**一、搭建数据库服务器**

服务器 （硬件主机 云主机）

什么是数据库？-------存储数据的仓库

部署一台数据库，提供数据存储服务

装包-----配置----启服务

软件包类型：

A、官网下载 b、光盘自带

软件包来源？

-----软件包封包类型（rpm 源码）？

源码可以自定义配置和安装 必须解决依赖关系

Rpm软件是封装好不可以自定义安装 安装卸载简单

使用开源软件还是商业软件？

开原软件：不等于免费

商业软件：完全需要付费的

软件版本的选择，提供数据库服务的软件有哪些？

关系型数据库（按照一定的规则存储数据）：

Mysql Oracle SQL server DB2

非关系型数据库：redis mongodb memcached

Mysql的特点：

开原且跨平台：unix linux windows 均可安装

支持主流的编程语言：php java python ruby perl

可移植性好：系统只要有c运行环境就可以

数据库服务通常和网站服务一起使用：LAMP LNMP

数据库服务用来存储什么杨的数据：

装包：

1. 查询当前系统是否安装 mariadb ----------卸载 rpm-e --nodeps mariadb-libs

如果已安装mariadb-server 需要先停服务 再卸载

.删除对应的配置文件 rm -rf /var/lib/mysql/\*

Rm -rf /etc/my.cnf

1. yum -y install perl-JSON

Rpm -uvh mysql-community-\*.rpm

1. 启服务：systemctl start mysqld

开机自启： systemctl enable mysqld

查看服务信息

端口号：3306 netstat -utnlp | grep mysqld

进程mysqld : ps -C mysqld

/var/lib/mysql

**二、把数据存储到数据库服务器上的过程**

1. 连接数据库服务器

Mysql -h ip -u用户名 -p密码

第一次进入数据库系统强制修改密码，有两种设置密码的方式：

一种：直接设置 （密码复杂程度需要符合系统要求）

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

二种：

修改系统验证长度：mysql> set global validate\_password\_policy=0

修改系统密码长度：mysql> set global validate\_password\_length=6;

更改系统密码：mysql> alter user root@"localhost" identified by "123456";

1. 创建库，库相当于系统的文件夹

Show databases; #显示已有的库

Create database 库名; #新建库

Select database(); #显示当前所在的库

Use 库名; #切换库

Show tables; #显示已有的表

Drop database 库名; #删除库

库名的命名规则可以是数字、字母、下划线，但不能是纯数字。

1. 创建表，相当于系统文件，表必须要创建再库里

Create database studb; #创建库

Create table studb.stuinfo( #创建表

Name char(15), #姓名

Sex char(10), #性别

Age int, #年龄

Tel char(11) #电话

);

mysql> desc studb.stuinfo; #查看表结构

mysql> select \* from studb.stuinfo; #查看表记录

1. 插入数据

Select \* from 表名; #查看表记录

Insert info 表名 values（值列表）; #插入表记录

Update 表名 set 字段=值; #修改表记录

Delete from 表名; #删除表记录

mysql>insert into studb.stuinfo

values("bob","boy",21,"888888"),("zhaozhibin","boy",19,"6666666");

1. 断开连接

Quit

表中使用中文的命令

用sql命令管理数据库，sql命令的使用规则和分类？

Create table 学生表（姓名 char（10），年龄int）default charset=utf8;

Show create table 学生表;

**三、Mysql数据类型**

数值类型：整型 浮点型

数值类型的宽度是显示宽度，不能限制字段附值

整数类型 小数类型

字符类型：char varchar text blob

枚举类型：enum set

日期和时间类型：年 -----------year yyyy

日期 ---------- date yyyymmdd 20181138

日期时间---------

mysql> create table t1(

-> name char(15),

-> birtday date,

-> work time,

-> s\_year year,

-> meetting dtatetime

-> );

Mysql服务时间函数

Select now(); #获取当前系统时间

Select year( now() ); #获取当前年份

Select year( 20191224 ); #获取2019年

Select month(now()); #获取系统月份

Select month(20191224); #获取指定时间月份

Select day(20191224); #获取指定时间几号

Select day(now()); #获取系统时间几号

Select date(now()); #获取系统日期

Select time(now()); #获取系统时间

Enum 从给定集合中选择单个值

Set 从给定集合中选择多个值

实验：使用关键字段zerofill时，填0替代补位

Create table t5(

Id int(4) unsigned zerofill,

Age int(2) unsigned zerofill

);

desc t5;

写数据：

Insert into t5 values(5,7);

Select \* from t5;

Insert into t5 values(21,39);

Insert into t5 values(199,211);

Insert into t5 values(9999,9999);

Select \* from t5

**一、约束条件**

作用：控制如何给字段附值

Null | Key | Default | Extra

是否允许赋空值 建值 默认值 额外设置

实验：

mysql> create table t7( Name char(15) not null, Class char(7) default "nsd1804", Age tinyint(2) not null default 19, Sex enum ("boy","girl") not null default "boy" );

Insert into t7 (name)values()

mysql> insert into t7 values("jim",null,22,"boy");

mysql> insert into t7 values(null,null,22,"boy");

mysql> insert into t7 values("bob2","nsd1803",21,"girl");

mysql> insert into t7(name)values("bob");

mysql> select \* from t7;

mysql> create table t7( Name char(15) not null, Class char(7) default "nsd1804", Age tinyint(2) not null default 19, Sex enum ("boy","girl") not null default "boy" );

**二、修改表结构：对已经创建好的表的结构做修改；**

**A.添加新字段 add：**

mysql> alter table t8 add email varchar(50) default "student@tedu.cn",add qq char(11);

在表中任意列插入新增列：

mysql> alter table t8 add stu\_mum char(9) first,add likes set("eat","sleeps","game","film") after sex; #在第一列插入stu\_mum列并在sex列后面插入likes列

**B.删除已有字段（drop）**

mysql> alter table t8 drop email; #删除t8表中email列

mysql> alter table t8 drop email,drop qq; #同时删除email和qq列

**C.修改已有字段类型（modify）**

\*修改字段类型时，若新的类型与字段已存储数据冲突，不允许修改。不修改的部分要原样写一遍，否则就还原。

mysql> alter table t8 modify sex enum('boy','girl','no') not null default "boy" after name;

#修改t8表中sex列，新增类型no并把sex移动到name列后面

\*因修改的字段列原本存在内容 因此需要把原有内容再抄写一遍

**D.修改字段名（change）**

mysql> alter table t8 change stu\_mum stu\_id varchar(9);

#把t8表stu\_mum字段修改为stu\_id

**E.修改表名(rename)**

mysql> alter table t8 rename stuinfo; #把表名由t8修改为stuinfo

**三、键值**

作用：约束如何给字段赋值

普通索引 ：一个表中可以有多个index字段

1.索引介绍：给字段值排序的树形结构，类似于“书的目录”

2.索引优点：加快查询数据的速度

3.索引缺点：减慢写入数据的速度，占用磁盘存储空间

4.使用索引：mysql> select \* from t1 where name="bob";

5.使用规则

6.查看：desc 表; 如果有索引那么key列的标记应为mul

mysql> show index from stuinfo\G; #查看stuinfo表中索引并以竖列显示

**A.索引：**

把已有表的字段设置为index字段

mysql> create index aaa on stuinfo(name); #把stuinfo表中name字段设置索引名为aaa

mysql> show index from stuinfo\G; #查看stuinfo表中索引并以竖列显示

mysql> desc stuinfo; #查看索引

创建表时指定index字段

mysql> create table t9(

-> name char(15) not null default "",

-> class char(7) default "nsd1804",

-> age tinyint(2) not null default 19,

-> sex enum("boy","girl") not null default "boy",

-> index(name),index(class) #把t9表中name、class字段设置为索引

-> );

mysql> desc t9; #查询t9表中哪些字段有索引信息

mysql> show index from t9\G; #查询t9表中索引字段的详细信息

mysql> drop index aaa on stuinfo; #删除名为aaa的索引

**B.主键（primary key）**

主键在表结构中的显示为PRI，且主键默认值不能为空，

一个表中有且只能有一个主键

创建主键

建表时创建主键：

方法一：

mysql> create table t10(

-> stu\_id char(9) primary key, #创建主键

-> name char(10),

-> age int(2)

-> );

