[mysql数据库服务器基本操作 2](#_Toc953528)

[mysql数据类型 3](#_Toc953529)

[管理数据库的sql (删库,加库) 4](#_Toc953530)

[管理表的 sql命令(删表,加表) 4](#_Toc953531)

[修改表结构 4](#_Toc953532)

[管理记录的sql命令(增删改查记录) 5](#_Toc953533)

[增 5](#_Toc953534)

[查 5](#_Toc953535)

[-------修改记录字段的值 6](#_Toc953536)

[-------删除记录 6](#_Toc953537)

[匹配条件（查看 修改 删除记录时可以加条件） 6](#_Toc953538)

[数值比较 字段名 符号 数字 6](#_Toc953539)

[字符比较 字段名 符号 “字符串” 6](#_Toc953540)

[范围内比较 /去重显示 6](#_Toc953541)

[逻辑比较（就是有个查询条件） 优先判断逻辑与 6](#_Toc953542)

[匹配空 字段名 is null 7](#_Toc953543)

[模糊匹配 7](#_Toc953544)

[正则匹配 7](#_Toc953545)

[四则运算(select 和 update 操作是可以做数学计算) 7](#_Toc953546)

[聚集函数（字段的类型要求是数值类型count,max,avg等） 8](#_Toc953547)

[查询不显示字段重复值 8](#_Toc953548)

[查询分组 8](#_Toc953549)

[查询排序 (按照数值类型的字段排队) 8](#_Toc953550)

[限制查询显示行数(默认显示所有查询的记录) 8](#_Toc953551)

[多表查询 9](#_Toc953552)

[字段约束条件 9](#_Toc953553)

[mysql键值 10](#_Toc953554)

[普通索引的使用（index） 10](#_Toc953555)

[主键 primary key 10](#_Toc953556)

[外 键(作用 限制如何给字段赋值的) 11](#_Toc953557)

[mysql存储引擎 11](#_Toc953558)

[数据导入导出（批量操作数据） 14](#_Toc953559)

[管理员密码管理 15](#_Toc953560)

[恢复数据库管理员本机登录密码 15](#_Toc953561)

[用户授权(grant) 16](#_Toc953562)

[撤销用户权限命令的语法格式 17](#_Toc953563)

[安装数据库图形管理工具 phpmyadmin 17](#_Toc953564)

[DBA 数据备份和恢复 18](#_Toc953565)

[物理备份与恢复（授权库备份） 18](#_Toc953566)

[逻辑备份 (mysqldump,mysql+binlog) 19](#_Toc953567)

[innobackupex 备份 / 恢复 23](#_Toc953568)

[部署mysql主从同步结构实现数据自动备份 29](#_Toc953569)

[lvm2 snapshot +binlog 29](#_Toc953570)

[视图 29](#_Toc953571)

[存储过程 34](#_Toc953572)

[分库分表 43](#_Toc953573)

[mysql集群 51](#_Toc953574)

[mysql主从同步 51](#_Toc953575)

[主从同步原理 55](#_Toc953576)

[MySQL 读写分离(maxscale/mycat) 57](#_Toc953577)

[mysql多实例 61](#_Toc953578)

[MHA 集群 64](#_Toc953579)

[准备集群环境 65](#_Toc953580)

[配置 MHA 集群 67](#_Toc953581)

[MySQL 性能调优 71](#_Toc953582)

[并发及连接控制 73](#_Toc953583)

[缓存参数控制 73](#_Toc953584)

[SQL 查询优化 73](#_Toc953585)

[慢查询日志 74](#_Toc953586)

[存储数据的网络结构部署有数据传输瓶颈； 76](#_Toc953587)

[数据库安全 76](#_Toc953588)

数据库常识

提供数据库服务的软件有那些？

开源软件 mysql 、 mongodb 、 redis

商业软件 oracle 、 db2 、 SQL SERVER

关系型数据型软件： 要按照一定组织结构存储数据，并且数据和

非关系型数据库软件（NoSQL）mongodb 、 redis 、 memcached

key = 值

微信名 消息内容

# mysql数据库服务器基本操作

不能同时安装mysql /mariadb

查看服务进程 mysqld

和端口号: 3306

服务的主配置文件 /etc/my.cnf

数据目录 /var/lib/mysql

日志文件 /var/log/mysqld.log

修改初始密码:

#mysql [-h数据库服务器] ip地址 -u用户名 -p'密码'

mysql> alter user root@"localhost" identified by "密码"；//重置本机连接密码

mysql>set global validate\_password\_policy=0; 只检查密码的长度

mysql>set global validate\_password\_length=6; 密码长度不能小于6个字符

永久生效，在/etc/my.cnf 修改配置

[mysqld]

validate\_password\_policy=0

validate\_password\_length=6

把数据存储到数据库服务器上的过程？

连接数据库服务器（命令行 API 图形工具）

选择库 （存放数据的文件夹）

选择表 （就是文件）

插入记录 （文件中的行）

断开连接

## mysql数据类型

**数值类型** （成绩 年龄 工资 ）每种类型的存储数据的范围都是固定

整数类型 （只能存储整数）

微小整型 小整型 中整型 大整型 极大整型

Tinyint smallint MEDIUMINT INT bigint

\*\*\*\*\*unsigned 使用数值类型有符号的范围。

浮点型 （存储小数）

float(M,N)

double(M,N) M 设置总位数 N 设置小数位位数

正数.小数 总位数 整数位 小数位

18088.88 7 5 2

数值类型的宽度 是显示宽度，不能够决定给字段赋值的大小。字段的值由类型决定。

**字符类型** （商品名称 籍贯 姓名 生产厂家）

char (255) 固定长度字符类型

varchar (65532) 变长字符类型

大文本类型 （音频文件 视频文件 图片文件）blob / text

**日期时间类型** （注册时间 约会时间 开会时间 入职时间 生日）

年 year YYYY 2018

日期 date YYYYMMDD 20180423

时间 time HHMMSS 161958

日期时间 datetime/timestamp

YYYYMMDDHHMMSS 20180423161958

获取日期时间给对应的日期时间类型的字段赋值

获取日期时间函数

now() 获取当期系统的时间

year(日期时间)获取指定时间中的年

month(日期时间)获取指定时间中的月

date(日期时间)获取指定时间中的日期

day(日期时间)获取指定时间中的号（天）

time(日期时间)获取指定时间中的时间

可以使用2位数字给year类型的字段赋值，规律如下：

01-69 20XX

70-99 19XX

datetime与timestamp 的区别？

**枚举类型**（插入记录 时 记录的值 在列举的范围内选择）性别 爱好 专业

enum(值列表) 单选

set(值列表) 多选

通用命令

select @@hostname //查看主机名

user() //当前用户

show processlist 当前活跃线程,会显示query mysqldump等

## 管理数据库的sql (删库,加库)

查看 show databases;

创建 create database 库名；

切换 use 库名；

删除 drop database 库名；

显示当前所在的库 select database();

## 管理表的 sql命令(删表,加表)

查看 show tables;

建表

create table 库名.表名（

字段名 类型（宽度） 约束条件，

字段名 类型（宽度） 约束条件，

.....

）;

删除 drop table 库名.表名 ;

查看表结构 desc 库名.表名 ;

### 修改表结构

mysql> alter table 库.表 执行动作 ；

**执行动作:**

**添加新字段**  add 字段名 类型(宽度) [ 约束条件] [first/after 字段]

**删除已有字段**  drop 字段名

**修改已有字段的类型宽度及约束条件** modify 字段名 类型(宽度) [ 约束条件] [first/after字段]

\*\*\*划重点\*\*\*\* //不能和已存在的数据冲突，否则不允许修改。 first/ after 字段可以修改字段的位置

**修改字段名** change 源字段名 新字段名 类型(宽度) [ 约束条件]

**修改表名**  rename [to] 新表名；

修改表引擎 alter table 表 engine=存储引擎名；

修改主键 alter table 表 add primary key(字段名,[字段名2，字段名3])；

alter table 表 drop primary key；

创建修改外键

## 管理记录的sql命令(增删改查记录)

查看 select \* from 库名.表名 ;

插入 insert into 库名.表名 values(字段值列表);

insert into gamedb.stuinfo values ("tom","beijing");

insert into gamedb.stuinfo values ("bob","beijing");

修改 update 库名.表名 set 字段名=值 where 条件；

update gamedb.stuinfo set addr="shanghai" where name="tom";

删除 delete from 库名.表名;

delete from gamedb.stuinfo;

### 增

• 注意事项

– 字段值要与字段类型相匹配

– 对于字符类型的字段,要用双或单引号括起来

– 依次给所有字段赋值时,字段名可以省略

– 只给一部分字段赋值时,必须明确写出对应的字段名称

插入记录 insert into （值要与字段类型和约束条件匹配）

插入1条记录给所有字段赋值

insert into 库.表 values(字段值列表);

插入N条记录给所有字段赋值

insert into 库.表 values(字段值列表)，(字段值列表);

插入1条记录给指定的字段赋值

insert into 库.表（字段名列表） values(字段值列表);

插入N条记录给指定的字段赋值

Insert into 库.表（字段名列表） values(字段值列表)，(字段值列表);

mysql> insert into user1 values

-> (43,"bob","x",1001,1001,"student","/home/bob","/bin/bash"),

-> (44,"lucy","x",1002,1002,"student","/home/lucy","/bin/bash")

-> ;

mysql> insert into user1(用户名,命令解释器) values ("haha","/bin/bash"), ("xixi","/sbin/nologin");

### 查

#命令格式

select 字段名列表 from 库.表 where 匹配条件；

#查看表中所有行的所有字段的值

select \* from 库.表 ；

# 查看表中所有行的指定字段的值

select 字段名1，字段名2 ，字段名n from 库.表 ；

#查看指定行的指定字段的值

select 字段名1，字段名2 ，字段名n from 库.表 where 匹配条件 ；

### -------修改记录字段的值

#修改所有记录指定字段的值

update 库.表 set 字段名=值，字段名=值；

#修改与条件匹配的记录指定字段的值

update 库.表 set 字段名=值，字段名=值 where 匹配条件 ；

### -------删除记录

#删除表中的所有行

delete from 库.表；

#仅删除与条件匹配的记录

delete from 库.表 where 匹配条件 ；

## 匹配条件（查看 修改 删除记录时可以加条件）

-------------查看 修改 删除 from 库.表 where 匹配条件 ；

### 数值比较 字段名 符号 数字

= != < <= > >=

select \* from usertab where id<=10;

### 字符比较 字段名 符号 “字符串”

= !=

select username,shell from usertab where shell!="/bin/bash";

### 范围内比较 /去重显示

字段名 between 数字1 and 数字2 在...之间...

字段名 in (值列表) 在...里

字段名 not in (值列表) 不在...里

去重显示 DISTINCT 字段名

select username,uid from usertab where username in ("root","rsync","mysql");

### 逻辑比较（就是有个查询条件） 优先判断逻辑与

逻辑与 and 多个条件同时成立 才匹配

逻辑或 or 多个条件,某一个条件成立 就匹配

逻辑非 ！ 或 not 取反

( ) 提高优先级

select username,uid from usertab where username="root" and uid=0 and shell="/bin/bash";

or 和 and同时有的时候，and优先，先判断and的整体

mysql> select \* from user1 where 用户id=3 or id=1 and 用户名="root";

先判断 id=1且用户名="root"

mysql> select \* from user1 where (用户id=3 or id=1) and 用户名="root";

先判断 用户id=3或id=1

### 匹配空 字段名 is null

匹配空 字段名 is not null

update usertab set username is null where id=2;

### 模糊匹配

字段名 like '表达式';

% 表示零个或多个字符

\_ 表任意一个字符

select username from usertab where username like '\_ \_ \_ \_'; //4个字符

select username from usertab where username like 'a\_ \_t'; //4个字符，a开头，t结尾

insert into usertab(username) values("a");

select username from usertab where username like 'a%'; //a开头，任意个字符

select username from usertab where username like '\_%\_'; //至少两个字符

select username from usertab where username like '%'; //不匹配null， 匹配“”

### 正则匹配

字段名 regexp '正则表达式';

^ $ . \* [ ]

select username from usertab where username regexp '^a.\*t$';

select username from usertab where username regexp '[0-9]'; //包含数字的

select username from usertab where username regexp '^[0-9]';

select username from usertab where username regexp '[0-9]$';

select username from usertab where username regexp 'a.\*t';

select username from usertab where username regexp '^a.\*t$';

select username,uid from usertab where uid regexp '..'; //2个字符以上的

select username,uid from usertab where uid regexp '^..$';

### 四则运算(select 和 update 操作是可以做数学计算)

字段类型必须数值类型（整型 或浮点型）

+ - \* / %

select username ,uid,gid, uid+gid as zh , (uid+gid)/2 as pjz from usertab where username="mysql"; //as可以省略

select id,username,uid from usertab where id <=10;

update usertab set uid=uid+1 where id <=10;

select username ,uid,gid from usertab where usernane="mysql";

select username ,uid,gid, uid+gid as zh from usertab where username="mysql";

select username ,uid,gid, uid+gid as zh , (uid+gid)/2 as pjz from usertab where username="mysql";

alter table usertab add age tinyint(2) unsigned default 21 after username;

mysql> select username,age from usertab;

select username , age , 2018-age s\_year from usertab where username="root";

### 聚集函数（字段的类型要求是数值类型count,max,avg等）

count(字段名）统计字段值的个数

sum(字段名） 求和

max(字段名） 输出字段值的最大值

min(字段名） 输出字段值的最小值

avg(字段名） 输出字段值的平均值

select count(username) from usertab where shell="/bin/bash";

select avg(uid) from usertab;

### 查询不显示字段重复值

distinct 字段名

select distinct shell from usertab where uid >10 and uid<=100;

### 查询分组

sql查询 group by 字段名；

select shell from usertab where uid >10 and uid<=100 group by shell;

### 查询排序 (按照数值类型的字段排队)

sql查询 order by 字段名 asc|desc;

select username,uid from usertab where uid >10 and uid<=100 order by uid; //升序

select username,uid from usertab where uid >10 and uid<=100 order by uid desc; //降序

### 限制查询显示行数(默认显示所有查询的记录)

sql查询 limit 数字； 显示查询结果的前几行

sql查询 limit 数字1，数字2； 显示查询结果指定范围的行

select username,uid from usertab where uid >10 and uid<=100 order by uid desc limit 1;

select username,uid from usertab where uid >10 and uid<=100 order by uid desc limit 2,3;

