9) primary key, name char(5));
```

对已经存在的数据表创建主键索引

```
mysql> create table school.student3(id char(9),name char(5));
mysql> alter table school.student3 add primary key(id);
```

提示：在已经存在的表中创建主键索引，一定要确保作为主键的字段数据中没有 null，如果确定已经有 null 值，可以 delete 删除数据或者 update 更新数据为非 null。

删除数据表中的 主键索引

```
mysql> alter table school.student3 drop primary key;
```

提示：删除主键后，对应的字段数据就可以出现重复的数据了。

复合主键索引（多个字段做主键）

不做主键（可能同一个技能不同）

姓名	单位	技能
孙悟空	西游记	100
孙悟空	西游记	80

以姓名为主键（不同书中人物技能水平不同，名称相同时无法写入，因为姓名不能重复）

姓名	单位	技能
孙悟空	西游记	100
孙悟空	沉香救母	80

创建复合主键索引（姓名和单位，不能重复，但是单独的姓名和单位可以重复）

```
mysql> create table school.bool(姓名 char(20),单位 char(20),技能 int(100),primary key(姓名,单位));
mysql> insert into school.bool values('孙悟空','西游记',100);
mysql> insert into school.bool values('孙悟空','沉香救母',100);
```

提示：姓名和单位都重复会报错

自动添加属性（可以自动将数据自动加 1）

```
mysql> create table school.demo(id int(100) auto_increment primary key, name char (10));
mysql> insert into school.demo(name) values ('tom');
mysql> insert into school.demo(name) values ('jerry');
mysql> insert into school.demo values(10,'jerry');
mysql> insert into school.demo(name) values('xxx');
mysql> select * from school.demo;
+----+-------+
| id | name  |
+----+-------+
|  1 | tom   |
```

```
|  2 | jerry |
| 10 | jerry |
| 11 | xxx   |
+----+-------+
```

提示：当同时有自增长和主键时，一定要通过 alter 将自增长删除后才可以删除主键，无法直接删除主键

提示：自增长都是最后一个数据加 1

3. foreign key 外键

一个作者信息表

一个图书信息表

要求图书表中的图书作者必须是作者表中的作者。

什么是外键?

让当前表字段的值在另一个表中某个字段值的范围内选择。

使用外键的条件:

表存储引擎必须是 innodb（默认就是）

字段类型必须一致

被参照字段必须是索引类型中的一种（primary key）

例如 强制要求图书表里的作者必须要作者表里

作者表

```
name     phone        addr
a        100000       beijing
b        188888       shanghai
c        199999       xian
```

图书表

```
book-name       name
book1           d
book2           a
book3           c
```

创建外键的语法:

```
foreign key(表 A 的字段名称)
references 表 B(字段名称)
on update casecade    #同步更新
on delete casecade    #同步删除
```

创建数据库:

```
mysql> create database press character set utf8;
mysql> create table press.author(姓名 char(10) primary key,地址 char(10));
```

往作者信息表中插入数据;

```
mysql> insert into press.author values('施耐庵','苏州'),('曹雪芹','辽宁'),('罗贯中','山西');
```

创建图书信息表并创建外键:

```
mysql> create table press.book(书名 char(20),作者 char(10),foreign key(作者) references press.author(姓名) on update cascade on delete cascade);
```

```
mysql> insert into press.book values('红楼','高鄂');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`press`.`book`, CONSTRAINT `book_ibfk_1` FOREIGN KEY (`作者`) REFERENCES `author` (`姓名`) ON DELETE CASCADE ON UPDATE CASCADE)
```

当作者不在 author 表中就报错

```
mysql> insert into press.book values('红楼梦','曹雪芹');
mysql> delete from press.author where 姓名='曹雪芹';
```

提示：当作者表中的作者被删除后，图书信息表中对应的数据也会被删除

删除外键，需要先查看外键的名称：

```
mysql> show create table press.book\G;
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE `book` (
  `书名` char(20) DEFAULT NULL,
  `作者` char(10) DEFAULT NULL,
  KEY `作者` (`作者`),
  CONSTRAINT `book_ibfk_1` FOREIGN KEY (`作者`) REFERENCES `author` (`姓名`) ON DELE
TE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

外键标识，删就是删除这个

```
mysql> alter table press.book drop foreign key book_ibfk_1;
mysql> drop table press.book;
```

在现有的数据表中创建外键：

```
mysql> create table press.book(书名 char(20),作者 char(10));
mysql> alter table press.book add foreign key(作者) references press.author(姓名) on update cascade on delete cascade;
```

Day3

案例一：数据库导入导出

1. 数据导入导出默认检索目录

```
mysql> show variables like "secure_file_priv";
+------------------+-----------------------+
| Variable_name    | Value                 |
+------------------+-----------------------+
| secure_file_priv | /var/lib/mysql-files/ |
+------------------+-----------------------+
```

提示：拷贝到默认的目录下，否则导入找不到

修改检索目录

```
[root@mysql50 ~]# vim /etc/my.cnf
[root@mysql50 ~]# secure_file_priv="/mydate"
[root@mysql50 ~]# mkdir /mydate
[root@mysql50 ~]# chown mysql:mysql /mydate
[root@mysql50 ~]# systemctl restart mysqld
```

```
mysql> show variables like "secure_file_priv";
+------------------+----------+
| Variable_name    | Value    |
+------------------+----------+
| secure_file_priv | /mydate/ |
+------------------+----------+
```

2.数据导入：把系统文件的内容存储到数据库的表里

命令格式

```
load date infile "目录名/文件名" into table 数据库名.表名 fields t
erminated by "分隔符" lines terminated by "\n";
```

把系统文件拷贝到检索目录下

```
[root@mysql50 ~]# cp /etc/passwd /mydate/
```

创建表结构

```
mysql> create database userdb;
mysql> create table userdb.user(username char(50),password char(1),uid int,git int,comment char(150),homedir char(180),shell char(50));
```

导入表数据

```
mysql> load data infile "/mydate/passwd" into table userdb.user fields terminated by ":" lines terminated by "\n";
```

修改表字段添加主键

```
mysql> alter table userdb.user add id int primary key auto_increment first;
```

查看表内容

```
mysql> select * from userdb.user;
```

```
mysql> select * from userdb.user;
+----+--------------------+----------+-------+-------+-----------------------------------------------------------+--------------------------+---------------+
| id | username           | password | uid   | git   | comment                                                   | homedir                  | shell         |
+----+--------------------+----------+-------+-------+-----------------------------------------------------------+--------------------------+---------------+
|  1 | root               | x        |     0 |     0 | root                                                      | /root                    | /bin/bash     |
|  2 | bin                | x        |     1 |     1 | bin                                                       | /bin                     | /sbin/nologin |
|  3 | daemon             | x        |     2 |     2 | daemon                                                    | /sbin                    | /sbin/nologin |
|  4 | adm                | x        |     3 |     4 | adm                                                       | /var/adm                 | /sbin/nologin |
|  5 | lp                 | x        |     4 |     7 | lp                                                        | /var/spool/lpd           | /sbin/nologin |
|  6 | sync               | x        |     5 |     0 | sync                                                      | /sbin                    | /bin/sync     |
|  7 | shutdown           | x        |     6 |     0 | shutdown                                                  | /sbin                    | /sbin/shutdown |
|  8 | halt               | x        |     7 |     0 | halt                                                      | /sbin                    | /sbin/halt    |
|  9 | mail               | x        |     8 |    12 | mail                                                      | /var/spool/mail          | /sbin/nologin |
| 10 | operator           | x        |    11 |     0 | operator                                                  | /root                    | /sbin/nologin |
| 11 | games              | x        |    12 |   100 | games                                                     | /usr/games               | /sbin/nologin |
| 12 | ftp                | x        |    14 |    50 | FTP User                                                  | /var/ftp                 | /sbin/nologin |
| 13 | nobody             | x        |    99 |    99 | Nobody                                                    | /                        | /sbin/nologin |
| 14 | systemd-network    | x        |   192 |   192 | systemd Network Management                                | /                        | /sbin/nologin |
| 15 | dbus               | x        |    81 |    81 | System message bus                                        | /                        | /sbin/nologin |
| 16 | polkitd            | x        |   999 |   998 | User for polkitd                                          | /                        | /sbin/nologin |
| 17 | libstoragemgmt     | x        |   998 |   996 | daemon account for libstoragemgmt                         | /var/run/lsm             | /sbin/nologin |
| 18 | rpc                | x        |    32 |    32 | Rpcbind Daemon                                            | /var/lib/rpcbind         | /sbin/nologin |
| 19 | colord             | x        |   997 |   995 | User for colord                                           | /var/lib/colord          | /sbin/nologin |
| 20 | saslauth           | x        |   996 |    76 | Saslauthd user                                            | /run/saslauthd           | /sbin/nologin |
| 21 | abrt               | x        |   173 |   173 |                                                           | /etc/abrt                | /sbin/nologin |
| 22 | rtkit              | x        |   172 |   172 | RealtimeKit                                               | /proc                    | /sbin/nologin |
| 23 | radvd              | x        |    75 |    75 | radvd user                                                | /                        | /sbin/nologin |
| 24 | chrony             | x        |   995 |   993 |                                                           | /var/lib/chrony          | /sbin/nologin |
| 25 | tss                | x        |    59 |    59 | Account used by the trousers package to sandbox the tcsd daemon | /dev/null                | /sbin/nologin |
| 26 | usbmuxd            | x        |   113 |   113 | usbmuxd user                                              | /                        | /sbin/nologin |
| 27 | geoclue            | x        |   994 |   991 | User for geoclue                                          | /var/lib/geoclue         | /sbin/nologin |
| 28 | qemu               | x        |   107 |   107 | qemu user                                                 | /                        | /sbin/nologin |
| 29 | rpcuser            | x        |    29 |    29 | RPC Service User                                          | /var/lib/nfs             | /sbin/nologin |
| 30 | nfsnobody          | x        | 65534 | 65534 | Anonymous NFS User                                        | /var/lib/nfs             | /sbin/nologin |
| 31 | setroubleshoot     | x        |   993 |   990 |                                                           | /var/lib/setroubleshoot  | /sbin/nologin |
| 32 | pulse              | x        |   171 |   171 | PulseAudio System Daemon                                  | /var/run/pulse           | /sbin/nologin |
| 33 | gdm                | x        |    42 |    42 |                                                           | /var/lib/gdm             | /sbin/nologin |
| 34 | gnome-initial-setup | x        |   992 |   987 |                                                           | /run/gnome-initial-setup/ | /sbin/nologin |
| 35 | sshd               | x        |    74 |    74 | Privilege-separated SSH                                   | /var/empty/sshd          | /sbin/nologin |
| 36 | avahi              | x        |    70 |    70 | Avahi mDNS/DNS-SD Stack                                   | /var/run/avahi-daemon    | /sbin/nologin |
| 37 | postfix            | x        |    89 |    89 |                                                           | /var/spool/postfix       | /sbin/nologin |
| 38 | ntp                | x        |    38 |    38 |                                                           | /etc/ntp                 | /sbin/nologin |
| 39 | tcpdump            | x        |    72 |    72 |                                                           | /                        | /sbin/nologin |
| 40 | lisi               | x        |  1000 |  1000 | lisi                                                      | /home/lisi               | /bin/bash     |
| 41 | apache             | x        |    48 |    48 | Apache                                                    | /usr/share/httpd         | /sbin/nologin |
| 42 | mysql              | x        |    27 |    27 | MySQL Server                                              | /var/lib/mysql           | /bin/false    |
+----+--------------------+----------+-------+-------+-----------------------------------------------------------+--------------------------+---------------+
42 rows in set (0.00 sec)
```

3.数据导出

命令格式： SQL 查询 into outfile "目录名/文件名" fields terminated by "分隔符" lines terminated by "\n";

提示：导出内容由 SQL 查询语句决定，导出的是表中的记录，不包括字段名

导出数据

```
mysql> select * from userdb.user into outfile "/mydate/a1.txt";
```

提示：默认以 tab 距离作为列的间隔符

```
mysql> select username,shell from userdb.user into outfile "/mydate/a2.txt";
mysql> select username,shell from userdb.user where id<=3 into outfile "/mydate/a3.txt" fields terminated by "#" lines terminated by "???";
```

案例二：管理表记录(复习)

1.插入表记录 insert into

向表中插入 1 条记录给所有字段赋值

```
insert into 库名.表名 values(字段值列表);
mysql> insert into userdb.user values (43, "bob", "x", 2000, 2000, "test user", "/home/bob", "/bin/bash");
```

向表中插入多条记录给所有字段赋值

```
insert into 库名.表名 values values (字段值列表),(字段值列表),(字段值列表);
mysql> insert into userdb.user values (44, "tom", "x", 2001, 2001, "test user", "/home/tom", "/bin/bash"),(45, "lucy", "x", 2002, 2002, "test user", "/home/lucy", "/bin/bash"), (46, "lili", "x", 2003, 2003, "test user", "/home/lili", "/bin/bash");
```

向表中插入 1 条记录给个别字段赋值

```
insert into 库名.表名(字段名列表) values(值列表);
mysql> insert into userdb.user(username,shell) values("jack","/sbin/nologin");
```

向表中插入多条记录给个别字段赋值

```
mysql> insert into userdb.user(username,shell) values("jack", "/sbin/nologin"),("jack", "/sbin/nologin"),("jack", "/sbin/nologin");
```

提示：字段值要与字段类型相匹配；对于字段类型的字段，要用双或单引号括起来；椅子给所有字段赋值时，字段名可以省略；只给一部分字段值时，必须明确写出对应的字段名称。

2.查询表记录

命令格式：select 字段名列表 from 库名.表名 [where 条件];

```
mysql> select * from userdb.user;
mysql> select username,homedir from userdb.user where id <=2;
```

提示：条件是控制行数，字段是控制列数

3.更新表记录

命令格式：update 库名.表名 set 字段名=值,字段名=值 [where 条件];

```
mysql> update userdb.user set password="a";
mysql> update userdb.user set password="x" where username="root";
mysql> update userdb.user set password="aa" where username="root";
ERROR 1406 (22001): Data too long for column 'password' at row 1
```

提示：修改字段值必须满足字段类型

4.删除表记录

命令格式：delete from 库名.表名 [where 条件];

```
mysql> delete from userdb.user where username="bob";
```

案例三：匹配条件

1.基本匹配条件

数值比较

字段类型必须数据数值类型

类型	用途
=	等于
>、>=	大于、大于等于
<、<=	小于、小于等于
!=	不等于

```
mysql> select * from userdb.user where id=10;
+----+----------+----------+------+------+----------+---------+---------------+
| id | username | password | uid  | git  | comment  | homedir | shell         |
+----+----------+----------+------+------+----------+---------+---------------+
| 10 | operator | x        |   11 |    0 | operator | /root   | /sbin/nologin |
+----+----------+----------+------+------+----------+---------+---------------+
mysql> select username,uid from userdb.user where uid<3;
+----------+------+
| username | uid  |
+----------+------+
| root     |    0 |
| bin      |    1 |
| daemon   |    2 |
+----------+------+
mysql> select username,uid,gid from user where uid=gid;
mysql> select username,uid from user where id=10;
```

字符比较、匹配空、非空

字符比较时，字段类型必须字符类型

类型	用途
=	相等
!=	不相等
IS NULL	匹配空
IS NOT NULL	非空

```
mysql> select username,shell from userdb.user where shell!="/bin/bash";
mysql> select username from userdb.user where username="root";

mysql> select username from userdb.user where gid is null;
mysql> select username,gid from userdb.user where gid is not null;
```

逻辑匹配（多个判断条件）

类型	用途	
OR	逻辑或	多个判断条件某一个成立即可
AND	逻辑与	多个判断条件必须同时成立
!	逻辑非	取反
()	提高优先级	

```
mysql> select * from userdb.user where username="bin" and uid=3 and shell="/bin/bash";
Empty set (0.00 sec)
mysql> select * from userdb.user where username="bin" or uid=3 or shell="/bin/bash";
 +----+----------+----------+------+------+-----------+------------+---------------+
| id | username | password | uid  | git  | comment   | homedir    | shell         |
+----+----------+----------+------+------+-----------+------------+---------------+
|  1 | root     | x        |    0 |    0 | root      | /root      | /bin/bash     |
|  2 | bin      | x        |    1 |    1 | bin       | /bin       | /sbin/nologin |
|  4 | adm      | x        |    3 |    4 | adm       | /var/adm   | /sbin/nologin |
| 40 | lisi     | x        | 1000 | 1000 | lisi      | /home/lisi | /bin/bash     |
| 44 | tom      | x        | 2001 | 2001 | test user | /home/tom  | /bin/bash     |
| 45 | lucy     | x        | 2002 | 2002 | test user | /home/lucy | /bin/bash     |
| 46 | lili     | x        | 2003 | 2003 | test user | /home/lili | /bin/bash     |
+----+----------+----------+------+------+-----------+------------+---------------+
```

提高执行的优先级

```
mysql> select username,uid,gid,(uid+gid)/2 hzxx from userdb.user where username="bin
";+----------+------+------+--------+
| username | uid  | gid  | hzxx   |
+----------+------+------+--------+
| bin      |    1 |    2 | 1.5000 |
+----------+------+------+--------+
```

范围内匹配/去重显示

匹配范围内的任意一个值即可

类型	用途
in(值列表)	在...里...
not in (值列表)	不在.. 里...
between 数字 1 and 数字 2	在...之间...
distinct 字段名	去重显示

```
mysql> select username from userdb.user where username in ("adm","sync","root","mysql");
+----------+
| username |
+----------+
| root     |
| adm      |
| sync     |
| mysql    |
+----------+
mysql>  select  username,shell  from  userdb.user  where  shell  not  in  ("/bin/bash",
"/sbin/nologin");
+----------+----------------+
| username | shell          |
+----------+----------------+
```

```
| sync     | /bin/sync      |
| shutdown | /sbin/shutdown |
| halt     | /sbin/halt     |
| mysql    | /bin/false     |
+----------+----------------+
mysql> select id,username from userdb.user where id  between 5 and 10;
+----+----------+
| id | username |
+----+----------+
|  5 | lp       |
|  6 | sync     |
|  7 | shutdown |
|  8 | halt     |
|  9 | mail     |
| 10 | operator |
+----+----------+
```

去重显示

```
mysql> select distinct shell from userdb.user;
+----------------+
| shell          |
+----------------+
| /bin/bash      |
| /sbin/nologin  |
| /bin/sync      |
| /sbin/shutdown |
| /sbin/halt     |
| /bin/false     |
+----------------+
```