方法二：

mysql> create table t11(

-> stu\_id char(9),

-> name char(10),

-> age int(2),

-> primary key(stu\_id) #创建主键

-> );

mysql> desc t10; #查看表结构

mysql> desc t11; #查看表结构

在已有表里创建主键字段

mysql> alter table t10 add primary key(stu\_id); #设置主键的字段里不能有重复的值 否则会创建失败

删除主键

mysql> alter table t10 drop primary key; #删除他t10表中的主键

**C.复合主键**

表中的多个字段一起做主键，赋值时，只要主键字段的值不允许同时重复。

mysql> create table t12(

-> name char(15),

-> class char(7),

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

-> primary key(name,class) #创建复合型主键

-> );

mysql> desc t12; #查看

验证：

mysql> insert into t12 values("bob","nsd1804","yes"); #第一条记录创建成功

mysql> insert into t12 values("bob","nsd1804","no");

ERROR 1062 (23000): Duplicate entry 'bob-nsd1804' for key 'PRIMARY' #第二条创建失败

删除复合主键：

mysql> alter table t12 drop primary key; #删除t12表中的复合主键

**D.让字段的值自动增长 做自加1计算。**

通常和auto\_increment连用 让字段的值自动增长 做自加1计算。

Primary key && 数值 #需要满足的条件

向表中插入记录时不给自动增长的字段赋值，字段的值时如何获得的呢，用当前字段最大的值+1后把结果作为当前新记录字段的值。

实验：

mysql> create table t1(

-> id int(2) primary key auto\_increment, #设置数值自动增长

-> name char(15) not null,

-> age tinyint(2) unsigned default 19,

-> pay float(7,2) default 26800

-> );

验证：

mysql> insert into t1(name,age,pay) values(tom,21,18000);

mysql> insert into t1(name,age,pay) values("tom",21,18000);

mysql> insert into t1(name,age,pay) values("lucy",23,18800);

mysql> select \* from t1;

mysql> insert into t1 values(7,"jerry",23,18800);

mysql> insert into t1(name,age,pay) values("xixi",26,15000);

mysql> select \* from t1;

**E.外键（foreignkey）**

让当前表字段的值在另一个表中字段值的范围内选择，

使用规则：表的存储引擎必须时innodb；字段的类型要一致；被参照类型必须是索引的一种

Create table yginfo（）engine=innodb #指定存储引擎

A.创建外键

实验：

建立员工信息表，设置表规则--自动增长 表引擎：

mysql> create table yginfo(

>yg\_id int(2) primary key auto\_increment,

>name char(15)

>)engine=innodb;

新建员工信息：

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

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

mysql> insert into yginfo(name) values("lucy");

mysql> select \* from yginfo;

#共创建5个员工信息

建立员工工资信息表，设置表规则---自动增长 表引擎：

mysql> create table gztab(

-> gz\_id int(2),

-> pay float(7,2),

-> foreign key(gz\_id) references yginfo(yg\_id)

#设置自动增长且数值参考yginfo表中的yg\_id字段

-> on delete cascade on update cascade

#设置同步删除 同步更新

-> )engine=innodb;

#设置表引擎

验证：

mysql> insert into gztab values(1,50000);

Query OK, 1 row affected (0.02 sec)

mysql> insert into gztab values(2,60000);

Query OK, 1 row affected (0.06 sec)

mysql> insert into gztab values(3,70000);

Query OK, 1 row affected (0.02 sec)

mysql> insert into gztab values(4,80000);

Query OK, 1 row affected (0.03 sec)

mysql> insert into gztab values(5,90000);

Query OK, 1 row affected (0.02 sec)

mysql> insert into gztab values(6,100000);

ERROR 1264 (22003): Out of range value for column 'pay' at row 1 #验证失败

#给员工1-5号发工资均成功 当给编号为6的号员工发工资时 显示失败 因为不存在彼岸为号6的员工

实验：

mysql> update yginfo set yg\_id=7 where yg\_id=2;

#同步将yginfo表中编号为2的改为编号为7

mysql> delete from yginfo where yg\_id=3;

#同步删除yginfo表中编号为3的信息

验证

mysql> select \* from yginfo;

mysql> select \* from gztab;

#验证员工信息表的编号和工资信息表的编号会同步变动

扩展 ：为了在发工资时不会重复发送和发送空值 可以把工资表的编号进行设置成主键

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

B.删除外键

mysql> show create table gztab; #查询外键名称

mysql> alter table gztab drop foreignkey gztab\_ibfk\_1; #删除外键gztab\_ibfk\_1

**四、mysql存储引擎**

1.1 存储引擎介绍: 是mysql数据库软件自带的功能程序，

每种存储引擎的功能和数据存储方式也不同

存储引擎就处理表的处理器

1.2 查看数据库服务支持的存储引擎有那些？

mysql> show engines;

InnoDB DEFAULT

1.3 查看已有的表使用的存储引擎

show create table 表名；

1.4 修改数据库服务默认使用的存储引擎

]#vim /etc/my.cnf

[mysqld]

default-storage-engine=myisam

:wq

]# systemctl restart mysqld

1.5 修改表使用的存储引擎，或 建表时指定表使用的存储引擎

alter table 表 engine=存储引擎名；

create table 表（字段列表）engine=存储引擎名；

1.6 常用存储引擎的特点

innodb特点：

支持事务 、 事务回滚 、行级锁 、外键

存储方式： 一个表对应2个存储文件

表名.frm 表结构

表名.ibd 数据和索引

myisam特点

不支持事务 、 事务回滚、外键

支持表级锁

存储方式： 一个表对应3个存储文件

表名.frm 表结构

表名.MYD 数据

表名.MYI 索引

事务：对数据库服务的访问过程（连接数据库服务器 操作数据 断开连接）

事务回滚 ： 在事务执行过程中，任何一步操作失败，都会恢复之前的所有操作。

插卡 （与数据库服务器建立连接）

转账： 对方卡号 888888

金额 50000

ok

提示转账成功 -50000 +50000

提示转账失败 +50000

退卡

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

cd /var/lib/mysql/

ib\_logfile0 -|

|------> 记录SQL命令

ib\_logfile1 -|

insert into t1 values(8888);

ibdata1 ----> 数据源（sql命令执行后产生的数据信息）

锁粒度：

表级锁（myisam）给整张表加锁

行级锁 (innodb) 只给表中当前被操作行加锁

锁的作用：解决对表的并发访问冲突问题。

select \* from t1 where id <=20;

insert

delete from t1;

update t1 set name="bob" where name="lucy";

update t1 set name="tom" where name="jerry";

锁类型

读锁 （共享锁）

写锁 （互斥锁）

1.7 建表时如何决定表使用那种存储引擎

执行写操作多的表适合使用innodb存储引擎，可以并发访问。

执行查操作多的表适合使用myisam存储引擎,可以节省系统资源

**一、数据导入与导出**

数据导入：把系统文件的内容存储到数据库服务器的表里。

数据导出：把数据库服务器的表里的记录存储到系统文件里。

系统导出、导入数据文件存储路径默认是：/var/lib/mysql-files/

**A.导入数据**

mysql> show variables like "secure\_file\_priv"; #查看文件默认路径

[root@host50 ~]# mkdir /mydata #新建文件

[root@host50 ~]# chown mysql /mydata/ #修改所有者

[root@host50 ~]# vim /etc/my.cnf #修改配置文件

[mysqld]

secure\_file\_priv="/mydata" #自定义文件路径

[root@host50 ~]# systemctl restart mysqld #重启服务

[root@host50 ~]# systemctl enable mysqld #开机自启

mysql> show variables like "secure\_file\_priv"; #查看文件默认路径

导入数据的步础：

1. 把系统文件拷贝到指定目录下：

mysql> system cp /etc/passwd /mydata/

mysql> system ls /mydata

passwd

2.在数据库里新建表文件

mysql> create table db3.user(

-> name char(50),

-> password char(1),

-> uid int(2),

-> gid int(2),

-> comment varchar(150),

-> homedir char(150),

->shell char(50)，

-> index(name) #以name字段为索引

-> );

1. 导入数据

mysql> load data infile "/mydata/passwd" into table db3.user fields terminated by ":" lines terminated by "\n";

#把/mydata/passwd下的文件导入到db3库下的user表中，以”:”为分隔符 以“\n”为换行符

mysql> select \* from user;

1. 在user表格前加入以行id字段

mysql> alter table user add id int(2) primary key auto\_increment first;

1. 验证查询第15行内容

mysql> select \* from user where id=15;

**B.导出数据**

mysql> select \* from t1 into outfile "/mydata/t1.txt";

