# NSD DBA1 DAY01

庞丽静 [panglj@tedu.cn](mailto:panglj@tedu.cn)

第三阶段：数据库管理15天（三个5天），安全与监控6天（必备）。

1. 构建MySQL服务器
2. 数据库基本管理
3. MySQL 数据类型

## 慨念：

### 1）什么是数据库：

DB，database:

数据库：依照某种数据模型进行组织并存放到存储器（硬盘）的数据集合

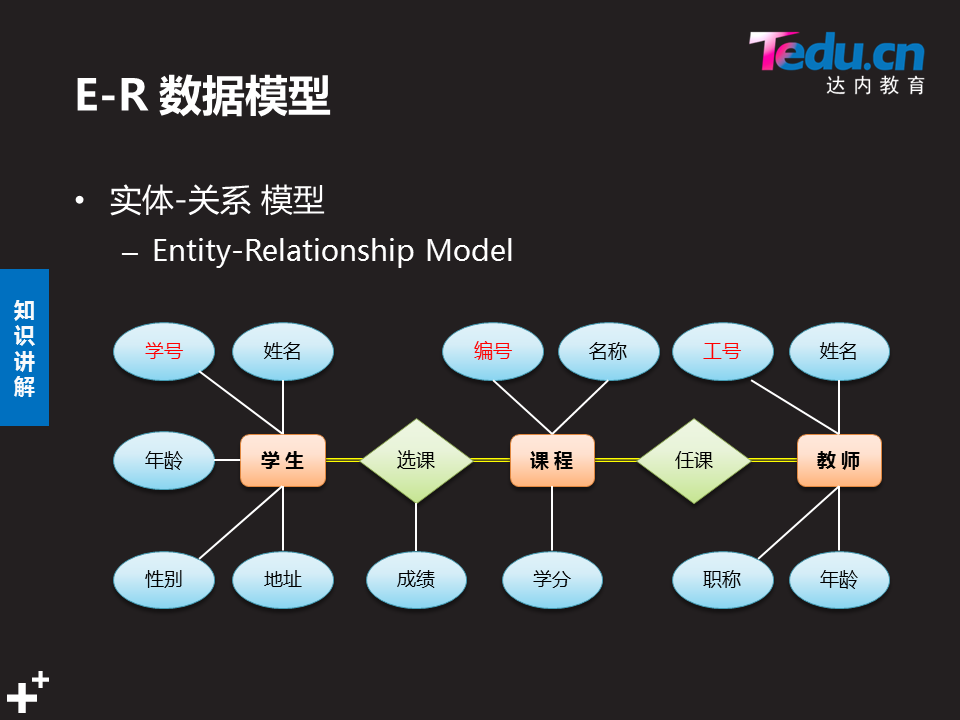
DBMS database management system：

数据库管理系统：用来操纵和管理数据库的大型服务软件

DBS，database system

数据库系统，DB+DBMS，指带有数据库并整合了数据管理软件的计算机系统

### 2）E-R数据模型



RDBMS：关系型数据库系统

### 3）常见数据库服务软件



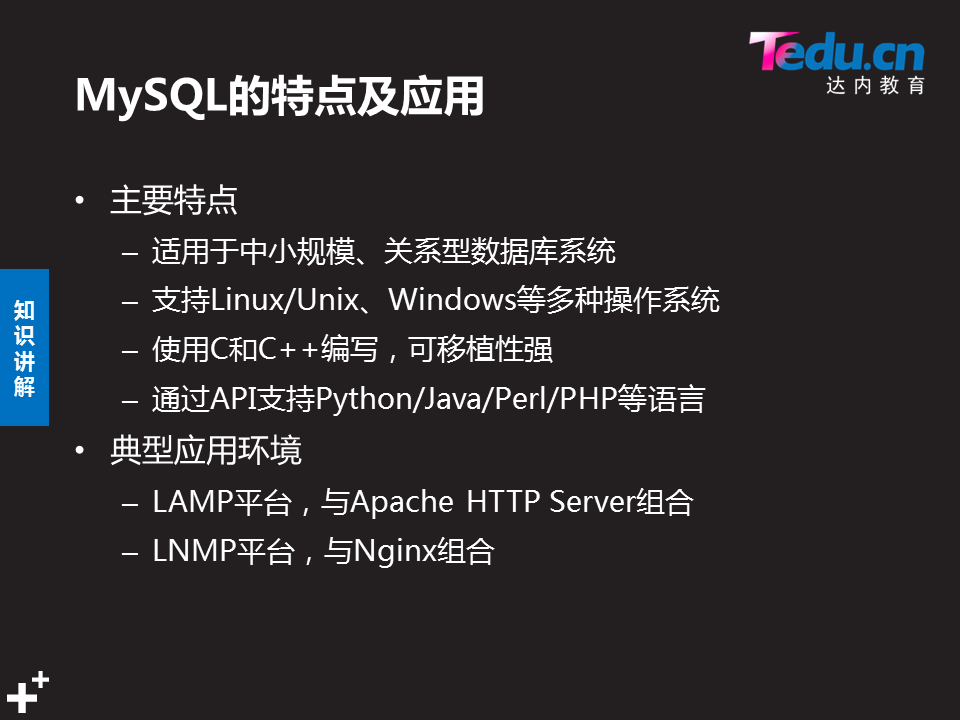
如何选择数据库软件：

是否开源：自由使用，二次开发，但是不等于免费

是否商业：微软的都是商业软件，需要收费（oracle sqlserver db2）

是否跨平台:unix linux windows 除了sqlserver，其余都是跨平台的

### 4）MySQL的特点及应用



API：访问程序接口

## 1 构建MySQL服务器

1.1 问题

本案例要求熟悉MySQL官方安装包的使用，快速构建一台数据库服务器：

* 安装MySQL-server、MySQl-client软件包
* 修改数据库用户root的密码
* 确认MySQL服务程序运行、root可控

1.2 方案

本课程将使用64位的RHEL 7操作系统，MySQL数据库的版本是5.7.17。

访问http://dev.mysql.com/downloads/mysql/，找到MySQL Community Server下载页面，平台选择“Red Hat Enterprise Linux 7/ Oracle Linux”，然后选择64位的bundle整合包下载，如图-1所示。

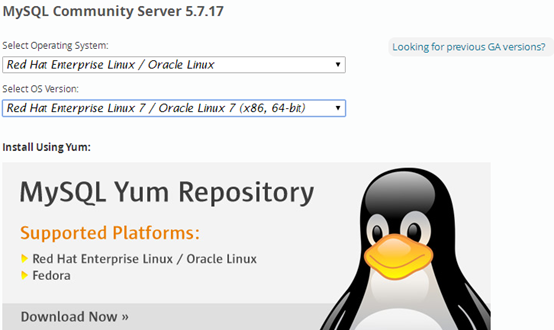


图-1

注意：下载MySQL软件时需要以Oracle网站账户登录，如果没有请根据页面提示先注册一个（免费）

1.3 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：准备工作

#### 1）停止mariadb服务

[root@localhost ~]# systemctl stop mariadb

#### 2）删除/etc/my.cnf配置文件

此配置文件由RHEL自带的mariadb-libs库提供：

[root@localhost ~]# rm -rf /etc/my.cnf

#### 3）删除数据

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

#### 4）卸载软件包（没有会显示未安装软件包）

[root@localhost ~]# rpm -e --nodeps mariadb-server mariadb

警告：/var/log/mariadb/mariadb.log 已另存为/var/log/mariadb/mariadb.log.rpmsave

#因为mariadb redhat 自带的，mariadb与mysql都是提供mysql服务，所以要先卸载自带的mariadb，

### 步骤二：安装软件包

#### 1）安装mysql时可能会缺少某些依赖包，需提前单独安装

[root@localhost ~]# yum -y install perl-Data-Dumper perl-JSON perl-Time-HiRes

#### 2）物理机传输解压包给虚拟机192.168.4.1

[root@room9pc01 ~]# cd 桌面

[root@room9pc01 桌面]# scp mysql-5.7.17.tar 192.168.4.1:/root/ //给虚拟机传包

root@192.168.4.1's password:

mysql-5.7.17.tar 100% 543MB 95.6MB/s 00:05

#### 虚拟机192.168.4.1解压mysql-5.7.17.tar 整合包

[root@sql51 ~]# du -sh mysql-5.7.17.tar #查看tar包的大小，确保完整

544M mysql-5.7.17.tar

[root@localhost ~]# tar -xvf mysql-5.7.17.tar //解压mysql整合包

./mysql-community-client-5.7.17-1.el7.x86\_64.rpm

#客户端应用程序，提供管理命令

./mysql-community-common-5.7.17-1.el7.x86\_64.rpm #数据库和客户端共享文件

./mysql-community-devel-5.7.17-1.el7.x86\_64.rpm #客户端应用程序的库和头文件

./mysql-community-embedded-5.7.17-1.el7.x86\_64.rpm #嵌入式函数库

./mysql-community-embedded-compat-5.7.17-1.el7.x86\_64.rpm

#嵌入式兼容函数库

./mysql-community-embedded-devel-5.7.17-1.el7.x86\_64.rpm

#头文件和库文件作为Mysql的嵌入式库文件

./mysql-community-libs-5.7.17-1.el7.x86\_64.rpm

#MySQL 数据库客户端应用程序的共享库

./mysql-community-libs-compat-5.7.17-1.el7.x86\_64.rpm

#客户端应用程序的共享兼容库

./mysql-community-minimal-debuginfo-5.7.17-1.el7.x86\_64.rpm

./mysql-community-server-5.7.17-1.el7.x86\_64.rpm #服务端

./mysql-community-test-5.7.17-1.el7.x86\_64.rpm

#### 安装

[root@localhost ~]# yum -y install mysql-community-\*.rpm

#### 5）查看mysql包

[root@localhost ~]#rpm -qa | grep -i mysql

### 步骤三：启动MySQL数据库服务并设置开机自启

[root@localhost ~]# systemctl start mysqld //启动mysql服务

[root@localhost ~]# systemctl enable mysqld //设置开机自启

[root@localhost ~]# systemctl status mysqld //查看mysql服务状态

● mysqld.service - MySQL Server

Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)

Active: active (running) since 二 2018-08-28 10:03:24 CST; 8min ago

Docs: man:mysqld(8)

http://dev.mysql.com/doc/refman/en/using-systemd.html

Main PID: 4284 (mysqld)

CGroup: /system.slice/mysqld.service

└─4284 /usr/sbin/mysqld --daemonize --pid-file=/var/r...

8月 28 10:02:56 localhost.localdomain systemd[1]: Starting MySQ...

8月 28 10:03:24 localhost.localdomain systemd[1]: Started MySQL...

Hint: Some lines were ellipsized, use -l to show in full.

### \*配置相关参数：

主配置文件：/etc/my.cnf

数据库目录：/var/lib/mysql

进程名：mysql

传输协议：tcp

进程所有者：mysql

进程所属组：mysql

#### 查进程

[root@sql51 ~]# ps -C mysqld

PID TTY TIME CMD

2715 ? 00:00:00 mysqld

#### 查端口

[root@sql51 ~]# netstat -utnlp |grep :3306

tcp6 0 0 :::3306 :::\* LISTEN 2715/mysqld

#### 首次启动初始化的文件

[root@sql51 ~]# ls /var/lib/mysql

auto.cnf ib\_buffer\_pool mysql public\_key.pem

ca-key.pem ibdata1 mysql.sock server-cert.pem

ca.pem ib\_logfile0 mysql.sock.lock server-key.pem

client-cert.pem ib\_logfile1 performance\_schema sys

client-key.pem ibtmp1 private\_key.pem

#### 查用户查组

[root@sql51 ~]# grep mysql /etc/passwd

mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/false

[root@sql51 ~]# grep mysql /etc/group

mysql:x:27:

#### 查进程所有者

[root@sql51 ~]# ps aux |grep -i mysql

mysql 2715 0.0 17.1 1119216 173888 ? Sl 10:21 0:00 /usr/sbin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.pid

root 2888 0.0 0.0 112676 996 pts/0 S+ 10:32 0:00 grep --color=auto -i mysql

### 步骤四：连接MySQL服务器，修改密码

#### 查看随机生成的root管理密码

初始密码存放载日志文件中，可查看主配置文件知道日志文件

[root@sql51 ~]# vim /etc/my.cnf

log-error=/var/log/mysqld.log

pid-file=/var/run/mysqld/mysqld.pid

[root@localhost ~]#grep 'temporary password' /var/log/mysqld.log

2017-04-01T18:10:42.948679Z 1 [Note] A temporary password is generated for root@localhost: mtoa>Av<p6Yk

//root＠在本机登录的　随机生成的管理密码为mtoa>Av<p6Yk

#### 2）使用客户端命令mysql连接到MySQL服务器

提示验证时，填入前一步获得的随机密码，验证成功后即可进入“mysql> ”环境：

[root@localhost ~]# mysql -u root -p'mtoa>Av<p6Yk'

mysql: [Warning] Using a password on the command line interface can be insecure.

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 11

Server version: 5.7.17

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> //登录成功后，进入SQL操作环境

用该密码登录到服务端后，必须马上修改密码，不然会报如下错误：

mysql> show databases;

ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this statement.

不修改密码要求参数，直接修改密码　　　＃修改root在本机登录的密码

mysql> alter user root@"localhost" identified by "123qqq...AAA" ;

Query OK, 0 rows affected (0.00 sec)

[root@sql51 ~]# mysql -uroot -p123qqq...AAA　#密码无特殊字符，不需要单引号

#### 3）查看密码策略

mysql> show variables like "%password%" ;

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

| Variable\_name | Value |

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

| default\_password\_lifetime 　| 0 |

| disconnect\_on\_expired\_password | ON |

| log\_builtin\_as\_identified\_by\_password 　| OFF |

| mysql\_native\_password\_proxy\_users | OFF |

| old\_passwords 　 | 0 |

| report\_password 　 | |

| sha256\_password\_proxy\_users | OFF |

| validate\_password\_check\_user\_name | OFF |

| validate\_password\_dictionary\_file 　　 | |

| validate\_password\_length 　 | 8 |　#默认为８

| validate\_password\_mixed\_case\_count | 1 |

| validate\_password\_number\_count | 1 |

| validate\_password\_policy 　 | MEDIUM |　#默认为１（MEDIUM）

| validate\_password\_special\_char\_count | 1 |

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

14 rows in set (0.01 sec)

#### 4) 临时修改密码策略,执行SET PASSWORD命令

这个其实与validate\_password\_policy的值有关，默认为1，所以刚开始设置的密码必须符合长度，且必须含有数字，小写或大写字母，特殊字符。如果我们不希望密码设置的那么复杂，需要修改两个全局参数：validate\_password\_policy与validate\_password\_length。validate\_password\_length默认值为8,最小值为4，如果你显性指定validate\_password\_length的值小于4，尽管不会报错，但validate\_password\_length的值将设为4。

可参考下列指令：

mysql>set global validate\_password\_policy=0; //只验证长度,临时修改

Query OK, 0 rows affected (0.00 sec)

参数：0 or LOW：只检查密码长度

　1 or MEDIUM：检测长度，数字，大小写和特殊字符（默认）

　2 or STRONG：检测长度，数字，大小写和特殊字符，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

Query OK, 0 rows affected (0.00 sec)

mysql> show variables like "%password%" ;

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

| Variable\_name | Value |

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

| default\_password\_lifetime 　| 0 |

| disconnect\_on\_expired\_password | ON |

| log\_builtin\_as\_identified\_by\_password　 | OFF |

| mysql\_native\_password\_proxy\_users | OFF |

| old\_passwords 　 | 0 |

| report\_password 　 | |

| sha256\_password\_proxy\_users | OFF |

| validate\_password\_check\_user\_name | OFF |

| validate\_password\_dictionary\_file 　　 | |

| validate\_password\_length 　 | 6 | #长度修改为６

| validate\_password\_mixed\_case\_count | 1 |

| validate\_password\_number\_count | 1 |

| validate\_password\_policy 　 | LOW |　#修改为0(LOW)

| validate\_password\_special\_char\_count 　 | 1 |

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

14 rows in set (0.00 sec)

#### 5）永久修改密码策略

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

validate\_password\_length=6

validate\_password\_policy=0

#### 修改密码

**修改密码语句一:**

mysql> alter user user() identified by "123456"; //修改登陆密码

Query OK, 0 rows affected (0.00 sec)

　　上述操作的结果是——更改数据库用户root从本机访问时的密码，设为123456。

退出“mysql> ”环境，重新登录验证，必须采用新的密码才能登入：

mysql> exit //退出 mysql> 环境

Bye

[root@localhost ~]# mysql -u root -p //重新登录

Enter password: //输入新设置的密码

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 15

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

**修改密码语句二:**

mysql> alter user root@"localhost" identified by "123qqq...AAA" ;

Query OK, 0 rows affected (0.00 sec)

[root@sql51 ~]# mysql -uroot -p123qqq...AAA　#密码无特殊字符，不需要单引号

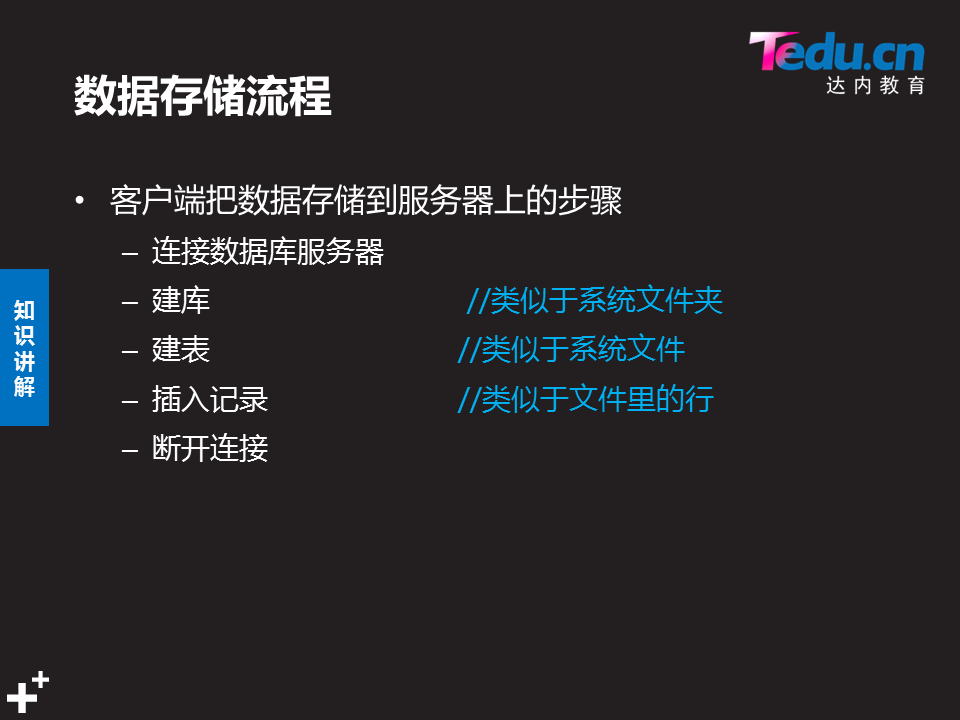
## 2 数据库基本管理

### 慨念

#### ＭySQL连接工具：

* 命令行
* 图形软件
* Web页面

#### 数据存储流程



2.1 问题

本案例要求熟悉MySQL的连接及数据库表的增删改查等基本管理操作，主要完成以下几个方便的操作：

使用mysql命令连接数据库

练习查看/删除/创建库的相关操作

练习查看/删除/创建表的相关操作，表数据参考如表-1所示内容

表－1 测试用表数据

2.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：使用mysql命令连接数据库

#### 连接数据库基本用法

连接MySQL服务器时，最基本的用法是通过 -u 选项指定用户名、-p指定密码。密码可以写在命令行（如果不写，则出现交互，要求用户输入），当然基于安全考虑一般不推荐这么做：

[root@dbsvr1 ~]# mysql -uroot -p123456 //紧挨着选项，不要空格

mysql: [Warning] Using a password on the command line interface can be insecure.

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 16

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> exit //退出已登录的mysql> 环境

Bye

**或者例如:**

[root@dbsvr1 ~]# mysql -u root -p #-u与root空格隔开

Enter password:

[root@client-50 ~]# mysql -u root -p123456

**错误登录:**

[root@client-50 ~]# mysql -u root -p 123456 #错误做法,123456

Enter password:

ERROR 1049 (42000): Unknown database '123456' #无123456的数据库

#### 指定远程主机连接

默认情况下，msyql命令会连接本机的MySQL服务。但在需要的时候，可以通过 -h 选项指定远程主机；

[root@dbsvr1 ~]# mysql -h 127.0.0.1 –u root –p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 17

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> exit //退出已登录的mysql环境

Bye

#### 登录时指定库名

[root@sql51 ~]# mysql -uroot -p123456 #未指定库名

mysql> select database();　　　　＃

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

| database() |

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

| NULL |　　　#当前库名

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

1 row in set (0.00 sec)

[root@sql51 ~]# mysql -uroot -p123456 mysql #指定mysql库名

mysql> select database();

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

| database() |

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

| mysql |　　　　#当前库名

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

1 row in set (0.00 sec)

### 步骤二：练习查看/删除/创建库的相关操作

以root用户登入“mysql> ”环境后，可以执行各种MySQL指令、SQL指令。

#### 操作指令类型：

MySQL指令：环境切换，看状态，退出等控制

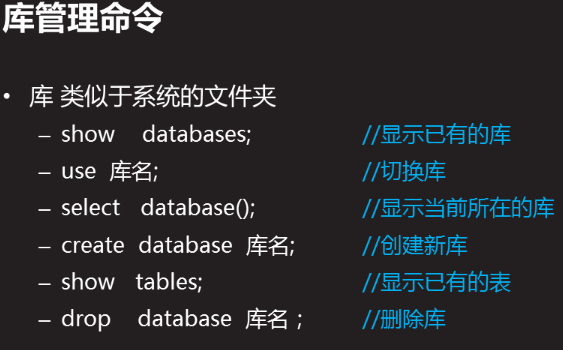
SQL指令：数据库定义／查询／操纵／授权等语句

#### 基本的用法事项如下：

* 操作指令不区分大小写（库名/表名、密码、变量值等除外）。
* 每条SQL指令以 ; 结束或分隔。少数不需要;结束的命令：exit，use
* 不支持 Tab 键自动补齐。
* \c 或Ctrl+c 可废弃当前编写错的操作指令。
* \h帮助信息

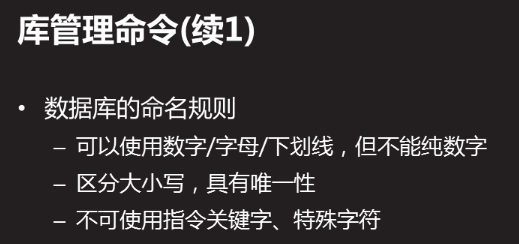


#### 库管理命令



新建一个库，在／var/lib/mysql目录中会多一个对应的文件夹，库删除，文件夹相应删除

#### 数据库命名规则



#### 在登录状态执行linux系统命令

在命令前面加＂system＂

mysql> system ls /var/lib/mysql

auto.cnf ib\_buffer\_pool mysql public\_key.pem

ca-key.pem ibdata1 mysql.sock server-cert.pem

ca.pem ib\_logfile0 mysql.sock.lock server-key.pem

client-cert.pem ib\_logfile1 performance\_schema sys

client-key.pem ibtmp1 private\_key.pem

#### 1）show databases;查看已有的库

mysql> show databases; //查看现有的库

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

| Database |

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

| information\_schema | //信息概要库

| mysql | //授权库

| performance\_schema | //性能结构库

| sys | //系统元数据库

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

4 rows in set (0.15 sec)

#### 2）use切换/使用指定的库

mysql> use sys; //切换到sys库

Database changed

mysql> select database(); //确认当前所在的库

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

| DATABASE() |

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

| sys |

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

1 row in set (0.00 sec)

切换到mysql库：

mysql> use mysql; //切换到mysql库

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> select database(); //确认当前所在的库

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

| DATABASE() |

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

| mysql |

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

1 row in set (0.00 sec)

5 rows in set (0.00 sec)

#### 3）create新建名为newdb的库，确认结果：

mysql> create database newdb; //新建名为newdb的库

Query OK, 1 row affected (0.00 sec)

mysql> show databases;

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

| Database |

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

| information\_schema |

| mydb | //新建的mydb库

| mysql |

| newdb | //新建的newdb库

| performance\_schema |

| sys |

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

6 rows in set (0.00 sec)

#### 4）drop删除指定的库

mysql> drop database newdb; //删除名为newdb的库

Query OK, 0 rows affected (0.01 sec)

mysql> show databases; //确认删除结果，已无newdb库

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

| Database |

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

| information\_schema |

| mydb |

| mysql |

| performance\_schema |

| sys |

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

5 rows in set (0.00 sec)

#### ５）select查看当前所在库

mysql> select database();

### 步骤三：练习查看/删除/创建表/修改表的相关操作

#### 1）show tables查看指定的库里有哪些表

查看mysql库里有哪些表：

mysql> use mysql;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> show tables;

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

| Tables\_in\_mysql |

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

| columns\_priv |

| db |

| engine\_cost |

| event |

| func |

| general\_log |

| gtid\_executed |

| help\_category |

| help\_keyword |

| help\_relation |

| help\_topic |

| innodb\_index\_stats |

| innodb\_table\_stats |

| ndb\_binlog\_index |

| plugin |

| proc |

| procs\_priv |

| proxies\_priv |

| server\_cost |

| servers |

| slave\_master\_info |

| slave\_relay\_log\_info |

| slave\_worker\_info |

| slow\_log |

| tables\_priv |

| time\_zone |

| time\_zone\_leap\_second |

| time\_zone\_name |

| time\_zone\_transition |

| time\_zone\_transition\_type |

| user | //存放数据库用户的表

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

31 rows in set (0.00 sec)

#### 2）desc查看指定表的字段结构

当前库为mysql，查看columns\_priv表的结构，以列表形式展现：

**mysql> desc columns\_priv\G** //查看表结构，以列表形式展现，末尾不用分号

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

Field: Host

Type: char(60)

Null: NO

Key: PRI

Default:

Extra:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Field: Db

Type: char(64)

Null: NO

Key: PRI

Default:

Extra:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Field: User

Type: char(32)

Null: NO

Key: PRI

Default:

Extra:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Field: Table\_name

Type: char(64)

Null: NO

Key: PRI

Default:

Extra:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Field: Column\_name

Type: char(64)

Null: NO

Key: PRI

Default:

Extra:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 6. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Field: Timestamp

Type: timestamp

Null: NO

Key:

Default: CURRENT\_TIMESTAMP

Extra: on update CURRENT\_TIMESTAMP

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 7. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Field: Column\_priv

Type: set('Select','Insert','Update','References')

Null: NO

Key:

Default:

Extra:

7 rows in set (0.01 sec)

查看columns\_priv表的结构，以表格形式展现：

**mysql> desc columns\_priv;**  //查看表结构，以表格形式展现末尾需要有分号

(字段名　　　＋字段类型　　＋是否为空＋是否为主键＋默认值＋描述信息）

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

| Field | Type 　　　 | Null 　| 　Key | Default | Extra |

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

| Host | char(60) | NO | PRI | | |

| Db | char(64) | NO | PRI | | |

| User | char(32) | NO | PRI | | |

| Table\_name | char(64) | NO | PRI | | |

| Column\_name | char(64) | NO | PRI | | |

| Timestamp | timestamp | NO | | CURRENT\_TIMESTAMP | on update CURRENT\_TIMESTAMP |

| Column\_priv | set('Select','Insert','Update','References') | NO | | | |

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

7 rows in set (0.00 sec)

上述操作中，当引用非当前库中的表时，可以用“**库名.表名**”的形式。比如，切换为mysql库再执行“desc columns\_priv;”，与以下操作的效果是相同的：

**mysql> desc mysql.columns\_priv;**

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

| Field | Type | Null | Key | Default | Extra |

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

| Host | char(60) | NO | PRI | | |

| Db | char(64) | NO | PRI | | |

| User | char(16) | NO | PRI | | |

| Table\_name | char(64) | NO | PRI | | |

| Column\_name | char(64) | NO | PRI | | |

| Timestamp | timestamp | NO | | CURRENT\_TIMESTAMP | on update CURRENT\_TIMESTAMP |

| Column\_priv | set('Select','Insert','Update','References') | NO | | | |

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

7 rows in set (0.00 sec)

#### 3）在test库中创建一个名为pwlist的表

包括name、password两列，其中name列作为主键。两个字段值均不允许为空，其中密码列赋予默认空值，相关操作如下所述。

切换到mydb库：

mysql> use mydb;

Database changed

新建pwlist表：

mysql> create table pwlist(

-> name char(16) not null,

-> password char(48)default '',

-> primary key(name)

-> );

Query OK, 0 rows affected (0.38 sec)

确认新创建的表：

mysql> show tables;

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

| Tables\_in\_mydb |

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

| pwlist | //新建的pwlist表

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

1 rows in set (0.01 sec)

查看pwlist表的字段结构：

mysql> desc pwlist;

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

| Field | Type | Null | Key | Default | Extra |

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

| name | char(16) | NO | PRI | NULL | |

| password | char(48) | YES | | | |

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

2 rows in set (0.01 sec)

#### 4）drop删除指定的表

删除当前库中的pwlist表：

mysql> drop table pwlist;

Query OK, 0 rows affected (0.01 sec)

确认删除结果：

mysql> show tables;

Empty set (0.00 sec)

#### 在mydb库中创建新表并-指定字符集

创建一个表，等于就是载／vat/lib/mysql/库名目录/表名文件

表格结构及数据内容如表-1所示。

　　　在MySQL表内存储中文数据时，需要更改字符集（默认为latin1不支持中文），以便MySQL支持存储中文数据记录；比如，可以在创建库或表的时候，手动添加“DEFAULT CHARSET=utf8”来更改字符集。

根据上述表格结构，创建支持中文的student表：

mysql> CREATE TABLE mydb.student(

-> 学号 char(9) NOT NULL,

-> 姓名 varchar(4) NOT NULL,

-> 性别 enum('男','女') NOT NULL,

-> 手机号 char(11) DEFAULT '',

-> 通信地址 varchar(64),

-> PRIMARY KEY(学号)

-> ) DEFAULT CHARSET=utf8; //手工指定字符集，采用utf8

Query OK, 0 rows affected (0.31sec)

注意：若要修改MySQL服务的默认字符集，可以更改服务器的my.cnf配置文件，添加character\_set\_server=utf8 配置，然后重启数据库服务。

[root@dbsvr1 ~]# vim /etc/my.cnf //修改运行服务配置

[mysqld]

.. ..

character\_set\_server=utf8

[root@dbsvr1 ~]# systemctl restart mysqld //重启服务

.. ..

#### 6) 查看desc表的字段结构：

mysql> desc mydb.student;

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

| Field | Type | Null | Key | Default | Extra |

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

| 学号 | char(9) | NO | PRI | NULL | |

| 姓名 | varchar(4) | NO | | NULL | |

| 性别 | enum('男','女') 　| NO | | NULL | |

| 手机号 | char(11) | YES | | | |

| 通信地址 | varchar(64) | YES | | NULL | |

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

5 rows in set (0.00 sec)

#### 7）show create table查看表的实际创建指令：

mysql> show create table mydb.student;

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

|Table |Create Table |

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

| student | CREATE TABLE `student` (

`学号` char(9) NOT NULL,

`姓名` varchar(4) NOT NULL,

`性别` enum('男','女') NOT NULL,

`手机号` char(11) DEFAULT '',

`通信地址` varchar(64) DEFAULT NULL,

PRIMARY KEY (`学号`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 |

+---------+-----------------------------------------------------------------------------------------------1 row in set (0.00 sec)

[root@dbsvr1 ~]# mysql –u root -p

Enter password:

.. ..

mysql> SHOW VARIABLES LIKE 'character%'; //确认更改结果

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

| Variable\_name | Value |

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

| character\_set\_client | utf8 |

| character\_set\_connection | utf8 |

| character\_set\_database | utf8 |

| character\_set\_filesystem | binary |

| character\_set\_results | utf8 |

| character\_set\_server | utf8 |

| character\_set\_system | utf8 |

| character\_sets\_dir | /usr/share/mysql/charsets/ |

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

8 rows in set (0.03 sec

#### show variables like搜索数据库中关键字

**1查看字符集**

mysql> show variables like 'character%';

**2查看权限自己 show grants**

mysql> show grants ; #默认是看root,,,show grants for tom可查看tom的

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

| Grants for root@localhost |

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

| GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'localhost' WITH GRANT OPTION |

| GRANT PROXY ON ''@'' TO 'root'@'localhost' WITH GRANT OPTION |

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

2 rows in set (0.00 sec)

**3查看主从中从库信息**

mysql>show slave status\G;

**4查看是否允许你加载模块**(mysql主从同步复制模块)

mysql> show variables like 'have\_dynamic\_loading' ;

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

| Variable\_name | Value |

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

| have\_dynamic\_loading | YES |

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

1 row in set (0.00 sec)

**5查看binlog日志show master status;**

### 步骤四：记录管理命令

#### 查看表记录select \* from 表名

mysql> select \* from t1;

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

| name | age | sex |

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

| bob | 21 | boy |

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

1 row in set (0.00 sec)

#### 插入表记录insert into 表名 values(值列表)

mysql> insert into t1 values ("bob",19,"boy") ;

#### 修改表记录update 库名.表名 set 字段=值 where

Update 库名.表名 set 字段＝值 where ....

将ygt表中的ygt\_id=3中的数据中的ygt\_id数据改为8

update ygt set ygt\_id=8 where ygt\_id=3;

mysql> select \* from t1;

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

| name | age | sex |

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

| bob | 19 | boy |

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

1 row in set (0.00 sec)

mysql> update db1.t1 set age=21;

Query OK, 1 row affected (0.02 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> select \* from t1;

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

| name | age | sex |

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

| bob | 21 | boy |

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

1 row in set (0.00 sec)

#### delete删除某一条表记录

mysql> select \* from t1;

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

| name | age | sex |

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

| bob | 21 | boy |

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

1 row in set (0.00 sec)

mysql> delete from db1.t1 where name="bob";

Query OK, 1 row affected (0.03 sec)

#### delete删除所有表记录（表还在）

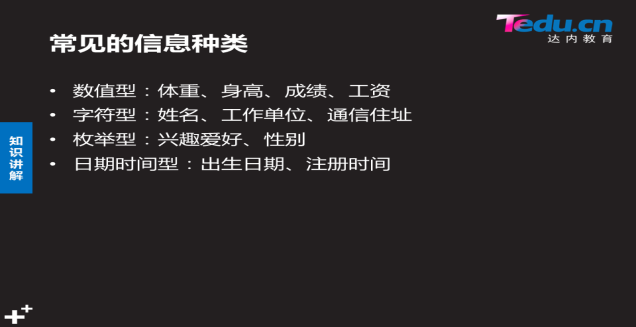
delete from 表名

mysql> delete from db1.t1;

Query OK, 2 rows affected (0.04 sec)

## 3 MySQL 数据类型

### 常见的信息种类



枚举型：指定的值范围中选一个，比如性别整数/

### 浮点数数值类型：



创建表是指定刘字符类型后，默认是有符号的存储范围

mysql> create table t2(age tinyint);

Query OK, 0 rows affected (0.22 sec)

#### 创建表时指定类型为无符号存储范围

mysql> create table t3(age tinyint unsigned);

Query OK, 0 rows affected (0.26 sec)

mysql> desc t2 ;

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

| Field | Type | Null | Key | Default | Extra |

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

| age | tinyint(4) | YES | | NULL | |

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

1 row in set (0.00 sec)

mysql> desc t3;

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

| Field | Type | Null | Key | Default | Extra |

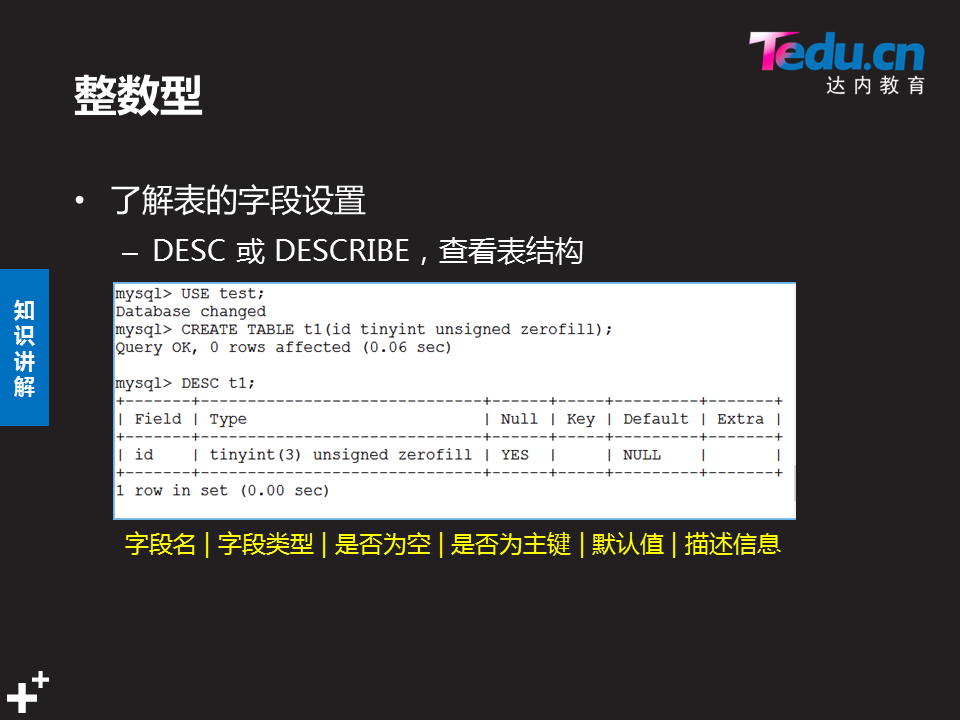
+-------+---------------------+------+-----+---------+-------+

| age | tinyint(3) unsigned | YES | | NULL | |

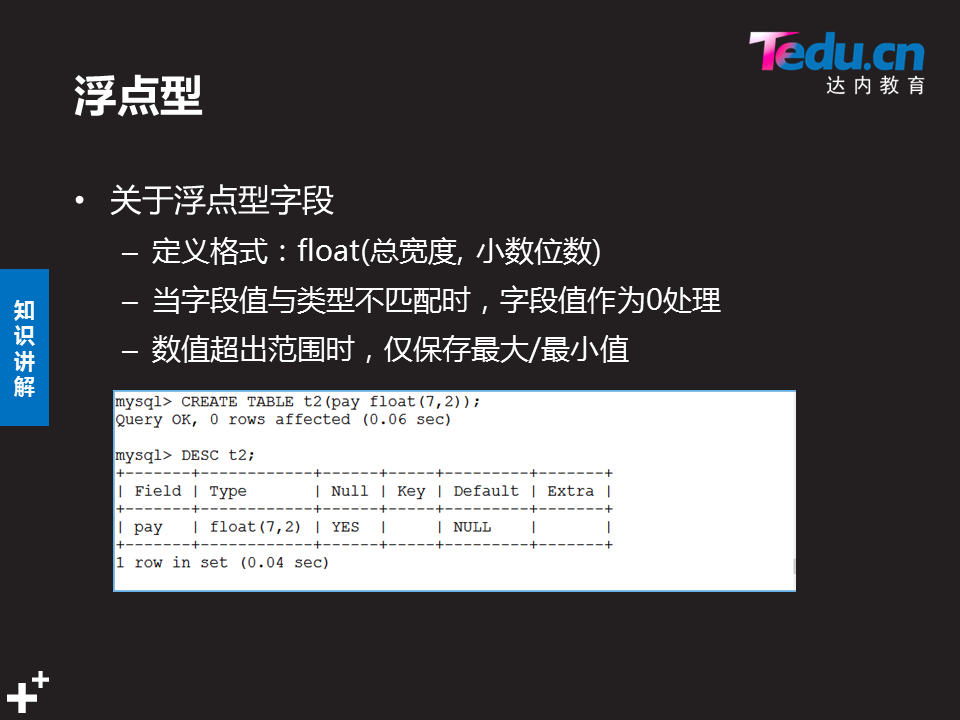
+-------+---------------------+------+-----+---------+-------+

1 row in set (0.00 sec)

#### 整数型



#### 浮点型



mysql> create table t6(pay float(7,2));

Query OK, 0 rows affected (0.25 sec)

mysql> desc t6 ;

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

| Field | Type | Null | Key | Default | Extra |

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

| pay | float(7,2) | YES | | NULL | |

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

1 row in set (0.00 sec)

mysql> insert into t6 values(111.222);

Query OK, 1 row affected (0.04 sec)

mysql> select \* from t6 ;

+--------+

| pay |

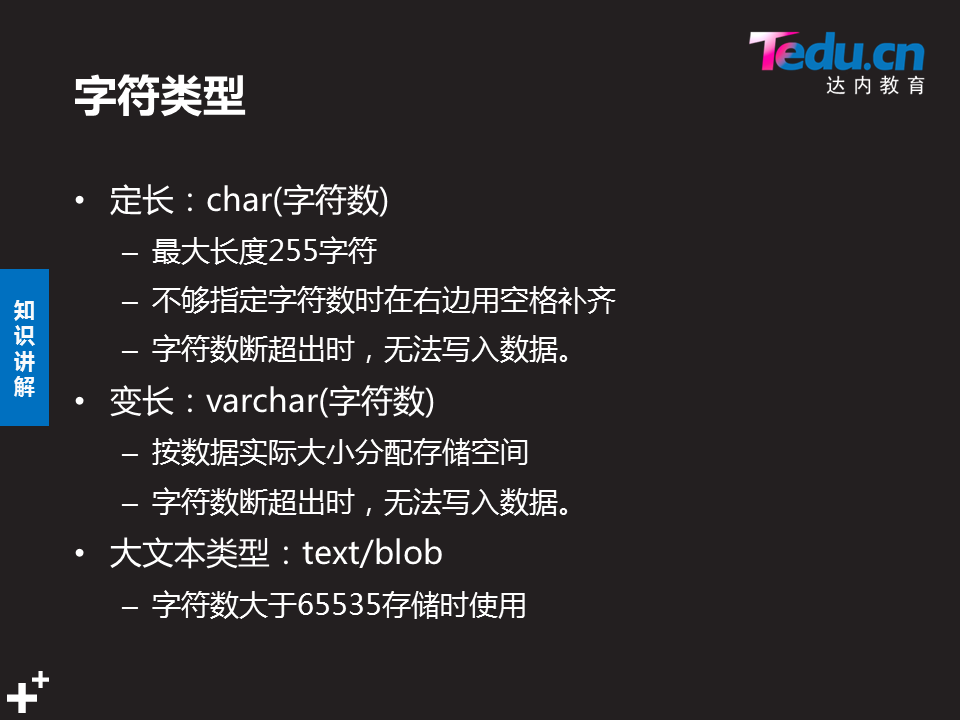
+--------+

| 111.22 |

+--------+

1 row in set (0.00 sec)

### 字符类型char/varchar：



一个英语字母为一个字符

char:未指定字符数，默认为１

　性能比varchar要好，不需要检查数据大小，节省cpu资源

varchar ：必须指定字符数．不然会报错

最大字符长度为：65532字符,

mysql> create table t7 (name char(20),homedir char(15),mail varchar(30));

Query OK, 0 rows affected (0.25 sec)

mysql> insert into t7 values("zdd","hunan","zdd@tedu.cn");

Query OK, 1 row affected (0.07 sec)

mysql> select \* from t7 ;

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

| name | homedir | mail |

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

| zdd | hunan | zdd@tedu.cn |

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

1 row in set (0.01 sec)

### 日期时间类型

**日期时间：datetime**　占用４个字节

范围：1000-01-01 00:00:00.000000～9999-12-31 23:59:59.9999

**日期时间：timestamp** 占用４个字节

范围：1970-01-01 00:00:00.000000~2038-01-19 03:14:07.9999

Datetime和timestamp存储格式：yyyymmddHHMMSS，201901012018

**日　　期：date**　占用４个字节　范围：0001-01-01~9999-12-31　格式:yyyymmdd

**年　　份：year**  占用１个字节　范围：1901－2155 格式:yyyy

**时　　间：time**　占用３个字节 格式：HH:MM:SS

mysql> create table t8 (

-> name char(15),

-> age tinyint unsigned,

-> birthday date,

-> upclass time,

-> party datetime,

-> cs year

-> );

Query OK, 0 rows affected (0.28 sec)

mysql> insert into t8 values("zdd",17,20191123,083000,20190214203000,1991);

Query OK, 1 row affected (0.03 sec)

mysql> desc t8;

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

| Field | Type | Null | Key | Default | Extra |

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

| name | char(15) | YES | | NULL | |

| age | tinyint(3) unsigned | YES | | NULL | |

| birthday | date | YES | | NULL | |

| upclass | time | YES | | NULL | |

| party | datetime | YES | | NULL | |

| cs | year(4) | YES | | NULL | |

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

6 rows in set (0.00 sec)

mysql> select \* from t8;

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

| name | age | birthday | upclass | party | cs |

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

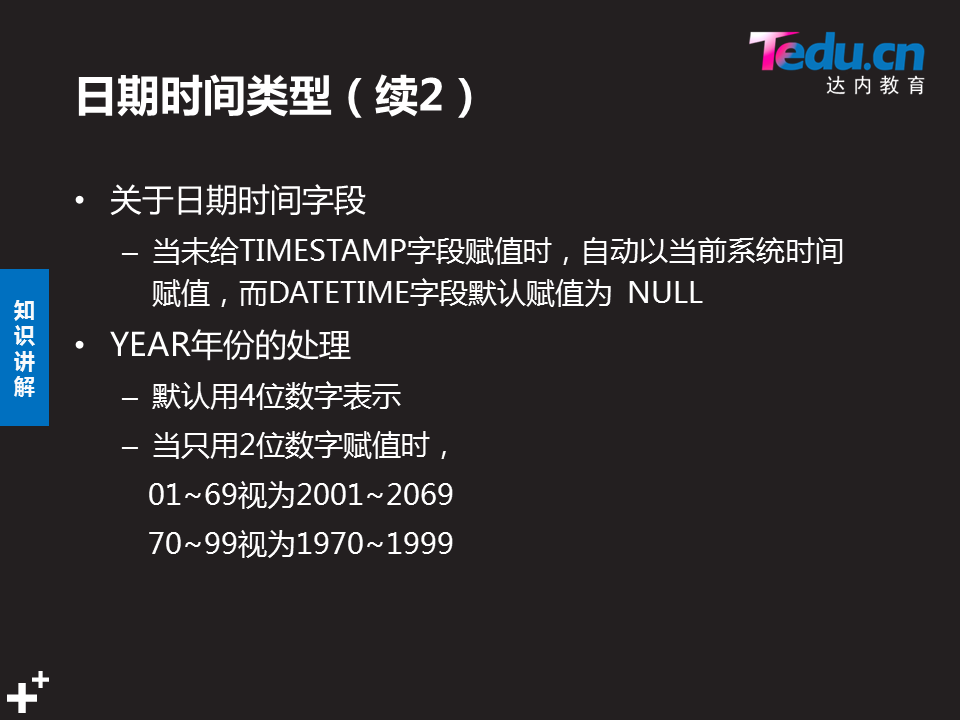
| zdd | 17 | 2019-11-23 | 08:30:00 | 2019-02-14 20:30:00 | 1991 |

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

1 row in set (0.00 sec)

#### Datetime和timestamp日期时间区别

#### 使用两位数给year类型字段赋值



mysql> create table t9(

-> meetting datetime,

-> party timestamp　);

Query OK, 0 rows affected (0.24 sec)

mysql> insert into t9 values(20190128135823,20190504180000);

Query OK, 1 row affected (0.04 sec)

mysql> insert into t9(meetting) values(20190128135823);

Query OK, 1 row affected (0.04 sec)

#party是timestamp日期类型，未插入数据自动用系统时间填充，请如下例第二条数据

mysql> insert into t9(party) values(20190128135823);

Query OK, 1 row affected (0.05 sec)

#meetting是datetime日期类型，未插入数据为空值，请看下例第三条数据

mysql> select \* from t9;

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

| meetting | party |

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

| 2019-01-28 13:58:23 | 2019-05-04 18:00:00 |

| 2019-01-28 13:58:23 | 2019-02-13 17:14:27 |

| NULL | 2019-01-28 13:58:23 |

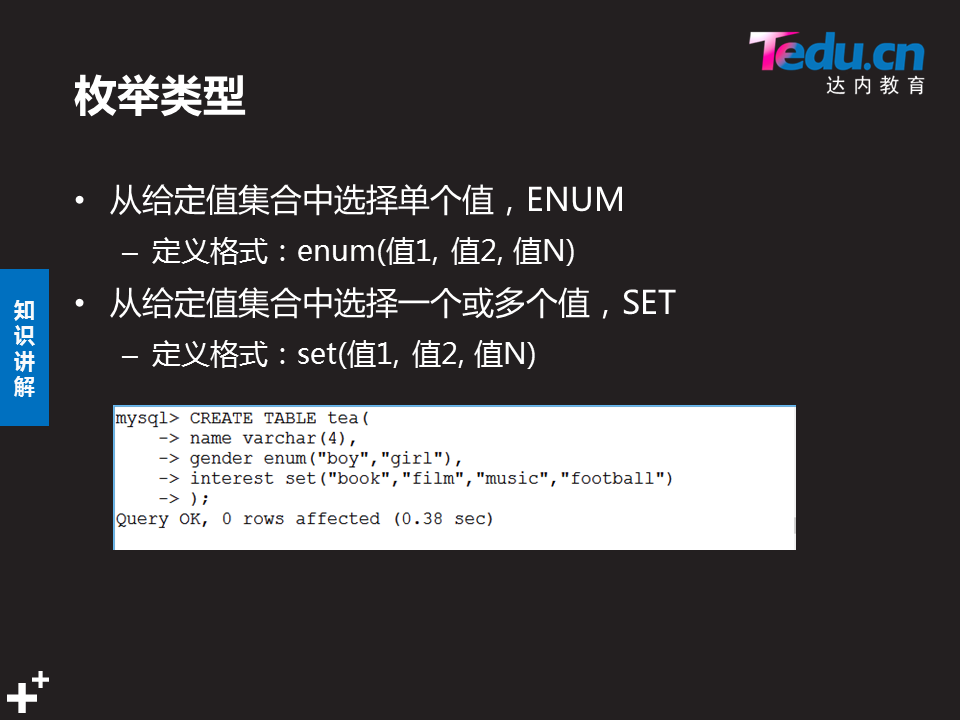
+---------------------+---------------------+

3 rows in set (0.00 sec)

#### 时间函数



### 枚举类型ENUM/SET



mysql> create table t10(

-> name char(10),

-> sex enum("boy","girl"),

-> likes set("hgn","game","sleep","eat")

-> );

Query OK, 0 rows affected (0.44 sec)

mysql> insert into t10 values( "zdd123","man","hgn,game");

ERROR 1265 (01000): Data truncated for column 'sex' at row 1

#”man”超出　enum的集合范围

mysql> insert into t10 values( "zdd123","boy","hgn,game");

Query OK, 1 row affected (0.04 sec)

mysql> select \* from t10 ;

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

| name | sex | likes |

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

| zdd123 | boy | hgn,game |

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

1 row in set (0.00 sec)

### 表中有多个字段，只给其中一个赋值

mysql> insert into t9(meetting) values(20190128135823);

3.1 问题

本案例要求熟悉MySQL的字段数据类型、表结构，各种时间函数的使用的使用，完成以下任务操作：

在studb库里创建stuinfo表，表结构、字段类型自定义

表结构自定义，并合理使用数据类型

练习各种时间函数的使用

3.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：创建studb库、stuinfo表

#### 1）新建studb库，并切换到studb库

mysql> CREATE DATABASE studb;

Query OK, 1 row affected (0.00 sec)

mysql> USE studb;

Database changed

#### 2）新建stuinfo表

假定stuinfo表用来记录每个家庭成员的姓名（name）、性别（gender）、出生日期（birth）、职业（job）、与户主关系（relation）。

mysql> CREATE TABLE stuinfo (

-> name varchar(16) NOT NULL,

-> gender enum('male','femal') DEFAULT 'male',

-> birth date NOT NULL,

-> job varchar(16) DEFAULT '',

-> relation varchar(24) NOT NULL,

-> PRIMARY KEY(name)

-> );

Query OK, 0 rows affected (0.61sec)

查看stuinfo表的字段结构：

mysql> DESC stuinfo;

(字段名　＋字段类型　　　　＋是否为空＋是否为主键＋默认值＋描述信息）

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

| Field | Type | Null | Key | Default | Extra |

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

| name | varchar(16) | NO | PRI | NULL | |

| gender | enum('male','femal') | YES | | male | |

| birth | date | NO | | NULL | |

| job | varchar(16) | YES | | | |

| relation | varchar(24) | NO | | NULL | |

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

5 rows in set (0.00 sec)

### 步骤二：练习各种时间函数的使用

#### 1）使用now()查看当前的日期和时间

mysql> select now();

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

| now() |

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

| 2019-02-13 17:24:31 |

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

1 row in set (0.01 sec)

#### 2）使用sysdate()查看系统日期和时间

mysql> SELECT sysdate();

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

| sysdate() |

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

| 2017-04-02 04:03:21 |

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

1 row in set (0.00 sec)

#### 3）使用curdate()获得当前的日期，不含时间

mysql> SELECT curdate();

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

| curdate() |

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

| 2017-04-02 |

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

1 row in set (0.00 sec)

#### 4）使用curtime()获得当前的时间，不含日期

mysql> SELECT curtime();

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

| curtime() |

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

| 04:04:55 |

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

1 row in set (0.00 sec)

#### 5）分别获取当前日期时间中的年份、月份、日

mysql> SELECT year(now()),month(now()),day(now());

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

| year(now()) | month(now()) | day(now()) |

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

| 2017 | 4 | 2 |

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

1 row in set (0.00 sec)