数据库目录出问题了，重置数据库目录

mysql\_install\_db --datadir=/var/lib/mysql --user=mysql

## 多表查询

1.1 复制表

作用？ 备份表 和 快速建表

命令格式？ create table 库.表 sql查询命令；

-------划重点

复制的内容由sql查询命令决定

不会复制源表字段的键值给新表

create table db3.user3 select username,uid,shell from db3.usertab limit 5;

1.2 where嵌套查询

定义： 把内层的查询结果作为外层查询的查询条件

命令格式？ select 字段名列表 from 库.表 条件 ( select 字段名列表 from 库.表 条件 );

select username,uid from db3.usertab where uid < (select avg(uid) from db3.usertab）；

select username from db3.usertab where username in (select user from mysql.user where host="localhost");

1.3多表查询

命令格式 select 字段名列表 from 表名列表 [ where 匹配条件]；

mysql> create table db4.t3

-> select username,uid,shell,homedir from db3.usertab

-> limit 3;

mysql> create table db4.t4

-> select username,uid,gid from db3.usertab limit 5;

select \* from t3,t4; 迪卡尔集 （一共3 \* 5 =15行）

select t3.username , t4.username from t3,t4 where t3.uid = t4.uid ;

1.4 连接查询

左连接查询 （当匹配条件成立时 以左表为主显示查询记录）显示A表全部记录，B表仅显示匹配的记录。若A表中有记录没有匹配B的记录，则联合表中B显示NULL

select 字段名列表 from 表A left join 表B on 匹配条件；

select \* from t6 right join t5 on t6.uid = t5.uid;

select \* from t6 left join t5 on t6.uid = t5.uid;

右连接查询（当匹配条件成立时 以右表为主显示查询记录）同上，AB互换

select 字段名列表 from 表A right join 表B on 匹配条件；

## 字段约束条件

作用： 限制如何给字段赋值的

包括内容有： NULL Key Default Extra

NULL 是否允许给字段赋null值

null 默认允许赋null值

not null 不允许赋null值

key 键值类型：普通索引 唯一索引 全文索引 主键 外 键

Default 默认值 作用：当不被字段赋值时，使用默认值给字段赋值

不设置默认值是 系统定义的是null

default 值

Extra 额外设置 ，字段是否设置为自动增加,默认没有自动增长功能

## mysql键值

设置在表中字段上的，作用是约束如何给字段赋值。同时会给字段做索引。

索引介绍： 树状目录结构 类似与书的“目录”

优点：加快查询表记录的速度

缺点 : 会减慢编辑表记录的速度，且占用磁盘的物理存储空间

(delete insert update)

键值：普通索引 唯一索引 全文索引 主键 外键

### 普通索引的使用（index）

使用规则?

显示 desc 表； key ----> MUL (这个表示是普通索引)

查看 desc 表名;

show index from 表名;

show index from 表名\G //纵向显示

Table: t2

Key\_name: aaa

Column\_name: age

Index\_type: BTREE (二叉树)

创建

在已有表创建

create index 索引名 on 表名（字段名）；

建表是时创建

create table 表名（

字段列表，

index(字段名)，

index(字段名)，

）;

删除

drop index 索引名 on 表名；

如果不建索引,数据库搜索会从头到尾搜索,占用内存

### 主键 primary key

（普通主键 复合主键 主键+auto\_increment）

创建复合主键的使用：多个字段一起做主键，插入记录时，只要做主键字段的值不同时重复，就可以插入记录。

主键primary key 通常和auto\_increment连用。让整型字段的值自动增长

一个表只能有一个主键

查看 desc 表； key ----> PRI

创建 在已有表创建

alter table 表 add primary key(字段名,[字段名2，字段名3])；

建表时创建

create table 表名（

字段列表，

primary key(字段名[,字段名2])

）；

create table 表名（

字段 类型（宽度） primary key，

......

）；

删除主键 mysql> alter table 表 drop primary key；

++++++++++++++++++++++++++++++++++++++

### 外 键(作用 限制如何给字段赋值的)

给当前表中字段赋值时，值只能在指定其他表的指定字段值的范围里选择。

使用外键的条件

– 表的存储引擎必须是 innodb

– 字段类型要一致

– 被参照字段必须要是索引类型的一种 (最好时primary key)

把外健同时设置为主键，防止空值，重复

**创建**：**建表时**

create table 表（

字段名列表，

foreign key(字段名) references 表名（字段名） on update cascade on delete cascade

）engine=innodb;

**在已有表创建**

mysql> alter table bjb add foreign key(字段名) references 表名

(被关联的字段) on update cascade on delete cascade;

查看 mysql> show create table 表名; 查看表中有没有外健

**删除外键**

alter table 表名 drop foreign key 外键名;

## mysql存储引擎

1. 存储引擎介绍:

是mysql数据库软件自带的功能程序，

每种存储引擎的功能和数据存储方式也不同

存储引擎就处理表的处理器

查看数据库服务支持的存储引擎有那些？

mysql> show engines; 或者 show engines\G

InnoDB DEFAULT

第三列conment 就是每个引擎的功能

2. 查看已有的表使用的存储引擎

show create table 表名；

3. 修改数据库服务默认使用的存储引擎

#vim /etc/my.cnf

[mysqld]

default-storage-engine=InnoDB

# systemctl restart mysqld

//只影响新建的表，不影响已经建好的表的引擎

4. 修改表使用的存储引擎，或建表时指定表使用的存储引擎

alter table 表 engine=存储引擎名；

//修改引擎会影响数据，最好在空表时修改

create table 表（字段列表）engine=存储引擎名；

5. 常用存储引擎的特点

innodb特点：

支持事务 、 事务回滚 、行级锁 、外键

存储方式： 一个表对应2个存储文件

表名.frm 表结构

表名.ibd 数据和索引,排队信息

myisam特点

不支持事务 、 事务回滚、外键

支持表级锁

存储方式： 一个表对应3个存储文件

表名.frm 表结构

表名.MYD 数据

表名.MYI 索引

1.7 建表时如何决定表使用那种存储引擎

执行写操作多的表适合使用innodb存储引擎，可以并发访问。

执行查操作多的表适合使用myisam存储引擎,可以节省系统资源

事务：对数据库服务的访问过程（连接数据库服务器 操作数据 断开连接）

事务回滚 ： 在事务执行过程中，任何一步操作失败，都会恢复之前的所有操作。

支持事务的表 有对应的事务日志文件记录执行过的sql操作,以下三个文件/var/lib/mlsql/

mysql数据库服务使用事务日志文件记录，对innodb存储引擎表执行的sql操作。

cd /var/lib/mysql/

ib\_logfile0 -||------> 记录SQL命令

ib\_logfile1 -||------>insert into t1 values(8888);

ibdata1 ----> 数据源（sql命令执行后产生的数据信息）

锁粒度：

表级锁（myisam）给整张表加锁

行级锁 (innodb) 只给表中当前被操作行加锁

锁的作用：解决对表的并发访问冲突问题。

锁类型

读锁 （共享锁）

写锁 （互斥锁）

事务特性 (ACID)

• 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。

• Consistency : 一致性

– 事务操作的过程中,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的。

• Durability :持久性

– 数据一旦提交,不可改变,永久改变表数据

mysql> show variables like "autocommit"; // 查看提交状态

mysql> set autocommit=off; // 关闭自动提交

mysql> rollback ; // 数据回滚

mysql> commit; // 提交数据

事务例子

mysql> set autocommit=off;

mysql> set autocommit=on;

mysql> create table t1234(id int) ;

mysql> insert into t1234 values(100);

mysql> insert into t1234 values(200);

mysql> set autocommit=off;

mysql> delete from t1234;

mysql> select \* from t1234;

Empty set (0.00 sec)

mysql> rollback;

mysql> select \* from t1234;

+------+

| id |

+------+

| 100 |

| 200 |

+------+

2 rows in set (0.00 sec)

mysql> set autocommit=on;

## 数据导入导出（批量操作数据）

**导入**

设置搜索路径

• 查看默认使用目录及目录是否存在

mysql> show variables like "secure\_file\_priv";

永久配置

]#vim /etc/my.cnf

[mysqld]

secure\_file\_priv="/mysqldata"

]# mkdir -p /mysqldata

]# chown mysql /mysqldata

]# systemctl restart mysqld

导入命令格式

LOAD DATA INFILE "目录/文件名"

->INTO TABLE 库.表

->FIELDS TERMINATED BY "列间隔符号"

->LINES TERMINATED BY "\n"

• 注意事项

– 字段分隔符要与文件内的一致

– 指定导入文件的绝对路径

– 导入数据的表字段类型要与文件字段匹配

– 禁用 SElinux

例子把passwd内容导入数据库

mysql> load data

-> infile "/var/lib/mysql-files/passwd"

-> into table db1.user1

-> fields terminated by ":"

-> lines terminated by "\n"

-> index(name);

**修改目录及查看修改结果**

[root@localhost ~]# mkdir /data ; chown mysql /mydata

[root@localhost ~]# vim /etc/my.cnf

[mysqld]

secure\_file\_priv="/mydata"

[root@localhost ~]# systemctl restart mysqld

mysql> show variables like "secure\_file\_priv";

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

| Variable\_name | Value

|

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

| secure\_file\_priv | /mydata/ |

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

**数据导出**

数据导出： 把表记录存储到系统文件里

命令格式:

sql查询命令 into outfile "目录/文件名" [ FIELDS TERMINATED BY "列间隔符号" LINES TERMINATED BY "\n" ]；

• 注意事项

– 导出的内容由 SQL 查询语句决定

– 禁用 SElinux

例子

mysql> select \* from user1

-> into outfile

-> "/mydata/user.txt";

[root@host50 ~]# cat /mydata/user.txt //查看导出的数据

用户授权与权限撤销

## 管理员密码管理

### 恢复数据库管理员本机登录密码

]# vim /etc/my.cnf

[mysqld]

skip-grant-tables

#validate\_password\_policy=0

#validate\_password\_length=6

:wq

]# systemctl restart mysqld

mysql> select host,user,authentication\_string from mysql.user;

mysql>update mysql.user set authentication\_string=password("abc123") where host="localhost" and user="root"; //authentication\_string时mysql数据库保存密码的字段

mysql> flush privileges; //修改mysql库，需要这个命令才能生效

mysql>quit

]# vim /etc/my.cnf

[mysqld]

#skip-grant-tables

validate\_password\_policy=0

validate\_password\_length=6

:wq //配置文件改回来

]# systemctl restart mysqld

修改数据库管理源本机登录的密码（需要知道当前登录的密码）

]#mysqladmin -hlocalhost -uroot -p password "新密码" //命令行修改

Mysql->alter user user() identified by “新密码” //mysql中修改

## 用户授权(grant)

在数据库服务器上添加连接用户，添加时可以设置用户的访问权限和连接的密码。默认只允许数据库管理员root用户在本机登录。默认只有数据库管理员root用户在本机登录才有授权权限。

3.2 用户授权命令的语法格式

mysql> grant 权限列表 on 数库名 to 用户名@"客户端地址"

identified by "密码" [with grant option];

权限列表的表示方式：

all 所有权限

select , insert,update(字段1，字段2)

数库名 的表示方式

\*.\* 所有库所有表

库名.\* 某个库

库名.表名 某张表

用户名的表示方式：

连接数据库服务器是使用的名字

授权时自定义，要有标识性

名字存储在mysql库下的user表里

客户端地址的表示方式：

% 所有地址

192.168.4.254 指定ip地址

192.168.4.% 网段

pc254.tedu.cn 主机名(要求数据库服务器可以解析主机名)

%.tedu.cn 域名(要求数据库服务器可以解析域名内的主机名)

identified by "密码" 授权用户连接数据库服务器密码自定义即可

with grant option 可选项，让新添加的授权用户有授权的权限。必须同时赋予mysql库的插入权限，才能对其他用户赋权。

grant select,update(name) on studb.t8 to yaya3@"%" identified by "123456";

grant select,insert,update on studb.dogperson to yaya@"localhost" identified by "123456";

在客户端使用授权用户连接数据库服务器

]# mysql -h数据库服务器ip地址 -u用户名 -p密码

相关命令

select user(); 显示连接用户名和客户端地址

select @@hostname ; 查看当前登录的主机名

show grants ; 登录用户查看自己的访问权限

select user,host from mysql.user; 查看当前已有的授权授权

show grants for 用户名 @“客户端地址”；查看已有授权用户的访问权限

drop user 用户名 @“客户端地址"; 删除授权用户

### 撤销用户权限命令的语法格式

------划重点

撤销的是用户的访问权限

用户对数据库有过授权才可以撤销

语法格式 revoke 权限列表 on 数库名 from 用户名@"客户端地址" ;

例子1 ： 撤销254主机 使用root用户连接时，授权权限。

mysql> revoke grant option on \*.\* from 'root'@'192.168.4.254';

例子2 ： 通过修改表记录的方式撤销用户的访问权限

mysql> update mysql.user set Select\_priv="N" where user= 'root' and host='192.168.4.254';

mysql> flush privileges;

3.5 数据库服务器使用授权库存储授权信息

mysql库 授权库 存储授权信息

user 存储授权用户的名及访问权限

db 存储授权用户对库的访问权限

tables\_priv 存储授权用户对表的访问权限

columns\_priv 存储授权用户对字段的访问权限

information\_schema 虚拟库 不占用物理存储空间 数据存储在物理内存里

存储已有库和表的信息

3.6 工作中如何授权:

给管理者授权 给完全权限 且有授权权限

给使用者授权 只给select 和 insert 权限

## 安装数据库图形管理工具 phpmyadmin

2.1 管理数据库服务器的方式有那些？

命令行 mysql

图形 （应用程序软件提供的图形界面 、 web页面）

2.2 在数据库服务器上安装图形管理工具phpmyadmin

准备软件的运行环境 lamp/lnmp

测试运行环境

安装软件phpMyAdmin-2.11.11-all-languages.tar.gz

]#tar -zxf phpMyAdmin-2.11.11-all-languages.tar.gz -C /var/www/html/

]# cd /var/www/html/

]#mv phpMyAdmin-2.11.11-all-languages phpmyadmin

修改软件的配置文件定义管理的数据库服务器

]#cd phpmyadmin

]#cp config.sample.inc.php config.inc.php

]#vim config.inc.php

17 $cfg['blowfish\_secret'] = 'plj123'; //随便输入六个字符

31 $cfg['Servers'][$i]['host'] = 'localhost';

