# SQL语言：

**SQL: 结构化查询语言**

## 安装MySql客户端：

### ubuntu:

sudo apt-get install mysql-server 安装数据库 服务端

sudo apt-get install mysql-client 安装数据库 客户端

sudo service mysql stop 停止数据库服务端

sudo service mysql start 启动数据库服务端

sudo service mysql restart 重启数据库服务端

sudo service mysql status 查看状态

### windows:

net start mysql 启动数据库

net stop mysql 停止数据库 （无重启命令）

## 快捷操作：

MySQL 中：

Ctrl + A 快速回到行首

Ctrl + E 快速回到行末

## 数据类型：

(a) 整数： int , bit

(b) 小数： decimal

(c) 字符串： varchar, char

(d) 日期时间： date, time, datetime

(e) 枚举类型（enum）: 列出智能选择的选项

(f) 特殊说明：

<1> decimal : 表示浮点数，如decimal（5,2）表示共有5位数,小数占2位

<2> char ：表示固定长度的字符串，如：char（3）填充“ab”时，会补一个空格“ab ”

<3> varchar : 表示可变长度的字符串，如varchar(3)，填充'ab'时就会存储'ab'

<4> 字符串text表示存储大文本，当字符大于4000时推荐使用

<5> 对于图片、音频、视频等文件，不存储在数据库中，而是上传到某个服务器上，后在表中存储这个文件的保存路径

## 约束：

（a）主键 primary key : 物理上存储的顺序（唯一值，不能重复）

（b）非空 not null : 此字段不允许填写空值

（c）唯一 unique : 此字段的值不允许重复

（d）默认 default : 当不填写此值时会使用默认值，如果填写以填写为准

（e）外键 foreign ： **对关系字段进行约束，当为关系字段填写值时，会到关表中**

**查询此值是否存在，如果存在则填写成功，如果不存在则填写失败并抛出异常**

**说明：虽然外键约束可以保证数据的有效性，但是在进行数据的crud（增加、修改、删除、查询）时，都会降低数据库的性能，所以不推荐使用，那么数据的有效性怎么保证呢？答：可以在逻辑层进行控制**

## MySQL:

登录命令

mysql –h主机地址 –u用户名 –p密码 远程登陆

mysql -u用户名 -p密码

退出命令

exit quit ctrl+d

# 数据《库》操作：

## 查看：

（1）show databases; 显示所有数据库：

（2）**select** datebase（） ; 查看当前所在的数据库

（3）show create database 库名; **查看创建数据库的语句**

（4）select version（）； **查看数据库版本**

（5）select now（）; **查看当前时间**

## 创建：

* **如添加数据库 名字与已有名字相同 会创建失败 报错**
* **数据库 不指定charset 默认 拉丁文**

create database `数据库名` charset=utf8 / charset utf8 ;

## 使用：

use `数据库名` ；

## 删除：

drop database `数据库名` ；

# 数据《表》操作：

## 创建：

create table  **[ if not exists ]**  表名（ 字段 类型 约束 ）； # 如果存在不创建

-- auto\_increment表示自动增长

-- not null 表示不能为空

-- primary key 表示主键

-- default 默认值

-- create table 数据表名字 (字段 类型 约束[, 字段 类型 约束]);

例：-- 创建classes表(id、name)

create table classes (

id int unsigned not null primary key auto\_increment ,

name varchar(30)

);

## 修改表：

**（1）添加字段：** alter table **表名** add 列名 类型 ；

例1 : 在某个字段后面添加：

**alter table** students add birthday datetime [ after 字段名 ] ；

例2 : 在第一个字段添加：

alter table students add birthday datetime [ first 字段名 ] ；

**（2） 修改字段 类型：** alter table **表名** modify **列名** **类型 及 约束**

alter table students modify birthday **time** ;

**（3） 修改字段 名： alter table 表名 change 字段原名 字段新名 类型 及 约束**

例：alter table **students** change birthday birth data default “2011-01-01” ;

（4） 删除字段： alter table **表名** drop **列名**

alter table **students** drop high ;

## 删除表：

drop table 表名 ；

## 重命名表：

rename table 旧表名 to 新表名 ;