#### 6）获取系统日期时间中的月份、日

mysql> SELECT month(sysdate()),day(sysdate());

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

| month(sysdate()) | day(sysdate()) |

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

| 4 | 2 |

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

1 row in set (0.00 sec)

#### 7）获取系统日期时间中的时刻

mysql> SELECT time(sysdate());

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

| time(sysdate()) |

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

| 04:06:08 |

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

1 row in set (0.00 sec)

### 步骤三：利用日期函数插入数据



mysql> select \* from t8 ;

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

| name | age | birthday | upclass | party | cs |

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

| zdd | 17 | 2019-11-23 | 08:30:00 | 2019-02-14 20:30:00 | 1991 |

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

1 row in set (0.00 sec)

mysql> insert into db1.t8 values(

-> "zdd2",28,curdate(),curtime(),now(),year(now())　);

Query OK, 1 row affected (0.03 sec)

mysql> select \* from t8;

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

| name | age | birthday | upclass | party | cs |

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

| zdd | 17 | 2019-11-23 | 08:30:00 | 2019-02-14 20:30:00 | 1991 |

| zdd2 | 28 | 2019-02-13 | 17:33:29 | 2019-02-13 17:33:29 | 2019 |

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

2 rows in set (0.00 sec)

# NSD DBA1 DAY02



## 约束条件：（表结构）



Null 空值:”” “null” 都不是空值

Key 键值

Default 字段的默认值

Extra 额外设置

### 操作案例

mysql> select database(); #查看当前所在库

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

| database() |

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

| db2 |

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

1 row in set (0.00 sec)

mysql> create table t1( #在当前库中创建t1表

-> name char(10) not null, #name 不为空

-> age tinyint unsigned not null default 18, #age不为空，默认值18

-> likes set("eat","game","film") default "eat" ); #likes枚举型set,默认值eat

Query OK, 0 rows affected (0.34 sec)

mysql> desc t1; #查看t1表结构

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

| Field | Type | Null | Key | Default | Extra |

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

| name | char(10) | NO | | NULL | |

| age | tinyint(3) unsigned | NO | | 18 | |

| likes | set('eat','game','film') | YES | | eat | |

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

3 rows in set (0.00 sec)

#### Default约束条件(字段默认)

mysql> insert into t1(name) values("bob");

Query OK, 1 row affected (0.03 sec)

mysql> select \* from t1 ;

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

| name | age | likes |

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

| bob | 18 | eat | #age自动插入18,likes自动插入eat

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

1 row in set (0.00 sec)

mysql> insert into t1 values("tom",21,"film");

Query OK, 1 row affected (0.03 sec)

mysql> select \* from t1;

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

| name | age | likes |

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

| bob | 18 | eat |

| tom | 21 | film |

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

2 rows in set (0.00 sec)

#### Null约束条件

mysql> insert into t1 values(null,31,"film");

ERROR 1048 (23000): Column 'name' cannot be null

mysql> insert into t1 values("",31,"film");

Query OK, 1 row affected (0.05 sec)

mysql> insert into t1 values("null",31,"film");

Query OK, 1 row affected (0.05 sec)

mysql> select \* from t1;

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

| name | age | likes |

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

| bob | 18 | eat |

| tom | 21 | film |

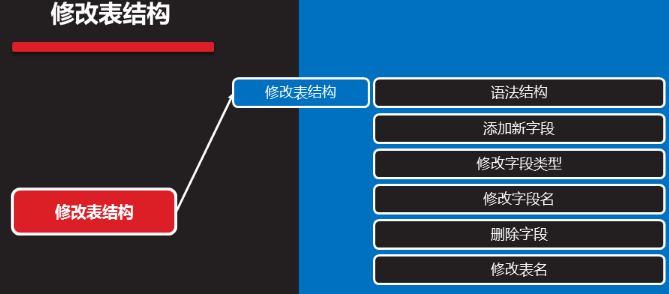
| | 31 | film | #insert into ti values("",31,"film");

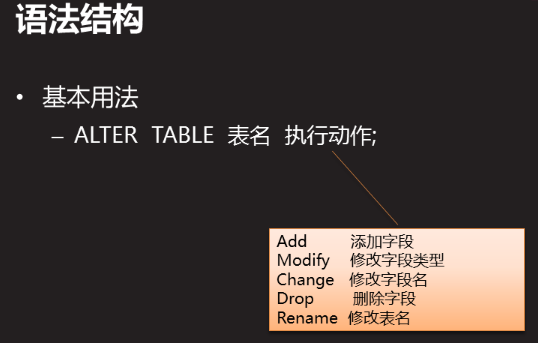
| null | 31 | film | #insert into t1 values("null",31,"film");

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

4 rows in set (0.00 sec)

## 修改表结构





### 表中添加新字段（alter .. add）：

基本用法：

alter table 表名 add 字段名 类型 约束条件; #默认在最后添加

alter table 表名 add 字段名 类型 约束条件 after 字段名a; #字段a后面添加

alter table 表名 add 字段名 类型 约束条件 first; #表最前面添加

mysql> show tables;

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

| Tables\_in\_db2 |

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

| t1 |

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

1 row in set (0.00 sec)

#### first表最前面添加新字段：

mysql> alter table t1 add class char(7) default "nsd1811" first;

Query OK, 0 rows affected (0.49 sec)

Records: 0 Duplicates: 0 Warnings: 0

#### 同时添加两个新字段

mysql> alter table t1 add email varchar(30),add qq varchar(11); Query OK, 0 rows affected (0.61 sec)

Records: 0 Duplicates: 0 Warnings: 0

#### after在某字段后面添加新字段

mysql> alter table t1 add sex enum("man","woman","no") default "no" after name;

Query OK, 0 rows affected (0.38 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc t1;

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

| Field | Type | Null | Key | Default | Extra |

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

| class | char(7) | YES | | nsd1811 | |

| name | char(10) | NO | | NULL | |

| sex | enum('man','woman','no') | YES | | no | |

| age | tinyint(3) unsigned | NO | | 18 | |

| likes | set('eat','game','film') | YES | | eat | |

| email | varchar(30) | YES | | NULL | |

| qq | varchar(11) | YES | | NULL | |

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

7 rows in set (0.00 sec)

### 修改字段类型（modify）

基本用法：

Alter table modify 字段名 类型（宽度） 约束条件;

Alter table modify 字段名 类型(宽度) 约束条件 after 字段a; #将字段放到字段a后

Alter table modify 字段名 类型（宽度） 约束条件 first;

1. 修改哪一部分就写新的，不修改就原样复制（否则变成默认值），
2. 如果该字段已有数据，修改的约束条件不能与原数据冲突

#### 修改前表结构和表内容：

mysql> desc t1 ;

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

| Field | Type | Null | Key | Default | Extra |

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

| class | char(7) | YES | | nsd1811 | |

| name | char(10) | NO | | NULL | |

| sex | enum('man','woman','no') | YES | | no | |

| age | tinyint(3) unsigned | NO | | 18 | |

| likes | set('eat','game','film') | YES | | eat | |

| email | varchar(30) | YES | | NULL | |

| qq | varchar(11) | YES | | NULL | |

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

7 rows in set (0.00 sec)

mysql> select \* from t1;

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

| class | name | sex | age | likes | email | qq |

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

| nsd1811 | bob | no | 18 | eat | NULL | NULL |

| nsd1811 | tom | no | 21 | film | NULL | NULL |

| nsd1811 | | no | 31 | film | NULL | NULL |

| nsd1811 | null | no | 31 | film | NULL | NULL |

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

4 rows in set (0.00 sec)

mysql> alter table t1 modify sex enum('man','woman','no') not null default "no";

Query OK, 0 rows affected (0.36 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table t1 modify class varchar(15);

Query OK, 4 rows affected (0.41 sec)

Records: 4 Duplicates: 0 Warnings: 0

mysql> alter table t1 modify age tinyint(3) unsigned not null default 18 after name;

Query OK, 0 rows affected (0.47 sec)

Records: 0 Duplicates: 0 Warnings: 0

#### 修改后表结构和表内容

mysql> desc t1;

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

| Field | Type | Null | Key | Default | Extra |

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

| class | varchar(15) | YES | | NULL | |

| name | char(10) | NO | | NULL | |

| age | tinyint(3) unsigned | NO | | 18 | | #在sex前面了

| sex | enum('man','woman','no') | NO | | no | |

| likes | set('eat','game','film') | YES | | eat | |

| email | varchar(30) | YES | | NULL | |

| qq | varchar(11) | YES | | NULL | |

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

7 rows in set (0.00 sec)

mysql> select \* from t1;

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

| class | name | age | sex | likes | email | qq |

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

| nsd1811 | bob | 18 | no | eat | NULL | NULL |

| nsd1811 | tom | 21 | no | film | NULL | NULL |

| nsd1811 | | 31 | no | film | NULL | NULL |

| nsd1811 | null | 31 | no | film | NULL | NULL |

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

4 rows in set (0.00 sec)

### 修改字段名（change）

基本用法：

Alter table change 源字段名 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

mysql> alter table t1 change email mail varchar(30);

Query OK, 0 rows affected (0.06 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc t1;

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

| Field | Type | Null | Key | Default | Extra |

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

| class | varchar(15) | YES | | NULL | |

| name | char(10) | NO | | NULL | |

| age | tinyint(3) unsigned | NO | | 18 | |

| sex | enum('man','woman','no') | NO | | no | |

| likes | set('eat','game','film') | YES | | eat | |

| mail | varchar(30) | YES | | NULL | |

| qq | varchar(11) | YES | | NULL | |

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

7 rows in set (0.00 sec)

### 删除字段名（drop）

基本用法：

Alter table 表名 drop 字段名1；

Alter table 表名 drop 字段名1，drop 字段名2；

mysql> alter table t1 drop mail , drop qq;

Query OK, 0 rows affected (0.41 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc t1;

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

| Field | Type | Null | Key | Default | Extra |

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

| class | varchar(15) | YES | | NULL | |

| name | char(10) | NO | | NULL | |

| age | tinyint(3) unsigned | NO | | 18 | |

| sex | enum('man','woman','no') | NO | | no | |

| likes | set('eat','game','film') | YES | | eat | |

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

5 rows in set (0.00 sec)

### 修改表名（rename）

基本用法：Alter table 表名 rename 新表名

修改表名之后

mysql> show tables;

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

| Tables\_in\_db2 |

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

| t1 |

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

1 row in set (0.00 sec)

mysql> alter table t1 rename stuinfo;

Query OK, 0 rows affected (0.08 sec)

mysql> show tables ;

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

| Tables\_in\_db2 |

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

| stuinfo |

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

1 row in set (0.00 sec)

mysql> system ls /var/lib/mysql/db2

db.opt stuinfo.frm stuinfo.ibd

### 清空表数据,结构不变(delete)

mysql> select \* from stuinfo;

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

| class | name | age | sex | likes |

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

| nsd1811 | bob | 18 | no | eat |

| nsd1811 | tom | 21 | no | film |

| nsd1811 | | 31 | no | film |

| nsd1811 | null | 31 | no | film |

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

4 rows in set (0.00 sec)

mysql> delete from stuinfo;

Query OK, 4 rows affected (0.04 sec) #有数据时删除的4条数据

mysql> show tables;

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

| Tables\_in\_db2 |

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

| stuinfo |

| t1 |

| t2 |

| t3 |

| t4 |

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

5 rows in set (0.00 sec)

mysql> select \* from stuinfo;

Empty set (0.00 sec)

mysql> delete from stuinfo;

Query OK, 0 rows affected (0.00 sec) #没有数据时删除,0条数据

## Mysql 键值

### 键值类型：

Index:普通索引 Unique:唯一索引 Fulltext:全文索引

Primary key:主键 Foreign key:外键

### Index普通索引

#### 索引

索引:是对记录集的多个字段进行排序的方法,类似于书的目录

索引优点:通过创建唯一性索引,可以保证数据库中每一行数据的唯一性

可以加快数据的检索速度

索引缺点:对数据进行增删改的时候,索引也要动态维护,降低了数据维护速度

索引占用物理空间

索引类型(排序方法)包括:btree b+tree(二叉树,默认使用) hash

#### Index使用说明属于键值:

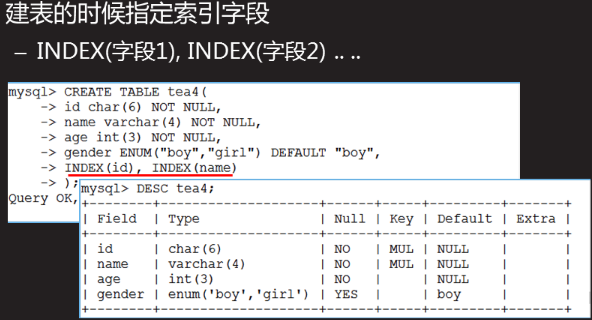
一个表中可以有多个index 字段

字段值允许有重复,且可以赋null值

经常把做查询条件的字段设置为index字段

Index字段的key标志是MUL

#### 建表的时候指定索引字段



mysql> system ls /var/lib/mysql/db2

db.opt stuinfo.frm stuinfo.ibd

mysql> create table t1(

-> name char(10),

-> age int,

-> class char(7),

-> index(name),index(age) );

Query OK, 0 rows affected (0.42 sec)

mysql> desc t1 ;

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

| Field | Type | Null | Key | Default | Extra |

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

| name | char(10) | YES | MUL | NULL | |

| age | int(11) | YES | MUL | NULL | |

| class | char(7) | YES | | NULL | |

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

3 rows in set (0.00 sec)

#### 查看索引

mysql> show index from t1;

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

| Table | Non\_unique | Key\_name | Seq\_in\_index | Column\_name | Collation | Cardinality | Sub\_part | Packed | Null | Index\_type | Comment | Index\_comment |

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

| t1 | 1 | name | 1 | name | A | 0 | NULL | NULL | YES | BTREE | | |

| t1 | 1 | age | 1 | age | A | 0 | NULL | NULL | YES | BTREE | | |

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

2 rows in set (0.00 sec)

mysql> show index from t1\G; #将显示内容竖向显示,一定要大写G

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

Table: t1

Non\_unique: 1

Key\_name: name

Seq\_in\_index: 1

Column\_name: name

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null: YES

Index\_type: BTREE

Comment:

Index\_comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: t1

Non\_unique: 1

Key\_name: age

Seq\_in\_index: 1

Column\_name: age

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null: YES

Index\_type: BTREE

Comment:

Index\_comment:

2 rows in set (0.00 sec)

#### 在已有表中指定索引字段

Creater index 索引名 on 表名(字段名);

mysql> desc t2

-> ;

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

| Field | Type | Null | Key | Default | Extra |

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

| name | char(10) | YES | | NULL | |

| age | int(11) | YES | | NULL | |

| class | char(7) | YES | | NULL | |

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

3 rows in set (0.00 sec)

mysql> create index age on t2(age);

Query OK, 0 rows affected (0.19 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create index abc on t2(name);

Query OK, 0 rows affected (0.16 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc t2

-> ;

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

| Field | Type | Null | Key | Default | Extra |

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

| name | char(10) | YES | MUL | NULL | |

| age | int(11) | YES | MUL | NULL | |

| class | char(7) | YES | | NULL | |

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

3 rows in set (0.00 sec)

mysql> show index from t2\G;

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

Table: t2

Non\_unique: 1

Key\_name: age

Seq\_in\_index: 1

Column\_name: age

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null: YES

Index\_type: BTREE

Comment:

Index\_comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: t2

Non\_unique: 1

Key\_name: abc

Seq\_in\_index: 1

Column\_name: name

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null: YES

Index\_type: BTREE

Comment:

Index\_comment:

2 rows in set (0.00 sec)

ERROR:

No query specified

#### 删除指定表中的索引字段

Drop index 索引名 on 表明;

### Primary key 主键

#### Primary key使用说明:

1. 一个表中只能有一个primary key 字段
2. 对应的字段值不允许有重复,且不允许赋null值
3. 如果有多个字段都作为primary key 称为复合主键,必须一起创建
4. 主键字段的key标志是PRI
5. 通常与Auto\_Increment(自增属性)连用
6. 经常把表中是唯一标识记录的字段设置为主键字段:如[记录编号字段]

#### 建表时候创建主键

第1种:在末尾添加

mysql> create table t3(

-> stu\_num char(9),

-> name char(10),

-> age int(11),

-> primary key (stu\_num) );

Query OK, 0 rows affected (0.30 sec)

mysql> desc t3 ;

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

| Field | Type | Null | Key | Default | Extra |

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

| stu\_num | char(9) | NO | PRI | NULL | |

| name | char(10) | YES | | NULL | |

| age | int(11) | YES | | NULL | |

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

3 rows in set (0.00 sec)

第2种:在字段后面 输入关键字

mysql> create table t4(

-> stu\_num char(9) primary key,

-> name char(10),

-> age int );

Query OK, 0 rows affected (0.26 sec)

mysql> desc t4;

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

| Field | Type | Null | Key | Default | Extra |

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

| stu\_num | char(9) | NO | PRI | NULL | |

| name | char(10) | YES | | NULL | |

| age | int(11) | YES | | NULL | |

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

3 rows in set (0.00 sec)

#### 在已有表里添加主键

选择一个无重复/无空值的字段,下面操作是先清空stuinfo表数据,再添加stu\_num字段做为主键

mysql> select \* from stuinfo;

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

| class | name | age | sex | likes |

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

| nsd1811 | bob | 18 | no | eat |

| nsd1811 | tom | 21 | no | film |

| nsd1811 | | 31 | no | film |

| nsd1811 | null | 31 | no | film |

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

4 rows in set (0.00 sec)

mysql> delete from stuinfo; #清空stuinfo表数据,结构还在

Query OK, 4 rows affected (0.04 sec)

mysql> show tables;

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

| Tables\_in\_db2 |

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

| stuinfo |

| t1 |

| t2 |

| t3 |

| t4 |

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

5 rows in set (0.00 sec)

mysql> select \* from stuinfo;

Empty set (0.00 sec)

mysql> alter table stuinfo add stu\_num char(9) primary key after class;

Query OK, 0 rows affected (0.44 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc stuinfo;

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

| Field | Type | Null | Key | Default | Extra |

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

| class | varchar(15) | YES | | NULL | |

| stu\_num | char(9) | NO | PRI | NULL | |

| name | char(10) | NO | | NULL | |

| age | tinyint(3) unsigned | NO | | 18 | |

| sex | enum('man','woman','no') | NO | | no | |

| likes | set('eat','game','film') | YES | | eat | |

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

6 rows in set (0.00 sec)

#### 在已有表中添加主键2

将gzt表中的gzt\_id字段设置为主键

mysql> alter table gzt add primary key(gzt\_id);

#### 删除主键

alter table 表名 modify id int not null #如果有自增属性,必须先去掉

alter table 表名 drop primarey key;

主键删除后,字段可以有重复数据了,但是空值不行,需要修改

mysql> desc t3 ;

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

| Field | Type | Null | Key | Default | Extra |

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

| stu\_num | char(9) | NO | PRI | NULL | |

| name | char(10) | YES | | NULL | |

| age | int(11) | YES | | NULL | |

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

3 rows in set (0.00 sec)

mysql> alter table t3 drop primary key;

Query OK, 0 rows affected (0.39 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc t3;

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

| Field | Type | Null | Key | Default | Extra |

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

| stu\_num | char(9) | NO | | NULL | |

| name | char(10) | YES | | NULL | |

| age | int(11) | YES | | NULL | |

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

3 rows in set (0.00 sec)

#### 创建复合主键(必须同时添加):

复合主键作用:插入多条数据,只要复合主键的值没有同时重复,即可插入成功,

mysql> create table t5(

-> name char(10),

-> class char(7),

-> status enum("yes","no"),

-> primary key (name,class) );

Query OK, 0 rows affected (0.19 sec)

mysql> desc t5;

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

| Field | Type | Null | Key | Default | Extra |

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

| name | char(10) | NO | PRI | NULL | |

| class | char(7) | NO | PRI | NULL | |

| status | enum('yes','no') | YES | | NULL | |

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

3 rows in set (0.00 sec)

**验证插入多条数据,只要复合主键的值没有同时重复,即可插入成功,**

mysql> insert into t5 values("bob","1811","no");

Query OK, 1 row affected (0.04 sec)

mysql> insert into t5 values("bob","1811","yes");

ERROR 1062 (23000): Duplicate entry 'bob-1811' for key 'PRIMARY'

mysql> insert into t5 values("bob","1812","yes");

Query OK, 1 row affected (0.05 sec)

mysql> insert into t5 values("dc","1812","no");

Query OK, 1 row affected (0.03 sec)

mysql> select \* from t5;

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

| name | class | status |

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

| bob | 1811 | no |

| bob | 1812 | yes |

| dc | 1812 | no |

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

3 rows in set (0.00 sec)

#### 与auto\_increment(自增长)连用

Auto\_increment:作用是让字段的值自增长

字段类型必须是数值类型,且必须是主键

如果定义自增长的字段的值,就不自增长添加,按照你定义的值添加,添加的不能重复

自增长的值都是按照上一条数据的值自增长1,不管数据有没有被删除

mysql> create table t6(

-> id int primary key auto\_increment,

-> name char(10),

-> sex enum("man","woman") );

Query OK, 0 rows affected (0.34 sec)

mysql> desc t6 ;

(字段名　　 ＋字段类型 ＋是否为空＋是否为主键＋默认值＋描述信息）

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | NO | PRI | NULL | auto\_increment |

| name | char(10) | YES | | NULL | |

| sex | enum('man','woman') | YES | | NULL | |

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

3 rows in set (0.00 sec)

mysql> insert into t6(name,sex) values("jack","woman");

Query OK, 1 row affected (0.06 sec)

mysql> insert into t6(name,sex) values("bob","woman");

Query OK, 1 row affected (0.04 sec)

mysql> insert into t6 values(1,"lili","woman");

ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'

mysql> insert into t6 values(null,"lili","woman");

Query OK, 1 row affected (0.04 sec)

mysql> select \* from t6;

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

| id | name | sex |

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

| 1 | jack | woman |

| 2 | bob | woman |

| 3 | lili | woman |

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

3 rows in set (0.00 sec)

mysql> insert into t6 values(9,"lili","woman");

Query OK, 1 row affected (0.04 sec)

mysql> insert into t6(name,sex) values("zdd","woman");

Query OK, 1 row affected (0.05 sec)

mysql> select \* from t6;

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

| id | name | sex |

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

| 1 | jack | woman |

| 2 | bob | woman |

| 3 | lili | woman |

| 9 | lili | woman |

| 10 | zdd | woman |

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

5 rows in set (0.00 sec)

#### 删除有auto\_increment(自增长)的主键

alter table 表名 modify id int not null #如果有自增属性,必须先去掉

mysql> desc t6;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | NO | PRI | NULL | auto\_increment |

| name | char(10) | YES | | NULL | |

| sex | enum('man','woman') | YES | | NULL | |

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

3 rows in set (0.00 sec)

mysql> alter table t6 drop primary key;

ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key

mysql> alter table t6 modify id int not null;

Query OK, 5 rows affected (0.52 sec)

Records: 5 Duplicates: 0 Warnings: 0

mysql> alter table t6 drop primary key;

Query OK, 5 rows affected (0.49 sec)

Records: 5 Duplicates: 0 Warnings: 0

mysql> desc t6;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | NO | | NULL | |

| name | char(10) | YES | | NULL | |

| sex | enum('man','woman') | YES | | NULL | |

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

3 rows in set (0.00 sec)

### Foreign key 外键

#### 什么是外键:

外键作用:让当前表字段的值在另一个表中字段值的范围内选择

A表中的name字段设置了外键,参考的是B表中的user字段

A表中的name字段数据内容必须在B表中的user字段内容选择

B表中的user字段必须是主键

#### 使用外键的条件:

1. 表的存储引擎必须是innodb
2. 字段类型要一致
3. 被参照字段必须要是索引类型的一种(primary key)

外键规则

#### 创建外键

Create table 表A名(

字段名1 类型 ,

字段名2 类型 ,

Foreign key (表A的字段名1) references 表B名(字段名)

On update cascade #表A与表B名(字段名)数据同步更新

On delete cascade #表A与表B名(字段名)数据同步删除

)engine=innodb;

练习

创建员工表ygt ygt\_id为主键,自增长 存储引擎为innodb;

mysql> create table ygt(

-> ygt\_id int primary key auto\_increment,

-> name char(10),

-> sex enum("b","g")

-> )engine=innodb;

Query OK, 0 rows affected (0.20 sec)

插入三条数据,ygt\_id为自增长,不需要定义数据

mysql> insert into ygt(name,sex) values("tom","b"),("lucy","g"),("jack","g");

Query OK, 3 rows affected (0.05 sec)

Records: 3 Duplicates: 0 Warnings: 0

mysql> select \* from ygt-> ;

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

| ygt\_id | name | sex |

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

| 1 | tom | b |

| 2 | lucy | g |

| 3 | jack | g |

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

3 rows in set (0.00 sec)

创建工资表gzt 创建外键,定义gzt\_id为外键,参考ygt表的(ygt\_id)字段,同步更新与删除

mysql> create table gzt(

-> gzt\_id int,

-> pay float(7,2),

-> foreign key(gzt\_id) references ygt(ygt\_id) on update cascade on delete cascade

-> )engine=innodb;

Query OK, 0 rows affected (0.38 sec)

mysql> desc gzt;

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

| Field | Type | Null | Key | Default | Extra |

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

| gzt\_id | int(11) | YES | MUL | NULL | |

| pay | float(7,2) | YES | | NULL | |

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

2 rows in set (0.00 sec)

查看表创建详情

mysql> show create table gzt;

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

| Table | Create Table |

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

| gzt | CREATE TABLE `gzt` (

`gzt\_id` int(11) DEFAULT NULL,

`pay` float(7,2) DEFAULT NULL,

KEY `gzt\_id` (`gzt\_id`),

CONSTRAINT `gzt\_ibfk\_1` FOREIGN KEY (`gzt\_id`) REFERENCES `ygt` (`ygt\_id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1 |

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

1 row in set (0.00 sec)

**解释:**CONSTRAINT `gzt\_ibfk\_1` 为外键名称

FOREIGN KEY (`gzt\_id`) 外键创建在(`gzt\_id`)字段上

REFERENCES `ygt` (`ygt\_id`) 参考为ygt表的ygt\_id字段

ON DELETE CASCADE 跟参考同步更新 ON UPDATE CASCADE 跟参考同步删除

ENGINE=InnoDB 存储引擎为InnoDB

#### 验证外键:

mysql> select \* from ygt;

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

| ygt\_id | name | sex |

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

| 1 | tom | b |

| 2 | lucy | g |

| 3 | jack | g |

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

3 rows in set (0.00 sec)

mysql> insert into gzt values(1,20000);

Query OK, 1 row affected (0.02 sec)

mysql> insert into gzt values(2,30000);

Query OK, 1 row affected (0.02 sec)

mysql> insert into gzt values(3,40000);

Query OK, 1 row affected (0.01 sec)

mysql> insert into gzt values(4,50000); #增加报错,ygt表只有1,2,3的id

ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`db2`.`gzt`, CONSTRAINT `gzt\_ibfk\_1` FOREIGN KEY (`gzt\_id`) REFERENCES `ygt` (`ygt\_id`) ON DELETE CASCADE ON UPDATE CASCADE)

mysql> select \* from gzt;

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

| gzt\_id | pay |

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

| 1 | 20000.00 |

| 2 | 30000.00 |

| 3 | 40000.00 |

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

3 rows in set (0.00 sec)

给ygt表中添加一条数据,id自动为4, gzt表的第四条数据也可以增加

mysql> insert into ygt(name,sex) values("bob","b");

Query OK, 1 row affected (0.03 sec)

mysql> insert into gzt values(4,50000);

Query OK, 1 row affected (0.03 sec)

mysql> select \* from gzt;

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

| gzt\_id | pay |

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

| 1 | 20000.00 |

| 2 | 30000.00 |

| 3 | 40000.00 |

| 4 | 50000.00 |

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

4 rows in set (0.00 sec)

同步更新:修改了ygt表中ygt\_id=2的ygt\_id值为8,gzt中的gzt\_id=2的也变成了8

mysql> select \* from gzt;

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

| gzt\_id | pay |

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

| 1 | 20000.00 |

| 2 | 30000.00 |

| 3 | 40000.00 |

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

3 rows in set (0.00 sec)

mysql> update ygt set ygt\_id=8 where ygt\_id=3;

Query OK, 1 row affected (0.04 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> select \* from gzt;

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

| gzt\_id | pay |

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

| 1 | 20000.00 |

| 2 | 30000.00 |

| 8 | 40000.00 |

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

3 rows in set (0.00 sec)

同步删除:删除了ygt表中的ygt\_id=4的数据,gzt中的gzt\_id数据也没有了

mysql> delete from ygt where ygt\_id=4;

Query OK, 1 row affected (0.04 sec)

mysql> select \* from gzt;

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

| gzt\_id | pay |

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

| 1 | 20000.00 |

| 2 | 30000.00 |

| 3 | 40000.00 |

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

3 rows in set (0.00 sec)

#### 删除外键

Alter table 表名 drop foreign key 约束名(外键名)

先查询出表创建过程

mysql> show create table gzt;

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

| Table | Create Table

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

| gzt | CREATE TABLE `gzt` (

`gzt\_id` int(11) NOT NULL,

`pay` float(7,2) DEFAULT NULL,

PRIMARY KEY (`gzt\_id`),

CONSTRAINT `gzt\_ibfk\_1` FOREIGN KEY (`gzt\_id`) REFERENCES `ygt` (`ygt\_id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=latin1 |

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

1 row in set (0.00 sec)

再执行删除

mysql> alter table gzt drop foreign key gzt\_ibfk\_1; #查询出来粘贴删除

Query OK, 0 rows affected (0.08 sec)

Records: 0 Duplicates: 0 Warnings: 0

再查看

mysql> show create table gzt;

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

| Table | Create Table

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

| gzt | CREATE TABLE `gzt` (

`gzt\_id` int(11) NOT NULL,

`pay` float(7,2) DEFAULT NULL,

PRIMARY KEY (`gzt\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1 |

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

1 row in set (0.00 sec)

#### 已有表中添加外键

要保证外键字段的数据值与参考表中参考字段的值相匹配

mysql> alter table gzt add foreign key(gzt\_id) references ygt(ygt\_id) on update cascade on delete cascade;

#### 注意事项

当表中有外键时:被参考的表和被参考的字段都不能删除

被参考的表的所有记录被删除,外键表的所有记录也将被删除

## 案例

1. 修改表结构
2. MySQL索引创建与删除

## 1 修改表结构

1.1 问题

本案例要求熟悉MySQL库中表的字段修改，主要练习以下操作：

添加字段

修改字段名

修改字段类型

删除字段

1.2 步骤

实现此案例需要按照如下步骤进行。

## 步骤一：添加字段

在studb中创建tea6表

mysql> CREATE TABLE studb.tea6(id int(4) PRIMARY KEY,

-> name varchar(4) NOT NULL,

-> age int(2) NOT NULL

-> );

Query OK, 0 rows affected (0.34 sec)

为tea6表添加一个address字段

添加前：

mysql> DESC tea6;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(4) | NO | PRI | NULL | |

| name | varchar(4) | NO | | NULL | |

| age | int(2) | NO | | NULL | |

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

3 rows in set (0.00 sec)

添加address字段：

mysql> ALTER TABLE tea6 ADD address varchar(48);

Query OK, 0 rows affected (0.84 sec)

Records: 0 Duplicates: 0 Warnings: 0

添加后（默认作为最后一个字段）：

mysql> DESC tea6;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(4) | NO | PRI | NULL | |

| name | varchar(4) | NO | | NULL | |

| age | int(2) | NO | | NULL | |

| address | varchar(48) | YES | | NULL | |

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

4 rows in set (0.00 sec)

3）在tea6表的age列之后添加一个gender字段

添加操作：

mysql> ALTER TABLE tea6 ADD gender enum('boy','girl') AFTER age;

Query OK, 0 rows affected (0.59 sec)

Records: 0 Duplicates: 0 Warnings: 0

确认添加结果：

mysql> DESC tea6;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(4) | NO | PRI | NULL | |

| name | varchar(4) | NO | | NULL | |

| age | int(2) | NO | | NULL | |

| gender | enum('boy','girl') | YES | | NULL | |

| address | varchar(48) | YES | | NULL | |

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

5 rows in set (0.00 sec)

## 步骤二：修改字段名和字段类型

将tea6表的gender字段改名为sex，并添加非空约束

修改操作：

mysql> ALTER TABLE tea6 CHANGE gender

-> sex enum('boy','girl') NOT NULL;

Query OK, 0 rows affected (0.08 sec)

Records: 0 Duplicates: 0 Warnings: 0

确认修改结果：

mysql> DESC tea6;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(4) | NO | PRI | NULL | |

| name | varchar(4) | NO | | NULL | |

| age | int(2) | NO | | NULL | |

| sex | enum('boy','girl') | NO | | NULL | |

| address | varchar(48) | YES | | NULL | |

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

5 rows in set (0.00 sec)

## 步骤三：删除字段

删除tea6表中名为sex的字段：

mysql> ALTER TABLE tea6 DROP sex; //删除操作

Query OK, 0 rows affected (0.52 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC tea6; //确认删除结果

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(4) | NO | PRI | NULL | |

| name | varchar(4) | NO | | NULL | |

| age | int(2) | NO | | NULL | |

| address | varchar(48) | YES | | NULL | |

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

4 rows in set (0.00 sec)

## 2 MySQL索引创建与删除

2.1 问题

本案例要求熟悉MySQL索引的类型及操作方法，主要练习以下任务：

普通索引、唯一索引、主键索引的创建/删除

自增主键索引的创建/删除

建立员工表yg、工资表gz，数据内容如表-1、表-2所示，设置外键实现同步更新与同步删除

表-1 员工表yg的数据

表-2 工资表gz的数据

2.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：索引的创建与删除

创建表的时候指定INDEX索引字段

创建库home：

mysql> create database home;

Query OK, 1 row affected (0.00 sec)

允许有多个INDEX索引字段。比如，以下操作在home库中创建了tea4表，将其中的id、name作为索引字段：

mysql> USE home;

Database changed

mysql> CREATE TABLE tea4(

-> id char(6) NOT NULL,

-> name varchar(6) NOT NULL,

-> age int(3) NOT NULL,

-> gender ENUM('boy','girl') DEFAULT 'boy',

-> INDEX(id),INDEX(name)

-> );

Query OK, 0 rows affected (0.59 sec)

查看新建tea4表的字段结构，可以发现两个非空索引字段的KEY标志为MUL：

mysql> DESC tea4;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | char(6) | NO | MUL | NULL | |

| name | varchar(6) | NO | MUL | NULL | |

| age | int(3) | NO | | NULL | |

| gender | enum('boy','girl') | YES | | boy | |

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

4 rows in set (0.00 sec)

2）删除现有表的某个INDEX索引字段

比如，删除tea4表中名称为named的INDEX索引字段：

mysql> drop INDEX name ON tea4; //删除name字段的索引

Query OK, 0 rows affected (0.18 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC tea4; //确认删除结果

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

| Field | Type | Null | Key | Default | Extra |

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

| id | char(6) | NO | MUL | NULL | |

| name | varchar(6) | NO | | NULL | |

| age | int(3) | NO | | NULL | |

| gender | enum('boy','girl') | YES | | boy | |

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

4 rows in set (0.00 sec)

3）在已有的某个表中设置INDEX索引字段

比如，针对tea4表的age字段建立索引，名称为 nianling：

mysql> CREATE INDEX nianling ON tea4(age); //针对指定字段创建索引

Query OK, 0 rows affected (0.62 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC tea4; //确认创建结果

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

| Field | Type | Null | Key | Default | Extra |

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

| id | char(6) | NO | MUL | NULL | |

| name | varchar(6) | NO | | NULL | |

| age | int(3) | NO | MUL | NULL | |

| gender | enum('boy','girl') | YES | | boy | |

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

4 rows in set (0.00 sec)

4）查看指定表的索引信息

使用SHOW INDEX 指令：

mysql> SHOW INDEX FROM tea4\G

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

Table: tea4

Non\_unique: 1

Key\_name: id

Seq\_in\_index: 1

Column\_name: id

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null:

Index\_type: BTREE //使用B树算法

Comment:

Index\_comment:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: tea4

Non\_unique: 1

Key\_name: nianling //索引名称

Seq\_in\_index: 1

Column\_name: age //字段名称

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null:

Index\_type: BTREE

Comment:

Index\_comment:

2 rows in set (0.00 sec)

5）创建表的时候指定UNIQUE索引字段

UNIQUE表示唯一性的意思，同一个表中可以有多个字段具有唯一性。

比如，创建tea5表，将id、name字段建立设置UNIQUE索引，age字段设置INDEX索引：

mysql> CREATE TABLE tea5(

-> id char(6),

-> name varchar(4) NOT NULL,

-> age int(3) NOT NULL,

-> UNIQUE(id),UNIQUE(name),INDEX(age)

-> );

Query OK, 0 rows affected (0.30 sec)

查看新建tea5表的字段结构，可发现UNIQUE字段的KEY标志为UNI；另外，由于字段name必须满足“NOT NULL”的非空约束，所以将其设置为UNIQUE后会自动变成了PRIMARY KEY主键字段：

mysql> DESC tea5; //确认设置结果

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

| Field | Type | Null | Key | Default | Extra |

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

| id | char(6) | YES | UNI | NULL | |

| name | varchar(4) | NO | PRI | NULL | |

| age | int(3) | NO | MUL | NULL | |

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

3 rows in set (0.03 sec)

6）删除UNIQUE索引、在已有的表中设置UNIQUE索引字段

先删除tea5表name字段的唯一索引（与删除INDEX索引的方法相同）：

mysql> DROP INDEX name ON tea5; //清除UNIQUE索引

Query OK, 0 rows affected (0.97 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC tea5; //确认删除结果

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

| Field | Type | Null | Key | Default | Extra |

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

| id | char(6) | YES | UNI | NULL | |

| name | varchar(4) | NO | | NULL | |

| age | int(3) | NO | MUL | NULL | |

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

3 rows in set (0.00 sec)

重新为tea5表的name字段建立UNIQUE索引，并确认结果：

mysql> CREATE UNIQUE INDEX name ON tea5(name); //建立UNIQUE索引

Query OK, 0 rows affected (0.47 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC tea5; //确认设置结果

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

| Field | Type | Null | Key | Default | Extra |

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

| id | char(6) | YES | UNI | NULL | |

| name | varchar(4) | NO | PRI | NULL | |

| age | int(3) | NO | MUL | NULL | |

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

3 rows in set (0.00 sec)

7）建表时设置PRIMARY KEY主键索引

主键索引实际上在前面已经接触过了，建表的时候可以直接指定。如果表内一开始没有主键字段，则新设置的非空UNIQUE字段相当于具有PRIMARY KEY主键约束。

每个表中的主键字段只能有一个。

建表的时候，可以直接在某个字段的“约束条件”部分指定PRIMARY KEY；也可以在最后指定PRIMARY KEY(某个字段名)。比如：

mysql> CREATE TABLE biao01(

-> id int(4) PRIMARY KEY, //直接在字段定义时约束

-> name varchar(8)

-> );

Query OK, 0 rows affected (0.19 sec)

或者：

mysql> CREATE TABLE biao02(

-> id int(4),

-> name varchar(8),

-> PRIMARY KEY(id) //所有字段定义完，最后指定

-> );

Query OK, 0 rows affected (0.17 sec)

在建表的时候，如果主键字段为int类型，还可以为其设置AUTO\_INCREMENT自增属性，这样当添加新的表记录时，此字段的值会自动从1开始逐个增加，无需手动指定。比如，新建一个tea6表，将id列作为自增的主键字段：

mysql> CREATE TABLE tea6(

-> id int(4) AUTO\_INCREMENT,

-> name varchar(4) NOT NULL,

-> age int(2) NOT NULL,

-> PRIMARY KEY(id)

-> );

Query OK, 0 rows affected (0.29 sec)

8）删除现有表的PRIMARY KEY主键索引

如果要移除某个表的PRIMARY KEY约束，需要通过ALTER TABLE指令修改。比如，以下操作将清除biao01表的主键索引。

清除前（主键为id）：

mysql> DESC biao01;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(4) | NO | PRI | NULL | |

| name | varchar(8) | YES | | NULL | |

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

2 rows in set (0.00 sec)

清除操作：

mysql> ALTER TABLE biao01 DROP PRIMARY KEY;

Query OK, 0 rows affected (0.49 sec)

Records: 0 Duplicates: 0 Warnings: 0

清除后（无主键）：

mysql> DESC biao01;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(4) | NO | | NULL | |

| name | varchar(8) | YES | | NULL | |

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

2 rows in set (0.00 sec)

当尝试删除tea6表的主键时，会出现异常：

mysql> ALTER TABLE tea6 DROP PRIMARY KEY;

ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key

这是因为tea6表的主键字段id具有AUTO\_INCREMNET自增属性，提示这种字段必须作为主键存在，因此若要清除此主键必须先清除自增属性——修改id列的字段定义：

mysql> ALTER TABLE tea6 MODIFY id int(4) NOT NULL;

Query OK, 0 rows affected (0.75 sec)

Records: 0 Duplicates: 0 Warnings: 0

然后再清除主键属性就OK了：

mysql> ALTER TABLE tea6 DROP PRIMARY KEY; //清除主键

Query OK, 0 rows affected (0.39 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc tea6; //确认清除结果

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(4) | NO | | NULL | |

| name | varchar(4) | NO | | NULL | |

| age | int(2) | NO | | NULL | |

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

3 rows in set (0.01 sec)

9）为现有表添加PRIMARY KEY主键索引

重新为tea6表指定主键字段，仍然使用id列：

mysql> ALTER TABLE tea6 ADD PRIMARY KEY(id); //设置主键字段

Query OK, 0 rows affected (0.35 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> DESC tea6; //确认设置结果

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(4) | NO | PRI | NULL | |

| name | varchar(4) | NO | | NULL | |

| age | int(2) | NO | | NULL | |

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

3 rows in set (0.00 sec)

### 步骤二：创建数据库并设置外键实现同步更新与同步删除

根据实验任务要求，两个表格的字段结构如表-1、表-2所示。

1）创建yg表，用来记录员工工号、姓名

其中yg\_id列作为主键，并设置自增属性

mysql> CREATE TABLE yg(

-> yg\_id int(4) AUTO\_INCREMENT,

-> name char(16) NOT NULL,

-> PRIMARY KEY(yg\_id)

-> );

Query OK, 0 rows affected (0.15 sec)

2）创建gz表，用来记录员工的工资信息

其中gz\_id需要参考员工工号，即gz表的gz\_id字段设为外键，将yg表的yg\_id字段作为参考键：

mysql> CREATE TABLE gz(

-> gz\_id int(4) NOT NULL,

-> name char(16) NOT NULL,

-> gz float(7,2) NOT NULL DEFAULT 0,

-> INDEX(name),

-> FOREIGN KEY(gz\_id) REFERENCES yg(yg\_id)

-> ON UPDATE CASCADE ON DELETE CASCADE

-> );

Query OK, 0 rows affected (0.23 sec)

3）为yg表添加2条员工信息记录

因yg\_id有AUTO\_INCREMENT属性，会自动填充，所以只要为name列赋值就可以了。

插入表记录可使用INSERT指令，这里先执行下列操作，具体在下一章学习：

mysql> INSERT INTO yg(name) VALUES('Jerry'),('Tom');

Query OK, 2 rows affected (0.16 sec)

Records: 2 Duplicates: 0 Warnings: 0

确认yg表的数据记录：

mysql> SELECT \* FROM yg;

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

| yg\_id | name |

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

| 1 | Jerry |

| 2 | Tom |

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

2 rows in set (0.00 sec)

4）为gz表添加2条工资信息记录

同上，数据参考图-2，插入相应的工资记录（gz\_id字段未指定默认值，也未设置自增属性，所以需要手动赋值）：

mysql> INSERT INTO gz(gz\_id,name,gz)

-> VALUES(1,'Jerry',12000),(2,'Tom',8000)

-> ;

Query OK, 2 rows affected (0.06 sec)

Records: 2 Duplicates: 0 Warnings: 0

确认gz表的数据记录：

mysql> SELECT \* FROM gz;

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

| gz\_id | name | gz |

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

| 1 | Jerry | 12000.00 |

| 2 | Tom | 8000.00 |

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

2 rows in set (0.05 sec)

5）验证表记录的UPDATE更新联动

将yg表中Jerry用户的yg\_id修改为1234：

mysql> update yg SET yg\_id=1234 WHERE name='Jerry';

Query OK, 1 row affected (0.05 sec)

Rows matched: 1 Changed: 1 Warnings: 0

确认修改结果：

mysql> SELECT \* FROM yg;

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

| yg\_id | name |

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

| 2 | Tom |

| 1234 | Jerry |

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

2 rows in set (0.00 sec)

同时也会发现，gz表中Jerry用户的gz\_id也跟着变了：

mysql> SELECT \* FROM gz;

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

| gz\_id | name | gz |

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

| 1234 | Jerry | 12000.00 |

| 2 | Tom | 8000.00 |

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

2 rows in set (0.00 sec)

6）验证表记录的DELETE删除联动

删除yg表中用户Jerry的记录：

mysql> DELETE FROM yg WHERE name='Jerry';

Query OK, 1 row affected (0.05 sec)

确认删除结果：

mysql> SELECT \* FROM yg;

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

| yg\_id | name |

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

| 2 | Tom |

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

1 row in set (0.00 sec)

查看gz表中的变化（Jerry的记录也没了）：

mysql> SELECT \* FROM gz;

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

| gz\_id | name | gz |

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

| 2 | Tom | 8000.00 |

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

1 row in set (0.00 sec)

7）删除指定表的外键约束

先通过SHOW指令获取表格的外键约束名称：

mysql> SHOW CREATE TABLE gz\G

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

Table: gz

Create Table: CREATE TABLE `gz` (

`gz\_id` int(4) NOT NULL,

`name` char(16) NOT NULL,

`gz` float(7,2) NOT NULL DEFAULT '0.00',

KEY `name` (`name`),

KEY `gz\_id` (`gz\_id`),

CONSTRAINT `gz\_ibfk\_1` FOREIGN KEY (`gz\_id`) REFERENCES `yg` (`yg\_id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8

1 row in set (0.00 sec)

其中gz\_ibfk\_1即删除外键约束时要用到的名称。

删除操作：

mysql> ALTER TABLE gz DROP FOREIGN KEY gz\_ibfk\_1;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

确认删除结果：

mysql> SHOW CREATE TABLE gz\G

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

Table: gz

Create Table: CREATE TABLE `gz` (

`gz\_id` int(4) NOT NULL,

`name` char(16) NOT NULL,

`gz` float(7,2) NOT NULL DEFAULT '0.00',

KEY `name` (`name`),

KEY `gz\_id` (`gz\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

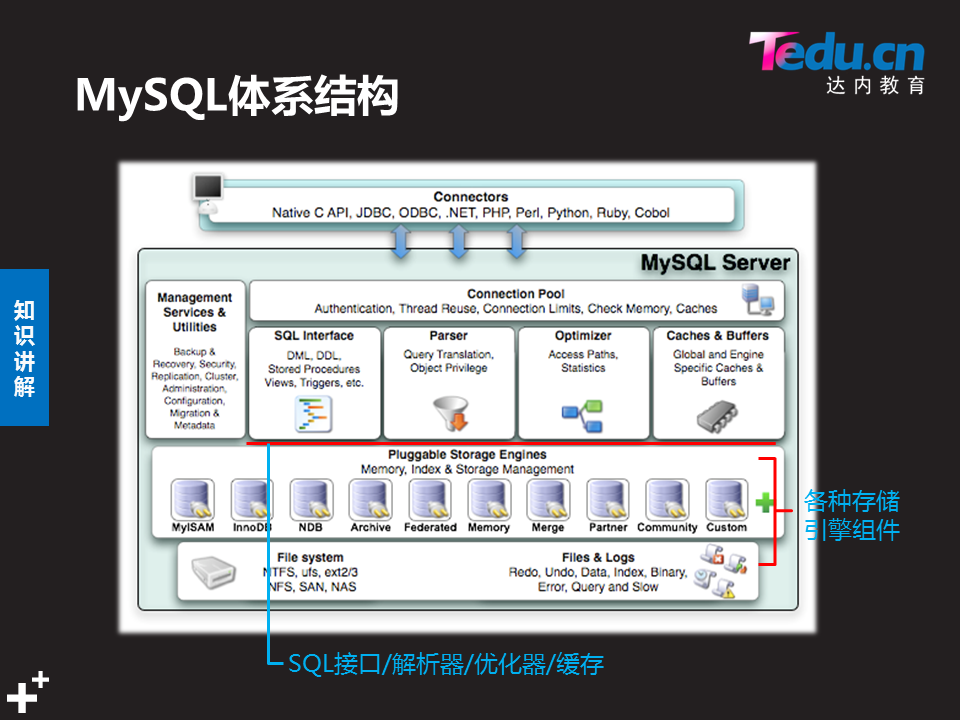
1 row in set (0.00 sec)

# NSD DBA1 DAY03



## 一Mysql存储引擎:

## 1.1 Mysql体系结构(服务功能模块)



Management....(管理工具):提供各种mysql 命令

[root@sql51 ~]# mysql #连续TAB

mysql mysqlimport

mysqladmin mysql\_install\_db

mysqlbinlog mysql\_plugin

mysqlcheck mysqlpump

mysql\_client\_test mysql\_secure\_installation

mysql\_client\_test\_embedded mysqlshow

mysql\_config mysqlslap

mysql\_config-64 mysql\_ssl\_rsa\_setup

mysql\_config\_editor mysqltest

mysqld mysqltest\_embedded

mysqld-debug mysql\_tzinfo\_to\_sql

mysqld\_pre\_systemd mysql\_upgrade

mysqldump mysqlxtest

mysqldumpslow

[root@sql51 ~]# which mysql

/usr/bin/mysql

[root@sql51 ~]# rpm -qf /usr/bin/mysql

mysql-community-client-5.7.17-1.el7.x86\_64

**Management....(管理工具):**提供各种mysql 命令

**Connection pool(连接池):**检查有无空闲线程,检查有无硬件支持线程,检查登录用户名和密码是否正确.三者满足,建立链接

**Sql interface (sql 接口):**接受执行命令

**Parser解析器(分析器):**分析命令格式是否正确.给错误的命令报错

**Optimizer(优化器):**当执行命令没有错误.调用优化器,进行命令优化,以最高效率处理命令

**Caches & buffers(查询缓存):**提供存储空间,存储着曾经查找过的数据,从数据库服务器物理内存划分出来的存储空间给MY.提高查询速度:先检索查询缓存,然后在查询硬盘上的数据

**File system(文件系统):**存储数据,服务器磁盘硬件

**Pluggable storage englnes(存储引擎):**当调用数据库文件系统中的数据,就会调用表使用的存储引擎处理数据.

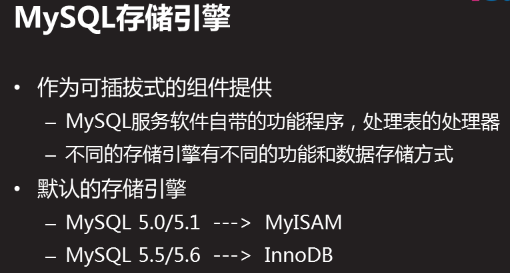
## 1.2存储引擎

Mysql 体系结构中的一个功能程序.当访问这个一个表数据时.会自动调用这个表指定的存储引擎处理数据.

只有InnoDB引擎支持外键

可拔插

默认存储引擎:版本不一样,默认不一样



mysql> system ls /var/lib/mysql/mysql

## 查看数据库服务支持的存储引擎

mysql> show engines;

mysql>show engines\G; #\G 每个内容竖向显示

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

Engine: InnoDB

Support: DEFAULT

Comment: Supports transactions, row-level locking, and foreign keys

Transactions: YES

XA: YES

Savepoints: YES

........

