**常见的sql操作指令**

**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）**

**mysql日志：binlog日志（增量备份日志，主从同步所需日志）文件名是：主机名-bin.index 主机名-bin.00000x**

**错误日志（默认启用，记录mysql启动和运行中产生的错误信息）文件名是：mysqld.log**

**查询日志（记录所有的sql命令）文件名是：主机名.log**

**慢查询日志（只记录超时时间显示查询结果的sql命令）：主机名-slow.log**

**中继日志（主从同步时，从开启的日志）文件名是：master.info relay-log.info 主机名-relay-bin.\***

**事务日志 文件名是：ibdata1 ib\_logfile0 ib\_logfile1**

**mysql初始配置**

修改root口令，默认的数据库管理帐号root，仅允许localhost访问，首次登陆密码随机生成，存在错误文件里（/var/log/mysqld.log）

mysql -uroot -p随机密码

mysql>set globel validate\_password=policy=0;//只验证长度（0只验证长度，1验证长度、数字、大/小写、特殊字符，2验证长度、数字、大/小写、特殊字符和字典文件）

mysql>set global validate\_password\_length=6;//修改密码长度，默认值是8个字符

mysql>alter user root() identified by “123456”;//修改登录密码

mysql -uroot -p123456//使用新密码登录

**mysql数据类型**

常见的信息种类：数值型、字符型、枚举型、日期时间型

数值型（整型、浮点型）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | -231~231-1 | 0~224-1 | 中整数 |
| INT | 4字节 | -231~231-1 | 0~223-1 | 大整数 |
| BIGINT | 8字节 | -263~263-1 | 0~264-1 | 极大整数 |
| FLOAT | 4字节 |  |  | 单精度浮点数 |
| DOUBLE | 8个字节 |  |  | 双精度浮点数 |
| DECIMAL | 对DEVIMAL(M,D)，其中M为有效位数、D为小数位数、M赢大于D，占用M+2字节 | | | |

整数型：

定义格式：类型 int

使用unsigned修饰时，对应的字段只保存正数，也就是无符号数

如果存小数，会对小数位四舍五入

数值不够指定宽度时，在左边填空格补位

宽度仅是显示宽度，存数值的大小由类型决定

使用关键字zerofill时，填0代替空格补位

浮点型

定义格式：float(总宽度,小数位) float(m,n) double(m,n)

当字段值与类型不匹配时，字段作为0处理

数值超出范围时，仅保存最大/最小值

字符型

默认宽度为1，varchar必须指定宽度，否则为语法错误，char可以不指定宽度

char定长：格式：char(字符数) char(6)

最大长度255字符

不够指定字符数时在右边用空格补齐

字符判断超出时，无法写入数据

varchar变长：格式：varvhar(字符数) varchar(6)

0-65532个字符

按数据实际分配存储空间

字符数段超出时，无法写入数据

大文本类型：text/blob（二进制文件，音频文件，不需要指定宽度）

格式：text/blob

字符数大于65535存储时使用

枚举类型，最多大于64个

从给定集合中选择单个值，enum

定义格式：enum(值1,值2,值N) enum(“girl”,”boy”,”in”)

从给定集合中选择一个或多个值,set

定义格式：set(值1,值2，值N) set(“aa”,”bb”,”cc”)

日期时间类型

日期时间，datetime

占用8个字节

范围：1000-01-01 00：00：00.000000~9999-12-31 23.59.59.999999

赋值：20180202111100，表示2018年2月2日11点11分11秒

日期时间，timestamp

占用4个字节

范围：1970-01-01 00:00:00.000000-2038-01-19 03:14:07:999999

赋值：20180202111100，表示2018年2月2日11点11分11秒

日期，date

占用4个字节

范围：0001-01-01~9999-12-31

赋值：20181232，表示2018年12月31日

年份，year

占用1个字节

范围：1901~2155

赋值：2018，表示2018年

时间，time

占用3个字节

格式：HH:MM:SS

赋值：111111，表示11点11分11秒

关于日期时间字段

当未给timestamp字段赋值时，自动以当前系统时间赋值，而datetime字段默认值为空

year年份的处理

默认用4位数字表示

当只用2位数字表示时，01~69视为2000~2069，而70~99视为1970~1999

时间函数

|  |  |
| --- | --- |
| 类型 | 用途 |
| now() | 获取系统当前日期和时间 select now()； |
| year() | 获取指定日期时间的年 select year(20190101)； |
| sleep(N) | 休眠N秒 select sleep(2)； |
| curdate() | 获取当前的系统日期select curdate(); |
| curtime() | 获取当前的系统时刻select curime(); |
| month() | 获取指定时间中的月份select month(now()); |
| date() | 获取指定时间中的日期select date(20190101); |
| time() | 获取指定时间中的时刻select time(now()); |

无需库、表，可直接调用，使用select指令输出函数结果

mysql>select now(),sysdate(),curldate();

**数据库基本管理**

数据存储流程：连接数据库服务器->建库->建表->插入记录->断开连接

连接mysql服务器：mysql [-h 服务器 -u用户名 -p密码 数据库]

quit或exit退出

注意事项：操作指令不区分大小写（密码、变量值除外）

每条sql指令以分号结束或分割

不支持tab补齐

\c可废弃当前编写错的操作指令

**库管理命令**

show databases;//显示已有的库

use 库名;//切换库

select database();//显示当前所在的库

create database 库名;//创建新库

show tables;//显示已有的表

drop database 库名;//删除库

数据库的命名规则

可以使用数字/字母/下划线，但不能纯数字

区分大小写，具有唯一性

不可使用指令关键字、特殊字符

**表管理命令**

新建指定名称的表

create table 库名.表名(

字段1 字段类型（宽度） 约束条件,

字段2 字段类型（宽度） 约束条件,

......