## 查看表：

（1）show tables ; 查看所有数据表

（2）desc 表名； 查看表结构

（3）show create table 表名 ; 查看表的创建语句

# 《数据》--操作：

## 添加：

* **添加数据 全部插入时 可以用default意思是取这个字段的默认值**

（1）增加一条数据： insert into 表名 values（全部列的值） ；

（2）部分字段添加数据： insert into 表名 （字段1，字段2） values（值1，值2）；

（3）插入多条： insert into 表名 values（全部列的值），（全部列的值），（全部列的值）

例：insert into students values(default, "西施", 20, "女", 1, "1990-01-01"),

(default, "王昭君", 20, "女", 1, "1990-01-01");

（4）部分字段 插入 多条：

insert into 表名（字段1，字段2） values（值1，值2），（值1，值2），（值1，值2）

例: insert into students (name, gender) values ("大乔", 2),("貂蝉", 2);

（5）如果字段类型 是枚举enum， 可以用数字代替其中的枚举数据

## 修改数据：

**update 表名 set 字段1=值，字段2=值， where 条件 ；**

例1： update students set gender=1; -- 全部都改

update students set gender=1 where name="小李飞刀";

-- 只要name是小李飞刀的 全部的修改

update students set gender=1 where id=3 ; -- 只要id为3的 进行修改

update students set age=22, gender=1 where id=3 ; --只要id为3的 进行修改

## 删除数据：

（1）物理删除： delete from 表名 where 条件 ；

delete from students ; -- 整个数据表中的所有数据全部删除

delete from students where name="小李飞刀" ;

**再次插入数据时，主键如果设的是自动增长，则不会从1开始，否则可以选择清空**

（2）逻辑删除

-- 用一个字段来表示 这条信息是否已经不能再使用了

-- 给students表添加一个is\_delete字段 bit 类型

alter table students add is\_delete bit default 0;

逻辑删除的本质是修改了某个字段的值，

update students set is\_delete=1 where id=6 ;

**清空数据**：会把全表都给清空掉，并且自增主键从 1 开始

truncate 表名；

## 查询数据：

（1）**select** \* from 表名 ； 列出所有的数据

（2）**select** \* from 表名 while 条件 ； 条件查询

例： **select** \* from students where name="小李飞刀" ;

-- 查询 name为小李飞刀的所有信息

**select** \* from students where id>3 ; -- 查询 name为小李飞刀的所有信息

（3）指定“列”查询：**select** 列1，列2，。。。 from 表名 ；

例： **select** name,gender from students ;

（4）使用 as 为列 或 表 指定别名：

-- **select** 字段 [as] 别名, 字段 [as] 别名 from 数据表 where ....;

例： **select** name as 姓名,gender as 性别 from students ;

（5）指定字段顺序查询：

例：**select** id as 序号, gender as 性别, name as 姓名 from students ;

## 《查询提高》：

### 基本查询

（1）使用as给表起别名：name as n **等价**与 name n

**select** **别名.字段** from 表名 as 别名;

例：select s.id, s.name from students as s ;

（2）消除重复行：

**select** distinct 字段 from 表名 ；

例： select distinct gender from students ;

### 条件判断： where

* 优先级由高到低的顺序为：小括号，not，比较运算符，逻辑运算符
* and比or先运算，如果同时出现并希望先算or，需要结合()使用

select x, y from 表名 where 条件 ；

（1）比较运算符：

* 等于: =
* 大于: >
* 大于等于: >=
* 小于: <
* 小于等于: <=
* 不等于: != 或 <>

例：select \* from students where name = “黄蓉” ;

（2）逻辑运算符：

* and
* or
* not

例：select \* from students where id > 3 and gender = 0 ;

（3）模糊查询：

* like
* %表示任意多个任意字符
* \_表示一个任意字符
* **rlike 正则**

例1: select \* from students where name like “黄%” ; # 查询姓黄的学生

例2: select \* from students where name like “黄\_” ; #查询姓黄并且“名”是一个字的学生

例3: select \* from students where name **rlike** “^周.\*伦$ #查询已 周开始、伦结尾的姓名”

**（4）范围查询：**

* **in表示在一个非连续的范围内**
* between ... and ...表示在一个连续的范围内

例：查询编号是1或3或8的学生

select \* from students where id in （1,3,8）；

例：查询编号为3至8的学生

select \* from students where id between 3 and 8 ;

例：查询编号是3至8的男生

select \* from students where （id between 3 and 8） and gender = 1；

（5）空判断：

* 注意：null与 ' ' 是不同的
* 判空is null
* 判非空is not nul

例：查询没有填写身高的学生

select \* from students where height is null ;