2.高级匹配条件

模糊查询 like

命令格式：where 字段名 like '通配符'

_匹配单个字符、%匹配 0~N 个字符

匹配四个字符的 username

```
mysql> select username from userdb.user where username like'____';
+----------+
| username |
+----------+
| root     |
| sync     |
| halt     |
| mail     |
| dbus     |
| abrt     |
| qemu     |
| sshd     |
| lisi     |
| lucy     |
| lili     |
| jack     |
+----------+
```

匹配含有 a 字符的 username

```
mysql> select username from userdb.user where username like '%a%';
+---------------------+
| username            |
+---------------------+
| daemon              |
| adm                 |
| halt                |
| mail                |
| operator            |
| games               |
| libstoragemgmt      |
| saslauth            |
| abrt                |
| radvd               |
| gnome-initial-setup |
| avahi               |
| apache              |
| jack                |
+---------------------+
```

匹配至少 4 个字符

```
mysql> select username from userdb.user where username like '__%__';
```

正则表达式

where 字段名 regexp '正则表达式'

正则元字符 ^ $. [] * |

```
mysql> select username,uid from userdb.user where uid regexp '^...$';
mysql> select username from userdb.user where  username regexp 't$'
```

用户名有数字

```
mysql> insert into userdb.user(username) values("2yaya"),("ya5ya"),("yay8a"),("yaya7");
mysql> select username from userdb.user where username regexp '[0-9]';
+----------+
| username |
+----------+
| 2yaya    |
| ya5ya    |
| yay8a    |
| yaya7    |
+----------+
mysql> select username from userdb.user where username regexp '^r.*t$';
+----------+
| username |
+----------+
| root     |
| rtkit    |
+----------+
```

四则运算

计算符号 + - * / %

```
mysql> update userdb.user set git=git+1 where id <=5;

mysql> alter table userdb.user add age tinyint default 29 after username;
mysql> select username,2019-age csnf from userdb.user where username="root";
```

```
+----------+------+
| username | csnf |
+----------+------+
| root     | 1990 |
+----------+------+
```

3.操作查询结果

聚集函数(mysql 服务软件自带的对数据做统计的命令)

```
   avg(字段名)                    //统计字段平均值
   sum(字段名)                    //统计字段之后
   min(字段名)                    //统计字段最小值
   max(字段名)                    //统计字段最大值
   count(字段名)                  //统计字段值个数
mysql> select avg(uid) from userdb.user;
+-----------+
| avg(uid)  |
+-----------+
| 1829.7778 |
+-----------+
mysql> select avg(uid) from userdb.user where id <= 10;
+----------+
| avg(uid) |
+----------+
|   4.7000 |
+----------+
mysql> select sum(uid) from user;
mysql> select min(uid) from user where shell!="/bin/bash";

mysql> select count(*) from user;
mysql> select count(username) from user where shell!="/bin/bash";
mysql> select count(username) from user;
```

查询结果排序 order by

order by 字段名 [asc|desc] 默认是 asc 升序

```
mysql> select username,uid from userdb.user where uid between 10 and 1000 order by uid desc;
```

查询结果分组 group by　　相同的分组

```
mysql> select shell from userdb.user group by shell;
+----------------+
| shell          |
+----------------+
| NULL           |
| /bin/bash      |
| /bin/false     |
| /bin/sync      |
| /sbin/halt     |
| /sbin/nologin  |
| /sbin/shutdown |
+----------------+

mysql> select shell from userdb.user where uid > 1000;
+---------------+
| shell         |
```

```
+---------------+
| /sbin/nologin |
| /bin/bash     |
| /bin/bash     |
| /bin/bash     |
+---------------+
mysql> select shell from userdb.user where uid > 1000 group by shell;
+---------------+
| shell         |
+---------------+
| /bin/bash     |
| /sbin/nologin |
+---------------+
```

查询结果过滤

基本用法

SQL 查询 having 条件表达式;

SQL 查询 where 条件 having 条件表达式;

SQL group by 字段名 having 条件表达式;

```
mysql> select username from userdb.user where shell!="/bin/bash" having username="gdm";
+----------+
| username |
+----------+
| gdm      |
+----------+
mysql> select username from userdb.user where uid<=1000 and username="gdm";
```

提示：这两条最大的区别就是上面是先放入内存再查找，并且也不用全表查找，下面的 SQL 语句是匹配一条破判断一条,及 having 速度更快。

```
mysql> select shell from userdb.user where shell!="/bin/bash" group by shell having shell="/bin/sync";
+-----------+
| shell     |
+-----------+
| /bin/sync |
+-----------+
```

限制查询结果显示行数

基本用法

SQL 查询 limit N;　　//显示查询结果前 N 条记录

SQL 查询 limit N,M;　　//显示指定范围的查询记录

SQL 查询 where 条件查询 limit 3;　　//显示插叙结果前 3 条记录

SQL 查询 where 条件查询 limit 3,3　　//从第 4 条开始，共显示 3 条

注意：0 代表第一行

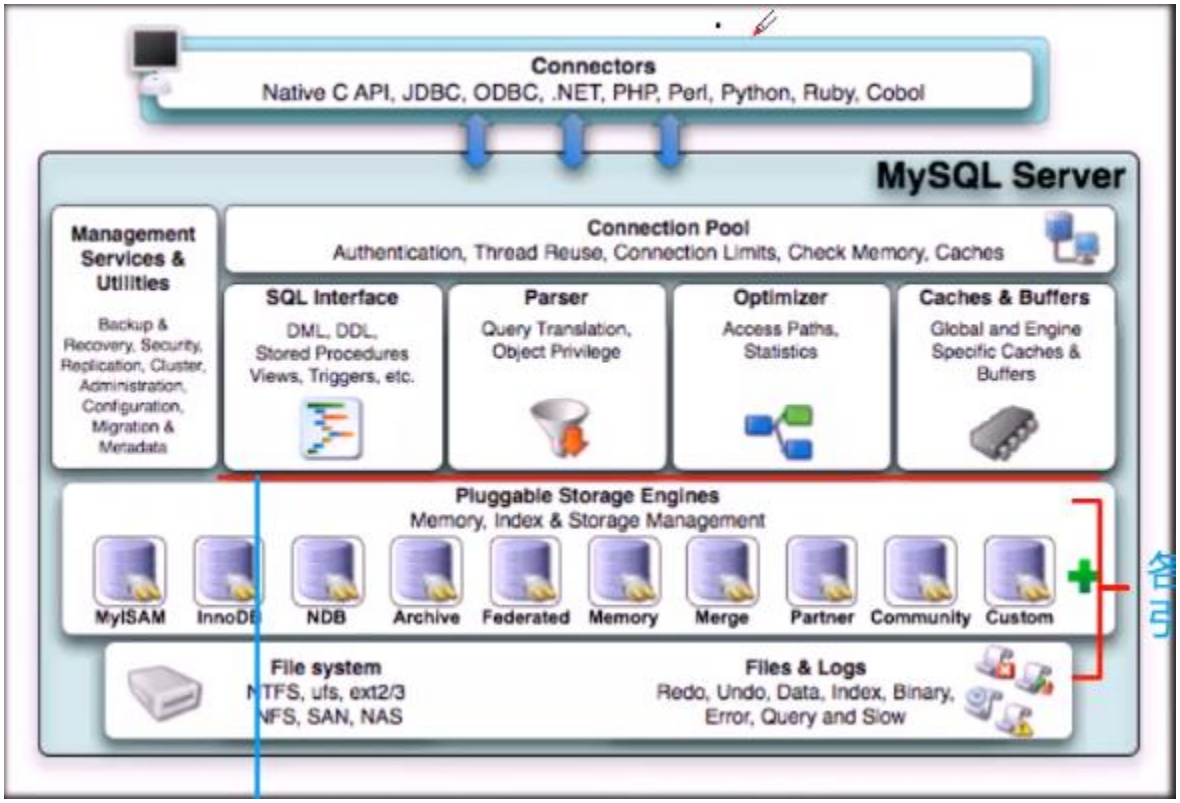
```
mysql> mysql> select * from userdb.user where uid>=10 and uid<=500 limit 3;
+----+----------+------+----------+------+------+----------+------------+---------------+
| id | username | age  | password | uid  | gid  | comment  | homedir    | shell         |
+----+----------+------+----------+------+------+----------+------------+---------------+
| 10 | operator |   29 | x        |   11 |    0 | operator | /root      | /sbin/nologin |
| 11 | games    |   29 | x        |   12 |  100 | games    | /usr/games | /sbin/nologin |
| 12 | ftp      |   29 | x        |   14 |   50 | FTP User | /var/ftp   | /sbin/nologin |
```

```
+----+----------+------+----------+------+------+----------+-----------+---------------+
mysql> select id,username,homedir from userdb.user where uid >=10 and uid<=500 limit 3,3;
+----+----------------+---------+
| id | username       | homedir |
+----+----------------+---------+
| 13 | nobody         | /       |
| 14 | systemd-network | /       |
| 15 | dbus           | /       |
+----+----------------+---------+
mysql> select username,uid,homedir from userdb.user where uid>1000 order by uid desc
 limit 3;+-----------+-------+--------------+
| username  | uid   | homedir      |
+-----------+-------+--------------+
| nfsnobody | 65534 | /var/lib/nfs |
| lili      |  2003 | /home/lili   |
| lucy      |  2002 | /home/lucy   |
+-----------+-------+--------------+
mysql> select username,uid,homedir from userdb.user where uid >1000 order by uid des
c limit 1;+-----------+-------+--------------+
| username  | uid   | homedir      |
+-----------+-------+--------------+
| nfsnobody | 65534 | /var/lib/nfs |
+-----------+-------+--------------+
```

案例四：MySQL 存储引擎

1.MySQL 存储引擎介绍

mysql 体系结构介绍



管理单元(Management Services & Utillities)：装上 mysql 之后，软件提供的命令

连接池(Connection Pool):当有客户端访问时，会检查本机是否有空闲的进程处理客户端连接，检查连接的用户对不对，检查有没有硬件资源分配。

SQL 组件：增删改查命令传递给 mysql 进程。

分析器：检查语法是否正确

优化器(Optimizer)：执行的命令做优化处理

查询缓存(Cache & Buffers)：物理内存划分，存储查询的结果（8MB 从物理内存划分出来的）

存储引擎：MySQL 服务软件自带的功能程序，处理表的处理器

文件系统：库表的存储（硬盘）

存储引擎介绍

作为可插拔的组件提供

MySQL 服务软件自带的功能程序，处理表的处理器

不同的存储引擎有不同的功能和数据存储方式

默认的存储引擎

MySQL 5.0/5.1 MyISAM

MySQL 5.5/5.6 InnoDB

2.查看存储引擎

查看已有表的使用存储引擎

```
mysql> show create table userdb.user\G;
*************************** 1. row ***************************
       Table: user
Create Table: CREATE TABLE `user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` char(50) DEFAULT NULL,
  `age` tinyint(4) DEFAULT '29',
  `password` char(1) DEFAULT NULL,
  `uid` int(11) DEFAULT NULL,
  `gid` int(11) DEFAULT NULL,
  `comment` char(150) DEFAULT NULL,
  `homedir` char(180) DEFAULT NULL,
  `shell` char(50) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=52 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

```
mysql> show engines;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
```

default 表示当前表默认的存储引擎

3.修改存储引擎

修改数据库服务默认使用的存储引擎

```
systemctl stop mysqld
vim /etc/my.cnf
default-storage-engine=myisam
systemctl start mysqld
mysql> create table userdb.t1(id int);
mysql> show create userdb.table t1\G;
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `id` int(11) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

```
mysql> mysql> show engines;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| InnoDB             | YES     | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| MyISAM             | DEFAULT | MyISAM storage engine                                          | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
```

修改表使用的存储引擎

```
mysql> alter table t3 engine=myisam;
[root@mysql50 userdb]# cd /var/lib/mysql/userdb
[root@mysql50 userdb]# ls
db.opt  t1.frm  t1.MYD  t1.MYI  user.frm  user.ibd
```

标黄：MyISAM 存储引擎

标蓝：InnoDB 存储引擎

建表时指定表使用的存储引擎

```
mysql> create table t2(name char(10)) engine=innodb;
```

4.常用存储引擎的特点

MyISAM 存储引擎特点

支持表级锁；不支持外键、事务、事务回滚；表文件对应个数 3.

t3.frm:存储表结构

t3.MYD:存储数据

t3.MYI：index 信息

InnoDB 存储引擎

支持外键；支持行级锁；支持事物和事务回滚；表文件对应个数 2

user.frm:存储表结构

user.ibd:存储表数据+index 信息

术语解释

MySQL 锁机制

锁粒度

表级锁：一次直接对整张表进行加锁

行级锁：只锁定某一行

页级锁：对整个页面(MySQL 管理数据的基本存储单位)进行加锁。

内存有 8M，一页就是 1M

锁类型

读锁(共享锁)：支持并发读

写锁(互斥锁、排他锁)：是独占锁，上锁期间其他线程不能读表或者写表

查看当前的锁状态

检查 Table_lock 开头的变量，%通配符

```
mysql> show status like "Table_lock%";
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Table_locks_immediate | 101   |
| Table_locks_waited    | 0     |
+-----------------------+-------+
```

Table_locks_waited:等待写锁释放的个数

事物：对数据访问开始到访问操作断开连接的过程

事物回滚：在访问过程中，任意一步操作失败，恢复之前的所有操作

事务日志文件：记录所有 innodb 存储引擎表的操作

ibdata1:没有提交的命令（没有正确完成的）

ib_logfile0：已经提交的命令（正确完成的）

ib_ligfile1：

事务特性：

Atomic(原子性)：事务的整个操作是一个整体，不可分割，要么全部成功，要么全部失败。

Consistecy(一致性)：事务操作的前后，表中的记录没有变化

Isolation(隔离性)：事务操作是相互隔离不受影响

Durability(持久性)：数据一旦提交，不可改变，永久改变表数据

为了看到效果，需要关闭自动提交功能

开两个不同的终端连接数据库

第一个终端，第二个终端

实验隔离性和一致性

```
mysql> show variables like "autocommit";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | ON    |
+---------------+-------+
mysql> update userdb.user set password="F";
```

一提交就执行结束，叫做自动提交

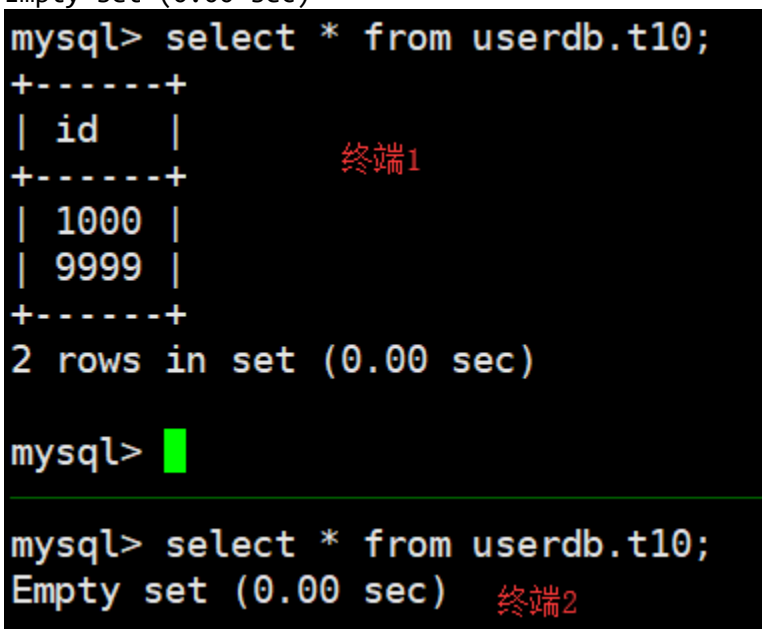
```
mysql> set autocommit=off;
```

提示：第一个终端关闭自动提交不影响第二个终端连接

```
mysql> create table userdb.t10(id int) engine=innodb;      //新建 innodb 引擎库
mysql> insert into userdb.t10 values(9999),(1000);
mysql> select * from userdb.t10;
+------+
| id   |
+------+
| 1000 |
| 9999 |
+------+
```

在另一个终端查看

```
mysql> select * from userdb.t10;
Empty set (0.00 sec)
```




```
mysql> insert into userdb.t10 values(888);
mysql> select * from userdb.t10;
+------+
| id   |
+------+
|  888 |
+------+
mysql> select * from userdb.t10;
+------+
| id   |
+------+
| 1000 |
| 9999 |
+------+
mysql> commit;
mysql> select * from userdb.t10;
+------+
| id   |
+------+
| 1000 |
```

```
| 9999 |
|  888 |
+------+
```

验证事务回滚

```
mysql> delete from userdb.t10;
mysql> select * from userdb.t10;
Empty set (0.00 sec)
mysql> select * from userdb.t10;
+------+
| id   |
+------+
| 1000 |
| 9999 |
|  888 |
+------+
mysql> rollback;
mysql> select * from userdb.t10;
+------+
| id   |
+------+
| 1000 |
| 9999 |
|  888 |
+------+
```

5.工作中，建表时，如何决定表使用哪种存储引擎

select(读)操作多的表适合使用 myisam 存储引擎，节约系统资源

insert(写)操作多的表适合使用 innodb 存储引擎，并发访问量大

原因：myisam 是表锁，例如

```
update user set gid=900 where id<=10;
update user set gid=100 where id>10;
```

只有上面语句结束才会执行下面的语句。

innodb 是行锁，例如

```
select * from user where id<=10;
select * from user where id>10;
```

那就需要给每个匹配的条件加读锁，因此用表级锁更节省资源

Day4

案例一：多表查询

0.复制表

命令格式：create table 库名.表名 SQL 查询;

备份表

```
mysql> create database db1;
mysql> create table db1.user2 select * from userdb.user;
```

快速建表（只要表结构）

```
mysql> create table db1.user3 select * from userdb.user where False;
mysql> select * from db1.user3;
Empty set (0.00 sec)
```

```
mysql> desc db1.user3;
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| id       | int(11)    | NO   |     | 0       |       |
| username | char(50)   | YES  |     | NULL    |       |
| age      | tinyint(4) | YES  |     | 29      |       |
| password | char(1)    | YES  |     | NULL    |       |
| uid      | int(11)    | YES  |     | NULL    |       |
| gid      | int(11)    | YES  |     | NULL    |       |
| comment  | char(150)  | YES  |     | NULL    |       |
| homedir  | char(180)  | YES  |     | NULL    |       |
| shell    | char(50)   | YES  |     | NULL    |       |
+----------+------------+------+-----+---------+-------+
```