#把t1表的数据导出到/mydata/t1.txt

mysql> select \* from t1 into outfile "/mydata/t2.txt" fields terminated by "#";

#把t1表的数据导出到/mydata/t2.txt，以#为分隔符

mysql> select \* from t1 into outfile "/mydata/t3.txt" fields terminated by "#" lines terminated by "!!!";

#把t1表的数据导出到/mydata/t3.txt 以#为字符间隔 ！！！为换行符号

**二、管理表记录**

**A.添加新纪录：insert into**

mysql> insert into user values (45,"bob","x",2000,2000,"student user","/home/bob","/bin/bash");

**B.查询记录：select**

mysql> select \* from user;

mysql> select name,uid,shell from user;

mysql> select name,uid,shell from user where id=1;

mysql> select name,uid,shell from user where shell="/bin/bash";

**C.更新记录字段：update**

Update 表 set 字段名=值，字段名=值，.........where 条件;

mysql> select name,uid,shell from user;

mysql> update user set password="A",gid=1000;

mysql> update user set password="X" where id=1;

**D.删除记录：delete**

删除表中的行 在生产环境中步要给程序员删除权限

mysql> select \* from t1;

mysql> delete from t1;

mysql> delete from user where name="bob"

mysql> select \* from user where name="bob";

**三、对记录做select update delete时可以使用的条件的标识方式：**

**A.数值比较 > >= < <= = !=**

Where 字段名 符号 数值

mysql> Select \* from user where id <= 10;

mysql> select name,uid from user where uid=9;

mysql> select name,uid,gid from user where uid=gid;

**B.字符比较 = !=**

Where 字段名 符号 “值”

mysql> select name from user where name="apache";

mysql> select name,shell from user where shell !="/bin/bash";

mysql> update user set gid=2000 where shell="/sbin/nologin";

**C.匹配空 is null，is not null**

mysql> select name,shell from user where shell is null;

mysql> select name,shell from user where shell is not null;

mysql> update user set password=null where name="bin";

**D.逻辑比较 and(与) or(或) ! Not(非)**

mysql> select \* from user where name="root" and uid=1 and shell="/bin/bash";

mysql> select name,uid from user where name="root" or uid=1;

mysql> select \* from user where name="root" and uid=1 and shell="/bin/bash";

#当或与非同时使用时先满足与的要求，再满足或的条件。

mysql> select name,uid from user where (name="root" or name="bin") and uid=1;

#括号的作用是用来提升优先级的，先执行括号内部的条件

**E.范围内匹配**

In 在...里

not in 不在...里

Between 数字1 and 数字2 在....之间

Distinct 字段名 去重显示

mysql> select name from user where name in("root","sync","lucy","bob");

#在user表name字段中查找root sync lucy bob

mysql> select name,uid from user where uid in(1,7,27,1000);

mysql> select name,shell from user where shell not in ("/bin/bash","/sbin/nologin");

mysql> select \* from user where uid between 10 and 20;

#在user表中查找uid在10到20之间的行

mysql> select \* from user where uid >=10 and uid <=20;

#在user表中查找uid在10到20之间的行

**F.Distinct(只适合select使用)：去掉字段的重复值显示**

mysql> select distinct shell from user; #显示user表中shell字段不重复的值

mysql> select distinct shell from user where uid <=50;

#显示user表中不重复且小于等于50的值

**四、高级匹配条件**

**A.模糊条件**

Where 字段名 like ‘表达式’;

“\_” 匹配一个字符 “%”匹配0～n个字符

mysql> select name from user where name like '\_\_'; #匹配name字段中只有两个字符的项

mysql> select name from user where name like '%a%'; #匹配name字段字段中至少包含一个a的

mysql> select name from user where name like '%\_\_\_%'; #匹配name字段中至少包含三个字符的

mysql> select id,name from user where name like '%'; #不匹配空

**B.正则匹配**

Where 字段名 regexp ‘正则表达式’

mysql> select name from user where name regexp "^a.\*t$"

#查询user表中name字段中a开头t结尾的用户

mysql> select name from user where name regexp "[0-9]"

#查询user中name字段中包含数字的用户

**C.四则运算 + - \* / %**

mysql> update user set uid=uid+1 where id<=10;

#把user表中id值小于等于10的用户的uid加一

mysql> select name,uid,gid,uid+gid jieguo from user where name="root";

#把user表中root用户的uid和gid相加并输出结果

mysql> select name,uid,gid,(uid+gid)/2 pjz from user where name="root";

mysql> select name,age,2018-age s\_year from user where name="root";

**D.聚集函数（mysql服务自带的对数据做统计的命令）**

字段类型必须是数值类型

Avg(字段名) 统计字段的平均值 Sum(字段名) 统计字段之和

Min(字段名) 统计字段最小值 Max(字段名) 统计字段最大值

Count(字段名) 统计字段值个数

mysql> select sum(uid) from user;

mysql> select sum(gid) from user where id<=10;

mysql> select avg(gid) from user where id<=10;

mysql> select min(gid) from user where id<=10;

mysql> select max(gid) from user where id<=10;

mysql> select count(name) from user where shell="/bin/bash";

#统计shell为/bin/bash的用户的个数有多少个

mysql> select sum(uid),avg(uid),max(uid),min(uid) from user;

mysql> select count(\*) from user;

**五、操作查询结果**

**A.分组、排序查询**

Sql查询 Group by 字段名； #查询分组

sql查询 order by 字段名 排序方式; # 排序 asc升序 desc 降序

mysql> select shell from user group by shell;

#查询user表中shell字段，并按shell字段进行分组

mysql> select name,uid from user where uid>=10 and uid<=500 order by uid desc;

#查询user表中uid大于等于10且小于等于500的结果按uid字段进行降序排列

**B.Limit限制查询结果显示的行数**

Sql查询 limit 数字; # 显示查询结果的前几行

Sql查询 limit 数字1，数字2; #显示指定范围内的行;

mysql> select name,shell from user where uid <=500 limit 2;

#查询user表中uid小于等于500的结果中的前两行

mysql> select name,shell from user limit 5;

mysql> select name,shell from user where uid <=500 limit 1,5;

#查询user表中uid小于等于500的结果中的第2行到后5行

**C.Having 条件**

#在查询结果里查找数据

Sql查询 having 条件;

mysql> select name from user where id<=10 having 条件;

mysql> select name,uid from user where uid>=1000 having uid=65534;

mysql> select name,uid from user where uid>=1000 having name="bob";

#查询user表中uid大于等于1000中name等于bob的结果

**一、多表操作**

**A.复制表**

（作用：备份表、快速建表） 建表时源表的键值不会复制给新表

Create table 表名 sql查询；

mysql> create table user2 select \* from db3.user;

#复制db3库中的user表所有数据到当前的user2表

mysql> create table user3 select name,uid,shell from db3.user order by uid desc limit 5;

#复制db3库中user表的降序的前5行数据到当前的user3表

mysql> create table user4 select \* from db3.user where 1=2;

#复制db3库中当1=2时所有数据的值到当前user4表 （勾成不等式条件，形成空表复制）

**B.多表查询**

格式一：Select 字段名列表 fron 表名列表 where 条件;

mysql> create table t1 select name,uid,shell from db3.user limit 3;

mysql> create table t2 select name,password,uid,homedir from db3.user limit 5;

mysql> select \* from t1,t2;

mysql> select t1.name,t2.name from t1,t2;

mysql> select t1.\*,t2.password,t2.homedir from t1,t2;

mysql> select t1.\*,t2.uid,t2.name from t1,t2 where t1.uid=t2.uid;

笛卡尔集：t1表行数\*t2行数共计15行

**C.嵌套查询 把内层的查询结果作为外层查询的查询条件**

在生产环境中能不用嵌套时查询就不用

Select 字段名列表 from 表名 where 条件 (select 字段名列表 from 表名 where 条件)；

mysql> select avg(uid) from db3.user; #显示user表中uid字段的平均值

mysql> select name,uid from db3.user where uid<(select avg(uid) from db3.user);

#显示user表中 uid小于user表中uid字段平均值的name字段

mysql> select name from db3.user where name in (select name from db4.t1);

#查询db3库下的user表中name字段的值有哪些在db4库中t1表中存在

**D.连接查**

左连接查询：以左边的表为主显示查询结果 left join

Select 字段名列表 from 表名 left join 表名 on 条件;