例：查询填写了身高的学生

select \* from students where height is not null ;

例：查询填写了身高的男生

select \* from students where height is not null and gender = 1;

### 排序: order by

**语法**：

select \* from 表名 order by 字段1 desc[asc] , 字段2 desc [asc] ….. ;

**说明:**

* 将行数据按照列1进行排序，如果某些行列1的值相同时，则按照列2排序，以此类推
* 默认按照列值从小到大排列（asc）
* asc从小到大排列，即升序
* desc从大到小排序，即降序

例：查询未删除男生信息，按学号降序

select \* from students where is\_delete = 0 and gender=1 order by id desc ;

例：查询未删除学生信息，按名称升序

select \* from students where is\_delete=0 order by name ;

例3：**显示所有的学生信息，先按照年龄从大到小排序，当年龄相同时 按照身高从高-->矮排序**

select \* from students order by age desc, height desc ;

### 聚合函数：

* 聚合函数不能在 where 中使用

（1）**总数：**

count(\*) ： 表示计算总行数，括号中写星与列名，结果是相同的

（2）**最大值：**

max(列) ： 表示求此列的最大值

（3）**最小值：**

min(列)：表示求此列的最小值

（4）**求和：**

**sum**（列） ： 表示求此列的和

（5）**平均值：**

**avg** （列） ： 表示求此列的平均值

（6）**四舍五入**：

round(数值 , 2) ：将数值 保留两位小数 例：round（ avg( x ) , 2）

例：查询学生总数

select count（\*） from students ；

例：查询女生的编号最大值

select max（id） from students where gender = 2 ;

例：查询未删除的学生最小编号

select min（id） from students where is\_delete = 0 ;

例：查询男生的总年龄

select sum（age） from students where gender = 1 ;

select sum（age）/count（\*） from students where gender = 1 ； **求男生平均年龄**

例：查询未删除女生的编号平均值

select avg（id） from students where is\_delete = 0 and gender = 2 ;

### 分组: group by

group by的含义 : **将查询结果按照1个或多个字段进行分组，字段值相同的为一组**

group by : **可用于单个字段分组，也可用于多个字段分组,** 单独使用时的实际意义不大

（1） group by + group\_concat(**字段1, 字段2**) : 可以作为输出字段来使用

select gender, group\_concat（name） from students group by gender ;

（2） group by + **聚合函数**：

select gender, count（\*） from students group by gender **#查询男女生个数**

（3） group by + having :

* having 条件表达式：用来分组查询后指定一些条件来输出查询结果
* having 作用和where一样，但having只能用于group by
* having 对结果进行条件判断 where ：对 表 的字段进行条件判断

select gender, count（\*） from students group by gender having count（\*） > 2;

（4）count（\*） + group by + with rollup :

**with rollup的作用是：在最后新增一行，来记录当前列里所有记录的总和**

select gender, count（\*） from students group by gender with rollup ;

### 分页：limit

语法：

select \* from 表名 limit n ; # 显示前 n 个

select \* from 表名 limit start , count ; # 从 start 开始，获取count条数据

使用：每页显示m个, 求第 n 页 的数据：

select \* from 表名 limit （n-1）\*m ，m ;

### 连接查询：inner[left,right]+join

* 内连接： inner join

查询的结果为两个表匹配到的数据

* 左连接： left join

**匹配两个表相同数据，左表全部输出，对于右表不存在的，使用null填充**

* 右连接： right join

**匹配两个表相同的数据，右表全部输出，对于左表不存在的，使用null填充**

**语法：**

**select \* from 表1 inner[left,right] join 表2 on 表1.列 = 表2.列 ；**

**多个链接：**

**select 字段 from 表1 inner join 表2 on 条件 inner join 表3 on 条件 ;**

**说明： on 后面的表示判断**

例：使用内连接查询班级表与学生表

**select \* from students as s inner join classes as c on s.cls\_id = c.id ;**

例：查询学生姓名及班级名称

**select s.name, c.name from students as s inner join classes as c on s.cls\_id = c.id ;**

### 自关联: 建立在连接inner之上

说明：在一个表内，设分级字段（主键id, 副id），等级高的主键id 是等级低的 副id

**最高级别的副id 用null 来填充**

**例：**

**select p.\*, c.name from areas as p inner join areas as c on p.pid=c.aid having p.name=河南省”；**

### 子查询

定义：在一个select 语句中，嵌入了另外一个select语句，那么被嵌入的这个select语句

被称之为子查询语句

**子查询分类：**

* **标量子查询: 子查询返回的结果是一个数据(一行一列)**
* **列子查询: 返回的结果是一列(一列多行)**
* **行子查询: 返回的结果是一行(一行多列)**