字段N 字段类型（宽度） 约束条件

)default charset=utf8;(设置格式为utf8)

desc 表名;//查看表结构（表的字段，有字段名|字段类型|是否为空|是否为主键|默认值|描述信息的相关信息）

select \* from 表名;//查看表记录（\*字段）

drop table 表名;//删除表

show create table 表名;//查看创建表的命令

**记录管理命令**

select \* from 库名.表名;//查看全部表记录

insert into 表名 values(值列表1),(值列表2);//插入表记录

update 表名 set 字段=值;//修改全部表记录

delete from 表名;//删除全部表记录

**约束条件**

NULL允许为空，默认为空，not null不允许为空

key索引类型，默认没有，需要单独指定

default，默认为null

extra额外设置，是否配置自动增长

**修改表结构**

基本用法：alter table 表名 执行动作;

常用动作：add添加字段

modify修改字段

change修改字段名

drop删除字段

rename重命名表

添加新字段

基本用法：alter table 表名 add 字段名 类型（宽度） 约束条件 after/first;

修改字段类型（可以修改类型和约束条件）

基本用法：alter table 表名 modify 字段名 类型（宽度） 约束条件 after/first;

注意：新类型如果与已存数据冲突，不允许修改；修改过程中，修改的写出来，不变的原样抄下来，否则还原为系统默认

修改字段名

基本用法：alter table 表名 change 字段名 新字段名 类型（宽度） 约束条件：

当跟新类型和约束条件时，也可修改字段类型

删除字段

基本用法：alter table 表名 drop 字段名;

表中有多条记录时，此时列的此字段的值都会被删除

修改表名

基本用法：alter table 表名 rename 新表名;

表对应的文件名也会改变

**mysql键值，约束如何给字段赋值**

mysql索引（类似于书的目录），常用类型包括：btree二叉树算法、b+tree、hash

优缺点：

优点：通过创建唯一性索引，可以保证数据库表中的每一行数据的唯一性

可以加快数据的查询速度

缺点：当对标中的数据进行增加、删除和修改的时候，索引也要动态维护，降低了数据的维护速度

索引需要占用物理空间

减慢写入数据的速度

键值类型

index：普通索引

primary key：主键

foreign key：外键

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，可以赋NULL值

经常把做查询的字段设置为Index字段

index字段的key标志是MUL

建表的时候指定索引字段

create table aa(

name char(4),

class char(15),

sex enum(“boy”,”girl”) not null default “boy”,

index(name),index(class)

);

在已有的表中设置index字段

create index 索引名 on 表名(字段名)；

删除指定表的索引字段

drop index 索引名 on 表名;

查看表的索引信息

show index from 表名;

primary key主键

使用说明

一个表中只能有一个primary key主键

对应的字段值不允许有重复，切不允许为NULL值

如果有多个字段都作为主键，称为符合主键，必须一起创建

主键字段的key标志是PRI

通常与auto\_increment连用

经常把表中能够唯一标识记录的字段设置为主键字段

建表时指定主键字段

create table aa(

name char(4) primary key,

class char(15),

);

在已有表中设置primary key(字段名)

alter table 表名 add primary key(字段名);

移除表中的primary key字段

alter table 表名 drop primary key;

移除主键前，如果有自增属性，必须先去掉

复合主键（一个以上的主键，这两个字段的值不同时，就不算重复）

在新表中添加复合主键

create table aa(

name char(4),

class char(15),

sex enum(“boy”,”girl”) not null default “boy”,

primary key(name,class)

);

删除复合主键

alter table aa drop primary key;

删除已有表中添加复合主键

alter table aa add primary key(cip,port);

自增长

auto\_increment字段值自增长，每次自增加一（必须是主键且是数值类型）

create table aa(

name char(4) primary key auto\_increment,

class char(15),

);

foreign key外键

概念：让当前表字段的值在另一个表中字段值的范围内选择

外键的条件：

表的存储引擎必须是innodb

字段类型要一致

被参照组字段必须要是索引类型的一种（主键）

基本用法

foreign key（表A的字段名）references 表B（字段名） on update cascade on delete cascade

删除外键字段

alter table 表名 drop foreign key 外键名;

如果有外键，不能删除被参考的主键的表和字段

**数据导入导出**

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”;

修改目录及查看修改结果：

mkdir /email;chown mysql /email

vim /etc/my.cnf

[mysqld]

secure\_file\_priv=”/email”

systemctl restart mysqld

数据导入（把系统的文件内容存储到数据库里面）

基本用法

load data infile “目录名/文件名” into table 表名

fields terminated by “分隔符” //列以什么分隔

lines terminated by “\n”; //行以什么分隔

注意事项

字段分隔符要与文件一致

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保护机制

数据导出（表记录存储到系统文件里）

基本用法

sql的查询结果 into outfile “目录名/文件名”

fields terminated by “分隔符”

lines terminated by “\n”;

注意事项

导出的内容由sql查询语句决定

导出的是表中的记录，不包括字段名

禁用selinux

**管理表记录**

增加表记录

语法格式

格式1：添加1条记录，给所有字段赋值

insert into 表名 values(字段值列表);

格式2：添加N条记录，给所有字段赋值

insert into 表名 vlues(字段列表1),(字段列表2)...(字段列表N);

格式3：添加一条记录，给指定字段赋值

insert into 表名(字段名列表) values(字段值列表);

格式4：添加N条记录，给指定字段赋值

insert into 表名(字段名列表，中间用逗号隔开) vlues(字段列表1),(字段列表2)...(字段列表N);

注意事项

字段值与字段类型相匹配

对于字符类型的字段，要用双或单引号括起来

一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

更新表记录

语法格式

格式1，更新表内所有记录

update 表名 set 字段1=字段1值,字段2=字段2值...;

格式2，只更新符合条件的部分记录

upodate 表名 set 字段1=字段1值，字段2=字段2值... where 条件表达式;

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，要用单或双引号括起来

若不使用where限定条件，会更新所有记录

限定条件时，只更新匹配条件的记录

删除表记录

语法格式

格式1，仅删除符合条件的记录

delete from 表名 where 条件表达式;

格式2，删除所有表记录

delete from 表面;

查看表记录

语法格式

格式1：

select 字段1,...,字段N from 库.表 where条件表达式;

格式2

select 字段1,...,字段N from 表名;

注意事项

使用\*可匹配所有字段

指定表名时，可采用 库名.表名 的形式

**mysql存储引擎**

8个结构构成：管理工具、连接池、sql接口、分析器、优化器、缓存、存储引擎、硬盘

三个事务日志

ibdata1：存放错误的命令

ib\_logfile0：存放正确的命令

ib\_logfile1：存放正确的命令

**（每执行一条sql命令会定义一个编号是LSN（日志序列号））**

常见的存储引擎：innodb myisam

列出可用的存储引擎：show enignes或show engines\G

设置默认存储引擎

vim /etc/my.cnf

[mysqld]

default-storage-engine=存储引擎名称

systemctl resatrt mysqld

创建表时指定存储引擎

create table aa(aa int)engine=innodb;