右连接查询：以右边的表为主显示所有查询结果 right join

Select 字段名列表 from 表名 right join 表名 on 条件;

mysql> select \* from t3 left join t4 on t3.uid = t4.uid;

mysql> select \* from t3 right join t4 on t3.uid = t4.uid;

mysql> select t3.name,t4.name from t3 right join t4 on t3.uid = t4.uid;

**二、数据库管理工具的使用**

**A.图形管理工具的安装**

1.装包：httpd或nginx php php-mysql mariadb mariadb-server mariadb-devel

2.启动服务

3.解压软件：/phpMyAdmin-2.11.11-all-languages.tar.gz

4.把解压后的文件移动到/var/www/html/phpadmin

5.创建配置文件，指定管理数据库服务器

[root@host50 ~]# cd /var/www/html/phpadmin

[root@host50 phpadmin]# cp config.sample.inc.php config.inc.php

[root@host50 phpadmin]# vim config.inc.php

17 $cfg['blowfish\_secret'] = 'pljabc';

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

1. 客户访问 http：//192.168.4.50./phpadmin

**B.设置数据库管理员root密码**  
Mysqladmin -hlocalhost -uroot -p password “新密码”；

**C.恢复数据库管理员密码**

1. 关服务：systemctl stop mysqld #关服务
2. Vim /etc/my.cnf #修改配置文件

[mysql]

Skip\_grant\_tables #新增

1. systemctl srart mysqld #重启服务
2. [root@host50 ~]# mysql
3. mysql> select host,user,authentication\_string from mysql.user; #查询user表中的表记录
4. mysql> update mysql.user set authentication\_string=password("123") where user="root" and host="localhost"; #修改密码
5. mysql> flush privileges; #加载
6. mysql> quit #退出登陆

[root@host50 ~]# vim /etc/my.cnf #修改配置文件

[mysql]

#Skip\_grant\_tables #注释掉

[root@host50 ~]# systemctl restart mysqld #重启服务

9.[root@host50 ~]# mysql -uroot -p123 #验证新密码登陆

**三、用户授权**

在数据库服务器上添加新的连接用户名，默认只有数库库管理员root用户在数据库服务器在本机登陆有操作权限。默认情况下管理员只能从数据库本机连接服务，没有其他用户能够连接数据库服务器了。

**A.授权命令格式：**

Grant 权限列表 on 数据库 to 用户名@“客户端地址” identified by “密码” [with grant option];

With grant option #让新添加的用户连接服务器后，也有授权权限。

Identified by “密码” #新添加的用户连接数据库服务器时，使用的密码

客户端地址 作用：网络中的那些主机可以使用新添加的用户连接数据库服务器。有如下表示方式：

所有主机 %

网段 192.168.4.% 指定主机 192.168.4.51 本机 localhost

主几名 pc1.tedu.cn 域名 %.tedu.cn

用户名：客户端主机连接数据库服务器时使用的名字，授权时自定义即时名字要有标识性

数据库名：新添加的连接的用户，访问后可以操作的库，表示方式如下：

\*.\* 所有库 所有表 库名.\* 库下所有的表 库名.表名 某张表

权限列表：新添加的连接用户，对可以操作的库的访问权限，权限的表示方式如下：

All 所有权限 命令，命令 某种权限（例如：select insert delete）

**B.授权库记录表信息**

User表 已经添加的连接用户 db 记录已添加的连接用户对库的访问权限

Tables\_priv 记录已添加的连接用户对表的访问权限

Columns\_priv 记录已添加的连接用户对表字段的访问权限

mysql> select user,host from mysql.user;

mysql> mysql> select \* from mysql.tables\_priv;

mysql> select \* from mysql.db;

mysql> select \* from mysql.columns\_priv;

Show grants for 用户@“客户端地址”; #查看授权用户的访问权限

mysql> show grants for root@"localhost";

mysql> select \* from user where user="root" and host="localhost"\G;

实验一：

服务器添加新用户 设置权限：

mysql> grant select,insert on db3.\* to admin@"192.168.4.%" identified by "123qqq...A";

#设置在192.168.4.0网段上授权查询、添加权限 密码设置为123qqq...A

客户端访问验证：

mysql> select @@hostname; #查看当前连接用户主机名

mysql> select user(); #查看当前用户信息

mysql> show grants; #登陆的用户查看自己的访问权限

实验二：

服务器端（192.168.4.50）

在50服务器授权主机52可以使用root连接自己，对所有库表有完全权限且有授权权限 登陆密码时123qqq.A

mysql> grant all on \*.\* to root@"192.168.4.52" identified by "123qqq...A" with grant option;

客户端验证（192.168.4.52）

[root@host52 ~]# mysql -h192.168.4.50 -uroot -p123qqq...A

实验三：

服务端（192.168.4.50）添加权限

mysql> grant select,update(name,uid) on db3.user to webuser@"%" identified by "123qqq...A";

客户端（192.168.4.51）测试

[root@host51 ~]# mysql -uwebuser -p123qqq...A;

mysql> show grants;

权限撤销

删除新添加用户的访问权限

Revoke 权限 on 数据库名 from 用户名@“客户端”

实验一：

服务端（192.168.4.50）

mysql> select user,host from mysql.user; #查看系统用户

mysql> show grants for [root@"192.168.4.52";](mailto:root@\"192.168.4.52\";) #查看授权主机的授权信息

mysql> revoke grant option on \*.\* from [root@"192.168.4.52";](mailto:root@\"192.168.4.52\";) #撤销对应的授权权限

客户端（192.168.4.52）

[root@host52 ~]# mysql -h192.168.4.50 -uroot -p123qqq...A

mysql> show grants; #验证权限

实验二：通过直接修改表记录的方式来修改权限

mysql> select \* from mysql.user where user="root" and host="192.168.4.52"\G;

mysql> update mysql.user set delete\_priv="N" where user="root" and host ="192.168.4.52";

mysql> flush privieges;

mysql> select \* from mysql.user where user = "root" and host = "192.168.4.52"\G;

mysql> show grants for root@"192.168.4.52";

注意：撤销权限时需要和授权时给的权限相同

**C.删除授权用户（删除添加的连接用户）**

Drop user 用户名@”客户端地址“

mysql> drop user [root@"192.168.4.52";](mailto:root@\"192.168.4.52\";)

**D.连接用户密码的修改：**

授权用户连接数据库服务器后修改，连接密码

mysql> set password=password(新密码);

管理员重置授权用户的连接密码

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

**数据备份与恢复**

**一、数据备份相关概念**

1.1 数据备份的目的？

数据被误删除 或 设备损害导致数据丢失 ，是备份文件恢复数据。

1.2数据备份方式？

**A.物理备份： 指定备份库和表对应的文件**

51

cp -r /var/lib/mysql /opt/mysql.bak

cp -r /var/lib/mysql/bbsdb /opt/bbsdb.bak

rm -rf /var/lib/mysql/bbsdb

cp -r /opt/bbsdb.bak /var/lib/mysql/bbsdb

chown -R mysql:mysql /var/lib/mysql/bbsdb

systemctl restart mysqld

51 scp /opt/mysql.bak 192.168.4.51:/root/

52 rm -rf /var/lib/mysql

cp -r /root/mysql.bak /var/lib/mysql

chown -R mysql:mysql /var/lib/mysql

systemctl restart mysqld

逻辑备份： 在执行备份命令时，根据备份的库表及数据生成对应的sql命令，把sql存储到指定的文件里。

1.3数据备份策略？

完全备份 备份所有数据（一张表的所有数据 一个库的所有数据 一台数据库的所有数据）

备份新产生数据（差异备份 和 增量备份 都备份新产生的数据 ）

差异备份 备份自完全备份后，所有新产生的数据。

增量备份 备份自上次备份后，所有新产生的数据。

1.4工作中如何对数据做备份？

1.4.1 选择备份策略：

完全备份+差异备份

完全备份+增量备份

1.4.2 数据备份时间 数据服务器访问量少的时候执行备份

1.4.3 数据备份频率 根据数据产生量，决定备份频率

1.4.4 备份文件的命名 库名-日期.sql

1.4.5 备份文件的存储设置 准备独立的存储设备存储备份文件

1.4.6 如何执行备份 使用周期性计划任务执行本机脚本

完全备份+差异备份

06:00 t1 文件名 数据

1 完全 10 1.sql 10

2 差异 3 2.sql 3

3 5 3.sql 8

4 2 4.sql 10

5 7 5.sql 17

6 4 6.sql 21

7 差异 1 7.sql 22

完全备份+增量备份

06:00 t1 文件名 数据

1 完全 10 1.sql 10

2 增量 3 2.sql 3

3 5 3.sql 5

4 2 4.sql 2

5 7 5.sql 7

6 4 6.sql 4

7 增量 1 7.sql 1

**B.完全备份：mysqldum**

]# man mysqldump

]# mysqldump -uroot -p123456 库名 > 目录名 /文件名.sql

库名的表示方式：

--all-databases 或 -A 所有库所有表

库名 备份库下的所有表

库名 表名 备份一张的所有记录

-B 库名1 库名2 备份某几个库的所有数据

**C.完全恢复：mysql**

]#mysql -uroot -p123456 库名 < 目录名 /文件名.sql

mysql> source 目录名 /文件名.sql

]# mkdir /bakdir

]# mysqldump -uroot -p123456 -A > /bakdir/alldb.sql

]# mysqldump -uroot -p123456 db3 > /bakdir/db3.sql

]# mysqldump -uroot -p123456 db3 user > /bakdir/db3-user.sql

]# mysqldump -uroot -p123456 -B db3 db55 > /bakdir/twodb.sql

50：

mysql> drop table db3.user;

mysql> select \* from db3.user;