字段说明:

No query specified:

Engine:

Support:是否可用和默认的(default)

Comment:

Transactions:是否支持事物

XA:

Savepoints:

## 查看已有表使用的存储引擎

用查看表的实际创建指令

Show create table 表名;

mysql> show create table t1;

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

| Table | Create Table

+-------+-------------------------------------------------------------------------------------------------| t1 | CREATE TABLE `t1` (

`id` int(4) DEFAULT NULL,

`name` varchar(8) DEFAULT NULL

) ENGINE=MyISAM DEFAULT CHARSET=latin1 |

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

1 row in set (0.00 sec)

t1表使用的存储引擎为MYISAM引擎

## 设置数据库服务默认使用的存储引擎

修改/etc/my.cnf 主配置文件如下:

[mysqld]

default-storage-engine=存储引擎名 #新加

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

default-storage-engine=myisam #设置myisam引擎为默认

[root@sql51 ~]# systemctl restart mysqld

[root@sql51 ~]# mysql -uroot -p123456

mysql> show engines\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: BLACKHOLE

Support: YES

Comment: /dev/null storage engine (anything you write to it disappears)

Transactions: NO

XA: NO

Savepoints: NO

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: MyISAM

Support: DEFAULT

Comment: MyISAM storage engine

Transactions: NO

XA: NO

Savepoints: NO

后面新建的表的默认存储引擎都将是: MyISAM

## 建表时指定使用的存储引擎

mysql> create table t2(id int(4),name varchar(8))engine=innodb;

Query OK, 0 rows affected (0.31 sec)

mysql> show create table t2;

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

| Table | Create Table

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

| t2 | CREATE TABLE `t2` (

`id` int(4) DEFAULT NULL,

`name` varchar(8) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1 |

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

1 row in set (0.00 sec)

## 修改已有表使用的存储引擎

alter table 表名 engine=存储引擎名

mysql> alter table t2 engine=myisam;

Query OK, 0 rows affected (0.22 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> show create table t2;

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

| Table | Create Table

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

| t2 | CREATE TABLE `t2` (

`id` int(4) DEFAULT NULL,

`name` varchar(8) DEFAULT NULL

) ENGINE=MyISAM DEFAULT CHARSET=latin1 |

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

1 row in set (0.00 sec)

## 常用存储引擎的特点

### Mysiam:

支持表级锁以及独享表空间.不支持事务.事务回滚,节省资源

对应有三个表文件

表名.frm:存的是表结构 desc 表名

表名.MYI:存的是索引 index(name)

表名.MYD:存的是表数据 select \* from 表名

### Innodb :

支持行级锁定,支持事务/事务回滚/外键

对应二个表文件

表名.frm:存表结构desc 表名

表名.ibd:存既存索引,又存数据

### 查看数据库默认使用哪个引擎:

mysql> show variables like '%storage\_engine%';

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

| Variable\_name | Value |

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

| default\_storage\_engine | MyISAM |

| default\_tmp\_storage\_engine | InnoDB |

| disabled\_storage\_engines | |

| internal\_tmp\_disk\_storage\_engine | InnoDB |

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

4 rows in set (0.00 sec)

## 锁介绍(锁粒度):

**为什么要加锁:**解决并发访问冲突问题,如多个客户端访问A表,有查询,插入,修改updete等 操作,同时修改同一内容,造成写入冲突,第一个人访问之后,就将锁定内容.

**表级锁:**只要对表做访问,整张表被锁上

**行级锁:**只给访问的每一行记录分别加锁,

**页级锁:**对整个页面进行加锁,页面(MYSQL管理数据的基本存储单位)

**锁类型:** 读锁(共享锁):支持并发读

写锁(排他锁/互斥锁):独占锁,上锁期间,其他线程不能读表或写表

### 查看当前表锁状态信息:

mysql> show status like '%table\_lock%';

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

| Variable\_name | Value |

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

| Performance\_schema\_table\_lock\_stat\_lost | 0 |

| Table\_locks\_immediate | 100 |

| Table\_locks\_waited | 0 |

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

3 rows in set (0.01 sec)

## 事务介绍

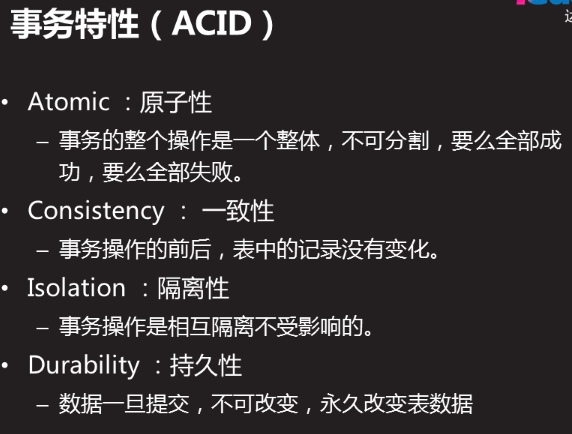
**事务:**对数据的一次访问,就是一次事务,只有innodb存储支持

**事务回滚介绍:**在事务执行过程中,任意一步执行失败,将回复之前所有操作,还原操作前

**如何回滚:**有事务日志文件.启动mysql服务,自动创建.记录对innodb存储引擎的表执行过 的操作,事务日志文件所在:/var/lib/mysql/ 目录下以下文件:

ib\_logfile0 ib\_logfile1 idbdata1

### 事务的特点:简称ACAD



#### 查看自动提交开关状态

开启了自动提交:命令输入插入/修改等命令,一回车,就将数据修改永久生效

Show Variables 显示所有变量 autocommit 自动提交

mysql> show variables like '%autocommit%';

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

| Variable\_name | Value |

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

| autocommit | ON |

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

1 row in set (0.00 sec)

#### 关闭自动提交

mysql> set autocommit=off;

Query OK, 0 rows affected (0.00 sec)

mysql> show variables like '%autocommit%';

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

| Variable\_name | Value |

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

| autocommit | OFF |

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

1 row in set (0.01 sec)

#### 隔离性

终端1关闭了自动提交后,终端2未生效,还是ON

终端1修改的数据必须手动提交(commit),终端2才能看得到;

终端1修改数据没手动提交之前可以进行数据回滚(rollback),恢复操作前的数据;

#### 事务特点ACAD实例操作:

mysql> create table t6(id int)engine=innodb; #创建t6表,指定存储为innodb

Query OK, 0 rows affected (0.20 sec)

mysql> insert into t6 values(999);

mysql> select \* from t6;

+------+

| id |

+------+

| 999 |

+------+

1 row in set (0.00 sec)

ysql> show variables like "%autocommit%"; #查看终端默认自动提交状态

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

| Variable\_name | Value |

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

| autocommit | ON |

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

1 row in set (0.00 sec)

mysql> set autocommit=off; #终端1关闭自动提交

Query OK, 0 rows affected (0.00 sec)

mysql> show variables like "%autocommit%"; #终端1查看确保自动提交关闭

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

| Variable\_name | Value |

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

| autocommit | OFF |

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

1 row in set (0.00 sec)

[root@sql51 ~]# mysql -uroot -p123456 #终端2登录mysql

mysql> use db3 #终端2进入db3库

mysql> delete from t6; #终端1删除t6表

Query OK, 1 row affected (0.00 sec)

mysql> select \* from t6; #终端1查看t6表,已无数据,终端1未提交

Empty set (0.00 sec)

mysql> mysql> select \* from t6; #终端2查看t6表还能看到数据

+------+

| id |

+------+

| 999 |

+------+

1 row in set (0.00 sec)

mysql> rollback; #终端1提交前进行回滚

Query OK, 0 rows affected (0.03 sec)

mysql> select \* from t6; #终端1又能看到之前的数据

+------+

| id |

+------+

| 999 |

+------+

1 row in set (0.00 sec)

mysql> insert into t6 values(666); #终端1修改

Query OK, 1 row affected (0.01 sec)

mysql> select \* from t6; #终端1查看能看到新增数据

+------+

| id |

+------+

| 999 |

| 666 |

+------+

2 rows in set (0.00 sec)

mysql> select \* from t6; #终端2上查看还是看不到,终端1未提交

+------+

| id |

+------+

| 999 |

+------+

1 row in set (0.00 sec)

mysql> commit ; #终端1提交之后

Query OK, 0 rows affected (0.03 sec)

mysql> select \* from t6; #终端2查看就能看到新增的数据了

+------+

| id |

+------+

| 999 |

| 666 |

+------+

2 rows in set (0.00 sec)

## 实际环境中如何决定一个表使用哪种存储引擎

查询表记录:查询(select)操作多的表适合选用myisam存储引擎,节省系统资源

myisam**表级锁:**只要对表做访问,整张表被锁上

读锁(共享锁):支持并发读

插入表记录:写入(insert update delete)操作多的表适合选用innodb存储引擎,并发访问量大

innodb**行级锁**:只给访问的每一行记录分别加锁,

写锁(排他锁/互斥锁):独占锁,上锁期间,其他线程不能读表或写表

到此建表知识点完毕!

## 二 数据导入/导出

## 2.1设置搜索路径

### 查看默认使用目录及目录是否存在

导入导出默认检索数据库文件的路径

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

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

| Variable\_name | Value |

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

| secure\_file\_priv | /var/lib/mysql-files/ |

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

1 row in set (0.00 sec)

[root@sql51 ~]# ls -ld /var/lib/mysql-files/

drwxr-x---. 2 mysql mysql 6 11月 29 2016 /var/lib/mysql-files/

### 修改默认使用目录及目录

[root@sql51 ~]# mkdir /mydatadir

[root@sql51 ~]# chown mysql /mydatadir #设置属组为mysql

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

secure\_file\_priv="/myload" #指定默认检索目录

[root@sql51 ~]# systemctl restart mysqld

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

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

| Variable\_name | Value |

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

| secure\_file\_priv | /mydatadir/ |

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

1 row in set (0.01 sec)

## 2.2 数据导入

导入作用:把系统文件的内容存储到数据库服务器的表里面

命令格式:load date infile “目录名/文件名” Into table 表名

Fields terminated by “分隔符” 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 保护机制

### 数据导入课堂实例:

把/etc/passwd文件的内容存储到db3库下的user表里面.并给每一行记录添加编号字段.

[root@sql51 ~]# mkdir /mydatadir

[root@sql51 ~]# chown mysql /mydatadir #赋予权限

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

secure\_file\_priv="/myload"

[root@sql51 ~]# systemctl restart mysqld

[root@sql51 ~]# mysql -uroot -p123456 db3 #登录mysql并进入读db3库

mysql> create table user(

-> name char(35),

-> password char(1),

-> uid int,

-> gid int,

-> comment char(150),

-> homedir char(100),

-> shell varchar(100),

-> index(name) );

Query OK, 0 rows affected (0.03 sec)

mysql> desc user; #查看表结构

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

| Field | Type | Null | Key | Default | Extra |

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

| name | char(35) | YES | MUL | NULL | |

| password | char(1) | YES | | NULL | |

| uid | int(11) | YES | | NULL | |

| gid | int(11) | YES | | NULL | |

| comment | char(150) | YES | | NULL | |

| homedir | char(100) | YES | | NULL | |

| shell | varchar(100) | YES | | NULL | |

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

7 rows in set (0.01 sec)

mysql> system cp /etc/passwd /mydatadir/ #将passwd文件复制到默认检索目录

mysql> system ls /mydatadir/

passwd

mysql> load data infile "/mydatadir/passwd" into table user fields terminated by ":" lines terminated by "\n"; #导入数据,

Query OK, 41 rows affected (0.01 sec)

Records: 41 Deleted: 0 Skipped: 0 Warnings: 0

mysql> alter table user add id int primary key auto\_increment first; #插入id

Query OK, 41 rows affected (0.07 sec)

Records: 41 Duplicates: 0 Warnings: 0

mysql> select \* from user #查看数据

## 2.3 数据导出

导出作用:把表记录导出到文件

命令格式:sql查询 into outfile “目录名/文件名”

Fields terminated by “分隔符” #可单独指定

lines terminated by “\n” #可单独指定

注意事项:导出的内容由sql查询语句决定

导出的文件路径必须是 secure\_file\_priv(默认搜索路径) 指定的目录里面

导出的是表中的记录,不包括字段名

导出的/目录/文件名 必须唯一,不能有重复

禁用SElinux

### 数据导出课堂实例:

mysql> select \* from user into outfile "/mydatadir/a1.txt";

Query OK, 41 rows affected (0.00 sec)

mysql> system ls /mydatadir

a1.txt passwd

mysql> system cat /mydatadir/a1.txt;

mysql> select \* from user where id <= 5 into outfile "/mydatadir/a2.txt";

Query OK,5 rows affected (0.00 sec)

mysql> system ls /mydatadir

a1.txt a2.txt passwd

mysql> system cat /mydatadir/a2.txt;

mysql> select name,uid,shell from user where id <= 5 into outfile "/mydatadir/a3.txt";

Query OK, 5 rows affected (0.00 sec)

mysql> system ls /mydatadir

a1.txt a2.txt a3.txt passwd

mysql> system cat /mydatadir/a3.txt

root 0 /bin/bash

bin 1 /sbin/nologin

daemon 2 /sbin/nologin

adm 3 /sbin/nologin

lp 4 /sbin/nologin

导出数据,并指定分隔符”#”与换行符”!!!”

mysql> select name,uid,shell from user where id <= 5 into outfile "/mydatadir/a4.txt" fields terminated by "#" lines terminated by "!!!";

Query OK, 5 rows affected (0.00 sec)

mysql> system cat /mydatadir/a4.txt

root#0#/bin/bash!!!bin#1#/sbin/nologin!!!daemon#2#/sbin/nologin!!!adm#3#/sbin/nologin!!!lp#4#/sbin/nologin!!!mysql>

## 三 管理表记录:

## 3.1增加表记录(insert):

### 格式1:增加1条记录

Insert into 表名 values(字段值列表);

### 格式2:增加N条记录

Insert into 表名 values(字段值列表),(字段值列表)(字段值列表);

### 格式3:增加1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:增加N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),(字段值列表)(字段值列表);

### 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,要用双或者单引号括起来
* 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

## 3.2查询表记录(select):

### 格式1:无条件

select 字段1,...,字段N from 表名;

### 格式2:带条件

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

### 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 形式

## 3.3改新记录(update):

### 格式1:更新表内所有记录

Update set 字段1=新值,字段2=新值,字段n=新值;

### 格式2:只更新符合条件的部分记录

Update se 字段1=新值,字段2=新值,字段n=新值; 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,要用双或者单引号括起来
* 若不使用where 限定条件,会更新所有记录
* 限定条件时,只是更新匹配条件的记录

## 3.4删除表记录(delete)

### 格式1:仅删除符合条件的记录

Delete from 表名 where 条件表达式;

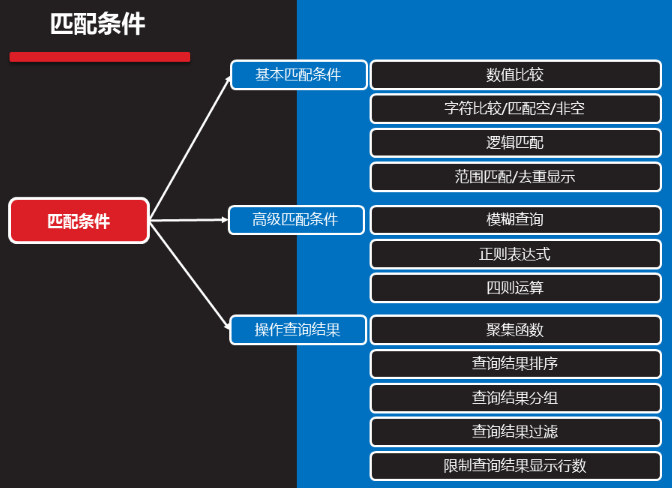
### 格式2:删除所有表记录

Delete from 表名;

### 注意事项:

* 删除指定数据时,最好先按条件查询,检查是否符合删除要求,再执行删除
* 删除所有表记录.表并未被删除,表结构还存在

## 四:基本匹配条件



## 4.1数值比较

大于:> 大于等于:>= 等于:= 小于等于:<= 小于:< 不等于:!=

mysql> select name,uid,gid from user where uid != gid ;

## 4.2字符比较/匹配空/非空

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

mysql> select name,uid,gid from user where name = "mysql";

mysql> select name,uid,gid from user where name != "mysql";

mysql> select name,uid,gid from user where shell is null ;

mysql> select name,uid,gid from user where shell is not null;

## 4.3逻辑匹配:多个判断条件使用

逻辑或:or 多个查询条件,其中某个条件成立即可满足查询条件

mysql> select name,uid shell from user where name="apache" or uid=88 or shell="/sbin/nologin";

逻辑与:and 多个查询条件,需要同时成立才满足查询条件

mysql> select name,uid shell from user where name="apache" and uid=88 and shell="/sbin/nologin";

逻辑非: !或not 取反

mysql> select name from user where name != "mysql";

mysql> select name from user where name is not null;

提高优先级:() 提高判断条件的优先级

## 4.4范围内匹配/去重显示

匹配范围内的任意一个值即可

----用途--------------------类型-------------------------------

在...里面... In (值列表)

mysql> select name from user where name in ("root","adm","rsync","bob");

mysql> select name,uid from user where uid in (1,7,3);

不在...里面... Not in (值列表)

mysql> select name,uid from user where uid not in (1,7,3);

在...之间... Between 数字1 and 数字2

mysql> select \* from user where id between 10 and 20;

去重显示 distinct 字段名

mysql> select distinct shell from user;

## 五:高级匹配查询

## 5.1模糊查询:

基本用法: where 字段名 like ‘通配符’

\_ 匹配单个字符

% 匹配0~N个字符

mysql> select name from user where name like '\_\_\_\_'; #四个\_

mysql> select name from user where name like '%r%'; #含有r

mysql> select name from user where name like '\_\_%\_\_'; #至少有四个

mysql> select name from user where name like 'r%'; #r开头

## 5.2正则表达式:

基本用法:where 字段名 regexp ‘正则表达式’

正则元字符:

^:开头 $:结尾 . :任意一个字符

[]:范围其中一个 \*:0个或多个前面的值 |:或者

mysql> select name from user where name regexp '^r|^a';

mysql> select name from user where name regexp '^r.\*t$';

mysql> insert into user(name)values("jim9"),("j7im"),("ji2m");

mysql> select name from user where name regexp '[0-9]';

## 5.3四则运算:

加:+ 减:- 乘:\* 除:/ 取余:%

mysql> update user set uid=uid+1 where uid <=10;

mysql> select name ,uid,gid,uid+gid ruslet from user where name="root";

() 提供条件的优先级

mysql> select name ,uid,gid, (uid+gid)/2 pjz from user where name="mysql";

## 5.4操作查询结果

### 聚集函数

聚集函数:MySQL 内置数据统计函数

sum(字段名) 统计字段之和

avg(字段名) 统计字段平均值

min(字段名) 统计字段最小值

max(字段名) 统计字段最大值

count(字段名) 统计字段值个数

mysql> select count(name) from user;

mysql> select count(name) from user where shell="/bin/bash";

mysql> select avg(uid) from user;

mysql> select sum(uid) from user;

mysql> select min(uid) from user;

mysql> select max(uid) from user;

mysql> select min(uid) from user where shell != "/bin/bash";

mysql> select min(uid) from user;

mysql> select count(\*) from user; #统计所有行数

### 查询结果排序order by 字段名

默认升序:sql 查询 order by 字段名

降 序:sql 查询 order by 字段名 desc

sql 查询 order by 字段名 [asc|desc]

在user表中查看uid在10-1000之间的name,并且按uid升序排序

mysql> select name,uid from user where uid between 10 and 1000 order by uid;

mysql> select name,uid from user where uid between 10 and 1000 order by uid desc;

### 分组 group by 字段名

Sql 查询 group by 字段名 通常是对字符类型字段分组

mysql> select shell from user group by shell;

mysql> select shell from user where uid between 10 and 1000 group by shell;

### 过滤 having 条件表达式

Sql 查询 having 条件表达式;

Sql 查询 where 条件 having 条件表达式;

Sql 查询 group by 字段名 having 条件表达式;

查询user表中uid在10和6000之间的,名字是”root”的数据

mysql> select name from user where uid between 10 and 6000 having name="root";

mysql> select name from user where uid between 10 and 6000 having name="mysql";

### 限制显示行数 limit

Sql 查询 limit;

Sql 查询 limit n; #显示查询结果的前N行记录

Sql 查询 limit n,m #显示指定范围内的查询记录

Sql 查询 where 条件查询 limit 3; #显示查询结果前3条记录

Sql 查询 where 条件查询 limit 3,3; #从第4行开始,共显示3条记录

mysql> select name from user where uid between 10 and 6000 limit 2;

mysql> select name from user where uid between 10 and 6000 limit 4,5;

综合应用:

在user表中查找shell不是/bin/bash的所有数据,且按照uid的值降序排序,显示前3行

mysql> select \* from user where shell!="/bin/bash" order by uid desc limit 3;

在user表中查找shell不是/bin/bash的所有数据,且按照uid的值升序排序,显示前3行

mysql> select \* from user where shell!="/bin/bash" order by uid limit 3;

## 案例

1. MySQL存储引擎的配置
2. 数据导入/导出
3. 操作表记录
4. 查询及匹配条件

## 1 MySQL存储引擎的配置

1.1 问题

本案例要求MySQL数据存储引擎的使用，完成以下任务操作：

查看服务支持的存储引擎

查看默认存储类型

更改表的存储引擎

设置数据库服务默认使用的存储引擎

1.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：查看存储引擎信息

登入MySQL服务器，查看当前支持哪些存储引擎。

使用mysql命令连接，以root用户登入：

[root@dbsvr1 ~]# mysql -u root –p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 9

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

执行SHOW ENGINES\G指令可列表查看，MySQL 5.6可用的存储引擎有9种（除最后的FEDERATED以外，其他8种都支持），其中默认采用的存储引擎为InnoDB：

mysql> SHOW ENGINES\G

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

Engine: InnoDB

Support: DEFAULT //此存储引擎为默认

Comment: Supports transactions, row-level locking, and foreign keys

Transactions: YES

XA: YES

Savepoints: YES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: MRG\_MYISAM

Support: YES

Comment: Collection of identical MyISAM tables

Transactions: NO

XA: NO

Savepoints: NO

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: MEMORY

Support: YES

Comment: Hash based, stored in memory, useful for temporary tables

Transactions: NO

XA: NO

Savepoints: NO

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: BLACKHOLE

Support: YES

Comment: /dev/null storage engine (anything you write to it disappears)

Transactions: NO

XA: NO

Savepoints: NO

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: MyISAM

Support: YES

Comment: MyISAM storage engine

Transactions: NO

XA: NO

Savepoints: NO

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 6. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: CSV

Support: YES

Comment: CSV storage engine

Transactions: NO

XA: NO

Savepoints: NO

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 7. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: ARCHIVE

Support: YES

Comment: Archive storage engine

Transactions: NO

XA: NO

Savepoints: NO

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 8. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: PERFORMANCE\_SCHEMA

Support: YES

Comment: Performance Schema

Transactions: NO

XA: NO

Savepoints: NO

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 9. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Engine: FEDERATED

Support: NO //此引擎不被支持

Comment: Federated MySQL storage engine

Transactions: NULL

XA: NULL

Savepoints: NULL

9 rows in set (0.01 sec)

### 步骤二：查看默认存储类型

查看系统变量default\_storage\_engine 的值，确认默认采用的存储引擎是InnoDB：

mysql> SHOW VARIABLES LIKE 'default\_storage\_engine';

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

| Variable\_name | Value |

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

| default\_storage\_engine | InnoDB |

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

1 row in set (0.00 sec)

### 步骤三：修改默认存储引擎

在 mysql> 环境中，可以直接通过SET指令更改默认的存储引擎（只在本次连接会话过程中有效，退出重进即失效） 。比如临时修改为MyISAM，可执行下列操作：

mysql> SET default\_storage\_engine=MyISAM; //改用MyISAM引擎

Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'default\_storage\_engine'; //确认结果

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

| Variable\_name | Value |

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

| default\_storage\_engine | MyISAM |

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

1 row in set (0.00 sec)

若希望直接修改MySQL服务程序所采用的默认存储引擎，应将相关设置写入配置文件/etc/my.cnf，并重启服务后生效。比如：

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

[mysqld]

.. ..

default\_storage\_engine=MEMORY //改用MEMORY引擎

[root@dbsvr1 ~]# systemctl restart mysqld.service //重启服务

重新登入 mysql> 确认修改结果：

[root@dbsvr1 ~]# mysql -u root -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 3

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW VARIABLES LIKE 'default\_storage\_engine';

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

| Variable\_name | Value |

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

| default\_storage\_engine | MEMORY | //默认引擎已修改

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

1 row in set (0.00 sec)

mysql> exit

Bye

### 步骤四：设置数据库服务默认使用的存储引擎

为了避免后续实验障碍，测试完后记得恢复原状——移除默认引擎设置，或者将其修改为InnoDB即可：

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

[mysqld]

.. ..

default\_storage\_engine=InnoDB

[root@dbsvr1 ~]# systemctl restart mysqld.service

确认恢复结果（选项 -e 可调用指定的SQL操作后返回Shell命令行）：

[root@dbsvr1 ~]# mysql -u root -p -e "SHOW VARIABLES LIKE 'default\_storage\_engine';"

Enter password:

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

| Variable\_name | Value |

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

| default\_storage\_engine | InnoDB |

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

## 2 **数据导入/导出**

2.1 问题

使用SQL语句完成下列导出、导入操作：

将/etc/passwd文件导入userdb库user表并给每条记录加编号

将userdb库user表中UID小于100的前10条记录导出，存为/myload/user2.txt文件

2.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：将/etc/passwd文件导入MySQL数据库

导入后的表结构取决于/etc/passwd配置文件。若一时记不住各字段的含义，也可以查看passwd配置文件的man手册页，找到格式描述相关的说明，比如：

[root@dbsvr1 ~]# man 5 passwd

.. ..

Each line of the file describes a single user, and contains seven colon-sep‐

arated fields:

name:password:UID:GID:GECOS:directory:shell //各字段的顺序、大致用途

The field are as follows: //以下详细解释各字段的作用

name This is the user's login name. It should not contain capital

letters.

password This is either the encrypted user password, an asterisk (\*), or

the letter 'x'. (See pwconv(8) for an explanation of 'x'.)

UID The privileged root login account (superuser) has the user ID 0.

GID This is the numeric primary group ID for this user. (Additional

groups for the user are defined in the system group file; see

group(5)).

GECOS stands for "General Electric Comprehensive Operating Sys‐

tem", which was renamed to GCOS when GE's large systems division

was sold to Honeywell. Dennis Ritchie has reported: "Sometimes

we sent printer output or batch jobs to the GCOS machine. The

gcos field in the password file was a place to stash the infor‐

mation for the $IDENTcard. Not elegant."

directory This is the user's home directory: the initial directory where

the user is placed after logging in. The value in this field is

used to set the HOME environment variable.

shell This is the program to run at login (if empty, use /bin/sh). If

set to a nonexistent executable, the user will be unable to

login through login(1). The value in this field is used to set

the SHELL environment variable.

.. ..

1）新建userdb库、user表

以数据库用户root登入MySQL服务：

[root@dbsvr1 ~]# mysql -u root -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 5

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

新建userdb库，切换到userdb库：

mysql> CREATE DATABASE userdb;

Query OK, 1 row affected (0.00 sec)

mysql> USE userdb;

Database changed

新建user表，字段设置及相关操作参考如下：

mysql> CREATE TABLE user(

-> username varchar(24) NOT NULL,

-> password varchar(48) DEFAULT 'x',

-> uid int(5) NOT NULL,

-> gid int(5) NOT NULL,

-> fullname varchar(48),

-> homedir varchar(64) NOT NULL,

-> shell varchar(24) NOT NULL

-> );

Query OK, 0 rows affected (0.70 sec)

确认user表的结构：

mysql> DESC user;

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

| Field | Type | Null | Key | Default | Extra |

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

| username | varchar(24) | NO | | NULL | |

| password | varchar(48) | YES | | x | |

| uid | int(5) | NO | | NULL | |

| gid | int(5) | NO | | NULL | |

| fullname | varchar(48) | YES | | NULL | |

| homedir | varchar(64) | NO | | NULL | |

| shell | varchar(24) | NO | | NULL | |

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

7 rows in set (0.01 sec)

2）如果直接导入会报错。在MySQL 5.7.6版本之后，导入文件只能在secure\_file\_priv指定的文件夹下。执行show variables like '%secure%'命令显示文件目录：

mysql> LOAD DATA INFILE '/etc/passwd' INTO TABLE userlist FIELDS TERMINATED BY ':';

ERROR 1290 (HY000): The MySQL server is running with the --secure-file-priv option so it cannot execute this statement

mysql> show variables like '%secure%';

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

| Variable\_name | Value |

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

| require\_secure\_transport | OFF |

| secure\_auth | ON |

| secure\_file\_priv | /var/lib/mysql-files/ |

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

3 rows in set (0.00 sec)

3）执行导入操作

将/etc/passwd文件复制到/var/lib/mysql-files/目录下，

读取/var/lib/mysql-files/passwd文件内容，以“:”为分隔，导入到user表中：

[root@dbsvr1 ~]#cp /etc/passwd /var/lib/mysql-files/

mysql> LOAD DATA INFILE '/var/lib/mysql-files/passwd'

-> INTO TABLE userlist

-> FIELDS TERMINATED BY ':';

Query OK, 39 rows affected (0.11 sec)

Records: 39 Deleted: 0 Skipped: 0 Warnings: 0

上述操作中省略了行分隔 LINES TERMINATED BY '\n'，因为这是默认的情况（每行一条原始记录），除非需要以其他字符分割行，才需要用到这个。比如，以下操作指定了行分隔为'\n'，将/var/lib/mysql-files/passwd文件的内容导入另一个表user2，最终user2表的内容与user的内容是一样的：

代码

4）确认导入结果

分别统计user、user2表内的记录个数：

mysql> SELECT COUNT(\*) FROM user;

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

| COUNT(\*) |

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

| 39 | //user表有39条记录

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

1 row in set (0.00 sec)

mysql> SELECT COUNT(\*) FROM user2;

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

| COUNT(\*) |

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

| 39 | //user2表也有39条记录

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

1 row in set (0.00 sec)

查看user表的前10条记录，列出用户名、UID、GID、宿主目录、登录Shell：

mysql> SELECT username,uid,gid,homedir,shell

-> FROM user LIMIT 10;

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

| username | uid | gid | homedir | shell |

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

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

| bin | 1 | 1 | /bin | /sbin/nologin |

| daemon | 2 | 2 | /sbin | /sbin/nologin |

| adm | 3 | 4 | /var/adm | /sbin/nologin |

| lp | 4 | 7 | /var/spool/lpd | /sbin/nologin |

| sync | 5 | 0 | /sbin | /bin/sync |

| shutdown | 6 | 0 | /sbin | /sbin/shutdown |

| halt | 7 | 0 | /sbin | /sbin/halt |

| mail | 8 | 12 | /var/spool/mail | /sbin/nologin |

| operator | 11 | 0 | /root | /sbin/nologin |

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

10 rows in set (0.00 sec)

查看user2表的前10条记录，同样列出用户名、UID、GID、宿主目录、登录Shell：

mysql> SELECT username,uid,gid,homedir,shell

-> FROM user2 LIMIT 10;

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

| username | uid | gid | homedir | shell |

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

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

| bin | 1 | 1 | /bin | /sbin/nologin |

| daemon | 2 | 2 | /sbin | /sbin/nologin |

| adm | 3 | 4 | /var/adm | /sbin/nologin |

| lp | 4 | 7 | /var/spool/lpd | /sbin/nologin |

| sync | 5 | 0 | /sbin | /bin/sync |

| shutdown | 6 | 0 | /sbin | /sbin/shutdown |

| halt | 7 | 0 | /sbin | /sbin/halt |

| mail | 8 | 12 | /var/spool/mail | /sbin/nologin |

| operator | 11 | 0 | /root | /sbin/nologin |

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

10 rows in set (0.00 sec)

### **步骤二**：为user表中的每条记录添加自动编号

这个只要修改user表结构，添加一个自增字段即可。

比如，添加一个名为sn的序号列，作为user表的第一个字段：

1）添加自增主键字段sn

mysql> ALTER TABLE user

-> ADD sn int(4) AUTO\_INCREMENT PRIMARY KEY FIRST;

Query OK, 0 rows affected (0.62 sec)

Records: 0 Duplicates: 0 Warnings: 0

2）验证自动编号结果

查看user表的前10条记录，列出序号、用户名、UID、GID、宿主目录：

mysql> SELECT sn,username,uid,gid,homedir

-> FROM user LIMIT 10;

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

| sn | username | uid | gid | homedir |

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

| 1 | root | 0 | 0 | /root |

| 2 | bin | 1 | 1 | /bin |

| 3 | daemon | 2 | 2 | /sbin |

| 4 | adm | 3 | 4 | /var/adm |

| 5 | lp | 4 | 7 | /var/spool/lpd |

| 6 | sync | 5 | 0 | /sbin |

| 7 | shutdown | 6 | 0 | /sbin |

| 8 | halt | 7 | 0 | /sbin |

| 9 | mail | 8 | 12 | /var/spool/mail |

| 10 | operator | 11 | 0 | /root |

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

10 rows in set (0.00 sec)

### 步骤三：从MySQL数据库中导出查询结果

以将userdb库user表中UID小于100的前10条记录导出为/myload/user2.txt文件为例。

1）确认存放导出数据的文件夹

[root@dbsvr1 ~]# ls -ld /var/lib/mysql-files/

drwxr-x---. 2 mysql mysql 19 4月 7 11:15 /var/lib/mysql-files/

2）修改目录及查看修改结果

[root@dbsvr1 ~]# mkdir /myload ; chown mysql /myload

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

[mysqld]

secure\_file\_priv="/myload"

[root@dbsvr1 ~]# systemctl restart mysqld

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

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

| Variable\_name | Value |

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

| secure\_file\_priv | /myload/ |

2）导出user表中UID小于100的前10条记录

如果以默认的'\n' 为行分隔，导出操作同样可不指定LINES TERMINATED BY：

mysql> SELECT \* FROM userdb.user WHERE uid<100

-> INTO OUTFILE '/myload/user2.txt'

-> FIELDS TERMINATED BY ':';

Query OK, 24 rows affected (0.00 sec)

3）确认导出结果

返回到Shell命令行，查看/myload/user2.txt文件的行数：

[root@dbsvr1 ~]# wc -l /myload/user2.txt

24 /myload/user2.txt

查看/myload/user2.txt文件的最后10行内容：

[root@dbsvr1 ~]# tail /myload/user2.txt

19:avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin

24:rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin

25:rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin

28:radvd:x:75:75:radvd user:/:/sbin/nologin

29:ntp:x:38:38::/etc/ntp:/sbin/nologin

33:gdm:x:42:42::/var/lib/gdm:/sbin/nologin

35:postfix:x:89:89::/var/spool/postfix:/sbin/nologin

36:sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin

37:tcpdump:x:72:72::/:/sbin/nologin

39:mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/false

## 3 **操作表记录**

3.1 问题

练习表记录的操作

表记录的插入

表记录的更新

表记录的查询

表记录的删除

3.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：创建stu\_info表，并确保stu\_info表记录为空。

在userdb库中创建stu\_info表：

[root@dbsvr1 ~]# mysql -uroot -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 19

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use userdb;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> CREATE TABLE stu\_info(

-> name varchar(12) NOT NULL,

-> gender enum('boy','girl') DEFAULT 'boy',

-> age int(3) NOT NULL

-> );

Query OK, 0 rows affected (0.23 sec)

删除stu\_info表的所有记录：

mysql> DELETE FROM stu\_info;

Query OK, 0 rows affected (0.00 sec) //stu\_info表刚建立 删除零条记录

确认删除结果：

mysql> SELECT \* FROM stu\_info;

Empty set (0.00 sec)

### 步骤二：练习表记录的操作

1）插入记录时，指定记录的每一个字段的值

这种情况下，不需要明确指出字段，但每条记录的值的顺序、类型都必须与表格结构向一致，否则可能无法正确插入记录。

比如，以下操作将向stu\_info表插入3条表记录：

mysql> INSERT stu\_info VALUES

-> ('Jim','girl',24),

-> ('Tom','boy',21),

-> ('Lily','girl',20);

Query OK, 3 rows affected (0.15 sec)

Records: 3 Duplicates: 0 Warnings: 0

完成插入后确认表记录：

mysql> SELECT \* FROM stu\_info;

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

| name | gender | age |

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

| Jim | girl | 24 |

| Tom | boy | 21 |

| Lily | girl | 20 |

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

3 rows in set (0.00 sec)

2）插入记录时，只指定记录的部分字段的值

这种情况下，必须指出各项值所对应的字段；而且，未赋值的字段应设置有默认值或者有自增填充属性或者允许为空，否则插入操作将会失败。

比如，向stu\_info表插入Jerry的年龄信息，性别为默认的“boy”，自动编号，相关操作如下：

mysql> INSERT INTO stu\_info(name,age)

-> VALUES('Jerry',27);

Query OK, 1 row affected (0.04 sec)

类似的，再插入用户Mike的年龄信息：

mysql> INSERT INTO stu\_info(name,age)

-> VALUES('Mike',21);

Query OK, 1 row affected (0.05 sec)

确认目前stu\_info表的所有记录：

mysql> SELECT \* FROM stu\_info;

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

| name | gender | age |

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

| Jim | girl | 24 |

| Tom | boy | 21 |

| Lily | girl | 20 |

| Jerry | boy | 27 |

| Mike | boy | 21 |

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

5 rows in set (0.00 sec)

3）更新表记录时，若未限制条件，则适用于所有记录

将stu\_info表中所有记录的age设置为10：

mysql> UPDATE stu\_info SET age=10;

Query OK, 5 rows affected (0.04 sec)

Rows matched: 5 Changed: 5 Warnings: 0

确认更新结果：

mysql> SELECT \* FROM stu\_info;

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

| name | gender | age |

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

| Jim | girl | 10 |

| Tom | boy | 10 |

| Lily | girl | 10 |

| Jerry | boy | 10 |

| Mike | boy | 10 |

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

5 rows in set (0.00 sec)

4）更新表记录时，可以限制条件，只对符合条件的记录有效

将stu\_info表中所有性别为“boy”的记录的age设置为20：

mysql> UPDATE stu\_info SET age=20

-> WHERE gender='boy';

Query OK, 3 rows affected (0.04 sec)

Rows matched: 3 Changed: 3 Warnings: 0

确认更新结果：

mysql> SELECT \* FROM stu\_info;

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

| name | gender | age |

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

| Jim | girl | 10 |

| Tom | boy | 20 |

| Lily | girl | 10 |

| Jerry | boy | 20 |

| Mike | boy | 20 |

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

5 rows in set (0.00 sec)

5）删除表记录时，可以限制条件，只删除符合条件的记录

删除stu\_info表中年龄小于18的记录：

mysql> DELETE FROM stu\_info WHERE age < 18;

Query OK, 2 rows affected (0.03 sec)

确认删除结果：

mysql> SELECT \* FROM stu\_info;

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

| name | gender | age |

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

| Tom | boy | 20 |

| Jerry | boy | 20 |

| Mike | boy | 20 |

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

3 rows in set (0.00 sec)

6）删除表记录时，如果未限制条件，则会删除所有的表记录

删除stu\_info表的所有记录：

mysql> DELETE FROM stu\_info;

Query OK, 3 rows affected (0.00 sec)

确认删除结果：

mysql> SELECT \* FROM stu\_info;

Empty set (0.00 sec)

## 4 **查询及匹配条件**

4.1 问题

练习常见的SQL查询及条件设置

创建stu\_info表，并插入数据

练习常见SQL查询及条件设置

4.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：根据任务要求建立员工档案表stu\_info（如上个实验已创建，可将上个实验stu\_info表中记录清除后继续使用）

1）在userdb库中创建stu\_info表

以root用户登入MySQL服务器：

[root@dbsvr1 ~]# mysql -u root -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 5

Server version: 5.6.15 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

打开test库：

mysql> USE userdb;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

创建stu\_info表，包括name、gender、age三个字段：

mysql> CREATE TABLE stu\_info(

-> name varchar(12) NOT NULL,

-> gender enum('boy','girl') DEFAULT 'boy',

-> age int(3) NOT NULL

-> );

Query OK, 0 rows affected (0.03 sec)

确认表结构：

mysql> DESC stu\_info;

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

| Field | Type | Null | Key | Default | Extra |

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

| name | varchar(12) | NO | | NULL | |

| gender | enum('boy','girl') | YES | | boy | |

| age | int(3) | NO | | NULL | |

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

3 rows in set (0.01 sec)

2）准备测试表格

向建立的stu\_info表插入几条测试记录

mysql> INSERT INTO stu\_info VALUES

-> ('Jim','girl',24),

-> ('Tom','boy',21),

-> ('Lily','girl',20),

-> ('Jerry','boy',27),

-> ('Mike','boy',21)

-> ;

Query OK, 5 rows affected (0.06 sec)

Records: 5 Duplicates: 0 Warnings: 0

确认stu\_info表的所有记录内容：

mysql> SELECT \* FROM stu\_info;

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

| name | gender | age |

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

| Jim | girl | 24 |

| Tom | boy | 21 |

| Lily | girl | 20 |

| Jerry | boy | 27 |

| Mike | boy | 21 |

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

5 rows in set (0.00 sec)

### 步骤二：练习常见SQL查询及条件设置

1）常用的表记录统计函数

查询stu\_info表一共有多少条记录（本例中为5条）：

mysql> SELECT count(\*) FROM stu\_info;

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

| count(\*) |

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

| 5 |

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

1 row in set (0.00 sec)

计算stu\_info表中各学员的平均年龄、最大年龄、最小年龄：

mysql> SELECT avg(age),max(age),min(age) FROM stu\_info;

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

| avg(age) | max(age) | min(age) |

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

| 22.6000 | 27 | 20 |

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

1 row in set (0.00 sec)

计算stu\_info表中男学员的个数：

mysql> SELECT count(gender) FROM stu\_info WHERE gender='boy';

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

| count(gender) |

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

| 3 |

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

1 row in set (0.00 sec)

2）字段值的数值比较

列出stu\_info表中年龄为21岁的学员记录：

mysql> SELECT \* FROM stu\_info WHERE age=21;

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

| name | gender | age |

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

| Tom | boy | 21 |

| Mike | boy | 21 |

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

2 rows in set (0.00 sec)

列出stu\_info表中年龄超过21岁的学员记录：

mysql> SELECT \* FROM stu\_info WHERE age>21;

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

| name | gender | age |

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

| Jim | girl | 24 |

| Jerry | boy | 27 |

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

2 rows in set (0.00 sec)

列出stu\_info表中年龄大于或等于21岁的学员记录：

mysql> SELECT \* FROM stu\_info WHERE age>=21;

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

| name | gender | age |

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

| Jim | girl | 24 |

| Tom | boy | 21 |

| Jerry | boy | 27 |

| Mike | boy | 21 |

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

4 rows in set (0.00 sec)

列出stu\_info表中年龄在20岁和24岁之间的学员记录：

mysql> SELECT \* FROM stu\_info WHERE age BETWEEN 20 and 24;

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

| name | gender | age |

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

| Jim | girl | 24 |

| Tom | boy | 21 |

| Lily | girl | 20 |

| Mike | boy | 21 |

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

4 rows in set (0.00 sec)

3）多个条件的组合

列出stu\_info表中年龄小于23岁的女学员记录：

mysql> SELECT \* FROM stu\_info WHERE age < 23 AND gender='girl';

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

| name | gender | age |

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

| Lily | girl | 20 |

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

1 row in set (0.00 sec)

列出stu\_info表中年龄小于23岁的学员，或者女学员的记录：

mysql> SELECT \* FROM stu\_info WHERE age < 23 OR gender='girl';

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

| name | gender | age |

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

| Jim | girl | 24 |

| Tom | boy | 21 |

| Lily | girl | 20 |

| Mike | boy | 21 |

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

4 rows in set (0.00 sec)

如果某个记录的姓名属于指定范围内的一个，则将其列出：

mysql> SELECT \* FROM stu\_info WHERE name IN

-> ('Jim','Tom','Mickey','Minnie');

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

| name | gender | age |

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

| Jim | girl | 24 |

| Tom | boy | 21 |

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

2 rows in set (0.00 sec)

4）使用SELECT做数学计算

计算1234与5678的和：

mysql> SELECT 1234+5678;

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

| 1234+5678 |

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

| 6912 |

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

1 row in set (0.00 sec)

计算1234与5678的乘积：

mysql> SELECT 1234\*5678;

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

| 1234\*5678 |

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

| 7006652 |

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

1 row in set (0.00 sec)

计算1.23456789除以3的结果：

mysql> SELECT 1.23456789/3;

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

| 1.23456789/3 |

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

| 0.411522630000 |

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

1 row in set (0.00 sec)

输出stu\_info表各学员的姓名、15年后的年龄：

mysql> SELECT name,age+15 FROM stu\_info;

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

| name | age+15 |

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

| Jim | 39 |

| Tom | 36 |

| Lily | 35 |

| Jerry | 42 |

| Mike | 36 |

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

5 rows in set (0.00 sec)

5）使用模糊查询，LIKE引领

以下划线 \_ 匹配单个字符，% 可匹配任意多个字符。

列出stu\_info表中姓名以“J”开头的学员记录：

mysql> SELECT \* FROM stu\_info WHERE name LIKE 'J%';

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

| name | gender | age |

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

| Jim | girl | 24 |

| Jerry | boy | 27 |

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

2 rows in set (0.00 sec)

列出stu\_info表中姓名以“J”开头且只有3个字母的学员记录：

mysql> SELECT \* FROM stu\_info WHERE name LIKE 'J\_\_';

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

| name | gender | age |

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

| Jim | girl | 24 |

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

1 row in set (0.00 sec)

6）使用正则表达式，REGEXP引领

列出stu\_info表中姓名以“J”开头且以“y”结尾的学员记录：

mysql> SELECT \* FROM stu\_info WHERE name REGEXP '^J.\*y$';

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

| name | gender | age |

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

| Jerry | boy | 27 |

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

1 row in set (0.00 sec)

效果等同于：

mysql> SELECT \* FROM stu\_info WHERE name Like 'J%y';

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

| name | gender | age |

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

| Jerry | boy | 27 |

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

1 row in set (0.00 sec)

列出stu\_info表中姓名以“J”开头或者以“y”结尾的学员记录：

mysql> SELECT \* FROM stu\_info WHERE name REGEXP '^J|y$';

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

| name | gender | age |

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

| Jim | girl | 24 |

| Lily | girl | 20 |

| Jerry | boy | 27 |

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

3 rows in set (0.00 sec)

效果等同于：

mysql> SELECT \* FROM stu\_info WHERE name Like 'J%' OR name Like '%y';

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

| name | gender | age |

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

| Jim | girl | 24 |

| Lily | girl | 20 |

| Jerry | boy | 27 |

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

3 rows in set (0.00 sec)

7）按指定的字段排序，ORDER BY

列出stu\_info表的所有记录，按年龄排序：

mysql> SELECT \* FROM stu\_info GROUP BY age;

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

| name | gender | age |

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

| Lily | girl | 20 |

| Tom | boy | 21 |

| Jim | girl | 24 |

| Jerry | boy | 27 |

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

4 rows in set (0.00 sec)

因默认为升序（Ascend）排列，所以上述操作等效于：

mysql> SELECT \* FROM stu\_info GROUP BY age ASC;

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

| name | gender | age |

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

| Lily | girl | 20 |

| Tom | boy | 21 |

| Jim | girl | 24 |

| Jerry | boy | 27 |

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

4 rows in set (0.00 sec)

若要按降序（Descend）排列，则将ASC改为DESC即可：

mysql> SELECT \* FROM stu\_info GROUP BY age DESC;

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

| name | gender | age |

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

| Jerry | boy | 27 |

| Jim | girl | 24 |

| Tom | boy | 21 |

| Lily | girl | 20 |

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

4 rows in set (0.00 sec)

8）限制查询结果的输出条数，LIMIT

查询stu\_info表的所有记录，只列出前3条：

mysql> SELECT \* FROM stu\_info LIMIT 3;

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

| name | gender | age |

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

| Jim | girl | 24 |

| Tom | boy | 21 |

| Lily | girl | 20 |

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

3 rows in set (0.00 sec)

列出stu\_info表中年龄最大的3条学员记录：

mysql> SELECT \* FROM stu\_info GROUP BY age DESC LIMIT 3;

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

| name | gender | age |

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

| Jerry | boy | 27 |

| Jim | girl | 24 |

| Tom | boy | 21 |

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

3 rows in set (0.00 sec)

9）分组查询结果，GROUP BY

针对stu\_info表，按性别分组，分别统计出男、女学员的人数：

mysql> SELECT gender,count(gender) FROM stu\_info GROUP BY gender;

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

| gender | count(gender) |

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

| boy | 3 |

| girl | 2 |

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

2 rows in set (0.00 sec)

列出查询字段时，可以通过AS关键字来指定显示别名，比如上述操作可改为：

mysql> SELECT gender AS '性别',count(gender) AS '人数'

-> FROM stu\_info GROUP BY gender;

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

| 性别 | 人数 |

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

| boy | 3 |

| girl | 2 |

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

2 rows in set (0.00 sec)

# NSD DBA1 DAY04



## 一 复制表

复制表作用:备份表与快速建表

## 1.1:将源表xxx复制为新表yyy

命令格式:crate tabe yyy select \* from xxx

例如:将db3库的user表,复制到db4库userbak表

mysql> create table userbak select \* from db3.user;

## 1.2:将指定查询结果复制为新表zzz

例如:

mysql> create table user3 select name,uid,shell from db3.user order by uid desc limit 5;

mysql> select \* from user3;

## 1.3:复制源表xxx的结构到新表vvv

例如:将db3库的user表结构,复制到db4库的user1表,主键/索引等无法复制过来

mysql> create table user1 select \* from db3.user where 1=2;

mysql> select \* from user1;

Empty set (0.00 sec)

例如:复制部分字段到新表

mysql> create table user4 select name,uid,shell from db3.user where 1=2;

## 1.4:将源表vvv 的名称改为www

Alter table vvv rename to www;

## 二 多表查询

## 2.1 多表查询

* 将2个或2个以上的表,按某个条件连接起来,从中选取需要数据
* 当多个表中 存在相同意义的字段时(字段名可以不同),可以通过该字段连接多个表
* 耗费资源多,尽量减少多表查询

环境准备:在db4库中创建t1 t2表,分别从db3库中user表中复制数据.

mysql> create table t1 select name,uid,shell,password from db3.user limit 3;

mysql> create table t2 select name,uid,shell,password from db3.user limit 4;

**格式1:**select 字段名列表 from 表a 表b ; (查询出来的结果为笛卡尔集)

### 笛卡尔集:

以上格式1: 查询的结果叫笛卡尔集,

查询结果的记录总条目数=(表A的记录数 \* 表b的记录数)

mysql> select \* from t1,t2; #查询出的是笛卡尔集 工作中没有不加条件查询的

**格式2:**select 字段名列表 from 表a 表b where 条件;(仅显示与条件匹配的记录)

mysql> select t1.name,t1.uid,t2.name from t1,t2 where t1.name=t2.name;

mysql> select t1.name,t1.uid,t2.name from t1,t2 where t1.name="root" and t2.name="root";

## 2.2 嵌套查询(where子查询)

把内层查询结果作为外层查询的查询条件

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)

例如:查询userbak表中的name和id值,显示出uid小于uid的平均值

mysql> select name,uid from userbak

where uid < (select avg(uid) from userbak);

例如:查询t2表中的name值,这些name在db3库中的user表中的shell不等于/bin/bash

mysql> select name from t2 where name in (select name from db3.user where shell != "/bin/bash");

## 2.3 左连接查询

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

作用:查询条件成立时,以左边表a为主,显示查询记录

## 2.4 右连接查询

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

作用:查询条件成立时,以右边表b为主,显示查询记录

**案例:**当两个表有相同的结构和部分相同数据,对比出两个表相同和不同数据;

mysql> create table t3 select name,uid,shell from db3.user limit 3;

mysql> create table t4 select name,uid,shell from db3.user limit 6;

mysql> select \* from t3;

mysql> select \* from t4;

**左连接:**

mysql> select \* from t3 left join t4 on t3.uid = t4.uid ;

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

| name | uid | shell | name | uid | shell |

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

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

| bin | 1 | /sbin/nologin | bin | 1 | /sbin/nologin |

| daemon | 2 | /sbin/nologin | daemon | 2 | /sbin/nologin |

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

3 rows in set (0.00 sec)

**右连接:**

mysql> select \* from t3 right join t4 on t3.uid = t4.uid ;

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

| name | uid | shell | name | uid | shell |

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

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

| bin | 1 | /sbin/nologin | bin | 1 | /sbin/nologin |

| daemon | 2 | /sbin/nologin | daemon | 2 | /sbin/nologin |

| NULL | NULL | NULL | adm | 3 | /sbin/nologin |

| NULL | NULL | NULL | lp | 4 | /sbin/nologin |

| NULL | NULL | NULL | sync | 5 | /bin/sync |

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

6 rows in set (0.00 sec)

## 内连接

## 全连接

## 三 MySQL 服务图形管理工具(phpmyadmin)

## 3.1 连接数据库服务的方式

命令行 web页面 图形工具

## 3.2常见的mysql管理工具



PhpMyAdmin:既适合管理员 也适合程序员使用

LAMP:LINUX APPACHE MYSQL PHP

## 3.3部署LAMP:

[root@sql51 ~]# yum -y install php php-mysql httpd

[root@sql51 ~]# systemctl start httpd

[root@sql51 ~]# systemctl enable httpd

[root@sql51 ~]# netstat -nutlp |grep :80

tcp6 0 0 :::80 :::\* LISTEN 5040/httpd

[root@room9pc01 ~]# scp /root/桌面/DBA/数据库/phpMyAdmin-2.11.11-all-languages.tar.gz 192.168.4.51:/root/

[root@sql51 ~]# tar -zvxf phpMyAdmin-2.11.11-all-languages.tar.gz

[root@sql51 ~]#mv phpMyAdmin-2.11.11-all-languages /var/www/html/admin

[root@sql51 ~]# cd /var/www/html/admin

[root@sql51 admin]# ls

[root@sql51 admin]# cp config.sample.inc.php config.inc.php

[root@sql51 admin]# vim config.inc.php

17 $cfg['blowfish\_secret'] = 'plj123'; /\* YOU MUST FILL IN THIS FOR COOKIE AUTH! \*/ #第17行修改内容plj123可以随便输入