:wq

在客户端访问软件 管理数据库服务器

<http://192.168.4.51/phpmyadmin>

## DBA 数据备份和恢复

**数据备份方式**

种解决方案分别针对于不同的场景

**直接cp,tar复制数据库文件**

如果数据量较小, 可以使用第一种方式, 直接复制数据库文件

**mysqldump+复制BIN LOGS**

如果数据量还行, 可以使用第二种方式, 先使用mysqldump对数据库进行完全备份, 然后定期备份BINARY LOG达到增量备份的效果

**lvm2快照+复制BIN LOGs**

如果数据量一般, 而又不过分影响业务运行, 可以使用第三种方式, 使用lvm2的快照对数据文件进行备份, 而后定期备份BINARY LOG达到增量备份的效果

**xtrabackup**

如果数据量很大, 而又不过分影响业务运行, 可以使用第四种方式, 使用xtrabackup进行完全备份后, 定期使用xtrabackup进行增量备份或差异备份

### 物理备份与恢复（授权库备份）

• 备份操作

– cp -rp /var/lib/mysql/ 数据库 备份目录 / 文件名

– tar -zcvf xxx.tar.gz /var/lib/mysql/ 数据库 /\*

• 恢复操作

– cp -rp 备份目录 / 文件名 /var/lib/mysql/

– tar -zxvf xxx.tar.gz -C /var/lib/mysql/ 数据库名/

[root@host50 ~]# mkdir /databak

[root@host50 ~]# cp -rp /var/lib/mysql/mysql /databak/ //确保文件的所属用户和所属主权限

[root@host50 ~]# rm -rf /var/lib/mysql/mysql

[root@host50 ~]# systemctl stop mysqld

[root@host50 ~]# systemctl start mysqld //报错无法启动

[root@host50 ~]# cp -rp /databak/mysql/ /var/lib/mysql/mysql

[root@host50 ~]# ls -ld /var/lib/mysql/mysql

drwxr-x--- 2 mysql mysql 4096 5月 23 10:24 /var/lib/mysql/mysql

[root@host50 ~]# systemctl start mysqld

使用50主机的授权库备份文件恢复51主机误删除的授权库

对另外一台主机host51

[root@host51 ~]# rm -rf /var/lib/mysql/mysql

[root@host51 ~]# systemctl restart mysqld //报错

[root@host51 ~]# scp -rp 192.168.4.50:/databak/mysql /var/lib/mysql/mysql

//文件的属主和属组并没有保持

[root@host51 ~]# chown -R mysql:mysql /var/lib/mysql/mysql //修改权限

[root@host51 ~]# systemctl start mysqld

### 逻辑备份 (mysqldump,mysql+binlog)

备份策略

完全备份 备份所有数据 mysqlddump

（一张表的所有记录 一个库的所有表 一台服务器上的所有库）

只备份新产生的数据

差异备份 备份自完全备份后，所有新产生的数据

增量备份 备份自上次备份后，所有新产生的数据

生产环境下使用的备份策略

完全备份+差异备份

完全备份+增量备份

数据备份时考虑哪些问题？

时间 策略 频率 存储空间 备份名

生产环境下执行备份数据的手段

crond + 备份脚本 (shell python perl ruby lua)

00 18 \* \* 1 /root/fullbak.sh(完全备份)

00 18 \* \* 27 /root/newbak.sh(增量备份)

systemctl enable crond

#############################################################################

**数据完全备份**

mysqldump --help

mysqldump -u用户 -p密码 数据库名 > 目录名/文件名.sql

数据库名的表示方式

--all-databases 或 -A 一台服务器上的所有库

数据库名 一个库的所有表

数据库名 表名 一张表的所有纪律

-B 库名 库名 库名 把多个库的所有表备份到同一个备份文件里

[root@host50 ~]# mysqldump -uroot -p123456 db1 > /databak/db1-all.sql

[root@host50 ~]# mysqldump -uroot -p123456 db1 user1 > /databak/db1-user1.sql

[root@host50 ~]# mysqldump -uroot -p123456 -A > /databak/all.sql

**数据完全恢复**

mysql -u用户 -p密码 数据库名 < 目录名/文件名.sql

方法一：[root@host50 ~]# mysql -uroot -p123456 db1 < /databak/db1-user1.sql

如果只把db1库中的表删除，则可以指定db1恢复

如果把删除了db1库，需要先创建db1，再指定db1 进行恢复 或者直接在.sql文件里添加下面的两行指令

create database db1;

use db1;

方法二：

mysql> use db1 //进入db1

mysql> source /databak/db1-user1.sql //在数据库中使用恢复

[root@host50 ~]# mysql -uroot -p123456 < /databak/all.sql

整个库全部备份不用填写库名

[root@host50 ~]# mysql -uroot -p123456 db88 < /databak/db1-all.sql

也可以把db1下所有表恢复到新的库中

[root@host50 ~]# mysql -uroot -p123456 db88 < /databak/db1-user1.sql

也可以把db1下user1表恢复到新的库中

也可以在另一台主机上恢复对应的库表

[root@host50 ~]# vim /root/fullbak.sh

#!/bin/bash

if [ ! -e /databak ];then

mkdir /databak

fi

day=`date +%F`

mysqldump -uroot -p123456 -A > /databak/allbak-${day}.sql

将上面的脚本放入计划任务

[root@host50 ~]# crontab -e

00 18 \* \* 1 /root/fullbak.sh &> /dev/null 每周一18点自动备份

[root@host50 ~]# crontab -l

二、增量备份

启动mysql服务的binlog日志文件实现实时增量备份

binlog日志介绍：mysql服务日志文件的一种，记录用户连接后执行的除查询之外的sql命令。默认没有启用。

二进制日志用途及配置方式

类 型 用 途 配 置

二进制日志 记录所有更改数据的操作 log\_bin[=dir/name]

除查询之外的所有sql命令 server\_id= 数字

max\_binlog\_size= 数字 m

启动binlog日志

[root@host50 ~]# vim /etc/my.cnf

[mysqld]

server\_id=50

log-bin

binlog\_format="mixed"

[root@host50 ~]# systemctl restart mysqld

三种记录格式 :

1. statement :每一条修改数据的sql命令都会记录在 binlog 日志中。

2. row: 不记录 sql 语句上下文相关信息,仅保存哪条记录被修改。

3. mixed: 是以上两种格式的混合使用。

查看日志文件内容

mysqlbinlog /var/lib/mysql/host50-bin.000001 //查看日志

cat host50-bin.index //日志索引

自定义日志文件名称和存储目录

[root@host50 mysql]# vim /etc/my.cnf

[mysqld]

log-bin=/logdir/gpf //修改默认日志的存放位置

[root@host50 mysql]# mkdir /logdir/

[root@host50 mysql]# chown mysql /logdir/

[root@host50 mysql]# systemctl restart mysqld

[root@host50 mysql]# ls /logdir/

gpf.000001 gpf.index

[root@host50 mysql]# mysqlbinlog /logdir/gpf.000001 | grep drop;

[root@host50 mysql]# mysqlbinlog /logdir/gpf.000001 | grep insert;

增删改，都会记录到日志中

binlog日志文件记录sql命令的方式

偏移量 pos

# at 402

#180529 15:19:04 server id 50 end\_log\_pos 499 CRC32 0x04f3dafc Query thread\_id=6 exec\_time=0 error\_code=0

SET TIMESTAMP=1527578344/\*!\*/;

create database db123

/\*!\*/;

时间点 time

执行binlog日志文件记录的sql命令恢复数据

**mysqlbinlog 选项 binlog日志文件 | mysql -uroot -p密码**

--start-position=起始偏移量 --stop-position=结束偏移量

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

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

在执行增量备份之前要先执行完全备份

生成新日志的4种方法（默认读取编号最大的日志文件），默认超过500M自动生成新日志

手动生成新的日志文件:

1. 重启 mysql 服务

2. 执行 SQL 操作 mysql> flush logs;

3. mysqldump --flush-logs

mysqldump -uroot -p123456 -A --flush-logs > /databak/allbak-${day}.sql

4. mysql -uroot -p密码 -e 'flush logs'

清理 binlog 日志

• 删除早于指定版本的 binlog 日志

– purge master logs to "binlog 文件名 "; //删除此版本之前的日志

• 删除所有 binlog 日志,重建新日志

– reset master;

显示数据库正在使用的binlog日志文件和偏移量

mysql> show master status;

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

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

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

| gpf.000001 | 154 | | | |

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

1 row in set (0.00 sec)

### innobackupex 备份 / 恢复

MySQL 备份工具

常用的 MySQL 备份工具

• 物理备份缺点

– 跨平台性差

– 备份时间长、冗余备份、浪费存储空间

• mysqldump 备份缺点

– 效率较低,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

XtraBackup 工具? 在恢复时/var/lib/mysql/下的目录要清空，否则回覆盖

• 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

安装 XtraBackup

• 下载适配的 RPM 包并安装

– percona-xtrabackup-24-2.4.7-1.el7.x86\_64.rpm

#yum -y install perl-Digest-MD5.x86\_64 rsync perl-DBD-MySQL

#rpm -ivh libev-4.15-1.el6.rf.x86\_64.rpm

#rpm -ivh percona-xtrabackup-24-2.4.7-1.el7.x86\_64.rpm

# rpm -ql percona-xtrabackup-24

/usr/bin/innobackupex // 备份 innodb 、 xtrdb 、 myisam 引擎的表

/usr/bin/xbcloud

/usr/bin/xbcloud\_osenv

/usr/bin/xbcrypt

/usr/bin/xbstream

/usr/bin/xtrabackup // 备份 innodb 、 xtrdb 引擎的表

innobackupex 基本选项

常用选项 含 义

--host 主机名

--user 用户名

--port 端口号

--password 密 码

--databases 数据库名

--no-timestamp 不用日期命名备份文件存储的子目录名

--databases=" 库名 " // 单个库

--databases=" 库 1 库 2" // 多个库

--databases=" 库.表 " // 单个表

innobackupex --user root --password 123456 --databases="mysql performance\_schema sys db1 db4 db88 persondb student test" /allbak --no-timestamp

安装第三方软件percona提供的命令innobackupex做增量备份

完全备份和恢复

增量备份和恢复

使用完全备份恢复单个表

完全备份和恢复

1、**innobackupex完全备份**

[root@host50 mysql]# innobackupex --user root --password 123456 \

> --databases="mysql sys performance\_schema db5" \

> /db5allbak --no-timestamp /db5allbak 这个文件一定要新的，唯一的

[root@host50 db5allbak]# ls

backup-my.cnf ibdata1（重要） sys xtrabackup\_info

db5 mysql xtrabackup\_binlog\_info xtrabackup\_logfile（重要）

ib\_buffer\_pool performance\_schema xtrabackup\_checkpoints（重要）

备份存放了2部分文件：数据库内容数据，备份日志信息

[root@host50 db5allbak]# cat xtrabackup\_checkpoints

backup\_type = full-backuped //当前备份的备份类型

from\_lsn = 0 //innodb 的事务记录sql命令，都有一个编号lsn，日志序列号

to\_lsn = 7352422 //sql命令的序列号范围

last\_lsn = 7352431

compact = 0

recover\_binlog\_info = 0

//lsn的序列号是从/var/log/mysql/下的几个事务日志文件提取出来的

//xtrabackup\_logfile 记录这个lsn范围内的sql命令

//该备份文件下也有个ibdata1文件，和数据库目录下的同名文件作用类似，用来记录xtrabackup\_logfile里面sql命令执行后的数据信息

/var/lib/mysql/ib\_logfile\* 事务日志记录，对每一个表的sql命令

/var/lib/mysql/ibdata1 存放每个sql命令执行结果的信息 这几个表统称为事务日志文件

2**、innobackupex完全恢复**

常用选项 含 义

--redo-only 日志合并

--apply-log 准备还原 ( 回滚日志 ) //关键步骤

--copy-back 恢复数据 //关键步骤

--incremental 目录名 增量备份

--incremental--basedir= 增量备份时,指定上一次备

目录名 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

[root@host50 ~]# innobackupex --user root --password 123456 --databases="mysql sys performance\_schema db5" --apply-log /db5allbak

[root@host50 db5allbak]# cat xtrabackup\_checkpoints

backup\_type = full-prepared //备份类型由 full-backuped变为full-prepared

from\_lsn = 0

to\_lsn = 7352891

last\_lsn = 7352900

compact = 0

recover\_binlog\_info = 0

[root@host50 ~]# innobackupex --user root --password 123456 --databases="mysql sys performance\_schema db5" --copy-back /db5allbak

//报错：Original data directory /var/lib/mysql is not empty!

//一定要清空数据库文件才能恢复

[root@host50 ~]# rm -rf /var/lib/mysql/\*

[root@host50 ~]# innobackupex --user root --password 123456 --databases="mysql sys performance\_schema db5" --copy-back /db5allbak

[root@host50 ~]# chown -R mysql:mysql /var/lib/mysql

//恢复的文件默认都是root的所属主和所属组，需要修改为mysql

3、重启服务

[root@host50 ~]# systemctl restart mysqld

登陆数据库查看数据信息

**innobackupex 增量备份**

--incremental 新目录

--incremental-basedir=上次恢复的备份目录

1、insert into db5.t1 values(666),(666),(666);

[root@host50 ~]# innobackupex --user root --password 123456 --databases="mysql sys performance\_schema db5" --incremental /new1dir --incremental-basedir=/db5allbak --no-timestamp

[root@host50 ~]# cd /new1dir/

[root@host50 new1dir]# cat xtrabackup\_checkpoints

backup\_type = incremental

from\_lsn = 7352891

to\_lsn = 7358319

last\_lsn = 7358328

compact = 0

recover\_binlog\_info = 0

2、insert into db5.t1 values(999),(999),(999);

[root@host50 ~]# innobackupex --user root --password 123456 --databases="mysql sys performance\_schema db5" --incremental /new2dir --incremental-basedir=/new1dir --no-timestamp

[root@host50 ~]# cd /new2dir/ //每次都要建新的文件

[root@host50 new2dir]# ls

[root@host50 new2dir]# cat xtrabackup\_checkpoints

backup\_type = incremental

from\_lsn = 7358319 //日志序列号从new1dir里面的to\_lsn=7358319开始

to\_lsn = 7360378

last\_lsn = 7360387

compact = 0

recover\_binlog\_info = 0

innobackupex 增量恢复

--apply-log

--redo-only 日志合并

--copy-back

--incremental-dir=目录名 //恢复数据使用的目录

1、准备恢复数据

[root@host50 ~]# innobackupex --user root --password 123456 --databases="mysql sys performance\_schema db5" --apply-log --redo-only /db5allbak

//把完全备份的日志进行回滚

[root@host50 db5allbak]# cat xtrabackup\_checkpoints

backup\_type = log-applied //备份类型由 full-prepared变为log-applied

from\_lsn = 0

to\_lsn = 7352891

last\_lsn = 7352900

compact = 0

recover\_binlog\_info = 0

2、合并日志（同时也把数据合并了）

将new1dir的日志合并进来

innobackupex --user root --password 123456 --databases="mysql sys performance\_schema db5" --apply-log --redo-only /db5allbak --incremental-dir=/new1dir