]# mysql -uroot -p123456 db3 < /bakdir/db3-user.sql

mysql> drop database db3;

mysql> create database db3;

]# mysql -uroot -p123456 db3 < /bakdir/db3.sql

休息到 10:10

db3.user 50

db3-user.sql

mysqldump 备份和恢复会锁表

vim /bakdir/db3-user.sql

lock tables user write

insert into

每周一 23点备份数据库服务器上db3库下的所有表到系统的/bakdir目录。

]# vim /root/alldb3.sh

#!/bin/bash

if [ ! -e /bakdir ];then

mkdir /bakdir

fi

x=`date +%F`

mysqldump -uroot -p123456 db3 > /bakdir/db3\_${x}.sql

:wq

]#chmod +x /root/alldb3.sh

]# /root/alldb3.sh

]# ls /bakdir

]# crontab -e

00 23 \* \* 1 /root/alldb3.sh &> /dev/null

:wq

**D.使用binlog日志做增量备份**

增量备份

Binlog日志的使用：

日志介绍：是服务日志文件中的一种（默认没有启用） 记录除查询之外的sql命令.

select show tables desc show databases 查询操作

insert update delete 写操作

启用日志 vim /etc/my.cnf

[mysqld]

server\_id=50 #指定服务器的编号，1-255之间的整数避免不要和其他服务器冲突

log-bin #启用binlog日志

binlog-format="mixed" #指定日志记录的格式，即记录数据命令，又记录命令执行后产生的数据信息

mysql> show variables like "binlog\_format"; #查看日志当前记录格式

日志文件的三种记录格式：

1. statement：每一条修改的数据的sql命令斗会记录在binlog日志中
2. Row：不记录sql语句上下文相关信息，仅保存哪条记录被修改
3. Mixed：是以上两种格式的混用

#默认存储日志文件的路径为 /var/lib/mysql 命名格式为：主机名-bin.000001

host50-bin.000001 日志文件 #默认情况下文件的容量大于500M之后系统会自动生成第二个日志文件

host50-bin.index 索引文件 #文本文件，记录当前的日志

查看日志内容

]# mysqlbinlog host50-bin.000001

实验：

创建他t10表，并写入一些数据：

mysql> desc t10;

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

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

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

| id | int(11) | YES | | NULL | |

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

1 row in set (0.00 sec)

mysql> insert into t10 values(111);

Query OK, 1 row affected (0.10 sec)

mysql> insert into t10 values(211);

Query OK, 1 row affected (0.04 sec)

mysql> insert into t10 values(311);

Query OK, 1 row affected (0.02 sec)

mysql> insert into t10 values(411);

Query OK, 1 row affected (0.04 sec)

在日志文件中可以查询到新建的数据命令

[root@host50 mysql]# mysqlbinlog /var/lib/mysql/host50-bin.000003 | grep -i insert

insert into t10 values(111)

insert into t10 values(211)

insert into t10 values(311)

insert into t10 values(411)

自定义binlog日志名称及存储目录

[root@host50 ~]# mkdir /logdir

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

修改配置文件

vim /etc/my.cnf

[mysqld]

log-bin=/logdir/db50 #自定义存储路径目录

]# systemctl restart mysqld

]# mysqlbinlog /logdir/db50.000001

**E.日志记录sql命令格式**

偏移量 ：记录sql命令长度的位置

时间点 ：执行sql命令的时间

通过日志文件查看内容

]# mysqlbinlog 选项 /logdir/db50.000001

选项：

通过偏移量查看内容

--start-position=数字 #起始

--stop-position=数字 #结束

通过时间点查看内容

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

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

**F.执行日志文件记录的sql命令恢复数据**

]# mysqlbinlog 选项 日志文件名 | mysql -uroot -p123456

偏移量恢复：

]# mysqlbinlog --start-position=616 --stop-position=794 /logdir/db50.000001 | mysql -uroot -p123456

时间点恢复：

]# mysqlbinlog --start-datetime="2018-07-17 14:21:15"

--stop-datetime="2018-07-17 14:21:18" /logdir/db50.000001 | mysql -uroot -p123456

Max\_binlog\_size=数字M #可以指定日志文件大小 ，在配置文件中加

mysql> show master status; #查看正在使用的日志信息

手动生成新的日志文件

第一种：]# mysqldump -uroot -p123456 --flush-logs db3 > /bakdir/db3.sql

#备份完当前数据库文件后生成新的日志文件

第二种：mysql> flush logs; #登陆状态下生成日志文件

第三种：]# mysql -uroot -p123456 -e "flush logs" #生成新的日志文件

第四种：]# systemctl restart mysqld #重启服务生成新的数据库日志文件

删除已有的日志文件

mysql> purge master logs to "db50.000005";

#把编号5之前的日志文件全部删除

mysql> reset master; #删除所有日志文件，重新生成第一个

]#rm -rf /logdir/\* #清除所有

**二、安装第3方软件percona提供的命令innobackupex**

**A.安装软件**

312 yum -y install perl-DBD-mysql perl-Digest-MD5

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

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

Xtrabackup ：以perl脚本封装xtrabackup,还支持mylsam

Innobackupex : c程序，支持innodb和xtadb

命令格式：innobackupex <选项>

即可增量备份与恢复 也可完全备份和恢复

增量备份时 存储引擎必须要时innodb

选项：

**B.完全备份**

[root@host50 ~]# innobackupex --user root --password 123 /b

#完全备份数据库到/b下

]# innobackupex --user root --password 123456 --databases="mysql performance\_schema sys gamedb " /allbak --no-timestamp

#备份数据库中mysql , performance\_schema ,sys,gamedb 几个库，不以时间戳为文件名

]# ls /allbak （备份目录下既有备份的数据又有日志文件）

**C.完全恢复**

完全恢复要求数据库目录必须时空的

--apply-log #准备恢复数据

--copy-back #拷贝

1 、准备恢复数据 （回滚备份目录下日志信息）

]#innobackupex --user root --password 123456 \

--databases="mysql performance\_schema sys gamedb " \

--apply-log /allbak

]# systemctl stop mysqld

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

2、把备份目录下数据库拷贝回数据库目录下

]#innobackupex --user root --password 123456 \

--databases="mysql performance\_schema sys gamedb " \

--copy-back /allbak

3、修改数据库目录所有者和组用户为mysql

]# chown -R mysql:mysql /var/lib/mysql

启动数据库服务

]# systemctl start mysqld

4、登录查看数据。

]# mysql -uroot -p123456

mysql> show databases;

mysql> select \* from gamedb.a;

**D.增量备份**

备份上次备份后，所有新产生的数据。

innobackupex 增量备份

第1次备份 所有数据

]# innobackupex --user root --password 123456 /fullbak --no-timestamp

完全备份后向表里存储新数据

insert into gamedb.a values(909090),(909090);

insert into gamedb.b values(808080),(808080);

select \* from gamedb.a; select \* from gamedb.b;

对数据做增量备份

]# innobackupex --user root --password 123456 --incremental /new1dir --incremental-basedir=/fullbak --no-timestamp

]# ls /new1dir

增量备份后继续向表中存储新数据

insert into gamedb.a values(6609090),(669090);

insert into gamedb.b values(778080),(778080);

select \* from gamedb.a; select \* from gamedb.b;