提示：复制表时不会复制键值属性

1.多表查询介绍

多表查询，也称为连接查询

将 2 个或 2 个以上的表按某个条件连接起来，从中选取需要的数据

当多个表中存在相同意义的字段(字段名可以不同)时，可以通过该字段连接多个

2.格式

`select 字段名列表 from 表 a,表 b;`

`select 字段名列表 from 表名 a,表 b [where 条件];`

提示：以上格式的查询结果叫笛卡尔集，查询结果的总条目数是=(表 a 的记录数 x 表 b 的记录数)

```
mysql> use db1;
mysql> create table t1 select username,uid,password from userdb.user limit 3;
mysql> select * from t1;
+----------+------+----------+
| username | uid  | password |
+----------+------+----------+
| root     |    0 | F        |
| bin      |    1 | F        |
| daemon   |    2 | F        |
+----------+------+----------+
mysql> create table t2 select username,uid,shell,homedir from userdb.user limit 5;
mysql> select * from t2;
+----------+------+---------------+----------------+
| username | uid  | shell         | homedir        |
+----------+------+---------------+----------------+
| root     |    0 | /bin/bash     | /root          |
| bin      |    1 | /sbin/nologin | /bin           |
| daemon   |    2 | /sbin/nologin | /sbin          |
| adm      |    3 | /sbin/nologin | /var/adm       |
| lp       |    4 | /sbin/nologin | /var/spool/lpd |
+----------+------+---------------+----------------+
mysql> select * from t1,t2;
+----------+------+----------+----------+------+---------------+----------------+
| username | uid  | password | username | uid  | shell         | homedir        |
+----------+------+----------+----------+------+---------------+----------------+
| root     |    0 | F        | root     |    0 | /bin/bash     | /root          |
```

```
| bin      |    1 | F         | root     |    0 | /bin/bash     | /root          |
| daemon   |    2 | F         | root     |    0 | /bin/bash     | /root          |
| root     |    0 | F         | bin      |    1 | /sbin/nologin | /bin           |
| bin      |    1 | F         | bin      |    1 | /sbin/nologin | /bin           |
| daemon   |    2 | F         | bin      |    1 | /sbin/nologin | /bin           |
| root     |    0 | F         | daemon   |    2 | /sbin/nologin | /sbin          |
| bin      |    1 | F         | daemon   |    2 | /sbin/nologin | /sbin          |
| daemon   |    2 | F         | daemon   |    2 | /sbin/nologin | /sbin          |
| root     |    0 | F         | adm      |    3 | /sbin/nologin | /var/adm       |
| bin      |    1 | F         | adm      |    3 | /sbin/nologin | /var/adm       |
| daemon   |    2 | F         | adm      |    3 | /sbin/nologin | /var/adm       |
| root     |    0 | F         | lp       |    4 | /sbin/nologin | /var/spool/lpd |
| bin      |    1 | F         | lp       |    4 | /sbin/nologin | /var/spool/lpd |
| daemon   |    2 | F         | lp       |    4 | /sbin/nologin | /var/spool/lpd |
+----------+------+-----------+----------+------+---------------+----------------+

mysql> select * from t1,t2 where t1.uid=t2.uid;
+----------+------+----------+----------+------+---------------+---------+
| username | uid  | password | username | uid  | shell         | homedir |
+----------+------+----------+----------+------+---------------+---------+
| root     |    0 | F        | root     |    0 | /bin/bash     | /root   |
| bin      |    1 | F        | bin      |    1 | /sbin/nologin | /bin    |
| daemon   |    2 | F        | daemon   |    2 | /sbin/nologin | /sbin   |
+----------+------+----------+----------+------+---------------+---------+
mysql> select * from t1,t2 where t1.uid=1 and t2.uid=1;
+----------+------+----------+----------+------+---------------+---------+
| username | uid  | password | username | uid  | shell         | homedir |
+----------+------+----------+----------+------+---------------+---------+
| bin      |    1 | F        | bin      |    1 | /sbin/nologin | /bin    |
+----------+------+----------+----------+------+---------------+---------+
mysql> select t1.*,t2.shell,t2.homedir from t1,t2 where t1.uid=1 and t2.uid=1;
+----------+------+----------+---------------+---------+
| username | uid  | password | shell         | homedir |
+----------+------+----------+---------------+---------+
| bin      |    1 | F        | /sbin/nologin | /bin    |
+----------+------+----------+---------------+---------+
```

where 子查询：把内层查询结果作为外层查询的查询条件

语法格式：select 字段名列表 from 表名 where 条件 (select 字段名列表 from 表名 where 条件);

准备：

```
   mysql> update userdb.user set age=19 where id<=10;
   mysql> update userdb.user set age=36 where id>10 and id<=40;
   mysql> select avg(age) from userdb.user;
   +----------+
   | avg(age) |
   +----------+
   |  31.2000 |
   +----------+
mysql> select username,age from userdb.user where age < avg(age);
ERROR 1111 (HY000): Invalid use of group function
```

语法错误

```
mysql> select username,age from userdb.user where age < (select avg(age) from userdb.user);
+----------+------+
| username | age  |
```

```
+----------+------+
| root     |   19 |
| bin      |   19 |
| daemon   |   19 |
| adm      |   19 |
| lp       |   19 |
| sync     |   19 |
| shutdown |   19 |
| halt     |   19 |
| mail     |   19 |
| operator |   19 |
| apache   |   29 |
| mysql    |   29 |
| tom      |   29 |
| lucy     |   29 |
| lili     |   29 |
| jack     |   29 |
| 2yaya    |   29 |
| ya5ya    |   29 |
| yay8a    |   29 |
| yaya7    |   29 |
+----------+------+
mysql> select username,shell from userdb.user where username in (select username from db1.t1);
+----------+---------------+
| username | shell         |
+----------+---------------+
| root     | /bin/bash     |
| bin      | /sbin/nologin |
| daemon   | /sbin/nologin |
+----------+---------------+
```

提示：除了同表的查询结果作为条件之外，不同表的查询结果也能作为条件

例如：select name from 班级表 where name in (select name from 缴费表 where 班级="nsd1812");

这就是表示查找了交钱的。

左链接：查询条件成立时，以左边的表为主显示查询结果

基本用法：select 字段名列表 from 表a left join 表b on 条件表达式;

```
mysql> create table t3 select username,uid,homedir,password from userdb.user limit 3;
mysql> create table t4 select username,uid,homedir,password from userdb.user limit 6;
```

t3,和 t4 有相同也有不同的，如果要显示相同的记录，用链接查询更合适

```
mysql> select * from t3 left join t4 on t3.uid=t4.uid;
+----------+------+---------+----------+----------+------+---------+----------+
| username | uid  | homedir | password | username | uid  | homedir | password |
+----------+------+---------+----------+----------+------+---------+----------+
| root     |    0 | /root   | F        | root     |    0 | /root   | F        |
| bin      |    1 | /bin    | F        | bin      |    1 | /bin    | F        |
| daemon   |    2 | /sbin   | F        | daemon   |    2 | /sbin   | F        |
+----------+------+---------+----------+----------+------+---------+----------+
```

右链接：查询条件成立时，以右边的表为主显示查询结果

基本用法：select 字段名列表 from 表a right join 表b on 条件表达式;

查看有相同的也有不相同的：

```
mysql> mysql> select * from t3 right join t4 on t3.uid=t4.uid;
+----------+------+---------+----------+----------+------+----------------+----------+
| username | uid  | homedir | password | username | uid  | homedir        | password |
+----------+------+---------+----------+----------+------+----------------+----------+
| root     |    0 | /root   | F        | root     |    0 | /root          | F        |
| bin      |    1 | /bin    | F        | bin      |    1 | /bin           | F        |
| daemon   |    2 | /sbin   | F        | daemon   |    2 | /sbin          | F        |
| NULL     | NULL | NULL    | NULL     | adm      |    3 | /var/adm       | F        |
| NULL     | NULL | NULL    | NULL     | lp       |    4 | /var/spool/lpd | F        |
| NULL     | NULL | NULL    | NULL     | sync     |    5 | /sbin          | F        |
+----------+------+---------+----------+----------+------+----------------+----------+
```

案例二：mysql 管理工具

1.访问数据库服务器的方式有哪些?

命令连接：数据传输速度快，跨平台

软件连接：提供图形界面

常见的 MySQL 管理工具

类型	界面	操作系统	说明
mysql	命令行	跨平台	MySQL 官方 bundle 包自带
MySQL-Workbench	图形	跨平台	MySQL 官方提供
MySQL-Front	图形	windows	开源，轻量级客户端软件
phpMyAdmin	浏览器	跨平台	开源，需要 LAMP 平台
Navicat	图形	Windows	专业、功能强大，商业版

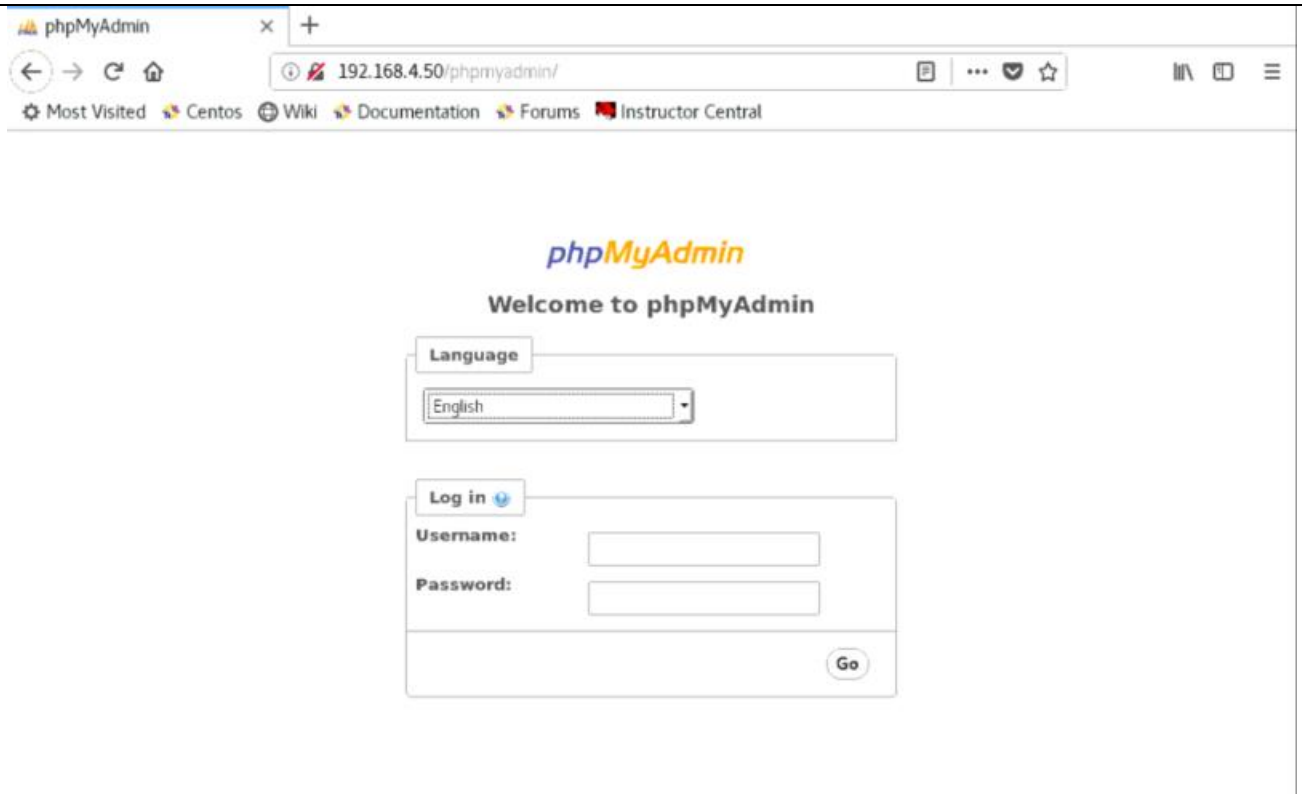
2.部署 phpMyAdmin(当前最新版本)

phpMyAdmin-4.8.5-all-languages.zip

```
[root@mysql50 ~]# yum -y install httpd php php-mysql       //最新版不可用
[root@mysql50 ~]# systemctl start httpd
[root@mysql50 ~]# mv phpMyAdmin-4.8.5-all-languages /var/www/html/phpmyadmin
[root@mysql50 phpmyadmin]# cp config.sample.inc.php config.inc.php
                          //软件的配置文件，指定管理哪一台 mysql 服务器
31 $cfg['Servers'][$i]['host'] = 'localhost'; //指定管理 mysql 的 ip
17 $cfg['blowfish_secret'] = 'hzxx';    //随便写，只要不为空
```

最新版环境需要 php-5.5 版本

```
rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
rpm -Uvh https://mirror.webtatic.com/yum/el7/webtatic-release.rpm
[root@mysql50 ~]# yum -y install httpd php55w php55w-mysql
```



Username:root
Password: mysql password → 123456
后面比较简单，就不再拓展。

案例三：用户授权与权限撤销

0.密码恢复及设置

恢复数据库管理员 root 用户本机登陆密码

```
[root@mysql50 ~]# systemctl stop mysqld
[root@mysql50 ~]# vim /etc/my.cnf
skip-grant-tables                  //跳过授权
#validate_password_policy=0
#validate_password_length=0
[root@mysql50 ~]# systemctl start mysqld
[root@mysql50 ~]# mysql
mysql> select host,user,authentication_string from mysql.user;
+-----------+-----------+-------------------------------------------+
| host      | user      | authentication_string                     |
+-----------+-----------+-------------------------------------------+
| localhost | root      | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
| localhost | mysql.sys | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
+-----------+-----------+-------------------------------------------+
mysql> select password("123456");
+-------------------------------------------+
| password("123456")                        |
+-------------------------------------------+
| *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
+-------------------------------------------+
mysql> update mysql.user set authentication_string=password("654321") where user="root";
```

```
mysql> flush privileges;    //密码配置生效
[root@mysql50 ~]# vim /etc/my.cnf
   之前加的注释去掉
#skip-grant-tables
[root@mysql50 ~]# systemctl restart mysqld
[root@mysql50 ~]# mysql -uroot -p'654321'
```

修改密码

```
mysqladmin -hlocalhost -uroot -p'654321' password "123456"
-p 原始密码            password 修改后密码
```

1.创建授权

在数据库服务器上，添加连接用户及设置访问权限

默认数据库管理员 root 用户本机登陆，才有授权权限

授权记录：授权库 mysql

user：已有的用户及访问权限

db： 记录已有的用户对数据库的访问权限

tables_priv：记录已有用户对表的访问权限

columns_priv:记录已有用户对字段的访问权限

授权命令的用法

mysql> grant 权限列表 on 数据库名 to 用户名@"客户端地址" identified by "密码" [with grant option];

提示：当库名.表名 为*.*时，匹配所有库所有表

权限列表

all：匹配所有权限

SELECT,UPDATE,INSERT ...

SELECT,IPDATE(字段 1,..，字段 N)

客户端地址

%：匹配所有主机

192.168.1.%：匹配指定的一个网段

192.168.1.1：匹配指定 IP 地址的单个主机

%.tarena.com：匹配一个 DNS 区域

svr1.tarena.com：匹配指定区域的单个主机

例子：新建用户 mydba，对所有库、表有完全权限，允许从任何地址访问，密码设为 123456，允许该用户为其他用户授权

```
mysql> grant all on *.* to mydba@'%' identified by '123456' with grant option;
[root@mysql50 ~]# mysql -umydba -p'123456';
mysql> show grants;
+-------------------------------------------------------------+
| Grants for mydba@%                                          |
+-------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'mydba'@'%' WITH GRANT OPTION |
+-------------------------------------------------------------+
```

```
mysql> grant select on userdb.user to hzxx@"localhost" identified by "123456";
```

管理员查看其他用户的权限

```
show grants for 用户名@'客户端地址';
```

结合

```
mysql> select user,host from mysql.user;
+-----------+-----------+
| user      | host      |
+-----------+-----------+
| mydba     | %         |
| hzxx      | localhost |
| mysql.sys | localhost |
| root      | localhost |
+-----------+-----------+
```

2.撤销权限

```
revoke 权限列表 on 数据库名 from 用户名@ '客户端地址';
mysql> revoke delete,insert on *.* from mydba@'%';
mysql> show grants\G;
*************************** 1. row ***************************
Grants for mydba@%: GRANT SELECT, UPDATE, CREATE, DROP, RELOAD, SHUTDOWN, PROCESS, FILE, REFERENCES, INDEX, ALTER, SHOW DATABASES, SUPER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE TABLESPACE ON *.* TO 'mydba'@'%' WITH GRANT OPTION
1 row in set (0.00 sec)
mysql> revoke all on *.* from mydba@'%';
```

提示:此时并没有撤销 grant 权限

```
mysql> show grants for  mydba@'%';
+---------------------------------------------------+
| Grants for mydba@%                                |
+---------------------------------------------------+
| GRANT USAGE ON *.* TO 'mydba'@'%' WITH GRANT OPTION |
+---------------------------------------------------+
mysql> revoke grant option on *.* from mydba@'%';
```

删除用户

```
drop user 用户名@'客户端'
mysql> drop user mydba@'%';
```

Day5 数据备份与数据恢复

相关概念:

数据备份方式:

物理备份:直接拷贝库或表对应的系统文件

逻辑备份:使用备份命令或软件对数据做备份,生成对应的备份文件

服务状态:冷备,热备,温备

数据备份策略:

完全备份:备份所有数据(1 张表、1 个库、1 台数据库服务器)

增加备份：备份上次备份后，所有新产生的数据

差异备份：备份完全备份后，所有新产生的数据

案例一：物理备份与恢复

数据物理备份和数据物理恢复

```
[root@mysql50 ~]# mkdir /mybak
[root@mysql50 ~]# cp -r /var/lib/mysql /mybak/mysql.bak
[root@mysql50 ~]# scp /mybak/mysql.tar.gz 192.168.4.51:/root

[root@mysql51 ~]# systemctl stop mysqld
[root@mysql51 ~]# rm -rf /var/lib/mysql/*
[root@mysql51 ~]# tar xf mysql.tar.gz
[root@mysql51 ~]# mv mybak/mysql.bak/* /var/lib/mysql
[root@mysql51 ~]# vim /etc/my.cnf
validate_password_policy=0
validate_password_length=6
[root@mysql51 ~]# systemctl start mysqld
```