**（1）标量子查询**

**例：**查询班级学生的平均年龄，输出大于平均年龄的学生

**select \* from students where age >** （select avg（age） from students ）;

**（2）列级子查询：**

**例：查询班级表中所有的 班级id对应 班级表中对应的名字**

**select \* from students as s where s.cls\_id in （select id from classes）;**

**（3）行级子查询：**

**例：查找班级年龄最大，身高最高 的学生**

**解1：select \* from students where age=（select max(age) from students）**

**or height= （select max(height) from students）;**

**解2：select \* from students where (age,height) = （select max(age), max(height) from**

**students）；**

### 总结：

完整的格式：

select x, xx, xxx from 表

**[where + 条件判断]**

**[group by x desc[asc] ]**

**[having + 条件判断]**

**[order by x desc[asc] ]**

**[where + 条件判断]**

**[limit start , count ]**

* 完整的select语句
* select distinct \*
* from 表名
* where ....
* group by ... having ...
* order by ...
* limit start,count

执行顺序为：

* from 表名
* where ....
* group by ...
* select distinct \*
* having ...
* order by ...
* limit start,count

# 与python交互：

### 安装 pymysql 模块

pip3 install pymysql # windows

sudo pip3 install pymysql # ubuntu

### 参数介绍：

**con = connect（参数列表）：返回一个对象：**

* 参数host：连接的mysql主机，如果本机是'localhost'
* 参数port：连接的mysql主机的端口，默认是3306
* 参数database：数据库的名称
* 参数user：连接的用户名
* 参数password：连接的密码
* 参数charset：通信采用的编码方式，推荐使用utf8

**con对象的方法：**

con.close（） 关闭连接对象

con.commit（） 提交（修改数据时需要提交）

cs = con.cursor（） 返回Cursor 对象，用于执行SQL语句并获得结果

con.rollback（） 回滚事务，插销操作

**cs对象的方法:**

**cs.execute（语句） 执行语句，返回受影响的行数**

cs.execute(sql, [find\_name] ) 防SQL注入，sql:语句，find\_name:用户输入

cs.fetchone （） 查询结果集的一条数据

cs.fetchmany（n） 查询结果集的n条数据

cs.fetchall（） 查询结果集的所有数据

**cs.close（） 关闭游标对象**

**对象2的属性：**

cs.rowcount（） 只读属性，表示最近一次execute()执行后受影响的行数

### 语法：

* **增删改数据库 需要链接对象提交，conn.commit**

from mysql import \*  **# 导入模块**

con = connect（host=””,port=””,database=””,user=””,password=””,charset=”utf8”）#**创建链接对象**

cs = conn.cursor（）  **# 创建游标对象**

count = cs.execute（”SQL语句”）  **# 执行SQL语句，查询操作返回记录数**

**find\_name = input("请输入要查询的商品的名字：") # 防SQL 注入**

**sql = "select *\** from goods wher e name = %s " # 防SQL 注入  
cs.execute(sql, [find\_name]) # 防SQL 注入**

cs.rowcount（） **# 最近一次execute返回数据的行数或影响的行数**

cs.fetchone（） **# 返回一个元组，获取查询结果集的一条数据**

cs.fetchmany（） **# 返回一个嵌套元组的元组，默认获取结果集的下一条数据**

cs.fetchmany（n） **# 返回一个嵌套元组的元组，返回结果集的n条数据**

cs.fetfchall（） **# 返回一个嵌套元组的元组，返回结果集的所有数据**

con.commit（）  **# 提交事物，把数据真正写到数据库（修改数据时用）**

**con.rollback（） # 返回上次commit提交节点**

cs.close（）  **# 关闭cursor游标 对象**

conn.close（）  **# 关闭connection 链接对象**

### 参数化：防SQL注入

* sql语句的参数化，可以有效防止sql注入
* 注意：此处不同于python的字符串格式化，全部使用%s占位

例：格式

find\_name = input("请输入要查询的商品的名字：")  
sql = "select *\** from goods where name = %s"  
cs.execute(sql, [find\_name])