修改表的存储引擎

alter table bb engine=myisam;

innodb存储引擎主要特点

支持行级锁定

支持事务、事务回滚、外键（ibdata1 ib\_logfile0 ib\_logfile1三个是事务日志）

相关表文件：.frm：存放表结构

.ibd：存放数据+索引

myisam存储引擎特点

支持页级锁定

不支持事务、事务回滚、外键

相关表文件：.frm：存放表结构

.MYI：存放索引信息

.MYD：存放数据

锁机制

表级锁：直接对整张进行加锁

行级锁：只锁定某一行

页级锁：（给内存里的东西加锁，1M就是一页）：对整个页面（Mysql管理数据的基本存储单位）进行加锁

读锁（共享锁）：支持并发读操作（进行select操作是读锁）

写锁（互斥锁、排它锁）：是独占锁，上锁期间其他线程不能读表或写表

事务特性

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，要不全部失败

一致性：事务操作的前后，表中的记录没有变化

隔离性：事务操作是相关隔离不受影响的

持久性：数据一旦提交，不可更改，永久改变表数据

**匹配条件（where后面）**

数值比较（字段类型必须是数据数值类型）where 字段名 符号 数字

= 等于 >、>= 大于、大于或等于 <、<= 小于、小于或等于 != 不等于

字符比较/匹配空/非空（字符比较时，字段类型必须是字符类型，字符必须用双引号括起来）where 字段名 符号 “值”

= 相等 != 不相等 is null 匹配空 is not null 匹配非空

逻辑匹配（多个判断时使用）where 条件A 符号 条件B

or 逻辑或，多个条件满足一个即可 and 逻辑与，多个条件都得成立 ！非 () 提高优先级

范围内匹配/去重显示（匹配范围内的任意一个数值）where 字段名 in (值列表)

in(值列表) 在...里 not in(值列表) 不在...里 betweent 数字1 and 数字2 在..之间.. distinct字段名 去重显示，不显示重复值，只显示第一次出现的

模糊查询

select \* from 表名where 字段名 like ‘通配符’（\_匹配单个字符、%匹配0-N个字符）

正则匹配

where 字段名 regexp ‘正则表达式’

正则元字符^ $ . [] \* |

四则运算（运算操作 字段必须是数值类型）select 字段名 运算操作 from 表

+ - \* / %

**操作查询结果**

聚集函数（字段类型必须是数值型的）select 函数 from 表名 where 匹配条件

avg（字段名） //统计字段的平均值

sum（字段名） //统计字段之和

min（字段名） //统计字段的最小值

max（字段名） //统计字段的最大值

count（字段名） //统计字段值个数

查询结果排序（默认升序）

sql的查询结果 order by （通常是数值类型字段） [asc|desc];

查询结果分组

sql的查询结果 group by 字段名（通常是字符类型字段）

查询结果过滤

sql的查询结果 having 表达式;

sql的查询结果 where 条件 having 条件表达式;

sql的查询结果 group by 字段名 having 条件表达式;

限制查询结果显示行数

sql的查询结果 limit N; //显示查询结果前N条记录

sql的查询结果 limit N,M; //显示指定范围内的查询记录（从0开始）

sql的查询结果 where 条件查询 limit 3; //显示查询结果前3条记录

sql的查询结果 where 条件查询 limit 3,3; //从第4条开始，共显示3条

**多表查询（查询种类：交叉连接 自然连接 内连接 外连接（左连接 右连接））**

复制表（可以备份表；可以快速建表；键值不会被复制）

将源表xxx复制为新表yyy

create table yyy select \* from xxx;

将指定的查询结果复制为新表

create table zzz sql查询语句结果;

快速建表（只复制表结构，不复制表数据）

create table vvv select \* from xxx where false;

将源表改名

alter table xxx rename to yyy;

多表查询

格式1：select 字段名列表 from 表a,表b;

格式2：select 字段名列表 from 表名a,表名b where 条件;

where嵌套查询

使用where嵌套查询，把内层查询结果作为外层的查询条件

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);

左连接查询（当条件成立时，以左表为主显示查询结果，左表记录全显示出来）

语法格式：

select 字段名列表 from 表a left join 表b on 条件表达式

右连接查询（当条件成立时，以右表为主显示查询结果）

语法格式：

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

**mysql管理工具Phpmyadmin安装**

yum -y install httpd php php-mysql

systemctl start httpd;

tar -xf phpmyadmin.tar.gz -C /var/www/html

cd /var/www/html

mv phpmyadmin aaa/

cp phpmyadmin/config.sample.inc.php phpmyadmin/config.inc.php

vim config.inc.php

$cfg['blowfish\_secret'] = 'aaa';

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

**用户授权及权限撤销**

用户授权：在数据库服务器上添加可以连接的用户，并可以设置添加用户的访问权限

默认只有数据库管理在本机登录

密码恢复及设置

恢复管理员密码

操作过程

vim /etc/my.cnf

[mysqld]

skip-grant-tables

systemctl restart mysqld

mysql

update mysql.user set authentication\_string=password(“888888”) where user=”root” and host=”localhost”

flush privileges;

quit

vim /etc/my.cnf

[mysqld]

#skip-grant-tables

systemctl restart mysqld

设置管理员密码

mysqladmin -uroot -p password “新密码”

enter password://输入旧密码

重设用户密码

授权用户连接后修改密码

set password=password(“新密码”);

管理员重置用户连接密码

set password for 用户名@”客户端地址”=password(“新密码”);

创建授权

mysql授权库和表信息

user表，存储授权用户的访问权限

db表，存储授权用户对数据库的访问权限

tables\_priv表，存储授权用户对表的访问权限

columns\_priv表，存储授权用户对字段的访问权限

grant配置授权

基本用法

grant 权限列表 on 库名.表名 to 用户名@”客户端地址” identfied by ‘密码’ [with grant option]（是否有授权权限）

权限列表

all：匹配所有权限

select,update,insert.....

sellect,update(字段1,...,字段N)

uage：表示无权限

客户端地址

%：匹配所有主机

192.168.1.%：匹配一个网段

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

svr.tarena.com：匹配指定域名的单个主机

查看用户授权

用户查看自己的授权：show grants;

管理员查看其它用户的授权：show grants for 用户名@”客户端地址”;

撤销授权

revoke 权限列表 on 库名.表名 from 用户名@”客户端地址”;

删除用户

drop user 用户名@”地址”;

**mysqldump备份/恢复**

备份方式

物理备份

冷备：cp tar

逻辑备份

mysqldump

mysql

备份策略

完全备份：备份所有数据

增量备份：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

物理备份

物理备份及恢复（innodb不适合物理备份，适合mysiam）

备份操作

cp -rp /var/lib/mysql/数据库 备份目录/文件名

tar -zxcf 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 -zxf xxx.tar.gz -C /var/lib/mysql/数据库名