物理备份的优点与缺点

跨平台性差

备份时间长,冗余备份,浪费存储空间

案例二： mysqldump

数据完全备份

```
[root@mysql50 ~]# rpm -qf /usr/bin/mysqldump
mysql-community-client-5.7.17-1.el7.x86_64
[root@mysql50 ~]# mysqldump --help
```

备份操作命令格式

mysqldump –uroot –p密码 库名 > 路径/xxx.sql

恢复操作命令格式

mysql –uroot –p密码 [库名] < 路径/xxx.sql

库名表示方法

--all-databases 或 –A	所有库
数据库名	单个库
数据库名 表名	单张表
-B 数据库1 数据库2	多个库

注意事项:无论备份还是恢复,都要验证用户权限

例子

50数据备份

```
[root@mysql50 ~]# mysqldump -uroot -p654321  --all-databases> /mybak/mysql-all.sql
[root@mysql50 ~]# cat /mybak/mysql-all.sql
```

实际备份的是生成数据库的SQL命令

```
[root@mysql50 ~]# mysqldump -uroot -p654321 userdb user > /mybak/mysql_user.sql
[root@mysql50 ~]# mysqldump -uroot -p654321 -B db1 userdb > /mybak/mysqltwodb.sql
```

52数据恢复

```
[root@mysql52 ~]# mysql -uroot -p123456 < mybak/mysqltwodb.sql
```

注意:当恢复时没有库时，会提示报错，需要先建立表所在的库

使用 mysqldump 做数据完全备份的缺点

目标库有同名库同名表时会覆盖，原因是

```
DROP TABLE IF EXISTS `a1`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `a1` (
  `user` char(32) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL DEFAULT ''
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

会先删除表，再恢复

另外会加入一个写锁，只有完全结束之后，才能访问

mysqldump 缺点

效率低,备份和还原速度慢

备份过程中,数据插入和更新操作会被挂起

案例二:实时增量备份/恢复

使用 MySQL 服务自带的 binlog 日志文件实现的

2.1mysql 服务 binlog 日志的管理

binlog 日志介绍

二进制日志，记录所有更改数据的操作，log_bin[=dir/name]

server_id=数字 (1-255)

max_binlog_size=数字 m

启用 binlog 日志

```
vim /etc/my.cnf
server_id=50
log_bin         //或者 log-bin 均可
[root@mysql50 ~]# systemctl restart mysqld
[root@mysql50 mysql]# ls /var/lib/mysql
mysql50-bin.000001 mysql50-bin.index
```

binlog 相关文件

主机名-bin.index 记录已有日志文件名

主机名-bin.000001 第 1 个二进制日志

主机名-bin.000002 第 2 个二进制日志

注意:000001 自动生成的文件，默认是 1G，超过 1G 生成新的日志文件，后面是日志索引文件

自定义 binlog 日志,存储目录和日志文件名

```
[root@mysql50 mysql]# vim /etc/my.cnf
log_bin=/logdir/db50              //后面的序号是自动生成的
mkdir /logdir
[root@mysql50 mysql]# chown mysql:mysql /logdir
[root@mysql50 mysql]# systemctl restart mysqld
```

```
[root@mysql50 mysql]# ls /logdir/
db50.000001  db50.index
```

管理 binlog 日志文件

手动创建新的日志文件

重启 mysql 服务：不适合生产服务，新产生的日志记录在数字最大的表

执行 SQL 操作 mysql>flush logs :执行一次生成一个

mysqldump –flush-logs：完全备份之后，产生新的二进制日志

mysql –uroot –p 密码 –e 'flush logs'

```
[root@mysql50 mysql]# mysql -uroot -p654321 -e "flush logs"
[root@mysql50 mysql]# ls /logdir/
db50.000001  db50.000002  db50.000003  db50.000004  db50.000005  db50.index
```

删除已有的日志文件

mysql> purge master logs to "db.000005"; //删除的是 db.000001-db.000004

mysql> reset master;   //删除所有日志从头开始

```
[root@mysql50 mysql]# mysql -uroot -p654321 -e "purge master logs to 'db50.000003';"
[root@mysql50 mysql]# ls /logdir/
db50.000003  db50.000004  db50.000005  db50.index
```

查看日志文件内容

查看日志文件内容命令

格式：mysqlbinlog [选项] binlog 日志文件名

常用选项

--start-datetime="yyyy-mm-dd hh:mm:ss"
--stop-datetime="yyyy-mm-dd hh:mm:ss"

--start-position=数字

--stop-position=数字

```
[root@mysql50 mysql]# mysqlbinlog /logdir/db50.000001
```

日志格式

```
mysql> show variables like "binlog_format";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
mysql> show master status;
+-------------+----------+--------------+------------------+-------------------+
| File        | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-------------+----------+--------------+------------------+-------------------+
| db50.000001 |      154 |              |                  |                   |
+-------------+----------+--------------+------------------+-------------------+
mysql> insert into db1.a1 values("tom");
mysql> show master status;
+-------------+----------+--------------+------------------+-------------------+
| File        | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-------------+----------+--------------+------------------+-------------------+
| db50.000001 |      448 |              |                  |                   |
+-------------+----------+--------------+------------------+-------------------+
```

三种记录格式：

(1)statement(报表格式):每一条修改数据的 sql 命令都会记录在 binlog 日志中.

优点：不需要记录每一条 SQL 语句与每行的数据变化，这样子 binlog 的日志也会比较少，减少了磁盘 IO，提高性能。

缺点：在某些情况下会导致 master-slave 中的数据不一致(如 sleep()函数， last_insert_id()，以及 user-defined functions(udf)等会出现问题)

(2)row(行模式):不记录 sql 语句上下文相关信息,仅保存那条记录被修改 //默认

优点：不会出现某些特定情况下的存储过程、或 function、或 trigger 的调用和触发无法被正确复制的问题。

缺点：会产生大量的日志，尤其是 alter table 的时候会让日志暴涨。

(3)mixed(混合模式):是以上两种分时的混合使用

一般的复制使用 STATEMENT 模式保存 binlog，对于 STATEMENT 模式无法复制的操作使用 ROW 模式保存 binlog，MySQL 会根据执行的 SQL 语句选择日志保存方式。

修改记录模式

```
vim /etc/my.cnf
binlog_format="mixed"
systemctl restart mysqld
[root@mysql50 mysql]# mysql -uroot -p654321 -e "show variables like 'binlog_format';"
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | MIXED |
+---------------+-------+
```

验证:

```
mysql> create database bbsdb;
mysql> create table bbsdb.user(name char(10),password char(6));
mysql> insert into bbsdb.user values("bob","abc123");
mysql> insert into bbsdb.user values("tom","123123");
mysql> insert into bbsdb.user values("jim","654321");
[root@mysql50 mysql]# mysqlbinlog /logdir/db50.000001 | grep -i insert
insert into bbsdb.user values("bob","abc123")
insert into bbsdb.user values("tom","123123")
insert into bbsdb.user values("jim","654321")
```

日志记录 sql 命令内容

时间点:记录命令执行的时间

偏移量:记录命令长度

2.2 使用 binlog 日志恢复

mysqlbinlog [选项] 日志文件 | mysql –uroot –p 密码

例子

```
[root@mysql50 mysql]# scp /logdir/db50.000001 192.168.4.52:/root/
[root@GYP-Work ~]# ssh root@mysql52
[root@mysql52 ~]# mysqlbinlog db50.000001 | mysql -uroot -p123456
mysql> select * from bbsdb.user;
+------+----------+
| name | password |
```

```
+------+----------+
| bob  | abc123   |
| tom  | 123123   |
| jim  | 654321   |
+------+----------+
```

在 50 添加 2 条新数据

```
mysql> insert into bbsdb.user values("jack","88888");
mysql> insert into bbsdb.user values("alice","778888");
```

再删除新添加的数据

```
mysql> delete from bbsdb.user where name in ("jack","alice");
[root@mysql50 ~]# mysqlbinlog /logdir/db50.000001
...
# at 1512
#190624 20:18:20 server id 50  end_log_pos 1629 CRC32 0x8dc8832b Query  thread_id=6
exec_time=0  error_code=0
SET TIMESTAMP=1561378700/*!*/;
insert into bbsdb.user values("jack","888888")
/*!*/;
# at 1629
#190624 20:18:20 server id 50  end_log_pos 1660 CRC32 0x6ede0bbe       Xid = 18
COMMIT/*!*/;
# at 1660
#190624 20:18:31 server id 50  end_log_pos 1725 CRC32 0xedc241dc       Anonymous_GTID
   last_committed=6   sequence_number=7
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
...
```

是一次事务记录，因此以偏移量恢复，应该从 1512 到 1660. 包含 commit，提交才能生效.

```
[root@mysql50     ~]#     mysqlbinlog     --start-position=1512     --stop-position=1660
/logdir/db50.000001 | mysql -uroot -p654321
[root@mysql50 ~]# mysql -uroot -p'654321' -e "select * from bbsdb.user"
+------+----------+
| name | password |
+------+----------+
| bob  | abc123   |
| tom  | 123123   |
| jim  | 654321   |
| jack | 888888   |
+------+----------+
```

案例三：innobackupex

1.软件介绍

一款强大的在线热备份工具

备份过程中不锁库表,适合生产环境

由专业组织 percona 提供(改进 MySQL 分支)

主要含两个组件

xtrabackup：C 程序，支持 innoDB/XtraDB

innobackupex：以 Perl 脚本封装 xtrabackup，还支持 MyISAM

优点:在线备份不锁表，为行级锁

2.安装软件

```
[root@mysql50 ~]# yum -y localinstall libev-4.15-1.el6.rf.x86_64.rpm
[root@mysql50 ~]# yum -y localinstall percona-xtrabackup-24-2.4.7-1.el7.x86_64.rpm
```

提示:不推荐，在 centos6 中因依赖问题容易失败.

拓展装法-官网下载包

https://www.percona.com/downloads/Percona-XtraBackup-LATEST/

3.命令用法

```
[root@mysql50 ~]# man innobackupex
[root@mysql50 ~]# innobackupex –help
```

innobackupex <选项>

常用选项:

--host	主机名
--user	用户名
--port	端口号
--password	密码
--no-timestamp	不用日期命名备份文件存储的子目录名
--databases	数据库名
--databases=”库名”	//单个库
--databases=”库 1 库 2”	//多个库
--databases=”库.表”	//单个表
如果没有此参数，则备份所有库所有表	
--redo-only	日志合并
--apply-log	准备还原(回滚日志)
--copy-back	恢复数据
--incremental 目录名	增量备份
--incremental-basedir=目录名	增量备份时,指定上一次备份数据存储的目录名
--incremental-dir=目录名	准备恢复数据时,指定增量备份数据存储的目录名
--export	导出表信息
--import	导入表空间

4.数据备份与恢复

数据完全备份与恢复

mysql50->备份

```
[root@mysql50 ~]# innobackupex --user=root --password=654321 /backup --no-timestamp
```

提示: 备份目录不需要提前创建. 如果没有--no--timestamp 选项，就会产生日期目录

```
[root@mysql50 ~]# ls /backup/
backup-my.cnf   ibdata1             school                  xtrabackup_checkpoints
bbsdb           mysql               sys                     xtrabackup_info
db1             performance_schema  userdb                  xtrabackup_logfile
ib_buffer_pool  press               xtrabackup_binlog_info
```

除了有备份数据之外，还有备份信息

```
[root@mysql50 ~]# scp -r /backup 192.168.4.51:/root
```

mysql51->恢复

安装 innbackup 包

```
[root@mysql51 ~]# cat backup/xtrabackup_checkpoints
backup_type = full-backuped
from_lsn = 0
to_lsn = 2952514
last_lsn = 2952523
compact = 0
recover_binlog_info = 0
```

做了恢复之后文件会变

```
[root@mysql51 ~]# innobackupex --apply-log /root/backup    //日志恢复
[root@mysql51 ~]# cat backup/xtrabackup_checkpoints
backup_type = full-prepared
from_lsn = 0
to_lsn = 2952514
last_lsn = 2952523
compact = 0
recover_binlog_info = 0
[root@mysql51 ~]# systemctl stop mysqld
[root@mysql51 ~]# rm -rf /var/lib/mysql/*
```

提示:恢复数据时，/var/lib/mysql 目录必须为空

```
[root@mysql51 ~]# innobackupex --copy-back /root/backup/         //数据拷贝回数据库
[root@mysql51 ~]# chown -R mysql:mysql /var/lib/mysql/*
[root@mysql51 ~]# systemctl start mysqld
```

数据增量备份与恢复

命令格式:

```
innobackupex --user root --password 密码 --incremental 目录名 --incremental-basedir=目录名
```

mysql50-> 备份

首次备份需要完全备份　　周一

```
[root@mysql50 ~]# innobackupex --user=root --password=654321 --no-timestamp /root/fullback
```

增量备份(第 1 次增量备份)　　周二

```
[root@mysql50 ~]# innobackupex --user=root --password=654321 --incremental /root/new1dir --incremental-basedir=/root/allback --no-timestamp
[root@mysql50 ~]# cat /root/new1dir/xtrabackup_checkpoints
backup_type = incremental
from_lsn = 2952831
to_lsn = 2952831
last_lsn = 2952840
compact = 0
recover_binlog_info = 0
mysql> insert into t1 values(12),(13),(14),(15),(16);
```

增量备份(第 2 次增量备份)　　周三

```
[root@mysql50 ~]# innobackupex --user=root --password=654321 --incremental /root/new2dir --incremental-basedir=/root/new1dir --no-timestamp
[root@mysql50 ~]# cat /root/new2dir/xtrabackup_checkpoints
backup_type = incremental
from_lsn = 2952831
to_lsn = 2952831
```

```
    last_lsn = 2952840
    compact = 0
    recover_binlog_info = 0
    [root@mysql50 ~]# scp -r /root/allback/ /root/new1dir/ /root/new2dir/ 192.168.4.51:/root/
```

innobackupex 增量恢复

命令格式

```
innobackupex --apply-log --redo-only 完全备份目录     //准备恢复数据
innobackupex --apply-log --redo-only 完全备份目录 --incremental-dir=目录名     //合并日志
innobackupex --copy-back 完全备份目录          //拷贝数据
```

mysql51->恢复

```
    [root@mysql51 ~]# systemctl stop mysqld
    [root@mysql51 ~]# rm -rf /var/lib/mysql/*
    [root@mysql51 ~]# innobackupex --apply-log --redo-only /root/allback/
    [root@mysql51  ~]#  innobackupex  --apply-log  --redo-only  /root/allback/  --incremental-
dir=/root/new1dir
    [root@mysql51  ~]#  innobackupex  --apply-log  --redo-only  /root/allback/  --incremental-
dir=/root/new2dir
    [root@mysql51 ~]# cat /root/allback/xtrabackup_checkpoints
    backup_type = log-applied
    from_lsn = 0
    to_lsn = 2952831
    last_lsn = 2952840
    compact = 0
    recover_binlog_info = 0
    [root@mysql51 ~]# innobackupex --copy-back /root/allback/
    [root@mysql51 ~]# chown -R mysql:mysql /var/lib/mysql/
    [root@mysql51 ~]# systemctl start mysqld
```

从完全备份数据里恢复单张表

```
    mysql> create table db1.t5(name char(10));
    mysql> insert into db1.t5 values("bob"),("lucy"),("jerry"),("jack");
    mysql> show tables;
    +---------------+
    | Tables_in_db1 |
    +---------------+
    | a1            |
    | t1            |
    | t2            |
    | t3            |
    | t4            |
    | t5            |
    | t7            |
    | user2         |
    | user3         |
    +---------------+
```

完全备份

```
    [root@mysql50 ~]# innobackupex --user=root --password=654321 /root/allbackup --no-timestamp
    [root@mysql50 ~]# mysql -uroot -p654321 -e "drop table db1.t5"
```

恢复完全备份中的单张表的步骤

1)把删除的按照原表结构创建出来

```
    mysql> create tables db1.t5(name char(10));
```

2)删除表空间

```
[root@mysql50 ~]# ls /var/lib/mysql/db1/
t5.frm  t5.ibd
```

提示:不能直接手动删除 t5.ibd 文件，也不能用备份出来直接替换，后期会产生问题.

原因是 innodb 存储引擎，需要和事务保持一致

```
mysql> alter table db1.t5 discard tablespace;
mysql> select * from db1.t5;
ERROR 1814 (HY000): Tablespace has been discarded for table 't5'
```

3)在备份文件里导出表信息

```
[root@mysql50 ~]# innobackupex --apply-log --export /root/allbackup/
[root@mysql50 ~]# ls allbackup/db1/
t5.frm  t5.cfg  t5.ibd  t5.exp  ...
```

4)包导出的表信息文件拷贝到数据库目录下,并修改所有者核组用户为 mysql

```
[root@mysql50 ~]# cp allbackup/db1/t5.{cfg,exp,ibd} /var/lib/mysql/db1/
[root@mysql50 ~]# chown mysql:mysql /var/lib/mysql/db1/t5*
```

5)导入表信息