# 视图、事务、主从：

### 视图：

* 视图就是一条select语句执行后返回的结果集。
* 视图是对若干张基本表的引用，“一张虚表”，查询语句执行的结果。
* 原数据表更改， 视图表也会更改

**创建：**create view 视图名称 as select语句 ;  **如果新表中有重复字段会创建失败**

**查看：**show tables ;

**使用：**select \* from 视图名称 ；

**删除：** drop view 视图名称 ;

**重命名：**rename table 旧视图名 to 新视图名 ;

**修改：** create or replace view 视图名 as 查询SQL 语句 ;

**作用：**

1. 提高了重用性，就像一个函数
2. 对数据库重构，却不影响程序的运行
3. 提高了安全性能，可以对不同的用户
4. 让数据更加清晰

### 事务：

* 所谓事务，它是一个操作序列，这些操作要么都执行，要么都不要执行，它是一个不可分割的工作单位。

**事务四大特性**：简称：ACID

* 原子性（Atomicity）
* 一致性（Consistency）
* 隔离性（Isolation）
* 持久性（Durability）

原子性：

一个事务必须被视为一个不可分割的最小工作单元，整个事务中的所有操作要么全部提交成功，要么全部失败回滚，对于一个事务来说，不可能只执行其中的一部分操作，这就是事务的原子性

一致性：

数据库总是从一个一致性的状态转换到另一个一致性的状态。

隔离性：

通常来说，一个事务所做的修改在最终提交以前，对其他事务是不可见的。类似于加上互斥锁。

持久性：

一旦事务提交，则其所做的修改会永久保存到数据库。

**开启事务：**

**begin 或者 start transaction; # mysql 中**

**python** 执行语句时，自动开启，需手动关闭，如要再次打开用 **链接对象.begin（）**

**mysql 执行语句时，自动开启，自动关闭事务**

**注意：在开启begin 时，上个未提交的事务，会自动提交**

**提交事务：**

**commit # mysql**

**链接对象.commit（） # python**

**链接对象.autocommit（1） # python 交互时声明自动提交事务**

**回滚事务：**

**rollback # mysql**

**链接对象.rollback（） # python 交互**

#### 注意：

1. **修改数据的命令会自动的触发事务，包括insert、update、delete**
2. **而在SQL语句中有手动开启事务的原因是：可以进行多次数据的修改，如果成功一起成功，否则一起会滚到之前的数据**

### 索引：

索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。

更通俗的说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度

**索引目的：**提高查询效率，只针对查询有效：

* **查看索引：**

show index from 表名 ；

**创建（普通）索引：**

1. **如果指定字段是字符串，需要指定长度，建议长度与定义字段时的长度一致**
2. **字段类型如果不是字符串，可以不填写长度部分**
3. **建立唯一索引时，那么该索引字段再添加的数据不能与原数据相同，否则添加不进去**

**create index 索引名称 on 表名(字段名称（长度）) ；**

alter table 表名 add index 索引名称（字段名称（长度））

**创建（唯一）索引：**

唯一索引，unique，保证这个字段只会有一个值；

alter table 表名 add unique index 索引名称(字段名称(长度))

create unique index 索引名称 on 表名(字段名称(长度))

**删除索引：**

**drop index 索引名称 on 表名 ；**

**查看表的索引：**

show index from 表名 ;

**查询索引的消耗用时：**

set profiling=1 ; 开启运行时间检测：

show profiles ; 查看执行时间

set profiling = 0 关闭运行时间检测

**说明：**

* 虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行insert、update和delete。

-- 因为更新表时，不仅要保存数据，还要保存一下索引文件。

* 建立索引会占用磁盘空间的索引文件。一般情况这个问题不太严重，但如果你在一个大表上创建了多种组合索引，

-- 索引文件的会增长很快。

-- 索引只是提高效率的一个因素，如果有大数据量的表，就需要花时间研究建立最优秀的索引，或优化查询语句。

-- https://www.cnblogs.com/luyucheng/p/6289714.html

### 账户管理：

使用root 账户 进入数据库中的mysql 数据库，查看desc user 结构，

* 字段Host : 表示允许访问的主机
* 字段User : 表示用户名
* 字段authentication\_string ：表示密码，为加密后的值

例：

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

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

| host | user | authentication\_string |

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

| localhost | root | \*E74858DB86EBA20BC33D0AECAE8A8108C56B17FA |

| localhost | mysql.sys | \*THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |

| localhost | debian-sys-maint | \*EFED9C764966EDB33BB7318E1CBD122C0DFE4827 |

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

3 rows in set (0.00 sec)