[root@host50 db5allbak]# cat xtrabackup\_checkpoints

backup\_type = log-applied

from\_lsn = 0

to\_lsn = 7358319 //日志序列已经增加到0-7358319，正好时new1dir的结尾

last\_lsn = 7358328

compact = 0

recover\_binlog\_info = 0

将new2dir的日志合并进来

innobackupex --user root --password 123456 --databases="mysql sys performance\_schema db5" --apply-log --redo-only /db5allbak --incremental-dir=/new2dir

[root@host50 db5allbak]# cat xtrabackup\_checkpoints

backup\_type = log-applied

from\_lsn = 0

to\_lsn = 7360378 //日志序列已经增加到0-7360378，正好时new2dir的结尾

last\_lsn = 7360387

compact = 0

recover\_binlog\_info = 0

3、恢复数据

先清空原来的数据库文件

[root@host50 db5allbak]# rm -rf /var/lib/mysql/\*

进行数据恢复

[root@host50 db5allbak]# innobackupex --user root --password 123456 --databases="mysql sys performance\_schema db5" --copy-back /db5allb

4、重启服务

chown -R mysql:mysql /var/lib/mysq //修改权限，默认恢复的文件为root所属

[root@host50 db5allbak]# systemctl restart mysqld

5、检查表数据

mysql> select \* from db5.t1;

+------+

| id |

+------+

| 100 |

| 200 |

| 300 |

| 400 |

| 500 |

| 666 |

| 666 |

| 666 |

| 666 |

| 666 |

| 666 |

| 999 |

| 999 |

| 999 |

+------+

已经完全恢复

6. 这时候new1dir new2dir 已经没用了,可以删除

**使用innobackupxp 完全备份恢复单个表**

[root@host52 ~]# innobackupex --user root --password 123456 --databases="db55" /db55dir --no-timestamp

mysql> drop table a;

恢复单个表的步骤

--export 导出表信息

--import 导入表空间

1、在备份文件里导出表信息

[root@host52 db55]# ls

a.frm a.ibd b.frm b.ibd db.opt

innobackupex --user root --password 123456 --databases="db55" --apply-log --export /db55dir

[root@host52 db55]# ls

a.cfg a.exp a.frm a.ibd b.cfg b.exp b.frm b.ibd db.opt

//每个表多出来2个文件 a.cfg a.exp 把备份文件中表的信息导出来

2、创建删除的表，并删除表空间文件

mysql> create table a(id int); 必须和原表的表结构一致

[root@host52 db55]# ls /var/lib/mysql/db55/ -l

总用量 220

-rw-r----- 1 mysql mysql 8556 5月 30 14:26 a.frm

-rw-r----- 1 mysql mysql 98304 5月 30 14:26 a.ibd //这里的表空间是刚刚创建的，没有数据

-rw-r----- 1 mysql mysql 8560 5月 30 14:09 b.frm

-rw-r----- 1 mysql mysql 98304 5月 30 14:10 b.ibd

-rw-r----- 1 mysql mysql 65 5月 30 14:08 db.opt

mysql> alter table db55.a discard tablespace; // 删除表空间,a.ibd就没有了

3、把备份目录下，导出的表信息文件和表空间文件，拷贝到对应的数据库目录

[root@host52 db55]# cp /db55dir/db55/a.{cfg,exp,ibd} /var/lib/mysql/db55/

//或者在mysql下 mysql> system cp /allbak/db55/a.{ibd,cfg,exp} /var/lib/mysql/db55/

[root@host52 db55]# chown mysql:mysql /var/lib/mysql/db55/a.\*

4、导入表空间 import

mysql> alter table db55.a import tablespace;

5、查看恢复的数据，删除a.exp a.cfg

mysql> select \* from a;

+-------+

| id |

+-------+

| 110 |

| 120 |

| 119 |

| 114 |

| 12306 |

| 112 |

+-------+

6 rows in set (0.00 sec)

### 部署mysql主从同步结构实现数据自动备份

### lvm2 snapshot +binlog

## 视图

视图介绍

• 什么是视图 (View)

– 虚拟表

– 内容与真实的表相似,包含一系列带有名称的列和行数据。

– 视图并不在数据库中以存储的数据的形式存在。

– 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

视图优点

• 简单

– 使用视图的用户完全不需要关心视图中的数据是通过什么查询得到的。

– 视图中的数据对用户来说已经是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

– 一旦视图的结构确定了,可以屏蔽表结构变化对用户的影响。

使用视图的限制

• 不能在视图上创建索引

• 在视图的 FROM 子句中不能使用子查询

• 以下情形中的视图是不可更新的

– 包含以下关键字的 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 子句的子查询引用了 FROM 子句中的表

– 使用了临时表,视图是不可更新

视图的基本使用

创建视图

• 语法格式

– create view 视图名称 as SQL 查询;

– create view 视图名称 (字段名列表) as SQL 查询;

mysql> create view t11 as select \* from t1;

注意:

在视图表中不定义字段名的话,默认使用表中的字段名,

若定义字段名的话,视图表中的字段名个数必须和基本中的字段个数相等。

查看视图

• 查看当前库下所有表的状态信息

– show table status;

– show table status where comment="view"\G;

• 查看创建视图的具体命令

– show create view 视图名;

使用视图

• 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); //视图字段少于基表的，基表的其他字段填充为null

• 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;

注意:对视图操作即是对基本操作,反之亦然!!!

删除视图

• 语法格式

– drop view 视图名;

视图进阶

创建视图完全格式

• 命令格式

– create

[or replace]

[algorithm = {undefined | merge | temptab

le}]

[definer = { user | current\_user }]

[sql security { definer | invoker }]

view view\_name [(column\_list)]

as select\_statement

[with [cascaded | local] check option]

设置字段别名

• 命令格式

– 视图中的字段名不可以重复 所以要定义别名

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;

//关联查询建的视图 默认不允许修改视图字段的值

mysql> create view v1 as

select a.\*,b.gid,b.homedir

from a left join b

on a.name=b.name and a.uid=b.uid;

mysql> create view v2 as

select \*

from a left join b

on a.name=b.name and a.uid=b.uid;

ERROR 1060 (42S21): Duplicate column name 'name'

mysql> create view v2 as

select a.name as aname,b.name as bname,b.uid as buid

from a left join b

on a.name=b.name;

mysql> create view v3 as

select a1.name as aname,b1.name as bname,b1.uid as buid

from a a1 left join b b1

on a1.name=b1.name;

OR REPLACE

• 语法格式

– 创建时,若视图已存在,会替换已有的视图

– Create or replace view 视图名 as select 查询;

mysql>create view v2 as select \* from t1;

mysql> create view v2 as select \* from t1;

ERROR 1050 (42S01): Table ‘v2’ already exists // 提示已存在

mysql> create or replace view v2 as select \* from t1; // 没有提示

Query OK, 0 rows affected (0.00 sec)

ALGORITHM

• 定义处理视图的方式

– ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}

• MERAGE (替换方式)

– 视图名直接使用视图的公式替换掉,把视图公式合并到了 select 中。

• TEMPTABLE (具体化方式)

– 先得到视图的执行结果,该结果形成一个中间结果暂时存在

内存中,之后,外面的 select 语句就调用了这些中间结果。

• UNDEFINED (未定义)

– ALGORITHM 选项的值是 UNDEFINED 表示使用的是 MER

AGE 替换方式。

WITH CHECK OPTION

• 当视图是根据另一个视图定义时 , 对视图更新 / 删除 / 插入

– LOCAL 和 CASCADED 关键字决定了检查的范围。

– LOCAL 仅检查当前视图的限制。

– CASCADED 同时要满足基表的限制。

mysql> create view v1 ( 默认值 )

as

select \* from a where uid < 10

with check option;

Query OK, 0 rows affected (0.09 sec )

mysql> create view v2

as

select \* from v1

where uid >=5 with local check option;

Query OK, 0 rows affected (0.09 sec)

没有检查控制的情况

mysql> create table tea select name,uid,shell from user where uid>=1 and uid <=40;

mysql> create view v1 as select \* from tea where uid<=10;

mysql> show create view v1\G;

mysql> select \* from v1;

mysql> update v1 set uid=11 where name="bin";

//可以修改成功，基表也修改了，但不满足建视图时 uid<=10 的规则，所以在视图v1中不显示

mysql> select \* from v1;

mysql> create view v2 as select name,uid from tea where uid>=5 and uid <=27;

mysql> select \* from v2;

mysql> create view v3 as select \* from v2 where uid <=15;

mysql> update v3 set uid=16 where name="bin";

//可以修改成功，v2基表也修改了，但不满足建视图时 uid<=15 的规则，所以在视图v3中不显示，在v2中显示

有检查控制的情况

mysql> create table tea select name,uid,shell from user where uid>=1 and uid <=40;

mysql> create view v1 as select name,uid from tea where uid >=5 and uid<=25;

mysql> create view v2 as select \* from v1 where uid<=15 with check option; //默认时cascaded

mysql> show create view v2\G;

mysql> update v2 set uid=16 where name="ftp";

ERROR 1369 (HY000): CHECK OPTION failed 'bbsdb.v2' //错误不能改，超过自身的15限制

mysql> update v2 set uid=4 where name="sync";

ERROR 1369 (HY000): CHECK OPTION failed 'bbsdb.v2' //错误不能改，超过基表的5-25限制

在创建v2时如果加上 with local check option,与上面比较，

mysql> update v2 set uid=16 where name="ftp"; //同样不能修改成功，超过自身的15限制

mysql> update v2 set uid=4 where name="sync"; //可以修改成功，只检查自己的限制，满足<=15

## 存储过程

存储过程介绍

• 什么存储过程

– 数据库中保存的一系列 sql 命令的集合

– 编写存储过程时,可以使用变量、条件判断、流程控制等

– 存储过程,就是 MySQL 中的脚本

存储过程优点

– 提高性能

– 可减轻网络负担

– 可以防止对表的直接访问

– 避免重复的 sql 操作

创建存储过程

语法格式

delimiter //

create procedure 名称

()

begin

功能代码

.....

.....

end

// 结束存储过程

delimiter ;

delimiter 关键字声明当前段分隔符

MySQL 默认以“ ;” 为分隔符,没有声明分割符,编译器会把存储过程当成 SQL 语

句进行处理,则存储过程的编译过程会报错。此处修改为使用//来表示执行sql命令，最后 再改回来

mysql> use db88 //必须先进入到对应的库中

mysql> delimiter //

mysql> create procedure say()

begin

select \* from db88.user;

select name from db88.user;

end

//

mysql> delimiter ;

查看存储过程

• 方法 1

– mysql> show procedure status;

• 方法 2

– mysql> select db,name,type from mysql.proc where name=“ 存储过程名 ";

mysql> select db,name,type from mysql.proc where name="say";

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

| db | name | type

|

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

| studydb | say | PROCEDURE |

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

mysql> select \* from mysql.proc where name="say"\G;

调用 / 删除存储过程

• 调用存储过程

– Call

存储过程名 (); //存储过程没有参数时, () 可以省略有参数时,在调用存储过程时,必须传参。

• 删除存储过程

– drop procedure 存储过程名;

mysql> call say();

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

| name | password | uid | gid | comment | homedir | shell |

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

| root | x | 0 | 0 | root | /root | /bin/bash |

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

mysql> drop procedure say;

参数类型

• MySQL 存储过程,共有三种参数类型 IN,OUT,INOUT

Create procedure 名称 (

类型 参数名 数据类型 ,类型 参数名 数据类型

)

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;默认类型是 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

注意:此三中类型的变量在存储过程中调用时不需要加 @ 符号 !!!

mysql> delimiter //

mysql> create procedure say2(IN username char(10))

-> begin

-> select count(name) from db88.user where name=username;

-> end

-> //

mysql> delimiter ;

mysql> call say2("root");

mysql> create procedure say11(

-> in username char(10),out usernum int(2),inout shellname char(20))

-> begin

-> select username;

-> select usernum;

-> select shellname;

-> end //

mysql> delimiter ;

mysql> call say11("root",@w,@w1); //in必须赋值 out和inout两个变量不用赋值，这里赋空占位

mysql> set @x="root";

mysql> call say11(@x,@w,@w1);

mysql> delimiter //

mysql> create procedure say142(inout shellname char(20))

-> begin

-> select name,shell from db88.user where shell=shellname;

-> set shellname = "shutdown";

-> select shellname;

-> end //

mysql> delimiter ;

mysql> call say142("/bin/bash");

ERROR 1414 (42000): OUT or INOUT argument 1 for routine db88.say142 is not a variable or NEW pseudo-variable in BEFORE trigger //报错

mysql> set @shellname="/bin/bash";

mysql> call say142(@shellname);//必须在调用之前定义一个变量,然后赋值给它,inout和out必须赋值为一个变量,不能是常量

mysql> delimiter //

mysql> create procedure say14(in username char(10),out usernum int(2))

-> begin

-> select usernum;

-> select count(name) into usernum from db88.user where name=username;

-> select usernum;

-> end //

mysql> delimiter ;

mysql> set @x=7;

mysql> call say14("root",@x); //out的变量,第一个usernum依旧为空,这里只是占位

+---------+

| usernum |

+---------+

| NULL |

+---------+

+---------+

| usernum |

+---------+

| 1 |

+---------+

变量类型

• 变量的种类 : 全局变量 \ 会话变量 \ 用户变量 \ 局部变量

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 命令定义;

全局变量 全局变量的修改会影响到整个服务器,但是对会

话变量的修改,只会影响到当前的会话。

用户变量 在客户端连接到数据库服务的整个过程中都是有效的。当当前连接断开后所有用户变量失效。

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。

注意:局部变量 和 参数变量 调用时 变量名前不需要加 @

mysql> show global variables; // 查看全局变量

mysql> show session variables; // 查看会话变量

mysql> set session sort\_buffer\_size = 40000; // 设置会话变量

mysql> show session variables like “sort\_buffer\_size”; // 查看会话变量

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

| Variable\_name | Value |

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

| sort\_buffer\_size | 40000 |

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

Mysql> show global variables like “% 关键字 %”; // 查看全局变量

mysql> set @y = 3; // 用户自定义变量,直接赋值

mysql> select max(uid) into @y from user; // 使用 sql 命令查询结果赋值

算数运算

• 算数运算符号

符号 描述 例子

+ 加法运算 SET @var1=2+2; 4

- 减法运算 SET @var2=3-2; 1

\* 乘法运算 SET @var3=3\*2 ; 6

/ 除法运算 SET @var4=10/3; 3.333333333

DIV 整除运算 SET @var5=10 DIV 3; 3

% 取模 SET @var6=10%3 ; 1

mysql> set @z=1+2;select @z;

mysql> set @x=1; set @y=2;set @z=@x\*@y; select @z;

mysql> set @x=1; set @y=2;set @z=@x-@y; select @z;

mysql> set @x=1; set @y=2;set @z=@x/@y; select @z;

mysql> drop procedure if exists say;

mysql> delimiter //

mysql> create procedure say(

in bash char(20), in nologin char(25), out x int , out y int)

begin

declare z int ;

set z=0;

select count(name) into @x from db88.user where shell=bash;

select count(name) into @y from db88.user where shell=nologin;

set

z=@x+@y; select z;

end //

mysql> delimiter ;

mysql> call say("/bin/bash","/sbin/nologin",@x,@y);

条件判断

• 数值的比较

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

流程控制

顺序结构

• 当“条件成立”时执行命令序列

• 否则,不执行任何操作

if 条件测试 then

代码......