逻辑备份（执行备份操作时，根据备份的库产生对应的sql命令，把sql命令存储到指定的文件里）

逻辑备份与恢复

备份操作：mysqldump -uroot -p密码 库名 > 路径/xxx.sql

恢复操作：mysql -uroot -p密码 库名 < 路径/xxx.sql

mysql>source 路径/xxx.sql

库名表示方式：--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

**实时增量备份/恢复binlog日志（从热备份的角度说 MyISAM是表级锁 innodb是行级锁 热备份的时候是要锁表的 MyISAM不支持行级锁 所以也就无法实现增量的功能 增量备份调用的事务日志 而MyISAM是不支持事务的 innodb支持事务 且特有binlog这个东西）**（二进制日志，记录除查询之外的sql命令，记录所有更改数据的操作）（可用于增量备份的恢复，配置mysql主从同步的必要条件）

binlog相关文件

主机名-bin.index 记录已有日志文件名

主机名-bin.000001 第一个二进制日志（大于500M会生成第二个）

主机名-bin.000002 第二个二进制日志

binlog记录sql命令的方式

偏移量：sql命令长度

时间点：执行sql命令对应的系统时间

启用binlog日志

vim /etc/my.cnf

[mysqld]

log\_bin //启用binlog日志

server\_id=100 //指定id值

max\_binlog\_size=数字m（默认500M） //文件到达多少新建文件

log-bin[=dir/name]（指路径的时候必须指文件名，要不只指文件名，要不什么都不指）

binlog\_format=”mixed”

systemctl restart mysqld

查看日志当前记录格式

mysql>show variables like “binlog\_format”;

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语句上下文相关内容，仅保留哪条记录被修改
3. mixed是以上两种格式的混合使用

手动生成新的日志文件

1. 重启mysql服务（一般线上的不用）
2. 执行sql操作 mysql>flush logs;
3. mysqldump --flush-logs
4. mysql -uroot -p密码 -e ’flush logs’（-e是执行简单的sql命令）

清理binlog日志（在sql命令行操作）

删除早于指定版本的binlog日志

purge master logs to “binlog文件名”;

删除所有binlog日志，重建新日志

reset master;

分析binlog日志

使用mysqlbinlog工具

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（起始偏移量）

--stop-position=数字（最后的偏移量）

binlog恢复数据

基本思路

使用mysqlbinlog提取历史sql操作

通过管道交给 mysql命令执行

mysqldbinlog [选项] mysql-bin.000001 | mysql -uroot -p密码

**innobackupex备份/恢复**

备份目录配置文件

xtrabckup\_checkpoints当前备份目录的数据对应的日志信息的日志序列号是多少

xtrabackup\_logfile记录已存的sql命令

ibdata1记录未提交的命令

backup-my.cnf占用系统资源参数

.meta和.delta代表的是增量备份数据

安装percona

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 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

完全备份与恢复（恢复时要求数据库目录是空的，所以备份指定库时，系统自带的库也要备份）

备份

备份当前所有库所有表，不生成时间戳子目录

innobackupex --user root --password 123456 /allback --no-timestamp

备份指定库

innobackupex --user root --password 123456 --databases=”buydb” /buy --no-timestamp

备份多个库，中间用空格区分

innobackupex --user root --password 123456 --databases=”buydb mysql” /buy --no-timestamp

备份指定表

innobackupex --user root --password 123456 --databases=”buydb.a” /buy --no-timestamp

恢复步骤

1. 要求数据库目录为空

rm -rf /var/lib/mysql/\*

1. 重做日志--apply-log

innobackupex --user root --password 123456 /allback --apply-log

1. 恢复数据--copy-back

innobackupex --root root --password 123456 /allback --copy-back

chown -R mysql.mysql /var/lib/mysql

1. 重启数据库

systemctl restart mysqld

增量备份与恢复

备份步骤

1. 有一次完整备份

innobackupex --user root --password 123456 --databases=”mysql sys performance\_schema buydb” /fullbuydb --no-timestamp

1. 再进行增量备份

--incremental 目录名 增量备份的目录名

--incremental-basedir=上次备份目录指定的文件名

innobackupex --user root --password 123456 --databases=”mysql sys performance\_schema buydb” --incremental /new1dir --incremental-basedir=/fullbuydb --no-timestamp

增量恢复

重做--apply-log 合并--redo-only --incremental-dir=增量备份目录

1. 重做并合并日志

第一次重做日志

innobackupex --user root --password 123456 --databases=”mysql sys performance\_schema buydb” --apply-log --redo-only /fullbuydb --no-timestamp

将第一次增量备份合并进去

innobackupex --user root --password 123456 --databases=”mysql sys performance\_schema buydb” --apply-log --redo-only /fullbuydb --incremental-dir=/new1dir/ --no-timestamp

1. 清空数据库目录

rm -rf /var/lib/mysql/\*

1. 停止服务

systemctl stop mysqld

1. 拷贝数据（恢复数据）

innobackupex --copy-back /fullbuydb

chown -R mysql.mysql /var/lib/mysql

1. 启动服务

systemctl start mysqld

在完整备份中恢复单个表（不需要清空整个数据库）

完整备份

innobackupex --user root --password 123456 --databases=”buydb” /buydbfull2 --no-timestamp

恢复

--export 导出表信息

删除表空间mysql>alter table 库名.表名 discard tablespace;

导入表空间mysql>alter table 库名.表名 import tablespace;

1. 从备份数据中导出表信息（会在备份数据库目录下生成exp、cfg文件）

innobackupex --user root --password 123456 --databases=”buydb” --apply-log --export /buydbfull2

mysql>drop table b;

1. 创建删除的表（表结构与之前的一致）

mysql>cerate table b(name char(10));

1. 删除表空间

mysql>alter table b discard tablespace;

1. 把导出的表信息文件拷贝到数据库目录下

cp /buydbfull2/buydb/b.{cfg,exp,ibd} /var/lib/mysql/buydb

chown mysql.mysql b.\*

1. 导入表空间

mysql>alter table buydb.b mysql.mysql b.\*

1. 查看数据

show tables;

innobackupex基本选项

--host 主机名（本机操作可以省略）

--user 用户名

--port 端口号（本机可以省略）

--password 密码

--databases 数据库

--databases=”库名” 单个库

--databases=”库1 库2” 多个库

--databases=”库.表” 单个表

--no-timestamp 不用日期命名备份文件存储的子目录

--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，指定上一次备份的数据存储的目录

--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

--export 导出表信息

import 导入表空间

discard 删除表空间

**mysql主从同步**

**角色分为2种：**

**数据库服务器 做主master库：被客户顿存储数据访问的库**

**数据库服务器，做从slave库：同步主库的数据到本机**

**作用：实现数据的自动备份**

从库的sql线程执行本机中继日志里的sql命令，不会记录在binlog日志文件里面

相关文件

master.info主库信息

relay-log.info中继日志信息

主机名-relay-bin.xxxxxx中继日志

主机名-relay-bin.index索引文件

常用配置选项

主库选项

适用master服务器（对所有的从库有影响）

binlog\_do\_db=name设置master对哪些库记日志（允许从库同步哪些日志）

binlog\_ignore\_db=name设置master对哪些库不记日志（不允许从库同步哪些日志）

从库选项

适用slave服务器（只对本机有影响）

log\_slave\_updates记录从库更新，级联复制（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，其他库将忽略，此选项可以设置多条（省略时复制所有库，多个库用逗号分隔）