对数据做增量备份

]# innobackupex --user root --password 123456 --incremental /new2dir --incremental-basedir=/new1dir --no-timestamp

]# ls /new2dir

lsn 日志序列号

**E.增量恢复**

1.准备恢复数据

]# innobackupex --user root --password 123456 --apply-log --redo-only /fullbak

]#innobackupex --user root --password 123456 --apply-log --redo-only /fullbak --incremental-dir=/new1dir

]#innobackupex --user root --password 123456 --apply-log --redo-only /fullbak --incremental-dir=/new2dir

]# rm -rf /new1dir/

]# rm -rf /new2dir/

2.把备份目录下的数据拷贝到数据库目录并修改所有者和组为mysql

systemctl stop mysqld

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

]# innobackupex --user root --password 123456 --copy-back /fullbak

3.启动数据库服务

]# chown -R mysql:mysql /var/lib/mysql

]# systemctl start mysqld

4.登录查看数据 select \* from gamedb.a;

5.恢复后继续存储数据，执行增量备份

]# innobackupex --user root --password 123456 --incremental /dir1 --incremental-basedir=/fullbak --no-timestamp

**F.生产中使用脚本实现**

vim /root/allbak.sh

#!/bin/bash

day=`date +%F`

innobackupex --user root --password 123456 /fullbak\_${day} --no-timestamp

:wq

vim /root/newallbak.sh

#!/bin/bash

jt=`date +%d`

zt=` expr $jt - 1 `

m=`date +%m`

innobackupex --user root --password 123456 --incremental /new${m}\_${jt}dir --incremental-basedir=/new${m}\_${zt}dir --no-timestamp

:wq

chmod +x /root/allbak.sh

chmod +x /root/newallbak.sh

00 23 \* \* 1 /root/allbak.sh

00 23 \* \* 2-7 /root/newallbak.sh

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

使用innobackupex完全备份文件恢复单个表

]#innobackupex --user root --password 123456 \

--databases="gamedb" /gamedbbak --no-timestamp

mysql> drop table gamedb.a;

mysql> system ls /var/lib/mysql/gamedb/a.\*

mysql> system ls /var/lib/mysql/gamedb/b.\*

恢复单个表

create table gamedb.a(id int);

mysql> system ls /var/lib/mysql/gamedb/a.\*

mysql> alter table gamedb.a discard tablespace; 删除表空间 a.ibd

mysql> system ls /var/lib/mysql/gamedb/a.\*

]#innobackupex --user root --password 123456 \

--apply-log --export /gamedbbak 导出表信息

]# cp /gamedbbak/gamedb/a.{cfg,exp,ibd} /var/lib/mysql/gamedb/

]# chown mysql:mysql /var/lib/mysql/gamedb/a.\*

mysql> alter table gamedb.a import tablespace; 导入表空间

]# rm -rf /var/lib/mysql/gamedb/a.{cfg,exp} 删除表信息文件

MySQL> select \* from gamedb.a;

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

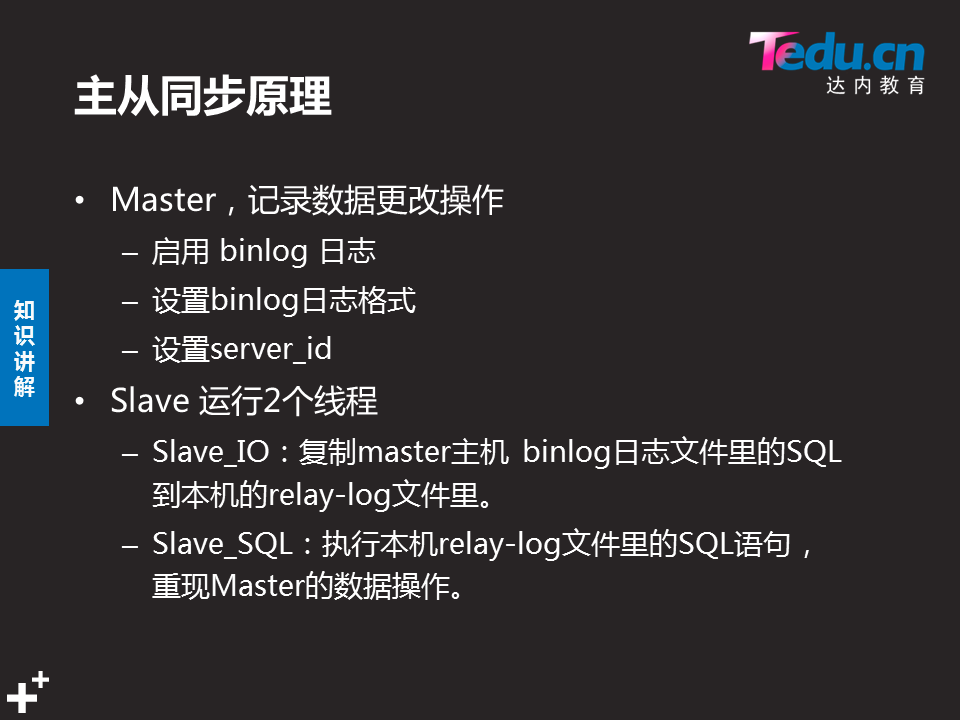
**Mysql主从同步**

配置mysql主从同步：

什么是mysql主从同步？

从数据库向主数据库同步数据的方式就是主从同步

**一、主从同步工作原理：**



怎么配置？

A.主库（master）配置如下：（192.168.4.51上操作）

1. 启用binlog日志文件

[root@host51 ~]# vim /etc/my.cnf

[mysqld]

server\_id=51

log-bin=master51

binlog\_format="mixed"

[root@host51 ~]# systemctl restart mysqld

[root@host51 ~]# systemctl enable mysqld

[root@host51 ~]# ls /var/lib/mysql/master51.\*

1. 用户授权：给从库添加一个连接时使用的用户名

mysql> grant replication slave on \*.\* to repluser@"192.168.4.52" identified by "123qqq...A";

1. 查看正在使用的binlog日志信息

mysql> show master status; #可以查看日志文件信息及偏移量位置

B.从库（slave）配置如下：（192.168.4.52上操作）

mysql> show slave status; #查看当前数据库是否为从库 不是则为空

1. 指定server\_id

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

[mysqld]

server\_id=52

[root@host52 ~]# systemctl restart mysqld

1. 测试授权用户

[root@host52 ~]# mysql -h192.168.4.51 -urepluser -p123qqq...A

mysql> show grants; #查看当前用户的权限

1. 管理员本机登录指定主库信息

mysql> change master to

>master\_host="192.168.4.51", #主库

>master\_user="repluser", #从库用户

>master\_password="123qqq...A", #从库用户密码

>master\_log\_file="master51.000001", #指定日志文件信息

>master\_log\_pos=452; #指定偏移量

mysql> show slave status\G;

.......

Slave\_IO\_Running: No

Slave\_SQL\_Running: No

..........

mysql> start slave; #启动配置

1. 查看配置

mysql> show slave status\G;

................

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

................

mysql> show processlist; #显示当前主机运行的程序列表

C.客户端测试主从同步配置

在master主库服务器上管理员本机登录，添加客户端存储数据时，使用的连接用户（192.168.4.51）

mysql> create database gamedb;

mysql> create table gamedb.t1(id int);

mysql> grant select,insert on gamedb.\* to webadmin@"%" identified by "123qqq...A";

客户端连接主库存储数据（192.168.4.50）

[root@host50 ~]# mysql -h 192.168.4.51 -uwebadmin -p123qqq...A

mysql> show grants;

mysql> insert into gamedb.t1 values(666);

mysql> insert into gamedb.t1 values(777);

mysql> insert into gamedb.t1 values(888);

mysql> insert into gamedb.t1 values(999);

mysql> insert into gamedb.t1 values(111);

在从库管理员root本机登录，也能看到客户端连接主库存储的数据（192.168.4.52）

[root@host52 ~]# mysql -uroot -p123456

mysql> select \* from gamedb.t1;

+------+

| id |

+------+

| 666 |

| 777 |

| 888 |

| 999 |

| 111 |

+------+

5 rows in set (0.00 sec)

客户端连接主库存储数据，客户端也能从从库看到主库一样的数据

C.如何让从库暂时不同步主库的数据

mysql> stop slave;