.....

end if ;

• 当“条件成立”时执行代码 1

• 否则,执行代码 2

if 条件测试 then

代码 1 ......

.....

else

代码 2......

.....

end if;

多分支

if 条件测试 then

代码 1 ......

.....

elif 条件测试 then

...

...

...

else

代码 ......

.....

end if;

mysql> drop procedure if exists say;

mysql> delimiter //

mysql> create procedure say(in x int(1) )

begin

if x <= 10 then

select \* from userdb.user where id <=x;

end if;

end

//

mysql> delimiter ;

mysql> call say(1); # 条件判断成立

mysql> drop procedure if exists say;

mysql> delimiter //

mysql> create procedure say(in x int(1) )

begin

if x is null then

set x = 1; //这里如果写成set @x=1; 输入空值时没有显示id=1的

select \* from userdb.user where id=x;

end if;

if x <= 10 then

select \* from userdb.user where id <=x;

end if;

end

//

mysql> delimiter ;

MySQL> call say(@x) // 调用未定义变量 x

mysql> call say(9);

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

| id | name | passwd | uid | gid | 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 |

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

循环结构

• 条件式循环

– 反复测试条件,

– 只要成立就执行命令序列

while 条件判断 do

循环体

.......

end while ;

mysql> delimiter //

mysql> create procedure say()

-> begin

-> declare i int;

-> set i=1;

-> while i <= 5 do

-> select i;

-> set i=i+1;

-> end while;

-> end

-> //

mysql> delimiter ;

mysql> call say();

• 条件式循环

– 无循环条件

loop

循环体

......

end loop ;

mysql> delimiter //

mysql> create procedure say2( )

-> begin

-> declare i int;

-> set i=1;

-> loop

-> select i;

-> set i=i+1;

-> end loop;

-> end

-> //

mysql> delimiter ;

mysql> call say2( ); # 不按 ctrl+c 结束 会一直输出变量 i 的值

条件式循环

– until 条件判断,不成立时结束循环

repeat

循环体

until 条件判断

end repeat ;

mysql> delimiter //

mysql> create procedure say3( )

-> begin

-> declare i int;

-> set i=1;

-> repeat

-> select i;

-> set i=i+1;

-> until i=6 // 此处不需要使用;

-> end repeat;

-> end

-> //

mysql> delimiter ;

控制语句

• 循环结构控制语句 , 控制循环结构的执行。

– 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

mysql> create procedure say()

-> begin

-> declare i int;

-> set i=1;

-> loab1:loop // 定义标签名为 loab1

-> select i;

-> set i=i+1;

-> if i=3 then //i值是3时结束本次循环

set i=i+1;

-> iterate loab1;

-> end if;

-> if i=7 then #i 值是 7 时 结束循环chenggonganzhuangsougoushurufa

-> leave loab1;

-> end if;

-> end loop;

-> end //

mysql> create procedure say()

-> begin

-> declare i int;

-> set i=0;

-> loab1:loop // 定义标签名为 loab1

-> set i=i+1;

-> if i=3 then //i值是3时结束本次循环

-> iterate loab1;

-> end if;

-> select i;

-> if i=7 then #i 值是 7 时 结束循环

-> leave loab1;

-> end if;

-> end loop;

-> end //

## 分库分表

如何解决数据库的并发访问压力

如何解决单表过大的问题

一张表最多可以存储多少条记录

分库分表

• 什么是分库分表

– 通过某种特定条件,将存放在一个数据库 ( 主机 ) 中的数据,分散存放到多个数据库 ( 主机 ) 中。

– 已达到分散单台设备负载的效果,即分库分表

– 数据的切分根据其切分规则的类型,分为 2 种切分模式

– 垂直分割 ( 纵向 ) 和 水平分割 ( 横向 )垂直分割

• 纵向切分

– 把单一的表,拆分成多个表,并分散到不同的数据库( 主机 ) 上。

– 一个数据库由多个表构成,每个表对应不同的业务,可以按照业务对表进行分类,

将其分布到不同的数据库 ( 主机 ) 上,实现专库专用,让不同的库 ( 主机 ) 分担不同的业务。水平分割

• 横向切分

– 按照表中某个字段的某种规则,把向表中写入的记录分散到多个库 ( 主机 ) 中。

– 简单来说,就是按照数据行切分,将表中的某些行存储到指定的数据库 ( 主机 ) 中。

有利有弊,弊端,查询检索表时,需要从多个表中全部查询,减慢查询速度

软件介绍

• Mycat 是基于 Java 的分布式数据库系统中间层,为高并发下的分布式提供解决方案

– 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

mycat分片规则，用户手册， schema.xml

分片规则

• mycat 服务提供 10 种分片规则。每种规则都有自己的算法提供

– 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 使用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

-----> mycat ---dn1

| \

| \

| dn2

dn3

当 Mycat 收到一个 SQL 时

– 会先解析这个 SQL 查找涉及到的表,然后看此表的定义

– 如果有分片规则,则获取到 SQL 里分片字段的值,并匹配分片函数,得到该 SQL 对应的分片列表

– 然后将 SQL 发往这些分片去执行,最后收集和处理所有分片返回的结果数据,并输出到客户端

以 select \* from Orders where prov=? 语句为例,查到 prov=wuhan ,

按照分片函数, wuhan 返回 dn1 ,于是 SQL 就发给了 MySQL1 ,去

取 DB1 上的查询结果,并返回给用户。

如果上述 SQL 改为 elect \* from Orders where prov in

(‘wuhan’,‘beijing’) ,那么, SQL 就会发给 MySQL1 与 MySQL2 去执行,

然后结果集合并后输出给用户。但通常业务中我们的 SQL 会有 Order

By 以及 Limit 翻页语法,此时就涉及到结果集在 Mycat 端的二次处理

防止查询速度慢

优化,把访问热度高的放到缓存服务器

配置 mycat

IP 规划

• 具体配置如下:

拓扑名称 主机名 角色 数据库名 IP地址

hostA client 客户端 无 192.168.4.254/24

hostB Mycat Mycat 服务器 无 192.168.4.56/24

hostC c1 数据库服务器 db1 192.168.4.55/24

hostD c2 数据库服务器 db2 192.168.4.54/24

1、装包

1.1安装 JDK

[ root@localhost ~]# rpm -qa | grep -i jdk # 安装系统自带的即可

java-1.8.0-openjdk-1.8.0.65-3.b17.el7.x86\_64

java-1.8.0-openjdk-headless-1.8.0.65-3.b17.el7.x86\_64

1.2安装 mycat 服务软件包

[ root@localhost ~]# tar -zxf Mycat-server-1.4-beta-

20150604171601-linux.tar.gz # 免安装,解压后即可使用

[ root@localhost ~]# mv mycat/ /usr/local/

[root@localhost ~]# ls /usr/local/mycat/

bin catlet conf lib logs version.txt

2、修改配置文件

• 目录结构说明

– bin mycat 命令 如 启动 停止 等

– catlet 扩展功能

– conf 配置文件

– lib mycat 使用的 jar 包 mycat 是 java 开发的

– log mycat 启动日志和运行日志

– wrapper.log mycat服务启动日志 ,启动有问题可以看这个日志的内容

– mycat.log 记录sql脚本执行后的具体报错内容

• 重要配置文件说明

– server.xml 设置连接 mycat 服务的账号 、密码等

– schema.xml 配置 mycat 使用的真实数据库和表

– rule.xml 定义 mycat 分片规则

• 配置标签说明

– <user>.......</user> 定义连接 mycat 服务时使用的用户和密码及逻辑库的名字

– <datanode>......</datanode> 指定数据节点(物理库的主机名和存储分片数据的数据库名)

– <datahost>......</datahost> 指定数据库服务器的 IP地址 及连接数据库时使用的授权用户名及密码

• 修改配置文件 /usr/local/mycat/conf/server.xml

<user name="test"> //连接 mycat 服务时使用的用户名 test

<property name="password">123456</property> //使用test用户连接 mycat用户时使用的密码

<property name="schemas">TESTDB</property> //连接上mycat服务后,可以看到的库名多个时,使用逗号分隔 (是逻辑上的库名)

</user>

<user name="user">

<property name="password">123456</property>

<property name="schemas">TESTDB</property>

<property name="readOnly">true</property> # 定义只读权限,使用定义的 user 用户连接mycat服务后只有读记录的权限

</user>

修改配置文件 /usr/local/mycat/conf/schema.xml

– 定义分片信息

<schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100">

<!-- auto sharding by id (long) -->

<table name="travelrecord" dataNode="dn1,dn2" rule="auto-sharding-long" />

<!-- global table is auto cloned to all defined data nodes ,so can join

with any table whose sharding node is in the same data node -->

<table name="company" primaryKey="ID" type="global" dataNode="dn1,dn2" />

<table name="goods" primaryKey="ID" type="global" dataNode="dn1,dn2" />

<!-- random sharding using mod sharind rule -->

<table name="hotnews" primaryKey="ID" dataNode="dn1,dn2"

rule="mod-long" />

<!-- <table name="dual" primaryKey="ID" dataNode="dnx,dnoracle2" type="global"

needAddLimit="false"/> <table name="worker" primaryKey="ID" dataNode="jdbc\_dn1,jdbc\_dn2,jdbc\_dn3"

rule="mod-long" /> -->

<table name="employee" primaryKey="ID" dataNode="dn1,dn2"

rule="sharding-by-intfile" />

<table name="customer" primaryKey="ID" dataNode="dn1,dn2"

rule="sharding-by-intfile">

<childTable name="orders" primaryKey="ID" joinKey="customer\_id"

parentKey="id">

<childTable name="order\_items" joinKey="order\_id"

parentKey="id" />

</childTable>

<childTable name="customer\_addr" primaryKey="ID" joinKey="customer\_id"

parentKey="id" />

</table>

<!-- <table name="oc\_call" primaryKey="ID" dataNode="dn1$0-743" rule="latest-month-calldate"

/> -->

</schema>

<dataNode name="dn1" dataHost="c1" database="db1" /> //修改c1、c2

<dataNode name="dn2" dataHost="c2" database="db2" />

<!--<dataNode name="dn3" dataHost="localhost1" database="db3" /> --> //注释第三个数据节点主机

<dataHost name="c1" maxCon="1000" minCon="10" balance="0"

writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

<heartbeat>select user()</heartbeat>

<!-- can have multi write hosts -->

<writeHost host="hostM1" url="192.168.4.55:3306" user="root"

password="123456">

<!-- can have multi read hosts -->

</writeHost>

</dataHost>

<dataHost name="c2" maxCon="1000" minCon="10" balance="0"

writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

<heartbeat>select user()</heartbeat>

<!-- can have multi write hosts -->

<writeHost host="hostM2" url="192.168.4.54:3306" user="root"

password="123456">

<!-- can have multi read hosts -->

</writeHost>

</dataHost>

• 修改数据库服务器配置文件

– 添加对应设置后重启 mysqld 服务

– 添加授权用户

host54 和host55上都操作

Vim /etc/my.cnf

[mysqld]

......

......

lower\_case\_table\_names = 1 # 表名区分字母大小写

[root@localhost ~]# systemctl restart mysqld

//mycat 连接数据库服务器用户

54和55上都操作

mysql> grant all on \*.\* to root@"%" identified by '123456';

mysql> grant all on \*.\* to root@"%" identified by '123456';

54上创建db2库

55上创建db1库

3、启动服务

• 启动服务

– 指定 java 路径、添加 PATH 路径、启动服务

[root@host56 conf]# /usr/local/mycat/bin/mycat --help

Usage: /usr/local/mycat/bin/mycat { console | start | stop | restart | status | dump }

[root@host56 conf]# /usr/local/mycat/bin/mycat start

Starting Mycat-server...

[root@host56 conf]# /usr/local/mycat/bin/mycat status

Mycat-server is running (3299).

[root@host56 conf]# ps -P 3299

[root@host56 conf]# ss -anptul | grep 8066

tcp LISTEN 0 100 :::8066 :::\* users:(("java",pid=3301,fd=49))

/usr/local/mycat/logs/wrapper.logmycat 服务启动日志 ,启动有问题可以看这个日志的内容

/usr/local/mycat/logs/mycat.log 记录sql脚本执行后的具体报错内容

4、测试配置

[root@room9pc01 ~]# mysql -h192.168.4.56 -P8066 -utest -p123456

//客户端使用test用户登录56mycat服务器，这里就是上面配置的用户名和密码，如果使用user用户名，只能读，不能写

mysql> show databases;

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

| DATABASE |

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

| TESTDB |

+----------+ //这里的TESTDB是逻辑库，不是真实存在，相当于host54的db2和host55的db1的虚拟库

mysql> use TESTDB;

mysql> show tables;

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

| Tables in TESTDB |

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

| company |

| customer |

| customer\_addr |

| employee |

| goods |

| hotnews |

| orders |

| order\_items |

| travelrecord |

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

//库里面的这些表就是host56 mycat服务器上/usr/local/mycat/conf/schema.xml里面定义的表

//默认有显示，但都是不存在的，需要手动创建，同时db1和db2里也自动创建

mysql> desc employee;

ERROR 1146 (42S02): Table 'db1.employee' doesn't exist

mysql> create table employee (ID int(2) primary key,

sharding\_id int not null,

name char(50),

homedir char(50));

可以看到，emloyee的表文件规则，在mycat服务器上的schema.xml，必须要有主键ID，规则要是sharding-by-intfile

<table name="employee" primaryKey="ID" dataNode="dn1,dn2"

rule="sharding-by-intfile" />

可以看到，在在mycat服务器上rule.xml里面 sharding-by-intfile的规则必须要有sharding\_id这个字段，

算法是hash-int，在这个配置文件中查找hash-int

<tableRule name="sharding-by-intfile">

<rule>

<columns>sharding\_id</columns>

<algorithm>hash-int</algorithm>

查找结果为：

<function name="hash-int"

class="org.opencloudb.route.function.PartitionByFileMap">

<property name="mapFile">partition-hash-int.txt</property>

</function>

该算法调用的class类的某种脚本，属性名字是一个文本文件，查看这个文件

[root@host56 conf]# cat /usr/local/mycat/conf/partition-hash-int.txt

10000=0

10010=1

这里需要说明的是，sharding\_id 如果是10000，就存放到dn1 也就是c1，也就是192.168.4.55的db1库里面去

sharding\_id 如果是10010，就存放到dn2 也就是c2，也就是192.168.4.54的db2库里面去

//可以添加多条记录，如果有多个分片数据库服务器的话

mysql> insert into employee(ID,sharding\_id,name,homedir) values(1,10000,"bao","beijing");

//在插入数据时，表字段名必须列出来

//sharding\_id的值只能是10000或者10010，这里会写进host55的db1底下的employee表里

mysql> insert into employee(ID,sharding\_id,name,homedir) values(2,10010,"haha","guangzhou");

//这里会写进host54的db2底下的employee表里

# mysql集群

## mysql主从同步

MySQL 主从同步：是存储数据的一种结构模式

主：被客户端访问的数据库服务器就是主库服务器 master

从：连接主库服务器，自动同步主库上的所有数据到本机 slave

• 对指定库的异地同步?