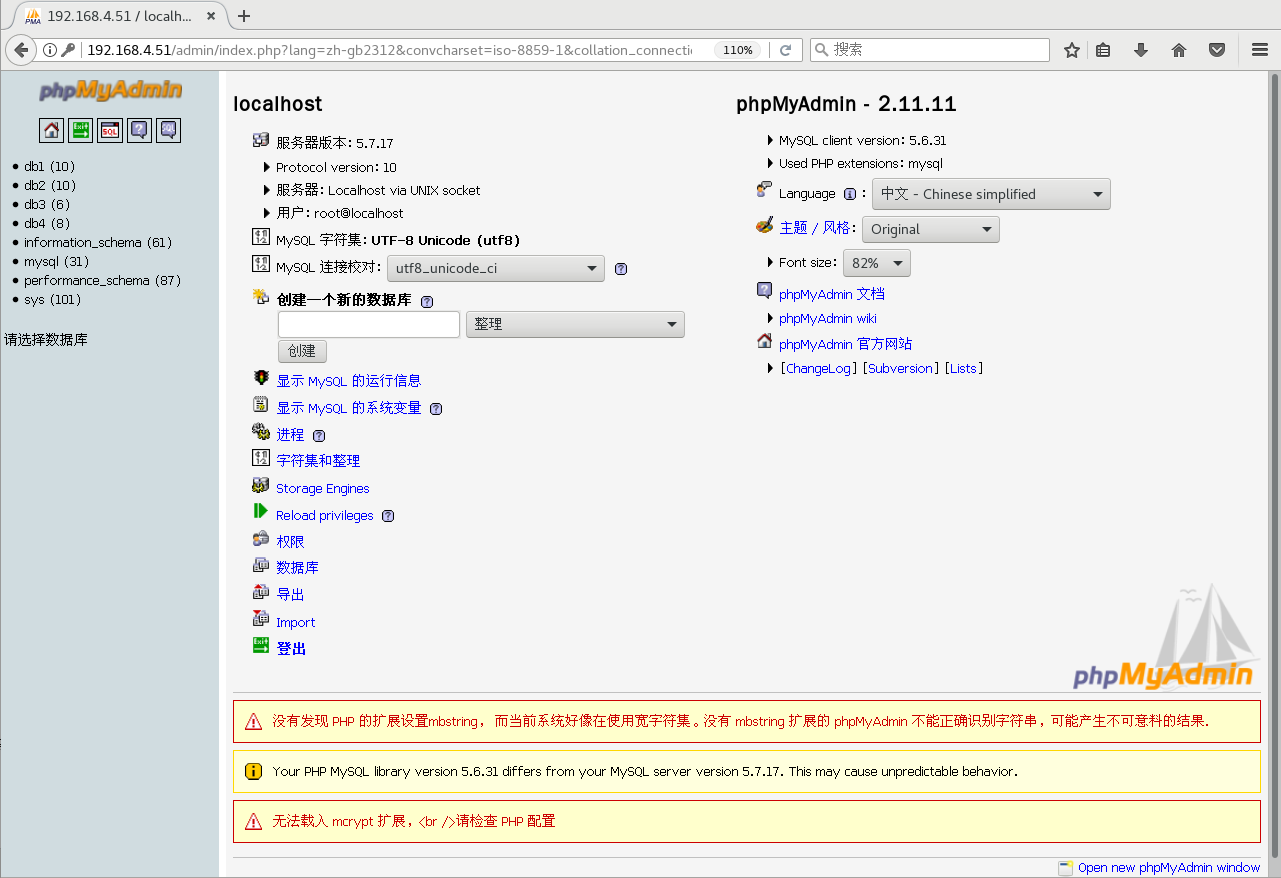
31 $cfg['Servers'][$i]['host'] = 'localhost'; #第31行修改如果数据库服务器在本机就是localhost, 如果不在本机,就填相应的服务器主机IP

客户端访问测试:

在浏览器里面输入 <http://192.168.4.51/admin>



登入名称:root 密码:123456



## 四 用户授权与权限撤销

## 4.1什么是授权

在数据库服务器上,添加连接用户并设置权限

## 4.2授权命令格式

grant 权限列表 on 数据库名 to 用户名@”客户端地址” identified by “密码”

grant 权限列表 on 数据库名.表名 to 用户名@”客户端地址” identified by “密码”

给这个用户授权,这个用户也有授权权限

grant 权限列表 on 数据库名 to 用户名@”客户端地址” identified by “密码”

With grant option; 给这个用胡授权,这个用户也有授权权限

### 4.2.1 权限列表

all : #赋予所有权限

Select,update,insert #赋予权限操作(查询/修改/插入)

Select,update (字段1,...,字段N) #赋予部分字段的查询与修改

### 4.2.2客户端地址

%:匹配所有主机

192.168.1.% :匹配指定的一个网段

192.168.1.1:匹配指定IP地址的单个主机

%.tarena.com:匹配一个DNS区域

svr1.tarena.con:匹配指定域名的单个主机(需要DNS服务器解析)

### 4.2.3授权示例

新建用户mydba,对所有库所有表(语句表示\*.\*)有完全权限

允许从任何地址访问,密码设为”123456” 允许该用户为其他用户授权

mysql> grant all on \*.\* to mydba@"%" identified by "123456" with grant option;

Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> select \* from mysql.user where user="mydba" \G;

### 4.2.4查看授权

mysql> select host,user from mysql.user;

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

| host | user |

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

| % | mydba |

| localhost | mysql.sys |

| localhost | root |

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

3 rows in set (0.00 sec)

mysql> show grants for mydba@"%"; #查看用户权限详情

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

| Grants for mydba@% |

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

| GRANT ALL PRIVILEGES ON \*.\* TO 'mydba'@'%' WITH GRANT OPTION |

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

1 row in set (0.00 sec)

### 4.2.5终端测试与查看

在192.168.4.52用新用户mydba登录

[root@sql52 ~]# mysql -h192.168.4.51 -umydba -p123456

mysql>

#### 查看登录的本机IP

mysql> select user();

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

| user() |

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

| mydba@192.168.4.52 |

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

1 row in set (0.00 sec)

#### 查看已连接的数据库服务器的主机名

mysql> select @@hostname;

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

| @@hostname |

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

| sql51 |

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

1 row in set (0.01 sec)

#### 查看本机登录的权限

mysql> show grants;

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

| Grants for mydba@% |

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

| GRANT ALL PRIVILEGES ON \*.\* TO 'mydba'@'%' WITH GRANT OPTION |

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

1 row in set (0.01 sec)

#### 可再测试对数据的访问权限

Select insert drop create update

#### mydba用户创建新用户并授权

创建admin用户,授予只对db3.user ,有select与insert权限,只限于在数据库服务器本机上登录.登录密码是”123456”

mysql> grant select,insert on db3.user to admin@"localhost" identified by "123456";

Query OK, 0 rows affected, 1 warning (0.01 sec)

创建后在数据库服务器本机(192.168.4.51)上登录测试,

[root@sql51 ~]# mysql -uadmin -p123456

mysql>

mysql> select user();

mysql> show grants;

mysql> 进行select /insert /drop /create等操作

## 4.3授权库(mysql库)

**记录授权信息的表**

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: 存储授权用户对表字段的访问权限

## 4.4撤销权限命令格式

Revoke 权限列表 on 数据库名 from 用户名@”客户端地址”;

mysql> show grants for mydba@"%"; #先查询出来mydba用户的权限:

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

| Grants for mydba@% |

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

| GRANT ALL PRIVILEGES ON \*.\* TO 'mydba'@'%' WITH GRANT OPTION |

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

1 row in set (0.00 sec)

### 撤销授权权限

mysql> revoke grant option on \*.\* from mydba@"%";

Query OK, 0 rows affected (0.00 sec)

mysql> show grants for mydba@"%";

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

| Grants for mydba@% |

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

| GRANT ALL PRIVILEGES ON \*.\* TO 'mydba'@'%' |

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

1 row in set (0.00 sec)

### 撤销删除与修改权限

mysql> revoke delete,update on \*.\* from mydba@"%";

Query OK, 0 rows affected (0.00 sec)

mysql> show grants for mydba@"%"; #

**删除yaya用户,在db7库里面对所有库的查询权限**

mysql> show grants for yaya;

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

| Grants for yaya@% |

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

| GRANT USAGE ON \*.\* TO 'yaya'@'%' |

| GRANT SELECT, INSERT ON `db7`.\* TO 'yaya'@'%' |

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

mysql> revoke select on db7.\* from yaya;

Query OK, 0 rows affected (0.00 sec)

mysql> show grants for yaya;

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

| Grants for yaya@% |

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

| GRANT USAGE ON \*.\* TO 'yaya'@'%' |

| GRANT INSERT ON `db7`.\* TO 'yaya'@'%' |

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

2 rows in set (0.00 sec)

### 撤销所有权限

mysql> revoke all on \*.\* from mydba@"%";

Query OK, 0 rows affected (0.00 sec)

mysql> show grants for mydba@"%";

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

| Grants for mydba@% |

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

| GRANT USAGE ON \*.\* TO 'mydba'@'%' |

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

1 row in set (0.00 sec)

## 4.5 删除授权用户

Drop user 用户名@”客户端地址”;

先查看用户

mysql> select host,user from mysql.user;

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

| host | user |

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

| % | mydba |

| localhost | admin |

| localhost | mysql.sys |

| localhost | root |

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

4 rows in set (0.00 sec)

**删除mydba用户**

mysql> drop user mydba@"%";

Query OK, 0 rows affected (0.00 sec)

mysql> select host,user from mysql.user;

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

| host | user |

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

| localhost | admin |

| localhost | mysql.sys |

| localhost | root |

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

3 rows in set (0.00 sec)

**注意:**此处删除admin时.后面要填”localhost”

mysql> drop user admin@"localhost";

## 4.6 密码恢复以及设置

修改数据库管理员root本机登录密码,只有操作系统管理员root能修改

只有操作系统管理员root才有权限修改配置文件,和修改user表记录

### 恢复步骤

1.停止MySQL服务程序

2.跳过授权表启动MySQL服务程序,将skip-grant-tables 写入/etc/my.cnf

3.重设root密码(更新user表记录)

4.以正常方式重启MySQL服务程序

[root@sql51 ~]# systemctl stop mysqld

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

#secure\_file\_priv="/mydatadir" #注释掉

#default-storage-engine=myisam #注释掉

#validate\_password\_length=6 #注释掉

#validate\_password\_policy=0 #注释掉

skip-grant-tables #新增

[root@sql51 ~]# systemctl start mysqld

[root@sql51 ~]# mysql

mysql> #不需要密码直接登录数据库成功

mysql> use mysql

mysql> show tables;

mysql> desc user;

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

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

| host | user | authentication\_string |

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

| localhost | root | \*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |

| localhost | mysql.sys | \*THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |

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

2 rows in set (0.00 sec)

mysql> update mysql.user set authentication\_string=password("654321")

-> where

-> host="localhost" and user="root";

Query OK, 1 row affected, 1 warning (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 1

mysql> flush privileges;

Query OK, 0 rows affected (0.00 sec)

mysql> quit

Bye

[root@sql51 ~]# systemctl stop mysqld

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

secure\_file\_priv="/mydatadir" #去掉注释

default-storage-engine=myisam #去掉注释

validate\_password\_length=6 #去掉注释

validate\_password\_policy=0 #去掉注释

#skip-grant-tables #注释掉跳过密码验证语句

[root@sql51 ~]# systemctl start mysqld

[root@sql51 ~]# mysql -uroot -p654321

mysql>

1. MySQL管理工具
2. 密码恢复及设置
3. 用户授权及撤销

## 1 MySQL管理工具

1.1 问题

部署LAMP+phpMyAdmin平台

1.2 方案

1. 安装httpd、mysql、php-mysql及相关包

2. 启动httpd服务程序

3. 解压phpMyAdmin包，部署到网站目录

4. 配置config.inc.php，指定MySQL主机地址

5. 创建授权用户

6. 浏览器访问、登录使用

今天课程需要使用1台RHEL7虚拟机，其中一台作为数据服务器（192.168.4.6）、另外一台作为测试用的Linux客户机（192.168.4.254），如图-1所示。

图-1

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：准备软件的运行环境 lamp

[root@mysql6~]# rpm -q httpd php php-mysql //检测是否安装软件包

未安装软件包 httpd

未安装软件包 php

未安装软件包 php-mysql

[root@mysql6~]# yum -y install httpd php php-mysql //装包

[root@mysql6~]# systemctl start httpd //启动服务

[root@mysql6~]# systemctl enable httpd //设置开机自启

Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service to /usr/lib/systemd/system/httpd.service.

步骤二：测试运行环境

[root@mysql6~]# vim /var/www/html/test.php //编辑页面测试文件

[root@mysql6~]# cat /var/www/html/test.php //查看页面测试文件

<?php

$x=mysql\_connect("localhost","root","123456");

if($x){ echo "ok"; }else{ echo "no"; };

?>

[root@mysql6~]# yum -y install elinks //安装测试网页工具

[root@mysql6~]# elinks --dump http://localhost/test.php

Ok //验证测试页面成功

步骤三：安装软件包

1）物理机传输解压包给虚拟机192.168.4.6

[root@room9pc桌面]# scp phpMyAdmin-2.11.11-all-languages.tar.gz 192.168.4.6:/root/

root@192.168.4.6's password:

phpMyAdmin-2.11.11-a 100% 4218KB 122.5MB/s 00:00

2）虚拟机192.168.4.6解压phpMyAdmin-2.11.11-all-languages.tar.gz压缩包

[root@mysql6~]# tar -zxf phpMyAdmin-2.11.11-all-languages.tar.gz -C /var/www/html/ //-C 表示改变至目录

[root@mysql6~]# cd /var/www/html/

[root@mysql6~]# mv phpMyAdmin-2.11.11-all-languages phpmyadmin //改变目录名

[root@mysql6~]# chown -R apache:apache phpmyadmin/ //改变phpmyadmin目录权限

步骤四：修改软件的配置文件定义管理的数据库服务器

切换到部署后的phpmyadmin程序目录，拷贝配置文件，并修改配置以正确指定MySQL服务器的地址

[root@mysql6html]# cd phpmyadmin

[root@mysql6 phpmyadmin]# cp config.sample.inc.php config.inc.php

//备份主配置文件

[root@mysql6 phpmyadmin]# vim config.inc.php //编辑主配置文件

17 $cfg['blowfish\_secret'] = 'plj123'; //给cookie做认证的值，可以随便填写

31 $cfg['Servers'][$i]['host'] = 'localhost'; //指定主机名，定义连接哪台服务器

:wq

步骤五：在客户端访问软件 管理数据库服务器

1. 在客户端访问软件,打开浏览器输入http://192.168.4.6/phpmyadmin(数据库服务器地址) 访问软件，如图-2所示，用户名是root，密码是123456



图-2

2）登入成功后，如图-3示，即可在授权范围内对MySQL数据库进行管理。

/

图-3

## 2 忘记密码后密码恢复及设置

2.1 问题

本案例要求密码恢复及设置，完成以下任务操作：

恢复MySQL管理列表

正常设置管理密码

2.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：重置MySQL管理密码

### 1）停止已运行的MySQL服务程序

[root@dbsvr1 ~]# systemctl stop mysqld.service //停止服务

[root@dbsvr1 ~]# systemctl status mysqld.service //确认状态

mysqld.service - MySQL Server

Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled)

Active: inactive (dead) since 五 2017-04-07 23:01:38 CST; 21s ago

Docs: man:mysqld(8)

http://dev.mysql.com/doc/refman/en/using-systemd.html

Process: 20260 ExecStart=/usr/sbin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.pid $MYSQLD\_OPTS (code=exited, status=0/SUCCESS)

Process: 20238 ExecStartPre=/usr/bin/mysqld\_pre\_systemd (code=exited, status=0/SUCCESS)

Main PID: 20262 (code=exited, status=0/SUCCESS)

### 2）跳过授权表启动MySQL服务程序

这一步主要利用mysqld的 --skip-grant-tables选项

修改my.cnf配置，添加 skip\_grant\_tables=1启动设置：

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

[mysqld]

skip\_grant\_tables=1

.. ..

[root@dbsvr1 ~]# systemctl restart mysqld.service

[root@dbsvr1 ~]# service mysql status

mysqld.service - MySQL Server

Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled)

Active: active (running) since 五 2017-04-07 23:40:20 CST; 40s ago

Docs: man:mysqld(8)

http://dev.mysql.com/doc/refman/en/using-systemd.html

Process: 11698 ExecStart=/usr/sbin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.pid $MYSQLD\_OPTS (code=exited, status=0/SUCCESS)

Process: 11676 ExecStartPre=/usr/bin/mysqld\_pre\_systemd (code=exited, status=0/SUCCESS)

Main PID: 11701 (mysqld)

CGroup: /system.slice/mysqld.service

└─11701 /usr/sbin/mysqld --daemonize --pid-file=/var/run/mysqld/mysqld.p...

### 重设root的密码

使用mysql命令连接到MySQL服务，

由于前一步启动的MySQL服务跳过了授权表，所以可以root从本机直接登录

[root@dbsvr1 ~]# mysql -u root

Enter password: //直接回车即可

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 4

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

进入 mysql> 环境后，通过修改mysql库中user表的相关记录，重设root用户从本机登录的密码：

mysql> UPDATE mysql.user SET authentication\_string=PASSWORD('1234567')

-> WHERE user='root' AND host='localhost'; //重设root的密码

Query OK, 1 row affected, 1 warning (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 1

mysql> FLUSH PRIVILEGES; //刷新授权表

Query OK, 0 rows affected (0.01 sec)

mysql> exit //退出mysql> 环境

Bye

通过执行“FLUSH PRIVILEGES;”可使授权表立即生效，对于正常运行的MySQL服务，也可以用上述方法来修改密码，不用重启服务。本例中因为是恢复密码，最好重启MySQL服务程序，所以上述“FLUSH PRIVILEGES;”操作可跳过。

### 4) 验证新密码

重新以正常方式启动MySQL服务程序，

如果前面是修改/etc/my.cnf配置的方法来跳过授权表，则重置root密码后，应去除相应的设置以恢复正常：

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

[mysqld]

#skip\_grant\_tables=1 //注释掉或删除此行

.. ..

按正常方式，通过mysql脚本重启服务即可：

[root@dbsvr1 ~]# systemctl restart mysqld.service

验证无密码登录时，将会被拒绝：

[root@dbsvr1 ~]# mysql -u root

Enter password: //没有跳过授权表回车会报错

ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)

只有提供重置后的新密码，才能成功登入：

[root@dbsvr1 ~]# mysql -u root –p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 4

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

### 步骤二：正常设置MySQL管理密码

正常的前提是：已知当前MySQL管理用户（root）的密码。

#### 1）方法1，在Shell命令行下设置

使用mysqladmin管理工具，需要验证旧的密码。比如，以下操作将会把root的密码设置为 1234567：

[root@dbsvr1 ~]# mysqladmin -u root -p password '1234567'

Enter password: //验证原来的密码

mysqladmin: [Warning] Using a password on the command line interface can be insecure.

Warning: Since password will be sent to server in plain text, use ssl connection to ensure password safety. //提示明文修改不安全，并不是报错

#### 2) 方法2，SET PASSWORD修改密码

以root登入mysql> 后，使用SET PASSWORD指令设置

这个与新安装MySQL-server后首次修改密码时要求的方式相同，平时也可以用：

mysql> SET PASSWORD FOR root@localhost=PASSWORD('1234567');

Query OK, 0 rows affected, 1 warning (0.00 sec)

#### 方法3，GRANT授权工具设置

以root登入mysql> 后，使用GRANT授权工具设置

这个是最常见的用户授权方式（下一节会做更多授权的练习）：

mysql> GRANT all ON \*.\* TO root@localhost IDENTIFIED BY '1234567';

Query OK, 0 rows affected, 1 warning (0.00 sec)

#### 4) 方法4，UPDATE更新相应的表记录

以root登入mysql> 后，使用UPDATE更新相应的表记录

这种方法与恢复密码时的操作相同：

mysql> UPDATE mysql.user SET authentication\_string=PASSWORD('1234567')

-> WHERE user='root' AND host='localhost'; //重设root的密码

Query OK, 0 rows affected, 1 warning (0.00 sec)

Rows matched: 1 Changed: 0 Warnings: 1

mysql> FLUSH PRIVILEGES; //刷新授权表

Query OK, 0 rows affected (0.00 sec)

在上述方法中，需要特别注意：当MySQL服务程序以 skip-grant-tables 选项启动时，如果未执行“FLUSH PRIVILEGES;”操作，是无法通过SET PASSWORD或者GRANT方式来设置密码的。比如，验证这两种方式时，都会看到ERROR 1290的出错提示：

mysql> SET PASSWORD FOR root@localhost=PASSWORD('1234567');

ERROR 1290 (HY000): The MySQL server is running with the --skip-grant-tables option so it cannot execute this statement

mysql> GRANT all ON \*.\* TO root@localhost IDENTIFIED BY '1234567';

ERROR 1290 (HY000): The MySQL server is running with the --skip-grant-tables option so it cannot execute this statement

## 3 用户授权及撤销

3.1 问题

允许root从192.168.4.0/24网段 访问，对所有库/表有完全权限，密码为tarena

添加一个管理账号dba007，完全控制及授权

撤销root从本机访问的权限，然后恢复

允许webuser从任意客户机登录，只对webdb库有完全权限，密码为 888888

撤销webuser的完全权限，改为查询权限

3.2 方案

使用2台RHEL 7虚拟机，如图-1所示。其中192.168.4.10是MySQL服务器，授权及撤销操作均在此服务器上执行；而192.168.4.120作为测试客户机，需要安装好MySQL-client软件包，以便提供mysql命令。

图-1

同时，MySQL服务器本身（192.168.4.10）也可以作为测试客户机。

3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：用户授权及撤销

1）允许root从192.168.4.0/24访问，对所有库表有完全权限，密码为tarena。

授权之前，从192.168.4.0/24网段的客户机访问时，将会被拒绝：

[root@host120 ~]# mysql -u root -p -h 192.168.4.10

Enter password: //输入正确的密码

ERROR 2003 (HY000): Host '192.168.4.120' is not allowed to connect to this MySQL server

授权操作，此处可设置与从localhost访问时不同的密码：

mysql> GRANT all ON \*.\* TO root@'192.168.4.%' IDENTIFIED BY 'tarena';

Query OK, 0 rows affected (0.00 sec)

再次从192.168.4.0/24网段的客户机访问时，输入正确的密码后可登入：

[root@host120 ~]# mysql -u root -p -h 192.168.4.10

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 20

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

从网络登入后，测试新建一个库、查看所有库：

mysql> CREATE DATABASE rootdb; //创建新库rootdb

Query OK, 1 row affected (0.06 sec)

mysql> SHOW DATABASES;

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

| Database |

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

| information\_schema |

| home |

| mysql |

| performance\_schema |

| rootdb | //新建的rootdb库

| sys |

| userdb |

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

7 rows in set (0.01 sec)

2）在Mysql服务器上建立一个管理账号dba007，对所有库完全控制，并赋予其授权的权限

新建账号并授权：

mysql> GRANT all ON \*.\* TO dba007@localhost

-> IDENTIFIED BY '1234567'

-> WITH GRANT OPTION;

Query OK, 0 rows affected (0.00 sec)

查看dba007的权限：

mysql> SHOW GRANTS FOR dba007@localhost;

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

| Grants for dba007@localhost |

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

| GRANT ALL PRIVILEGES ON \*.\* TO 'dba007'@'localhost' WITH GRANT OPTION |

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

1 row in set (0.00 sec)

3）撤销root从本机访问的权限，然后恢复

注意：如果没有事先建立其他管理账号，请不要轻易撤销root用户的本地访问权限，否则恢复起来会比较困难，甚至不得不重装数据库。

撤销root对数据库的操作权限：

mysql> REVOKE all ON \*.\* FROM root@localhost;

Query OK, 0 rows affected (0.00 sec)

mysql> SHOW GRANTS FOR root@localhost;

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

| Grants for root@localhost |

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

| GRANT USAGE ON \*.\* TO 'root'@'localhost' WITH GRANT OPTION |

| GRANT PROXY ON ''@'' TO 'root'@'localhost' WITH GRANT OPTION |

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

2 rows in set (0.00 sec)

验证撤销后的权限效果：

mysql> exit //退出当前MySQL连接

Bye

[root@dbsvr1 ~]# mysql -u root -p //重新以root从本地登入

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 6

Server version: 5.6.15 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE newdb2014; //尝试新建库失败

ERROR 1044 (42000): Access denied for user 'root'@'localhost' to database 'newdb2014'

mysql> DROP DATABASE rootdb; //尝试删除库失败

ERROR 1044 (42000): Access denied for user 'root'@'localhost' to database 'rootdb'

尝试以当前的root用户恢复权限，也会失败（无权更新授权表）：

mysql> GRANT all ON \*.\* TO root@localhost IDENTIFIED BY '1234567';

ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)

怎么办呢？

退出当前MySQL连接，以上一步添加的管理账号dba007登入：

mysql> exit //退出当前MySQL连接

Bye

[root@dbsvr1 ~]# mysql -u dba007 -p //以另一个管理账号登入

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 24

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

由管理账号dba007重新为root添加本地访问权限：

mysql> GRANT all ON \*.\* TO root@localhost IDENTIFIED BY '1234567';

Query OK, 0 rows affected (0.00 sec)

mysql> SHOW GRANTS FOR root@localhost; //查看恢复结果

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

| Grants for root@localhost |

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

| GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'localhost' WITH GRANT OPTION |

| GRANT PROXY ON ''@'' TO 'root'@'localhost' WITH GRANT OPTION |

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

2 rows in set (0.00 sec)

退出，再重新以root登入，测试一下看看，权限又恢复了吧：

mysql> exit //退出当前MySQL连接

Bye

[root@dbsvr1 ~]# mysql -u root -p //重新以root登入

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 25

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE newdb2014; //成功创建新库

Query OK, 1 row affected (0.00 sec)

4）允许webuser从任意客户机登录，只对webdb库有完全权限，密码为 888888

添加授权：

mysql> GRANT all ON webdb.\* TO webuser@'%' IDENTIFIED BY '888888';

Query OK, 0 rows affected (0.00 sec)

查看授权结果：

mysql> SHOW GRANTS FOR webuser@'%';

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

| Grants for webuser@% |

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

| GRANT USAGE ON \*.\* TO 'webuser'@'%' |

| GRANT ALL PRIVILEGES ON `webdb`.\* TO 'webuser'@'%' |

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

2 rows in set (0.00 sec)

5）撤销webuser的完全权限，改为查询权限

撤销所有权限：

mysql> REVOKE all ON webdb.\* FROM webuser@'%';

Query OK, 0 rows affected (0.00 sec)

只赋予查询权限：

mysql> GRANT select ON webdb.\* TO webuser@'%';

Query OK, 0 rows affected (0.00 sec)

确认授权更改结果：

mysql> SHOW GRANTS FOR webuser@'%';

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

| Grants for webuser@% |

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

| GRANT USAGE ON \*.\* TO 'webuser'@'%' |

| GRANT SELECT ON `webdb`.\* TO 'webuser'@'%' |

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

2 rows in set (0.00 sec)

# NSD DBA1 DAY05

## 一 数据备份方式

* 物理备份
* 逻辑备份

## 常见Mysql备份工具

物理备份

Mysqldump备份

XtraBackup 工具

## 物理备份:

又叫冷备:直接拷贝数据库里面的数据文件,冷备:cp tar ...

### 物理备份缺点

* 占用物理存储空间
* 备份时间长
* 备份简单
* 跨平台性差

#### 方式一:cp方式备份

[root@sql51 ~]# mkdir /mysqlbak

[root@sql51 ~]# cp -r /var/lib/mysql /mysqlbak/mysql201902180919

[root@sql51 ~]# cd /mysqlbak/mysql201902180919

#### 方式二:tar方式备份

[root@sql51 ~]# tar -zcvf /mysqlbak/mysql.tar.gz /var/lib/mysql

[root@sql51 ~]# cd /mysqlbak/

[root@sql51 mysqlbak]# ls

mysql201902180919 mysql.tar.gz

## 物理恢复:

:先停数据库,再将拷贝解压的服务器放到

拷贝备份的数据库文件 查看服务

[root@sql51 mysqlbak]# scp mysql.tar.gz 192.168.4.52:/root/

[root@sql52 ~]# rpm -qa |grep mysql

将冷备份的所有文件拷贝到新的数据库目录里面.并设置所有者所属组为mysql

## 逻辑备份:

使用命令对数据做备份,(开发者开发好程序备份),根据执行备份时已有的数据生成对应的备份文件.

## 逻辑恢复:

使用命令调用备份文件,做数据恢复

## 二 数据备份策略

以下全是逻辑备份

一 完全备份 :备份所有数据

二 备份新产生的数:

1.增量备份:备份上次备份后所有新产生的数据

2.差异备份:备份完全备份后所有新产生的数据

选择备份策略考虑的问题:

数据备份方式:物理 逻辑

数据备份策略:完全 增量 差异

数据备份频率:根据数据产生的量

数据备份时间:数据库服务器访问量少时,执行数据备份操作

数据备份文件的命名:日期\_库名.sql

数据备份文件的存储空间:将备份文件,放到单独存储服务器上(可用iscsi,ceph),并且存储 服务器空间可动态扩展(可用LV)

数据备份手段:1 crond + 数据备份脚本

2 部署服务,实现数据备份(mysql 主从同步)

实际环境中通常使用的备份策略

完全+差异

完全+差异

## 2.1完全备份/恢复 mysqldump

### 2.1.1 mysqldump备份缺点

* 在备份与恢复时对正在备份表会加写锁,
* 完全备份后,新产生的数据无法恢复
* 效率低,备份和还原速度慢

### 2.1.2 mysqldump备份

备份单个数据库:

mysqldump -u用户 -p密码 数据库名 >备份文件名

恢复时需要先建立相同名称的空库

备份所有的数据库:

mysqldump -u用户 -p密码 all >备份文件名

[root@dbsvr1 ~]# mysqldump -u root -p --all-databases > /root/alldb.sql

备份单张表:

mysqldump -u用户 -p密码 数据库名 表名 >备份文件名 #数据库名表名不带点

备份多个库:

mysqldump -u用户 -p密码 -B 数据库名1 数据库名2 表名 >备份文件名

### 2.1.3 mysqldump恢复

恢复数据库:

备份数据库时用-B 备份的多个数据库时,恢复时可以省略数据库名

mysql -u用户 -p密码 <备份文件名 #用户必须有登录备份权限

恢复数据表需要加库名

mysql -u用户 -p密码 库名 <备份文件名 #用户必须有登录备份权限

### 2.1.4 mysqldump案例:

备份51上的db3库的user表,在53上恢复

[root@sql51 mysqlbak]# mysqldump -uroot -p654321 db3 user >/mysqlbak/db3\_user.sql

[root@sql51 mysqlbak]# scp /mysqlbak/db3\_user.sql 192.168.4.53:/root

root@sql53 lib]# mysql -uroot -p654321 db3

mysql> drop tanle user;

mysql> show tables;

[root@sql53 ~]# mysql -uroot -p654321 db3 < /root/db3\_user.sql

[root@sql53 ~]# mysql -uroot -p654321 db3

mysql> show tables; #事先删除了user表,现查看多了user表

备份51上的db2库和db4库, 在53上恢复

备份db2 db4

[root@sql51 mysqlbak]# mysqldump -uroot -p654321 -B db2 db4 >/mysqlbak/twodb.sql

[root@sql51 mysqlbak]# scp /mysqlbak/twodb.sql 192.168.4.53:/root/

先在53删除db2 , db4库

mysql> drop database db2;

Query OK, 10 rows affected (1.52 sec)

mysql> drop database db4;

Query OK, 8 rows affected (0.00 sec)

mysql> exit

Bye

[root@sql53 ~]# mysql -uroot -p654321 </root/twodb.sql

## 2.2实时增量备份/恢复 binlog日志

备份上次备份后所有新产生的数据

启用mysql服务的binlog日志实现

### 2.2.1 binlog日志介绍

binlog日志是mysql服务日志文件的一种,又被称为二进制日志,记录在服务器上执行的除查询之外的sql命令

查询命令:desc select show

### 2.2.2启用binlog日志

mysql> show master status; #查看是否启用

Empty set (0.00 sec) #查询出来的为空未启用

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

server\_id=51 #不能与别的数据库服务器相同,习惯性做法为主机地址

log\_bin #启用

[root@sql51 ~]# systemctl restart mysqld

[root@sql51 ~]# mysql -uroot -p654321

mysql> show master status;

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

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

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

| sql51-bin.000001 | 154 | | | |

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

1 row in set (0.00 sec)

[root@sql51 ~]# ls /var/lib/mysql

ca.pem db2 ibdata1 mysql private\_key.pem sql51-bin.000001

client-cert.pem db3 ib\_logfile0 mysql.sock public\_key.pem sql51-bin.index

#启用之后默认会有主机名-bin.000001(日志文件) 和 主机名-bin.index(索引)两个文件

#默认日志大于500M就会新增一个日志文件

### 2.2.3启用binlog并指定日志文件目录:

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

server\_id=51

log\_bin=/logdir/plj #日志文件将以plj开头

[root@sql51 ~]# mkdir /logdir

[root@sql51 ~]# chown mysql /logdir

[root@sql51 ~]# systemctl restart mysqld

root@sql51 ~]# ls /logdir/ #日志文件产生在这里了

plj.000001 plj.index

[root@sql51 ~]# mysql -uroot -p654321

mysql> show master status;

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

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

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

| plj.000001 | 154 | | | |

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

1 row in set (0.00 sec)

### 2.2.4手动创建新的binlog日志文件

#### 2.2.4.1重启mysql服务

[root@sql51 ~]# systemctl restart mysqld

[root@sql51 ~]# ls /logdir/ #新产生了plj.000002日志文件

plj.000001 plj.000002 plj.index

#### 2.2.4.2执行sql操作 mysql>flush logs;

mysql> flush logs;

Query OK, 0 rows affected (0.11 sec)

mysql> exit

Bye

[root@sql51 ~]# ls /logdir/ #新产生了plj.000003日志文件

plj.000001 plj.000002 plj.000003 plj.index

#### 2.2.4.3执行mysql -uroot -p654321 -e "flush logs"

[root@sql51 ~]# mysql -uroot -p654321 -e "flush logs"

mysql: [Warning] Using a password on the command line interface can be insecure.

[root@sql51 ~]# ls /logdir/ #新产生了plj.000004日志文件

plj.000001 plj.000002 plj.000003 plj.000004 plj.index

#### 2.4.2.4备份时--flush-logs

[root@sql51 ~]# mysqldump -uroot -p654321 --flush-logs db1 > /mysqlbak/db1logtest.sql

[root@sql51 ~]# ls /logdir/ #新产生了plj.000005日志文件

plj.000001 plj.000002 plj.000003 plj.000004 plj.000005 plj.index

### 2.2.5清理binlog日志

删除binlog日志文件,索引(主机名.index)文件也会跟着变更

删除早于指定版本的binlog日志:

purge master logs to “binlog日志文件名”

例如:删除000005之前的日志文件不包括00005

mysql> purge master logs to "plj.000005"; Query OK, 0 rows affected (0.09 sec)

mysql> system ls /logdir

plj.000005 plj.index

删除所有binlog日志,重建新日志

reset master;

mysql> reset master;

Query OK, 0 rows affected (0.11 sec)

mysql> system ls /logdir

plj.000001 plj.index

### 2.2.6查看分析binlog日志文件内容

#### 2.2.6.1 binlog三种日志格式

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

2 row(行模式):不记录sql语句上下文相关信, 仅保存哪条记录被修改,记录具体的所有操 作,binlog增长非常快,

3 mixed(混合模式):是以上两种格式的混合使用

#### 2.2.6.2 查看/修改binlog日志格式

mysql> show variables like "binlog\_format"; #查看

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

| Variable\_name | Value |

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

| binlog\_format | ROW |

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

1 row in set (0.01 sec)

[root@sql51 ~]# vim /etc/my.cnf #修改

[mysqld]

server\_id=51

log\_bin=/logdir/plj

binlog\_format=mixed #row statement

[root@sql51 ~]# systemctl restart mysqld

mysql> show variables like "binlog\_format";

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

| Variable\_name | Value |

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

| binlog\_format | MIXED |

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

1 row in set (0.01 sec)

#### 2.2.6.3 日志文件如何区分记录的多条sql命令

**1.偏移量**

mysql> show master status;

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

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

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

| plj.000001 | 154 | | | |

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

1 row in set (0.00 sec)

**2.时间点**

[root@sql51 ~]# mysqlbinlog /logdir/plj.000002

#190218 15:45:16 server id 51 end\_log\_pos 154 CRC32 0x4927dc4f

#### 2.2.6.4 查看日志文件内容

mysqlbinlog 日志文件

[root@sql51 ~]# mysqlbinlog /logdir/plj.000002

/\*!50530 SET @@SESSION.PSEUDO\_SLAVE\_MODE=1\*/;

/\*!50003 SET @OLD\_COMPLETION\_TYPE=@@COMPLETION\_TYPE,COMPLETION\_TYPE=0\*/;

DELIMITER /\*!\*/;

# at 4

#190218 15:45:15 server id 51 end\_log\_pos 123 CRC32 0x4f02b0cb Start: binlog v 4, server v 5.7.17-log created 190218 15:45:15 at startup

# Warning: this binlog is either in use or was not closed properly.

ROLLBACK/\*!\*/;

BINLOG '

i2JqXA8zAAAAdwAAAHsAAAABAAQANS43LjE3LWxvZwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAACLYmpcEzgNAAgAEgAEBAQEEgAAXwAEGggAAAAICAgCAAAACgoKKioAEjQA

AcuwAk8=

'/\*!\*/;

# at 123

#190218 15:45:16 server id 51 end\_log\_pos 154 CRC32 0x4927dc4f Previous-GTIDs

# [empty]

SET @@SESSION.GTID\_NEXT= 'AUTOMATIC' /\* added by mysqlbinlog \*/ /\*!\*/;

DELIMITER ;

# End of log file

/\*!50003 SET COMPLETION\_TYPE=@OLD\_COMPLETION\_TYPE\*/;

/\*!50530 SET @@SESSION.PSEUDO\_SLAVE\_MODE=0\*/;

**例将binlog日志修改格式之后插入数据,再看日志**

mysql> show master status;

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

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

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

| plj.000001 | 154 | | | |

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

1 row in set (0.00 sec)

mysql> desc db1.t2 ;

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

| Field | Type | Null | Key | Default | Extra |

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

| age | tinyint(4) | YES | | NULL | |

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

1 row in set (0.00 sec)

mysql> select \* from db1.t2;

Empty set (0.00 sec)

mysql> insert into db1.t2 values(1);

Query OK, 1 row affected (0.07 sec)

mysql> insert into db1.t2 values(2);

Query OK, 1 row affected (0.09 sec)

mysql> insert into db1.t2 values(3);

Query OK, 1 row affected (0.03 sec)

mysql> select \* from db1.t2;

+------+

| age |

+------+

| 1 |

| 2 |

| 3 |

+------+

3 rows in set (0.00 sec)

mysql> show master status;

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

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

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

| plj.000001 | 955 | | | |

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

1 row in set (0.00 sec)

[root@sql51 ~]# mysqlbinlog /logdir/plj.000001 #开始查看

#190218 15:57:15 server id 51 end\_log\_pos 390 CRC32 0x658ffeba Query thread\_id=3 exec\_time=0 error\_code=0

SET TIMESTAMP=1550476635/\*!\*/;

insert into db1.t2 values(1)

/\*!\*/;

# at 390

#190218 15:57:22 server id 51 end\_log\_pos 657 CRC32 0x48a16d1b Query thread\_id=3 exec\_time=0 error\_code=0

SET TIMESTAMP=1550476642/\*!\*/;

insert into db1.t2 values(2)

/\*!\*/;

# at 657

#190218 15:57:25 server id 51 end\_log\_pos 924 CRC32 0xf9dd53ec Query thread\_id=3 exec\_time=0 error\_code=0

SET TIMESTAMP=1550476645/\*!\*/;

insert into db1.t2 values(3)

/\*!\*/;

# at 924

另一种查看方式:

[root@sql51 ~]# mysqlbinlog /logdir/plj.000001 | grep insert

insert into db1.t2 values(1)

insert into db1.t2 values(2)

insert into db1.t2 values(3)

### 2.2.7使用日志文件恢复数据

#### 2.2.7.1输出日志格式

Mysqlbinlog [选项] binlog 日志文件名

常用选项:

--start-datetime=”yyyy-mm-dd hh:mm:ss” 开始时间

--stop-datetime=”yyyy-mm-dd hh:mm:ss” 结束时间

--start-position=数字 #开始偏移量

--stop-position=数字 #结束偏移量

指定开始和结束可以指定一个范围

[root@sql51 ~]# scp /logdir/plj.000001 192.168.4.52:/root

查看日志里面已有的插入操作

[root@sql51 ~]# mysqlbinlog /logdir/plj.000001 | grep insert

insert into db1.t2 values(1)

insert into db1.t2 values(2)

insert into db1.t2 values(3)

#### 2.2.7.2恢复这个binlog日志的所有操作:

[root@sql52 mysqld]# mysqlbinlog /root/plj.000001 | mysql -uroot -p654321

mysql> select \* from db1.t2;

+------+

| age |

+------+

| 1 |

| 2 |

| 3 |

+------+ #以binlog日志文件给恢复了三条数据

3 rows in set (0.00 sec)

#### 2.2.7.3恢复指定范围内的数据:

[root@sql52 ~]# mysqlbinlog --start-position=560 --stop-position=924 /root/plj.000001 |mysql -uroot -p654321

mysql> select \* from db1.t2 ;

+------+

| age |

+------+

| 1 |

| 2 |

| 3 |

| 2 | #以binlog日志文件给恢复了三条数据中的一条

+------+

4 rows in set (0.00 sec)

\*\*选择范围必须包含日志文件的commit

\*\*不指定开头,默认从最前面开始

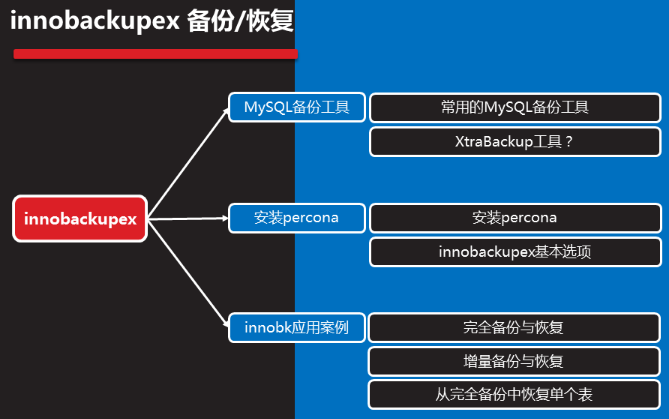
\*\*不指定结束,默认执行到最后

## 2.3差异备份

备份上次备份后所有新产生的数据

常用的Mysql备份工具

## 三 innobackupex备份/恢复



## 3.1 XtraBackup 工具介绍

是一款强大的在线热备份工具

备份过程中不锁库表,行级锁,适合实际生产环境

由专业组织percona开源软件提供(改进Mysql分支)

主要含有两个组件:

xtrabackup:C程序,支持innodb(数据库引擎)/XtraDB

Innobackupex命令:以perl脚本封装xtrabackup,安装percona软件提供

## 3.2 percona-xtrabackup软件安装使用

环境准备:

* 删掉所有的没用的库
* 修改默认的存储引擎为:innodb,要使用xtrabackup工具,必须使用此存储引擎
* 安装percona-xtrabackup,需要依赖libev-4.15-1.el6.rf.x86\_64,光盘不提供,提前拷贝到数据库服务器,并安装

[root@sql51 ~]# yum -y install libev-4.15-1.el6.rf.x86\_64.rpm

[root@sql51 ~]# yum install -y percona-xtrabackup-24-2.4.7-1.el7.x86\_64.rpm

Innobackupex

## 3.3 Innobackupex命令格式

Innobackupex 选项

--host 主机名

--user 用户名

--port 端口号(默认3306可不用写)

--databases 数据库名(不写将备份所有数据库)

--no-timestamp 不用时间戳,不用日期命名备份文件存储的子目录名

案例:将所有库完全备份到/backup(目录不需事先创建,备份程序自动创建)

格式:Innobackupex --user root --password 123456 /backup -no-timestamp

Innobackupex 备份之后有两种文件1是数据文件,2是备份信息文件

1数据文件,用来恢复数据

2备份配置文件:有备份信息,用来做增量备份或者差异备份等的参考



## 3.4数据备份与恢复

### 3.4.1数据完全备份与恢复

Innobackupex恢复数据要求数据库清空mysql目录,因此那怕你只需要备份多个库中其中一个库,也要将数据库三个系统库(mysql sys performance\_schema)一起备份

#### 3.4.1.1完全备份:

案例:将所有库完全备份到/backup(目录不需事先创建,备份程序自动创建)

格式:Innobackupex --user root --password 123456 /allbak -no-timestamp

Innobackupex 选项

--host 主机名

--user 用户名

--port 端口号(默认3306可不用写)

--databases 数据库名(不写将备份所有数据库)

--no-timestamp 不用时间戳,不用日期命名备份文件存储的子目录名

Innobackupex 备份之后有两种文件1是数据文件,2是备份信息文件

1数据文件,用来恢复数据

2备份配置文件:有备份信息,用来做增量备份或者差异备份等的参考

#### 3.4.1.2完全恢复:

选项:



备份的数据中有以下文件:

Xtra backup\_logfile已提交日志

Ibdata1文件是未提交数据日志

1 准备恢复

[root@sql51 ~]# cat /allbak//xtrabackup\_checkpoints

backup\_type = full-backuped #备份内容完全备

from\_lsn = 0

to\_lsn = 3038691

last\_lsn = 3038700 #数据库下次备份开始序列号

compact = 0

recover\_binlog\_info = 0

from\_lsn = 0 to\_lsn = 3038691 日志范围

[root@sql51 ~]# innobackupex --apply-log /allbak #准备还原

190219 10:12:44 completed OK! #看到OK就正常

[root@sql51 ~]# cat /allbak/xtrabackup\_checkpoints

backup\_type = full-prepared

from\_lsn = 0

to\_lsn = 3038691

last\_lsn = 3038700

compact = 0

recover\_binlog\_info = 0

2 把备份目录下的数据拷贝到数据库目录下

[root@sql51 ~]# systemctl stop mysqld.service

[root@sql51 ~]# rm -rf /var/lib/mysql

[root@sql51 ~]# ls /var/lib/mysql

ls: 无法访问/var/lib/mysql: 没有那个文件或目录

[root@sql51 ~]# innobackupex --copy-back /allbak #开始还原

190219 10:19:35 innobackupex: Starting the copy-back operation

190219 10:19:44 completed OK!

[root@sql51 ~]# ls /var/lib/mysql -l #查看数据库目录里面的所有文件权限为root

[root@sql51 ~]# chown -R mysql:mysql /var/lib/mysql

[root@sql51 ~]# systemctl start mysqld

[root@sql51 ~]# ls -l /var/lib/mysql

[root@sql51 ~]# mysql -uroot -p654321

mysql>

mysql> show databases;

#### 3.4.1.3综合练习:

将51上的备份文件 恢复到52数据库服务器上

51:

[root@sql51 allbak]# scp -r /allbak 192.168.4.52:/

52:

1安装percona-xtrabackup

[root@room9pc01 ~]# scp /root/桌面/libev-4.15-1.el6.rf.x86\_64.rpm 192.168.4.52:/root

[root@room9pc01 ~]# scp /root/桌面/percona-xtrabackup-24-2.4.7-1.el7.x86\_64.rpm 192.168.4.52:/root

[root@sql52 ~]# yum -y install libev-4.15-1.el6.rf.x86\_64.rpm

[root@sql52 ~]# yum -y install percona-xtrabackup-24-2.4.7-1.el7.x86\_64.rpm

[root@sql52 ~]# systemctl stop mysqld.service

[root@sql52 ~]# rm -rf /var/lib/mysql

[root@sql52 ~]# ls /var/lib/mysql

ls: 无法访问/var/lib/mysql: 没有那个文件或目录

[root@sql52 ~]# innobackupex --apply-log /allbak #准备还原

[root@sql52 ~]# innobackupex --copy-back /allbak #执行还原

[root@sql52 ~]# ls -l /var/lib/mysql

[root@sql52 ~]# chown -R mysql:mysql /var/lib/mysql

[root@sql52 ~]# systemctl start mysqld

[root@sql52 ~]# mysql -uroot -p654321

mysql> show databases;

### 3.4.2数据增量备份与恢复

[root@sql51 allbak]# ls

backup-my.cnf ibdata1 ibtmp1 sys xtrabackup\_checkpoints

db5 ib\_logfile0 mysql xtrabackup\_binlog\_info xtrabackup\_info

ib\_buffer\_pool ib\_logfile1 performance\_schema xtrabackup\_binlog\_pos\_innodb xtrabackup\_logfile

查看配置文件

[root@sql51 ~]# cat /allbak//xtrabackup\_checkpoints

backup\_type = full-backuped #备份内容完全备

from\_lsn = 0

to\_lsn = 3038691

last\_lsn = 3038700 #数据库下次备份开始序列号

compact = 0

recover\_binlog\_info = 0

#from\_lsn = 0 to\_lsn = 3038691 0--3038691日志序列号范围

如何知道新产生的数据

**格式:**innobackupex --user root --password 654321 --incremental /new1dir --incerments-basedir=/fullbak

**增量备份:先做完全备份,在完全备份的基础上进行增量备份**

备份

#### 首先做增量完全备份-备份到/fullbak

[root@sql51 allbak]# innobackupex --user root --password 654321 /fullbak --no-timestamp

[root@sql51 allbak]# cat /fullbak/xtrabackup\_checkpoints

backup\_type = full-backuped

from\_lsn = 0

to\_lsn = 3039291

last\_lsn = 3039300

compact = 0

recover\_binlog\_info = 0

mysql> insert into db5.a values(11),(22); #备份完之后再插入数据

Query OK, 2 rows affected (0.08 sec)

Records: 2 Duplicates: 0 Warnings: 0

#### 第1次做增量备份-备份到/new1dir目录

[root@sql51 allbak]# innobackupex --user root --password 654321 --incremental /new1dir --incremental-basedir=/fullbak --no-timestamp

190219 11:53:46 completed OK!