D.如何把从库还原成独立的数据库服务

[root@host52 mysql]# rm -rf master.info felay-log.info

[root@host52 mysql]# rm -rf host52-relay-bin.index

**二、Mysql主从同步模式：**

主从同步结构模式：

1. **一主一从：**

**B.一主多从：**

主库（192.168.4.51）：

mysql> drop user [repluser@"192.168.4.52";](mailto:repluser@\"192.168.4.52\";) #删除原有授权用户

mysql> drop user [lisi@"192.168.4.53";](mailto:lisi@\"192.168.4.53\";)

mysql> grant replication slave on \*.\* to repluser@"%" identified by "123qqq...A"; #授权从库用户

[root@host51 ~]# mysqldump -uroot -p123456 gamedb > /root/gamedb.sql #备份原仓库文件

[root@host51 ~]# scp /root/gamedb.sql [root@192.168.4.53:/root/](mailto:root@192.168.4.53:/root/) #把备份文件发给从库

mysql> grant select,insert on bbsdb.\* to testuser@"%" identified by "123qqq...A";

#为客户端创建验证用户

从库（192.168.4.53）：

mysql> create database gamedb;

[root@host53 ~]# mysql -uroot -p123456 gamedb < /root/gamedb.sql

测试授权帐号：

[root@host52 ~]# mysql -h192.168.4.51 -urepluser -p123qqq...A

[root@host53 ~]# mysql -h192.168.4.51 -urepluser -p123qqq...A

指定主机id号：

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

[mysqld]

server\_id=53

[root@host53 ~]# systemctl restart mysqld

在53上指定主库信息：

mysql> change master to

-> master\_host="192.168.4.51",

-> master\_user="repluser",

-> master\_password="123qqq...A",

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

-> master\_log\_pos=5095;

mysql> start slave; #启动线程服务

同上配置192.168.4.52

**C.主从从结构**

**1.还原host53主机，成为独立的存在**

[root@host53 ~]# systemctl stop mysqld

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

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

[root@host53 mysql]# systemctl restart mysqld

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

mysql> drop database gamedb;

同上还原52主机，成为独立的存在

**2.以192.168.4.51为主库**

mysql> system vim /etc/my.cnf

[mysqld]

server\_id=51

log-bin=db51

binlog\_format="mixed"

mysql> grant replication slave on \*.\* to yaya@"%" identified by "123qqq...A";

#设置从服务器的权限用户

**3.把192.168.4.52为主库和从库**

配置为192.168.4.53的主库的配置：

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

[mysqld]

server\_id=52

log-bin=master52

binlog\_format="mixed"

log\_slave\_updates #开启极连复制 如果不加此项则当客户端在主库上添加数据时53号主机上无法同步

mysql> grant replication slave on \*.\* to reluser@"%" identified by "123qqq...A";

mysql> show master status;

配置为192.168.4.51的从库的配置：

[root@host52 mysql]# mysql -h192.168.4.51 -uyaya -p123qqq...A #测试帐号

[root@host52 mysql]# mysql -uroot -p123456

mysql> change master to

-> master\_host="192.168.4.51",

-> master\_user="yaya",

-> master\_password="123qqq...A",

-> master\_log\_file="db51.000001",

-> master\_log\_pos=154;

mysql> start slave;

mysql> show slave status\G;

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

**4.以192.168.4.53配置为192.168.4.52的从库**

[root@host53 mysql]# systemctl stop mysqld

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

[mysqld]

server\_id=53

[root@host53 mysql]# systemctl start mysqld

[root@host53 mysql]# mysql -h192.168.4.52 -ureluser -p123qqq...A #测试授权帐号

[root@host53 mysql]# mysql -uroot -p123456 #用管理员帐号登录

mysql> show slave status\G; #查看本库是否为从库

mysql> change master to #配置主库结构

-> master\_host="192.168.4.52",

-> master\_user="reluser",

-> master\_password="123qqq...A",

-> master\_log\_file="master52.000001",

-> master\_log\_pos=440;

mysql> start slave; #启动配置

mysql> show slave status; #查看线程是否开启

..............

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

...........

**5.验证：**

在主库上建库建表（192.168.4.51）

mysql> create database bbsdb;

mysql> create table bbsdb.a(id int);

客户端上添加数据（192.168.4.50）：

[root@host50 ~]# mysql -h192.168.4.51 -utestuser -p123qqq...A

mysql> show grants;

mysql> insert into a values(123456);

mysql> insert into a values(654321);

mysql> insert into a values(789456123);

mysql> insert into a values(111111111);

从库上验证（192.168.4.52）（192.168.4.53）

mysql> select \* from bbsdb.a;

Mysql主从同步常用配置选项

1.log\_slave\_updates #开启极连复制 当一个数据库机即做主库又做从库时必须加上此选项

2.Binlog\_do\_db=库名列表 #只允许同步的库

3.Binlog\_ignore\_db=库名列表 #只不允许同步的库

#2、3两种情况不能同时使用，以上三种情况斗都是在主库上配置

1. replicate\_do\_db=库名列表 #只同步的库 用在从库上
2. replicate\_ignore\_db=库名列表 #只不同步的库 用在从库上

主从同步复制模式

启用半同步复制模式：

（192.168.4.54上操作）

Master：

mysql> show variables like 'have\_dynamic\_loading'; #查看是否允许加载模块

查看加载的模块：

mysql> select plugin\_name,plugin\_status

->from information\_schema.plugins

->where plugin\_name like '%semi%';

mysql> install plugin rpl\_semi\_sync\_master #为主库安装模块

-> soname 'semisync\_master.so'; #模块名称

mysql> set global rpl\_semi\_sync\_master\_enabled=1; #启动加载模块

mysql> show variables like 'rpl\_semi\_sync\_%\_enabled'; #查看模块开启情况

Slave：

mysql> install plugin rpl\_semi\_sync\_slave #为从库安装模块

->soname 'semisync\_slave.so'; #模块名称

查看加载的模块：

mysql> select plugin\_name,plugin\_status

->from information\_schema.plugins

->where plugin\_name like '%semi%';

mysql> set global rpl\_semi\_sync\_slave\_enabled=1; #启用从库半同步模块

mysql> show variables like 'rpl\_semi\_sync\_%\_enabled'; #查看模块加载情况

（192.168.4.55上操作）

修改配置文件在配置文件中添加模块

[root@host55 ~]# vim /etc/my.cnf

..........

[mysqld]

plugin-load=”rplsemi\_sync\_master=semisync\_master.so rpl\_semi\_sync\_master\_enabled=1”

plugin-load=”rplsemi\_sync\_master=semisync\_slave.so rpl\_semi\_sync\_slave\_enabled=1”

.....

[root@host55 ~]# systemctl restart mysqld

#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#$#

Mysql数据读写分离

把客户端查询数据库的请求和写入数据的请求，分别给不同数据库服务器处理。

目的：减轻主库的并发访问压力，提高从库的硬件利用率。

人肉分离：需要程序员协同完成，规定程序员在存储数据时连接主数据库服务器；查询数据时连接slave数据库服务器。

部署实现数据读写分离服务实现：mysql中间件，实现软件有（mycat、mysql-proxy、maxscale）

配置数据读写分离服务器：

1. 部署mysql主从同步结构（一主一从）

192.168.4.51为主库

192.168.4.52为从库

192.168.4.56 为代理

192.168.4.50为访问客户端

1. 测试主从同步配置

2.1创建给客户端授存储数据时，连接服务启的用户名密码、库表

mysql> create database testdb;

mysql> create table testdb.t1(id int);

mysql> grant select,insert on testdb.\* to zhangsan@"192.168.4.50" identified by "123qqq...A";

2.2在从库上面查看主库上的配置

.......

2.3在客户端连接主库存储数据 测试

......

1. 配置读写分离服务器192.168.4.56

装包

[root@host56 ~]# yum -y install /maxscale-2.1.2-1.rhel.7.x86\_64.rpm

[root@host56 ~]# rpm -qa | grep -i maxscale #查看文件安装是否成功

maxscale-2.1.2-1.x86\_64

[root@host56 ~]# rpm -qc maxscale #查看文件安装路径

/etc/maxscale.cnf.template

修改配置文件

[root@host56 ~]# cp /etc/maxscale.cnf /etc/maxscale.cnf.bak #备份源配置文件

[root@host56 ~]# vim /etc/maxscale.cnf

......

9 [maxscale]

10 threads=auto #线程数改为自动

..........