• MySQL 主 --> 从复制架构的实现?

• MySQL 服务器的只读控制 ?

主从同步结构模式介绍

• 基本应用

– 单向复制: 主 --> 从 1主1从

• 扩展应用

– 链式复制: 主 --> 从 --> 从 主从从

– 双向复制: 主 <--> 从 缺点：同时访问时存在写冲突 互为主从

– 放射式复制: 从<-- 主 --> 从 1主多从

|

从

**一主一从配置 host52 host53**

主库配置步骤：

启用binlog日志

做用户授权(添加一个从库同步数据时的连接用户)

查看正在使用的binlog日志文件信息 show master status;

从库配置步骤：

1、指定server\_id 并重启数据库服务

2、管理员登陆指定主数据库服务器的信息

3、启动slave进程

公共配置：

运行数据库服务管理员root能够在本机登陆

互相能够ping

从库上和主库上的初始库数据要一致

主master 192.168.4.52

[root@host52 ~]# vim /etc/my.cnf

[mysqld]

server\_id=52

log-bin=master

binlog-format="mixed"

validate\_password\_policy=0

validate\_password\_length=6

[root@host52 ~]# systemctl restart mysqld

[root@host52 ~]# cd /var/lib/mysql/

[root@host52 mysql]# ls master\*

master.000001 master.index

mysql> show master status;

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

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

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

| master.000002 | 448 | | | |

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

mysql> grant replication slave on \*.\*

-> to

-> yaya@"192.168.4.53"

-> identified by "123456";

mysql> show grants for yaya@"192.168.4.53";

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

| Grants for yaya@192.168.4.53 |

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

| GRANT REPLICATION SLAVE ON \*.\* TO 'yaya'@'192.168.4.53' |

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

从slave 192.168.4.53

数据库同步,slave数据只能比master多

[root@host53 ~]# mysql -uyaya -p123456 -h192.168.4.52

可以登陆

1、指定server\_id 并重启数据库服务

[root@host53 ~]# vim /etc/my.cnf

[mysqld]

server\_id=53

[root@host53 ~]# systemctl restart mysqld

[root@host53 ~]# mysql -uroot -p123456

2、管理员登陆指定主数据库服务器的信息

mysql> change master to

-> master\_host="192.168.4.52", //主库ip

-> master\_user="yaya", //连接主库的用户名

-> master\_password="123456", //对应的密码

-> master\_log\_file="master.000002", //主库的binlog日志

-> master\_log\_pos=448; //主库的目前的偏移量

3、启动slave进程

mysql> start slave;

mysql> show slave status\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Slave\_IO\_State: Waiting for master to send event

Master\_Host: 192.168.4.52

Master\_User: yaya

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: master.000002

Read\_Master\_Log\_Pos: 448

Relay\_Log\_File: host53-relay-bin.000002

Relay\_Log\_Pos: 317

Relay\_Master\_Log\_File: master.000002

Slave\_IO\_Running: Yes //IO线程yes表示启动

Slave\_SQL\_Running: Yes //SQL线程yes表示启动

...

//如果不是yes，用show slave stauts\G;下面由IO ERROR 和 SQL ERROR 的提示

//停掉 stop slave; 修改错误的配置，重新启动

[root@host53 ~]# cd /var/lib/mysql/

[root@host53 mysql]# ls

[root@host53 mysql]# cat master.info //连接主库的信息

[root@host53 mysql]# ls host53-relay-bin.\*

host53-relay-bin.000001

host53-relay-bin.000002 //中继日志文件

host53-relay-bin.index //中继日志文件索引

[root@host53 mysql]# cat relay-log.info //中继日志信息文件

不让从库同步主库的数据

临时不同步

mysql> stop slave;

把从库还原独立的库服务器

[root@host53 mysql]# cd /var/lib/mysql/

[root@host53 mysql]# rm -rf master.info

[root@host53 mysql]# rm -rf host53-relay-bin.\*

[root@host53 mysql]# rm -rf relay-log.info //删除四个文件就永久取消

[root@host53 mysql]# systemctl restart mysqld

mysql> show slave status;

Empty set (0.00 sec)

**主-从-从配置**

host52 主库

host53 从库 配置完成

host53 主库配置

启动binlog日志 和级联复制

用户授权

查看当前正在使用的binlog日志

[mysqld]

server\_id=53

log-bin=yaya88

binlog-format="mixed"

log\_slave\_updates //允许开启链式复制，

[root@host53 mysql]# systemctl restart mysqld

mysql> grant replication slave on \*.\* to yaya888@"%" identified by "123456";

mysql> show grants for yaya888@"%";

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

| Grants for yaya888@% |

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

| GRANT REPLICATION SLAVE ON \*.\* TO 'yaya888'@'%' |

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

mysql> show master status;

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

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

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

| yaya88.000002 | 440 | | | |

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

host54 从库配置

1、指定server\_id 并重启数据库服务

2、管理员登陆指定主数据库服务器的信息

3、启动slave进程

mysql> change master to

-> master\_host="192.168.4.53",

-> master\_user="yaya888",

-> master\_password="123456",

-> master\_log\_file="yaya88.000002",

-> master\_log\_pos=440;

mysql> start slave;

mysql> show slave status\G;

### 主从同步原理

主库有写入操作后,主库的mysql-dump会通知从库的IOthread线程

主库 从库

| IO thread

binlog 把主库binlog日志里的sql命令记录到本机的中继日志文件

SQL thread

执行本机中继日志文件里的sql命令，把数据写进本机。

mysql> show processlist; //显示当前数据库服务器正在运行的进程

主上有个命令binlog dump 给从上发送所有的binlog日志

从上的IO线程读取发送过来的binlog日志，记录到本机的中继日志

总结：

什么是mysql主从同步

mysql主从同步结构模式

mysql主从同步工作原理

主-从-从配置

主库配置选项 (主库的配置影响所有从库)

适用于 Master 服务器

选 项 用 途

binlog\_do\_db=name 设置 Master 对哪些库记日志

binlog\_ignore\_db=name 设置 Master 对哪些库不记日志

mysql> show master status;

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

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

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

| master.000004 | 154 | | | |

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

从库配置选项 (只对自己有影响)

适用于 Slave 服务器

选 项 用 途

log\_slave\_updates 记录从库更新,允许链式复制( A-B-C )

relay\_log=dbsvr2-relay-bin 指定中继日志文件名

replicate\_do\_db=mysql 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=test 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

**复制模式**

复制模式介绍

• 异步复制( Asynchronous replication ) //默认

– 主库在执行完客户端提交的事务后会立即将结果返给 客户端,并不关心从库是否已经接收并处理。

• 全同步复制( Fully synchronous replication )

– 当主库执行完一个事务,所有的从库都执行了该事务 才返回给客户端。

• 半同步复制( Semisynchronous replication )

– 介于异步复制和全同步复制之间,主库在执行完客户 端提交的事务后不是立刻返回给客户端,而是等待至少一个从库接收到并写到 relay log 中才返回给客户端

模式配置

• 查看是否允许动态加载模块

– 默认允许

mysql> show variables like "have \_ dynamic \_ loading";

mysql> show variables like '%dynamic%';

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

| Variable\_name | Value |

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

| have\_dynamic\_loading | YES |

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

命令行加载插件

– 用户需有 SUPER 权限

主库: mysql> INSTALL PLUGIN rpl\_semi\_sync\_master SONAME 'semisync\_master.so';

从库: mysql> INSTALL PLUGIN rpl\_semi\_sync\_slave SONAME 'semisync\_slave.so';

查看: mysql> SELECT PLUGIN\_NAME, PLUGIN\_STATUS FROM

INFORMATION\_SCHEMA.PLUGINS WHERE PLUGIN\_NAME

LIKE '%semi%';

启用半同步复制

– 在安装完插件后,半同步复制默认是关闭的

mysql> show variables like '%semi%enabled';

主: mysql> SET GLOBAL rpl\_semi\_sync\_master\_enabled = 1;

从: mysql> SET GLOBAL rpl\_semi\_sync\_slave\_enabled = 1;

查看: mysql> show variables like "rpl\_semi\_sync\_ %\_enabled";

配置文件永久启用半同步复制

– 命令配置临时配置,重启服务会失效

– 修改后需要重启服务

– 写在主配置文件 /etc/my.cnf 的 [mysqld] 下方

主:

plugin-load=rpl\_semi\_sync\_master=semisync\_master.so

rpl\_semi\_sync\_master\_enabled=1

从:

plugin-load=rpl\_semi\_sync\_slave=semisync\_slave.so

rpl\_semi\_sync\_slave\_enabled=1

**在高可用架构下, master 和 slave 需同时启动**

– 以便在切换后能继续使用半同步复制

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

## MySQL 读写分离(maxscale/mycat)

把客户端查询数据的请求和写入数据的请求分发给不同的数据库服务器处理

slect insert/update/delete

slave master

4.54 4.52

实现数据读写分离的方法

人肉分离：执行select sql命令时访问数据库服务器host54

执行insert sql命令时访问数据库服务器host52

select： mysql -h192.168.4.54 -u用户名 -p密码

insert： mysql -h192.168.4.52 -u用户名 -p密码

通过第三方软件提供的服务，实现数据读写分离

mysql中间件 mysql-proxy mycat maxscale

client:

select/insert： mysql -hproxy\_ip (虚拟ip) -u用户名 -p密码

mysql> select

mysql> insert

|

代理服务host53（接受客户端访问

数据库服务器的请求）使用maxscale

|

-------------------------------------------------------------------------

| |

select insert

192.168.4.54 192.168.4.52

slave master

**配置数据读写分离**

1、配置mysql一主（52）一从（53）同步结构

2、配置maxscale 服务器192.168.4.54

host53 关闭数据库服务

2.1 装包

[root@host53 mysql]# rpm -qa | grep maxscale

maxscale-2.1.2-1.x86\_64

2.2 修改配置文件

vim /etc/maxscale.cnf

[maxscale]

threads=auto //根据cpu核心数，自动启用线程数

[server1] //设置数据库的信息

type=server

address=192.168.4.52

port=3306

protocol=MySQLBackend

[server2] //设置数据库的信息

type=server

address=192.168.4.53

port=3306

protocol=MySQLBackend

[MySQL Monitor] //定义要监视的数据库服务器

type=monitor

module=mysqlmon

servers=server1, server2 //监视数据库服务是否运行，主从同步状态是否正常

user=scalemon //监视2个数据库时使用的用户密码

passwd=123456

monitor\_interval=10000

注释 read-only service的部分

[Read-Write Service] //定义实现读写分离的数据库服务器

type=service

router=readwritesplit

servers=server1,server2

user=maxscale

//客户端连接54时，使用用户名和密码，54代理使用maxscale用户去检查52和53是否有客户端连接用的用户

passwd=123456

max\_slave\_connections=100%

[MaxAdmin Service] //定义管理服务，默认

41 type=service

42 router=cli

注释 read-only listener的部分

[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=4019

2.3 根据配置文件的设置,主从上添加对应的授权用户scalemon和maxscale

只在主上创建就可以，从上自动同步

创建监控用户scalemon：

mysql> grant replication slave,replication client on \*.\* to

scalemon@'%' identified by "123456"; //这两个权限是用来监控数据库主从是否正常，数据库服务

创建路由用户maxscale：

mysql> grant select on mysql.\* to maxscale@'%' identified by "123456";

从上检查： select user from mysql.user where user in ("scalemon","maxscale")；

2.4 启动服务

2.4.1 测试数据库服务器的授权用户，在54上测试

[root@host53 ~]# mysql -h192.168.4.52 -uscalemon -p123456

[root@host53 ~]# mysql -h192.168.4.53 -uscalemon -p123456

[root@host53 ~]# mysql -h192.168.4.53 -umaxscale -p123456

[root@host53 ~]# mysql -h192.168.4.52 -umaxscale -p123456

2.4.2 启动maxscale服务

[root@host53 ~]# maxscale -f /etc/maxscale.cnf //指定配置文件启动

2.5查看服务状态信息

[root@host53 ~]# ps -C maxscale

PID TTY TIME CMD

3054 ? 00:00:00 maxscale

[root@host53 ~]# ss -anptul | grep maxscale

tcp LISTEN 0 128 :::4019 :::\* users:(("maxscale",pid=3054,fd=12)) 管理监听端口

tcp LISTEN 0 128 :::4006 :::\* users:(("maxscale",pid=3054,fd=11)) 读写分离服务端口

停止服务方法: pkill -9 maxscale

2.4.3 在本机连接管理端口

[root@host53 ~]# maxadmin -uadmin -pmariadb -P4019

MaxScale> list servers

Servers.