#### 第2次增量备份-备份到/new2dir目录

[root@sql51 allbak]# innobackupex --user root --password 654321 --incremental /new2dir --incremental-basedir=/new1dir --no-timestamp

#### 第3次增量备份-备份到/new3dir目录

[root@sql51 ~]# innobackupex --user root --password 654321 --incremental /new3dir --incremental-basedir=/new2dir --no-timestamp

#### 做完备份之后所有的的备份目录

[root@sql51 allbak]# ls /fullbak/ #首先做增量完全备份的目录

backup-my.cnf ibdata1 sys xtrabackup\_info

db5 mysql xtrabackup\_binlog\_info xtrabackup\_logfile

ib\_buffer\_pool performance\_schema xtrabackup\_checkpoints

[root@sql51 allbak]# ls /new1dir/ #第一次增量备份的目录

backup-my.cnf ib\_buffer\_pool ibdata1.meta performance\_schema xtrabackup\_binlog\_info xtrabackup\_info

db5 ibdata1.delta mysql sys xtrabackup\_checkpoints xtrabackup\_logfile

[root@sql51 allbak]# ls /new2dir/ #第二次增量备份的目录

backup-my.cnf ib\_buffer\_pool ibdata1.meta performance\_schema xtrabackup\_binlog\_info xtrabackup\_info

db5 ibdata1.delta mysql sys xtrabackup\_checkpoints xtrabackup\_logfile

#### 依次做完备份后xtrabackup\_checkpoints备份信息

[root@sql51 ~]# cat /fullbak/xtrabackup\_checkpoints

backup\_type = full-backuped #注意此处是完全备份状态

from\_lsn = 0

to\_lsn = 3041903 #fullbak范围0-3041903

last\_lsn = 3041912

compact = 0

recover\_binlog\_info = 0

[root@sql51 ~]# cat /new1dir/xtrabackup\_checkpoints

backup\_type = incremental

from\_lsn = 3041903

to\_lsn = 3042229 #new1范围3041903-3042229

last\_lsn = 3042238

compact = 0

recover\_binlog\_info = 0

[root@sql51 ~]# cat /new2dir/xtrabackup\_checkpoints

backup\_type = incremental

from\_lsn = 3042229

to\_lsn = 3042571 #new2范围3042229-3042571

last\_lsn = 3042580

compact = 0

recover\_binlog\_info = 0

范围第二次的to\_lsn>第一次的to\_lsn>首先增量完整备份

#### 增量恢复

整合首次完整备份的文件

[root@sql51 ~]# innobackupex --apply-log --redo-only /fullbak

[root@sql51 ~]# cat /fullbak/xtrabackup\_checkpoints

backup\_type = log-applied #注意此处:变成了增量恢复状态

from\_lsn = 0

to\_lsn = 3041903 #范围0-3041903

last\_lsn = 3041912

compact = 0

recover\_binlog\_info = 0

整合第1次增量备份的文件

[root@sql51 ~]# innobackupex --apply-log --redo-only /fullbak --incremental-dir=/new1dir

[root@sql51 ~]# cat /fullbak/xtrabackup\_checkpoints

backup\_type = log-applied

from\_lsn = 0

to\_lsn = 3042229 #序号范围变成了0-3042229(new1的结束)

last\_lsn = 3042238

compact = 0

recover\_binlog\_info = 0

整合第2次增量备份的文件

[root@sql51 ~]# innobackupex --apply-log --redo-only /fullbak --incremental-dir=/new2dir

[root@sql51 ~]# cat /fullbak/xtrabackup\_checkpoints

backup\_type = log-applied

from\_lsn = 0

to\_lsn = 3042571 #序号范围变成了0-3042571(new2的结束)

last\_lsn = 3042580 )

compact = 0

recover\_binlog\_info = 0

开始增量备份

[root@sql51 ~]# rm -rf /var/lib/mysql

[root@sql51 ~]# mkdi

mkdict mkdir

[root@sql51 ~]# mkdir /var/lib/mysql

[root@sql51 ~]# innobackupex --copy-back /fullbak/

[root@sql51 ~]# ls -l /var/lib/mysql

[root@sql51 ~]# chown -R mysql:mysql /var/lib/mysql

[root@sql51 ~]# systemctl restart mysqld

[root@sql51 ~]# mysql -uroot -p654321

#### 差异备份(需要整理)

352 innobackupex --user root --password 654321 /cybak --no-timestamp

353 history

354 324 innobackupex --user root --password 654321 --incremental /new1dir --incremental-basedir=/fullbak --no-timestamp

355 innobackupex --user root --password 654321 --incremental /new1dir --incremental-basedir=/fullbak --no-timestamp

356 innobackupex --user root --password 654321 --incremental /cy1dir --incremental-basedir=/cybak --no-timestamp

357 mysql -uroot -p654321

358 rm -rf /cy1dir

359 innobackupex --user root --password 654321 --incremental /cy1dir --incremental-basedir=/cybak --no-timestamp

360 mysql -uroot -p654321

361 innobackupex --user root --password 654321 --incremental /cy2dir --incremental-basedir=/cybak --no-timestamp

[root@sql51 ~]# cat /cybak/xtrabackup\_checkpoints

backup\_type = full-backuped

from\_lsn = 0

to\_lsn = 3042856

last\_lsn = 3042865

compact = 0

recover\_binlog\_info = 0

[root@sql51 ~]# cat /cy1dir/xtrabackup\_checkpoints

backup\_type = incremental

from\_lsn = 3042856

to\_lsn = 3043211

last\_lsn = 3043220

compact = 0

recover\_binlog\_info = 0

[root@sql51 ~]# cat /cy2dir/xtrabackup\_checkpoints

backup\_type = incremental

from\_lsn = 3042856

to\_lsn = 3043553

last\_lsn = 3043562

compact = 0

recover\_binlog\_info = 0

Cy2dir备份的文件包含了 Cy1dir的备份文件,,,

Cy2dir 的序号包含了 Cy1dir

### 3.4.3 从完全备份文件里恢复单个表的数据

#### 准备实验数据,新建表b并插入数据

mysql> create table db5.b(name char(10));

mysql> insert into db5.b values("hk");

mysql> insert into db5.b values("hk");

mysql> insert into db5.b values("tom");

mysql> insert into db5.b values("tom");

#### 表b插入数据后备份数据库

[root@sql51 ~]# innobackupex --user root -p654321 --databases="db5" /db5bak --no-timestamp

#### 备份好数据后删除表b

Database changed

mysql> show tables;

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

| Tables\_in\_db5 |

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

| a |

| b |

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

2 rows in set (0.00 sec)

mysql> drop table b;

Query OK, 0 rows affected (0.13 sec)

mysql> system ls /var/lib/mysql/db5

a.frm a.ibd db.opt

#### 恢复步骤1:创建相同结构的表

在需要恢复b表的数据库服务器内创建相同结构的表

mysql> create table db5.b(name char(10));

Query OK, 0 rows affected (0.23 sec)

mysql> system ls /var/lib/mysql/db5/b.\* #查看创建b表后产生的文件

/var/lib/mysql/db5/b.frm /var/lib/mysql/db5/b.ibd

b.ibd为表空间文件

b.frm 为表结构文件

#### 恢复步骤2:创建相同结构的表的表空间

mysql> alter table db5.b discard tablespace;

Query OK, 0 rows affected (0.10 sec)

mysql> system ls /var/lib/mysql/db5/b.\*

/var/lib/mysql/db5/b.frm #只剩frm表结构文件

#### 恢复步骤3:在备份数据里面导出表信息

[root@sql51 ~]# innobackupex --user root --password 654321 --apply-log --export /db5bak

[root@sql51 ~]# ls /db5bak/db5/

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

所有的库多了: .cfg .exp 两种文件

#### 恢复步骤4:把导出的信息文件拷贝到数据库目录下

mysql> system cp /db5bak/db5/b.{exp,cfg,ibd} /var/lib/mysql/db5/

#### 恢复步骤5:修改所有者和组用户为mysql

mysql> system chown mysql:mysql /var/lib/mysql/db5/b.\*

#### 恢复步骤6:导入表信息

mysql> alter table db5.b import tablespace;

Query OK, 0 rows affected, 1 warning (0.33 sec)

#### 恢复步骤7查看数据

mysql> select \* from db5.b;

mysql> system rm -rf /var/lib/mysql/db5/b.exp

mysql> system rm -rf /var/lib/mysql/db5/b.cfg

b.exp文件:导出库文件, #可有可无的文件,建议删除

B.cfg #可有可无的文件,建议删除

## 案例

1 数据备份与恢复

1.1 问题

本案例要求熟悉MySQL的备份与恢复，完成以下任务操作：

逻辑备份工具 mysqldump

使用mysql 恢复数据库

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：使用mysqldump进行逻辑备份

1）备份MySQL服务器上的所有库

将所有的库备份为mysql-all.sql文件：

[root@dbsvr1 ~]# mysqldump -u root -p --all-databases > /root/alldb.sql

Enter password: //验证口令

[root@dbsvr1 mysql]# file /root/alldb.sql //确认备份文件类型

/root/alldb.sql: UTF-8 Unicode English text, with very long lines

查看备份文件alldb.sql的部分内容：

[root@dbsvr1 ~]# grep -vE '^/|^-|^$' /root/alldb.sql | head -15

CREATE DATABASE /\*!32312 IF NOT EXISTS\*/ `home` /\*!40100 DEFAULT CHARACTER SET latin1 \*/;

USE `home`;

DROP TABLE IF EXISTS `biao01`;

CREATE TABLE `biao01` (

`id` int(2) NOT NULL,

`name` varchar(8) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

LOCK TABLES `biao01` WRITE;

UNLOCK TABLES;

DROP TABLE IF EXISTS `biao02`;

CREATE TABLE `biao02` (

`id` int(4) NOT NULL,

`name` varchar(8) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

.. ..

注意：若数据库都使用MyISAM存储引擎，可以采用冷备份的方式，直接复制对应的数据库目录即可；恢复时重新复制回来就行。

2）只备份指定的某一个库

将userdb库备份为userdb.sql文件：

[root@dbsvr1 ~]# mysqldump -u root -p userdb > userdb.sql

Enter password: //验证口令

查看备份文件userdb.sql的部分内容：

[root@dbsvr1 ~]# grep -vE '^/|^-|^$' /root/userdb.sql

DROP TABLE IF EXISTS `stu\_info`;

CREATE TABLE `stu\_info` (

`name` varchar(12) NOT NULL,

`gender` enum('boy','girl') DEFAULT 'boy',

`age` int(3) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

LOCK TABLES `stu\_info` WRITE;

.. ..

3）同时备份指定的多个库

同时备份mysql、userdb库，保存为mysql+userdb.sql文件：

[root@dbsvr1 ~]# mysqldump -u root -p -B mysql userdb > mysql+test+userdb.sql

Enter password: //验证口令

查看备份文件userdb.sql的部分内容：

[root@dbsvr1 ~]# grep '^CREATE DATA' /root/mysql+userdb.sql

CREATE DATABASE /\*!32312 IF NOT EXISTS\*/ `mysql` /\*!40100 DEFAULT CHARACTER SET latin1 \*/;

CREATE DATABASE /\*!32312 IF NOT EXISTS\*/ `userdb` /\*!40100 DEFAULT CHARACTER SET latin1 \*/;

步骤二：使用mysql命令从备份中恢复数据库、表

以恢复userdb库为例，可参考下列操作。通常不建议直接覆盖旧库，而是采用建立新库并导入逻辑备份的方式执行恢复，待新库正常后即可废弃或删除旧库。

1）创建名为userdb2的新库

mysql> CREATE DATABASE userdb2;

Query OK, 1 row affected (0.00 sec)

2）导入备份文件，在新库中重建表及数据

[root@dbsvr1 ~]# mysql -u root -p userdb2 < /root/userdb.sql

Enter password: //验证口令

3）确认新库正常，启用新库

mysql> USE userdb2; //切换到新库

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> SELECT sn,username,uid,gid,homedir //查询数据，确认可用

-> FROM userlist LIMIT 10;

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

| sn | username | uid | gid | homedir |

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

| 1 | root | 0 | 0 | /root |

| 2 | bin | 1 | 1 | /bin |

| 3 | daemon | 2 | 2 | /sbin |

| 4 | adm | 3 | 4 | /var/adm |

| 5 | lp | 4 | 7 | /var/spool/lpd |

| 6 | sync | 5 | 0 | /sbin |

| 7 | shutdown | 6 | 0 | /sbin |

| 8 | halt | 7 | 0 | /sbin |

| 9 | mail | 8 | 12 | /var/spool/mail |

| 10 | operator | 11 | 0 | /root |

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

10 rows in set (0.00 sec)

4）废弃或删除旧库

mysql> DROP DATABASE userdb;

Query OK, 2 rows affected (0.09 sec)

2 使用binlog日志

2.1 问题

利用binlog恢复库表，要求如下：

启用binlog日志

创建db1库tb1表，插入3条记录

删除tb1表中刚插入的3条记录

使用mysqlbinlog恢复删除的3条记录

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：启用binlog日志

1）调整/etc/my.cnf配置，并重启服务

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

[mysqld]

.. ..

log-bin-index=mysql-bin //启用二进制日志，并指定前缀

server\_id=1

binlog\_format=STATEMENT

//在Mysql5.7中，binlog日志格式默认为ROW，但它不记录sql语句上下文相关信息。需要将binlog日志格式修改为STATEMENT

.. ..

[root@dbsvr1 ~]# systemctl restart mysqld.service

2）确认binlog日志文件

新启用binlog后，每次启动MySQl服务都会新生成一份日志文件：

[root@dbsvr1 ~]# ls /var/lib/mysql/mysql-bin.\*

/var/lib/mysql/mysql-bin.000001 /var/lib/mysql/mysql-bin.index

其中mysql-bin.index文件记录了当前保持的二进制文件列表：

[root@dbsvr1 ~]# cat /var/lib/mysql/mysql-bin.index

./mysql-bin.000001

重启MySQL服务程序，或者执行SQL操作“FLUSH LOGS;”，会生成一份新的日志：

[root@dbsvr1 ~]# ls /var/lib/mysql/mysql-bin.\*

/var/lib/mysql/mysql-bin.000001 /var/lib/mysql/mysql-bin.index

/var/lib/mysql/mysql-bin.000002

[root@dbsvr1 ~]# cat /var/lib/mysql/mysql-bin.index

./mysql-bin.000001

./mysql-bin.000002

步骤二：利用binlog日志重做数据库操作

1）执行数据库表添加操作

创建db1·库tb1表，表结构自定义：

mysql> CREATE DATABASE db1;

Query OK, 1 row affected (0.05 sec)

mysql> USE db1;

Database changed

mysql> CREATE TABLE tb1(

-> id int(4) NOT NULL,name varchar(24)

-> );

Query OK, 0 rows affected (0.28 sec)

插入3条表记录：

mysql> INSERT INTO tb1 VALUES

-> (1,'Jack'),

-> (2,'Kenthy'),

-> (3,'Bob');

Query OK, 3 rows affected (0.12 sec)

Records: 3 Duplicates: 0 Warnings: 0

确认插入的表记录数据：

mysql> SELECT \* FROM tb1;

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

| id | name |

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

| 1 | Jack |

| 2 | Kenthy |

| 3 | Bob |

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

3 rows in set (0.00 sec)

2）删除前一步添加的3条表记录

执行删除所有表记录操作：

mysql> DELETE FROM tb1;

Query OK, 3 rows affected (0.09 sec)

确认删除结果：

mysql> SELECT \* FROM tb1;

Empty set (0.00 sec)

步骤三：通过binlog日志恢复表记录

binlog会记录所有的数据库、表更改操作，所以可在必要的时候重新执行以前做过的一部分数据操作，但对于启用binlog之前已经存在的库、表数据将不适用。

根据上述“恢复被删除的3条表记录”的需求，应通过mysqlbinlog工具查看相关日志文件，找到删除这些表记录的时间点，只要恢复此前的SQL操作（主要是插入那3条记录的操作）即可。

1）查看mysql-bin.000002日志内容

[root@dbsvr1 ~]# mysqlbinlog /var/lib/mysql/mysql-bin.000002

/\*!50530 SET @@SESSION.PSEUDO\_SLAVE\_MODE=1\*/;

/\*!50003 SET @OLD\_COMPLETION\_TYPE=@@COMPLETION\_TYPE,COMPLETION\_TYPE=0\*/;

DELIMITER /\*!\*/;

# at 4

#170412 12:05:32 server id 1 end\_log\_pos 123 CRC32 0x6d8c069c Start: binlog v 4, server v 5.7.17-log created 170412 12:05:32 at startup

# Warning: this binlog is either in use or was not closed properly.

ROLLBACK/\*!\*/;

BINLOG '

jKftWA8BAAAAdwAAAHsAAAABAAQANS43LjE3LWxvZwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAACMp+1YEzgNAAgAEgAEBAQEEgAAXwAEGggAAAAICAgCAAAACgoKKioAEjQA

AZwGjG0=

'/\*!\*/;

# at 123

#170412 12:05:32 server id 1 end\_log\_pos 154 CRC32 0x17f50164 Previous-GTIDs

# [empty]

# at 154

#170412 12:05:59 server id 1 end\_log\_pos 219 CRC32 0x4ba5a976 Anonymous\_GTID last\_committed=0 sequence\_number=1

SET @@SESSION.GTID\_NEXT= 'ANONYMOUS'/\*!\*/;

# at 219

#170412 12:05:59 server id 1 end\_log\_pos 310 CRC32 0x5b66ae13 Query thread\_id=3 exec\_time=0 error\_code=0

SET TIMESTAMP=1491969959/\*!\*/;

SET @@session.pseudo\_thread\_id=3/\*!\*/;

SET @@session.foreign\_key\_checks=1, @@session.sql\_auto\_is\_null=0, @@session.unique\_checks=1, @@session.autocommit=1/\*!\*/;

SET @@session.sql\_mode=1436549152/\*!\*/;

SET @@session.auto\_increment\_increment=1, @@session.auto\_increment\_offset=1/\*!\*/;

/\*!\C utf8 \*//\*!\*/;

SET @@session.character\_set\_client=33,@@session.collation\_connection=33,@@session.collation\_server=8/\*!\*/;

SET @@session.lc\_time\_names=0/\*!\*/;

SET @@session.collation\_database=DEFAULT/\*!\*/;

CREATE DATABASE db1

/\*!\*/;

# at 310

#170412 12:06:23 server id 1 end\_log\_pos 375 CRC32 0x2967cc28 Anonymous\_GTID last\_committed=1 sequence\_number=2

SET @@SESSION.GTID\_NEXT= 'ANONYMOUS'/\*!\*/;

# at 375

#170412 12:06:23 server id 1 end\_log\_pos 502 CRC32 0x5de09aae Query thread\_id=3 exec\_time=0 error\_code=0

use `db1`/\*!\*/;

SET TIMESTAMP=1491969983/\*!\*/;

CREATE TABLE tb1(

id int(4) NOT NULL,name varchar(24)

)

/\*!\*/;

# at 502

#170412 12:06:55 server id 1 end\_log\_pos 567 CRC32 0x0b8cd418 Anonymous\_GTID last\_committed=2 sequence\_number=3

SET @@SESSION.GTID\_NEXT= 'ANONYMOUS'/\*!\*/;

# at 567

#170412 12:06:55 server id 1 end\_log\_pos 644 CRC32 0x7e8f2fa0 Query thread\_id=3 exec\_time=0 error\_code=0

SET TIMESTAMP=1491970015/\*!\*/;

BEGIN

/\*!\*/;

# at 644

#170412 12:06:55 server id 1 end\_log\_pos 772 CRC32 0x4e3f728e Query thread\_id=3 exec\_time=0 error\_code=0 //插入表记录的起始时间点

SET TIMESTAMP=1491970015/\*!\*/;

INSERT INTO tb1 VALUES(1,'Jack'),(2,'Kenthy'), (3,'Bob')

/\*!\*/;

# at 772

#170412 12:06:55 server id 1 end\_log\_pos 803 CRC32 0x6138b21f Xid = 10

//确认事务的时间点

COMMIT/\*!\*/;

# at 803

#170412 12:07:24 server id 1 end\_log\_pos 868 CRC32 0xbef3f472 Anonymous\_GTID last\_committed=3 sequence\_number=4

SET @@SESSION.GTID\_NEXT= 'ANONYMOUS'/\*!\*/;

# at 868

#170412 12:07:24 server id 1 end\_log\_pos 945 CRC32 0x5684e92c Query thread\_id=3 exec\_time=0 error\_code=0

SET TIMESTAMP=1491970044/\*!\*/;

BEGIN

/\*!\*/;

# at 945

#170412 12:07:24 server id 1 end\_log\_pos 1032 CRC32 0x4c1c75fc Query thread\_id=3 exec\_time=0 error\_code=0 //删除表记录的时间点

SET TIMESTAMP=1491970044/\*!\*/;

DELETE FROM tb1

/\*!\*/;

# at 1032

#170412 12:07:24 server id 1 end\_log\_pos 1063 CRC32 0xccf549b2 Xid = 12

COMMIT/\*!\*/;

SET @@SESSION.GTID\_NEXT= 'AUTOMATIC' /\* added by mysqlbinlog \*/ /\*!\*/;

DELIMITER ;

# End of log file

/\*!50003 SET COMPLETION\_TYPE=@OLD\_COMPLETION\_TYPE\*/;

/\*!50530 SET @@SESSION.PSEUDO\_SLAVE\_MODE=0\*/;

2） 执行指定Pos节点范围内的sql命令恢复数据

根据上述日志分析，只要恢复从2014.01.12 20:12:14到2014.01.12 20:13:50之间的操作即可。可通过mysqlbinlog指定时间范围输出，结合管道交给msyql命令执行导入重做：

[root@dbsvr1 ~]# mysqlbinlog \

--start-datetime="2017-04-12 12:06:55" \

--stop-datetime="2017-04-12 12:07:23" \

/var/lib/mysql/mysql-bin.000002 | mysql -u root -p

Enter password: //验证口令

3）确认恢复结果

mysql> SELECT \* FROM db1.tb1;

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

| id | name |

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

| 1 | Jack |

| 2 | Kenthy |

| 3 | Bob |

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

3 rows in set (0.00 sec)

3 innobackupex备份工具

3.1 问题

安装percona软件包

innobackupex完整备份、增量备份操作。

恢复数据

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：安装XtraBackup软件包

1）了解软件包描述信息

[root@dbsvr1 pub]# rpm -qpi percona-xtrabackup-24-2.4.6-2.el7.x86\_64.rpm

Name : percona-xtrabackup-24

Version : 2.4.6

Release : 2.el7

Architecture: x86\_64

Install Date: (not installed)

Group : Applications/Databases

Size : 32416340

License : GPLv2

Signature : DSA/SHA1, 2017年02月27日 星期一 20时28分17秒, Key ID 1c4cbdcdcd2efd2a

Source RPM : percona-xtrabackup-24-2.4.6-2.el7.src.rpm

Build Date : 2017年02月27日 星期一 20时27分21秒

Build Host : vps-centos7-x64-01.ci.percona.com

Relocations : (not relocatable)

URL : http://www.percona.com/software/percona-xtrabackup

Summary : XtraBackup online backup for MySQL / InnoDB

Description :

Percona XtraBackup is OpenSource online (non-blockable) backup solution for InnoDB and XtraDB engines

2）安装依赖包perl-DBD-MySQL perl-Digest-MD5 libev

使用RHEL 7自带的即可，yum方式安装：

[root@dbsvr1 pub]# yum -y install perl-DBD-MySQL perl-Digest-MD5

libev使用网上找的rpm包 libev-4.15-1.el6.rf.x86\_64.rpm //该包由讲师提供

[root@dbsvr1 pub]#rpm –ivh libev-4.15-1.el6.rf.x86\_64.rpm

如果未安装这些依赖包，则直接安装percona-xtrabackup时会报错：

代码

3）安装percona-xtrabackup

[root@dbsvr1 pub]#rpm -ivh percona-xtrabackup-\*.rpm

警告：percona-xtrabackup-24-2.4.6-2.el7.x86\_64.rpm: 头V4 DSA/SHA1 Signature, 密钥 ID cd2efd2a: NOKEY

准备中... ################################# [100%]

正在升级/安装...

1:percona-xtrabackup-24-2.4.6-2.el7################################# [ 33%]

2:percona-xtrabackup-test-24-2.4.6-################################# [ 67%]

3:percona-xtrabackup-24-debuginfo-2################################# [100%]

4）确认安装的主要程序/脚本

[root@dbsvr1 pub]# rpm -ql percona-xtrabackup-24-2.4.6-2.el7.x86\_64

/usr/bin/innobackupex

/usr/bin/xbcloud

/usr/bin/xbcloud\_osenv

/usr/bin/xbcrypt

/usr/bin/xbstream

/usr/bin/xtrabackup

/usr/share/doc/percona-xtrabackup-24-2.4.6

/usr/share/doc/percona-xtrabackup-24-2.4.6/COPYING

/usr/share/man/man1/innobackupex.1.gz

/usr/share/man/man1/xbcrypt.1.gz

/usr/share/man/man1/xbstream.1.gz

/usr/share/man/man1/xtrabackup.1.gz

步骤二：innobackupex完整备份、增量备份操作

--host 主机名

--port 3306

--user 用户名

--password 密码

--databases="库名"

--databases="库1 库2"

--databases="库.表"

--no-timestamp 不用日期命名备份文件存储的子目录，使用备份的数据库名做备份目录名

--no-timestmap 不使用日期命名备份目录名

1）做一个完整备份

默认情况下，备份文件存储的子目录会用日期命名，

innobackupex作为客户端工具，以mysql协议连入mysqld，将数据备份到/backup文件夹：

[root@dbsvr1 ~]# innobackupex --user=root --password=1234567 /backup/mysql –no-timestamp

170425 11:05:44 innobackupex: Starting the backup operation

IMPORTANT: Please check that the backup run completes successfully.

At the end of a successful backup run innobackupex

prints "completed OK!".

Unrecognized character \x01; marked by <-- HERE after <-- HERE near column 1 at - line 1374.

170425 11:05:45 Connecting to MySQL server host: localhost, user: root, password: set, port: not set, socket: not set

Using server version 5.7.17

innobackupex version 2.4.6 based on MySQL server 5.7.13 Linux (x86\_64) (revision id: 8ec05b7)

xtrabackup: uses posix\_fadvise().

xtrabackup: cd to /var/lib/mysql

xtrabackup: open files limit requested 0, set to 1024

xtrabackup: using the following InnoDB configuration:

xtrabackup: innodb\_data\_home\_dir = .

xtrabackup: innodb\_data\_file\_path = ibdata1:12M:autoextend

xtrabackup: innodb\_log\_group\_home\_dir = ./

xtrabackup: innodb\_log\_files\_in\_group = 2

xtrabackup: innodb\_log\_file\_size = 50331648

InnoDB: Number of pools: 1

170425 11:05:45 >> log scanned up to (2543893)

xtrabackup: Generating a list of tablespaces

InnoDB: Allocated tablespace ID 2 for mysql/plugin, old maximum was 0

170425 11:05:45 [01] Copying ./ibdata1 to /backup/ibdata1

170425 11:05:45 [01] ...done

170425 11:05:46 [01] Copying ./mysql/plugin.ibd to /backup/mysql/plugin.ibd

170425 11:05:46 [01] ...done

170425 11:05:46 [01] Copying ./mysql/servers.ibd to /backup/mysql/servers.ibd

170425 11:05:46 [01] ...done

170425 11:05:46 [01] Copying ./mysql/help\_topic.ibd to /backup/mysql/help\_topic.ibd

170425 11:05:46 [01] ...done

170425 11:05:46 >> log scanned up to (2543893)

.. ..

170425 11:06:00 [01] Copying ./sys/x@0024waits\_global\_by\_latency.frm to /backup/sys/x@0024waits\_global\_by\_latency.frm

170425 11:06:00 [01] ...done

170425 11:06:00 [01] Copying ./sys/session\_ssl\_status.frm to /backup/sys/session\_ssl\_status.frm

170425 11:06:00 [01] ...done

170425 11:06:00 [01] Copying ./db1/db.opt to /backup/db1/db.opt

170425 11:06:00 [01] ...done

170425 11:06:00 [01] Copying ./db1/tb1.frm to /backup/db1/tb1.frm

170425 11:06:00 [01] ...done

170425 11:06:00 Finished backing up non-InnoDB tables and files

170425 11:06:00 Executing FLUSH NO\_WRITE\_TO\_BINLOG ENGINE LOGS...

xtrabackup: The latest check point (for incremental): '2543884'

xtrabackup: Stopping log copying thread.

.170425 11:06:00 >> log scanned up to (2543893)

170425 11:06:00 Executing UNLOCK TABLES

170425 11:06:00 All tables unlocked

170425 11:06:00 [00] Copying ib\_buffer\_pool to /backup/ib\_buffer\_pool

170425 11:06:00 [00] ...done

170425 11:06:00 Backup created in directory '/backup/'

170425 11:06:00 [00] Writing backup-my.cnf

170425 11:06:00 [00] ...done

170425 11:06:00 [00] Writing xtrabackup\_info

170425 11:06:00 [00] ...done

xtrabackup: Transaction log of lsn (2543884) to (2543893) was copied.

170425 11:06:01 completed OK

确认备份好的文件数据：

[root@dbsvr1 ~]#ls /backup/

backup-my.cnf ib\_buffer\_pool mysql sys xtrabackup\_info

db1 ibdata1 performance\_schema xtrabackup\_checkpoints xtrabackup\_logfile

2）做一个增量备份（基于前一步的完整备份）

随意做一些新增或更改库表的操作，比如在db1库中新建一个mytb的表：

mysql> USE db1;

Database changed

mysql> CREATE TABLE mytb(id int(4), name varchar(24));

Query OK, 0 rows affected (0.38 sec)

mysql> INSERT INTO tb1 VALUES

-> (1,'bon'),

-> (2,'bo'),

Query OK, 2 rows affected (0.12 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT \* FROM tb1;

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

| id | name |

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

| 1 | bob |

| 2 | bo |

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

2 rows in set (0.00 sec)

以前一次保存到/backup的完整备份为基础，做一个增量备份，保存到/incr01/，指定增量备份参照的基本目录（完整备份目录）需要用到选项--incremental-basedir。相关操作如下：

[root@dbsvr1 ~]# innobackupex --user=root --password=12345678 --incremental /incr01 --incremental-basedir=/backup/ --no-timestamp

170425 11:30:14 innobackupex: Starting the backup operation

IMPORTANT: Please check that the backup run completes successfully.

At the end of a successful backup run innobackupex

prints "completed OK!".

Unrecognized character \x01; marked by <-- HERE after <-- HERE near column 1 at - line 1374.

170425 11:30:14 Connecting to MySQL server host: localhost, user: root, password: set, port: not set, socket: not set

Using server version 5.7.17

innobackupex version 2.4.6 based on MySQL server 5.7.13 Linux (x86\_64) (revision id: 8ec05b7)

incremental backup from 2543884 is enabled.

xtrabackup: uses posix\_fadvise().

xtrabackup: cd to /var/lib/mysql

xtrabackup: open files limit requested 0, set to 1024

xtrabackup: using the following InnoDB configuration:

xtrabackup: innodb\_data\_home\_dir = .

xtrabackup: innodb\_data\_file\_path = ibdata1:12M:autoextend

xtrabackup: innodb\_log\_group\_home\_dir = ./

xtrabackup: innodb\_log\_files\_in\_group = 2

xtrabackup: innodb\_log\_file\_size = 50331648

InnoDB: Number of pools: 1

170425 11:30:14 >> log scanned up to (2549933)

xtrabackup: Generating a list of tablespaces

InnoDB: Allocated tablespace ID 2 for mysql/plugin, old maximum was 0

xtrabackup: using the full scan for incremental backup

170425 11:30:15 [01] Copying ./ibdata1 to /incr01/ibdata1.delta

170425 11:30:15 [01] ...done

170425 11:30:15 >> log scanned up to (2549933)

170425 11:30:15 [01] Copying ./mysql/plugin.ibd to /incr01/mysql/plugin.ibd.delta

170425 11:30:15 [01] ...done

... ...

170425 11:30:35 Executing UNLOCK TABLES

170425 11:30:35 All tables unlocked

170425 11:30:35 [00] Copying ib\_buffer\_pool to /incr01/ib\_buffer\_pool

170425 11:30:35 [00] ...done

170425 11:30:35 Backup created in directory '/incr01/'

170425 11:30:35 [00] Writing backup-my.cnf

170425 11:30:35 [00] ...done

170425 11:30:35 [00] Writing xtrabackup\_info

170425 11:30:35 [00] ...done

xtrabackup: Transaction log of lsn (2549924) to (2549933) was copied.

170425 11:30:35 completed OK!

确认备份好的文件数据：

[root@dbsvr1 ~]# ls /incr01/

backup-my.cnf ib\_buffer\_pool ibdata1.meta performance\_schema xtrabackup\_checkpoints xtrabackup\_logfile

db1 ibdata1.delta mysql sys

对比完整备份、增量备份的大小：

[root@dbsvr1 ~]# du -sh /backup/ /incr01/

142M /backup/ //完整备份的大小

3.5M /incr01/ //增量备份的大小

步骤三：恢复数据

通过XtraBackup工具备份的数据库目录，若要恢复到另一个MySQL服务器，需要先做一个“--apply-log --redo-only ”的准备操作。

1）准备恢复“完整备份”

完成准备以后，最终/backup可用来重建MySQL服务器。这种情况下，需要先做一个“--apply-log --redo-only ”的准备操作，以确保数据一致性：

[root@dbsvr1 ~]#innobackupex --user=root --password=12345678 --apply-log --redo-only /backup/

170425 11:42:19 innobackupex: Starting the apply-log operation

IMPORTANT: Please check that the apply-log run completes successfully.

At the end of a successful apply-log run innobackupex

prints "completed OK!".

innobackupex version 2.4.6 based on MySQL server 5.7.13 Linux (x86\_64) (revision id: 8ec05b7)

xtrabackup: cd to /backup/

xtrabackup: This target seems to be already prepared.

InnoDB: Number of pools: 1

xtrabackup: notice: xtrabackup\_logfile was already used to '--prepare'.

xtrabackup: using the following InnoDB configuration for recovery:

xtrabackup: innodb\_data\_home\_dir = .

xtrabackup: innodb\_data\_file\_path = ibdata1:12M:autoextend

xtrabackup: innodb\_log\_group\_home\_dir = .

xtrabackup: innodb\_log\_files\_in\_group = 2

xtrabackup: innodb\_log\_file\_size = 50331648

xtrabackup: using the following InnoDB configuration for recovery:

xtrabackup: innodb\_data\_home\_dir = .

xtrabackup: innodb\_data\_file\_path = ibdata1:12M:autoextend

xtrabackup: innodb\_log\_group\_home\_dir = .

xtrabackup: innodb\_log\_files\_in\_group = 2

xtrabackup: innodb\_log\_file\_size = 50331648

xtrabackup: Starting InnoDB instance for recovery.

xtrabackup: Using 104857600 bytes for buffer pool (set by --use-memory parameter)

InnoDB: PUNCH HOLE support available

InnoDB: Mutexes and rw\_locks use GCC atomic builtins

InnoDB: Uses event mutexes

InnoDB: GCC builtin \_\_atomic\_thread\_fence() is used for memory barrier

InnoDB: Compressed tables use zlib 1.2.7

InnoDB: Number of pools: 1

InnoDB: Not using CPU crc32 instructions

InnoDB: Initializing buffer pool, total size = 100M, instances = 1, chunk size = 100M

InnoDB: Completed initialization of buffer pool

InnoDB: page\_cleaner coordinator priority: -20

InnoDB: Highest supported file format is Barracuda.

xtrabackup: starting shutdown with innodb\_fast\_shutdown = 1

InnoDB: Starting shutdown...

InnoDB: Shutdown completed; log sequence number 2544177

InnoDB: Number of pools: 1

170425 11:42:20 completed OK!

准备恢复“增量备份”

[root@dbsvr1 ~]#innobackupex --user=root --password=12345678 --apply-log --redo-only /backup/ --incremental-dir=/incr01

170425 11:42:55 innobackupex: Starting the apply-log operation

IMPORTANT: Please check that the apply-log run completes successfully.

At the end of a successful apply-log run innobackupex

prints "completed OK!".

innobackupex version 2.4.6 based on MySQL server 5.7.13 Linux (x86\_64) (revision id: 8ec05b7)

incremental backup from 2543884 is enabled.

xtrabackup: cd to /backup/

xtrabackup: This target seems to be already prepared with --apply-log-only.

InnoDB: Number of pools: 1

xtrabackup: xtrabackup\_logfile detected: size=8388608, start\_lsn=(2549924)

xtrabackup: using the following InnoDB configuration for recovery:

xtrabackup: innodb\_data\_home\_dir = .

xtrabackup: innodb\_data\_file\_path = ibdata1:12M:autoextend

xtrabackup: innodb\_log\_group\_home\_dir = /incr01/

xtrabackup: innodb\_log\_files\_in\_group = 1

xtrabackup: innodb\_log\_file\_size = 8388608

xtrabackup: Generating a list of tablespaces

InnoDB: Allocated tablespace ID 2 for mysql/plugin, old maximum was 0

xtrabackup: page size for /incr01//ibdata1.delta is 16384 bytes

Applying /incr01//ibdata1.delta to ./ibdata1...

... ...

170425 11:43:09 [01] Copying /incr01/performance\_schema/global\_status.frm to ./performance\_schema/global\_status.frm

170425 11:43:09 [01] ...done

170425 11:43:09 [01] Copying /incr01/performance\_schema/session\_status.frm to ./performance\_schema/session\_status.frm

170425 11:43:09 [01] ...done

170425 11:43:09 [00] Copying /incr01//xtrabackup\_info to ./xtrabackup\_info

170425 11:43:09 [00] ...done

170425 11:43:10 completed OK!

2）关闭mysql服务，并将/var/lib/mysql/下的文件删除，假设数据被删除。

[root@dbsvr1 ~]#systemctl stop mysqld

[root@dbsvr1 ~]#rm -rf /var/lib/mysql

3）恢复“完整备份+增量备份”

完成准备以后，最终仍然是/backup用来重建MySQL服务器，但这种情况下需提前合并相关增量备份的数据

[root@dbsvr1 ~]# innobackupex --user=root --password=12345678 --copy-back /backup/

... ...

170425 11:51:39 [01] Copying ./performance\_schema/global\_status.frm to /var/lib/mysql/performance\_schema/glo.frm

170425 11:51:39 [01] ...done

170425 11:51:39 [01] Copying ./performance\_schema/session\_status.frm to /var/lib/mysql/performance\_schema/seus.frm

170425 11:51:39 [01] ...done

170425 11:51:39 [01] Copying ./ib\_buffer\_pool to /var/lib/mysql/ib\_buffer\_pool

170425 11:51:39 [01] ...done

170425 11:51:39 [01] Copying ./ibtmp1 to /var/lib/mysql/ibtmp1

170425 11:51:39 [01] ...done

170425 11:51:39 [01] Copying ./xtrabackup\_info to /var/lib/mysql/xtrabackup\_info

170425 11:51:39 [01] ...done

170425 11:51:39 completed OK!

4）修改/var/lib/mysql/下文件属主与属组,查看数据：

恢复后，/var/lib/mysql下文件属组与属主皆为root，需要更改为mysql

[root@dbsvr1 ~]#chown -R mysql:mysql /var/lib/mysql

[root@dbsvr1 ~]#systemctl start mysqld.service

[root@dbsvr1 ~]#mysql -uroot -p12345678 -e "select \* from db1.tb1"

mysql: [Warning] Using a password on the command line interface can be insecure.

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

| id | name |

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

| 1 | bob |

| 2 | bo |

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

# NSD DBA2 DAY01

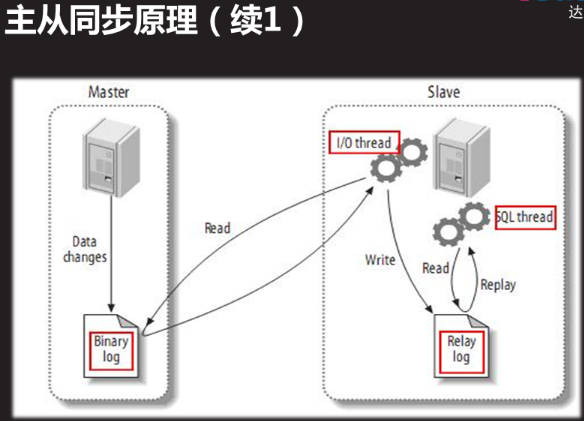
# 一 MySQL主从同步

1.1 主从同步介绍

把多台数据库服务器架构起主从关系.主服务器对外访问,从服务器自动到主服务器上同步数据

对指定库的异地同步

1.2主从同步原理



Master (主),记录数据更改操作

Dump线程:告诉从库自己库上有数据变化

启用binlog日志(二进制日志)

Slave(从)

IO线程:告诉主服务器,binlog的偏移量.让主服务器从偏移量开始传送数据给从服务器,同时将获取的数据,放到到中继日志里面

SQL线程:执行中继日志里面的命令

## 1.3配置主从同步

### 1.3.1配置主库

#### 配置主配置文件启用日志

[root@sql51 /]# vim /etc/my.cnf

[mysqld]

server\_id=51

log\_bin=master51

[root@sql51 /]# systemctl restart mysqld

#### 查看产生的binlog日志文件:

[root@sql51 /]# ls /var/lib/mysql/master51.\*

master51.000001 master51.index

mysql> show master status; #查看日志信息

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

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

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

| plj.000005 | 441 | | | |

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

1 row in set (0.00 sec)

#### 用户连接授权

参考:grant 权限列表 on 数据库名 to 用户名@”客户端地址” identified by “密码”

#注意密码的复杂度要求

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

Query OK, 0 rows affected, 1 warning (0.03 sec)

### 1.3.2配置从库

#### 配置主配置文件

[root@sql52 ~]# vim /etc/my.cnf

[mysqld]

server\_id=52

[root@sql52 ~]# systemctl restart mysqld

#### 指定主库信息

mysql> change master to

master\_host="192.168.4.51", #主服务器IP地址

master\_user="repluser", #主服务器授权的用户

master\_password="123456", #授权用户的密码

master\_log\_file="master51.000001", #日志文件

master\_log\_pos=441; #偏移位置

1. master信息会自动保存到 /var/lib/mysql/master.info
2. 更改master信息时,应先stop slave.

修改主库信息(需要验证)

mysql> stop slave

mysql>change master to 选项=值;

mysql>Stat slave

#### 启动slave程序

mysql> start slave;

Query OK, 0 rows affected (0.01 sec)

#### 查看slave从服务器情况

mysql> show slave status\G;

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

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

Master\_Host: 192.168.4.51

Master\_User: repluser

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: plj.000005

Read\_Master\_Log\_Pos: 441

Relay\_Log\_File: sql52-relay-bin.000002

Relay\_Log\_Pos: 601

Relay\_Master\_Log\_File: plj.000005

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

IO线程:复制主库binlog日志里面的命令到中继日志

SQL线程:执行中继日志里面的命令

#### 从库产生的新文件:

主库信息文件:master.info master.info

连接主服务器信息:ip 日志文件,偏移位置.指定主用户密码等连接信息

中继日志信息文件:relay-log.info relay-log.info

中继日志文件:主机名-relay-bin.XXX sql52-relay-bin.000002

中继日志索引文件:主机名-relay-bin.index sql52-relay-bin.index

#### 问题处理

查看报错信息

mysql> show slave status\G;

Slave\_IO\_Running: Yes #如果此处不是YES,下面 Last\_IO\_Error: 会有错误信息

Slave\_SQL\_Running: Yes #如果此处不是YES,下面Last\_SQL\_Error:会有错误信息

Last\_IO\_Errno: 0

Last\_IO\_Error:

Last\_SQL\_Errno: 0

Last\_SQL\_Error:

mysql> stop slave

mysql>change master to 选项=值;

mysql>Stat slave

恢复从库为独立普通库,重新指定主库信息

[root@sql52 /]# systemctl stop mysqld

[root@sql52 /]# cd /var/lib/mysql

rm -rf master.info relay-log.info

rm -rf sql52-relay-bin.\*

mysql> change master to

master\_host="192.168.4.51", #主服务器IP地址

master\_user="repluser", #主服务器授权的用户

master\_password="123456", #授权用户的密码

master\_log\_file="master51.000001", #日志文件

master\_log\_pos=441; #偏移位置

### 1.3.3客户端测试主从同步配置

1在主库授权访问数据的连接用户

mysql> create database db7;

Query OK, 1 row affected (0.04 sec)

mysql> create table db7.t1(id int);

Query OK, 0 rows affected (0.23 sec)

mysql> grant select,insert on db7.\* to yaya@"%" identified by "123456";

Query OK, 0 rows affected, 1 warning (0.03 sec)

2.在客户端使用授权用户连接主库

[root@client-50 ~]# mysql -h192.168.4.51 -uyaya -p123456;

mysql> insert into db7.t1 values(88),(99);

Query OK, 2 rows affected (0.06 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql> select \* from db7.t1;

+------+

| id |

+------+

| 88 |

| 99 |

+------+

2 rows in set (0.01 sec)

3.在从库上用主库授权用户登录也可以查看到一样的数据

[root@sql52 ~]# mysql -uyaya -p123456

mysql> select \* from db7.t1;

+------+

| id |

+------+

| 88 |

| 99 |

+------+

2 rows in set (0.01 sec)

### 1.3.3常用配置选项

#### 主库配置选项

#### 从库配置选项

## 1.4主从同步结构模式

### 1.4.1主从同步结构模式

基本应用:

单项复制:主-->从

扩展应用:

链式复制:主-->从-->从 主从从

互为主从:主<--->主 主主

一主多从:从<--主-->从 一主多从

### 1.4.2配置一主多从结构

#### 第一步:准备环境

继续1.3配置主从同步环境做51为主,52为从,现添加一个53为从

51主服务器备份数据库,传送给53从服务器还原

[root@sql51 ~]# mysqldump -uroot -p123456 -B db7 >/root/sql51-db7.sql;

mysqldump: [Warning] Using a password on the command line interface can be insecure.

[root@sql51 ~]# scp sql51-db7.sql 192.168.4.53:/root

root@192.168.4.53's password:

sql51-db7.sql 100% 1904 2.0MB/s 00:00

[root@sql51 ~]# mysql -uroot -p123456

mysql> show master status;

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

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

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

| plj.000006 | 1032 | | | |

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

1 row in set (0.00 sec)

#### 第二步:配置从服务器

[root@sql53 ~]# vim /etc/my.cnf

[mysqld]

server\_id=53

[root@sql53 ~]#systemctl restart mysqld

[root@sql53 ~]# mysql -uroot -p123456 <sql51-db7.sql#还原51拷贝来的数据库

mysql: [Warning] Using a password on the command line interface can be insecure.

[root@sql53 ~]# mysql -uroot -p123456

mysql> show databases;

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

| Database |

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

| information\_schema |

| db7 |

| mysql |

| performance\_schema |

| sys |

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

5 rows in set (0.01 sec)

mysql> exit

Bye

mysql> change master to

-> master\_host="192.168.4.51",

-> master\_user="repluser",

-> master\_password="123456",

-> master\_log\_file="plj.000006", #第一步主服务器查询而来

-> master\_log\_pos=1032; #第一步主服务器查询而来

Query OK, 0 rows affected, 2 warnings (0.02 sec)

mysql> start slave ;

Query OK, 0 rows affected (0.00 sec)

mysql> show slave status\G;

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

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

Master\_Host: 192.168.4.51

Master\_User: repluser

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: plj.000006

Read\_Master\_Log\_Pos: 1032

Relay\_Log\_File: sql53-relay-bin.000004

Relay\_Log\_Pos: 314

Relay\_Master\_Log\_File: plj.000006

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

Replicate\_Do\_DB:

### 1.4.2配置主>从>从结构

#### 第一步:环境准备

在上面环境基础上:51为52的主,51也是为53的从

使得53为51的从服务器器.同时53又为54的主服务器,

53要开启binlog日志, 级联复制log\_slave\_updates

[root@sql53 ~]# vim /etc/my.cnf

[mysqld]

server\_id=53

log\_bin=53binlog

log\_slave\_updates #开启级联复制功能

[root@sql53 ~]# systemctl restart mysqld

[root@sql53 ~]# ls /var/lib/mysql/53binlog\*

/var/lib/mysql/53binlog.000001 /var/lib/mysql/53binlog.index

[root@sql53 ~]# mysqldump -uroot -p123456 -B db7 >/root/53-db7.sql

mysqldump: [Warning] Using a password on the command line interface can be insecure.

[root@sql53 ~]# scp 53-db7.sql 192.168.4.54:/root

[root@sql53 ~]# mysql -uroot -p123456

mysql> show master status;

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

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

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

| 53binlog.000001 | 154 | | | |

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

1 row in set (0.00 sec)

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

Query OK, 0 rows affected, 1 warning (0.03 sec)

#### 第二步:从服务器

[root@sql54 ~]# vim /etc/my.cnf

[mysqld]

server\_id=54

[root@sql54 ~]# mysql -uroot -p123456 <53-db7.sql

mysql: [Warning] Using a password on the command line interface can be insecure.

[root@sql54 ~]# mysql -uroot -p123456

mysql> show databases;

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

| Database |

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

| information\_schema |

| db7 |

| mysql |

| performance\_schema |

| sys |

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

5 rows in set (0.00 sec)

mysql> change master to

-> master\_host="192.168.4.53",

-> master\_user="repluser",

-> master\_password="123456",

-> master\_log\_file="53binlog.000001",

-> master\_log\_pos=154;

Query OK, 0 rows affected, 2 warnings (0.33 sec)

mysql> show slave status\G;

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

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

Master\_Host: 192.168.4.53

Master\_User: repluser

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: 53binlog.000002

Read\_Master\_Log\_Pos: 441

Relay\_Log\_File: sql54-relay-bin.000003

Relay\_Log\_Pos: 652

Relay\_Master\_Log\_File: 53binlog.000002sh

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

#### 第三步:测试

在50客户上连接51数据库,插入数据,在51,52,53,54,上查看

[root@client-50 ~]# mysql -h192.168.4.51 -uyaya -p123456;

mysql> insert into db7.t1 values(11);

Query OK, 1 row affected (0.05 sec)

mysql> insert into db7.t1 values(22);

Query OK, 1 row affected (0.07 sec)

50,51,52,53,54上查看db7.t7表数据都能成功

mysql> select \* from db7.t1;

+------+

| id |

+------+

| 88 |

| 99 |

| 11 |

| 22 |

+------+

4 rows in set (0.00 sec)

### 1.4.2配置主主结构

55,56互为主从服务器

#### 第一步:55配置

[root@sql55 ~]# vim /etc/my.cnf

[mysqld]

server\_id=55

log\_bin=55binlog

[root@sql55 ~]# systemctl restart mysqld

[root@sql55 ~]# mysql -uroot -p123456

mysql> show databases;

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

| Database |

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

| information\_schema |

| mysql |

| performance\_schema |

| sys |

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

4 rows in set (0.00 sec)

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

Query OK, 0 rows affected, 1 warning (0.04 sec)

mysql> show master status;

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

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

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

| 55binlog.000001 | 441 | | | |

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

1 row in set (0.00 sec)

#### 第二步:56配置

[root@sql56 ~]# vim /etc/my.cnf

[mysqld]

server\_id=56

log\_bin=56binlog

[root@sql56 ~]# systemctl restart mysqld

[root@sql56 ~]# mysql -uroot -p123456;

mysql> show databases;

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

| Database |

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

| information\_schema |

| mysql |

| performance\_schema |

| sys |

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

4 rows in set (0.02 sec)

mysql> change master to

-> master\_host="192.168.4.55",

-> master\_user="repluser",

-> master\_password="123456",

-> master\_log\_file="55binlog.000001",

-> master\_log\_pos=441;

Query OK, 0 rows affected, 2 warnings (0.31 sec)

mysql> start slave;

Query OK, 0 rows affected (0.02 sec)

mysql> show slave status\G;

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

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

Master\_Host: 192.168.4.55

Master\_User: repluser

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: 55binlog.000047

Read\_Master\_Log\_Pos: 154

Relay\_Log\_File: sql56-relay-bin.000050

Relay\_Log\_Pos: 365

Relay\_Master\_Log\_File: 55binlog.000047

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

Replicate\_Do\_DB:

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

Query OK, 0 rows affected, 1 warning (0.04 sec)

mysql> show master status;

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

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

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

| 56binlog.000003 | 441 | | | |

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

1 row in set (0.00 sec)

#### 第三步:55配置

mysql> change master to

-> master\_host="192.168.4.56",

-> master\_user="repluser",

-> master\_password="123456",

-> master\_log\_file="56binlog.000003",

-> master\_log\_pos=441;

Query OK, 0 rows affected, 2 warnings (0.28 sec)

mysql> show slave status\G;

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

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

Master\_Host: 192.168.4.56

Master\_User: repluser

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: 56binlog.000003

Read\_Master\_Log\_Pos: 441

Relay\_Log\_File: sql55-relay-bin.000002

Relay\_Log\_Pos: 319

Relay\_Master\_Log\_File: 56binlog.000003

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

Replicate\_Do\_DB:

#### 第四步:测试

55创建数据库,56上能查看到

56上创建数据,55上能查看到

#### 出现过的错误:

mysql> show slave status\G;

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

Slave\_IO\_State:

Master\_Host: 192.168.4.55

Master\_User: repluser

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: 55binlog.000001

Read\_Master\_Log\_Pos: 441

Relay\_Log\_File: sql56-relay-bin.000002

Relay\_Log\_Pos: 4

Relay\_Master\_Log\_File: 55binlog.000001

Slave\_IO\_Running: No

Slave\_SQL\_Running: Yes

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

报错信息:从机I/O线程停止，因为主服务器和从服务器具有相同的mysql服务器uuid；这些uuid必须不同才能使复制工作。

[root@sql56 ~]# vim /var/lib/mysql/auto.cnf

[auto]

server-uuid=66ad62cc-34d8-11e9-8d49-525400b5016a

解决办法:

删除:/var/lib/mysql/auto.cnf 文件,重启mysql将重新生成一个新的auto.cnf文件

[root@sql56 ~]# rm /var/lib/mysql/auto.cnf

[root@sql56 ~]# systemctl restart mysqld

## 1.5主从同步常用/etc/my.cnf配置参数

### 1.5.1 主服务器配置参数(对所有从库有效)

binlog\_do\_db=name 设置master对哪些库记日志(允许同步的库)

binlog\_ignore\_db=name 设置master对哪些库不记日志(不允许同步的库)

mysql> show master status;

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

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

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

| plj.000006 | 1568 | | | |

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

配置案例

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

server\_id=51

log\_bin=/logdir/plj

binlog\_do\_db=db9

mysql> show master status;

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

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

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

| plj.000007 | 154 | db9 | | |

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

1 row in set (0.00 sec)

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

server\_id=51

log\_bin=/logdir/plj

#binlog\_do\_db=db9

binlog\_ignore\_db=db7,db6,db5

mysql> show master status;

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

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

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

| plj.000008 | 154 | | db7,db6,db5 | |

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

1 row in set (0.00 sec)

### 1.5.2 从服务器配置参数(仅对从库本机有效)

log\_slave\_updates #允许级联复制,允许从服务器再充当主服务器,并可以同步库

Relay\_log=name #指定中继日志文件名

Replicate\_do\_db=库名列表 仅同步的库名列表,其他库被忽略,

Replicate\_ignore\_db=库名列表 仅不同步的库名列表,其他库被同步

Replicate\_do\_db 和 Replicate\_ignore\_db二选其一

#### 只同步指定库案例:

[root@sql52 ~]# vim /etc/my.cnf

[mysqld]

server\_id=52

replicate\_do\_db=db8,db9

mysql> show slave status\G;

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

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

Master\_Host: 192.168.4.51

Master\_User: repluser

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: plj.000008

Read\_Master\_Log\_Pos: 154

Relay\_Log\_File: sql52-relay-bin.000011

Relay\_Log\_Pos: 314

Relay\_Master\_Log\_File: plj.000008

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

Replicate\_Do\_DB: db8,db9

Replicate\_Ignore\_DB:

#### 不同步指定库案例

[root@sql52 ~]# vim /etc/my.cnf

[mysqld]

server\_id=52

#replicate\_do\_db=db8,db9

replicate\_ignore\_db=db7,db6

mysql> show slave status\G;

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

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

Master\_Host: 192.168.4.51

Master\_User: repluser

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: plj.000008

Read\_Master\_Log\_Pos: 154

Relay\_Log\_File: sql52-relay-bin.000015

Relay\_Log\_Pos: 314

Relay\_Master\_Log\_File: plj.000008

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

Replicate\_Do\_DB:

Replicate\_Ignore\_DB: db7,db6

# 二 MySQL主从同步复制模式

## 2.1异步复制模式:

主库执行完一次事务后,立即将结果返给客户端,并不关心从库是否已接收并处理

## 2.2全同步复制模式:

当主库执行完一次事务后,且所有从库都执行了该事务后才返回给客户端

## 2.3半同步复制模式:

介于异步复制和全同步复制之间

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

## 2.4配置半同步复制模式

把主库设置为半同步复制模式 把从库设置为半同步复制模式

**配置命令集:**

查看是否允许动态加载模块:

mysql>show variables like 'have\_dynamic\_loading';

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

| Variable\_name | Value |

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

| have\_dynamic\_loading | YES | #允许

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

1 row in set (0.00 sec)

查看模块是否被加载:

mysql>select plugin\_name,plugin\_status from information\_schema.plugins where plugin\_name like '%semi%';

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

| plugin\_name | plugin\_status |

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

| rpl\_semi\_sync\_master | ACTIVE | #活跃的

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

1 row in set (0.00 sec)

查看模块是否启用:

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

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

| Variable\_name | Value |

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

| rpl\_semi\_sync\_master\_enabled | OFF | #关的

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

1 row in set (0.01 sec)

**主服务**加载半同步复制模块:

mysql>Install plugin rpl\_semi\_sync\_master SONAME 'semisync\_master.so';

**主库**启用模块:

set global rpl\_semi\_sync\_master\_enabled = 1;

**主库**永久配置

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

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

rpl\_semi\_sync\_master\_enabled=1

**从服务**加载半同步复制模块;

mysql> Install plugin rpl\_semi\_sync\_slave soname 'semisync\_slave.so';

Query OK, 0 rows affected (0.09 sec)

**从库**启用模块

mysql>set global rpl\_semi\_sync\_slave\_enabled = 1;

;**从库**永久配置

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

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

rpl\_semi\_sync\_slave\_enabled=1

### 2.4.1把主库51设置为半同步复制模式

#### 1查看是否允许动态加载模块

mysql> show variables like 'have\_dynamic\_loading';

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

| Variable\_name | Value |

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

| have\_dynamic\_loading | YES |

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

1 row in set (0.00 sec)

#### 2查看模块是否被加载

mysql>select plugin\_name,plugin\_status from information\_schema.plugins where plugin\_name like '%semi%';

#### 3主服务加载半同步复制模块

mysql>Install plugin rpl\_semi\_sync\_master SONAME 'semisync\_master.so';

Query OK, 0 rows affected (0.05 sec)

加载完后查看一下

mysql> select plugin\_name,plugin\_status from information\_schema.plugins where plugin\_name like '%semi%';

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

| plugin\_name | plugin\_status |

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

| rpl\_semi\_sync\_master | ACTIVE |

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

1 row in set (0.00 sec)

#### 4主库查看并启用模块

**查看**

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

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

| Variable\_name | Value |

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

| rpl\_semi\_sync\_master\_enabled | OFF |

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

1 row in set (0.00 sec)

**启用后再查看**

mysql>set global rpl\_semi\_sync\_master\_enabled = 1

Query OK, 0 rows affected (0.00 sec)

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

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

| Variable\_name | Value |

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

| rpl\_semi\_sync\_master\_enabled | ON |

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

1 row in set (0.00 sec)

**以上命令设置在重启服务后会失效**

#### 5修改主配置文件永久生效

[root@sql51 ~]# vim /etc/my.cnf

[mysqld]

plugin-load=rpl\_semi\_sync\_master=semisync\_master.so #加载模块

rpl\_semi\_sync\_master\_enabled=1 #启用半同步复制模式

[root@sql51 ~]# systemctl restart mysqld

### 2.4.2把从库52设置为半同步复制模式

#### 1.查看是否允许动态加载模块

mysql> show variables like 'have\_dynamic\_loading';

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

| Variable\_name | Value |

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

| have\_dynamic\_loading | YES |

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

1 row in set (0.00 sec)

#### 2.从服务器加载半同步复制模块

mysql> Install plugin rpl\_semi\_sync\_slave soname 'semisync\_slave.so';

Query OK, 0 rows affected (0.09 sec)

#### 3.查看模块是否被加载

mysql> select plugin\_name,plugin\_status from information\_schema.plugins where plugin\_name like '%semi%';

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

| plugin\_name | plugin\_status |

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

| rpl\_semi\_sync\_slave | ACTIVE |

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

1 row in set (0.01 sec)

#### 4.启用模块

mysql> set global rpl\_semi\_sync\_slave\_enabled = 1;

Query OK, 0 rows affected (0.00 sec)

#### 5.查看模块是否被启用

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

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

| Variable\_name | Value |

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

| rpl\_semi\_sync\_slave\_enabled | ON |

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

#### 6.从库永久配置

[root@sql52 ~]# vim /etc/my.cnf

[mysqld]

plugin-load=rpl\_semi\_sync\_slave=semisync\_slave.so #加载模块

rpl\_semi\_sync\_slave\_enabled=1 #启用半同步复制模块

\

案例1：MySQL一主一从

案例2：配置主从从同步结构

配置半同步复制模式

1 案例1：MySQL一主一从

1.1 问题

构建 主-->从 复制结构

其中主机192.168.4.10作为主库

主机192.168.4.20作为从库

1.2 方案

使用2台RHEL 7虚拟机，如图-1所示。其中192.168.4.10是MySQL主服务器，负责提供同步源；另一台192.168.4.20作为MySQL从服务器，通过调取主服务器上的binlog日志，在本地重做对应的库、表，实现与主服务器的AB复制（同步）。

图－1

提前为两台MySQL服务器安装好MySQL-server、MySQL-Client软件包，并为数据库用户root修改密码；Linux客户机上则只需安装MySQL-Client软件包即可。

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：初始化现有库

为了在启用binlog日志及同步之前保持主、从库的一致性，建议进行初始化——备份主服务器上现有的库，然后导入到从服务器上。

当现有库、表都采用MyISAM引擎时，可执行离线备份、恢复，这样更有效率；否则，可通过mysqldump等工具来实现库的导出、导入。

1）备份MySQL Master（192.168.4.10）上现有的库

如果服务器已经启用binlog，建议对日志做一次重置，否则可忽略：

[root@dbsvr1 ~]# mysql -u root -p

Enter password: //以数据库用户root登入

.. ..

mysql> RESET MASTER; //重置binlog日志

Query OK, 0 rows affected (0.06 sec)

mysql> quit //退出mysql> 环境

Bye

以备份mysql库、sys库为例，导出操作如下：

[root@dbsvr1 ~]# mysqldump -u root -p –all-databases > /root/mytest.sql

Enter password: //验证口令

[root@dbsvr1 ~]# ls -lh /root/mytest.sql //确认备份结果

-rw-r--r--. 1 root root 777172 4月 23 12:21 /root/mytest.sql

2）在MySQL Slave（192.168.4.20）上导入备份的库

先清理目标库，避免导入时冲突。主要是采用InnoDB引擎的库，授权库mysql多采用MyISAM引擎，可不做清理。

[root@dbsvr2 ~]# mysql -u root -p

Enter password: //以数据库用户root登入

.. ..

mysql> DROP DATABASE test; //删除test库

Query OK, 0 rows affected (0.03 sec)

mysql> quit //退出mysql> 环境

Bye

使用scp工具下载备份文件：

[root@dbsvr2 ~]# scp /root/mytest.sql root@192.168.4.20:/

root@dbsvr1's password: //验证对方系统用户root的口令

mytest.sql 100% 759KB 759.0KB/s 00:00

[root@dbsvr2 ~]# ls -lh mytest.sql //确认下载结果

-rw-r--r--. 1 root root 759K 4月 23 12:22 /mytest.sql

执行导入操作：

[root@dbsvr2 ~]# mysql -u root -p < /mytest.sql

Enter password: //验证口令

导入成功后，可重新登入 mysql> 环境，确认清理的目标库已恢复：

mysql> show databases;

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

| Database |

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

| information\_schema |

| mysql |

| performance\_schema |

| sys |

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

4 rows in set (0.00 sec)

步骤二：配置MySQL Master（主服务器，192.168.4.10）

1）修改/etc/my.cnf配置，重新启动MySQL服务程序

指定服务器ID号、允许日志同步：

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

[mysqld]

log\_bin=dbsvr1-bin //启用binlog日志，并指定文件名前缀

server\_id = 10 //指定服务器ID号

......