```
mysql> alter table db1.t5 import tablespace;
```

6)删除数据库目录下的表信息文件

```
[root@mysql50 ~]# rm -rf /var/lib/mysql/db1/t5.{exp.cfg}
```

7)查看表记录

```
mysql> select * from t5;
+-------+
| name  |
+-------+
| bob   |
| lucy  |
| jerry |
| jack  |
+-------+
```

Day06 MySQL 主从同步模式

案例一　mysql 主从同步原理及配置

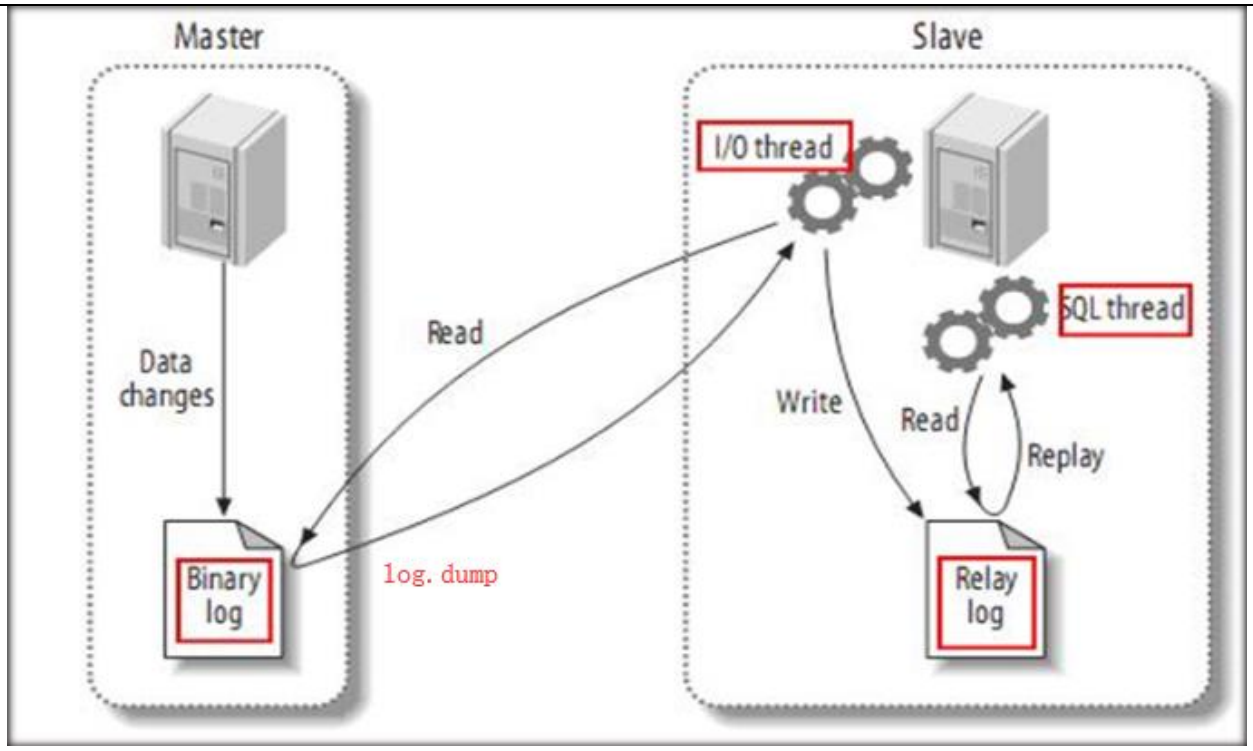
1．什么是 MySQL 主从同步

实现数据自动同步的服务结构，结构中分 2 种角色;

主服务器：接受客户端访问的数据库服务器

从服务器：自动接收主库服务器同步数据到本地的数据服务器

2.MySQL 主从同步工作原理



主库开启 binlog 日志，主库有一个 I/O 线程，名字叫 log.dump．从库 I/O 线程对主库发起同步请求，主库把 binlog 日志推送给从库．从库 I/O 进程接受数据并写入自己的中继日志(relay log)．从库 SQL 线程读取中继日志写入本库．

3．配置 MySQL 主从同步

192.168.4.50 clinet

192.168.4.51 主(master)

192.168.4.52 从(slave)

配置主服务器 192.168.4.51

1)启用 binlog 日志

```
[root@mysql51 ~]# vim /etc/my.cnf
log_bin=mysql51
server_id=51
binlog_format="mixed"
mysql> show master status;
```

2)用户授权

```
mysql> grant replication slave on *.* to repluser@"192.168.4.52" identified by "123456";
```

3)查看 binlog 日志信息

```
mysql> mysql> show master status;
+----------------+----------+--------------+------------------+-------------------+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+----------------+----------+--------------+------------------+-------------------+
| mysql51.000001 |      452 |              |                  |                   |
+----------------+----------+--------------+------------------+-------------------+
```

4)当主从配置完成后，查看进程

```
mysql> show processlist;
```

```
mysql> show processlist;
+----+----------+--------------------+------+-------------+------+-----------------------------------------------------------------+------------------+
| Id | User     | Host               | db   | Command     | Time | State                                                           | Info             |
+----+----------+--------------------+------+-------------+------+-----------------------------------------------------------------+------------------+
|  3 | root     | localhost          | NULL | Query       |    0 | starting                                                        | show processlist |
|  4 | repluser | 192.168.4.52:57954 | NULL | Binlog Dump |   75 | Master has sent all binlog to slave; waiting for more updates | NULL             |
+----+----------+--------------------+------+-------------+------+-----------------------------------------------------------------+------------------+
```

配置从服务器 192.168.4.52

1)指定 server_id=52

```
[root@mysql52 ~]#: vim /etc/my.cnf
server_id=52
[root@mysql52 ~]#: systemctl restart mysqld
```

2)指定主服务器信息

```
mysql> change master to
->master_host="192.168.4.51",
->master_user="repluser",
->master_password="123456",
->master_log_file="mysql51.000001",
->master_log_pos=452;
```

3)启动 slave 进程

```
mysql> start slave;
```

4)查看进程信息

```
mysql> show slave status\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 192.168.4.51
                  Master_User: repluser
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql51.000001
          Read_Master_Log_Pos: 452
               Relay_Log_File: mysql52-relay-bin.000002
                Relay_Log_Pos: 318
        Relay_Master_Log_File: mysql51.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 452
              Relay_Log_Space: 527
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
```

```
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 51
                  Master_UUID: d44fa6ad-9ae9-11e9-8f7f-525400c0123f
             Master_Info_File: /var/lib/mysql/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log; waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set:
            Executed_Gtid_Set:
                Auto_Position: 0
         Replicate_Rewrite_DB:
                 Channel_Name:
           Master_TLS_Version:
```

IO 出错一般都是用户授权问题

SQL 出错中继日志出现的命令在从库中缺少库表或者用户

4.还原数据库

从库相关文件 cd /var/lib/mysql ->master.info

master-relay-bin.000001 等都是中继日志

```
[root@mysql52 mysql]#: rm -rf /var/lib/mysql/mysql52-relay-bin.*
```

提示：从库只保留最新的两个中继日志

```
[root@mysql52 mysql]#: cat relay-log.info     //中继文件记录信息
7
./mysql52-relay-bin.000002
318
mysql51.000001
452
0
0
1
```

以上文件全删了，重启服务就可以恢复独立数据库

5.验证配置

1）在主服务器添加访问数据库的连接用户

mysql51

```
mysql> create database db7;
mysql> grant all on db7.* to webuser@"%" identified by "123456";
```

2）在客户端链接主服务器,对数据做操作

mysql50

```
[root@mysql50 ~]#: mysql -uwebuser -p123456 -h192.168.4.51
mysql> show grants;
+-------------------------------------------------+
| Grants for webuser@%                            |
+-------------------------------------------------+
| GRANT USAGE ON *.* TO 'webuser'@'%'             |
| GRANT ALL PRIVILEGES ON `db7`.* TO 'webuser'@'%' |
+-------------------------------------------------+
mysql> create table db7.t1(id int);
mysql> insert into db7.t1 values(10101);
```

3）在从服务器本机查看是否和主库服务器数据一致

mysql52

```
mysql> select * from db7.t1;
```

案例二：同步模式

1.MySQL 主从同步结构模式

一主一从

一主多从

案例一中 52 是 51 的从库，相同的方式把 53 配置为 51 的从库

1）在服务器 53 运行 mysql

2）在没有配置从服务器之前,要有主服务器上数据

mysql51

```
[root@mysql51 ~]#: mysqldump -uroot -p123456 db7 > /root/db7.sql
[root@mysql51 ~]#: scp /root/db7.sql 192.168.4.53:/root/
```

mysql53

```
mysql> create database db7;
mysql> source /root/db7.sql
```

3）设置主机 53 的 server_id

```
[root@mysql53 ~]#: vim /etc/my.cnf
server_id=53
[root@mysql53 ~]#: systemctl restart mysqld
```

4）指定主服务器信息

```
mysql>    change    master    to    master_host="192.168.4.51",    master_user="repluser",
master_password="123456", master_log_file="mysql51.000001", master_log_pos=899;
```

5）启动 slave 进程

```
mysql> start slave;
```

6）查看进程信息

```
mysql> show slave status\G;
         Slave_IO_Running: Yes
         Slave_SQL_Running: Tes
```

7）客户端测试

在客户端 50 主机连接主库数据服务器访问数据

```
mysql -h192.168.4.51 -uwebuser -p123456
mysql> insert into db7.t1 values(888);
mysql> create table db7.t2(name char(10));
mysql> insert into db7.t2 values(“bob”);
mysql> select * from db7.t2;
```

分别从 2 个从库服务器查看数据(和主库服务器一致)

```
mysql52\mysql53
```

```
mysql> select * from db7.t1;
mysql> select * from db7.t2;
```

主从从

当主库坏掉之后，客户端可以连接下一级主库,进行写操作

mysql51-->mysql52-->mysql54

1)配置 mysql52(需要开启级联复制)

```
[root@mysql52 ~]#: vim /etc/my.cnf
binlog_fotmat="mixed"
log-bin=mysql52
log_slave_updates        //允许级联复制
```

提示:开启级联复制的原因是 mysql52 通过执行中继日志的 SQL 语句并不会记录在自己的 binlog 日志中

```
[root@mysql52 ~]#: systemctl restart mysqld
mysql> grant replication slave on *.* to repluser@"%" identified by "123456";
mysql> show master status;
+----------------+----------+--------------+------------------+-------------------+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+----------------+----------+--------------+------------------+-------------------+
| mysql52.000001 |      154 |              |                  |                   |
+----------------+----------+--------------+------------------+-------------------+
[root@mysql52 ~]#: mysqldump -uroot -p123456 db7 > /root/db7.sql
[root@mysql52 ~]#: scp /root/db7.sql 192.168.4.54:/root
```

2)配置 mysql54

```
mysql> create database db7;
mysql> use db7;
mysql> source /root/db7.sql
[root@mysql54 ~]#: vim /etc/my.cnf
server_id=54
[root@mysql54 ~]#: systemctl restart mysqld
mysql>  change  master  to  master_host="192.168.4.52",  master_user="repluser",
master_password="123456", master_log_file="mysql52.000001",master_log_pos=154;
mysql> start slave;
slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

3)测试配置

在客户端连接主服务器 51 存数据

```
[root@mysql50 ~]#: mysql -uwebuser -p123456 -h192.168.4.51
mysql> insert into db7.t1 values(777),(888),(999);
```

在从服务器 52 和 54 主机上都可以看到主服务器一样的数据

```
mysql52,mysql54
mysql > select * from db7.t1;
```

主主结构(互为主从)　　一般不单独用,用于高可用集群

2.MySQL 主从同步复制模式

异步复制(Asynchronous replication)(默认)

主库执行玩一次事务后,立即将结果返回给客户端,并不关系从库是否已经接受并处理.

全同步服务(Fully syschronous replication)

当主库执行完一次事务，且所有从库都执行了该事务后才返回给客户端

半同步复制(Semissynchronous replication)

介于异步复制和完全同步复制之间

主库在执行完一次事务后,等待至少一个从库收到并写到 relay log 中才返回给客户端

举例:启用数据库服务器 52 得半同步复制模式

1)查看是否允许动态加载模块

```
mysql> show variables like 'have_dynamic_loading';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| have_dynamic_loading | YES   |
+----------------------+-------+
```

2)加载模块(用户需要有 super 权限)(52 同时是主库也是从库)

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so'; //主库执行
mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';  //从库执行
```

3)查看加载信息

```
mysql> select PLUGIN_NAME,PLUGIN_STATUS from INFORMATION_SCHEMA.PLUGINS where PLUGIN_NAME like '%semi%';
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| rpl_semi_sync_master | ACTIVE        |
| rpl_semi_sync_slave  | ACTIVE        |
+----------------------+---------------+
```

4)启用模块

在安装完插件后,半同步复制默认是关闭的

```
mysql> set global rpl_semi_sync_master_enabled=1;    //主库执行
mysql> set global rpl_semi_sync_slave_enabled=1;     //从库执行
```

5)查看启用信息

```
mysql> show variables like 'rpl_semi_sync_%_enabled';
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| rpl_semi_sync_master_enabled | ON    |
| rpl_semi_sync_slave_enabled  | ON    |
+------------------------------+-------+
```

6)修改配置文件,使其永久生效

```
vim /etc/my.cnf
[root@mysql52 ~]#: vim /etc/my.cnf
plugin-load="rpl_semi_sync_master=semisync_master.so;rpl_semi_sync_slave=semisync_slave.so"
rpl-semi-sync-master-enabled=1
rpl-semi-sync-slave-enabled=1
[root@mysql52 ~]#: systemctl restart mysqld
```

3. MySQL 主从同步常用配置选项(/etc/my.cnf)

适用于 Master 服务器

选项	用途
binlog_do_db=name	设置 master 对那些库记日志

binlog_ignore_db=name　　设置 mastrer 对哪些库不记日志

适用于 slave 服务器

log_slave_updates　　记录从库更新，允许链式复制

relay_log=日志名　　指定中继日志文件名

replicate_do_db=数据库名　　仅复制指定库，其他库将被忽略,此选项可设置多条(省略时复制所有库)

replicate_ignore_db=数据库名 不复制哪些库,其他库将被忽略,ignore-db 与 do-db 只需选用其中一种

Day07 读写分离多实例以及性能调优

准备工作

所有从服务器都恢复为独立的数据库服务器

把主从配置在/etc/my.cnf 文件并注释

重启 mysql 服务

50 client

51、52、57 数据库服务器

案例一：MySQL 数据读写分离

客户端访问数据的查询请求和写请求分别给不同的数据库服务器处理

查询　　select

写　　insert update delete

读写分离的目的:

减轻 1 台数据库服务器的并发访问压力

提高机器硬件的利用率(主从同步的情况下)

实现数据读写分离方法

通过程序实现(让程序员写代码实现)

通过安装软件提供的(中间件)

中间件:架设在客户端和服务器中间的软件

mycat mysql-proxy maxscale....

配置数据读写分离(maxscale+mysql 一主一从同步)

client50

maxscale57

mysql-master51(写)　mysql-slave52(读)

配置步骤

1.部署 mysql 一主一从同步结构

步骤见 Day06 的笔记

2.配置数据读写分离服务器 57

```
[root@GYP-HOME ~]# scp maxscale-2.1.2-1.rhel.7.x86_64.rpm 192.168.4.57:/root
[root@mysql57 ~]#: rpm -ivh maxscale-2.1.2-1.rhel.7.x86_64.rpm
[root@mysql57 ~]#: vim /etc/maxscale.cnf
[maxscale]
```

```
threads=auto          //运行时开启的线程数量
[server1]
type=server
address=127.0.0.1
port=3306
protocol=MySQLBackend

[server1]                          //定义数据库
type=server
address=192.168.4.51
port=3306
protocol=MySQLBackend

[server2]                          //定义第二台数据库
type=server
address=192.168.4.52
port=3306
protocol=MySQLBackend

[MySQL Monitor]                    //监控两台数据库
type=monitor
module=mysqlmon
servers=server1,server2
user=scalemon                      //监控用户，区分谁是主库,谁是从库
passwd=123456
monitor_interval=10000

#[Read-Only Service]               //只读服务器全部注释
...

[Read-Write Service]
type=service
router=readwritesplit
servers=server1,server2
user=scaleroute                    //路由用户:查看数据库访问用户是否存在
passwd=123456
max_slave_connections=100%

[MaxAdmin Service]                 //管理服务
type=service
router=cli

#[Read-Only Listener]              //注释只读服务器的端口号
...

[Read-Write Listener]
type=listener
service=Read-Write Service
protocol=MySQLClient
port=4006

[MaxAdmin Listener]
```

```
type=listener
service=MaxAdmin Service
protocol=maxscaled
socket=default
port=4016                          //管理服务的端口
```

3.用户授权(根据配置文件的设置在 2 台数据库服务器上添加对应的用户)

mysql51

监控用户

```
mysql> grant replication slave, replication client on *.* to scalemon@'%' identified by '123456';
```

replication slave 是用来监控主从同步

replication client 监控数据库服务是否正常

路由用户

```
mysql> grant select on mysql.* to scaleroute@'%' identified by '123456';
```

路由用户的作用:

```
mysql -h192.168.4.57 -uyaya -p123456
```

此时 57 上并没有用户 yaya，这个用户应该再 51 和 52 上面，因此需要验证 51 和 52 上面是否有 yaya 这个用户以及密码．那么 57 连接数据库 51,52 用来验证的用户就是路由用户．

mysql52 验证用户授权是否正确

```
mysql> select user,host from mysql.user where user in ("scalemon","scaleroute");
+------------+------+
| user       | host |
+------------+------+
| scalemon   | %    |
| scaleroute | %    |
+------------+------+
```

4.启动服务

mysql57

测试两个用户是否能正常连接

```
[root@mysql57 ~]#: mysql -h192.168.4.51 -uscalemon -p123456
[root@mysql57 ~]#: mysql -h192.168.4.51 -uscaleroute -p123456
```

启动

```
[root@mysql57 ~]#: maxscale -f /etc/maxscale.cnf
[root@mysql57 ~]#: ps -C maxscale
  PID TTY          TIME CMD
12140 ?        00:00:00 maxscale
[root@mysql57 ~]#: ss -antpu | grep maxscale
tcp  LISTEN  0  128  :::4006           :::*       users:(("maxscale",pid=12140,fd=11))
tcp  LISTEN  0  128  :::4016           :::*       users:(("maxscale",pid=12140,fd=12))
[root@mysql57 ~]#: killall -9 maxscale        //停止服务
```

5.测试配置

查看管理信息,在 57 本机自己访问自己

```
[root@mysql57 ~]#: maxadmin -uadmin -pmariadb -P4016
MaxScale> list servers
Servers.
-------------------+-----------------+-------+-------------+--------------------
```

```
Server              | Address         | Port  | Connections | Status
--------------------+-----------------+-------+-------------+--------------------
server1             | 192.168.4.51    |  3306 |           0 | Master, Running
server2             | 192.168.4.52    |  3306 |           0 | Slave, Running
--------------------+-----------------+-------+-------------+--------------------
```

测试数据读写分离（客户端连接 192.168.4.57 访问数据）

在主数据库服务器上添加访问数据的用户

```
[root@mysql57 ~]#: mysql -uroot -p123456 -h192.168.4.51
mysql> create database db8;
mysql> create table db8.t1(id int);
mysql> grant select,insert on db8.* to yaya@'%' identified by "123456";
```

客户端连接 57 主机访问数据