replicate\_ignore\_db=test不复制哪些库，其他库将被忽略，两者选一种即可

主从同步原理

master，记录数据更改操作

启用binlog日志

设置binlog日志格式

设置server\_id

slave运行2个线程

slave\_io：复制master主机binlog日志文件里的sql到本机的relay-log文件里

slave\_sql：执行本机relay-log文件里的sql语句，重现master数据操作

构建主从基本思路

前提

确保数据相同

从库上必须要有主库上的数据

配置主服务器

启用binlog日志、授权用户、查看当前使用的日志

配置从服务器

设置server\_id、指定主库信息

测试配置

主从同步模式

基本应用

单向复制：主-->从

扩展应用

链式复制：主从从

互为主从：主<-->

一主多从：从<--主-->从

配置一主一从

配置主服务器

调整运行参数

启用binlog日志及允许同步

vim /etc/my.cnf

[mysqld]

log\_bin=日志名

server\_id=id值

binlog\_format=”mixed”

systemctl restart mysqld

授权用户

mysql>grant replication slave on \*.\* to 用户名@”从库ip地址” identified by “密码”;（默认不允许对单个库授权）

查看master状态（记录当前的日志文件名、偏移位置）

mysql>show master status\G;

配置从服务器

调整运行参数（指定server\_id不允许与主库server\_id值相同）

vim /etc/my.cnf

[mysqld]

server\_id=id值

systemctl restart mysqld

指定主库信息（首次是指定，第二次是修改）

msql>change master to

->master\_host=’192.168.4.10’,//主库ip地址

->master\_user=’replicater’,//主库授权用户名

->master\_password=’pass123’,//授权用户密码

->master\_log\_file=’dbsvr1-bin.000004’,//日志文件

->master\_log\_pos=334;

mysql>star slave;//启动slave进程

master信息会自动保存到/var/lib/mysql/master.info文件，以后如果要改master信息时，应先stop slave;

查看slave状态（确认IO线程、sql线程都已运行）

mysql>show slave status\G;

测试配置

在主库上进行操作

让从库临时不同步主库上的内容

mysql>stop slave;

在从库修改主库信息

mysql>stop slave;

mysql>change master to 选项=”值”,选项=”值”;

mysql>start slave

配置主从从模式

配置主库

1. 用户授权
2. 启用binlog日志
3. 重启服务

mysql>grant replication slave on \*.\* to 用户名@’从库IP地址’ identified by ‘密码’;

vim /etc/my.cnf

[mysqld]

log-bin=日志号

server\_id=id号

binlog\_format=’mixed’

systemctl restart mysqld

配置第一台从库

1. 修改配置文件
2. 用户授权
3. 指定主库信息
4. 启动slave从库线程

vim /etc/my.cnf

[mysqld]

server\_id=id号

log-bin=日志名

binlog\_format=’mixed’

log\_slave\_updates

systemctl restart mysqld

mysql>grant replication slave on \*.\* to 用户名@’第2台从库的IP地址’ identifief by ‘密码’;

mysql>change master to master\_host=’主库IP地址’,

->master\_user=’用户名’,

->master\_password=’密码’,

->master\_log\_file=’binlog日志文件名’

->master\_log\_pos=偏移量;

mysql>start slave;

mysql>show slave status\G;

配置第二台从库

1. 修改配置文件
2. 指定主库信息
3. 启动slave进程
4. 查看状态信息

vim /etc/my.cnf

server\_id=id号

systemctl restart mysqld

mysql>change master to master\_host=’第1台从库IP地址’,

->master\_user=’用户名’,

->master\_password=’密码’,

->master\_log\_file=’binlog日志文件名’

->master\_log\_pos=偏移量;

mysql>start slave;

mysql>show slave status\G;

测试配置

1. 在主库授权访问数据的用户
2. 访问主库，执行建库、建表、插入记录等操作
3. 访问第1台从库，可看到主库操作结果
4. 访问第2台从库，也可以看到主库操作结果

复制模式

模式种类

异步模式（默认的复制模式）

主库执行完一次事务后，立即将结果返回客户端，并不关心从库是否已经接受并处理

全同步复制

当主库执行完一次事务，且所有从库执行了该事务后才返回给客户端

半同步复制

介于上两者之间

主库执行完一次事务后，等待至少一个从库写到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\_dynameic\_loading’;

命令行加载插件

mysql>install plugin rpl\_semi\_sync\_master soname ‘semisync\_master.so’; //主库上执行

mysql>install plugin rpl\_semi\_sync\_slave soname ‘semisync\_slave.so’; //从库上执行

mysql>select plugin\_name,plugin\_status from information\_schema.plugins where plugin\_name like ‘%semi%’;//查询结果

启用半同步模式（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; //在主库上执行

mysql>set global rpl\_semi\_sync\_slave\_enabled = 1; //在从库上执行

mysql>show variables like ‘rpl\_semi\_sync\_%\_enabled’; //查看

永久启用半同步复制（需要添加主配置文件/etc/my.cnf）

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

rpl\_semi\_sync\_master\_enabled=1 //主库配置

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

rpl\_semi\_sync\_slave\_enabled=1 //从库配置

在高可用架构下，master和slave需同时启动（以便在切换后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性能调优**

工作原理：

管理工具：mysql服务软件安装后提供的命令

连接池：检查本机是否有资源处理当前的连接请求（空闲的线程 内存）

sql接口：把sql命令传递给sql服务的进程处理

分析器：检查执行的sql命令是否有语法错误

优化器：优化执行的sql命令，使其能以最节省系统资源的方式执行

查询缓存：查询缓存的存储空间是从系统的物理内存里划分出来的，用来存储查询过的结果

存储引擎：软件自带的功能程序，用来处理表的处理器

文件系统：数据库服务器存储数据的磁盘

mysql执行流程

客户端发起连接请求，连接池检查并建立连接，如果有查询请求，分析语法是否有错误，优化sql命令，去查询缓存里头找，如果有直接返回给客户端，如果没有，会去表里头找，然后查到查询缓存，再返还给客户端，下次直接去查询缓存里找

mysql性能调优

提高mysql系统的性能、响应速度

替换有问题的硬件（CPU/磁盘/内存等）

服务程序的运行参数调整

对sql查询进行优化

并发及连接控制

连接数、连接超时

max\_connections允许的最大并发连接数，用有过的最大连接（show global status like “makx\_used\_connections”;）除以并发连接数，大约为0.85为合适