#### 创建账户授权：

grant 权限列表 on 数据库.\*[\*.\*] to '用户名'@'访问主机' identified by '密码'

**多 个 权限 之间用 逗号 , 隔开**

* 可以操作python数据库的所有表，方式为 : jing\_dong.\*
* **访问主机通常使用 百分号% 表示此账户可以使用任何ip的主机登录访问此数据库**
* 访问主机可以设置成 localhost或具体的ip，表示只允许本机或特定主机访问

##### 常用权限：

* create、alter、drop、insert、update、delete、select
* 所有权限 all privileges

例：jing\_dong数据库中所有表的 查询 权限

grant select on jing\_dong.\* to 'laowang'@'localhost' identified by '123456';

例：jing\_dong数据库所用表的 所有 权限，且任意电脑可以进行链接访问

grant all privileges on jing\_dong.\* to "laoli"@"%" identified by "12345678" ；

#### 修改权限：

grant 权限名称 on 数据库 to 账户@主机 with grant option ;

注意：修改之后，要更新权限。flush privileges ;

#### 修改密码：

update user set authentication\_string=password(“新密码”) where user = “用户名” ;

例：

update user set authentication\_string=password('123') where user= 'laowang';

然后 刷新权限：flush privileges ;

#### 远程登陆：（危险慎用）

1. 修改 /etc/mysql/mysql.conf.d/mysqld.cnf 文件
2. 把43行的bind-address =127.0.0.1 前用 # 注释
3. 重启mysql服务端 （service mysql restart） # ubuntu
4. mysql -h IP地址 -u用户名 -p密码

#### 删除账户：

drop user “用户名”@”主机” ;

delete from user where user =”用户名” ;

例：

drop user “老王”@”%” ;

delete from user where user='laowang';

刷新账户，flush privileges ;

#### 忘记root密码：

[http://blog.csdn.net/lxpbs8851/article/details/10895085](http://blog.csdn.net/lxpbs8851/article/details/10895085" \t "_blank)

### 主从

#### 主从同步定义：

主从同步使得数据可以从一个数据库服务器复制到其他服务器上，在复制数据时，一个服务器充当主服务器（master），其余的服务器充当从服务器（slave）

#### 配置主从同步的基本步骤：

1. 在主服务器上，必须开启二进制日志机制和配置一个独立的ID
2. 在每一个从服务器上，配置一个唯一的ID，创建一个用来专门复制主服务器数据的账号
3. 在开始复制进程前，在主服务器上记录二进制文件的位置信息
4. 如果在开始复制之前，数据库中已经有数据，就必须先创建一个数据快照（可以使用mysqldump导出数据库，或者直接复制数据文件）
5. 配置从服务器要连接的主服务器的IP地址和登陆授权，二进制日志文件名和位置

在主服务器Ubuntu上进行备份，执行ubuntu命令：

mysqldump -uroot -pmysql --all-database --lock-all-tables > ~/master\_db.sql

**说明：**

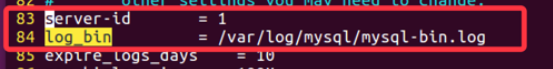
* -u ：用户名
* -p ：示密码
* --all-databases ：导出所有数据库
* --lock-all-tables ：执行操作时锁住所有表，防止操作时有数据修改
* > ： 重定向
* ~/master\_db.sql :导出的备份数据（sql文件）位置，可自己指定

在命令黑窗口中执行还原操作:DOS 窗口

mysql –uroot –pmysql < master\_db.sql # “<” 重定向

编辑设置mysqld的配置文件，设置log\_bin和server-id

sudo vim /etc/mysql/mysql.conf.d/mysqld.cnf



重启mysql 服务：

sudo service mysql restart

创建从服务器同步数据使用的帐号：

mysql –uroot –pmysql

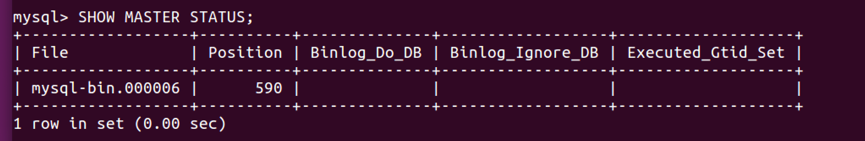
grant replication slave on \*.\* to 'slave'@'%' identified by 'slave';

flush privileges ; 刷新权限

获取主服务器的二进制日志信息：

show master status;