重启mysql服务：

[root@dbsvr1 ~]# systemctl restart mysqld.service

2）新建一个备份用户，授予复制权限

需要的权限为REPLICATION SLAVE，允许其从Slave服务器访问：

mysql> GRANT REPLICATION SLAVE ON \*.\* TO 'replicater'@'192.168.4.%' IDENTIFIED BY 'pwd123';

Query OK, 0 rows affected, 1 warning (0.09 sec)

3）检查Master服务器的同步状态

在已经初始化现有库的情况下，查看MASTER状态，记录下当前的日志文件名、偏移的位置（下面SLAVE发起复制时需要用到）：

mysql> SHOW MASTER STATUS\G

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

File: dbsvr1-bin.000001 //记住当前的日志文件名

Position: 154 //记住当前的位置

Binlog\_Do\_DB:

Binlog\_Ignore\_DB:

Executed\_Gtid\_Set:

1 row in set (0.00 sec)

步骤三：配置MySQL Slave（从服务器，192.168.4.20）

1）修改/etc/my.cnf配置，重新启动MySQL服务程序

指定服务器ID号、允许日志同步：

[root@dbsvr2 ~]# vim /etc/my.cnf

[mysqld]

log\_bin=dbsvr2-bin //启动SQL日志，并指定文件名前缀

server\_id = 20 //指定服务器ID号，不要与Master的相同

.. ..

在生产环境中，还可以根据需要设置更详细的同步选项。比如，指定当主、从网络中断时的重试超时时间（slave-net-timeout=60 ）等，具体可参考MySQL手册。

配置完成后，重启mysql服务：

[root@dbsvr2 ~]# systemctl restart mysqld.service

通过CHANGE MASTER语句指定MASTER服务器的IP地址、同步用户名/密码、起始日志文件、偏移位置（参考MASTER上的状态输出）：

mysql> CHANGE MASTER TO MASTER\_HOST='192.168.4.10',

-> MASTER\_USER='replicater',

-> MASTER\_PASSWORD='pwd123',

-> MASTER\_LOG\_FILE='dbsvr1-bin.000002', //对应Master的日志文件

-> MASTER\_LOG\_POS=334; //对应Master的日志偏移位置

Query OK, 0 rows affected, 2 warnings (0.12 sec)

然后执行START SLAVE（较早版本中为SLAVE START）启动复制：

mysql> START SLAVE; //启动复制

Query OK, 0 rows affected (0.00 sec)

注意：一旦启用SLAVE复制，当需要修改MASTER信息时，应先执行STOP SLAVE停止复制，然后重新修改、启动复制。

通过上述连接操作，MASTER服务器的设置信息自动存为master.info文件，以后每次MySQL服务程序时会自动调用并更新，无需重复设置。查看master.info文件的开头部分内容，可验证相关设置：

[root@dbsvr2 ~]# ls -lh /var/lib/mysql/master.info

-rw-r-----. 1 mysql mysql 132 4月 23 12:06 /var/lib/mysql/master.info

[root@dbsvr2 ~]# head /var/lib/mysql/master.info

25

dbsvr1-bin.000001

154

192.168.4.10

replicater

pwd123

3306

60

0

2）检查Slave服务器的同步状态

通过SHOW SLAVE STATUS语句可查看从服务器状态，确认其中的IO线程、SQL线程正常运行，才能成功同步：

mysql> SHOW SLAVE STATUS\G

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

Master\_Host: 192.168.4.1

Master\_User: replicater

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: dbsvr1-bin.000001

Read\_Master\_Log\_Pos: 154

Relay\_Log\_File: db2-relay-bin.000003

Relay\_Log\_Pos: 321

Relay\_Master\_Log\_File: dbsvr1-bin.000001

Slave\_IO\_Running: Yes //IO线程应该已运行

Slave\_SQL\_Running: Yes //SQL线程应该已运行

Replicate\_Do\_DB:

Replicate\_Ignore\_DB:

Replicate\_Do\_Table:

Replicate\_Ignore\_Table:

Replicate\_Wild\_Do\_Table:

Replicate\_Wild\_Ignore\_Table:

Last\_Errno: 0

Last\_Error:

Skip\_Counter: 0

Exec\_Master\_Log\_Pos: 154

Relay\_Log\_Space: 2490

Until\_Condition: None

Until\_Log\_File:

Until\_Log\_Pos: 0

Master\_SSL\_Allowed: No

Master\_SSL\_CA\_File:

Master\_SSL\_CA\_Path:

Master\_SSL\_Cert:

Master\_SSL\_Cipher:

Master\_SSL\_Key:

Seconds\_Behind\_Master: 0

Master\_SSL\_Verify\_Server\_Cert: No

Last\_IO\_Errno: 0

Last\_IO\_Error:

Last\_SQL\_Errno: 0

Last\_SQL\_Error:

Replicate\_Ignore\_Server\_Ids:

Master\_Server\_Id: 10

Master\_UUID: 2d4d8a11-27b7-11e7-ae78-52540055c180

Master\_Info\_File: /var/lib/mysql/master.info

SQL\_Delay: 0

SQL\_Remaining\_Delay: NULL

Slave\_SQL\_Running\_State: Slave has read all relay log; waiting for more updates

Master\_Retry\_Count: 86400

Master\_Bind:

Last\_IO\_Error\_Timestamp:

Last\_SQL\_Error\_Timestamp:

Master\_SSL\_Crl:

Master\_SSL\_Crlpath:

Retrieved\_Gtid\_Set:

Executed\_Gtid\_Set:

Auto\_Position: 0

Replicate\_Rewrite\_DB:

Channel\_Name:

Master\_TLS\_Version:

1 row in set (0.00 sec)

若START SLAVE直接报错失败，请检查CHANGE MASTER相关设置是否有误，纠正后再重试；若IO线程或SQL线程有一个为“No”，则应检查服务器的错误日志，分析并排除故障后重启主从复制。

步骤四：测试主从同步效果

1）在Master上操作数据库、表、表记录

新建newdb库、newtable表，随意插入几条表记录：

mysql> CREATE DATABASE newdb; //新建库newdb

Query OK, 1 row affected (0.17 sec)

mysql> USE newdb; //切换到newdb库

Database changed

mysql> CREATE TABLE newtable(id int(4)); //新建newtable表

Query OK, 0 rows affected (0.46 sec)

mysql> INSERT INTO newtable VALUES(1234),(5678); //插入2条表记录

Query OK, 2 rows affected (0.24 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT \* FROM newtable; //确认表数据

+------+

| id |

+------+

| 1234 |

| 5678 |

+------+

2 rows in set (0.00 sec)

2）在Slave上确认自动同步的结果

直接切换到newdb库，并查询newtable表的记录，应该与Master上的一样，这才说明主从同步已经成功生效：

mysql> USE newdb; //直接切换到newdb库

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> SELECT \* FROM newtable; //输出表记录

+------+

| id |

+------+

| 1234 |

| 5678 |

+------+

2 rows in set (0.02 sec)

3）在Master服务器上可查看Slave主机的信息

mysql> SHOW SLAVE HOSTS;

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

| Server\_id | Host | Port | Master\_id | Slave\_UUID |

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

| 2 | | 3306 | 10 | 512cf7c1-27c4-11e7-8f4b-5254007b030b |

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

1 row in set (0.00 sec)

2 案例2：配置主从从同步结构

2.1 问题

具体要求如下：

配置主机192.168.4.51为主数据库服务器

配置主机192.168.4.52为51主机的从库服务器

配置主机192.168.4.53为52主机的从库服务器

客户端连接主数据库服务器51主机创建的数据，连接52和53主机时，也可以访问到库、表、记录。

2.2 方案

使用3台RHEL 7虚拟机，如图-2所示。其中192.168.4.51是MySQL主服务器，负责提供同步源；另一台192.168.4.52作为192.168.4.51从服务器，最后一台192.168.4.53作为192.168.4.52从服务器，通过调取主服务器上的binlog日志，客户端访问主库51 时 创建库表记录 在52 和53 数据库服务器都可以看到

图－2

2.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：环境准备

为了在启用binlog日志及同步之前保持主、从库的一致性，主从同步未配置之前，要保证从库上要有主库上的数据，禁用selinux，关闭防火墙服务，保证物理连接正常

[root@db51 ~]# systemctl stop firewalld

[root@db51 ~]# setenforce 0

[root@db51 ~]# ping -c 2 192.168.4.51

PING 192.168.4.51 (192.168.4.51) 56(84) bytes of data.

64 bytes from 192.168.4.51: icmp\_seq=1 ttl=64 time=0.059 ms

64 bytes from 192.168.4.51: icmp\_seq=2 ttl=64 time=0.063 ms

--- 192.168.4.51 ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 999ms

rtt min/avg/max/mdev = 0.059/0.061/0.063/0.002 ms

[root@db51 ~]# ping -c 2 192.168.4.52

PING 192.168.4.52 (192.168.4.52) 56(84) bytes of data.

64 bytes from 192.168.4.52: icmp\_seq=1 ttl=64 time=0.698 ms

64 bytes from 192.168.4.52: icmp\_seq=2 ttl=64 time=0.365 ms

--- 192.168.4.52 ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 1000ms

rtt min/avg/max/mdev = 0.365/0.531/0.698/0.168 ms

步骤二：配置主服务器192.168.4.51

1）对yaya用户进行授权

[root@db51 ~]# mysql -uroot -p123456

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

Query OK, 0 rows affected, 1 warning (0.03 sec)

2）启用binlog日志，修改/etc/my.cnf配置，重新启动MySQL服务程序

指定服务器ID号、允许日志同步：

[root@db51 ~]# vim /etc/my.cnf

[mysqld]

log\_bin=db51 //启用binlog日志，并指定文件名前缀

server\_id=51 //指定服务器ID号

binlog-format="mixed" // 指定binlog日志格式

重启mysql服务：

[root@db51 ~]# systemctl restart mysqld

确保/var/lib/mysql下面有两个文件：

[root@db51 ~]# ls /var/lib/mysql/db51.\*

/var/lib/mysql/db51.000001 /var/lib/mysql/db51.index

查看主服务正在使用的日志信息

查看主服务器状态，记录下当前的日志文件名、偏移的位置（下面SLAVE发起复制时需要用到）：

mysql> show master status;

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

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

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

| db51.000002 | 437 | | | |

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

1 row in set (0.00 sec)

步骤三：配置从服务器192.168.4.52

1）在服务器192.168.4.52上对user53用户进行授权

[root@db52 ~]# mysql -u root -p123456

mysql> grant replication slave on \*.\* to user53@"192.168.4.53" identified by "654321";

Query OK, 0 rows affected, 1 warning (0.00 sec)

2）修改/etc/my.cnf配置，启用binlog日志，指定server\_id 和 允许级联复制

[root@db52 ~]# vim /etc/my.cnf

[mysqld]

server\_id=52

log-bin=db52

binlog-format="mixed"

log\_slave\_updates //允许级联复制

3）配置完成后，重启mysql服务：

[root@db52 ~]# systemctl restart mysqld

4）确保/var/lib/mysql下面有两个文件：

[root@db52 ~]# ls /var/lib/mysql/db52.\*

/var/lib/mysql/db52.000001 /var/lib/mysql/db52.index

5）查看正在使用的日志信息

[root@db52 ~]# mysql -uroot -p123456

mysql> show master status;

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

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

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

| db52.000001 | 154 | | | |

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

1 row in set (0.00 sec) //查看日志文件名、偏移的位置

6）验证主库的授权用户

[root@db52 ~]# mysql -h192.168.4.51 -uyaya -p123456

mysql: [Warning] Using a password on the command line interface can be insecure.

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 4

Server version: 5.7.17-log MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> //验证成功

7）通过change master语句指定master服务器的IP地址、同步用户名/密码、起始日志文件、偏移位置（参考master上的状态输出）：

[root@db52 ~]# mysql -uroot -p123456

mysql> change master to

-> master\_host="192.168.4.51",

-> master\_user="yaya",

-> master\_password="123456",

-> master\_log\_file="db51.000002",

-> master\_log\_pos=437;

Query OK, 0 rows affected, 2 warnings (0.43 sec)

8）启动slave进程

mysql> start slave;

Query OK, 0 rows affected (0.03 sec)

9）查看进程状态信息，通过show slave status语句可查看从服务器状态，确认其中的IO线程、SQL线程正常运行，才能成功同步,IO线程和SQL线程必须是Yes

mysql> show slave status \G;

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

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

Master\_Host: 192.168.4.51

Master\_User: yaya

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: db51.000002

Read\_Master\_Log\_Pos: 437

Relay\_Log\_File: db52-relay-bin.000002

Relay\_Log\_Pos: 315

Relay\_Master\_Log\_File: db51.000002

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

Replicate\_Do\_DB:

Replicate\_Ignore\_DB:

Replicate\_Do\_Table:

Replicate\_Ignore\_Table:

Replicate\_Wild\_Do\_Table:

Replicate\_Wild\_Ignore\_Table:

Last\_Errno: 0

Last\_Error:

Skip\_Counter: 0

Exec\_Master\_Log\_Pos: 437

Relay\_Log\_Space: 521

Until\_Condition: None

Until\_Log\_File:

Until\_Log\_Pos: 0

Master\_SSL\_Allowed: No

Master\_SSL\_CA\_File:

Master\_SSL\_CA\_Path:

Master\_SSL\_Cert:

Master\_SSL\_Cipher:

Master\_SSL\_Key:

Seconds\_Behind\_Master: 0

Master\_SSL\_Verify\_Server\_Cert: No

Last\_IO\_Errno: 0

Last\_IO\_Error:

Last\_SQL\_Errno: 0

Last\_SQL\_Error:

Replicate\_Ignore\_Server\_Ids:

Master\_Server\_Id: 51

Master\_UUID: 81a13101-aa66-11e8-ad11-525400019e62

Master\_Info\_File: /var/lib/mysql/master.info

SQL\_Delay: 0

SQL\_Remaining\_Delay: NULL

Slave\_SQL\_Running\_State: Slave has read all relay log; waiting for more updates

Master\_Retry\_Count: 86400

Master\_Bind:

Last\_IO\_Error\_Timestamp:

Last\_SQL\_Error\_Timestamp:

Master\_SSL\_Crl:

Master\_SSL\_Crlpath:

Retrieved\_Gtid\_Set:

Executed\_Gtid\_Set:

Auto\_Position: 0

Replicate\_Rewrite\_DB:

Channel\_Name:

Master\_TLS\_Version:

1 row in set (0.00 sec)

步骤四：配置从服务器192.168.4.53

1）验证主库的授权用户

[root@db53 ~]# mysql -h192.168.4.52 -uuser53 -p654321

mysql: [Warning] Using a password on the command line interface can be insecure.

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 7

Server version: 5.7.17-log MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> //验证成功

2）指定server\_id

[mysqld]

validate\_password\_policy=0

validate\_password\_length=6

server\_id=53

3）重新启动服务

[root@db53 ~]# systemctl restart mysqld

4）管理员登录指定主库信息

[root@db53 ~]# mysql -uroot -p123456

mysql> change master to

-> master\_host="192.168.4.52",

-> master\_user="user53",

-> master\_password="654321",

-> master\_log\_file=" db52.000001",

-> master\_log\_pos=154;

Query OK, 0 rows affected, 2 warnings (0.37 sec)

5）启动slave进程

mysql> start slave;

Query OK, 0 rows affected (0.04 sec)

6）查看进程状态信息

mysql> show slave status\G

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

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

Master\_Host: 192.168.4.52

Master\_User: user53

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: db52.000001

Read\_Master\_Log\_Pos: 154

Relay\_Log\_File: db53-relay-bin.000003

Relay\_Log\_Pos: 315

Relay\_Master\_Log\_File: db52.000001

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

步骤五：客户端验证配置

1）在主服务器上在主库上授权访问gamedb库的用户

[root@db51 ~]# mysql -uroot -p123456

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

Query OK, 0 rows affected, 1 warning (0.03 sec)

2）客户端使用授权用户连接主库，建库、表、插入记录

[root@room9pc01 ~]# mysql -h192.168.4.51 -udada -p123456

Welcome to the MariaDB monitor. Commands end with ; or \g.

Your MySQL connection id is 7

Server version: 5.7.17-log MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> //验证成功

MySQL [(none)]> create database gamedb; //创建测试库

Query OK, 1 row affected (0.04 sec)

MySQL [(none)]> create table gamedb.t1(id int); //在gamedb下创建t1表

Query OK, 0 rows affected (0.17 sec)

MySQL [(none)]> insert into gamedb.t1 values(8888); //在t1表中插入数值

Query OK, 1 row affected (0.22 sec)

3）客户端使用授权用户连接2台从库时，也可以看到主库上新的库表记录

[root@room9pc01 ~]# mysql -h192.168.4.52 -udada -p123456 //验证52主机的状态

Welcome to the MariaDB monitor. Commands end with ; or \g.

Your MySQL connection id is 10

Server version: 5.7.17-log MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> select \* from gamedb.t1; //查询插入的表格

+------+

| id |

+------+

| 8888 |

+------+

1 row in set (0.00 sec)

MySQL [(none)]> exit

[root@room9pc01 ~]# mysql -h192.168.4.53 -udada -p123456 //验证53主机的状态

Welcome to the MariaDB monitor. Commands end with ; or \g.

Your MySQL connection id is 6

Server version: 5.7.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> select \* from gamedb.t1;

+------+

| id |

+------+

| 8888 |

+------+

1 row in set (0.00 sec)

3 配置半同步复制模式

3.1 问题

开启案例1 主库192.168.4.51 半同步复制模式

开启案例1 从库192.168.4.52 半同步复制模式

开启案例1 从库192.168.4.53 半同步复制模式

查看半同步复制模式是否开启

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：查看是否允许动态加载模块，

1）查看是否允许动态加载模块默认允许

mysql> show variables like 'have\_dynamic\_loading';

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

| Variable\_name | Value |

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

| have\_dynamic\_loading | YES |

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

1 row in set (0.01 sec)

2）命令行加载插件，用户需有SUPER权限

主库上面操作：

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

从库上面操作：

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

查看系统库下的表，模块是否安装成功：

mysql> SELECT PLUGIN\_NAME, PLUGIN\_STATUS FROM INFORMATION\_SCHEMA.PLUGINS WHERE PLUGIN\_NAME LIKE '%semi%';

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

| PLUGIN\_NAME | PLUGIN\_STATUS |

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

| rpl\_semi\_sync\_master | ACTIVE |

| rpl\_semi\_sync\_slave | ACTIVE |

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

2 rows in set (0.00 sec)

3）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

主库上面执行：

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

Query OK, 0 rows affected (0.00 sec)

从库上面执行：

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

Query OK, 0 rows affected (0.00 sec)

查看半同步复制模式是否启用：

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

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

| Variable\_name | Value |

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

| rpl\_semi\_sync\_master\_enabled | ON |

| rpl\_semi\_sync\_slave\_enabled | ON |

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

2 rows in set (0.00 sec)

4）永久启用半同步复制

主库配置

[root@master51 ~]# vim /etc/my.cnf

[mysqld]

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

rpl\_semi\_sync\_master\_enabled=1

从库配置

[root@slave52 ~]# vim /etc/my.cnf

[mysqld]

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

[root@master51 ~]# vim /etc/my.cnf

[mysqld]

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

# NSD DBA2 DAY02

# 一 MySQL数据读写分离

## 1.1 Mysql数据读写分离介绍:

把客户端访问时的查询(读)请求与存储(写)数据请求分别给不同的数据库服务器处理

查询(读):select 等操作分配给从数据库服务器

存储(写):insert delete update等操作分配给主数据库服务器

MySQL代理进程使用rw-splitting.lua脚本实现读写分离

## 2.2为什么要配置数据读写分离:

## 2.3配置数据读写分离

### 2.3.1拓扑结构

50为客户端 57为代理 51为主服务器 52为从服务器

50访问57,并且57判断的访问类型并相应转发:

如果50访问数据库操作是查询(读):57转发给51

如果50访问数据库操作是存储(写):57转发给52

### 2.3.2配置mysql 主从同步结构

清空至51 52为独立数据库服务器,并取消所有授权用户

### 2.3.3配置数据读写分离服务器57

了解内容:mysql中间件实现数据读写分离功能

mysql中间件软件有:maxscale mysql-proxy mycat

准备好maxscale软件包之后,直接安装,没有依赖

作用:提供管理命令 和 配置文件

[root@room9pc01 ~]# scp '/root/桌面/DBA/数据库/maxscale-2.1.2-1.rhel.7.x86\_64.rpm' 192.168.4.57:/root

[root@sql57 ~]# rpm -ivh maxscale-2.1.2-1.rhel.7.x86\_64.rpm

#### 1.安装提供服务的软件

maxscale-2.1.2-1.rhel.7.x86\_64.rpm

[root@sql57 ~]# ls /etc/maxscale.

maxscale.cnf maxscale.cnf.template maxscale.modules.d/

[root@sql57 ~]# max #两下tab显示全部命令

maxadmin maxbinlogcheck maxpasswd

maxavrocheck maxkeys maxscale

#### 修改主配置文件

[root@sql57 ~]# cp /etc/maxscale.cnf /root/

[root@sql57 ~]# vim /etc/maxscale.cnf

9 [maxscale]

10 threads=auto #服务运行后开启的线程数量默认为1,auto为根据cpu自动分配

18 [server1] #数据库服务器1主库

19 type=server

20 address=192.168.4.51

21 port=3306

22 protocol=MySQLBackend

24 [server2] #数据库服务器2从库 复制server1修改

25 type=server

26 address=192.168.4.52

27 port=3306

28 protocol=MySQLBackend

36 [MySQL Monitor] #定义监视的数据库

37 type=monitor

38 module=mysqlmon #监视调用的程序

39 servers=server1,server2 #监控的服务器名(本配置文件上方的配置)

40 user=maxmon #server1与server2授权的用户用来监视两个服务

41 passwd=123456 #maxmon密码

42 monitor\_interval=10000

53 #[Read-Only Service] #定义只读服务器,先不定义全部注释掉

54 #type=service

55 #router=readconnroute

56 #servers=server1

57 #user=myuser

58 #passwd=mypwd

59 #router\_options=slave

64 [Read-Write Service] #定义读写分离服务器

65 type=service

66 router=readwritesplit

67 servers=server1,server2

68 user=scaluser #路由用户用来检查数据库服务有没有客户机访问的授权用户

69 passwd=123456 #此用户要在两个数据库服务上授权相应权限

70 max\_slave\_connections=100%

**#路由用户:**客户端使用-uroot -p123456用户访问数据库,代理用路由用户登录数据库服务器,检查授权表,验证root用户的密码和权限是否符合访问

76 [MaxAdmin Service] #定义管理服务不修改

77 type=service

78 router=cli

86 #[Read-Only Listener] #定义只读服务的端口,不需要先全注释掉

87 #type=listener

88 #service=Read-Only Service

89 #protocol=MySQLClient

90 #port=4008

92 [Read-Write Listener] #定义读写监听的端口号

93 type=listener

94 service=Read-Write Service #执行读写服务的配置项(本配置上面有此项64行)

95 protocol=MySQLClient

96 port=4006 #监听端口

98 [MaxAdmin Listener] #管理端口

99 type=listener

100 service=MaxAdmin Service

101 protocol=maxscaled

102 socket=default

103 port=4016 #手动添加管理端口为:4016默认为端口要启动后才

#### 根据配置文件的定义,创建对应的用户(51/52上)

创建监控用户 :maxmon 密码:123456赋予权限:replication slave, replication client

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

Query OK, 0 rows affected, 1 warning (0.04 sec)

路由用户:scaluser 密码:123456 赋予查询权限select

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

主服务器51 , 52上查看创建的用户权限,:

mysql> show grants for maxmon;

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

| Grants for maxmon@% |

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

| GRANT REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO 'maxmon'@'%' |

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

1 row in set (0.00 sec)

mysql> show grants for scaluser;

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

| Grants for scaluser@% |

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

| GRANT SELECT ON \*.\* TO 'scaluser'@'%' |

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

1 row in set (0.00 sec)

查看用户

mysql> select host,user from mysql.user where user in ("maxmon","scaluser");

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

| host | user |

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

| % | maxmon |

| % | scaluser |

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

2 rows in set (0.00 sec)

用maxmon 与 scaluser 用户在57上登录

#### 启动服务

[root@sql57 ~]# maxscale -f /etc/maxscale.cnf

[root@sql57 ~]# ps -C maxscale

PID TTY TIME CMD

3020 ? 00:00:00 maxscale

[root@sql57 ~]# netstat -utnlp | grep maxscale

tcp6 0 0 :::4016 :::\* LISTEN 3020/maxscale

tcp6 0 0 :::4006 :::\* LISTEN 3020/maxscale

#### 5停止服务(kill杀死进程)

[root@sql57 ~]# killall -9 maxscale

[root@sql57 ~]# ps -C maxscale

PID TTY TIME CMD

[root@sql57 ~]# netstat -utnlp | grep maxscale

#### 6.在本机访问管理服务,查看监控信息

[root@sql57 ~]# maxadmin -P4016 -uadmin -pmariadb

MaxScale> list servers

Servers.

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

Server | Address | Port | Connections | Status

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

server1 | 192.168.4.51 | 3306 | 0 | Master, Running

server2 | 192.168.4.52 | 3306 | 0 | Slave, Running

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

#### 7.在主服务器51上创建拥有查询与插入权限的用户

mysql> grant select,insert on \*.\* to jim@"%" identified by "123456";

Query OK, 0 rows affected, 1 warning (0.02 sec)

#### 访问测试

50为客户端 57为代理 51为主服务器 52为从服务器

**主库51**上建立db09库,t1表,插入两条数据,如下

mysql> select \* from db09.t1;

+------+

| id |

+------+

| 11 |

| 22 |

+------+

2 rows in set (0.00 sec)

**从库52**上查询有与51上相同库db09库t1b表相同的两条数据

另外.在52上新插入两条数据,主库51不会同步这两条数据,52数据如下

mysql> select \* from db09.t1;

+------+

| id |

+------+

| 11 |

| 22 |

| 33 |

| 44 |

+------+

**在客户端50** 上登录访问查看和插入

[root@client-50 ~]# mysql -h192.168.4.100 -P4006 -ujim -p123456

mysql> select \* from db09.t1; #查询出来的是从服务器数据

+------+

| id |

+------+

| 11 |

| 22 |

| 33 |

| 44 |

+------+

mysql> insert into db09.t1 values(55);

Query OK, 1 row affected (0.07 sec)

**客户端50**查询出来的数据如下

mysql> select \* from db09.t1;

+------+

| id |

+------+

| 11 |

| 22 |

| 33 |

| 44 |

| 55 |

+------+

5 rows in set (0.00 sec)

**主库51**此时查出来的数据可看到,客户端插入的数据

mysql> select \* from db09.t1;

+------+

| id |

+------+

| 11 |

| 22 |

| 55 |

+------+

**从库52**查出来的数据可看到,从主库同步过来的第5条数据

mysql> select \* from db09.t1;

+------+

| id |

+------+

| 11 |

| 22 |

| 33 |

| 44 |

| 55 |

+------+

# 二MySQL多实例

## 2.1 Mysql多实例介绍

在一台物理机上运行多个数据库服务 作用节约成本 提高硬件利用率

## 2.2配置mysql多实例

### 2.2.1安装包(源码包)

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

[root@client-50 ~]# systemctl stop mysqld.service

[root@client-50 ~]# systemctl disable mysqld.service

[root@client-50 ~]# tar -xf mysql-5.7.20-linux-glibc2.12-x86\_64.tar.gz

[root@client-50 ~]# mv mysql-5.7.20-linux-glibc2.12-x86\_64 /usr/local/mysql

[root@client-50 ~]# ls /usr/local/mysql

bin COPYING docs include lib man README share support-files

复习下源码包的使用和命令

### 2.2.2创建配置文件定义多实例服务

[root@client-50 ~]# mv /etc/my.cnf /root/

[root@client-50 ~]# vim /etc/my.cnf

[mysqld\_multi] #定义启动多实例程序

mysqld=/usr/local/mysql/bin/mysqld\_safe #定义进程文件路径

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

user=root #指定进程用户

[mysqld1] #第一个实例

datadir=/dir1 #自定义数据库文件目录,需要手动创建

port=3307 #自定义端口号

log-error=/dir1/mysql3307.err #自定义日志文件

pid-file=/dir1/mysql3307.pid #自定义pid号文件

socket=/dir1/mysql3307.sock #自定义socket套接字文件

[mysqld2] #第二个实例

datadir=/dir2 #自定义数据库文件目录,需要手动创建

port=3308 #自定义端口号

log-error=/dir2/mysql3308.err #自定义日志文件

pid-file=/dir2/mysql3308.pid #自定义pid号文件

socket=/dir2/mysql3308.sock #自定义socket文件

Socket:文件:套接字文件,自己访问自己的

[root@sql52 ~]# ls -l /var/lib/mysql/mysql.sock

srwxrwxrwx. 1 mysql mysql 0 2月 21 08:11 /var/lib/mysql/mysql.sock

运行数据库时会有 /var/lib/mysql/mysql.sock文件,

[root@client-50 ~]# mkdir /dir1

[root@client-50 ~]# mkdir /dir2

## 2.3使用mysql多实例

如果没有用命令初始化授权库,那么在启动时候将会自动初始化授权库,创建数据

教课案例上有添加变量 , 给数据库目录/dri1 /dir2授属组mysql属主mysql , 初始化授权库等操作都不用做直接启动数据库.

### 2.3.1启动服务

首次启动,会创建数据库相关文件,并且生成初始密码

[root@client-50 ~]# /usr/local/mysql/bin/mysqld\_multi start 1

:2019-02-21T06:38:26.943809Z 1 [Note] A temporary password is generated for root@localhost: +lak&lTbh2sc #初始密码

[root@client-50 ~]# ls /dir1 #查看数据库目录.确保有sock文件

auto.cnf ibdata1 ib\_logfile1 mysql mysql3307.pid mysql3307.sock.lock sys

ib\_buffer\_pool ib\_logfile0 ibtmp1 mysql3307.err mysql3307.sock performance\_schema

[root@client-50 ~]# /usr/local/mysql/bin/mysqld\_multi start 2

2019-02-21T07:08:03.076294Z 1 [Note] A temporary password is generated for root@localhost: pqMSsswNL8<l

[root@client-50 ~]# ls /dir2

auto.cnf ibdata1 ib\_logfile1 mysql mysql3308.pid mysql3308.sock.lock sys

ib\_buffer\_pool ib\_logfile0 ibtmp1 mysql3308.err mysql3308.sock performance\_schema

### 2.3.2查看服务信息

[root@client-50 ~]# netstat -ntulp | grep 3307

tcp6 0 0 :::3307 :::\* LISTEN 5278/mysqld

[root@client-50 ~]# netstat -ntulp |grep 3308

tcp6 0 0 :::3308 :::\* LISTEN 5805/mysqld

[root@client-50 ~]# netstat -nultp |grep mysqld

tcp6 0 0 :::3307 :::\* LISTEN 6117/mysqld

tcp6 0 0 :::3308 :::\* LISTEN 5805/mysqld

### 2.3.3访问服务

[root@client-50 ~]# /usr/local/mysql/bin/mysql -uroot -p'+lak&lTbh2sc' -S /dir1/mysql3307.sock

mysql> alter user root@"localhost" identified by "123456"; #修改初始密码

Query OK, 0 rows affected (0.00 sec)

[root@client-50 ~]# /usr/local/mysql/bin/mysql -uroot -p"pqMSsswNL8<l" -S /dir2/mysql3308.sock

mysql> alter user root@"localhost" identified by "123456";

Query OK, 0 rows affected (0.00 sec)

[root@client-50 ~]# /usr/local/mysql/bin/mysql -uroot -p123456 -S /dir1/mysql3307.sock

### 2.3.4存储数据

mysql> create database bbsdb;

Query OK, 1 row affected (0.00 sec)

mysql> create table bbsdb.a(id int);

Query OK, 0 rows affected (0.24 sec)

mysql> insert into bbsdb.a values(11);

Query OK, 1 row affected (0.03 sec)

### 2.3.5停止服务

[root@client-50 ~]# /usr/local/mysql/bin/mysqld\_multi --user=root --password=123456 stop 1

[root@client-50 ~]# /usr/local/mysql/bin/mysqld\_multi --user=root --password=123456 stop 2

#### 2.3.6 性能调试

[root@client-50 ~]# vim /etc/my.cnf

[mysqld1]

server\_id=1

log-bin=mysql111

[root@client-50 ~]# /usr/local/mysql/bin/mysql -uroot -p123456 -S /dir1/mysql3307.sock

mysql> show master status;

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

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

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

| mysql111.000001 | 154 | | | |

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

1 row in set (0.00 sec)

授权使用:

允许客户端使用admin用户连接50数据库服务,数据存储在实例2服务的db8库的a表

1. 开启实例2,并登录

[root@client-50 ~]# /usr/local/mysql/bin/mysqld\_multi start 2

[root@client-50 ~]# /usr/local/mysql/bin/mysql -uroot -p123456 -S /dir2/mysql3308.sock

1. 管理员授权admin

mysql> grant all on db8.\* to admin@"%" identified by "123456";

Query OK, 0 rows affected, 1 warning (0.00 sec)

1. 客户端登录,按照端口登录相应实例

[root@sql52 ~]# mysql -h192.168.4.50 -P3308 -uadmin -p123456

1. 操作数据库

mysql> show databases;

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

| Database |

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

| information\_schema |

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

1 row in set (0.00 sec)

mysql> create database db8;

Query OK, 1 row affected (0.00 sec)

mysql> create table db8.a(id int);

Query OK, 0 rows affected (0.29 sec)

mysql> insert into db8.a values(66);

Query OK, 1 row affected (0.04 sec)

mysql> insert into db8.a values(88);

Query OK, 1 row affected (0.02 sec)

mysql> select \* from db8.a;

+------+

| id |

+------+

| 66 |

| 88 |

+------+

2 rows in set (0.00 sec)

# 三 Mysql性能调忧

(在nosql 第一天中午时段讲)

提高mysql系统的戏能,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
* 网络带宽,网络架构

[root@client-50 ~]# uptime #查看cpu使用情况

11:28:01 up 3:25, 1 user, load average: 0.06, 0.03, 0.05

[root@client-50 ~]# iostat

Linux 3.10.0-693.el7.x86\_64 (client-50) 2019年02月27日 \_x86\_64\_ (1 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle

0.05 0.00 0.08 0.19 0.03 99.64

Device: tps kB\_read/s kB\_wrtn/s kB\_read kB\_wrtn

vda 1.24 22.79 5.16 281286 63650

dm-0 1.19 22.04 4.99 272051 61602

dm-1 0.00 0.09 0.00 1060 0

#Iowait -等待

[root@client-50 ~]# free -m #查看内存

total used free shared buff/cache available

Mem: 992 313 344 7 334 488

Swap: 0 0 0

## 查看数据库相关配置

### 连接数/并发量

mysql> show variables like "%conn%";

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

| Variable\_name | Value |

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

| character\_set\_connection | utf8 |

| collation\_connection | utf8\_general\_ci |

| connect\_timeout | 10 |

| disconnect\_on\_expired\_password | ON |

| init\_connect | |

| max\_connect\_errors | 100 |

| max\_connections | 151 | #并发连接数

| max\_user\_connections | 0 |

| performance\_schema\_session\_connect\_attrs\_size | 512 |

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

9 rows in set (0.00 sec)

mysql> set global max\_connections=200; #修改并发连接为200

Query OK, 0 rows affected (0.00 sec)

mysql> show variables like "%max\_connection%";

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

| Variable\_name | Value |

+-----------------+-------+

| max\_connections | 200 |

+-----------------+-------+

1 row in set (0.01 sec)

### 连接超时

mysql> show variables like "%timeout%";

+-----------------------------+----------+

| Variable\_name | Value |

+-----------------------------+----------+

| connect\_timeout | 10 | #单位为s,10s内连接失败,

| delayed\_insert\_timeout | 300 |

| have\_statement\_timeout | YES |

| innodb\_flush\_log\_at\_timeout | 1 |

| innodb\_lock\_wait\_timeout | 50 |

| innodb\_rollback\_on\_timeout | OFF |

| interactive\_timeout | 28800 |

| lock\_wait\_timeout | 31536000 |

| net\_read\_timeout | 30 |

| net\_write\_timeout | 60 |

| rpl\_stop\_slave\_timeout | 31536000 |

| slave\_net\_timeout | 60 |

| wait\_timeout | 28800 |

+-----------------------------+----------+

13 rows in set (0.00 sec)

mysql>

### 缓存相关参数

mysql> show variables like "%buffer%";

### 查询优化

mysql> show global status like"%qcache%" ;

+-------------------------+---------+

| Variable\_name | Value |

+-------------------------+---------+

| Qcache\_free\_blocks | 1 |

| Qcache\_free\_memory | 1031832 |

| Qcache\_hits | 0 | #查询缓存里被查找到数据的次数

| Qcache\_inserts | 0 | #查询请求总数

| Qcache\_lowmem\_prunes | 0 | #清除查询缓存数据的次数

| Qcache\_not\_cached | 1 |

| Qcache\_queries\_in\_cache | 0 |

| Qcache\_total\_blocks | 1 |

+-------------------------+---------+

8 rows in set (0.00 sec)

### 优化程序元访问数据的sql命令

数据库服务日志类型:服务的日记文件

错误日志 binlog日志 查询日志 慢查询日志

#### 错误日志:

默认启用 记录服务启动和运行过程中的信息 /var/log/mysql.log

#### binlog日志:

#### 查询日志:

记录执行的所有sql命令

在my.cnf文件中添加:general-log 重启服务,得到 主机名.log 日志文件

可实时监控 tail -f /var/lib/mysql/主机名.log

[root@client-50 ~]# vim /etc/my.cnf

[mysqld]

general-log

[root@client-50 ~]# systemctl restart mysqld

[root@client-50 ~]# vim /var/lib/mysql/client-50.log

[root@client-50 ~]#tail -f /var/lib/mysql/主机名.log

/usr/sbin/mysqld, Version: 5.7.17-log (MySQL Community Server (GPL)). started with:

Tcp port: 0 Unix socket: /var/lib/mysql/mysql.sock

Time Id Command Argument

2019-02-27T07:28:42.921195Z 1 Query SELECT TABLE\_SCHEMA, TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES WHERE CREATE\_OPTIONS LIKE '%partitioned%';

2019-02-27T07:30:57.504272Z 3 Connect root@localhost on using Socket

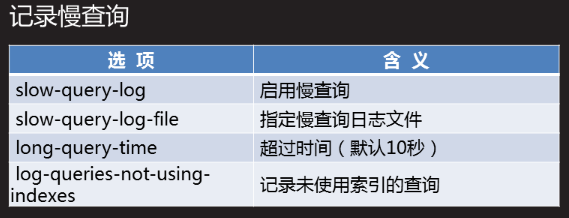
2019-02-27T07:30:57.504996Z 3 Query select @@version\_comment limit 1

2019-02-27T07:31:11.846030Z 3 Query show databases

#### 慢查询日志

记录耗时较长或不使用索引的查询操作

记录超过指定时间(默认10s)显示



[root@client-50 ~]# vim /etc/my.cnf

[mysqld]

slow-query-log

long-query-time=5

log-queries-not-using-indexes

[root@client-50 ~]# systemctl restart mysqld

[root@client-50 ~]# tail -f /var/lib/mysql/client-50-slow.log

/usr/sbin/mysqld, Version: 5.7.17-log (MySQL Community Server (GPL)). started with:

Tcp port: 0 Unix socket: /var/lib/mysql/mysql.sock

Time Id Command Argument

# Time: 2019-02-27T07:47:27.910641Z

# User@Host: root[root] @ localhost [] Id: 3

# Query\_time: 6.000552 Lock\_time: 0.000000 Rows\_sent: 1 Rows\_examined: 0

SET timestamp=1551253647;

select sleep(6);

[root@client-50 ~]# mysqldumpslow /var/lib/mysql/client-50-slow.log

#统计日志记录信息

Reading mysql slow query log from /var/lib/mysql/client-50-slow.log

Count: 1 Time=0.00s (0s) Lock=0.00s (0s) Rows=0.0 (0), 0users@0hosts

# Time: N-N-27T07:N:N.910641Z

# User@Host: root[root] @ localhost [] Id: N

# Query\_time: N.N Lock\_time: N.N Rows\_sent: N Rows\_examined: N

SET timestamp=N;

select sleep(N)

[root@client-50 ~]# mysqldumpslow /var/lib/mysql/client-50-slow.log >sql.txt

案例1：实现MySQL读写分离

案例2：配置MySQL多实例

案例3：MySQL性能优化

1 案例1：实现MySQL读写分离

1.1 问题

搭建一主一从结构

配置maxscale代理服务器

测试分离配置

1.2 方案

使用4台RHEL 7虚拟机，如图-1所示。其中192.168.4.10和192.168.4.20，分别提供读、写服务，均衡流量，通过主从复制保持数据一致性，由MySQL代理192.168.4.100面向客户端，收到SQL写请求时，交给服务器A处理，收到SQL读请求时，交给服务器B处理。linux客户机用于测试配置，可以使用真机代替

图－1

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：搭建主从

1）搭建一主一从结构，主库192.168.4.10上面操作

[root@master10 ~]# vim /etc/my.cnf

[mysqld]

server\_id=10 //指定服务器ID号

log-bin=master10 //启用binlog日志，并指定文件名前缀

...

[root@master10 ~]# systemctl restart mysqld //重启mysqld

2）从库192.168.4.20上面操作

[mysqld]

server\_id=20 //指定服务器ID号，不要与Master的相同

log-bin=slave20 //启动SQL日志，并指定文件名前缀

read\_only=1 //只读模式

...

[root@slave20 ~]# systemctl restart mysqld

3）主库授权一个用户并查看master的状态

[root@master10 ~]# mysql -u root -p123456

mysql> grant all on \*.\* to 'replicater'@'%' identified by '123456';

Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> show master status;

+-----------------+----------+--------------+------------------+-------------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

+-----------------+----------+--------------+------------------+-------------------+

| master10.000002 | 449 | | | |

+-----------------+----------+--------------+------------------+-------------------+

1 row in set (0.00 sec)

4）从库通过CHANGE MASTER语句指定MASTER服务器的IP地址、同步用户名/密码、起始日志文件、偏移位置（参考MASTER上的状态输出）

[root@slave20 ~]# mysql -u root -p123456

mysql> change master to master\_host='192.168.4.10',

-> master\_user='replicater',

-> master\_password='123456',

-> master\_log\_file='master10.000002',

-> master\_log\_pos=738;

Query OK, 0 rows affected, 2 warnings (0.01 sec)

mysql> start slave;

Query OK, 0 rows affected (0.01 sec)

mysql> show slave status\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Slave\_IO\_State: Waiting for master to send event

Master\_Host: 192.168.4.10

Master\_User: replicater

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: master10.000002

Read\_Master\_Log\_Pos: 738

Relay\_Log\_File: slave20-relay-bin.000002

Relay\_Log\_Pos: 319

Relay\_Master\_Log\_File: master10.000002

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

Replicate\_Do\_DB:

Replicate\_Ignore\_DB:

Replicate\_Do\_Table:

Replicate\_Ignore\_Table:

Replicate\_Wild\_Do\_Table:

Replicate\_Wild\_Ignore\_Table:

Last\_Errno: 0

Last\_Error:

Skip\_Counter: 0

Exec\_Master\_Log\_Pos: 738

Relay\_Log\_Space: 528

Until\_Condition: None

Until\_Log\_File:

Until\_Log\_Pos: 0

Master\_SSL\_Allowed: No

Master\_SSL\_CA\_File:

Master\_SSL\_CA\_Path:

Master\_SSL\_Cert:

Master\_SSL\_Cipher:

Master\_SSL\_Key:

Seconds\_Behind\_Master: 0

Master\_SSL\_Verify\_Server\_Cert: No

Last\_IO\_Errno: 0

Last\_IO\_Error:

Last\_SQL\_Errno: 0

Last\_SQL\_Error:

Replicate\_Ignore\_Server\_Ids:

Master\_Server\_Id: 10

Master\_UUID: 95ada2c2-bb24-11e8-abdb-525400131c0f

Master\_Info\_File: /var/lib/mysql/master.info

SQL\_Delay: 0

SQL\_Remaining\_Delay: NULL

Slave\_SQL\_Running\_State: Slave has read all relay log; waiting for more updates

Master\_Retry\_Count: 86400

Master\_Bind:

Last\_IO\_Error\_Timestamp:

Last\_SQL\_Error\_Timestamp:

Master\_SSL\_Crl:

Master\_SSL\_Crlpath:

Retrieved\_Gtid\_Set:

Executed\_Gtid\_Set:

Auto\_Position: 0

Replicate\_Rewrite\_DB:

Channel\_Name:

Master\_TLS\_Version:

1 row in set (0.00 sec)

5）测试，主库创建aa库

mysql> create database aa;

Query OK, 1 row affected (0.00 sec)

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| aa |

| mysql |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.00 sec)

6）从库上面查看，有aa库

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| aa |

| mysql |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.00 sec)

步骤二：实现mysql读写分离

1）配置数据读写分离服务器192.168.4.100

环境准备关闭防火墙和SElinux，保证yum源可以正常使用

[root@maxscale ~]# cd mysql/

[root@maxscale mysql]# ls

maxscale-2.1.2-1.rhel.7.x86\_64.rpm

[root@maxscale mysql]# rpm -ivh maxscale-2.1.2-1.rhel.7.x86\_64.rpm

//安装maxscale

warning: maxscale-2.1.2-1.rhel.7.x86\_64.rpm: Header V4 RSA/SHA1 Signature, key ID 8167ee24: NOKEY

Preparing... ################################# [100%]

Updating / installing...

1:maxscale-2.1.2-1 ################################# [100%]

2）配置maxscale

[root@maxscale mysql]# vim /etc/maxscale.cnf.template

[maxscale]

threads=auto //运行的线程的数量

[server1] //定义数据库服务器

type=server

address=192.168.4.10 //数据库服务器的ip

port=3306

protocol=MySQLBackend //后端数据库

[server2]

type=server

address=192.168.4.20

port=3306

protocol=MySQLBackend

[MySQL Monitor] //定义监控的数据库服务器

type=monitor

module=mysqlmon

servers=server1, server2 //监控的数据库列表，不能写ip

user=scalemon //监视数据库服务器时连接的用户名scalemon

passwd=123456 //密码123456

monitor\_interval=10000 //监视的频率 单位为秒

#[Read-Only Service] //不定义只读服务器

#type=service

#router=readconnroute

#servers=server1

#user=myuser

#passwd=mypwd

#router\_options=slave

[Read-Write Service] //定义读写分离服务

type=service

router=readwritesplit

servers=server1, server2

user=maxscaled //用户名 验证连接代理服务时访问数据库服务器的用户是否存在

passwd=123456 //密码

max\_slave\_connections=100%

[MaxAdmin Service] //定义管理服务

type=service

router=cli

#[Read-Only Listener] //不定义只读服务使用的端口号

#type=listener

#service=Read-Only Service

#protocol=MySQLClient

#port=4008

[Read-Write Listener] //定义读写服务使用的端口号

type=listener

service=Read-Write Service

protocol=MySQLClient

port=4006

[MaxAdmin Listener] //管理服务使用的端口号

type=listener

service=MaxAdmin Service

protocol=maxscaled

socket=default

port=4099 //手动添加，不指定时使用的是默认端口在启动服务以后可以知道默认端口是多少

3）根据配置文件的设置，在数据库服务器上添加授权用户（主库执行，从库查看）

mysql> grant replication slave,replication client on \*.\* to scalemon@'%' identified by "123456"; //监控数据库服务器时，连接数据库服务器的用户

Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> grant select on mysql.\* to maxscaled@"%" identified by "123456";

//验证 访问数据时，连接数据库服务器使用的用户，是否在数据库服务器上存在的，连接用户

Query OK, 0 rows affected, 1 warning (0.01 sec)

4）查看授权用户

在主库上面查看

mysql> select user,host from mysql.user where user in ("scalemon","maxscaled");

+-----------+------+

| user | host |

+-----------+------+

| maxscaled | % |

| scalemon | % |

+-----------+------+

2 rows in set (0.00 sec)

在从库上面查看

mysql> select user,host from mysql.user where user in ("scalemon","maxscaled");

+-----------+------+

| user | host |

+-----------+------+

| maxscaled | % |

| scalemon | % |

+-----------+------+

2 rows in set (0.00 sec)

测试授权用户

[root@maxscale mysql]# mysql -h 192.168.4.10 -u scalemon -p123456

[root@maxscale mysql]# mysql -h 192.168.4.20 -u scalemon -p123456

[root@maxscale mysql]# mysql -h 192.168.4.10 -u maxscaled -p123456

[root@maxscale mysql]# mysql -h 192.168.4.20 -u maxscaled -p123456

5）启动服务

[root@maxscale ~]# maxscale -f /etc/maxscale.cnf

[root@maxscale ~]# ps -C maxscale //查看进程

PID TTY TIME CMD

17930 ? 00:00:00 maxscale

[root@maxscale ~]# netstat -antup | grep maxscale //查看端口

tcp6 0 0 :::4099 :::\* LISTEN 17930/maxscale

tcp6 0 0 :::4006 :::\* LISTEN 17930/maxscale

6）测试，在本机访问管理端口查看监控状态

maxadmin -P端口 -u用户名 -p密码

[root@maxscale ~]# maxadmin -P4099 -uadmin -pmariadb

MaxScale>

MaxScale> list servers

Servers.