connect\_timeout等待连接超时时间，默认是10秒，仅登录时有效

wait\_time等待关闭连接的不活动超时秒数，默认28800秒（8小时）

缓存参数控制

缓冲区、线程数量、开表数量

key\_buffer\_size用于myisam引擎的关键索引缓存大小，分配多少字节用于内存存储索引，默认为8兆，当key\_reads/key\_read\_requests较低时，可适当加大此缓存值

sort\_buffer\_size为每个要排序的线程分配此大小的缓存时间，增大此值可提高order和group的速度

read\_buffer\_size为顺序读取表记录保留的缓存大小，此值影响sql查询的响应速度

thread\_cache\_size可以重复使用的线程的数量

table\_open\_cache为所有线程缓存的打开的表的数量

sql查询优化

设置查询的缓存

查看缓存大小（show variables like “query\_cache%”;）

第一行如果查询结果大于这个数，不允许往查询缓存中存放，默认是1M

查询缓存的存储单位最小是4K

查询缓存空间的大小，默认为1M

是否开启查询缓存，默认是OFF，可以设置0、1、2，0代表关，1代表只要不超过限制，都可以保存，2代表查询的时候必须加关键字sql\_in\_cache否则不往查询缓存里保存

查询缓存写锁有效（对myisam存储引擎）当对myisam存储引擎的表，查询的时候，若此时有客户端对表执行写操作，mysql服务不会从缓存里查找数据返回给客户端，而是等写操作完成后，重新从表里查找数据返回给客户端

查看当前的查询缓存统计（show global status like “qcache%”;）

qcache\_hits：只要客户端的查询结果是在查询缓存中找到的，就会记录一遍，值自动加1

qcache\_inserts：只要接受到一次查询请求，值就+1

qcache\_lowmem\_prunes：查询缓存如果满了，会自动清理查询缓存，清理一次值+1

qcache\_not\_cached：不往查询缓存里放的次数，越大意味着limit值设置小了

优化查询寻（启用慢查询）

相关配置

slow-query-log启用慢查询

slow-query-log-file指定慢查询日志

long-query-time=时间 超时时间（默认10秒）

log-queries-not-using-indexes=1记录未使用索引的查询

mysql性能调优

升级硬件：CPU、内存、硬盘

加大网络带宽：付费加大网络带宽

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、索引查询

启用慢日志查询：slow-query-log

网站架构不合理：调整网络架构

**mysql中间件：mysql-proxy mycat maxscale**

**mysql读写分离：把客户端访问数据库服务时的查询请求和写数据的请求给不同的数据库服务器处理**

读写分离原理

多台mysql服务器

分别提供读、写服务，均衡流量

通过主从复制保持数据的一致性

mysql代理面向客户端

收到写请求时，交给服务器A处理

收到读请求时，交给服务器B处理

构建思路

1. 构建读写分离
2. 添加一台mysql代理服务器

部署/启用maxscale

1. 客户端通过代理主机访问mysql数据库

访问代理服务器

部署maxscale服务（只有在写的时候才切换到主库，查在从库）

下载地址

<https://downloads.mariadb.com/files/MaxScale>

修改配置文件/etc/maxscale.cnf

[maxscale]

threads=auto//设置线程数量

[server1]//定义数据库服务器

type=server

address=ip地址 //master服务器ip地址

port=3306

protocol=MySQLBackend

[server2]//定义数据库服务器

type=server

address=ip地址 //slave服务器ip地址

port=3306

protocol=MySQLBackend

[MySQL Monitor] //定义要监控的数据库节点

type=monitor

module=mysqlmon

servers=server1,server2//主、从数据库的主机名，中间用逗号分隔

user=scalemon

passwd=123456//用户要有相应权限（执行查看服务器运行状况的权限和查看主从命令的权限）

monitor\_interval=10000

#[Read-Only Service]//注释此项

#type=server

#router=readconnroute

#servers=server1

#user=scalemon

#passwd=123456

#router\_options=slave

[Read-Write Service]//定义读写分离的数据库节点

type=server

router=readwritesplit

servers=server1,server2//主、从数据库的主机名，中间用逗号分隔

user=maxscale

passwd=123456//该用户名密码用来验证接受到客户端的访问请求时，验证账户在数据库服务器上是否存在

max\_slave\_connections=100%

[MaxAdmin Service]

type=service

router=cli

#[Read-Only Listener]//注释此项

#type=listener

#service=Read-Only Service

#portocol=MySQLClient

#port=4008

[Read-Only Listener]

type=listener

service=Read-Write Service

portocol=MySQLClient

port=4006

[MaxAdmin Listener]

type=listener

service=MaxAdmin Service

portocol=maxscaled

port=4099//默认是4008，可以指定

在主、从数据库服务器创建授权用户

grant replication slave,replication client on \*.\* to scalemon@’%’ identified by ‘123456’;//创建监控用户

grant select on mysql.\* to maxscale@’%’ identified by ‘123456’;//创建后一个用户

启动服务

maxscale --config=/etc/maxscale.cnf

netstat -utnalp | grep maxscale

kill -9 端口号

测试配置

在maxscale本机连接管理端口

maxadmin -uadmin -pmariadb -P 端口（4099）

maxscale>list servers

maxscale>exit

访问maxscale代理

mysql -h服务器地址 -P（4006）端口 -u用户名 -p密码

**mysql多实例（一个物理主机上运行多个数据库服务）**

配置步骤

安装支持多实例服务的软件名

解压软件、修改目录名、调整PATH变量

tar -xf mysql-5.7.20-linux-glibc2.12-x86\_64.tar.gz

mv mysql-5.7.20-linux-glibc2.12-x86\_64 /usr/local/mysql

vim /etc/profile

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

source /etc/profile

修改主配置文件

mkdir /dir3307

mkdir /dir3308

vim /etc/my.cnf

[mysqld\_multi]

mysqld=/usr/local/mysql/bin/mysqld\_safe

mysqladmin=/usr/local/mysql/bin/mysqladmin

user=root

[mysqld1]

port=3307

socket=/dir3307/mysqld3307.sock

pid-file=/dir3307/mysqld.pid

log-error=/dir3307/mysqld.err

datadir=/dir3307

[mysqld2]

port=3308

socket=/dir3307/mysqld3308.sock

pid-file=/dir3308/mysqld.pid

log-error=/dir3308/mysqld.err

datadir=/dir3308

初始化授权库（会提示root用户登录的初始密码）

mysqld --user=mysql --basedir=软件安装目录 --datadir=数据库目录 --initialize

启动实例进程

mysqld\_multi start 实例编号

停止指定实例

mysqld\_multi --user=root --password=密码 stop 实例编号

客户端访问

本机连接

使用初始密码连接

mysql -uroot -p密码 -S sock文件

修改本机登录密码

mysql>alter user user() identified by ‘新密码’;