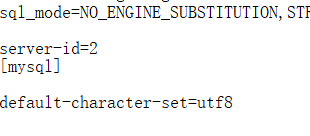
**File为使用的日志文件名字，Position为使用的文件位置，这两个参数须记下，配置从服务器时会用到**



配置从服务器slave（Windows中的MySQL）

1.找到Windows中MySQL的配置文件 my.ini

  2. 编辑my.ini文件，将server-id修改为2，并保存退出。



3 . net restart mysql 重启服务客户端 （windows 下）

进入windows的mysql，设置连接到master主服务器

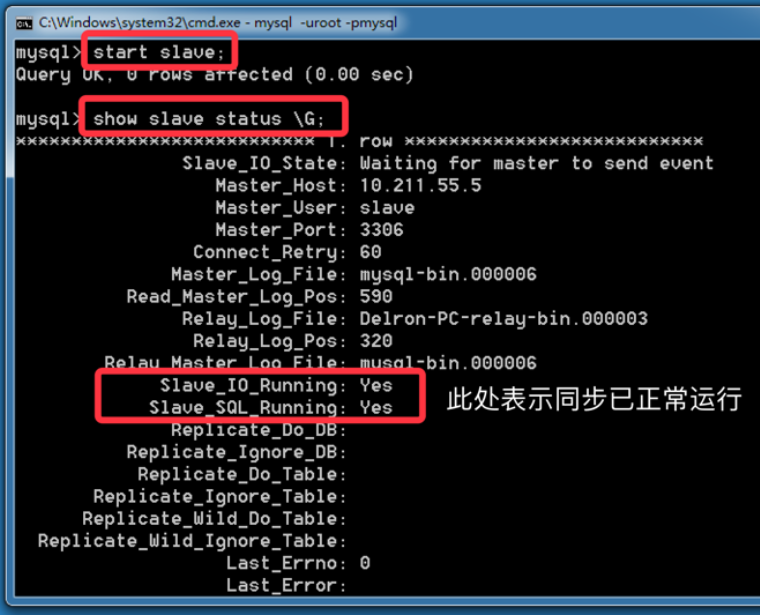
change master to master\_host='192.168.22.122', master\_user='slave', master\_password='slave', master\_log\_file= 'mysql-bin.000001', master\_log\_pos=600;

* master\_host：主服务器Ubuntu的ip地址
* master\_log\_file: 前面查询到的主服务器日志文件名
* master\_log\_pos: 前面查询到的主服务器日志文件位置

**开启同步，查看同步状态**

**start slave;**

**show slave status \G;**



# 特别注意：

## 插入数据：

insert into 表 分组 # **向表中插入组，如果插入的是组不用values（不能是元组）**

例：

insert into goods\_cates (name) select cate**\_**name from goods group by cate**\_**name;

## 创建表时添加组：

create table goods\_brands （

id int unsigned primary key auto\_increment,

name varchar(40) not null）**select brand\_name as name from goods group by brand\_name;**

## 三个表相关联：

select g.id, g.name, c.name, b.name, g.price from goods as g

inner join goods\_cates as c on g.cate\_id=c.id

inner join goods\_brands as b on g.brand\_id=b.id;

## 报错

* sql 报错 near '': 表示在最后出错

## 条件判断：

* where # 对**表**进行判断
* having # 对出来的**结果**进行判断

## 导入数据：

source 表名.sql # 导入 表 到数据库

## SQL语句：

（1）**` `** : 两个“ ` ”表示中间符号中间的是一个整体

（2）system clear : 清屏

（3）设置编码： charset=utf8 等价与 charset utf8

## 定义类型：

price decimal（10，3） # 定义price 类型为小数，最高10位，小数点后有3位

bit 相当于 bit（1） # 智能 为 0 或1

## 外键：

* **防止无效信息的插入,就是可以在插入前判断**
* **外键约束:对数据的有效性进行验证**
* **关键字: foreign key,只有 innodb数据库引擎 支持外键约束**
* **查看有无外键：**

**通过查看表创建语句来获取名称**

**show create table 表名 ;**

* **添加：**

**例：**alter table goods add foreign key (cate\_id) references goods\_cates(id);

* 取消外键约束：

**例：alter table goods drop foreign key 外键名称;**