18 [server1] #指定数据库服务器名称

19 type=server #指定数据库服务器类型

20 address=192.168.4.51 #指定ip地址

21 port=3306 #指定端口号

22 protocol=MySQLBackend #指定协议类型

23 [server2]

24 type=server

25 address=192.168.4.52

26 port=3306

27 protocol=MySQLBackend

...........

35 [MySQL Monitor] #监视服务器

36 type=monitor #类型为监视

37 module=mysqlmon #模块

38 servers=server1, server2 #监视的服务器

39 user=scalemon #监视服务器授权用户

40 passwd=123qqq...A #监视服务器授权用户密码

41 monitor\_interval=10000 #监视间隔为10000毫秒

.................

52 #[Read-Only Service] #只读服务器配置

53 #type=service

54 #router=readconnroute

55 #servers=server1

56 #user=myuser

57 #passwd=mypwd

58 #router\_options=slave

..........

63 [Read-Write Service] #定义读写分离服务

64 type=service

65 router=readwritesplit

66 servers=server1, server2

67 user=maxscale

68 passwd=123qqq...A

69 max\_slave\_connections=100%

..................

85 #[Read-Only Listener] #指定读写分离的监听端口

86 #type=listener

87 #service=Read-Only Service

88 #protocol=MySQLClient

89 #port=4008

90

91 [Read-Write Listener]

92 type=listener

93 service=Read-Write Service

94 protocol=MySQLClient

95 port=4006

96

97 [MaxAdmin Listener] #定义管理服务

98 type=listener

99 service=MaxAdmin Service

100 protocol=maxscaled

101 socket=default

102 prot=4016

在数据库服务器上 根据配置文件的设置添加对应的授权用户（192.168.4.51上操作）

mysql> grant replication slave,replication client on \*.\* to scalemon@'%' identified by '123qqq...A';

mysql> grant select on mysql.\* to maxscale@"%" identified by "123qqq...A";

[root@host56 ~]# which mysql #查看系统时否按张mariadb服务 没有则安装

测试配置帐号：

[root@host56 ~]# mysql -h 192.168.4.51 -uscalemon -p123qqq...A

[root@host56 ~]# mysql -h 192.168.4.52 -uscalemon -p123qqq...A

[root@host56 ~]# mysql -h 192.168.4.51 -umaxscale -p123qqq...A

[root@host56 ~]# mysql -h 192.168.4.52 -umaxscale -p123qqq...A

启动服务

查看服务信息

4.测试读写分离

Mysql多实例

在一台物理主机上运行多个mysql数据库服务

装包

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

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

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

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

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

[root@host50 ~]# echo $PATH

创建主配置文件

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

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

[mysqld\_multi]

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

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

user = root

[mysqld1]

port = 3307

datadir = /data3307

socket = /data3307/mysql.sock

pid-file = /data3307/mysqld.pid

log-error = /data3307/mysqld.err

[mysqld2]

port = 3308

datadir = /data3308

socket = /data3308/mysql.sock

pid-file = /data3308/mysqld.pid

log-error = /data3308/mysqld.err

[root@host50 ~]# mkdir /data3307

[root@host50 ~]# mkdir /data3308

初始化授权库

[root@host50 ~]# mysqld --user=mysql --basedir=/usr/local/mysql --datadir=/data3307 --initialize

.................

2018-07-21T10:05:45.616349Z 1 [Note] A temporary password is generated for root@localhost: pwJ,f14su4\*M

[root@host50 ~]# mysqld --user=mysql --basedir=/usr/local/mysql --datadir=/data3308 --initialize

......

2018-07-21T10:07:16.009765Z 1 [Note] A temporary password is generated for root@localhost: 2xtu(iy8FRvg

............

启动服务

格式：mysql\_multi start 编号

[root@host50 ~]# mysqld\_multi start 1 #启动编号为1的服务

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

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

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

测试配置

格式：Mysql -uroot -p初始密码 -S sock文件

[root@host50 ~]# mysql -uroot -p"pwJ,f14su4\*M" -S /data3307/mysql.sock

mysql> alter user root@"localhost" identified by "123456";

mysql> show databases;

[root@host50 ~]# mysql -uroot -p"2xtu(iy8FRvg" -S /data3308/mysql.sock

mysql> alter user root@"localhost" identified by "123456";

mysql> show databases;

关闭服务

[root@host50 ~]# mysqld\_multi --user=root --password="123456" stop 1

数据分片

如何解决并发访问压力？

如何解决单表过大的问题？

准备环境

配置mycat服务器51

1.装包

[root@host51 ~]# tar -xf /Mycat-server-1.4-beta-20150604171601-linux.tar.gz

[root@host51 ~]# mv mycat/ /usr/local/

[root@host51 ~]# ls /usr/local/mycat/

bin catlet conf lib logs version.txt

1. 工作目录说明

Conf：配置文件 记录的分片规则和方法 不用修改

-----Rule.xml

-----Schema.xml #定义表名称 表字段 分片规则

-----Server.xml #定义客户端连接mycat时的用户名、密码及默认虚拟库名称

Bin :管理命令

3.修改配置文件 启动分片

[root@host51 ~]# cd /usr/local/mycat/conf/

[root@host51 conf]# cp server.xml server.xml.bak

[root@host51 conf]# cp schema.xml schema.xml.bak

#备份以下源配置文件防止改错后恢复

[root@host51 conf]# vim server.xml

............

33 </system>

34 <user name="admin"> #定义连接用户的用户名

35 <property name="password">123456</property> #定义用户名密码

36 <property name="schemas">TESTDB</property> #定义虚拟库的名称

37 </user>

38

39 <user name="webuser">

40 <property name="password">654321</property>

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

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

43 </user>

[root@host51 conf]# vim schema.xml

<?xml version="1.0"?>

<!DOCTYPE mycat:schema SYSTEM "schema.dtd">

<mycat:schema xmlns:mycat="http://org.opencloudb/">

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

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

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

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

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

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

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

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

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

rule="mod-long" />

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

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

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

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

rule="sharding-by-intfile" />

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

rule="sharding-by-intfile">

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

parentKey="id">

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

parentKey="id" />

</childTable>

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

parentKey="id" />

</table>

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

/> -->

</schema>

<!-- <dataNode name="dn1$0-743" dataHost="localhost1" database="db$0-743"

/> -->

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

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

<!-- <dataNode name="dn3" dataHost="localhost1" database="db3" /> -->

<!-- <dataNode name="jdbc\_dn1" dataHost="jdbchost" database="db1" /> <dataNode

name="jdbc\_dn2" dataHost="jdbchost" database="db2" /> <dataNode name="jdbc\_dn3"

dataHost="jdbchost" database="db3" /> -->

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

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

<heartbeat>select user()</heartbeat>

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

<writeHost host="hostM1" url="192.168.4.52:3306" user="yaya"

password="123qqq...A">

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

</writeHost>

<!-- <writeHost host="hostS1" url="localhost:3316" user="root"

password="123456" /> -->

<!-- <writeHost host="hostM2" url="localhost:3316" user="root" password="123456"/> -->

</dataHost>

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

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

<heartbeat>select user()</heartbeat>

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

<writeHost host="hostM2" url="192.168.4.53:3306" user="yaya"

password="123qqq...A">

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

</writeHost>

<!-- <writeHost host="hostS1" url="localhost:3316" user="root"

password="123456" /> -->

<!-- <writeHost host="hostM2" url="localhost:3316" user="root" password="123456"/> -->

</dataHost>

根据配置文件做相应设置：

在192.168.4.52上：

mysql> create database db1;

Query OK, 1 row affected (0.03 sec)

mysql> grant all on \*.\* to yaya@"%" identified by "123qqq...A";

Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> quit

Bye

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

[mysqld]

lower\_case\_table\_names = 1 #忽略表名大小写

[root@host52 ~]# systemctl restart mysqld

在192.168.4.53上：

mysql> create database db2;

Query OK, 1 row affected (0.00 sec)

mysql> grant all on \*.\* to yaya@"%" identified by "123qqq...A";

Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> quit

Bye

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

[mysqld]

lower\_case\_table\_names = 1 #忽略表名大小写

[root@host53 ~]# systemctl restart mysqld

启动服务

[root@host51 conf]# /usr/local/mycat/bin/mycat start

Starting Mycat-server...

[root@host51 conf]# ss -lnutp | grep 8066

tcp LISTEN 0 100 :::8066 :::\* users:(("java",pid=6818,fd=49))

测试配置