```
[root@mysql50 ~]# mysql -uyaya -p123456 -h192.168.4.57 -P4006
mysql> insert into db8.t1 values(111),(222);
[root@mysql50 ~]# mysql -uyaya -p123456 -h192.168.4.52 -e "insert into db8.t1 values(52)"
[root@mysql50 ~]# mysql -uyaya -p123456 -h192.168.4.57 -P4006 -e "select * from db8.t1"
+------+
| id   |
+------+
|  111 |
|  222 |
|   52 |
+------+
```

案例二　MySQL 多实例

多实例概念:在一台物理主机上运行多个数据库服务

使用多实例原因:

节约运维成本

提高硬件利用率

配置多实例步骤

1 安装软件

```
[root@mysql50 ~]# mv mysql-5.7.20-linux-glibc2.12-x86_64 /usr/local/mysql
[root@mysql50 mysql]# ls /usr/local/mysql/bin/
[root@mysql50 mysql]# systemctl stop mysqld
[root@mysql50 mysql]# mv /etc/my.cnf /root/
```

2 修改服务的主配置文件

```
[root@mysql50 mysql]# vim /etc/my.cnf
[mysqld_multi]
mysqld=/usr/local/mysql/bin/mysqld_safe
mysqladmin=/usr/local/mysql/bin/mysqladmin
user=root

[mysqld1]
datadir=/dir1
socket=/dir1/mysql3307.sock
port=3307
log-error=/dir1/mysql3307.log
pid-file=/dir1/mysql3307.pid
[mysqld2]
```

```
datadir=/dir2
socket=/dir2/mysql3308.sock
port=3308
log-error=/dir2/mysql3308.log
pid-file=/dir2/mysql3308.pid
```

数据库需要存在的有:

数据库目录

端口号

错误日志文件

pid 号文件

socket 文件(套接字文件，自己访问自己,传输信息用的)

```
[root@mysql50 mysql]# mkdir /dir{1,2}
```

3 启动服务

```
[root@mysql50 mysql]# /usr/local/mysql/bin/mysqld_multi start 1
[root@mysql50 ]# /usr/local/mysql/bin/mysql -uroot -p'!;REao#PQ2&V' -S /dir1/mysql3307.sock
mysql> alter user user() identified by '123456';
[root@mysql50 mysql]# /usr/local/mysql/bin/mysqld_multi start 2
[root@mysql50 ]# /usr/local/mysql/bin/mysql -uroot -p'lnA76%yA(r<Z' -S /dir2/mysql3308.sock
[root@mysql50 mysql]# netstat -antpu | grep mysqld
tcp6       0      0 :::3307                 :::*                    LISTEN      7660/mysqld
tcp6       0      0 :::3308                 :::*                    LISTEN      8039/mysqld
[root@mysql50 mysql]# /usr/local/mysql/bin/mysqld_multi stop 1
```

4.验证

访问多实例服务，对数据做操作

案例三 MySQL 性能调优

升级硬件(cpu 内存 存储)

网络带宽

优化服务架构(网络架构是否有数据传输瓶颈)

优化数据库服务运行参数

```
mysql> show variables;                          //查看所有的变量
mysql> show variables like '%timeout%'          //查看关键字信息
mysql> set [global] 变量名=值
mysql> set global connect_timeout=5;            //临时修改
vim /etc/my.cnf
变量名=值
mysql> show status;                             //当前状态的值、
mysql> show status like '%connect%';            //查看关键字信息
```

并发及连接控制

连接数、连接超时

选项	含义
max_connections	允许的最大并发连接数
connect_timeout	等待连接超时,默认 10 秒,仅登录时有效
wait_timeout	等待关闭连接的不活动超时秒数,默认 28800 秒(8 小时)

```
mysql> flush status;        //刷新状态信息
mysql> show global status like "max_used_connections";     //曾经最大的并发连接数
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| Max_used_connections | 4     |
+----------------------+-------+
mysql> show variables like 'max_connections';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_connections | 151   |
+-----------------+-------+
```

设置并发连接数的公式

最大并发连接/最大并发连接数=85%

Max_used_connections/max_connections=85%

缓存参数控制

缓冲区、线程数量、开表数量

选项	含义
key_buffer_size	用于 MyISAM 引擎的关键索引缓存大小
sort_buffer_size	为每个要排序的线程分配此大小的缓存空间
read_buffer_size	为顺序读取表记录保留的缓存大小
thread_cache_size	允许保存在缓存中备重用的线程数量
table_open_cache	为所有线程缓存的打开的表的数量(减少从硬盘调入内存的次数)

```
mysql> show variables like "key_buffer_size";        //数字显示的为字节为 8M
+-----------------+---------+
| Variable_name   | Value   |
+-----------------+---------+
| key_buffer_size | 8388608 |
+-----------------+---------+

mysql> show variables like 'sort_buffer_size';       //为 256k
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| sort_buffer_size | 262144 |
+------------------+--------+
```

此参数的控制的是 mysqld 进程做排序的内存大小(此值增大可提高 order 和 group 的速度)

比如 select name.uid from user order by uid desc;

```
mysql> show global status like "open%tables";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Open_tables   | 118   |
| Opened_tables | 139   |
+---------------+-------+
```

```
mysql> show variables like "table_open_cache";
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| table_open_cache | 2000  |
+------------------+-------+
```

Opened_tables/table_open_cache 理想比率<=95%

sql 查询优化

优化服务查询缓存

数据库服务处理查询请求的过程

select 查询请求先查看查询缓存(内存),如果不在缓存上,则从以硬盘中查找并把结果缓存在内存里

```
mysql> show variables like "query_cache%";           //查看与查询缓存相关的参数
+------------------------------+---------+
| Variable_name                | Value   |
+------------------------------+---------+
| query_cache_limit            | 1048576 |
| query_cache_min_res_unit     | 4096    |
| query_cache_size             | 1048576 |           //1M
| query_cache_type             | OFF     |
| query_cache_wlock_invalidate | OFF     |
+------------------------------+---------+
```

提示:在生产环境下,一般不用 mysql 的物理内存做缓存，而是用专门的缓存服务器.

```
mysql> show global status like "qcache%";            //查看当前的查询缓存统计
+-------------------------+---------+
| Variable_name           | Value   |
+-------------------------+---------+
| Qcache_free_blocks      | 1       |
| Qcache_free_memory      | 1031832 |
| Qcache_hits             | 0       |               //客户端通过查询缓存查到数据的次数
| Qcache_inserts          | 0       |               //在数据查询缓存中查找的次数
| Qcache_lowmem_prunes    | 0       |
| Qcache_not_cached       | 560     |
| Qcache_queries_in_cache | 0       |
| Qcache_total_blocks     | 1       |
+-------------------------+---------+
```

提示:Qcache_hits/Qcache_inserts 率越高表示命中率越高，如果值低了,那么增加 query_cache_size 大小

优化程序员访问数据的 sql 命令

MySQL 日志类型

常用日志类型及选项

类型	用途	配置	含义
错误日志	记录启动/运行/停止过程中的错误消息	log-error[=name]	
查询日志	记录客户端连接和查询操作	general-log general-log-file=	
慢查询日志	记录韩式较长或不使用索引的查询操作	slow-query-log	启用慢查询
		slow-query-log-file=	指定慢查询日志文件

long-query-time=	超时时间,默认 10 秒
log-queries-not-using-indexes	记录未使用索引的查询

```
vim /etc/my.cnf
[root@mysql51 ~]#: vim /etc/my.cnf
slow-query-log
long-query-time=5
log-queries-not-using-indexes
[root@mysql51 ~]#: systemctl restart mysqld
[root@mysql51 ~]#: mysqldumpslow /var/lib/mysql/mysql51-slow.log
mysql> select sleep(10);
```

统计慢日志查询的个数,内容等信息

这个日志可以给程序员去优化他的 SQL 语句.

Day08 MHA 集群

案例一: MHA 集群概念及部署

0.前提准备

56 可以无密码 ssh 访问 51-55

51-55 主机可以彼此互相无密码 ssh 访问

mha-soft-student 拷贝给 51-56

51-55 恢复为独立的数据库服务器

ip 规划

角色	ip 地址	主机名
Master 主节点服务器	192.168.4.51	mysql51
备用 1 主节点服务器	192.168.4.52	mysql52
备用 2 主节点服务器	192.168.4.53	mysql53
第 1 台 slave 服务器	192.168.4.54	mysql54
第 2 台 slave 服务器	192.168.4.55	mysql55
MHA_manager 服务器	192.168.4.56	mysql56
VIP 地址	192.168.4.100	

ssh 免密登录做法:

只需要 6 台服务器上有相同配对的公私钥即可.

1. MHA 介绍(Master High Availability)

由日本 DeNA 公司 youshimaton 开发, perl 语言编写

是一套优秀的实现 MySQL 高可用的解决方案

数据库的自动故障切换操作能做到 0~30 秒之内

MHA 能确保再故障切换过程中保持数据的一致性,以达到真正意义的高可用

MHA 组成

Manager(管理节点)

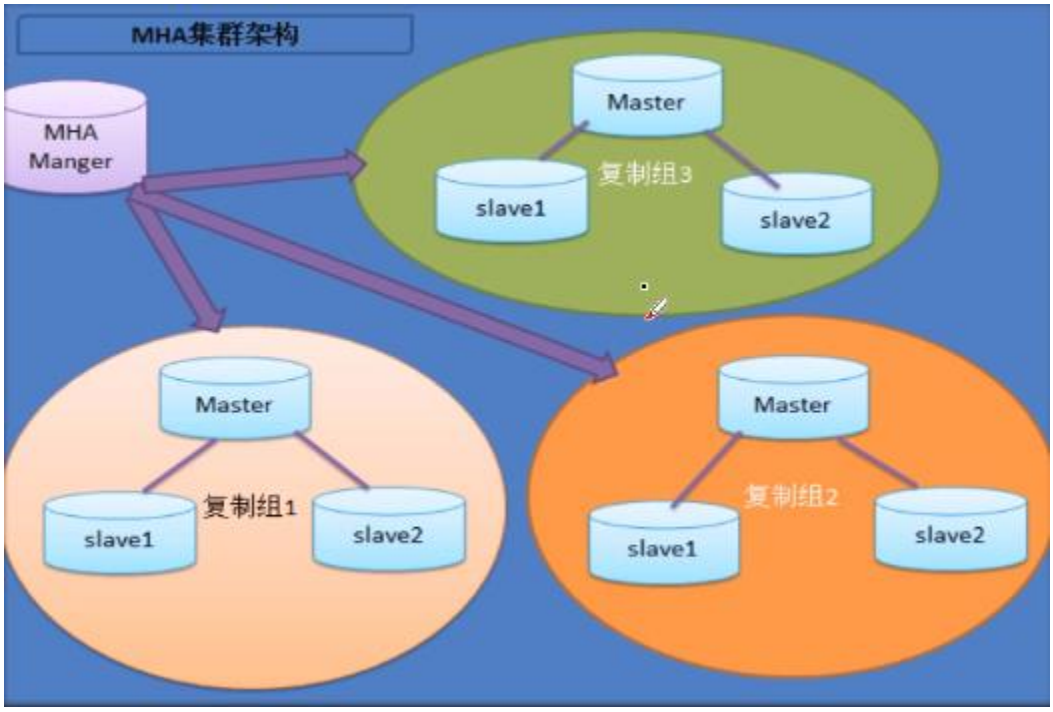
可以单独部署在一台独立的主机上，管理其他节点

也可以部署在一台 slave 节点上

Node(数据节点)

运行在每台 MySQL 服务器上

MHA 工作过程(原理)

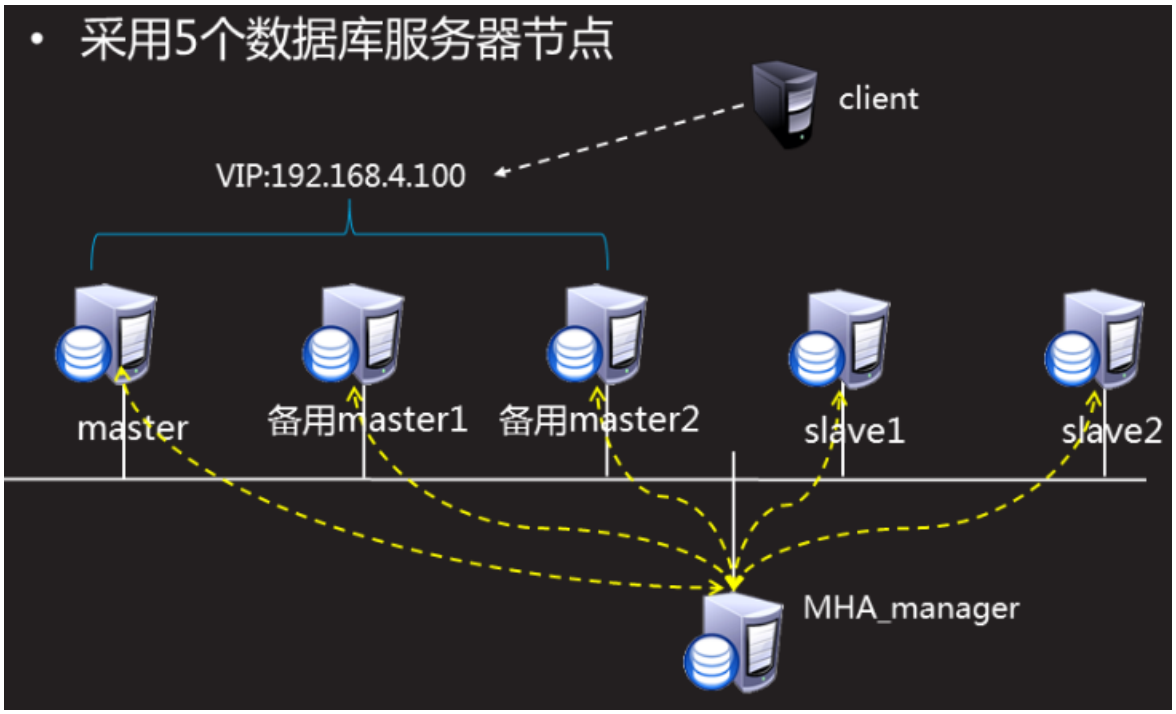


一主多从，主库被访问,如果主库宕机,那么在两个从里面选举新主库即可.

51 主,52,53 为从库. 56 监控 51 主库，如果 51 宕机,那么 52 和 53 中选举出一台做新的主库.

54,55 纯做备份,不参与主库的选举.客户端连接主库的 ip 应该为 VIP.

拓扑图



2.安装步骤

1).51-55 安装 node 相关 rpm 包

```
[root@GYP-HOME ~]# for i in mysql{51..55}
> do
> ssh $i "yum localinstall -y /root/mha-soft-student/mha4mysql-node-0.56-0.el6.noarch.rpm"
> done
```

在管理主机上安装 mha_node 和 mha_manager 包

```
[root@mysql56 mha-soft-student]#: yum -y localinstall mha4mysql-node-0.56-0.el6.noarch.rpm
[root@mysql56 mha-soft-student]#: yum -y localinstall perl-*
[root@mysql56 mha-soft-student]#: yum -y localinstall mha4mysql-manager-0.56-0.el6.noarch.rpm
[root@mysql56 mha-soft-student]#: tar xf mha4mysql-manager-0.56.tar.gz
[root@mysql56 mha-soft-student]#: cd mha4mysql-manager-0.56/
[root@mysql56 mha4mysql-manager-0.56]#: yum -y install perl-ExtUtils-* perl-CPAN-*
[root@mysql56 mha4mysql-manager-0.56]#: perl Makefile.PL
*** Module::AutoInstall version 1.03
*** Checking for Perl dependencies...
[Core Features]
- DBI                   ...loaded. (1.627)
- DBD::mysql            ...loaded. (4.023)
- Time::HiRes           ...loaded. (1.9725)
- Config::Tiny          ...loaded. (2.14)
- Log::Dispatch         ...loaded. (2.41)
- Parallel::ForkManager ...loaded. (1.18)
- MHA::NodeConst        ...loaded. (0.56)
*** Module::AutoInstall configuration finished.
Checking if your kit is complete...
Looks good
Writing Makefile for mha4mysql::manager
Writing MYMETA.yml and MYMETA.json
[root@mysql56 mha4mysql-manager-0.56]#: make && make install
```

2)配置 MySQL 主从同步(一主多从)

配置主服务器 51

```
[root@mysql51 ~]#: vim /etc/my.cnf
[mysqld]
validate_password_policy=0
validate_password_length=6
log_bin=mysql51
server_id=51
binlog_format="mixed"
plugin-load="rpl_semi_sync_master=semisync_master.so;rpl_semi_sync_slave=semisync_slave.so"
rpl-semi-sync-master-enabled=1
rpl-semi-sync-slave-enabled=1
relay_log_purge=off
[root@mysql51 ~]#: systemctl restart mysqld
mysql> grant replication slave on *.* to rpluser@'%' identified by '123456';
```

配置从服务器 52(备用主库)

```
[root@mysql52 ~]#: vim /etc/my.cnf
[mysqld]
validate_password_policy=0
validate_password_length=6
server_id=52
log_bin=mysql52
binlog_format="mixed"
plugin-load="rpl_semi_sync_master=semisync_master.so;rpl_semi_sync_slave=semisync_slave.so"
```

```
    rpl-semi-sync-master-enabled=1
    rpl-semi-sync-slave-enabled=1
    relay_log_purge=off
    [root@mysql52 ~]#: systemctl restart mysqld
    mysql>  change  master  to  master_host="192.168.4.51",  master_user="rpluser",  master_password="123456",
master_log_file="mysql51.000001", master_log_pos=598;
    mysql> start slave;
```

配置从服务器 53(备用主库)

```
    [root@mysql53 ~]#: vim /etc/my.cnf
    [mysqld]
    validate_password_policy=0
    validate_password_length=6
    log_bin=mysql53
    binlog_format="mixed"
    server_id=53
    relay_log_purge=off
    plugin-load="rpl_semi_sync_master=semisync_master.so;rpl_semi_sync_slave=semisync_slave.so"
    rpl-semi-sync-master-enabled=1
    rpl-semi-sync-slave-enabled=1
    [root@mysql54 ~]#: systemctl restart mysqld
    mysql>  change  master  to  master_host="192.168.4.51",  master_user="rpluser",  master_password="123456",
master_log_file="mysql51.000001", master_log_pos=598;
    mysql> start slave;
```

配置从服务器 54(纯从库)