连接实例

mysql -uroot -p新密码 -S sock文件

**MHA集群（MHA软件+主从同步实现的，主备模式，当主角色宕机后，备份主机自动接替主角色的主机提供服务给客户端）**

**集群：使用多台服务器提供相同的服务**

**工作原理：一台主库服务器，多台从库服务器，进行同步主库操作，会有一个vip地址，一开始在主库上面，当主库宕机后，vip地址会自动转移到备用主句的服务器上（数据最接近主库的那台），然后自动修改主从里的备用主库**

前提：一主多从

安装依赖的软件包

ssh root用户无密码登录

配置MHA

配置管理主机

配置数据主机（一主多从 安装依赖的软件包 彼此之间可以ssh root用户无密码登录）

准备集群环境

安装软件包

所有主机上安装perl依赖包

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

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

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

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

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

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

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

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

yum -y install perl\*.rpm

在所有数据节点上授权监控用户

mysql>grant all on \*.\* root@’%’ identified by ‘123456’;

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 -xf mha4mysql-manager-0.56.tar.gz

cd mha4mysql-manager-0.56

perl Makefile.PL

make

make install

配置ssh密钥对认证登录

所有节点之间互相以root密钥对认证登录

管理主机以root密钥对认证所有数据节点主机

ssh-keygen

ssh-copy-id 对方IP地址

配置MHA集群

配置master主节点

修改配置文件

vim /etc/my.cnf

[mysqld]

server\_id=1

log-bin=master

binlog\_format="mixed"

validate\_password\_policy=0

validate\_password\_length=6

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

systemctl restart mysqld

添加主从同步授权用户

mysql>set global relay\_log\_purge-off;//不自动删除本机中的中继日志

mysql>grant replication slave on \*.\* to repluser@”%” identified by ‘123456’;

配置备用1节点

vim /etc/my.cnf

[mysqld]

server\_id=2

log-bin=master2

binlog\_format="mixed"

validate\_password\_policy=0

validate\_password\_length=6

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

systemctl restart mysqld

mysql>set global relay\_log\_purge-off;//不自动删除本机中的中继日志

mysql>change master to

->master\_host=”192.168.4.51”

->master\_user=”repluser”,

->master\_password=”123456”,

->master\_log\_file=”master51.000001”,

->master\_log\_pos=441;

mysql>start slave;

配置备用2节点

vim /etc/my.cnf

[mysqld]

server\_id=3

log-bin=master3

binlog\_format="mixed"

validate\_password\_policy=0

validate\_password\_length=6

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

systemctl restart mysqld

mysql>set global relay\_log\_purge-off;//不自动删除本机中的中继日志

mysql>change master to

->master\_host=”192.168.4.51”

->master\_user=”repluser”,

->master\_password=”123456”,

->master\_log\_file=”master51.000001”,

->master\_log\_pos=441;

mysql>start slave;

配置从库（可有可无）

vim /etc/my.cnf

[mysqld]

server\_id=4

validate\_password\_policy=0

validate\_password\_length=6

systemctl restart mysqld

mysql>set global relay\_log\_purge-off;//不自动删除本机中的中继日志

mysql>change master to

->master\_host=”192.168.4.51”

->master\_user=”repluser”,

->master\_password=”123456”,

->master\_log\_file=”master51.000001”,

->master\_log\_pos=441;

mysql>start slave;

配置管理主机

mkdir /etc/mha\_manager

cp /mha4mysql-manager-0.56/samples/conf/app1.cnf /etc/mha\_manager

vim /etc/mha\_manager/app1.cnf

[server default]

manager\_workdir=/mha\_manager

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

ssh\_port=22

ssh\_user=root

master\_ip\_failover\_script=/mha\_manager/master\_ip\_failover

repl\_user='repluser'

repl\_password='123456'

user=root

password=123456

[server1]

hostname=192.168.1.10

candidate\_master=1

[server2]

hostname=192.168.1.83

candidate\_master=1

[server3]

hostname=192.168.1.238

candidate\_master=1

#[server4]

#hostname=132.232.56.166

#no\_master=1

#port=3306

#

#[server5]

#hostname=116.196.101.172

#no\_master=1

cp /mha4mysql-manager-0.56/samples/scripts/master\_ip\_failover /etc/mha\_manager

vim /etc/mha\_manager/master\_ip\_failover

my $vip = '192.168.1.139/24'; # Virtual IP

my $key = "1";

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

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

配置虚拟IP，进行绑定到主数据库服务器上

ifconfig eth0:1 192.168.1.139/24

ifconfig eth0:1

测试配置

测试ssh密钥对登录

masterha\_check\_ssh --conf=管理节点主配置文件

查看主从同步状态

masterha\_check\_repl --conf=管理节点主配置文件

启用MHA\_Manager工具

使用master\_manager工具

--remove\_dead\_master\_conf//删除宕机主库配置

--ignore\_last\_failover//忽略xxx.health文件

查看状态：master\_check\_status --conf=管理节点主配置文件

停止服务：masterha\_stop --conf=管理节点主配置文件

如果宕掉，需要重新写配置文件，配置服务，重新启动服务

manager节点提供的命令工具

masterha\_check\_ssh检查ssh配置的情况

masterha\_check\_repl检查mysql复制状况

masterha\_manager启动MHA

masterha\_check\_status检查MHA运行状态

masterha\_master\_monitor检查master是否宕机

**mysql视图（虚拟表，假表）**

内容与真实表相似，有字段有记录

只会创建表结构文件，不会创建表记录文件

行和列的数据来自定义视图查询时所引用的基表，并且在引用基表时动态生成

更新视图的数据，就是更新基表的数据

更新基表的是数据，视图的数据也会跟着改变

视图使用限制

不能在视图上创建索引

在视图的from字句中不能使用子查询

以下情形中的视图不可更新

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字句的子查询引用了FROM字句中的表

使用了临时表

视图基本使用

创建：create view 视图名称 as select \* from 表名（也就是sql查询）;

create view 视图名称(字段列表) as sql查询

在视图中不定义字段名的话，默认使用基表的字段名，若定义字段名的话，视图中的字段必须与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（用来查看表是否为视图）

show table status;

show table status where comment=”view”\G;

查看创建视图具体命令（用来查看该视图基于哪个基表）

show create view 视图名;

删除视图

drop 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);

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;

视图表修改记录，基表的记录也会更改，基表记录更改，视图表的记录也会更改

创建视图的完整格式

create

[or replace]

[algorithm={undefined|mergre|temptable}]

[definer={user|current\_user}]//指定视图的创建者

[sql security{definer | invoker}]

view view\_name [(column\_list)]

as select\_statement

[with[cascaded|local]check option]