-------------------+-----------------+-------+-------------+--------------------

Server | Address | Port | Connections | Status

-------------------+-----------------+-------+-------------+--------------------

server1 | 192.168.4.10 | 3306 | 0 | Master, Running

server2 | 192.168.4.20 | 3306 | 0 | Slave, Running

-------------------+-----------------+-------+-------------+--------------------

7）在客户端访问读写分离服务器（没有mysql命令可以安装）

mysql -h读写分离服务ip -P4006 -u用户名 -p密码

[root@slave53 ~]# mysql -h192.168.4.100 -P4006 -ureplicater -p123456

mysql> select @@hostname; //查看当前主机名

+------------+

| @@hostname |

+------------+

| slave20 |

+------------+

1 row in set (0.00 sec)

mysql> create table t2(id int(4) );

Query OK, 0 rows affected (0.02 sec)

mysql> insert into aa.t2 values(777);

Query OK, 1 row affected (0.01 sec)

在主库上面查看

mysql> use aa

mysql> select \* from t2;

+------+

| id |

+------+

| 777 |

+------+

1 row in set (0.00 sec)

从库（主库同步到从库）

mysql> use aa

mysql> select \* from t2;

+------+

| id |

+------+

| 777 |

+------+

1 row in set (0.00 sec)

2 案例2：配置MySQL多实例

2.1 问题

在主机192.168.4.56上，配置第1个MySQL实例

实例名称mysql1、端口3307

数据库目录/data3307、pid文件mysql1.pid

错误日志mysql1.err

在主机192.168.4.56上，配置第2个MySQL实例

实例名称mysql2、端口3308

数据库目录/data3308、pid文件mysql2.pid

错误日志mysql2.err

步骤一：配置多实例（192.168.4.56上面操作）

什么是多实例：

在一台物理主机上运行多个数据库服务，可以节约运维成本，提高硬件利用率

1）解压软件、修改目录名

[root@mysql ~]# cd mysql/

[root@mysql mysql]# ls

mysql-5.7.20-linux-glibc2.12-x86\_64.tar.gz

[root@mysql mysql]# tar -xf mysql-5.7.20-linux-glibc2.12-x86\_64.tar.gz

[root@mysql mysql]# mv mysql-5.7.20-linux-glibc2.12-x86\_64 /usr/local/mysql

2）调整PATH变量

[root@mysql mysql]# echo "export PATH=/usr/local/mysql/bin:$PATH" \

>> /etc/profile

[root@mysql mysql]# source /etc/profile

[root@mysql mysql]# echo $PATH

/usr/local/mysql/bin:/usr/local/mycat/bin:/usr/local/mycat/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin:/root/bin

3）编辑主配置文件/etc/my.cnf

每个实例要有独立的数据库目录、监听端口号、实例名称和独立的sock文件

[mysqld\_multi] //启用多实例

mysqld = /usr/local/mysql/bin/mysqld\_safe //指定进程文件路径

mysqladmin = /usr/local/mysql/bin/mysqladmin //指定管理命令路径

user = root //指定进程用户

[mysqld1] //实例进程名称

port=3307 //端口号

datadir=/data3307 //数据库目录 ，要手动创建

socket=/data3307/mysqld.sock //指定sock文件的路径和名称

pid-file=/data3307/mysql1.pid //进程pid号文件位置

log-error=/data3307/mysql1.err //错误日志位置

[mysqld2]

port=3308

datadir=/data3308

socket=/data3308/mysqld.sock

pid-file=/data3308/mysql2.pid

log-error=/data3308/mysql2.err

4）创建数据库目录

[root@mysql mysql]# mkdir -p /data3307

[root@mysql mysql]# mkdir -p /data3308

5）创建进程运行的所有者和组 mysql

[root@mysql mysql]# useradd mysql

[root@mysql mysql]# chown mysql:mysql /data\*

6）初始化授权库

[root@mysql mysql]# mysqld --user=mysql --basedir=/usr/local/mysql

--datadir=/data3307 --initialize

...

2018-09-26T07:07:33.443378Z 1 [Note] A temporary password is generated for root@localhost: 7L?Vi!dGKmgu //root用户登录的初始化密码

[root@mysql mysql]# mysqld --user=mysql --basedir=/usr/local/mysql

--datadir=/data3308 --initialize

...

2018-09-26T07:08:07.770289Z 1 [Note] A temporary password is generated for root@localhost: kC)BbyUp1a-b //root用户登录的初始化密码

7）启动多实例

[root@mysql mysql]# mysqld\_multi start 1 //1为实例编号

[root@mysql mysql]# mysqld\_multi start 2

8）查看端口

[root@mysql mysql]# netstat -utnlp | grep :3307

tcp6 0 0 :::3307 :::\* LISTEN 21009/mysqld

[root@mysql mysql]# netstat -utnlp | grep :3308

tcp6 0 0 :::3308 :::\* LISTEN 21177/mysqld

[root@mysql mysql]# ps -C mysqld

PID TTY TIME CMD

21009 pts/1 00:00:00 mysqld

21177 pts/1 00:00:00 mysqld

9）访问多实例

使用初始化密码登录多实例1

[root@mysql mysql]# mysql -u root -p'7L?Vi!dGKmgu' -S /data3307/mysqld.sock

mysql> alter user root@"localhost" identified by '123456'; //修改密码

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| mysql |

| performance\_schema |

| sys |

+--------------------+

4 rows in set (0.00 sec)

使用初始化密码登录多实例2

[root@mysql bin]# mysql -u root -p'kC)BbyUp1a-b' -S /data3307/mysqld.sock

mysql> alter user root@"localhost" identified by '123456'; //修改密码

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| mysql |

| performance\_schema |

| sys |

+--------------------+

4 rows in set (0.00 sec)

10）创建库

mysql> create database db1;

Query OK, 1 row affected (0.00 sec)

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| db1 |

| mysql |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.00 sec)

11）停止启动的实例服务

mysqld\_multi --user=root --password=密码 stop 实例编号

[root@mysql mysql]# mysqld\_multi --user=root --password=123456 stop 1

[root@mysql mysql]# netstat -utnlp | grep :3307 //查看没有端口

[root@mysql mysql]# mysqld\_multi --user=root --password=123456 stop 2

[root@mysql mysql]# netstat -utnlp | grep :3308 //查看没有端口

[root@mysql mysql]# mysql -uroot -p123456 -S /data3307/mysqld.sock

//拒绝连接

mysql: [Warning] Using a password on the command line interface can be insecure.

ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/data3307/mysqld.sock' (2)

3 案例3：MySQL性能优化

3.1 问题

练习相关优化选项

启用慢查询日志

查看各种系统变量、状态变量

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：mysql性能优化

1）查看服务运行时的参数配置

mysql> show variables\G;

......

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 171. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Variable\_name: innodb\_log\_file\_size

Value: 50331648

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 172. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Variable\_name: innodb\_log\_files\_in\_group

Value: 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 173. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Variable\_name: innodb\_log\_group\_home\_dir

Value: ./

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 174. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Variable\_name: innodb\_log\_write\_ahead\_size

Value: 8192

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 175. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Variable\_name: innodb\_lru\_scan\_depth

Value: 1024

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 176. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Variable\_name: innodb\_max\_dirty\_pages\_pct

Value: 75.000000

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 177. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Variable\_name: innodb\_max\_dirty\_pages\_pct\_lwm

Value: 0.000000

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 178. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Variable\_name: innodb\_max\_purge\_lag

Value: 0

......

mysql> show variables like "%innodb%";

+------------------------------------------+------------------------+

| Variable\_name | Value |

+------------------------------------------+------------------------+

| ignore\_builtin\_innodb | OFF |

| innodb\_adaptive\_flushing | ON |

| innodb\_adaptive\_flushing\_lwm | 10 |

| innodb\_adaptive\_hash\_index | ON |

| innodb\_adaptive\_hash\_index\_parts | 8 |

| innodb\_adaptive\_max\_sleep\_delay | 150000 |

......

......

| innodb\_undo\_log\_truncate | OFF |

| innodb\_undo\_logs | 128 |

| innodb\_undo\_tablespaces | 0 |

| innodb\_use\_native\_aio | ON |

| innodb\_version | 5.7.17 |

| innodb\_write\_io\_threads | 4 |

+------------------------------------------+------------------------+

134 rows in set (0.01 sec)

2）并发连接数量

查看当前已经使用的连接数

mysql> flush status;

Query OK, 0 rows affected (0.00 sec)

mysql> show global status like "Max\_used\_connections";

+----------------------+-------+

| Variable\_name | Value |

+----------------------+-------+

| Max\_used\_connections | 3 |

+----------------------+-------+

1 row in set (0.00 sec)

查看默认的最大连接数

mysql> show variables like "max\_connections%";

+-----------------+-------+

| Variable\_name | Value |

+-----------------+-------+

| max\_connections | 151 |

+-----------------+-------+

1 row in set (0.00 sec)

3）连接超时时间

mysql> show variables like "%timeout%";

+-----------------------------+----------+

| Variable\_name | Value |

+-----------------------------+----------+

| connect\_timeout | 10 |

| delayed\_insert\_timeout | 300 |

| have\_statement\_timeout | YES |

| innodb\_flush\_log\_at\_timeout | 1 |

| innodb\_lock\_wait\_timeout | 50 |

| innodb\_rollback\_on\_timeout | OFF |

| interactive\_timeout | 28800 |

| lock\_wait\_timeout | 31536000 |

| net\_read\_timeout | 30 |

| net\_write\_timeout | 60 |

| rpl\_stop\_slave\_timeout | 31536000 |

| slave\_net\_timeout | 60 |

| wait\_timeout | 28800 |

+-----------------------------+----------+

13 rows in set (0.00 sec)

4）允许保存在缓存中被重用的线程数量

mysql> show variables like "thread\_cache\_size";

+-------------------+-------+

| Variable\_name | Value |

+-------------------+-------+

| thread\_cache\_size | 9 |

+-------------------+-------+

1 row in set (0.00 sec)

5）用于MyISAM引擎的关键索引缓存大小

mysql> show variables like "key\_buffer\_size";

+-----------------+---------+

| Variable\_name | Value |

+-----------------+---------+

| key\_buffer\_size | 8388608 |

+-----------------+---------+

1 row in set (0.00 sec)

6）为每个要排序的线程分配此大小的缓存空间

mysql> show variables like "sort\_buffer\_size";

+------------------+--------+

| Variable\_name | Value |

+------------------+--------+

| sort\_buffer\_size | 262144 |

+------------------+--------+

1 row in set (0.00 sec)

7）为顺序读取表记录保留的缓存大小

mysql> show variables like "read\_buffer\_size";

+------------------+--------+

| Variable\_name | Value |

+------------------+--------+

| read\_buffer\_size | 131072 |

+------------------+--------+

1 row in set (0.01 sec)

8）为所有线程缓存的打开的表的数量

mysql> show variables like "table\_open\_cache";

+------------------+-------+

| Variable\_name | Value |

+------------------+-------+

| table\_open\_cache | 2000 |

+------------------+-------+

1 row in set (0.00 sec)

步骤二：SQL查询优化

1）常用日志种类及选项，如图-1所示：

图-1

记录慢查询，图-2所示：

启用慢查询日志

[root@master10 ~]# vim /etc/my.cnf

...

slow\_query\_log=1

slow\_query\_log\_file=mysql-slow.log

long\_query\_time=5

log\_queries\_not\_using\_indexes=1

...

[root@master10 ~]# systemctl restart mysqld

2）查看慢查询日志

[root@master10 ~]# 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

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 |

+------------------------------+---------+

5 rows in set (0.00 sec)

3）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 | 0 |

| Qcache\_inserts | 0 |

| Qcache\_lowmem\_prunes | 0 |

| Qcache\_not\_cached | 40 |

| Qcache\_queries\_in\_cache | 0 |

| Qcache\_total\_blocks | 1 |

+-------------------------+---------+

8 rows in set (0.00 sec)

性能调忧DBA2\_DAY05讲

# NSD DBA2 DAY03

部署MHA集群

# 一 准备环境

**拓扑结构:**

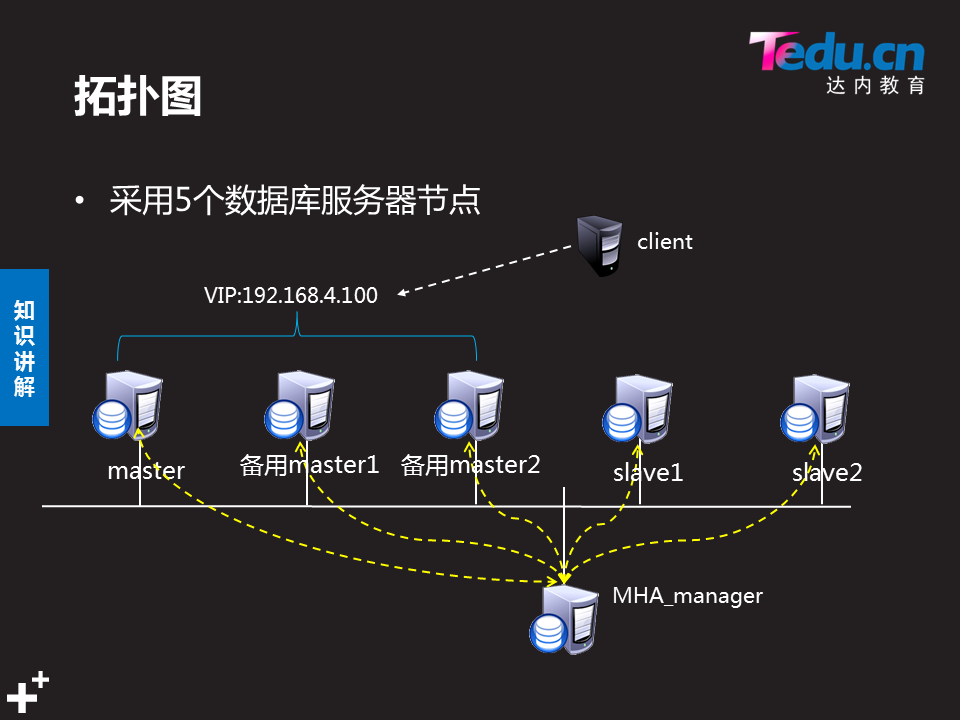
client 50 :无数据库服务

管理主机56:无数据库服务

数据库服务器51-55:恢复为独立的数据库服务器,

删除所有自定义授权用户

删除所有自定义库,只保留初始的4个库





**配置ssh无密码登录:**

所有数据库服务器之间ssh无密码登录配置

管理主机56可以免密登录所有数据库服务器

命令参考:生成密码Ssh-keygen 发送秘钥Ssh-copy-id

**安装依赖软件:**

在所有主机51-56:安装yum -y install perl-\*

**将集群软件拷贝到51-56:**

软件 mha-soft-student

**部署mysql 一主多从 主从结构:**

主库:192.168.4.51 做主库,宕机之后变成从库

启用binlog日志,用户授权,并开启主从同步2种角色半同步功能

从库:192.168.4.52 做备用主库

启用binlog日志,用户授权,并开启主从同步2种角色半同步功能

从库:192.168.4.53 做备用主库

启用binlog日志,用户授权,并开启主从同步2种角色半同步功能

从库:192.168.4.54 纯从库

指定server\_id 指定主库信息 启用从角色半同步角色

从库:192.168.4.55 纯从库

指定server\_id 指定主库信息 启用从角色半同步角色

## 1.1主从配置命令参考:

### 1.1.1主库服务器配置

Vim /etc/my.cnf

[mysqld]

server\_id=51

log\_bin=51log #启用binlog日志

**用户连接授权配置命令:**

mysql>grant replication slave on \*.\* to repluser@"%" identified by "123456";

mysql> show grants for repluser;

**查看binlog日志信息**

mysql> show master status; #查看日志信息

+-----------------+----------+--------------+------------------+-------------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

+-----------------+----------+--------------+------------------+-------------------+

| master51.000005 | 441 | | | |

+-----------------+----------+--------------+------------------+-------------------+

1 row in set (0.00 sec)

### 1.1.2从库服务器配置:

mysql> change master to

master\_host="192.168.4.51", #主服务器IP地址

master\_user="repluser", #主服务器授权的用户

master\_password="123456", #授权用户的密码

master\_log\_file="master51.000001", #日志文件

master\_log\_pos=441; #偏移位置

mysql> start slave; #启用从服务

mysql> stop slave; #停止从服务

mysql> show slave status\G; #查看从服务配置

### 1.1.3主从半同步复制模式 相关配置:

**查看是否允许动态加载模块:**

mysql>show variables like 'have\_dynamic\_loading';

**查看模块是否被加载:**

mysql>select plugin\_name,plugin\_status from information\_schema.plugins where plugin\_name like '%semi%';

**查看模块是否启用:**

mysql> show variables like "rpl\_semi\_sync\_%\_enabled";

**主库加载半同步复制模块:**

mysql>Install plugin rpl\_semi\_sync\_master SONAME 'semisync\_master.so';

**主库启用半同步复制模块:**

mysql>set global rpl\_semi\_sync\_master\_enabled = 1

**从库加载半同步复制模块**

mysql> Install plugin rpl\_semi\_sync\_slave soname 'semisync\_slave.so';

**从库启用半同步复制模块**

mysql> set global rpl\_semi\_sync\_slave\_enabled = 1;

### 1.1.4永久配置半同步复制模式

Vim /etc/my.cnf

[mysqld]

plugin-load=rpl\_semi\_sync\_master=semisync\_master.so

#加载主库的半同步复制模式

rpl\_semi\_sync\_master\_enabled=1

#启用主库的半同步复制模式

plugin-load=rpl\_semi\_sync\_slave=semisync\_slave.so

#加载从库的半同步复制模式

rpl\_semi\_sync\_slave\_enabled=1

#启用从库的半同步复制模式

**查看是否允许动态加载模块:**

mysql>show variables like 'have\_dynamic\_loading';

查看模块是否被加载:

mysql>select plugin\_name,plugin\_status from information\_schema.plugins where plugin\_name like '%semi%';

查看模块是否启用:

mysql> show variables like "rpl\_semi\_sync\_%\_enabled";

# 二 集群:

## 2.1 MHA软件介绍

MHA由perl语言编写而来.所以所有主机都要安装

yum -y install perl-\*

* 由日本DeNA公司youshimaton开发
* 数据库的自动故障切换操作能做到在0-30秒内完成
* 能确保在切换过程中保证数据的一致性,达到真正意义上的高可用
* Keepalivd能做所有服务的高可用集群,MHA只能做mysql数据库高可用集群

MHA Manager(管理节点)

可以单独部署在一台独立机器上管理其他节点

也可部署在其中一台slave 节点上

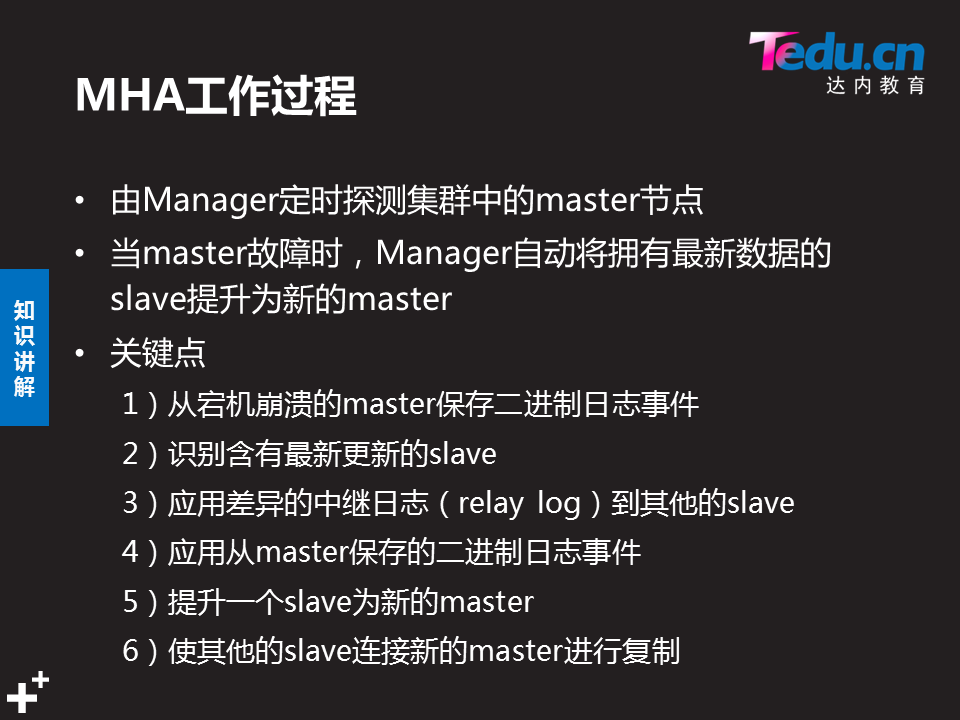
本案例的56

MHA node(数据节点)

运行着数据库服务的mysql服务器

本案例的51-55

MHA工作过程:



## 2.2 配置集群

### 2.1 在所有主机51-56上安装软件包(以51为例)

mha-soft-student.zip 需要传送安装包

[root@sql51 ~]# unzip mha-soft-student.zip

[root@sql51 ~]# cd mha-soft-student/

[root@sql51 mha-soft-student]# yum -y install perl-\*.rpm

之前安装的是yum源上的所有perl-\*.rpm,这个是额外的几个包

### 2.2 配置管理主机56

#### 2.2.1安装提供管理服务的软件包

**必须先装数据节点安装包:**mha4mysql-node-0.56-0.el6.noarch.rpm

[root@sql56 mha-soft-student]# rpm -ivh mha4mysql-node-0.56-0.el6.noarch.rpm

**再安装管理包:**mha4mysql-manager-0.56.tar.gz

[root@sql56 mha-soft-student]# tar -xvf mha4mysql-manager-0.56.tar.gz

[root@sql56 mha-soft-student]# ls

mha4mysql-manager-0.56

[root@sql56 mha-soft-student]# cd mha4mysql-manager-0.56/

[root@sql56 mha4mysql-manager-0.56]# ls

AUTHORS COPYING inc Makefile.PL META.yml rpm t

bin debian lib MANIFEST README samples tests

[root@sql56 mha4mysql-manager-0.56]# which perl #查看perl命令

/usr/bin/perl

[root@sql56 mha4mysql-manager-0.56]# perl Makefile.PL

\*\*\* Module::AutoInstall version 1.03

\*\*\* Checking for Perl dependencies...

[Core Features]

- DBI ...loaded. (1.627) #括号内不为0,表示依赖包被安装

- DBD::mysql ...loaded. (4.023) #括号内不为0,表示依赖包被安装

- Time::HiRes ...loaded. (1.9725) #括号内不为0,表示依赖包被安装

- Config::Tiny ...loaded. (2.14) #括号内不为0,表示依赖包被安装

- Log::Dispatch ...loaded. (2.41) #括号内不为0,表示依赖包被安装

- Parallel::ForkManager ...loaded. (1.18) #括号内不为0,表示依赖包被安装

- MHA::NodeConst ...loaded. (0.56) #括号内不为0,表示依赖包被安装

\*\*\* Module::AutoInstall configuration finished.

Checking if your kit is complete...

Looks good

Writing Makefile for mha4mysql::manager

Writing MYMETA.yml and MYMETA.json

[root@sql56 mha4mysql-manager-0.56]# make

[root@sql56 mha4mysql-manager-0.56]# make install

[root@sql56 mha4mysql-manager-0.56]# masterha\_ #连续tab,显示全部命令

#### 2.2.2创建服务的主配置文件

**参考模板app1.cnf**

[root@sql56~]#cp /root/mha-soft-student/mha4mysql-manager-0.56/samples/conf/app1.cnf /etc/mha\_manager/ #拷贝模板到这个目录(目录需要手动创建)

[root@sql56 ~t]# ls /etc/mha\_manager

app1.cnf

[root@sql56 mha\_manager]# pwd

/etc/mha\_manager

[root@sql56 mha\_manager]# vim app1.cnf

[server default] #主要内容如下

manager\_workdir=/etc/mha\_manager #指定的工作目录

manager\_log=/etc/mha\_manager/manager.log #日志文件所在目录以及名字

master\_ip\_failover\_script=/usr/local/bin/master\_ip\_failover #加载故障切换脚本

ssh\_user=root #检测无密码验证

ssh\_port=22 #免密登录端口

repl\_user=repluser #主从同步用户名

repl\_password=123456

user=root #监控数据库用户名56用来监视51-55

password=123456

[server1]

hostname=192.168.4.51

candidate\_master=1 #设置为候选master

[server2]

hostname=192.168.4.52

candidate\_master=1 #设置为候选master

[server3]

hostname=192.168.4.53

candidate\_master=1 #设置为候选master

[server4]

hostname=192.168.4.54

no\_master=1 #不竞选master

[server5]

hostname=192.168.4.55

no\_master=1 #不竞选master

#### 2.2.3部署故障切换脚本

在管理机主配置文件中如下语句里指定了故障切换脚本的目录与名称:

master\_ip\_failover\_script=/usr/local/bin/master\_ip\_failover #加载故障切换脚本

将脚本放置/usr/local/bin/目录中.并且命名为:master\_ip\_failover

[root@sql56 ~]# vim /usr/local/bin/master\_ip\_failover

**--------新脚本内容------------------------------------------------**

#!/usr/bin/env perl

# Copyright (C) 2011 DeNA Co.,Ltd.

#

# This program is free software; you can redistribute it and/or modify

# it under the terms of the GNU General Public License as published by：

# the Free Software Foundation; either version 2 of the License, or

# (at your option) any later version.

#

# This program is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

# GNU General Public License for more details.

#

# You should have received a copy of the GNU General Public License

# along with this program; if not, write to the Free Software

# Foundation, Inc.,

# 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

## Note: This is a sample script and is not complete. Modify the script based on your environment.

use strict;

use warnings FATAL => 'all';

use Getopt::Long;

use MHA::DBHelper;

my (

$command, $ssh\_user, $orig\_master\_host,

$orig\_master\_ip, $orig\_master\_port, $new\_master\_host,

$new\_master\_ip, $new\_master\_port, $new\_master\_user,

$new\_master\_password

);

my $vip = '192.168.4.100/24'; # Virtual IP #此处修改实际的VIP

my $key = "1";

my $ssh\_start\_vip = "/sbin/ifconfig eth0:$key $vip";

my $ssh\_stop\_vip = "/sbin/ifconfig eth0:$key down";

GetOptions(

'command=s' => \$command,

'ssh\_user=s' => \$ssh\_user,

'orig\_master\_host=s' => \$orig\_master\_host,

'orig\_master\_ip=s' => \$orig\_master\_ip,

'orig\_master\_port=i' => \$orig\_master\_port,

'new\_master\_host=s' => \$new\_master\_host,

'new\_master\_ip=s' => \$new\_master\_ip,

'new\_master\_port=i' => \$new\_master\_port,

'new\_master\_user=s' => \$new\_master\_user,

'new\_master\_password=s' => \$new\_master\_password,

);

exit &main();

sub main {

if ( $command eq "stop" || $command eq "stopssh" ) {

# $orig\_master\_host, $orig\_master\_ip, $orig\_master\_port are passed.

# If you manage master ip address at global catalog database,

# invalidate orig\_master\_ip here.

my $exit\_code = 1;

eval {

# updating global catalog, etc

&stop\_vip();

$exit\_code = 0;

};

if ($@) {

warn "Got Error: $@\n";

exit $exit\_code;

}

exit $exit\_code;

}

elsif ( $command eq "start" ) {

# all arguments are passed.

# If you manage master ip address at global catalog database,

# activate new\_master\_ip here.

# You can also grant write access (create user, set read\_only=0, etc) here.

my $exit\_code = 10;

eval {

my $new\_master\_handler = new MHA::DBHelper();

# args: hostname, port, user, password, raise\_error\_or\_not

$new\_master\_handler->connect( $new\_master\_ip, $new\_master\_port,

$new\_master\_user, $new\_master\_password, 1 );

## Set read\_only=0 on the new master

$new\_master\_handler->disable\_log\_bin\_local();

print "Set read\_only=0 on the new master.\n";

$new\_master\_handler->disable\_read\_only();

## Creating an app user on the new master

print "Creating app user on the new master..\n";

$new\_master\_handler->enable\_log\_bin\_local();

$new\_master\_handler->disconnect();

## Update master ip on the catalog database, etc

&start\_vip();

$exit\_code = 0;

};

if ($@) {

warn $@;

# If you want to continue failover, exit 10.

exit $exit\_code;

}

exit $exit\_code;

}

elsif ( $command eq "status" ) {

# do nothing

exit 0;

}

else {

&usage();

exit 1;

}

}

sub start\_vip() {

`ssh $ssh\_user\@$new\_master\_host \" $ssh\_start\_vip \"`;

}

sub stop\_vip() {

return 0 unless ($ssh\_user);

`ssh $ssh\_user\@$orig\_master\_host \" $ssh\_stop\_vip \"`;

}

sub usage {

print

"Usage: master\_ip\_failover --command=start|stop|stopssh|status --orig\_master\_host=host --orig\_master\_ip=ip --orig\_master\_port=port --new\_master\_host=host --new\_master\_ip=ip --new\_master\_port=port\n";

}

---------------------------------------------------------------

#### 2.2.4在主库服务器51上配置VIP

把vip 192.168.4.100配置当前主数据库服务器192.168.4.51上

[root@sql51 ~]# ifconfig eth0:1 192.168.4.100

[root@sql51 ~]# ifconfig eth0:1

eth0:1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.4.100 netmask 255.255.255.0 broadcast 192.168.4.255

ether 52:54:00:57:34:54 txqueuelen 1000 (Ethernet)

#### 2.2.5 管理主机56检测配置文件

**证ssh 免密登陆数据节点主机**

[root@sql56 mha\_manager]# cd /usr/local/bin/

[root@sql56 bin]# ls

master\_ip\_failover

[root@sql56 bin]# masterha\_check\_ssh --conf=/etc/mha\_manager/app1.cnf

Fri Feb 22 14:31:21 2019 - [info] All SSH connection tests passed successfully.

**验证数据节点的主从同步配置**

[root@sql56 mha\_manager]# cd /usr/local/bin/

[root@sql56 bin]# masterha\_check\_repl --conf=/etc/mha\_manager/app1.cnf

Fri Feb 22 19:31:00 2019 - [info] Alive Servers: #数据库服务器列表

Fri Feb 22 19:31:00 2019 - [info] 192.168.4.51(192.168.4.51:3306)

Fri Feb 22 19:31:00 2019 - [info] 192.168.4.52(192.168.4.52:3306)

Fri Feb 22 19:31:00 2019 - [info] 192.168.4.53(192.168.4.53:3306)

Fri Feb 22 19:31:00 2019 - [info] 192.168.4.54(192.168.4.54:3306)

Fri Feb 22 19:31:00 2019 - [info] 192.168.4.55(192.168.4.55:3306)

Fri Feb 22 19:31:00 2019 - [info] Alive Slaves: #存活的从服务器

Fri Feb 22 19:31:00 2019 - [info] 192.168.4.52(192.168.4.52:3306) Version=5.7.17-log (oldest major version between slaves) log-bin:enabled

Fri Feb 22 19:31:00 2019 - [info] Replicating from 192.168.4.51(192.168.4.51:3306)

验证数据节点时可以看到有哪些数据库服务器,以及目前主库服务器是192.168.4.51,

MySQL Replication Health is OK. #此结果显示成功

#如果显示 MySQL Replication Health is NOT OK! 不成功

**验证数据节点的主从同步配置遇到过的问题:**

**使用旧脚本**

[root@sql56 ~]# vim /etc/mha\_manager/app1.cnf

master\_ip\_failover\_script=/usr/local/bin/master\_ip\_failover #加载故障切换脚本

[root@sql56 ~]# vim /usr/local/bin/master\_ip\_failover

#FIXME\_xxx;

**以上两个配置文件中的两条语句只要注释其中一条即可**

要么注释master\_ip\_failover\_script=/usr/local/bin/master\_ip\_failover

要么注释 FIXME\_xxx;

最终更换master\_ip\_failover 文件,使用新的脚本文件无此问题

#### 2.2.6启动管理服务MHA\_Manager

--remove\_dead\_master\_conf //删除宕机主库配置

当前主库宕机后,把主库配置从配置文件中删除

--ignore\_last\_failover //忽略xxx.health文件

默认8个小时内主库连续宕机,不切换主库,如果添加 --ignore\_last\_failover,则8小时内主库宕机了,就会切换

**启动:**

[root@sql56 ~]# masterha\_manager --conf=/etc/mha\_manager/app1.cnf --remove\_dead\_master\_conf --ignoer\_last\_failover

Fri Feb 22 15:36:13 2019 - [warning] Global configuration file /etc/masterha\_default.cnf not found. Skipping.

Fri Feb 22 15:36:13 2019 - [info] Reading application default configuration from /etc/mha\_manager/app1.cnf..

Fri Feb 22 15:36:13 2019 - [info] Reading server configuration from /etc/mha\_manager/app1.cnf..

启动信息会占用一个终端显示,,,,此时不能ctrl + c

新开一个终端查看运行状态

[root@sql56 ~]# masterha\_check\_status --conf=/etc/mha\_manager/app1.cnf

app1 (pid:13953) is running(0:PING\_OK), master:192.168.4.51

当前主库:192.168.4.51

#### 2.2.7查看服务状态

[root@sql56 ~]# masterha\_check\_status --conf=/etc/mha\_manager/app1.cnf

app1 (pid:13953) is running(0:PING\_OK), master:192.168.4.51

当前主库:192.168.4.51

### 2.3配置数据库服务器51-55:

1授权监控用户,在51上做,52-55自动同步

mysql> grant all on \*.\* to root@"%" identified by "123456";

Query OK, 0 rows affected, 1 warning (0.03 sec)

2在51-55上安装mha4mysql-node-0.56-0.el6.noarch.rpm(数据服务器节点)

[root@sql51 ~]# cd mha-soft-student/

[root@sql51 mha-soft-student]# ls

mha4mysql-manager-0.56-0.el6.noarch.rpm

mha4mysql-manager-0.56.tar.gz

mha4mysql-node-0.56-0.el6.noarch.rpm

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

[root@sql51 mha-soft-student]# rpm -ivh mha4mysql-node-0.56-0.el6.noarch.rpm

3设置不自动删除本机的中继日志文件

Vim /etc/my.cnf

[mysqld]

server\_id=55

relay\_log\_purge=off #不自动删除本机的中继日志文件

mysql> set global relay\_log\_purge=off;

### 2.4测试集群配置

#### 2.4.1在主库51上授权访问客户,供客户端连接使用

mysql> grant select,insert on bbsdb.\* to yaya@"%" identified by "123456";

Query OK, 0 rows affected, 1 warning (0.04 sec)

mysql> create database bbsdb;

Query OK, 1 row affected (0.04 sec)

mysql> create table bbsdb.t1(id int);

Query OK, 0 rows affected (0.26 sec)

#### 2.4.2客户端50访问vip连接数据库服务,存取数据

[root@client-50 ~]# mysql -h192.168.4.100 -uyaya -p123456;

mysql> insert into bbsdb.t1 values(11);

Query OK, 1 row affected (0.06 sec)

mysql> insert into bbsdb.t1 values(22);

Query OK, 1 row affected (0.11 sec)

mysql> select \* from bbsdb.t1;

+------+

| id |

+------+

| 11 |

| 22 |

+------+

2 rows in set (0.01 sec)

[root@sql51 ~]# mysql -uroot -p123456

mysql> select \* from bbsdb.t1;

+------+

| id |

+------+

| 11 |

| 22 |

+------+

2 rows in set (0.01 sec)

#### 2.4.3测试高可用:目前主库51故障

主库服务器51宕机后50访问VIP继续使用数据库

[root@sql51 bin]# systemctl stop mysqld.service

[root@sql56 mha\_manager]# masterha\_manager --conf=/etc/mha\_manager/app1.cnf --remove\_dead\_master\_conf --igno\_last\_failover

Fri Feb 22 17:00:45 2019 - [warning] Global configuration file /etc/masterha\_default.cnf not found. Skipping.

Fri Feb 22 17:00:45 2019 - [info] Reading application default configuration from /etc/mha\_manager/app1.cnf..

Fri Feb 22 17:00:45 2019 - [info] Reading server configuration from /etc/mha\_manager/app1.cnf..

Creating /var/tmp if not exists.. Ok. #切换开始

Checking output directory is accessible or not..

ok.

Binlog found at /var/lib/mysql, up to 51log.000007

Fri Feb 22 17:03:01 2019 - [warning] Global configuration file /etc/masterha\_default.cnf not found. Skipping.

Fri Feb 22 17:03:01 2019 - [info] Reading application default configuration from /etc/mha\_manager/app1.cnf..

Fri Feb 22 17:03:01 2019 - [info] Reading server configuration from /etc/mha\_manager/app1.cnf..

[root@sql56 mha\_manager]# #切换中以上状态会自动结束,到此处就切换完成

**检查1:**

50登录192.168.4.100成功,能插入数据

[root@client-50 ~]# mysql -h192.168.4.100 -uyaya -p123456;

mysql> insert into bbsdb.t1 values(33);

Query OK, 1 row affected (0.08 sec)

mysql> insert into bbsdb.t1 values(44);

Query OK, 1 row affected (0.66 sec)

**检查2:**

所有的主库配置自动由51变成52

mysql> show slave status\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Slave\_IO\_State: Waiting for master to send event

Master\_Host: 192.168.4.52

Master\_User: repluser

Master\_Port: 3306

.....

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

**问题:**停止51:192.168.4.100未转移删除了app1.failover.error日志文件恢复

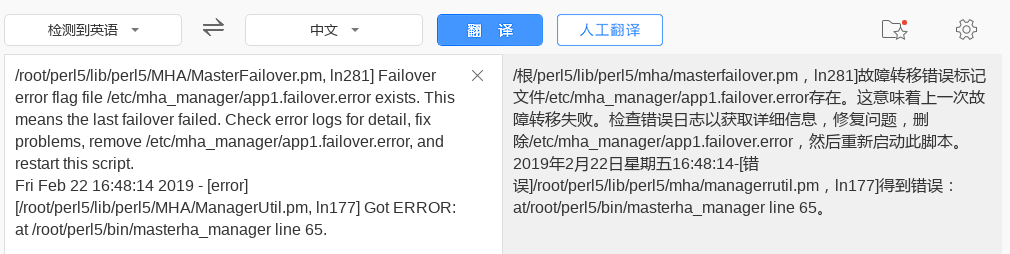
[root@sql56 mha\_manager]# ls

app1.cnf app1.failover.error manager.log

[root@sql56 mha\_manager]# vim app1.failover.error

[root@sql56 mha\_manager]# rm app1.failover.error

查看manager.log日志得知EERO报错



#### 2.4.4主机51恢复,将其加入集群,主库飘到了52,

**1恢复宕机期间的数据,从新主库52上备份数据库,放到51上恢复,**

[root@sql51 bin]# mysql -uroot -p123456

mysql> select \* from bbsdb.t1;

+------+

| id |

+------+

| 11 |

| 22 |

+------+

2 rows in set (0.01 sec)

[root@sql52 ~]# select \* form bbsdb.t1;

mysql> select \* from bbsdb.t1;

+------+

| id |

+------+

| 11 |

| 22 |

| 33 |

| 44 |

+------+

4 rows in set (0.00 sec)

[root@sql52 bbsdb]# mysqldump -uroot -p123456 bbsdb t1 >/root/bbsdbt1.sql

mysqldump: [Warning] Using a password on the command line interface can be insecure.

[root@sql51 ~]# mysql -uroot -p123456 bbsdb < bbsdbt1.sql

mysql: [Warning] Using a password on the command line interface can be insecure.

mysql> select \* from bbsdb.t1; #52上备份数据51还原之后查询数据两边一致

+------+

| id |

+------+

| 11 |

| 22 |

| 33 |

| 44 |

+------+

**2配置51为从服务器,主服务配置为现有主服务器52**

mysql> change master to

-> master\_host="192.168.4.52",

-> master\_user="repluser",

-> master\_password="123456",

-> master\_log\_file="52log.000002",

-> master\_log\_pos=654;

Query OK, 0 rows affected, 2 warnings (0.29 sec

mysql> start slave;

Query OK, 0 rows affected (0.04 sec)

mysql> show slave status \G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Slave\_IO\_State: Waiting for master to send event

Master\_Host: 192.168.4.52

Master\_User: repluser

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: 52log.000002

Read\_Master\_Log\_Pos: 654

Relay\_Log\_File: sql51-relay-bin.000002

Relay\_Log\_Pos: 316

Relay\_Master\_Log\_File: 52log.000002

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

**3配置/etc/mha\_manager/app1.cnf 文件添加重新添加server1的配置**

[server1]

candidate\_master=1

hostname=192.168.4.51

**在56上重新验证数据节点的主从同步配置**

[root@sql56 bin]# masterha\_check\_repl --conf=/etc/mha\_manager/app1.cnf

MySQL Replication Health is OK.

# NSD DBA2 DAY04

# 一:视图概述

## 什么是视图(view)

* 虚拟表(假表)
* 内容与真实的表相似,有字段有记录
* 视图并不在数据库中以存储的数据形式存在
* 行和列的数据来自定义视图时查询所引用的基表,并且在具体引用视图时动态生成
* 更新视图的数据,就是更新基表的数据
* 更新基表数据,视图的数据也会跟着改变

## 视图的优点:

简单:

用户不需关心视图中的数据如何查询获得

视图中的数据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:

一旦视图结构确定,可以屏蔽表结构对用户的影响

# 二:视图使用

## 视图使用限制:

* 不能在视图上创建索引
* 在试图的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,:

1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

2常量视图,join,from一个不能更新的视图

3 Where子句的子查询引用了from子句中的表

4 使用了临时表

## 创建视图:

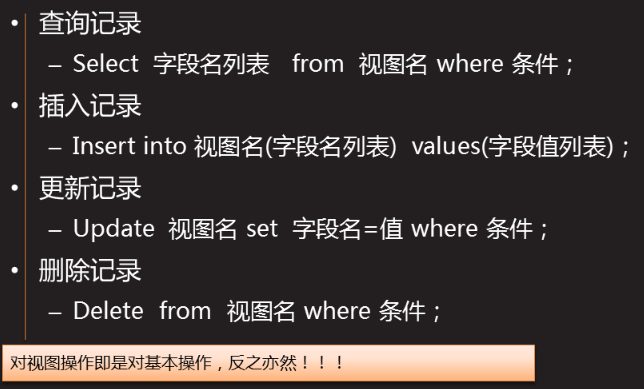
语法格式:

create view 视图名 as SQL查询;

create view 视图名称(字段名列表) as SQL查询; #可重新命名字段名创建

在视图表中不重新定义命名字段名的话,默认使用基表的字段名,若定义字段名的话,视图表中的字段必须和基表的字段个数相同

## 使用视图:



**练习:**

**创建视图v1** 结构及数据user表的字段、记录一样

mysql> create view v1 as select \* from user;

Query OK, 0 rows affected (0.00 sec)

**创建视图v2** 只有user表shell是/bin/bash用户信息

mysql> create view v2 as select shell from user;

Query OK, 0 rows affected (0.01 sec)

**分别对视图表和基表执行insert update delete 操作**

mysql> insert into v1(username,uid) values("jarry",9); //插入记录

Query OK, 1 row affected (0.00 sec)

mysql> update v1 set uid=9 where username="adm"; //更新记录

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> delete from v1 where uid=9; //删除记录

Query OK, 2 rows affected (0.01 sec)

## 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; #查看视图类型的

**查看创建视图具体命令**

Show create view 视图名;

mysql> show create view v6\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

View: v6

Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `v6` AS select `user`.`username` AS `name`,`user2`.`username` AS `bname` from (`user` join `user2`) where (`user`.`uid` = `user2`.`uid`)

character\_set\_client: utf8

collation\_connection: utf8\_general\_ci

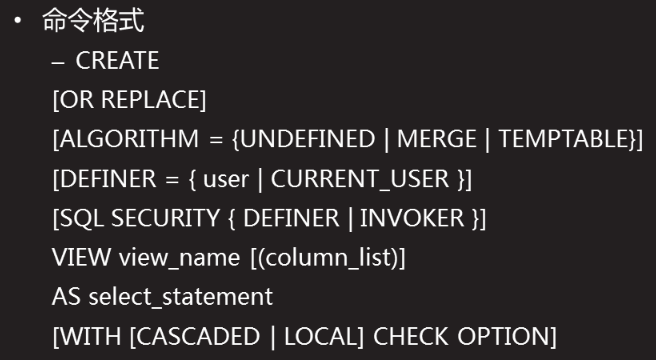
1 row in set (0.00 sec)

ERROR:

No query specified

# 三:视图进阶

创建视图的完全格式



以下实验是建立在

mysql> show tables;

+-------------------+

| Tables\_in\_db9 |

+---------------+

| user |

+------------------+

4 rows in set (0.00 sec)

User 是导入的/etc/passwd 文件的内容,详情请参阅数据导入导出

## 创建视图时覆盖原视图创建or replace

例如create or replace view v1 as select \* from user

mysql> create view v1 as select username,uid from user;

Query OK, 0 rows affected (0.03 sec)

mysql> create view v1 as select username,uid from user;

ERROR 1050 (42S01): Table 'v1' already exists #已有视图V1无法创建

mysql> create or replace view v1 as select username,uid from user;

Query OK, 0 rows affected (0.02 sec) #创建成功

## 创建视图时指定算法:algorithm

指定算法方式后,客户端访问视图时按指定算法处理

定义Algorithm=Undefined | Merage | temptables

Undefined :未定义(默认)

Merage:替换方式--as后面的查询与查询视图t1表同时执行

Temptables:具体化方式--对视图操作时,先执行创建视图时as后面的查询,结果放入内存中生成临时表,再操作视图的语句在临时表中执行.

mysql> create algorithm=temptable view v2 as select username,uid from user;

Query OK, 0 rows affected (0.04 sec)

mysql> show create view v2\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

View: v2

Create View: CREATE ALGORITHM=TEMPTABLE DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `v2` AS select `user`.`username` AS `username`,`user`.`uid` AS `uid` from `user`

character\_set\_client: utf8

collation\_connection: utf8\_general\_ci

1 row in set (0.00 sec)

ERROR:

No query specified

## 创建视图时指定检查方式(local | cascaded)

指定检查方式后,客户端访问视图做修改数据等操作时,按照指定检查进行操作限制

定义:with local | cascaded check option

Local 和 cascaded 关键字决定检查的范围

Local---仅检查当前视图的限制,只检查视图创建时定义的条件

Cascaded (默认)---同时要满足基表的限制条件,检查视图创建条件外还检查基表创建条件

例如:不指定为默认cascaded

Create view v1 as select \* from a where uid<10 with check option;

解释:创建了视图v1 ,创建时定义的条件为:uid<10,定义了local检查方式后,用户修改数据时,只能修改uid<10,同时还要检查基表a 创建时 where的条件范围

例如:指定为local

Create view v1 as select \* from a where uid<10 with local check option;

解释:创建了视图v1 ,创建时定义的条件为:uid<10,定义了local检查方式后,用户修改数据时,只能修改where 条件的uid<10范围内

1创建user2表,数据来自于user表中,条件是:user表中uid>=100 and uid<=1000;

mysql> create table user2 sekect username,uid,shell,homedir from user where uid>=100 and uid<=1000;

2创建视图v3 数据来自与user2表中,条件是:uid >=10 and uid<=500

mysql> create view v3 as select \* from user2 where uid >=10 and uid<=500 with local check option;

Query OK, 0 rows affected (0.03 sec)

mysql> select \* from v3;

+-----------------+------+---------------+----------------+

| username | uid | shell | homedir |

+-----------------+------+---------------+----------------+

| systemd-network | 192 | /sbin/nologin | / |

| abrt | 173 | /sbin/nologin | /etc/abrt |

| rtkit | 172 | /sbin/nologin | /proc |

| usbmuxd | 113 | /sbin/nologin | / |

| qemu | 107 | /sbin/nologin | / |

| pulse | 171 | /sbin/nologin | /var/run/pulse |

+-----------------+------+---------------+----------------+

mysql> show create view v3\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

View: v3

Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `v3` AS select `user2`.`username` AS `username`,`user2`.`uid` AS `uid`,`user2`.`shell` AS `shell`,`user2`.`homedir` AS `homedir` from `user2` where ((`user2`.`uid` >= 10) and (`user2`.`uid` <= 500)) WITH LOCAL CHECK OPTION

修改数据操作:

mysql> update v3 set uid =9 where username="abrt"; #修改uid=9的名字

ERROR 1369 (HY000): CHECK OPTION failed 'db9.v3' #报错

mysql> update v3 set uid =99 where username="abrt"; #修改uid=99(范围内)

Query OK, 1 row affected (0.03 sec) #成功

Rows matched: 1 Changed: 1 Warnings: 0

3创建视图v4 数据来自于v3 条件是:uid>=100 指定检查为:

mysql> create view v4 as select \* from v3 where uid>=100 with cascaded check option;

Query OK, 0 rows affected (0.02 sec)

mysql> show create view v4\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

View: v4

Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `v4` AS select `v3`.`username` AS `username`,`v3`.`uid` AS `uid`,`v3`.`shell` AS `shell`,`v3`.`homedir` AS `homedir` from `v3` where (`v3`.`uid` >= 100) WITH CASCADED CHECK OPTION

修改数据操作:

mysql> update v4 set uid=99 where username="qemu";

ERROR 1369 (HY000): CHECK OPTION failed 'db9.v4'

#视图创建条件`uid` >= 100 所以修改uid为99就不符合

mysql> update v4 set uid=1001 where username="qemu";

ERROR 1369 (HY000): CHECK OPTION failed 'db9.v4'

#因为视图v4创建是v3是基表,而v3的创建的条件为:uid >=10 and uid<=500,所以uid=1001就不复合

mysql> update v4 set uid=101 where username="qemu";

Query OK, 1 row affected (0.05 sec)

Rows matched: 1 Changed: 1 Warnings: 0

## 设置视图表字段别名

视图中的字段名不可重复,多表查询中,如果有字段名一样,所以要定义别名

mysql> select user.username,user2.username from user,user2 where user.uid = user2.uid;

+------------------------------+--------------------------+

| username | username | #查询出来的两个字段名相同

+-----------------------------+---------------------------+

| systemd-network | systemd-network |

| polkitd | polkitd |

这时候利用这条语句创建视图是不行的,查询出来的两个username字段是无法创建的.

mysql> select user.username as ausername,user2.username as busername from user,user2 where user.uid = user2.uid;

+-----------------------------+----------------------------+

| ausername | busername | #查询出来的字段名不同

+-----------------------------+-------------------------------+

| systemd-network | systemd-network |

| polkitd | polkitd |

| libstoragemgmt | libstoragemgmt |

| colord | colord |

创建视图测试:

未取别名创建

取别名后创建

取别名方法

Select 字段名 别名 from 表名 #没有用as

Select 字段名 as 别名 from 表名 #用了as方法

可以给表定义别名,也可以给表字段取别名

给表字段别名:

mysql> create view v6(name,bname) as select user.username,user2.username from user,user2 where user.uid = user2.uid;

Query OK, 0 rows affected (0.04 sec)

mysql> select \* from v6;

+---------------------+---------------------+

| name | bname |

+---------------------+---------------------+

| nobody | abrt |

| systemd-network | systemd-network |

| polkitd | polkitd |

| libstoragemgmt | libstoragemgmt |

| colord | colord |

| saslauth | saslauth |

给表取别名

mysql> create view v5(aname,wname) as

-> select b.name, c.name from

-> studentinfo b left join user2 c

-> on

-> b.name = c.name;

mysql> show tables;

将studentinfo 表命名为 b user2 命名为 c 创建视图v5,数据来源为studentinfo表和user2表中的name,studentinfo表中的name字段重命名为aname,user2中的name字段重命名为bname

# 四:存储过程

存储过程,相当于是mysql语句组成的脚本

* 指的是数据库中保存的一系列sql命令的集合
* 可以在存储过程中使用变量/条件判断/流程控制等

存储过程优点:

* 提高性能
* 减轻网络负担
* 可以防止对表的直接访问
* 避免重复编写sql操作

## 创建存储过程

mysql>delimiter // create procedure 名称() begin ...功能代码 end //

mysql>delimiter ;