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

Server | Address | Port | Connections | Status

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

server1 | 192.168.4.52 | 3306 | 0 | Master, Running

server2 | 192.168.4.54 | 3306 | 0 | Slave, Running

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

3、测试配置：在客户端测试 192.168.4.254

3.1 在数据库服务器上添加访问数据时，使用的连接用户tom，密码123456

在52上创建 mysql> grant all on \*.\* to tom@"%" identified by "123456";

254:

连接 MySQL 代理服务器

mysql -h 代理的 IP 地址 -P 端口 -u 用户名 -p密码

mysql -h192.168.4.53 -P4006 -utom -p123456

mysql> 建库 建表 插入记录 查询记录

配置主机55 运行maxscale服务 提供数据读写分离服务

把接受到的数据

如果是一主多从,有多台读的服务器的时候.需要在maxscale管理机上修改配置文件如下

取消注释1:

[Read-Only Service]

type=service

router=readconnroute

servers=server2,server3

user=maxscale

passwd=123456

router\_options=slave

取消注释2:

[Read-Only Listener]

type=listener

service=Read-Only Service

protocol=MySQLClient

port=4008

[root@host52 ~]# ss -anptul | grep maxscale

tcp LISTEN 0 128 :::4019 :::\* users:(("maxscale",pid=2819,fd=15))

tcp LISTEN 0 128 :::4006 :::\* users:(("maxscale",pid=2819,fd=14))

tcp LISTEN 0 128 :::4008 :::\* users:(("maxscale",pid=2819,fd=12))

然后在多个从的服务器上,把数据库的一个变量打开

mysql> show variables like "%general\_log%";

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

| Variable\_name | Value |

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

| general\_log | OFF |

| general\_log\_file | /var/lib/mysql/host55.log |

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

2 rows in set (0.01 sec)

mysql> set global general\_log=1;

客户端访问时

实现读写分离时,用4006端口(这里的读服务器只会在第1个slave)

实现读的负载均衡时,使用4008端口

## mysql多实例

• 什么是多实例

– 在一台物理主机上运行多个数据库服务

• 为什么要使用多实例

– 节约运维成本

– 提高硬件利用率

配置多实例

配置步骤说明

– 安装支持多实例服务的软件包

– 修改主配置文件

– 根据配置文件做相应设置

– 初始化授权库

– 启动服务

– 客户端访问

• 具体配置

– 解压软件

– 修改目录名

– 修改 PATH 变量

[root@host50 ~]# tar -zxvf mysql-5.7.20-linux-glibc2.12-x86\_64.tar.gz

[root@host50 ~]# mv mysql-5.7.20-linux-glibc2.12-x86\_64 /usr/local/mysql

[root@host50 ~]# tail -1 /etc/profile

export PATH=/usr/local/mysql/bin:$PATH

[root@host50 ~]# source /etc/profile

配置文件参数说明

• 主配置文件 /etc/my.cnf

– 每个实例要有独立的数据库目录和监听端口号

– 每个实例要有独立的实例名称和独立的 sock 文件

[root@host50 ~]# mkdir /dir3307 /dir3308 //2个服务的数据库目录

[root@host50 ~]# vim /etc/my.cnf

[mysqld\_multi] // 启用多实例

mysqld=/usr/local/mysql/bin/mysqld\_safe // 指定进程文件的路径

mysqladmin=/usr/local/mysql/bin/mysqladmin // 指定管理命令路径

user=root // 指定调用进程的用户

[mysqld1] // 实例进程名称

port=3307 // 端口号

datadir=/dir3307 // 数据库目录 ,要手动创建

socket=/dir3307/mysqld3307.sock // 指定 sock 文件的路径和名称

pid-file=/dir3307/mysqld.pid // 进程 pid 号文件位置

log-error=/dir3307/mysqld.err // 错误日志位置

[mysqld2]

port=3308

datadir=/dir3308

socket=/dir3308/mysqld3308.sock

pid-file=/dir3308/mysqld.pid

log-error=/dir3308/mysql.err

• 初始化授权库:

– 会提示 root 用户登录的初始化密码

• 启动实例进程

• 停止实例进程

[root@localhost bin]# ./mysqld --user=mysql --basedir= 软件安

装目录 --datadir= 数据库目录 – initialize // 初始化授权库

[root@stu ~]# mysqld\_multi start 实例编号

// 启动实例进程

[root@localhost bin]# ./mysqld\_multi --user=root --password=

密码 stop 实例编号 // 停止实例进程

[root@host50 ~]# mysqld --user=mysql --basedir=/usr/local/mysql --datadir=/dir3307 --initialize

2018-06-02T03:28:44.598484Z 1 [Note] A temporary password is generated for root@localhost: :>dzYwc01W)p //登陆的第一个密码

[root@host50 ~]# mysqld --user=mysql --basedir=/usr/local/mysql --datadir=/dir3308 --initialize

2018-06-02T03:32:17.398505Z 1 [Note] A temporary password is generated for root@localhost: tNLBfgwQC1:\_

[root@host50 ~]# ls /dir3307 /dir3308 //初始化之后，两个目录下都有文件了

[root@host50 ~]# mysqld\_multi start 1

[root@host50 ~]# mysqld\_multi start 2

[root@host50 ~]# ss -anptul | grep 3307

[root@host50 ~]# ss -anptul | grep 3308

客户端访问

• 本机连接

– 使用初始密码连接

– 修改本机登陆密码

– 连接实例

[root@localhost bin]# ./mysql -uroot –p 初始密码 -S sock 文件

mysql> alter user user() identified by “ 新密码 ";

[root@localhost bin]# ./mysql –uroot –p 新密码 -S sock 文件

mysql -uroot -p':>dzYwc01W)p' -S /dir3307/mysqld3307.sock

修改密码为123456

mysql -uroot -p'tNLBfgwQC1:\_' -S /dir3308/mysqld3308.sock

修改密码为654321

[root@host50 ~]# mysqld\_multi --user=root --password654321 stop 2

[root@host50 ~]# ss -anptul | grep 3308

## MHA 集群

MHA 简介

• MHA ( Master High Availability )

– 由日本 DeNA 公司 youshimaton (现就职于 Facebook 公司)开发

– 是一套优秀的作为 MySQL 高可用性环境下故障切换和主从提升的高可用软件。

– 目前在 MySQL 高可用方面是一个相对成熟的解决方案

– 在 MySQL 故障切换过程中, MHA 能做到在 0~30 秒之内自动完成数据库的故障切换操作

– 并且在进行故障切换的过程中, MHA 能在最大程度上保证数据的一致性,以达到真正意义上的高可用。

MHA 组成

• MHA Manager (管理节点)

– 可以单独部署在一台独立的机器上管理多个 master-slave 集群,也可以部署在一台 slave 节点上。

• MHA Node (数据节点)

– 运行在每台 MySQL 服务器上。

最少三台服务器才能组建mha高可用集群

MHA 工作过程

• MHA Manager 会定时探测集群中的 master 节点,当master 出现故障时,

它可以自动将最新数据的 slave 提升为新的 master ,然后将所有其他的 slave

重新指向新的 master 。整个故障转移过程对应用程序完全透明。

– ( 1 )从宕机崩溃的 master 保存二进制日志事件( binlog events)

– ( 2 )识别含有最新更新的 slave

– ( 3 )应用差异的中继日志( relay log )到其他的 slave

– ( 4 )应用从 master 保存的二进制日志事件( binlog events )

– ( 5 )提升一个 slave 为新的 master ;

– ( 6 )使其他的 slave 连接新的 master 进行复制;

拓扑结构

client ----> mysql -hvip -uadmin -p123456

master51

|

|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| | | | |

slave52 slave53 slave54 slave55 mgm56

Manager

IP 规划

角色 IP地址 主机名

Master数据库服务器 192.168.4.51 master51

备用1master数据库服务器 192.168.4.52 master52

备用2master数据库服务器 192.168.4.53 master53

第1台slave服务器 192.168.4.54 slave54

第2台slave服务器 192.168.4.55 slave55

Mha\_manager服务器 192.168.4.56 mgm56

VIP地址 192.168.4.100

### 准备集群环境

配置mysql 一主多从 主从同步结构

(主库和2个备用主库开启半同步复制 且设置不自动删除本机的中继日志文件)见MAH配置.txt

所有数据节点主机之间可以互相无密码ssh连接。

管理主机56可以无密码ssh连接所有数据库服务器

配置MHA：

在所有主机上安装依赖的perl软件

在所有数据节点主机安装mha-node包

在在管理主机上安装mha-node和mha-manager包

编写管理主机56的管理服务的主配置文件 app1.cnf

测试配置：

1、测试所有主机之间是否可以无密码ssh连接

2、测试数据节点主机之间的一主多从 主从同步结构是否正常

2.1 根据主从同步的测试结果把规划的vip地址部署在主库51上

2.2 在客户主机254上访问vip地址100连接数据库服务

3、修改故障切换脚本 使用vip地址及把vip地址部署在哪块网卡上

4、启动管理服务,查看状态

5、客户连接vip地址访问数据库服务

6、测试mha的高可用配置：

当当前的主库51宕机后，客户仍然可以连接vip地址访问到数据库服务。

51宕机后 修复后启动mysqld服务，会变成独立的数据库，需要手动指定当前主库host52的从库

52自动变成主库，100的vip会自动变到host52上

[root@host52 ~]# ip addr show | grep 192.168.4.

inet 192.168.4.52/24 brd 192.168.4.255 scope global eth0

inet 192.168.4.100/24 brd 192.168.4.255 scope global secondary eth0:1

这时，51如果恢复了，需要手动把51配置成52的从，在56上改配置文件把51改成竞选主库

安装软件包

• 所有主机上安装 Perl 依赖包

[root@db108 share]# ls perl-\*.rpm

perl-Config-Tiny-2.14-7.el7.noarch.rpm

perl-Mail-Sender-0.8.23-1.el7.noarch.rpm

perl-MIME-Types-1.38-2.el7.noarch.rpm

perl-Email-Date-Format-1.002-15.el7.noarch.rpm

perl-Mail-Sendmail-0.79-21.el7.art.noarch.rpm

perl-Parallel-ForkManager-1.18-2.el7.noarch.rpm

perl-Log-Dispatch-2.41-1.el7.1.noarch.rpm

perl-MIME-Lite-3.030-1.el7.noarch.rpm

[root@db108 share]# yum -y install perl-\*.rpm

mysql> grant all on \*.\* to 'root'@'%' identified by '123456';

//在所有数据节点上授权监控用户,只需要在host51上授权，其他都自动同步了,给host56使用的

• 在所有数据库服务器上安装 mha-node 包

]# yum -y install perl-DBD-mysql perl-DBI

]# rpm -ivh mha4mysql-node-0.56-0.el6.noarch.rpm

• 在管理主机上安装 mha\_node 和 mha-manager包

]# yum -y install perl-DBD-mysql perl-DBI

]# rpm -ivh mha4mysql-node-0.56-0.el6.noarch.rpm

]# yum -y install perl-ExtUtils-\* perl-CPAN-\*

]#tar -zxf mha4mysql-manager-0.56.tar.gz

]#cd mha4mysql-manager-0.56

]# perl Makefile.PL

]# make

]# make install

配置 ssh 密钥对认证登陆

• 所有数据库服务器彼此之间互相以 root 用户 ssh 秘钥对认证登录

• 配置管理主机以 root 用户 ssh 秘钥对认证登录所有数据节点主机

相关命令

安装 manager 软件包 后产生的命令

命令 作用

masterha\_check\_ssh 检查 MHA 的 SSH 配置状况

masterha\_check\_repl 检查 MySQL 复制状况

masterha\_manger 启动 MHA

masterha\_check\_status 检测 MHA 运行状态

masterha\_master\_monitor 检测 master 是否宕机

### 配置 MHA 集群

配置 master 数据库服务器

master51 数据库服务器配置文件

vim /etc/my.cnf

[mysqld]

plugin-load =

"rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisyn

c\_slave.so"

rpl-semi-sync-master-enabled = 1

rpl-semi-sync-slave-enabled = 1

server\_id=51

log-bin=master51

binlog-format="mixed"

:wq

# systemctl restart mysqld

添加主从同步授权用户

mysql> set global relay\_log\_purge=off;// 不自动删除本机的中继日志文件,在51，52，53上设置

mysql> grant replication slave on \*.\* to repluser@"%" identified by "123456";//在51，52，53上设置

配置管理主机

管理节点56主机配置文件

]# cp mha4mysql-manager-0.56/bin/\* /root/bin/ 将命令拷贝到$PATH的一个目录下

]#mkdir /etc/mha\_manager/

[root@host114 mha4mysql-manager-0.56]# cp samples/conf/app1.cnf /etc/mha\_manager/

]# vim /etc/mha\_manager/app1.cnf

[server default]

manager\_workdir=/etc/mha\_manager

manager\_log=/etc/mha\_manager/manager.log

master\_ip\_failover\_script=/usr/local/bin/master\_ip\_failover // 自动failover 时候的切换脚本

ssh\_user=root

ssh\_port=22

repl\_user=repluser // 主从同步用户名

repl\_password=123456 // 主从同步密码

user=root // 连接高可用集群数据库服务器用户名

password=123456 // 密码

[server1]

hostname=192.168.4.51

port=3306

[server2]

hostname=192.168.4.52

port=3306

candidate\_master=1 // 设置为候选 master

[server3]

hostname=192.168.4.53

port=3306

candidate\_master=1 // 设置为候选 master

[server4]

hostname=192.168.4.54

port=3306

no\_master=1 // 不竞选 master

[server5]

hostname=192.168.4.55

port=3306

no\_master=1 // 不竞选 master

**测试集群配置**

测试 ssh 密钥对认证登陆

• 在管理节点上 通过 master\_check\_ssh 做 ssh 检查

]# /usr/local/bin/masterha\_check\_ssh --conf= 管理节点主机主配置文件

[root@host56 ~]# masterha\_check\_ssh --conf=/etc/mha\_manager/app1.cnf

//做这一步时需要把/etc/mha\_manager/app1.cnf里面的master\_ip\_failover\_script=/usr/local/bin/master\_ip\_failover注释掉

**测试主从同步状态**

• 在管理节点上监控复制环境: 通过 masterha\_check\_repl 脚本查看整个集群的状态

]# /usr/local/bin/masterha\_check\_repl --conf= 管理节点主机主配置文件

[root@host56 ~]# masterha\_check\_repl --conf=/etc/mha\_manager/app1.cnf

//做这一步时需要把/etc/mha\_manager/app1.cnf里面的master\_ip\_failover\_script=/usr/local/bin/master\_ip\_failover注释掉

根据测试结果把规划配置主机node52的VIP(一个机器可以有多个网卡,一个网卡可以有多个ip)

ifconfig eth0:1 192.168.1.100/24

[root@host56 mha\_manager]# cd /usr/local/bin/

[root@host56 bin]# wget ftp://192.168.4.254/mysql/master\_ip\_failover

//正常这个脚本在安装的源码包samples/scripts

[root@host56 bin]# chmod +x master\_ip\_failover

[root@host56 ~]# vim /usr/local/bin/master\_ip\_failover //自动切换

$new\_master\_password

);

my $vip = '192.168.4.100/24'; # Virtual IP

my $key = "1";

my $ssh\_start\_vip = "/sbin/ifconfig eth0:$key $vip";

my $ssh\_stop\_vip = "/sbin/ifconfig eth0:$key down";

GetOptions(

查看 VIP 地址

• 当主库服务器宕机后,在备用 1 主库数据库服务器上查看 VIP 地址

[root@host51 ~]# ip addr show | grep vip 地址

• 手动配置 vip 地址

[root@host51 ~]# ifconfig ethX:1 x.x.x.x/32

host51配置vip

[root@host51 ~]# ifconfig eth0:1 192.168.4.100/24

客户机进行连接测试

[root@room9pc01 ~]# mysql -uyaya -p123456 -h192.168.4.100

启动 MHA\_Manager

• 开启 MHA Manager 监控

– masterha\_manager --conf=/etc/mha\_manager/app1.cnf

– --remove\_dead\_master\_conf // 在 app1.cnf 文件里删除宕机的主库的信息 加了会删除，默认不加

– --ignore\_last\_failover // 忽略 .health 文件 //2次故障切换之间可以小于8小时. 不加这个选项两次切换之间必须大于8小时.