设置字段别名

视图中的字段名不可以重复 所以要定义别名

给表和字段起别名：

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;

给字段起别名：

create view 视图名

as

select 源表名.源字段名 as 字段别名

from 源表名 left join 源表名 表别名

on 条件;

关联视图查询建的图，默认不允许修改视图字段的值

重要事项说明

or replace

create or replace 视图名 as select 查询;

创建时，若视图已经存在，会替换已有的视图

algorithm(视图的执行方式)

algorithm={underfined|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,具体换方式

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数据库内存中之后，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merage替换的方式

sql security（指定视图查询数据时的安全验证方式）

definer表示按定义者拥有的权限来执行

invoker表示用调用者的权限来执行。默认情况下，系统指定为definer

任意用户访问view时，能否成功取决于是否有调用该view的权限，以及definer是否拥有view中的select的权限，只需要修改创建同名用户或者修改definer即可。

with [local|cascaded] check option（作用是定义对视图表里的数据做操作时的限制方式）

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关键字决定检查的范围

local仅检查当前视图的限制（满足视图本身的限制即可）

cascaded同时要满足基表的限制（默认值，要同时满足基表的限制，让改但是数据没有了，改的是最原始的基表）

create view v2 select \* from user2 limit 3;不能修改数据，因为修改了系统就不认为是数据是前3行了。

**mysql存储过程**

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、条件判断、流程控制等

存储过程，相当于是mysql中的脚本

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语句

创建存储过程

mysql>delimiter //

mysql>create procedure 名称()

->begin

->...功能代码

->end

->// 结束存储过程

mysql>delimiter ;

delimiter关键字用来指定存储过程的分隔符(默认为;)，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。

查看存储过程

mysql>show procedure status\G;

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

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

存储过程有参数时，调用时必须传给参数

删除存储过程

drop procedure 存储过程名;

变量类型

会话变量 全局变量 用户变量 局部变量

会话变量、全局变量：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，只会影响当前会话 select @@hostname

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=值;

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，语句执行完毕后，其变量失效。declare专门用来定义局部变量 declare 变量 类型(宽度) default 预设值 或者 declare 变量 类型(宽度);set 变量=”值”;

参数类型

调用参数时，名称前也不需要加@

create procedure 名称(

类型 参数名 数据类型,类型 参数名 数据类型

)

类型：

in 输入参数 作用是给存储过程传值，必须在调用过程中赋值，在存储过程中该参数的值不允许修改;默认值是in

out 输出参数 该值可在存储过程内部改变，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

算数运算

+ 加法运算 set @var1=2+2;select @var1;

- 减法运算 set @var2=3-2;select @var2;

\* 乘法运算 set @var3=3\*2;select @var3;

/ 除法运算 set @var4=3/2;select @var4;

DIV 整除运算 set @var5=10 DIV 3;select @var5;

% 取模 set @var6=10%3;select @var6;

流程控制

条件测试

数值比较

= 等于

>、>= 大于、大于或等于

<、<= 小于、小于或等于

!= 不等于

between...and.. 在...于...之间

逻辑比较、范围、空、非空、模糊、正则

OR AND ! 与 或 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

单分支结构

当”条件成立”时执行

if 条件测试 then

代码;

end if;

双分支结构

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;

循环结构

while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;

loop死循环

无条件、反复执行某一段代码

loop

循环体

end loop;

repeat条件式循环

当条件成时立即结束循环

repeat

循环体

until 条件判断

end repeat;

控制循环结构的执行（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（一般loop与leave连用）

iterate 标签 //放弃本次循环，并开始下一次循环（一般与while连用）

**分库分表**

概念：将存放在一个数据库中的数据，按照特定方法进行拆分，分散放到多个数据库（主机）中，以达到分散单台设备负载的效果；解决单数据服务器的访问压力过大和存储压力；解决单表过大的问题

种类：垂直分割：将单个表，拆分为多个表，分散到不同的数据库中

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：按照表中的某个字段的某种规则，把表中的许多记录按行分割，分散到多个数据库中。

mycat：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提供解决方案，可以提供读写分离系统，可以实现数据库高可用，提供数据分片服务

mycat支持的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-parttern
7. ascii码求模通配sharding-by-prefixpattern
8. 编码指定sharding-by-substring
9. 字符串拆分hash解析sharding-by-stringhash
10. 一致性hash sharding-by-murmur

工作过程：插入数据时，如果有分片规则，就按照相应分片规则找到相应的字段，写入不同的库中。 当mycat收到一个sql查询时，先解析这个sql查找涉及到的表，然后查看此表的定义，如果有分片的规则，则获取sql里分片字段的值，并分配行数，获取分片列表，然后黄sql发往这些分片去执行，最后收集和处理所有分片结果数据，并返回到客户端

配置mycat

环境：两个数据库服务器，一台mycat服务器

装包：安装java jdk

查看：rpm -qa | grep -i jdk

安装mycat服务软件包

tar -xf mycat-server-1.4-beta-20150604171601-linux.tar.gz

mv mycat /usr/local

修改配置文件

vim /usr/local/mycat/conf/server.xml

<user name="test">

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

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

</user>

<user name="user">

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

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

<property name="readOnly">true</property>

</user>

vim /usr/local/mycat/conf/schema.xml

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

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

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

.....

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

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

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

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

<heartbeat>select user()</heartbeat>

<writeHost host="hostM1" url="139.9.234.84:3306" user="root"

password="123456">

</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>

<writeHost host="hostM1" url="139.9.237.41:3306" user="root"

password="123456">

</writeHost>

</dataHost>

修改数据库配置文件并授权

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

vim /etc/my.cnf

lower\_case\_table\_names = 1

systemctl start mysqld

启动服务

指定java路径、添加PATH路径、启动服务

java -version

/usr/local/mycat/bin/mycat start

cat mycat.pid

测试配置

mysql -h mycat服务器地址 -utest -ptest -P8066

create table employee(id int primary key,name varchar(100),age int(2),sharding\_id int not null);

insert into employee(id,name,age,sharding\_id) values(2,"lucy",18,10010);

insert into employee(id,name,age,sharding\_id) values(1,"bob",18,10000);

**使用技巧：**

**在mysql命令行下面执行系统命令：加上system即可，也就是system+系统命令**

**select user();：查看连接的用户和地址**

**select @@hostname：查看连接的主机名**

**show grants;：查看自己的授权信息**

**show processlist;显示自己数据库运行的程序列表**

**flush status;刷新状态，归零**

**show variables;查看所有变量**

**set [global] 变量名=值;修改变量值**

**[mysqld]**

**变量名=值 修改变量值永久生效**

**show status like ”变量”;查看变量状态**

**mysql>set global relay\_log\_purge-off;不自动删除本机中的中继日志**

**show table status;查看表状态**