指定分隔符为”//” 创建过程 名称名() 开始 ...功能代码 结束 // 分隔符改回 ;

#dilimiter 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

mysql> delimiter //

mysql> create procedure a1()

-> begin

-> select \* from db9.user

-> where id <= 10;

-> end

-> //

Query OK, 0 rows affected (0.01 sec)

## 调用存储过程

mysql>call 存储过程名称();

mysql> call a1;

## 删除存储过程

mysql>drop procedure 库名.存储过程名;

mysql>drop procedure db9.a1;

## 查看存储过程

### 查看已有存储过程

mysql> show procedure status;

mysql> select db,name,type from mysql.proc where name="p3" and type="procedure";

+-----+------+-----------+

| db | name | type |

+-----+------+-----------+

| db9 | p3 | PROCEDURE |

+-----+------+-----------+

1 row in set (0.00 sec)

mysql> Select db,name,type from mysql.proc where type="procedure"\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

db: db9

name: a1

type: PROCEDURE

### 查看存储过程具体代码

mysql> select body from mysql.proc where name="a1"\G ;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

body: begin

select \* from db9.user

where id <= 10;

end

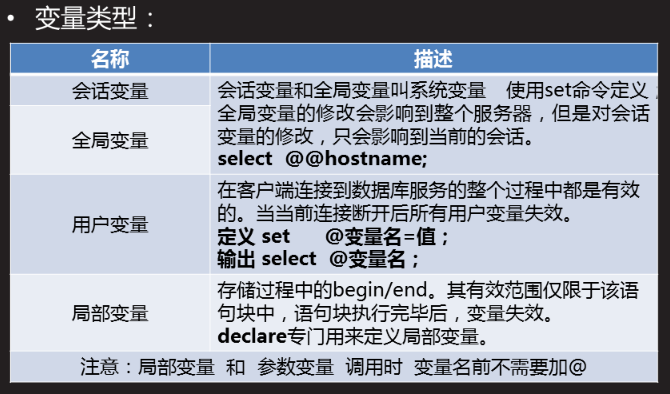
1 row in set (0.00 sec)

ERROR:

No query specified

# 五:存储过程进阶

## 变量(类型 定义 调用)



### 全局变量

#### 查看全局变量:

mysql> show global variables; #查看所有全局变量

mysql> show global variables like "%ver"; #模糊查找

+----------------------+-------------------+

| Variable\_name | Value |

+----------------------+-------------------+

| character\_set\_server | latin1 |

| collation\_server | latin1\_swedish\_ci |

+----------------------+-------------------+

2 rows in set (0.00 sec)

mysql> show global variables like"%name%";

+-----------------------------------+------------------------------------+

| Variable\_name | Value |

+-----------------------------------+------------------------------------+

| hostname | client-50 |

| innodb\_buffer\_pool\_filename | ib\_buffer\_pool |

| lc\_time\_names | en\_US |

| log\_bin\_basename | |

| lower\_case\_table\_names | 0 |

+-----------------------------------+------------------------------------+

8 rows in set (0.00 sec)

#### 输出全局变量

mysql> select @@hostname;

+------------+

| @@hostname |

+------------+

| client-50 |

+------------+

1 row in set (0.00 sec)

mysql> select @@hostname , @@version\_comment;

+------------+------------------------------+

| @@hostname | @@version\_comment |

+------------+------------------------------+

| client-50 | MySQL Community Server (GPL) |

+------------+------------------------------+

1 row in set (0.00 sec)

#### 定义全局变量

mysql> set global 变量名=值；

### 会话变量

#### 查看会话变量

mysql> show session variables like "sort\_buffer\_size";

+------------------+--------+

| Variable\_name | Value |

+------------------+--------+

| sort\_buffer\_size | 262144 |

+------------------+--------+

1 row in set (0.00 sec)

#### 设置会话变量

mysql> set session sort\_buffer\_size=40000;

Query OK, 0 rows affected (0.00 sec)

mysql> show session variables like "sort\_buffer\_size";

+------------------+-------+

| Variable\_name | Value |

+------------------+-------+

| sort\_buffer\_size | 40000 |

+------------------+-------+

1 row in set (0.01 sec)

### set定义自定义变量

mysql> set @age=9;

mysql> set @name="bob";

mysql> select @age;

+------+

| @age |

+------+

| 9 |

+------+

mysql> select @name;

+-------+

| @name |

+-------+

| bob |

+-------+

mysql> select @name,@age;

+-------+------+

| @name | @age |

+-------+------+

| bob | 9 |

+-------+------+

1 row in set (0.00 sec)

### declare定义局部变量

只能用在begin 与 end 之间,调用时没有@符号

mysql> delimiter //

mysql> create procedure p2()

-> begin

-> declare x int;

-> declare y int;

-> declare name char(10);

-> set x=9;

-> set y=11;

-> set name="bob";

-> select x,y,name;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> call p2;

+------+------+------+

| x | y | name |

+------+------+------+

| 9 | 11 | bob |

+------+------+------+

1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

create procedure p4()

-> begin

-> declare x int;

-> select count(username) into x from db9.user; #查询结果复制给变量

-> select x;

-> end

-> //

mysql> delimiter ;

### 算数运算



mysql> set @z=1+2;select @z;

Query OK, 0 rows affected (0.00 sec)

+------+

| @z |

+------+

| 3 |

+------+

1 row in set (0.00 sec)

mysql> set @x=1; set @y=2; set @z=@x\*@y;select @z;

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

+------+

| @z |

+------+

| 2 |

+------+

1 row in set (0.00 sec)

mysql> set @x=1; set @y=2; set @z=@x+@y;select @z;

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

+------+

| @z |

+------+

| 3 |

+------+

1 row in set (0.00 sec)

mysql> set @var5=10 DIV 3; select @var5; #整除运算,并输出@var5

Query OK, 0 rows affected (0.00 sec)

+-------+

| @var5 |

+-------+

| 3 |

+-------+

1 row in set (0.00 sec)

## 存储过程参数类型

in 输入类型

out 输出类型 该值可在存储过程内部被改变,并可返回

inout 输入/输出类型 调用是指定,并且可被改变和返回

调用参数时,名称前不需要加@

多个参数之间用逗号分隔

### in 输入类型:

mysql> create procedure sa1( in username char(10)) #定义输入参数

-> begin

-> select username; #输出定义的参数

-> select \* from user where name=username; #使用定义的参数

-> end

-> //

Query OK, 0 rows affected (0.01 sec)

mysql>call sa1(“bob”) #调用存储过程时,输入参数

##调用后相当于执行:select \* from user where name=”bob”;

如果定义好输入参数,调用存储过程时未输入参数”bob”,会报错

### out 输出类型

该值可在存储过程内部被改变,并可返回

mysql>delimiter //

mysql> create procedure sa2(out x int) #定义输入参数

-> begin

-> select count(username) into x from db9.user wher shell!=”/bin/bash”;

-> select x;

-> end

-> //

Query OK, 0 rows affected (0.01 sec)

mysql> call sa2();

ERROR 1318 (42000): Incorrect number of arguments for PROCEDURE db9.sa2; expected 1, got 0

mysql> call sa2(@z); #调用sa2同时将x 的值赋值给变量@z

+------+

| x |

+------+

| 39 |

+------+

1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> select @z; #输出

+------+

| @z |

+------+

| 39 |

+------+

1 row in set (0.00 sec)

### inout 输入/输出类型

调用时指定,并且可被改变和返回

mysql> delimiter //

mysql> create procedure sa3(inout x int)

-> begin

-> select \* from db9.user where id<=x;

-> select count(id) into x from db9.user;

-> select x;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> call sa3();

ERROR 1318 (42000): Incorrect number of arguments for PROCEDURE db9.sa3; expected 1, got 0 #需要赋值1个,已复制0个

mysql> call sa3(9);

ERROR 1414 (42000): OUT or INOUT argument 1 for routine db9.sa3 is not a variable or NEW pseudo-variable in BEFORE trigger

**第一种:**

mysql> call sa3(@i); #赋值一个空变量

Empty set (0.00 sec)

+------+

| x |

+------+

| 41 |

+------+

1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

**第二种:**

mysql> set @k=4; #先赋值给k

Query OK, 0 rows affected (0.00 sec)

mysql> call sa3(@k); #调用sa3同时把k的值4输入给x,处理结果赋值给k

+----+----------+----------+------+------+---------+----------+---------------+

| id | username | password | 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 |

+----+----------+----------+------+------+---------+----------+---------------+

4 rows in set (0.00 sec)

+------+

| x |

+------+

| 41 |

+------+

1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> select @k; #保存输出类型参数x 的处理结果

+------+

| @k |

+------+

| 41 |

+------+

1 row in set (0.00 sec)

## 流程控制

流程控制 条件判断 流程控制语句

### 流程控制

选择结构 循环结构

#### 选择结构

单分支选择结构

if 条件测试 then

代码...

...

end if;

双分支选择结构

if 条件测试 then

代码1...

else

代码2...

end if;

mysql> delimiter //

mysql> create procedure db9.sqy7(in linenum int)

-> begin

-> if linenum <= 10 then

-> select \* from db9.user where id <= 10 ;

-> else

-> select \* from db9.user where id > 10 ;

-> end if;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> call sqy7(3); #给linenum 赋值3,进行判断,执行第一句查询

mysql> call sqy7(11); #给linenum 赋值11,进行判断,执行第二句查询

#### 循环结构

**While条件循环:反复测试条件,只要成立就执行命令代码**

mysql> delimiter //

mysql> create procedure say9(in y int)

-> begin

-> declare x int;

-> set x = 1 ;

-> while x <= y do

-> select x;

-> set x = x + 1 ;

-> end while;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> call say9(3);

**loop死循环:无条件,反复执行某一段代码**

loop

循环体

end loop;

mysql>delimiter //

mysql>creater procedure say12()

-> begin

-> loop

-> select \* from db9.user where id=1;

-> end loop;

-> end

-> //

mysql> delimiter ;

mysql>call say12()

**repeat条件式循环:当条件成立时结束循环**

repeat

循环体

Until 条件判断

end repeat;

mysql> delimiter //

mysql> create procedure say11(in y int)

-> begin

-> declare x int;

-> set x = 1 ;

-> repeat

-> select x;

-> set x = x + 1 ;

-> until x > y

-> end repeat;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> call say11(3);

### 条件判断

### 流程控制语句

控制循环结构的执行 while repeat loop

#### Leave 结束循环

mysql> delimiter //

mysql> create procedure say13()

-> begin

-> declare x int;

-> set x = 1;

-> loadb1 :while x <= 10 do #给循环取个名:loadb1

-> if x = 5 then #判断

-> leave loadb1; #如果成立.则停止名为loadb1的循环;

-> end if;

-> select x ;

-> set x = x + 1;

-> end while;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> call say13();

#### Iterate 结束当前循环

结束当前循环,开始下一轮循环

mysql> delimiter //

mysql> create procedure say14()

-> begin

-> declare x int ;

-> set x=1;

-> loadb2:while x <= 10 do

-> if x = 5 then

-> set x = x+1; #如果不在此处自加1,将会永远卡在x=5,不断判断并结束

-> iterate loadb2;

-> end if;

-> select x ;

-> set x = x+1;

-> end while;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> call say14();　　　#结果是1-10中的5不输出

# 案例

案例1：视图的基本使用

案例2：视图进阶操作

案例3：创建存储过程

案例4：存储过程参数的使用

案例5：使用循环结构

1 案例1：视图的基本使用

1.1 问题

把/etc/passwd文件的内容存储到db9库下的user表里

添加新字段id 存储记录的行号(在所有字段的前边)

创建视图v1 结构及数据user表的字段、记录一样。

创建视图v2 只有user表shell是/bin/bash用户信息 。

分别对视图表和基表执行insert update delete 操作。

删除视图v1 和 v2

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：视图的基本使用

什么是视图：是一种虚拟存在的表

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本表，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

1）把/etc/passwd文件的内容存储到db9库下的user表里

[root@mysql51 ~]# mysql -u root -p123456

mysql> create database db9;

Query OK, 1 row affected (10.00 sec)

mysql> create table db9.user(username char(20),password char(1),uid \

int(2),gid int(2),comment char(100),homedir char(100),shell char(50));

//创建存储数据的表结构

Query OK, 0 rows affected (0.02 sec)

[root@mysql51 ~]# cp /etc/passwd /var/lib/mysql-files/

[root@mysql51 ~]# ls /var/lib/mysql-files/

passwd

mysql> load data infile "/var/lib/mysql-files/passwd" into table db9.user fields terminated by ":" lines terminated by "\n"; //导入文件内容到db9.user

Query OK, 41 rows affected (0.02 sec)

Records: 41 Deleted: 0 Skipped: 0 Warnings: 0

2）添加新字段id 存储记录的行号(在所有字段的前边)

mysql> alter table db9.user add id int(2) primary key auto\_increment first;

Query OK, 0 rows affected (0.04 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> use db9;

mysql> desc user;

+----------+-----------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+----------+-----------+------+-----+---------+----------------+

| id | int(2) | NO | PRI | NULL | auto\_increment |

| username | char(20) | YES | | NULL | |

| password | char(1) | YES | | NULL | |

| uid | int(2) | YES | | NULL | |

| gid | int(2) | YES | | NULL | |

| comment | char(100) | YES | | NULL | |

| homedir | char(100) | YES | | NULL | |

| shell | char(50) | YES | | NULL | |

+----------+-----------+------+-----+---------+----------------+

8 rows in set (0.00 sec)

3）创建视图v1 结构及数据user表的字段、记录一样

mysql> create view v1 as select \* from user;

Query OK, 0 rows affected (0.00 sec)

4）创建视图v2 只有user表shell是/bin/bash用户信息

mysql> create view v2 as select shell from user;

Query OK, 0 rows affected (0.01 sec)

5）分别对视图表和基表执行insert update delete 操作

mysql> insert into v1(username,uid) values("jarry",9); //插入记录

Query OK, 1 row affected (0.00 sec)

mysql> update v1 set uid=9 where username="adm"; //更新记录

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> delete from v1 where uid=9; //删除记录

Query OK, 2 rows affected (0.01 sec)

6）删除视图v1 和 v2

mysql> drop view v1;

Query OK, 0 rows affected (0.00 sec)

mysql> drop view v2;

Query OK, 0 rows affected (0.00 sec)

注意：对视图操作即是对基本操作，反之亦然！！！

2 案例2：视图进阶操作

2.1 问题

练习OR REPLACE的选项使用

练习WITH LOCAL CHECK OPTION 选项的使用

练习WITH CASCADED CHECK OPTION 选项的使用

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：视图进阶操作

1）创建视图完全格式

mysql> create table user2 select username,uid,gid from user limit 3;

//快速建表（user2表）

Query OK, 3 rows affected (0.01 sec)

Records: 3 Duplicates: 0 Warnings: 0

mysql> create table info select username,uid,homedir,shell from user limit 5;

//快速建表（info表）

Query OK, 5 rows affected (0.02 sec)

Records: 5 Duplicates: 0 Warnings: 0

查询user2.username=info.username的字段

mysql> select \* from user2 left join info on user2.username=info.username;

+----------+------+------+----------+------+---------+---------------+

| username | uid | gid | username | uid | homedir | shell |

+----------+------+------+----------+------+---------+---------------+

| root | 0 | 0 | root | 0 | /root | /bin/bash |

| bin | 1 | 1 | bin | 1 | /bin | /sbin/nologin |

| daemon | 2 | 2 | daemon | 2 | /sbin | /sbin/nologin |

+----------+------+------+----------+------+---------+---------------+

3 rows in set (0.00 sec)

2）关联查询建的视图 默认不允许修改视图字段的值

mysql> create view v4 as select \* from user2 left join info on user2.username=info.username; //创建失败

ERROR 1060 (42S21): Duplicate column name 'username'

mysql> create view v4 as select a.username as ausername,b.username as busername, a.uid as auid,b.uid as buid from user2 a left join info b on a.username=b.username;

//创建成功

Query OK, 0 rows affected (0.00 sec)

mysql> select \* from v4;

+-----------+-----------+------+------+

| ausername | busername | auid | buid |

+-----------+-----------+------+------+

| root | root | 0 | 0 |

| bin | bin | 1 | 1 |

| daemon | daemon | 2 | 2 |

+-----------+-----------+------+------+

3 rows in set (0.00 sec)

mysql> desc v4;

+-----------+----------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-----------+----------+------+-----+---------+-------+

| ausername | char(20) | YES | | NULL | |

| busername | char(20) | YES | | NULL | |

| auid | int(2) | YES | | NULL | |

| buid | int(2) | YES | | NULL | |

+-----------+----------+------+-----+---------+-------+

4 rows in set (0.00 sec)

3）OR REPLACE的选项使用

创建时，若视图已存在，会替换已有的视图

语法格式：create or replace view视图名as select 查询； //达到修改已有视图的目的

mysql> create or replace view v4 as select a.username as ausername,b.username as busername, a.uid as auid,b.uid as buid from user2 a left join info b on a.username=b.username;

Query OK, 0 rows affected (0.00 sec)

4）WITH LOCAL CHECK OPTION

LOCAL和CASCADED关键字决定检查的范围

LOCAL 仅检查当前视图的限制

CASCADED 同时要满足基表的限制（默认值）

mysql> create table user1 select username,uid,shell from user where uid>=5 and uid <=40;

Query OK, 11 rows affected (0.01 sec)

Records: 11 Duplicates: 0 Warnings: 0

mysql> create view v1 as select username,uid from user1 where uid<=20;

Query OK, 0 rows affected (0.01 sec)

mysql> update v1 set uid=21 where username="sync";

//操作超过视图表的条件限制（uid<=20）之后，在视图表里面查看不到，在基表里可以查看到

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> update user1 set uid=41 where username="ftp";

//基表在超过条件限制（uid>=5 and uid <=40),在基表里依然可以查看到

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> create table a select \* from user where uid < 10;

//快速创建一个新表a

Query OK, 7 rows affected (0.01 sec)

Records: 7 Duplicates: 0 Warnings: 0

mysql> create view v3 as select \* from a where uid < 10 with check option;

//不写默认为CASCADED检查自己和a要满足的要求即可

Query OK, 0 rows affected (0.00 sec)

mysql> update v3 set uid=9 where username="adm"; //更改成功

Query OK, 0 rows affected (0.01 sec)

Rows matched: 0 Changed: 0 Warnings: 0

mysql> create view v2 as select \* from v1 where uid >= 5 with local check option;

//满足自身v2的要求

Query OK, 0 rows affected (0.00 sec)

mysql> update v2 set uid=9 where username="sync";

Query OK, 0 rows affected (0.00 sec)

Rows matched: 0 Changed: 0 Warnings: 0

5）WITH CASCADED CHECK OPTION

mysql> create view v5 as select \* from v1 where uid >= 5 with cascaded check option;

Query OK, 0 rows affected (0.00 sec)

3 案例3：创建存储过程

3.1 问题

存储过程名称为p1

功能显示user表中 shell是/bin/bash的用户个数

调用存储过程p1

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：存储过程基本使用

1）创建存储过程

mysql> delimiter // //定义定界符

mysql> create procedure say() //say()随便写括号一定要有

-> begin

-> select \* from user where id<=10;

-> end

-> //

Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ; //把命令的定界符改回来，分号前有空格

mysql> call say(); //调用存储过程名,在括号里面不写参数时，可以不加括号

2）查看存储过程

方法一：

mysql> show procedure status\G;

方法二：

mysql> select db,name,type from mysql.proc where name= "say";

3）删除存储过程

mysql> drop procedure say;

Query OK, 0 rows affected (0.00 sec)

4）创建存储过程名称为p1

 功能显示user表中 shell是/bin/bash的用户

 调用存储过程p1

mysql> delimiter //

mysql> create procedure p1()

-> begin

-> select count(username) from user where shell="/bin/bash";

-> end

-> //

mysql> delimiter ;

mysql> call p1();

+-----------+

| shell |

+-----------+

| /bin/bash |

| /bin/bash |

+-----------+

2 rows in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

4 案例4：存储过程参数的使用

4.1 问题

创建名为p2的存储过程

可以接收用户输入shell的名字

统计user表中用户输入shell名字的个数

4.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：存储过程参数的使用

1）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

）

in 输入参数 传递值给存储过程，必须在调用存储过程时指定，在存储过程中修改该参数的值不能；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

mysql> delimiter //

mysql> create procedure say2(in username char(10))

-> begin

-> select username;

-> select \* from user where username=username;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> call say2("tom");

2）创建名为p2的存储过程，可以接收用户输入shell的名字，统计user表中用户输入shell名字的个数

mysql> delimiter //

mysql> create procedure p2(out number int)

-> begin

-> select count(username) into @number from user where shell!="/bin/bash";

-> select @number;

-> end

-> //

Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;

mysql> call p2(@number);

+---------+

| @number |

+---------+

| 38 |

+---------+

1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

5 案例5：使用循环结构

5.1 问题

定义名称为p3的存储过程

用户可以自定义显示user表记录的行数

若调用时用户没有输入行数，默认显示第1条记录

5.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：算数运算

1）算数运算符号，如图-1所示：

图-1

mysql> set @z=1+2;select @z;

Query OK, 0 rows affected (0.00 sec)

+------+

| @z |

+------+

| 3 |

+------+

1 row in set (0.00 sec)

mysql> set @x=1; set @y=2;set @z=@x\*@y; select @z;

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

+------+

| @z |

+------+

| 2 |

+------+

1 row in set (0.00 sec)

mysql> set @x=1; set @y=2;set @z=@x-@y; select @z;

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

+------+

| @z |

+------+

| -1 |

+------+

1 row in set (0.00 sec)

mysql> set @x=1; set @y=2;set @z=@x/@y; select @z;

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

+-------------+

| @z |

+-------------+

| 0.500000000 |

+-------------+

1 row in set (0.00 sec)

declare调用变量不需要@其他都需要

调用变量时，有@符号的变量 如@x：调用的是用户自定义变量

没有@符号的变量 如x：调用的是存储过程的参数变量

mysql> delimiter //

mysql> create procedure say5(in bash char(20), in nologin char(25), out x int , out y int)

-> begin

-> declare z int ;

-> set z=0;

-> select count(username) into @x from user where shell=bash;

-> select count(username) into @y from user where shell=nologin;

-> set z=@x+@y;

-> select z;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> call say5("/bin/bash","/sbin/nologin",@x,@y);

+------+

| z |

+------+

| 36 |

+------+

1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

2）条件判断，数值的比较如图-2所示：

图-2

逻辑比较、范围、空、非空、模糊、正则，如图-3所示：

图-3

顺序结构（if判断）当“条件成立”时执行命令序列,否则，不执行任何操作

mysql> delimiter //

mysql> create procedure say6(in x int(1) )

-> begin

-> if x <= 10 then

-> select \* from user where id <=x;

-> end if;

-> end

-> //

Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;

mysql> call say6(1); //条件判断成立，等于1是否成立

+----+----------+----------+------+------+---------+---------+-----------+

| id | username | password | uid | gid | comment | homedir | shell |

+----+----------+----------+------+------+---------+---------+-----------+

| 1 | root | x | 0 | 0 | root | /root | /bin/bash |

+----+----------+----------+------+------+---------+---------+-----------+

1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> call say6(2);

+----+----------+----------+------+------+---------+---------+---------------+

| id | username | password | uid | gid | comment | homedir | shell |

+----+----------+----------+------+------+---------+---------+---------------+

| 1 | root | x | 0 | 0 | root | /root | /bin/bash |

| 2 | bin | x | 1 | 1 | bin | /bin | /sbin/nologin |

+----+----------+----------+------+------+---------+---------+---------------+

2 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

3）定义名称为p3的存储过程，用户可以自定义显示user表记录的行数，若调用时用户没有输入行数，默认显示第1条记录

mysql> delimiter //

mysql> create procedure p3(in linenum char(10) )

-> begin

-> if linenum is null then

-> set @linenum=1;

-> select \* from user where id=@linenum;

-> else

-> select linenum;

-> select \* from user where id=linenum;

-> end if;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> call p3(null); //不输入查看的行数

+----+----------+----------+------+------+---------+---------+-----------+

| id | username | password | uid | gid | comment | homedir | shell |

+----+----------+----------+------+------+---------+---------+-----------+

| 1 | root | x | 0 | 0 | root | /root | /bin/bash |

+----+----------+----------+------+------+---------+---------+-----------+

1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> call p3(3); //输入查看的行数

+---------+

| linenum |

+---------+

| 3 |

+---------+

1 row in set (0.00 sec)

+----+----------+----------+------+------+---------+---------+---------------+

| id | username | password | uid | gid | comment | homedir | shell |

+----+----------+----------+------+------+---------+---------+---------------+

| 3 | daemon | x | 2 | 2 | daemon | /sbin | /sbin/nologin |

+----+----------+----------+------+------+---------+---------+---------------+

1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

案例5(课堂)

mysql> create procedure p3(in x int)

-> begin

-> if x is null then

-> select \* from db9.user where id=1;

-> else

-> select \* from db9.user where id<=x;

-> end if;

-> end

-> //

Query OK, 0 rows affected (0.00 sec)

mysql> set @w=10;

-> ^C

mysql> delimiter ;

mysql> set @w=10;

Query OK, 0 rows affected (0.00 sec)

# NSD DBA2 DAY05

# 一数据分片

## 1.1数据分片介绍

分库分表

将存放在一个数据库(主机)中的数据,按照特定方式进行拆分,分散存放到多个数据库主机中,以达到分散单台设备负载的效果

垂直分隔(纵向切分)

* 将单个表,拆分成多个表,分散到不同的数据库
* 将单个数据库的多个表进行分类,按业务类别分散到不同的数据库上

水平分隔(横向切分)

* 按照表中某个字段的某种规则,把表中的许多记录按行切分,分散到多个数据库中

## 1.2配置数据分片服务武器

### 1.2.1Mycat软件介绍

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,使用需要JDK环境

* 支持JDBC形式连接
* 支持MYSQL OREACLE 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(大量查询需求不合适)

分片规则

1枚举法

2固定分片

3范围约定

4求模法

5日期列分区法

6通配取模

7 ASCII码求模通配

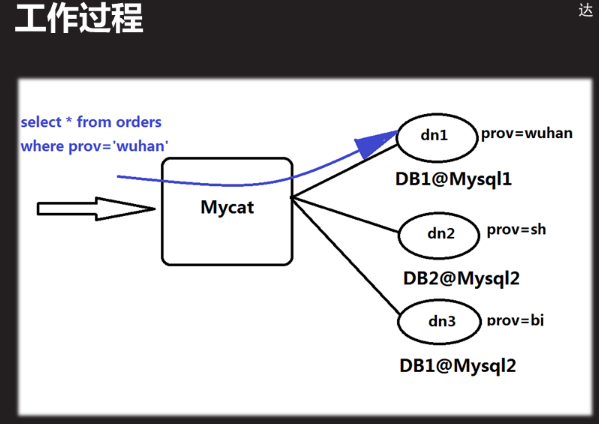
8编程指定

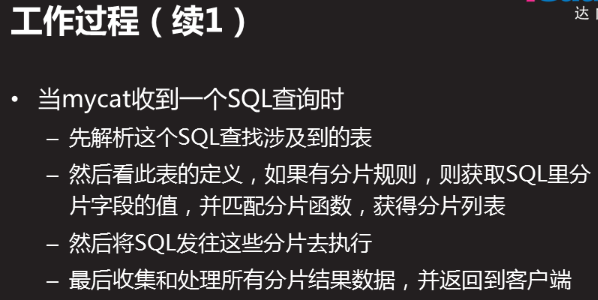
9字符串拆分hash解析

10一致性hash

### 1.2.2服务工作过程

查询,插入过程





### 1.2.3实验拓扑结构

sql53 数据库服务器1 192.168.53/24

sql54 数据库服务器2 192.168.54/24

sql55 数据库服务器3 192.168.55/24

sql56 mycat服务器 192.168.56/24

### 1.2.4配置数据分片服务

#### 1:装包

[root@sql56 ~]# rpm -qa |grep jdk

java-1.8.0-openjdk-1.8.0.131-11.b12.el7.x86\_64

java-1.8.0-openjdk-headless-1.8.0.131-11.b12.el7.x86\_64

copy-jdk-configs-2.2-3.el7.noarch

[root@sql56 ~]# java -version

openjdk version "1.8.0\_131"

[root@sql56 ~]# tar -xf Mycat-server-1.6-RELEASE-20161028204710-linux.tar.gz

[root@sql56 ~]# mv mycat /usr/local/

[root@sql56 ~]# ls /usr/local/mycat/

bin catlet conf lib logs version.txt

目录结构说明

-bin: mycat管理命令

catlet: mycat扩展功能

conf: mycat配置文件

lib: mycat使用的jar(库文件:开发者写好的功能脚本)

log: mycat启动日志和运行日志

wrapper.log: mycat服务启动日志

mycat.log: mycat记录sql脚本执行后报错内容

[root@sql56 ~]# cd /usr/local/mycat/conf/

[root@sql56 conf]# ls #配置目录重要的文件

Server.xml 设置连mycat的账号信息

Schema.xml 配置mycat的真实库表,

rule.xml 定义mycat分片规则

[root@sql56 conf]# vim rule.xml #简单查看分片规则,此处不做操作

12 <tableRule name="rule1"> #规则名称

13 <rule>

14 <columns>id</columns>

15 <algorithm>func1</algorithm> #算法

16 </rule>

17 </tableRule>

18

规则定义

110 <function name="func1" class="io.mycat.route.function.PartitionByLong">

111 <property name="partitionCount">8</property>

112 <property name="partitionLength">128</property>

113 </function>

#### 2修改配置文件

**定义连接mycat服务器使用用户名密码及数据库名和权限**

server.xml文件作用:设置连mycat的账号信息

[root@sql56 ]# vim /usr/local/mycat/conf/server.xml 设置连mycat的账号信息

[root@sql56 ~]# cd /usr/local/mycat/conf/

[root@sql56 conf]# cp server.xml /root #备份server.xml配置文件

80 <user name="root"> #连接mycat用户名

81 <property name="password">123456</property> #密码

82 <property name="schemas">TESTDB</property> #能看到库名

83

84 <!-- 表级 DML 权限设置 -->

85 <!-- #注释开始

86 <privileges check="false">

87 <schema name="TESTDB" dml="0110" >

88 <table name="tb01" dml="0000"></table>

89 <table name="tb02" dml="1111"></table>

90 </schema>

91 </privileges>

92 --> #注释结束

93 </user>

94

95 <user name="user"> #连接mycat用户名

96 <property name="password">user</property> #密码

97 <property name="schemas">TESTDB</property> #能看到库名

98 <property name="readOnly">true</property> #定义只读

99 </user>

100

**配置数据分片**

schema.xml文件作用:配置mycat的真实库表,

[root@sql56 ]# vim /usr/local/mycat/conf/schema.xml

[root@sql56 ~]# cd /usr/local/mycat/conf/

[root@sql56 conf]# cp schema.xml /root #备份schema.xml 配置文件

[root@sql56 conf]# sed -i '56,77d' schema.xml

[root@sql56 conf]# sed -i '39,42d' schema.xml

[root@sql56 conf]# vim schema.xml #定义规则

5 <schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100">

6 <!-- auto sharding by id (long) -->

7 <table name="travelrecord" dataNode="dn1,dn2,dn3" rule="auto-sharding-long" /> #

11 <table name="company" primaryKey="ID" type="global" dataNode="dn1,dn2,dn3" />

33</schema> #global全局规则,dn1,dn2,dn3数据库服务节点名

**#节点名对应的数据库服务器主机localhost53 localhost54 localhost55自己定义**

**#例如:**<dataNode></dataNode>**定义数据库主机”localhost53”用数据库db1来存储数据**

36 <dataNode name="dn1" dataHost="localhost53" database="db1" />

37 <dataNode name="dn2" dataHost="localhost54" database="db2" />

38 <dataNode name="dn3" dataHost="localhost55" database="db3" />

**#删除不相关行,如下,并修改<datahost></datahost>指定数据库地址以及用户信息**

40<dataHost name="localhost53" maxCon="1000" minCon="10" balance="0"

41 writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

42 <heartbeat>select user()</heartbeat>

43 <writeHost host="hostM1" url="192.168.4.53:3306" user="adminplj"

44 password="123qqq...A">

45 </writeHost>

46 </dataHost>

47<dataHost name="localhost54" maxCon="1000" minCon="10" balance="0"

48 writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

49 <heartbeat>select user()</heartbeat>

50 <writeHost host="hostM2" url="192.168.4.54:3306" user="adminplj"

51 password="123qqq...A">

52 </writeHost>

53 </dataHost>

54<dataHost name="localhost55" maxCon="1000" minCon="10" balance="0"

55 writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

56 <heartbeat>select user()</heartbeat>

57 <writeHost host="hostM3" url="192.168.4.55:3306" user="adminplj"

58 password="123qqq...A">

59 </writeHost>

60</dataHost>

#### 3启动服务并查看是否正常

[root@sql56 conf]# /usr/local/mycat/bin/mycat start

Starting Mycat-server...

[root@sql56 conf]# netstat -nutlp | grep :8066 #一定要查看端口

tcp6 0 0 :::8066 :::\* LISTEN 6836/java

[root@sql56 conf]# ps -C java

PID TTY TIME CMD

6836 ? 00:00:02 java

[root@sql56 conf]# /usr/local/mycat/bin/mycat status #查看服务

Mycat-server is running (6834).

[root@sql56 conf]# /usr/local/mycat/bin/mycat stop #停止服务

[root@sql56 ~]# /usr/local/mycat/bin/mycat help #查看命令

Usage: /usr/local/mycat/bin/mycat { console | start | stop | restart | status | dump }

#### 4处理异常

一个终端查看启动日志wrapper.log,

再开一个终端去启动mycat,则可以实时查看启动信息

[root@sql56 ~]#tail -f /usr/local/mycat/logs/wrapper.log

有时候启动不了是mycat服务机内存不够,重启主机,

#### 5客户机连接:

[root@client-50 ~]# mysql -h192.168.4.56 -P8066 -uroot -p123456

mysql> show databases;

+----------+

| DATABASE |

+----------+

| TESTDB |

+----------+

mysql>use TESTDB;

mysql> show tables; #此时的表都是不存在的

+------------------+

| Tables in TESTDB |

+------------------+

| company |

| customer |

| customer\_addr |

| employee |

| goods |

| hotnews |

| orders |

| order\_items |

| travelrecord |

+------------------+

9 rows in set (0.00 sec)

根据数据分片的配置做相应的设置

### 1.2.5测试数据分片

根据表使用的分片规则建表

schema.xml文件作用:配置mycat的真实库表,此文件定义了表规则

#### 测试全局”global”规则

例如:commpany表的规则为:global全局存放在dn1,dn2,dn3,且表里面必须有个ID字段为主键

<table name="company" primaryKey="ID" type="global" dataNode="dn1,dn2,dn3" />

**客户端登录56**

[root@client-50 ~]# mysql -h192.168.4.56 -P8066 -uroot -p123qqq...A

mysql> show databases;

+----------+

| DATABASE |

+----------+

| TESTDB |

+----------+

mysql> show tables;

+------------------+

| Tables in TESTDB |

+------------------+

| company |

| customer |

| customer\_addr |

| employee |

| goods |

| hotnews |

| orders |

| order\_items |

| travelrecord |

+------------------+

9 rows in set (0.00 sec)

mysql> desc company;

mysql> create table company(ID int primary key auto\_increment,cname char(50),address char(50),tel char(11));

Query OK, 0 rows affected (0.52 sec)

mysql> desc company;

+---------+----------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+---------+----------+------+-----+---------+----------------+

| ID | int(11) | NO | PRI | NULL | auto\_increment |

| cname | char(50) | YES | | NULL | |

| address | char(50) | YES | | NULL | |

| tel | char(11) | YES | | NULL | |

+---------+----------+------+-----+---------+----------------+

4 rows in set (0.01 sec)

mysql> insert into company(cname,address,tel) values("tarena","bejing","888888");

Query OK, 1 row affected (0.17 sec)

mysql> insert into company(cname,address,tel) values("tarena","bejing","888888");

Query OK, 1 row affected (0.08 sec)

**53 54 55都能看到客户端50登录56数据库后创建的company表以及插入的数据**

**以53为例:**

[root@sql53 ~]# mysql -uroot -p123456

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| db1 |

| mysql |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.00 sec)

mysql> use db1;

mysql> show tables;

+---------------+

| Tables\_in\_db1 |

+---------------+

| company |

+---------------+

1 row in set (0.00 sec)

mysql> select \* from company ;

+----+--------+---------+--------+

| ID | cname | address | tel |

+----+--------+---------+--------+

| 1 | tarena | bejing | 888888 |

| 2 | tarena | bejing | 888888 |

+----+--------+---------+--------+

2 rows in set (0.00 sec)

#### 测试枚举法”sharding-by-intfile”规则

<table name="employee" primaryKey="ID" dataNode="dn1,dn2,dn3"

rule="sharding-by-intfile" />

说明:必须有"employee"字段,和”ID”字段,且ID为主键,数据节点服务器为"dn1,dn2,dn3"

rule="sharding-by-intfile"表明--->rule.xml文件写了sharding-by-intfile这个规则

[root@sql56 ~]# cd /usr/local/mycat/conf/

[root@sql56 conf]# vim rule.xml

<tableRule name="sharding-by-intfile">

<rule>

<columns>sharding\_id</columns>

<algorithm>hash-int</algorithm> #定义规则名字

</rule>

</tableRule>

本文件往下看

<function name="hash-int" #定义规则使用的算法文件

class="io.mycat.route.function.PartitionByFileMap">

<property name="mapFile">partition-hash-int.txt</property>

</function>

[root@sql56 conf]# ls \*.txt

autopartition-long.txt auto-sharding-rang-mod.txt partition-range-mod.txt

auto-sharding-long.txt partition-hash-int.txt sharding-by-enum.txt

[root@sql56 conf]# vim partition-hash-int.txt

10000=0 #存入第一台服务器

10010=1 #存入第二台服务器

10020=2 #存入第三台服务器

在50上连接分片服务器之后进入TESTDB库创建表,并插入数据

[root@client-50 ~]# mysql -h192.168.4.56 -P8066 -uroot -p123qqq...A

mysql> create table employee(ID int primary key auto\_increment,sharding\_id int,name char(10),age tinyint);

Query OK, 0 rows affected (0.46 sec)

mysql> desc employee;

+-------------+------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-------------+------------+------+-----+---------+----------------+

| ID | int(11) | NO | PRI | NULL | auto\_increment |

| sharding\_id | int(11) | YES | | NULL | |

| name | char(10) | YES | | NULL | |

| age | tinyint(4) | YES | | NULL | |

+-------------+------------+------+-----+---------+----------------+

4 rows in set (0.01 sec)

mysql> insert into employee(sharding\_id,name,age) values(10000,"tom",30);

Query OK, 1 row affected (0.11 sec)

mysql> insert into employee(sharding\_id,name,age) values(10010,"tom",40);

Query OK, 1 row affected (0.04 sec)

mysql> insert into employee(sharding\_id,name,age) values(10020,"tom",50);

Query OK, 1 row affected (0.03 sec)

53:为10000=0数据库查看employee表数据

mysql> select \* from employee;

+----+-------------+------+------+

| ID | sharding\_id | name | age |

+----+-------------+------+------+

| 1 | 10000 | tom | 30 |

+----+-------------+------+------+

1 row in set (0.00 sec)

54:为10010=1数据库查看employee表数据

mysql> select \* from employee;

+----+-------------+------+------+

| ID | sharding\_id | name | age |

+----+-------------+------+------+

| 1 | 10010 | tom | 40 |

+----+-------------+------+------+

1 row in set (0.00 sec)

55:为10020=2数据库查看employee表数据

mysql> select \* from employee;

+----+-------------+------+------+

| ID | sharding\_id | name | age |

+----+-------------+------+------+

| 1 | 10020 | tom | 50 |

+----+-------------+------+------+

1 row in set (0.00 sec)

如果出现invalid sourcedata(无效数据源),配置文件或者授权用户出错

#### 添加新表配置数据分片

断开客户端,停止mycat

56mycat服务配置文件添加table

[root@sql56 conf]# /usr/local/mycat/bin/mycat stop

Stopping Mycat-server...

Stopped Mycat-server.

[root@sql56 conf]# vim /usr/local/mycat/conf/schema.xml

<table name="employee" primaryKey="ID" dataNode="dn1,dn2,dn3"

rule="sharding-by-intfile" />

<table name="employee2" primaryKey="ID" dataNode="dn1,dn2,dn3"

rule="sharding-by-intfile" /> #复制一个table

[root@sql56 conf]# /usr/local/mycat/bin/mycat start 启动服务

Starting Mycat-server...

50登录,查看TESTDB库里面的表

mysql> show tables;

+------------------+

| Tables in TESTDB |

+------------------+

| company |

| customer |

| customer\_addr |

| employee |

| employee2 |

| goods |

| hotnews |

| orders |

| order\_items |

| travelrecord |

+------------------+

10 rows in set (0.00 sec)

添加了新表employee2,规则是枚举法,按照枚举方式插入数据即可

mysql> create table employee2(ID int primary key auto\_increment,sharding\_id int,name char(10),age tinyint);

Query OK, 0 rows affected (0.55 sec)

mysql> insert into employee2(sharding\_id,name,age) values(10010,"jim",40);

Query OK, 1 row affected (0.13 sec)

只给100010--54数据库插入了数据,查询54的库即可

#### 添加新库

断开客户端,停止mycat

56mycat服务配置文件添加数据库bbsdb

**修改配置文件server.xml --添加一个**bbsdb**库**

[root@sql56 ~]# vim /usr/local/mycat/conf/server.xml

<user name="root">

<property name="password">123456</property>

<property name="schemas">TESTDB,bbsdb</property> #逗号分隔

<!-- 表级 DML 权限设置 -->

<!--

<privileges check="false">

<schema name="TESTDB" dml="0110" >

<table name="tb01" dml="0000"></table>

<table name="tb02" dml="1111"></table>

</schema>

</privileges>

-->

</user>

**修改配置文件schema.xml --添加两个(user ygtab)表**

[root@sql56 conf]# vim /usr/local/mycat/conf/schema.xml

<schema name="bbsdb" checkSQLschema="false" sqlMaxLimit="100">

<table name="user" primaryKey="ID" type="global" dataNode="dn1,dn2,dn3" />

<table name="ygtab" primaryKey="ID" dataNode="dn1,dn2,dn3"

rule="sharding-by-intfile" />

</schema>

然后按照相关规则添加

# 案例

案例1：搭建mycat 分片服务器

1 案例1：搭建mycat 分片服务器

1.1 问题

数据库主机 192.168.4.55 使用db1库存储数据

数据库主机 192.168.4.56 使用db2库存储数据

主机 192.168.4.54 运行mycat服务，逻辑库名称为test，连接用户名为admin，密码123456

在主机 192.168.4.254 访问测试配置

1.2 方案

准备四台主机，搭建mycat分片服务器，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。其中192.168.4.56作为mycat服务器，192.168.4.54和192.168.4.55作为数据库服务器，192.168.4.254作为客户端。如图-1所示：

图-1

数据分片的拓扑如图-2所示：

图-2

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：公共配置

1）关闭防火墙和selinux，配置yum源（系统源），这里不再操作

2）把54，55还原成独立数据库服务器，且只保留默认4个库其他都删除，停止56主机的mha管理服务，下载mycat 软件到56主机

3）在54（主机c2）和55（主机c1）上面创建db1和db2库

[root@c1 ~]# mysql -u root -p123456

mysql> create database db1; //c1上面创建db1库

Query OK, 1 row affected (0.00 sec)

[root@c2 ~]# mysql -u root -p123456

mysql> create database db2; //c2上面创建db2库

Query OK, 1 row affected (0.00 sec)

在54上面授权一个用户

mysql> grant all on \*.\* to admin@"%" identified by "123456";

Query OK, 0 rows affected, 1 warning (0.00 sec)

在55上面授权一个用户

mysql> grant all on \*.\* to admin@"%" identified by "123456";

Query OK, 0 rows affected, 1 warning (0.00 sec)

4）修改数据库的配置文件

注意：1代表不区分表名的大小写，0为区分大小写

主机c1上面：

[root@c1 ~]# vim /etc/my.cnf

[mysqld]

lower\_case\_table\_names=1 //表名忽略大小写

[root@c1 ~]# systemctl restart mysqld

主机c2上面：

[root@c2 ~]# vim /etc/my.cnf

[mysqld]

lower\_case\_table\_names=1

[root@c2 ~]# systemctl restart mysqld

5）在56主机上面安装JDK

[root@mycat ~]# rpm -qa | grep -i jdk //安装自带的即可

java-1.8.0-openjdk-1.8.0.131-11.b12.el7.x86\_64

java-1.8.0-openjdk-headless-1.8.0.131-11.b12.el7.x86\_64

copy-jdk-configs-2.2-3.el7.noarch

[root@mycat ~]# yum -y install java-1.8.0-openjdk

6）在56主机上面安装mycat

[root@mycat ~]# cd mysql/

[root@mycat mysql]# tar -xf Mycat-server-1.4-beta-20150604171601-linux.tar.gz //免安装，解压即可使用

[root@mycat mysql]# mv mycat/ /usr/local/

[root@mycat mysql]# ls /usr/local/mycat/

bin catlet conf lib logs version.txt

[root@mycat mysql]# cd /usr/local/mycat/

[root@mycat mycat]# ./bin/mycat --help

Usage: ./bin/mycat { console | start | stop | restart | status | dump }

7）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

log mycat启动日志和运行日志

wrapper.log mycat服务启动日志

mycat.log 记录SQL脚本执行后的报错内容

重要配置文件说明：

server.xml 设置连mycat的账号信息

schema.xml 配置mycat的真实库表

rule.xml 定义mycat分片规则

配置标签说明

<user>.. ..</user> 定义连mycat用户信息

<datanode>.. ..</datanode> 指定数据节点

<datahost>.. ..</datahost> 指定数据库地址及用户信息

查看server.xml配置文件

[root@mycat mycat]# cd conf/

[root@mycat conf]# vim server.xml

</system>

<user name="test"> //连接mycat服务时使用的用户名 test

<property name="password">test</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,服务器上没有这个库名，随便取，但要记住）

</user>

<user name="user">

<property name="password">user</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,不写这一行则是可读可写

</user>

修改schema.xml配置文件

[root@mycat conf]# vim schema.xml

<table name="travelrecord" dataNode="dn1,dn2" rule="auto-sharding-long" />

//travelrecord（逻辑上的，名字不能随便写，一般不动）表分片到数据节点dn1和dn2，dn1和dn2随便取的名字

<table name="company" primaryKey="ID" type="global" dataNode="dn1,dn2" />

<table name="hotnews" primaryKey="ID" dataNode="dn1,dn2" rule="mod-long" />

<dataNode name="dn1" dataHost="c1" database="db1" />

//数据节点对应的服务器 name="dn1"名称要与上面的对应 dataHost="c1"写本机主机名，database="db1"存在的数据库名,定义分片使用的库，所在的物理主机，真正存储数据的db1库在物理主机mysql55上

<dataNode name="dn2" dataHost="c2" database="db2" />

//定义分片使用的库，所在的物理主机，真正存储数据的db1库在物理主机mysql55上

指定c1名称主机对应的ip地址

<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="c1" url="192.168.4.55:3306" user="admin"

password="123456">

//访问数据库时，mycat服务连接数据库服务器时使用的用户名和密码

<!-- can have multi read hosts -->

</writeHost>

</dataHost>

指定c2名称主机对应的ip地址

<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="c2" url="192.168.4.54:3306" user="admin"

password="123456">

//访问数据库时，mycat服务连接数据库服务器时使用的用户名和密码

<!-- can have multi read hosts -->

</writeHost>

</dataHost>

8）添加PATH路径

[root@mycat conf]# export PATH=/usr/local/mycat/bin:$PATH

[root@mycat conf]# echo "PATH=/usr/local/mycat/bin:$PATH" >> /etc/profile

[root@mycat conf]# source /etc/profile

[root@mycat conf]# echo $PATH

/usr/local/mycat/bin:/usr/local/mycat/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin

[root@mycat conf]# which mycat

/usr/local/mycat/bin/mycat

9）启动服务并查看端口

[root@mycat conf]# mycat start

Starting Mycat-server...

[root@mycat conf]# netstat -antup | grep :8066

tcp6 0 0 :::8066 :::\* LISTEN 6421/java

[root@mycat conf]# ps -C java

PID TTY TIME CMD

6421 ? 00:00:04 java

用admin用户登录

[root@client ~]# mysql -h192.168.4.54 -uadmin -p123456

mysql> show processlist;

+----+-------+--------------------+------+---------+------+----------+------------------+

| Id | User | Host | db | Command | Time | State | Info |

+----+-------+--------------------+------+---------+------+----------+------------------+

| 3 | admin | 192.168.4.56:34580 | db2 | Sleep | 80 | | NULL |

| 4 | admin | 192.168.4.56:34570 | db2 | Sleep | 100 | | NULL |

| 5 | admin | 192.168.4.56:34572 | db2 | Sleep | 40 | | NULL |

| 6 | admin | 192.168.4.56:34562 | db2 | Sleep | 30 | | NULL |

| 7 | admin | 192.168.4.56:34564 | db2 | Sleep | 90 | | NULL |

| 8 | admin | 192.168.4.56:34566 | db2 | Sleep | 60 | | NULL |

| 9 | admin | 192.168.4.56:34574 | db2 | Sleep | 70 | | NULL |

| 10 | admin | 192.168.4.56:34576 | db2 | Sleep | 10 | | NULL |

| 11 | admin | 192.168.4.56:34578 | db2 | Sleep | 20 | | NULL |

| 12 | admin | 192.168.4.56:34568 | db2 | Sleep | 50 | | NULL |

| 14 | admin | 192.168.4.51:58354 | NULL | Query | 0 | starting | show processlist |

+----+-------+--------------------+------+---------+------+----------+------------------+

11 rows in set (0.00 sec)

[root@client ~]# mysql -h192.168.4.55 -uadmin -p123456

mysql> show processlist;

+----+-------+--------------------+------+---------+------+----------+------------------+

| Id | User | Host | db | Command | Time | State | Info |

+----+-------+--------------------+------+---------+------+----------+------------------+

| 3 | root | localhost | NULL | Sleep | 2352 | | NULL |

| 4 | admin | 192.168.4.56:45148 | db1 | Sleep | 2 | | NULL |

| 5 | admin | 192.168.4.56:45150 | db1 | Sleep | 62 | | NULL |

| 6 | admin | 192.168.4.56:45160 | db1 | Sleep | 12 | | NULL |

| 7 | admin | 192.168.4.56:45162 | db1 | Sleep | 92 | | NULL |

| 8 | admin | 192.168.4.56:45152 | db1 | Sleep | 32 | | NULL |

| 9 | admin | 192.168.4.56:45154 | db1 | Sleep | 42 | | NULL |

| 10 | admin | 192.168.4.56:45156 | db1 | Sleep | 22 | | NULL |

| 11 | admin | 192.168.4.56:45158 | db1 | Sleep | 82 | | NULL |

| 12 | admin | 192.168.4.56:45164 | db1 | Sleep | 52 | | NULL |

| 13 | admin | 192.168.4.56:45166 | db1 | Sleep | 72 | | NULL |

| 14 | admin | 192.168.4.51:32796 | NULL | Query | 0 | starting | show processlist |

+----+-------+--------------------+------+---------+------+----------+------------------+

12 rows in set (0.00 sec)

[root@mycat conf]# ls /usr/local/mycat/logs/

mycat.log mycat.pid wrapper.log //wrapper.log为错误日志

[root@mycat conf]# ldconfig -v //更新加载的模块

10）客户端访问

命令： mysql -hmycat主机的IP -P端口号 -u用户 -p密码

[root@clent ~]# mysql -h192.168.4.56 -P8066 -utest -ptest

mysql> show databases;

+----------+

| DATABASE |

+----------+

| TESTDB |

+----------+

1 row in set (0.00 sec)

mysql> USE TESTDB;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql>

mysql> show tables;

+------------------+

| Tables in TESTDB |

+------------------+

| company |

| customer |

| customer\_addr |

| employee |

| goods |

| hotnews |

| orders |

| order\_items |

| travelrecord |

+------------------+

9 rows in set (0.00 sec)

11）客户端测试：

mysql> create table employee(id int not null primary key,name varchar(100),sharding\_id int not null);

Query OK, 0 rows affected (0.10 sec)

mysql> insert into employee(id,name,sharding\_id) values(1,"bob",10000);

Query OK, 1 row affected (0.03 sec)

mysql> insert into employee(id,name,sharding\_id) values(1,"lucy",10010);

Query OK, 1 row affected (0.02 sec)

mysql> select \* from employee;

+----+------+-------------+

| id | name | sharding\_id |

+----+------+-------------+

| 1 | bob | 10000 |

| 1 | lucy | 10010 |

+----+------+-------------+

2 rows in set (0.06 sec)

12）在c1上面查看结果

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| db1 |

| mysql |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.00 sec)

mysql> use db1;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> show tables;

+---------------+

| Tables\_in\_db1 |

+---------------+

| employee |

+---------------+

1 row in set (0.00 sec)

mysql> select \* from employee; //查看结果

+----+------+-------------+

| id | name | sharding\_id |

+----+------+-------------+

| 1 | bob | 10000 |

+----+------+-------------+

1 row in set (0.00 sec)

13）在c2上面查看结果

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| db2 |

| mysql |

| performance\_schema |

| sys |

+--------------------+

5 rows in set (0.00 sec)

mysql> use db2;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

mysql> show tables;

+---------------+

| Tables\_in\_db2 |

+---------------+

| employee |

+---------------+

1 row in set (0.00 sec)

mysql> select \* from employee;

+----+------+-------------+

| id | name | sharding\_id |

+----+------+-------------+

| 1 | lucy | 10010 |

+----+------+-------------+

1 row in set (0.00 sec)