```
    [root@mysql54 ~]#: vim /etc/my.cnf
    [mysqld]
    validate_password_policy=0
    validate_password_length=6
    server_id=54
    plugin-load="rpl_semi_sync_slave=semisync_slave.so"
    rpl_semi_sync_slave-enabled=1
    relay_log_purge=off
    [root@mysql54 ~]#: systemctl restart mysqld
    mysql>  change  master  to  master_host="192.168.4.51",  master_user="rpluser",  master_password="123456",
master_log_file="mysql51.000001", master_log_pos=598;
    mysql> start slave;
```

配置从服务器 55(纯从库)

```
    [root@mysql55 ~]#: vim /etc/my.cnf
    [mysqld]
    validate_password_policy=0
    validate_password_length=6
    server_id=55
    plugin-load="rpl_semi_sync_slave=semisync_slave.so"
    rpl_semi_sync_slave-enabled=1
    relay_log_purge=off
    [root@mysql55 ~]#: systemctl restart mysqld
    mysql>  change  master  to  master_host="192.168.4.51",  master_user="rpluser",  master_password="123456",
master_log_file="mysql51.000001", master_log_pos=598;
    mysql> start slave;
```

3)配置管理主机 192.168.4.56

编写配置文件

```
    [root@mysql56 ~]#: mkdir /etc/mha-manager
    [root@mysql56 ~]#: cp /root/mha-soft-student/mha4mysql-manager-0.56/samples/conf/app1.cnf
/etc/mha-manager/
    [root@mysql56 ~]#: vim /etc/mha-manager/app1.cnf
    [server default]                                    //服务默认配置
```

```
manager_workdir=/etc/mha-manager                //工作目录
manager_log=/etc/mha-manager/manager.log        //日志
master_ip_failover_script=/etc/mha-manager/master_ip_failover    //自动 failover 得切换脚本
ssh_user=root
ssh_port=22
repl_user=rpluser           //主从同步用户名
repl_password=123456        //主从同步密码
user=monitor                //连接数据库用户名
password=123456             //密码
[server1]
hostname=192.168.4.51
candidate_master=1          //竞选主库
[server2]
hostname=192.168.4.52
candidate_master=1
[server3]
hostname=192.168.4.53
candidate_master=1
[server4]
hostname=192.168.4.54
no_master=1                 //不竞选主库
[server5]
hostname=192.168.4.55
no_master=1
```

创建故障切换脚本

```
[root@mysql56 mha-manager]#: cp /root/mha-soft-student/mha4mysql-manager-0.56/samples/scripts/master_ip_failover /etc/mha-manager/     //需要修改，用老师改的
[root@mysql56 ~]#: chmod +x /etc/mha-manager/master_ip_failover
```

把 vip 地址部署在当前的主库上(51)

```
[root@mysql51 ~]#: ifconfig eth0:1 192.168.4.100/24
[root@mysql51 ~]#: ifconfig eth0:1
eth0:1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.4.100  netmask 255.255.255.0  broadcast 192.168.4.255
        ether 52:54:00:c0:12:3f  txqueuelen 1000  (Ethernet)
```

用户授权(根据配置文件)

给备选主库设置同步数据的连接用户 rpluser

```
[root@mysql52 ~]#: mysql -uroot -p123456
mysql> grant replication slave on *.* to rpluser@'%' identified by '123456';
[root@mysql53 ~]#: mysql -uroot -p123456
mysql> grant replication slave on *.* to rpluser@'%' identified by '123456';
```

管理主机 56 监控数据库服务器状态的连接用户 monitor(51-55)

```
[root@mysql51 ~]#: mysql -uroot -p123456
mysql> grant all on *.* to monitor@'%' identified by '123456';
```

3.测试配置(192.168.4.56)

1)测试 ssh 无密码登录

[root@mysql56 ~]#: masterha_check_ssh --conf=/etc/mha-manager/app1.cnf

2)测试 mysql 主从同步

[root@mysql56 ~]#: masterha_check_repl --conf=/etc/mha-manager/app1.cnf

3)启动管理服务

[root@mysql56 ~]#: masterha_check_status --conf=/etc/mha-manager/app1.cnf
app1 is stopped(2:NOT_RUNNING).

使用 masterha_master

--remove-dead_master_conf //删除宕机主库配置

--ignore_last_failover //忽略 xx.health 文件．8 个小时连续宕机也会切换

[root@mysql56 ~]#: masterha_manager --conf=/etc/mha-manager/app1.cnf --remove-dead_master_conf --ignore_last_failover
[root@mysql56 ~]#: masterha_check_status --conf=/etc/mha-manager/app1.cnf
app1 (pid:15953) is running(0:PING_OK), master:192.168.4.51
[root@mysql56 ~]#: cat /etc/mha-manager/app1.master_status.health
15953 0:PING_OK master:192.168.4.51

4)测试 mysql 服务高可用

在主库 51 上添加访问数据的连接用户

[root@mysql51 ~]#: mysql -uroot -p123456
mysql> create database db9;
mysql> create table db9.t1(id int);
mysql> grant select,insert on db9.* to yaya9@'%' identified by '123456';

在客户端 50 连接 vip 地址访问数据库服务

[root@mysql50 ~]#: mysql -uyaya9 -p123456 -h192.168.4.100
mysql> insert into db9.t1 values(100),(200),(300);

所有从库都能同步到数据

停止 51 数据库服务，50 主机依然可以访问到数据

[root@mysql51 ~]#: systemctl stop mysqld

当 56 主机监控到 51 主机停止服务时，56 的管理服务会自杀，触发主配置文件的故障脚本，这里面定义了 vip．52 会升级为主库，其他的从库会变成 52 的从库．

[root@mysql50 ~]#: mysql -umonitor -p123456 -h192.168.4.54 -e "show slave status\G" | grep -i 52
Master_Host: 192.168.4.52
Master_Log_File: mysql52.000002
Relay_Master_Log_File: mysql52.000002

可以发现已经修改了主库的信息,再查看 56 的 app1.cnf 配置，server1 配置已经被删除

4.修复损坏的数据库,添加入集群

mysql52
mysql> show master status;

File	Position	Binlog_Do_DB	Binlog_Ignore_DB	Executed_Gtid_Set
mysql52.000002	440			

当 51 宕机中新产生的数据需要手动同步到 51 中．mysqldump 可以根据偏移量备份

mysql51
[root@mysql51 ~]#: systemctl start mysqld
mysql> change master to master_host="192.168.4.52", master_user="rpluser", master_password="123456", master_log_file="mysql52.000002", master_log_pos=440;

```
    mysql> start slave;
    mysql56
    [root@mysql56 ~]#: vim /etc/mha-manager/app1.cnf
    ...
    [server1]
    hostname=192.168.4.51
    candidate_master=1
    ...
    [root@mysql56 ~]#: masterha_check_repl --conf=/etc/mha-manager/app1.cnf
    [root@mysql56      ~]#:      masterha_manager      --conf=/etc/mha-manager/app1.cnf      --
remove_dead_master_conf --ignore_last_failover
    [root@mysql56 ~]#: masterha_check_status --conf=/etc/mha-manager/app1.cnf
    app1 (pid:17478) is running(0:PING_OK), master:192.168.4.52
```

Day09 MySQL 视图以及存储过程

案例一:MySQL 视图

0.准备表 mysql55

使用数据导入,把/etc/passwd 文件的内容存储到 db9 库下的 user 表里

```
    [root@mysql55 ~]#: cp /etc/passwd /var/lib/mysql-files/
    mysql> create table user(name char(50),password char(50),uid smallint,gid smallint,comment
char(150),homedir char(150),shell char(50));
    mysql> load data infile "/var/lib/mysql-files/passwd" into table user fields terminated by
":" lines terminated by "\n";
    mysql> alter table user add id smallint primary key auto_increment first;
```

1.什么是视图

虚拟表

内容与真是的表相似,有字段有记录

视图并不在数据库中以存储的数据形式存在

行和列的数据来自定义视图时查询所引用的基表,并且在具体引用视图时动态生成

更新视图的数据, 就是更新基表的数据

更新基表数据,视图的数据也会跟着改变

优点

简单:

用户不需要关心视图中的数据如何查询获得

视图中的数据已经时过滤好的符合条件的结果集

安全:

用户只能看到视图中的数据

数据独立:

一旦视图结构确定,可以屏蔽表结构对用户的影响

2.视图使用规则

不能在视图上创建索引

在视图的 FTOM 子句中不能使用子查询

一下情形中的视图是不可更新的

包含一下关键字的 SQL 语句:聚合函数(SUM,MIN,MAX,COUNT 等),DISTINCT,GROUP BY,HAVING,UNION 或 UNION ALL

常量视图, JOIN,FROM 一个不能更新的视图

WHERE 子句的子查询应用了 FROM 子句中的表

使用了临时表

3.视图管理

创建视图语法格式

```
   create view 视图名称 as SQL 查询;
mysql> create view db9.v1 as select name,uid from user;
mysql> grant select,insert,update,delete on db9.v1 to yaya10@'%' identified by "123456";
   create view 视图名称(字段名列表) as SQL 查询;
mysql> create view db9.v2(a,b,c) as select name,uid,shell from db9.user;
mysql> desc v2;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | char(50)    | YES  |     | NULL    |       |
| b     | smallint(6) | YES  |     | NULL    |       |
| c     | char(50)    | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

提示:在视图表中不定义字段名的话,默认使用基表的字段名,若定义字段名,是图标中的字段必须和基表的字段个数相等

查看视图

查看当前库下所有表的状态信息

```
show table status;
show tables status where comment="view"\G;
```

验证视图特点

```
mysql -uyaya10 -p123456
```

查询记录

```
    select 字段名列表 from 视图名 where 条件;
```

插入记录

```
    Insert into 视图名(字段名列表) values(字段值列表);
```

更新记录

```
    update 视图名 set 字段名=值 where 条件
```

删除记录

```
    Delete from 视图名 where 条件
```

更新视图数据,就是更新基表的数据

更新基表数据,视图的数据也会跟着改变

删除视图

```
drop view 视图名称;
mysql> drop view v1;
```

4.视图进阶

创建视图的完全格式

```
create [or replace] [algorithm={undefined|merge|temptable}] [definer={user|current_user}] [SQL security {definer|invoker}] view view_name [(column_list)] AS select_statement [WITH [cascaded|local] check option]
```

设置指点别名

create view 视图名 as select 表别名.源字段名 as 字段别名 from 源表名 表别名 left join 源表名 表别名 on 条件;

```
mysql> create table t2 select name,uid,shell from user limit 3;
mysql> create table t3 select name,uid,password,homedir from user limit 5;
mysql> create view v4 as select * from t2 left join t3 on t2.uid=t3.uid;
ERROR 1060 (42S21): Duplicate column name 'name'
create view v2 as select a.name as aname, b.name as bname, a.uid as auid, b.uid as buid from user a left join info b on a.uid=b.uid;
```

as 可省略

user 别名是 a,info 别名是 b

```
create view v4 as select a.name x1, a.uid x2, a.shell x3, b.name x4, b.uid x5, b.password x6, b.homedir x7 from t2 a left join t3 b on a.uid=b.uid;
mysql> select * from v4;
+--------+------+--------------+--------+------+------+-------+
| x1     | x2   | x3           | x4     | x5   | x6   | x7    |
+--------+------+--------------+--------+------+------+-------+
| root   |    0 | /bin/bash    | root   |    0 | x    | /root |
| bin    |    1 | /sbin/nologin | bin    |    1 | x    | /bin  |
| daemon |    2 | /sbin/nologin | daemon |    2 | x    | /sbin |
+--------+------+--------------+--------+------+------+-------+
```

重要选线说明

OR replace 强制覆盖创建新视图

create or replace view 视图名 as select 查询;

创建时，若视图已存在,会替换已有的视图

```
mysql> create view v4 as select uid,gid,name from user;
ERROR 1050 (42S01): Table 'v4' already exists       //提示以存在
mysql> create or replace view v4 as select uid,gid,name from user;
```

(ALGOTIRHM)算法，访问视图时，mysql 服务的处理方法

```
mysql> show create view v4\G;
*************************** 1. row ***************************
                View: v4
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `v4` AS select `user`.`uid` AS `uid`,`user`.`gid` AS `gid`,`user`.`name` AS `name` from `user`
character_set_client: utf8
collation_connection: utf8_general_ci
```

如果算法 undefined(未定义)，默认是 merge(替换)，另一种是 temptable(具体化方式)

```
mysql> create ALGORITHM=TEMPTABLE view v5 as select uid,gid,name from user;
select * from v5;
temptable
```

mysql 首先执行创建视图的查询(select uid,gid,name from user),再创建之后的查询(select * from v5)

merge

mysql 不会执行创建时的查询语句,而是直接执行(select * from v5)

with check option 限制视图操作

LOCAL 和 CASCADED 关键字觉点检查的范围

LOCAL 对视图操作时,必须满足视图自身的限制

```
mysql> create table t5 select name,uid,gid,shell from user where gid >=20 and gid<=1000;
mysql> create view v7 as select * from t5 where gid<=500 with local check option;
mysql> update v7 set gid=501 where name="games";
ERROR 1369 (HY000): CHECK OPTION failed 'db9.v7'
mysql> select * from v7 where name="games";
mysql> select * from t5 where name="games";
```

CASCADED 对视图操作时,机要满足视图自身的限制,又要满足基本的限制

```
mysql> create view v8 as select * from v7 where gid >=100 with cascaded check option;
mysql> select * from v8;
+-----------------+------+------+---------------+
| name            | uid  | gid  | shell         |
+-----------------+------+------+---------------+
| games           | 6000 |  100 | /sbin/nologin |
| systemd-network | 6000 |  192 | /sbin/nologin |
+-----------------+------+------+---------------+
mysql> select view v9 as select user where gid<=100 wich cascaded check option;
mysql> update v9 set gid=101 where name="root";
ERROR 1369 (HY000): CHECK OPTION failed 'db9.v9'
```

首先满足自身的限制

```
mysql> create view v10 as select * from t5 where gid>=100 with cascaded check option;
mysql> update v10 set gid=1100 where name="games";
Query OK, 1 row affected (0.00 sec)
```

按照必须满足基表的条件，应该是不能更新成功啊，需要继续斟酌.

```
mysql> create view v11 as select * from v7 where gid>30 with cascaded check option;
mysql> update v11 set gid=501 where name="nobody";
ERROR 1369 (HY000): CHECK OPTION failed 'db9.v11'
```

案例二:存储过程(开发并不建议使用,因为调试功能不太好使)

1.存储过程介绍及优点

相当于 MySQL 语句组成的脚本

指的是数据库中保存的一系列 SQL 命令的集合

可以在存储过程中使用变量、条件判断、流程控制等

优点:

提高性能

可减轻网络负担

可以防止对表的直接访问

避免重复编写 SQL 操作

2.存储过程基本使用

1)创建存储过程

```
delimiter //                        //修改命令结束符
create procedure 库名.名称()
begin
```

```
   功能代码
   select * from db9.user;
end
//
delimiter ;                          //修改回默认的
```

提示:若没有指定分隔符,编译器会把存储过程当成 SQL 语句进行处理,从而执行出错

2)执行存储过程

```
call 库名.名称();
```

提示:存储过程没有参数时,()可以省略; 存储过程有参数时,调用时必须传给参数

3)查看存储过程

查看服务器上已有存储过程

```
mysql> show procedure status\G;
mysql> select db,name from mysql.proc where type="PROCEDURE";
+-----+--------------------------------------+
| db  | name                                 |
+-----+--------------------------------------+
| sys | create_synonym_db                    |
| sys | diagnostics                          |
| sys | execute_prepared_stmt                |
| sys | ps_setup_disable_background_threads  |
| sys | ps_setup_disable_consumer            |
| sys | ps_setup_disable_instrument          |
| sys | ps_setup_disable_thread              |
| sys | ps_setup_enable_background_threads   |
| sys | ps_setup_enable_consumer             |
| sys | ps_setup_enable_instrument           |
| sys | ps_setup_enable_thread               |
| sys | ps_setup_reload_saved                |
| sys | ps_setup_reset_to_default            |
| sys | ps_setup_save                        |
| sys | ps_setup_show_disabled               |
| sys | ps_setup_show_disabled_consumers     |
| sys | ps_setup_show_disabled_instruments   |
| sys | ps_setup_show_enabled                |
| sys | ps_setup_show_enabled_consumers      |
| sys | ps_setup_show_enabled_instruments    |
| sys | ps_statement_avg_latency_histogram   |
| sys | ps_trace_statement_digest            |
| sys | ps_trace_thread                      |
| sys | ps_truncate_all_tables               |
| sys | statement_performance_analyzer       |
| sys | table_exists                         |
+-----+--------------------------------------+
```

查看存储过程代码

```
mysql> select db,name,body from mysql.proc where type="PROCEDURE" and name="存储过程名"\G;
```

4)删除

```
drop procedure 库名.名称;
```

举例:

```
mysql> delimiter //
mysql> create procedure db9.p1()
```

```
    -> begin
    -> select count(*) from db9.user where shell!="/bin/bash";
    -> select count(*) from db9.user where shell="/bin/bash";
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p1();
+----------+
| count(*) |
+----------+
|       20 |
+----------+
1 row in set (0.00 sec)

+----------+
| count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
mysql> select db,name from mysql.proc where type="procedure" and name="p1";
+-----+------+
| db  | name |
+-----+------+
| db9 | p1   |
+-----+------+
mysql> select body from mysql.proc where type="procedure" and name="p1"\G;
*************************** 1. row ***************************
body: begin
select count(*) from db9.user where shell!="/bin/bash";
select count(*) from db9.user where shell="/bin/bash";
end
```

3.存储过程进阶使用

变量类型:

名称	描述
会话变量	会话变量和全局变量叫系统变量，使用 set 命名定义
全局变量	全局变量的修改会影响到整个服务器,但是对绘画变量的修改,只会影响到当前的会话 select @@hostname
用户变量	在客户端链接到数据库服务的整个过程中都是有效的.当前连接断开后所有用户比那辆失效. 定义 set @变量名=值 输出 select @变量名;
局部变量	存储过程中的 begin/end. 其有效范围仅限于该语句块中,语句块执行完毕后,变量失效 declare 专门用来定义局部变量.需要在 begin ... end 之间用

注意:局部变量和参数变量调用时变量名前不需要加@

举例:

```
mysql> show session variables;          //查看会话变量
mysql> set session sort_buffer_size=40000;
mysql> show global variables;           //查看全局变量
mysql> show global variables like "%hostname%";
```

```
mysql> set @x=99;                          //用户变量
mysql> set @name="bob";
mysql> select @x, @name;
+------+-------+
| @x   | @name |
+------+-------+
|   99 | bob   |
+------+-------+
mysql> delimiter //
mysql> create procedure db9.p2()
    -> begin
    -> declare x int;                      //局部变量
    -> declare y int;
    -> set x=99;
    -> set y=11;
    -> select x,y;
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p2;
```