启动后会挂起终端

• 查看状态 : masterha\_check\_status

检查 mha 服务状态:

[root@host56 bin]# masterha\_check\_status --conf=/etc/mha/app1.cnf

• 停止服务 : masterha\_stop

停止 mha 服务

[root@host56 bin]# masterha\_stop --conf=/etc/mha/app1.cnf

我碰到的错误

开启服务后,自动停止.查看//etc/mha\_manager/mha.log ,看到一下内容

Bareword "FIXME\_xxx" not allowed while "strict subs" in use at /usr/local/bin/master\_ip\_failover line 97.

Execution of /usr/local/bin/master\_ip\_failover aborted due to compilation errors.

Wed Aug 15 16:53:30 2018 - [error][/usr/local/share/perl5/MHA/MasterMonitor.pm, ln226] Failed to get master\_ip\_failover\_script status with return code 255:0.

Wed Aug 15 16:53:30 2018 - [error][/usr/local/share/perl5/MHA/MasterMonitor.pm, ln424] Error happened on checking configurations. at /usr/local/bin/masterha\_manager line 50.

Wed Aug 15 16:53:30 2018 - [error][/usr/local/share/perl5/MHA/MasterMonitor.pm, ln523] Error happened on monitoring servers.

Wed Aug 15 16:53:30 2018 - [info] Got exit code 1 (Not master dead).

把97行注释掉就好了

**测试**

登录 mysql -uroot -p123456 -h192.168.1.100

->select @@hostname

node52

增删改查,再在slave端查看有没有同步

关闭node52的mysql服务,这时masterha\_manager监控停止.再次启动masterha\_manager后,主服务器自动指定到node53.

# MySQL 性能调优

MySQL 体系结构(由8个功能组件组成)

1、管理工具：软件安装后提供的命令

2、连接池：当数据库接收到连接请求时，检查并发数是否满了，资源是否由剩余，帐号密码是否正确

3、SQL接口：把sql命令传递mysql服务的线程去处理

4、分析器：分析sql命令是否正确

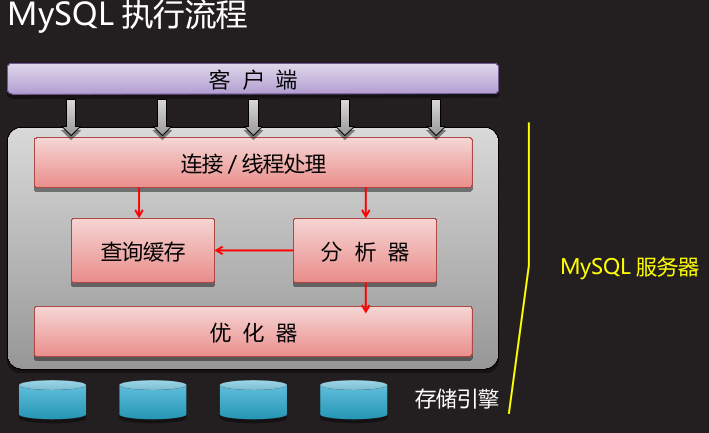
5、优化器：自动优化，以最省的方式去执行sql命令

6、查询缓存：存储曾经查找过的记录，查询缓存的空间是从物理内存里划分出来的

7、存储引擎：myisam/innodb

8、文件系统 /var/lib/mysql/db1/t1.\*(myisam/innodb)

ext3/2/4



请阐述一下你优化一台数据库服务器的优化思路？

考虑影响数据库服务器性能因素有那些：

1. 硬件配置低 导致处理速度 解决办法 查看硬件资源的使用率 升级硬件

CPU 内存 存储设备I/O

2. 网络速度 花钱买带宽 测试软件 测网络带宽

3. 提供数据库服务软件的版本低 （查看服务运行时的配置参数）

查看当前的运行参数（参数的值）

mysql> show variables;

mysql> show variables like "%关键字%";

mysql> show variables like "%time%";

mysql> show variables like "innodb\_lock\_wait\_timeout";

mysql> set innodb\_lock\_wait\_timeout = 100; //临时修改，永久修改在配置文件中

临时修改参数的值

mysql> set [global] 变量名 = 值

永久修改参数的值

vim /etc/my.cnf

[mysqld]

变量名 = 值

:wq

## 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 等待建立连接的超时秒数,默认 10 秒,只在登录时有效

wait\_timeout 等待关闭连接的不活动超时秒数,

默认 28800 秒( 8 小时) //建立连接后，等待用户执行命令的时间

mysql> show variables like "max\_connections";

mysql> set global max\_connections = 300;

查看已使用的连接数

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

mysql> flush status;

max\_used\_connections/max\_connections=85%

mysql> show variables like "connection\_timeout";

tcp三次握手的超时时间

## 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 允许保存在缓存中被重用的线程数量 //事先开启9个线程等待用户连接

table\_open\_cache 为所有线程缓存的打开的表的数量

## SQL 查询优化

• 查看缓存的大小

mysql> show variables like "query\_cache%";

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

| Variable\_name | Value |

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

| query\_cache\_limit | 1048576 |

| query\_cache\_min\_res\_unit | 4096 |

| query\_cache\_size | 1048576 |

| query\_cache\_type | OFF |

| query\_cache\_wlock\_invalidate | OFF | //对myisam存储引擎的表有效，控制脏读

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

pc1 ----> mysqld select name from user where name="jim";

----> memory

pc2 ----> mysqld select name from user where name="jim";

> memory name=jim

pc3 ----> mysqld update user set name="bob" where name=jim;

控制脏读的过程，当有人在数据库中有写操作时，mysql不从内存中读取数据，等写操作完毕后，从表中直接读取

• 查看当前的查询缓存统计

mysql> show global status like "qcache%";

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

| Variable\_name | Value |

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

| Qcache\_free\_blocks | 1 |

| Qcache\_free\_memory | 1031832 |

| Qcache\_hits | 90 | //记录在查询缓存里查找到的数据的次数

| Qcache\_inserts | 100 | //记录数据库服务器接受查询请求的次数，命中率90%

| Qcache\_lowmem\_prunes | 0 | //记录清除查询缓存空间里数据的次数。

| Qcache\_not\_cached | 25 |

| Qcache\_queries\_in\_cache | 0 |

| Qcache\_total\_blocks | 1 |

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

程序员编写的对数据库访问的sql命令复杂，导致数据库服务处理慢

启用mysql服务的慢查询日志，记录超过指定时间显示执行结果的sql命令

binlog日志

查询日志

错误日志 /var/log/mysqld.log

my.cnf

socket=/var/lib/mysql/mysql.sock //自己连接自己的数据接口文件

## 慢查询日志

记录慢查询

选 项 含 义

slow-query-log 启用慢查询

slow-query-log-file 指定慢查询日志文件

long-query-time 超过指定秒数(默认 10秒)的查询才被记录

log-queries-not-using-indexes 记录未使用索引的查询

• 调整服务配置

[root@dbsvr1 ~]# vim /etc/my.cnf

[mysqld]

.. ..

slow\_query\_log=1

slow\_query\_log\_file=mysql-slow.log

long\_query\_time=5

log\_queries\_not\_using\_indexes=1

[root@dbsvr1 ~]# service mysql restart

• 常用日志种类及选项

类 型 用 途 配 置

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 记录耗时较长或不使用索引的查询操作 slow-query-log

slow-query-log-file=

long-query-time=

在mysql.cnf 添加

重启服务

在/var/lib/mysql/

host51.log //genaral-log

host51-slow.log //slowlog

• 查看慢查询日志

– 使用 mysqldumpslow 工具

[root@dbsvr1 ~]# mysqldumpslow /var/lib/mysql/mysql-slow.log

Reading mysql slow query log from /var/lib/mysql/mysql-slow.log

Count: 1 Time=0.00s (0s) Lock=0.00s (0s) Rows=0.0 (0),

0users@0hosts

把超过10s的记录发给程序员

## 存储数据的网络结构部署有数据传输瓶颈；

调优思路总结

手段 具体操作

升级硬件 CPU 、内存、硬盘

加大网络带宽 付费加大带宽

调整mysql服务运行参数 并发连接数、连接超时时间、重复使用的

线程数 ........

调整与查询相关的参数 查询缓存、索引缓存 .......

启用慢查询日志 slow-query-log

网络架构不合理 调整网络架构

# 数据库安全

1） 初始化安全脚本

安装完MariaDB或MySQL后，默认root没有密码，并且提供了一个任何人都可以操作的test测试数据库。有一个名称为mysql\_secure\_installation的脚本，该脚本可以帮助我们为root设置密码，并禁止root从远程其他主机登陆数据库，并删除测试性数据库test。

1. **[**root@svr5 **~]**# systemctl status mariadb
2. //确保服务已启动
3. **[**root@svr5 **~]**# mysql\_secure\_installation
4. //执行初始化安全脚本

2）密码安全

手动修改MariaDB或MySQL数据库密码的方法：

1. **[**root@svr5 **~]**# mysqladmin **-**uroot **-**predhat password 'mysql'
2. //修改密码，旧密码为redhat，新密码为mysql
3. **[**root@svr5 **~]**# mysql **-**uroot **-**pmysql
4. MariaDB **[(**none**)]>**set password **for** root@'localhost'**=**password**(**'redhat'**)**
5. //使用账户登录数据库，修改密码
6. MariaDB **[(**none**)]>** select user**,**host**,**password from mysql**.**user**;**

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

| user     | host | password |

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

| root     | localhost     | \*84BB5DF4823DA319BBF86C99624479A198E6EEE9 |

| root     | 127.0.0.1     | \*84BB5DF4823DA319BBF86C99624479A198E6EEE9 |

| root     | ::1     | \*84BB5DF4823DA319BBF86C99624479A198E6EEE9 |

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

修改密码成功，而且密码在数据库中是加密的，有什么问题吗？问题是你的密码被明文记录了，下面来看看名为密码： history -c 清除历史命令

1. **[**root@svr5 **~]**# cat **.**bash\_history
2. mysqladmin **-**uroot **-**pxxx password 'redhat'
3. //通过命令行修改的密码，bash会自动记录历史，历史记录中记录了明文密码
4. **[**root@svr5 **~]**# cat **.**mysql\_history
5. set password **for** root@'localhost'**=**password**(**'redhat'**);**
6. select user**,**host**,**password from mysql**.**user**;**
7. flush privileges**;**
8. //通过mysql命令修改的密码，mysql也会有所有操作指令的记录，这里也记录了明文密码

另外数据库还有一个binlog日志里也有明文密码（5.6版本后修复了）。

怎么解决？

管理好自己的历史，不使用明文登录，选择合适的版本5.6以后的版本，

日志，行为审计（找到行为人），使用防火墙从TCP层设置ACL（禁止外网接触数据库）。

3）数据备份与还原

首先，备份数据库（注意用户名为root，密码为redhat）：

1. **[**root@svr5 **~]**# mysqldump **-**uroot **-**predhat mydb table **>** table**.**sql
2. //备份数据库中的某个数据表
3. **[**root@svr5 **~]**# mysqldump **-**uroot **-**predhat mydb **>** mydb**.**sql
4. //备份某个数据库
5. **[**root@svr5 **~]**# mysqldump **-**uroot **-**predhat **--**all**-**databases **>** all**.**sql
6. //备份所有数据库

接下来，还原数据库（注意用户名为root，密码为redhat）：

1. **[**root@svr5 **~]**# mysql **-**uroot **-**predhat mydb **<** table**.**sql            //还原数据表
2. **[**root@svr5 **~]**# mysql **-**uroot **-**predhat mydb **<** mydb**.**sql            //还原数据库
3. **[**root@svr5 **~]**# mysql **-**uroot **-**predhat **<** all**.**sql                    //还原所有数据库

4）数据安全

在服务器上（192.168.4.5），创建一个数据库账户：

1. **[**root@svr5 **~]**# mysql **-**uroot **-**predhat
2. //使用管理员，登陆数据库
3. MariaDB **[(**none**)]>** grant all on **\*.\*** to tom@'%' identified by '123'**;**
4. //创建一个新账户tom, %匹配任意字符串，表示所有主机。

使用tcpdump抓包（192.168.4.5）

1. **[**root@svr5 **~]**# tcpdump **-**w log **-**i any src or dst port **3306**
2. //抓取源或目标端口是3306的数据包，保存到log文件中

客户端（192.168.4.100）从远程登陆数据库服务器（192.168.4.5）

1. **[**root@client **~]**# mysql **-**utom **-**p123 **-**h **192.168.4.5**
2. //在192.168.4.100这台主机使用mysql命令登陆远程数据库服务器（192.168.4.5）
3. //用户名为tom，密码为123
4. MariaDB **[(**none**)]>** select **\*** from mysql**.**user**;**
5. //登陆数据库后，任意执行一条查询语句

回到服务器查看抓取的数据包

1. **[**root@svr7 **~]**# tcpdump **-**A **-**r log
2. //使用tcpdump查看之前抓取的数据包，很多数据库的数据都明文显示出来

如何解决？

可以使用SSH远程连接服务器后，再从本地登陆数据库（避免在网络中传输数据，因为网络环境中不知道有没有抓包者）。

或者也可以使用SSL对MySQL服务器进行加密，类似与HTTP+SSL一样，MySQL也支持SSL加密（确保网络中传输的数据是被加密的）。