使用查询结果给变量赋值

```
mysql> select count(name) into @x from db9.user;     //使用 sql 命令你个查询结果复制
mysql> select @x;
+------+
| @x   |
+------+
|   21 |
+------+
mysql> delimiter //
mysql> create procedure db9.p3()
    -> begin
    -> declare x int;
    -> declare y int;
    -> select count(name) into x from db9.user where gid<=1000;
    -> select count(name) into y from db9.user where gid>1000;
    -> select x,y;
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p3();
+------+------+
| x    | y    |
+------+------+
|   21 |    0 |
+------+------+
```

参数类型

```
create procedure 名称(
类型      参数名        数据类型,
类型      参数名        数据类型,
)
```

提示:调用参数时,名称前也不需要加@

关键字	名称	描述
in	输入参数	作用是给存储过程传值,必须在调用存储过程时赋值，在存储过程中该参数的值不允许修改;默认类型 in
out	输出参数	该值可在存储过程内部被改变,并可返回
inout	输入/输出参数	调用时指定,并且可被改变和返回

in 举例

```
mysql> delimiter //
mysql> create procedure db9.p9(in username char(10))       //username 可以随便定义
    -> begin
    -> select * from db9.user where name=username;
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p9("root");
+----+------+----------+------+------+---------+---------+-----------+
| id | name | password | uid  | gid  | comment | homedir | shell     |
+----+------+----------+------+------+---------+---------+-----------+
|  1 | root | x        | 6000 |    0 | root    | /root   | /bin/bash |
+----+------+----------+------+------+---------+---------+-----------+
mysql> call db9.p9("adm");
+----+------+----------+------+------+---------+----------+---------------+
| id | name | password | uid  | gid  | comment | homedir  | shell         |
+----+------+----------+------+------+---------+----------+---------------+
|  4 | adm  | x        | 6000 |    4 | adm     | /var/adm | /sbin/nologin |
+----+------+----------+------+------+---------+----------+---------------+
mysql> call db9.p9();
ERROR 1318 (42000): Incorrect number of arguments for PROCEDURE db9.p9; expected 1, got 0
```

out 举例

```
mysql> delimiter //
mysql> create procedure db9.p10(in x int, in y int, out z int)
    -> begin
    -> set z=x+y;
    -> end
    -> //
mysql> delimiter ;
mysql> select @i;
+------+
| @i   |
+------+
| NULL |
+------+
mysql> call db9.p10(19,37,@i);
mysql> select @i;
+------+
| @i   |
+------+
|   56 |
+------+
mysql> delimiter //
```

```
mysql> create procedure db9.p11(in uid_num int, in shell_name char(50), out x int)
    -> begin
    -> declare i int;
    -> declare j int;
    -> select count(uid) into i from db9.user where uid <=uid_num;
    -> select count(shell) into j from db9.user where shell=shell_name;
    -> set x=i+j;
    -> select x;
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p11(6,"/bin/bash",@w);
+------+
| x    |
+------+
|    1 |
+------+
mysql> select @w;
+------+
| @w   |
+------+
|    1 |
+------+
```

inout 举例

```
mysql> delimiter //
mysql> create procedure db9.p12(inout x char(30))
    -> begin
    -> select name from db9.user where name=x;
    -> select count(*) into x from db9.user;
    -> select x;
    -> end
    -> //
mysql> delimiter ;
mysql> set @i="jerry";
mysql> call db9.p12(@i);
Empty set (0.00 sec)

+------+
| x    |
+------+
| 21   |
+------+
mysql> set @i="root";
mysql> call db9.p12(@i);
+------+
| name |
+------+
| root |
+------+
+------+
| x    |
+------+
| 21   |
+------+
```

4.四则运算

符号	描述	示例
+	加法运算	SET @var1=2+2; 4
-	减法	SET @var2=3-1; 1
*	乘法运算	SET @var3=3*2; 6
/	除法运算	SET @var4=10/3; 3.33333
DIV	整除运算	SET @var5=10 DIV 3; 3
%	取模	SET @var6=10%3; 1

举例:

```
mysql> set @x=7, @y=8;
mysql> set @z=@x+@y;
mysql> select @z;
+------+
| @z   |
+------+
|   15 |
+------+
mysql> delimiter //
mysql> create procedure db9.p4()
    -> begin
    -> declare x int;
    -> declare y int;
    -> declare z int;
    -> select count(name) into x from db9.user where gid<=1000;
    -> select count(name) into y from db9.user where gid>1000;
    -> set z=x+y;
    -> select x,y,z;
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p4;
+------+------+------+
| x    | y    | z    |
+------+------+------+
|   21 |    0 |   21 |
+------+------+------+
```

5.条件判断(是给流程控制使用)

数值比较

类型	用途
=	等于
>,>=	大于、大于或等于
<,<=	小于、小于或等于
!=	不等于
BETWEEN..AND	在..与..之间

逻辑比较、范围、空、非空、模糊、正则

类型	用途
OR,AND,!	逻辑或、逻辑与、逻辑非
IN...NOT IN	在..范围内、不在..范围内
IS NULL	字段的值为空
IS NOT NULL	字段的值不为空
LIKE	模糊匹配
REGEXP	正则匹配

选择结构

单分支

```
if 条件测试 then
    代码 ..
end if;
```

双分支

```
if 条件测试 then
    代码 1..
else
    代码 2..
end if;
```

举例:

```
mysql> delimiter //
mysql> create procedure db9.p13(in line_num int)
    -> begin
    -> if line_num > 10 then
    ->    select * from db9.user where id > 10;
    -> else
    ->    select * from db9.user where id <=10;
    -> end if;
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p13(2);
```

循环结构

```
while 条件 do
    循环体
   ..
end while;
```

举例:

```
mysql> create procedure db9.p15()
    -> begin
    -> declare x int;
    -> set x=1;
    -> while x<=10 do
    ->   select x;
    ->   set x=x+1;
    -> end while;
```

```
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p15;
```

loop 死循环： 无条件、反复执行某一段代码

```
loop
    循环体
   ..
end loop
```

举例:

```
mysql> delimiter //
mysql> create procedure db9.p16()
    -> begin
    -> loop
    ->   select * from db9.user limit 1;
    -> end loop;
    -> end
    -> //
mysql> delimiter ;
```

repeat 条件式循环：当条件成立时结束循环

```
repeat
    循环体
    ..
    until 条件判断
end repeat;
```

举例:

```
mysql> delimiter //
mysql> create procedure db9.p18()
    -> begin
    -> declare x int;
    -> set x=10;
    -> repeat
    ->   select x;
    ->   set x=x+1;
    ->   until x > 20;
    -> end repeat;
    -> end
    -> //
mysql> delimiter ;
```

控制循环的执行

```
   LEAVE 标签名              //跳出循环
   ITERATE 标签 名           //放弃本次循环，执行下一次循环
```

举例:

```
mysql> delimiter //
mysql> create procedure db9.p17()
    -> begin
    -> declare i int;
    -> set i=0;
```

```
    -> loab1:loop
    ->   set i=i+1;
    ->   if i=3 then
    ->     iterate loab1;
    ->   end if;
    ->   if i=7 then
    ->     leave loab1;
    ->   end if;
    ->   select i;
    -> end loop;
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p17;
+------+
| i    |
+------+
|    1 |
+------+
1 row in set (0.00 sec)

+------+
| i    |
+------+
|    2 |
+------+
1 row in set (0.00 sec)

+------+
| i    |
+------+
|    4 |
+------+
1 row in set (0.00 sec)

+------+
| i    |
+------+
|    5 |
+------+
1 row in set (0.00 sec)

+------+
| i    |
+------+
|    6 |
+------+
1 row in set (0.00 sec)
```

练习：

满足以下要求

定义名称为 p18 的存储过程

用户可以自定义显示 user 表记录的行数

若调用时用户没有输入行数，默认显示第 1 条记录

```
mysql> delimiter //
mysql> create procedure db9.p18(in line_num int)
    -> begin
    -> if line_num is not null then
    ->   select * from db9.user limit line_num;
    -> else
    ->   select * from db9.user limit 1;
    -> end if;
    -> end
    -> //
mysql> delimiter ;
mysql> call db9.p18(2);
mysql> set@x=null;
mysql> db9.p18(@x);
```

Day10 mycat 分库分表

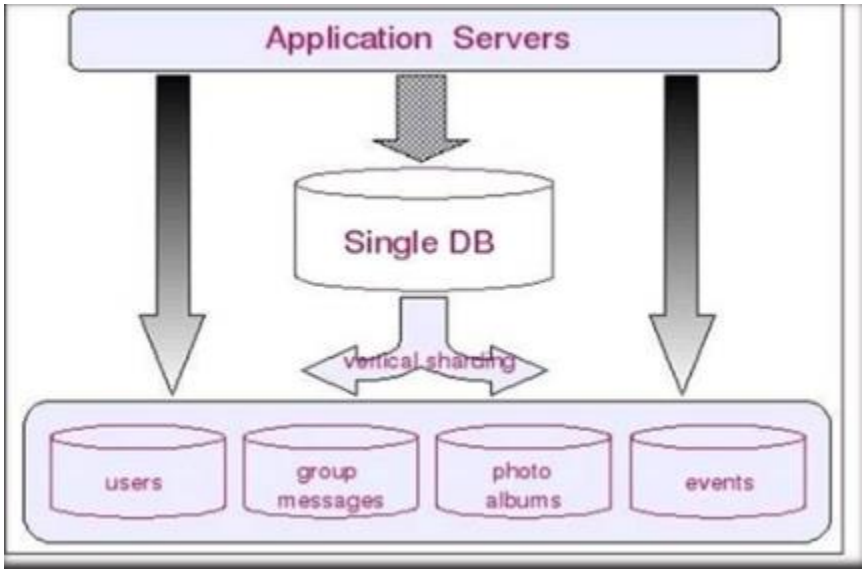
什么是分库分表

将存放在一个数据（主机）中的数据，按照特定方式进行拆分，分散存放到多个数据库（主机）中，以达到分散单台设备负载的效果

垂直分割（纵向切分）

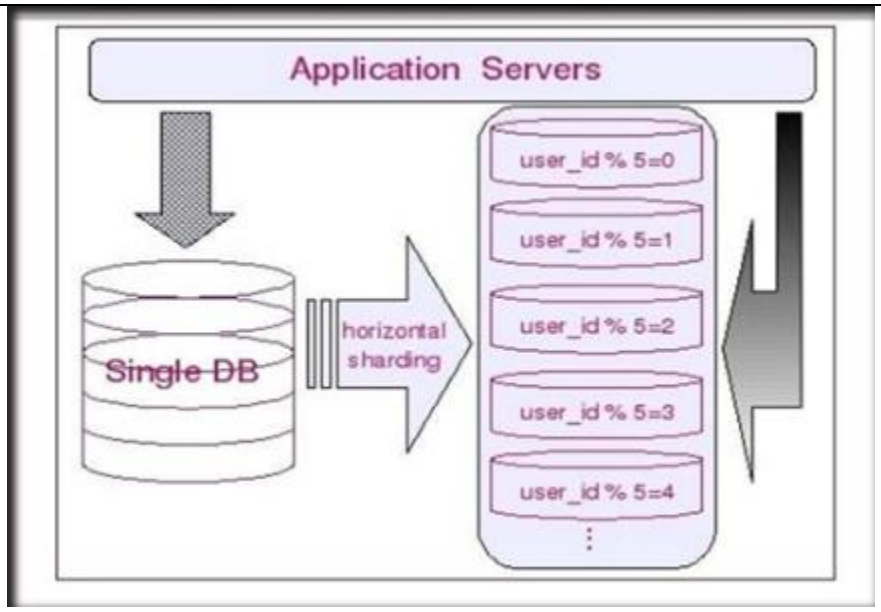
将单个表，拆分成多个表，分散道不同的数据库

将单个数据库的多个表进行分类，按业务类别分散到不同的数据库上



水平分割（横向切分）

按照表中某个字段的某种规则，把表中的许多记录按行切分，分享到多个数据库中



mycat 介绍

mycat 是基于 java 的分布式数据库系统中间层，为高并发环境的分布式访问特工解决方案

支持 JDBC 形式连接

支持 MySQL、Oracle、Sqlserver、Mongodb 等

提供数据读写分离服务

可以实现数据库服务器的高可用

提供数据分片服务

基于阿里巴巴 cobar 进行研发的开源软件

适合数据大量写入数据的存储需求

提示：此结构适合大量写入的存储需求，并不适合大量读请求

mycat 分片规则

支持 10 种分片规则

1.枚举法	sharding-by-intfile
2.固定分片	rule1
3.范围约定	auto-sharding-long
4.求模法	mod-long
5.日期列分区法	sharding-by-date
6.通配取模	sharding-by-pattern
7.ASCII 码求模通配	sharding-by-prefixpattern
8.编程指定	sharding-by-substring
9.字符串拆分 hash 解析	sharding-by-stringhash
10.一致性 hash	sharding-by-murmur

工作原理

当 mycat 收到 SQL 查询时，先解析这个 SQL 查询涉及到的表，然后看此表的定义，如果有分片规则，则获取 SQL

里分片字段的值，并匹配分片汉书，获得分片列表，然后将 SQL 发往这些分片去执行，最后手机和处理所有分片结果数据，并返回到客户端

案例一：配置数据分片

IP 规划

拓扑名称	主机名	角色	数据库名	ip 地址
hostA	mysql50	客户端	无	192.168.4.50/24
hostB	mycat	mycat 服务器	无	192.168.4.56/24
hostC	mysql53	数据库服务器	db1	192.168.4.53/24
hostD	mysql54	数据库服务器	db2	192.168.4.54/24
hostE	mysql55	数据库服务器	db3	192.168.4.55/24

用重置脚本重置 mysql 服务

1.配置 mycat 服务器 –192.168.4.56

```
[root@mysql56 ~]#: yum -y install java-1.8.0-openjdk
[root@mysql56 ~]#: tar xf Mycat-server-1.6-RELEASE-20161028204710-linux.tar.gz
[root@mysql56 ~]#: mv mycat/ /usr/local/
```

目录结构

```
[root@mysql56 mycat]#: ls
bin  catlet  conf  lib  logs  version.txt
```

bin.　　//可执行命令（mycat）

catlet　　//mycat 和其他软件相应用时安装的程序

conf　　//配置文件

lib　　//mycat 使用的 jar

log　　//mycat 启动日志和运行日志

wrapper.log　　//mycat 服务启动日志

mycat.log　　//记录 SQL 脚本执行后的报错内容

重要配置文件说明

server.xml　　//设置连 mycat 的帐号信息　　扩展标记语言

schema.xml　　//配置数据分片

rule.xml　　//定义 mycat 分片规则

配置标签说明

<user> .. ..</user>　　//定义连 mycat 用户信息

<datanode>.. ..</datanode>　　//指定数据节点

<datahost>.. ..</datahost>　　//指定数据库地址及用户信息

修改配置文件

```
[root@mysql56 mycat]#: vim /usr/local/mycat/conf/server.xml
80     <user name="root">                                          //连接用户名
```

```
81          <property name="password">123456</property>     //连接密码
82          <property name="schemas">TESTDB</property>            //56 上的逻辑库(虚拟库)
..
95      <user name="user">
96          <property name="password">user</property>
97          <property name="schemas">TESTDB</property>
98          <property name="readOnly">true</property>       //只读
[root@mysql56 mycat]#: vim /usr/local/mycat/conf/schema.xml
[root@mysql56 mycat]#: sed -i '56,77d;39,42d;16,18d' /usr/local/mycat/conf/schema.xml
11-18 行之间缺少 dn3 的添上
33      <dataNode name="dn1" dataHost="mysql53" database="db1" />
34      <dataNode name="dn2" dataHost="mysql54" database="db2" />
35      <dataNode name="dn3" dataHost="mysql55" database="db3" />
#定义分片使用的库，所在的物理主机，真正存储数据的 db1 库在物理主机 mysql53
36      <dataHost name="mysql53" maxCon="1000" minCon="10" balance="0"
37                          writeType="0"  dbType="mysql"  dbDriver="native"  switchType="1"
slaveThreshold="100">
38          <heartbeat>select user()</heartbeat>
39                      <writeHost  host="hostM1"  url="192.168.4.53:3306"  user="mycat_user"
password="123456"/>
40      </dataHost>
41      <dataHost name="mysql54" maxCon="1000" minCon="10" balance="0"
42                          writeType="0"  dbType="mysql"  dbDriver="native"  switchType="1"
slaveThreshold="100">
43          <heartbeat>select user()</heartbeat>
44                      <writeHost  host="hostM2"  url="192.168.4.54:3306"  user="mycat_user"
password="123456"/>
45      </dataHost>
46      <dataHost name="mysql55" maxCon="1000" minCon="10" balance="0"
47                          writeType="0"  dbType="mysql"  dbDriver="native"  switchType="1"
slaveThreshold="100">
48          <heartbeat>select user()</heartbeat>
49                      <writeHost  host="hostM3"  url="192.168.4.55:3306"  user="mycat_user"
password="123456"/>
50      </dataHost>
#指定 mysql53 名称对应的 ip 地址，访问数据时 mycat 服务连接数据库服务器时使用的用户名和密码

配置数据库服务器(192.168.4.53-55)
创建存储数据的库
[root@GYP-HOME ~]# ssh mysql53 'mysql -uroot -p123456 -e "create database db1"'
[root@GYP-HOME ~]# ssh mysql54 'mysql -uroot -p123456 -e "create database db2"'
[root@GYP-HOME ~]# ssh mysql55 'mysql -uroot -p123456 -e "create database db3"'
授权连接用户
[root@GYP-HOME ~]# for i in mysql{53..55}
                    > do
                   > ssh $i "mysql -uroot -p123456 -e \"grant all on *.* to mycat_user@'%'
identified by '123456'\""
                    > done
[root@mysql56 mycat]#: ./bin/mycat start
Starting Mycat-server...
[root@mysql56 mycat]#: ss -antpu | grep 8066
tcp       LISTEN         0           100          :::8066                                 :::*
```

```
users:(("java",pid=13076,fd=76))
```

补充:

maxscale 做从库的轮询查找

MySQL 性能调优拓展