

目录

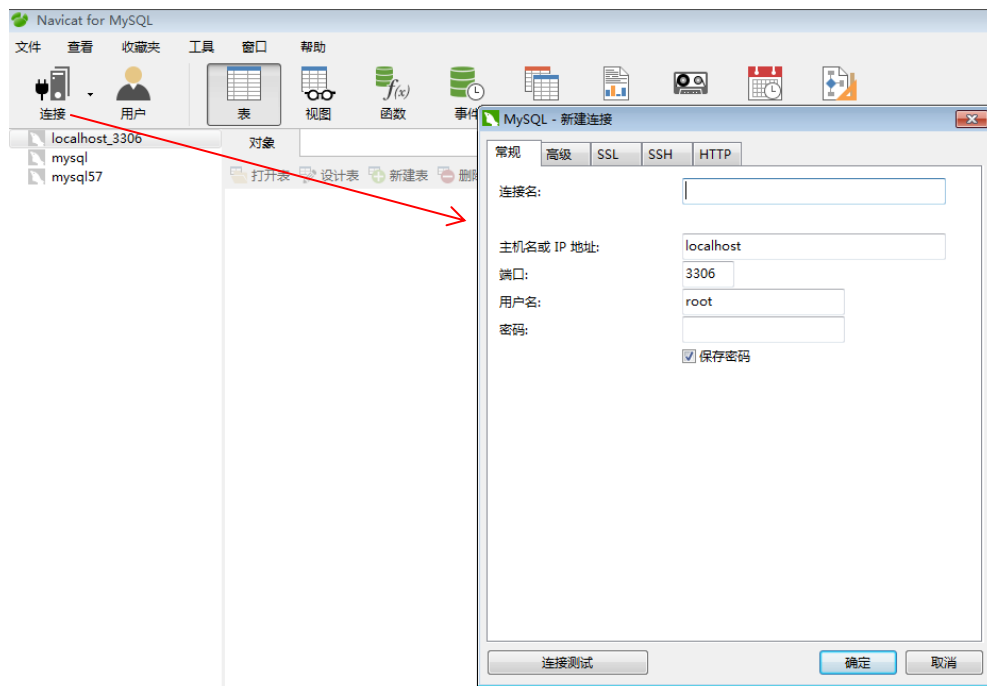
1 如何使用可视化工具 navicat 创建数据库、表	7
1.1 建立数据库.....	7
1.2 创建表.....	10
2 逻辑删除与物理删除.....	13
2.1 物理删除.....	13
2.2 逻辑删除.....	13
3 数据库操作.....	13
3.1 显示当前所有的数据库.....	13
3.2 创建数据库.....	13
3.3 删除数据库.....	13
3.4 切换数据库.....	13
3.5 查看当前选择了那个数据库.....	14
4 表操作.....	14
4.1 查看当前数据库中有哪些表.....	14
4.2 创建表.....	14
4.3 查看表结构.....	14
4.4 删除表.....	14
4.5 查看建表语句.....	15
4.6 重命名表名.....	15
4.7 修改表（这个在黑屏终端不要使用，了解就行）	15
5 数据操作.....	16
5.1 像表中添加数据-----增	16
5.2.1 全列插入（插入所有字段）	16
5.2.2 部分列插入（插入部分列字段）	16
5.2.3 同时插入多条数据.....	16
5.2 数据删除操作-----删	16

5.2.1 逻辑删除.....	17
5.3 数据修改操作-----改	17
5.4 简单查询语句-----查	17
6 数据备份与数据恢复.....	18
6.1 数据备份.....	18
6.2 原始服务器将数据导出到一个文件夹.....	18
6.3 现在数据库创建一个数据库.....	18
6.4 数据恢复.....	18
6.5 查看数据恢复情况.....	18
7 数据查询-----详细“查询”语句.....	19
7.1 基本语法.....	19
7.1.1 查看所有字段.....	19
7.1.2 查看特定字段.....	19
7.2 消除重复行-----distinct.....	19
7.3 条件查询-----where 数据筛选	20
7.3.1 比较运算符.....	20
7.3.2 逻辑运算符.....	20
7.3.3 模糊查询-----like.....	21
7.3.4 范围查询-----in、 between...and	22
7.3.5 空判断-----is null.....	22
7.3.6 非空判断-----is not null.....	22
7.3.7 优先级比较.....	22
7.3.8 聚合函数-----count(),max(),min(),sum(),avg()	22
7.3.9 分组-----group by	23
7.3.8 分组后的数据筛选-----having	23
7.3.9 排序-----order by	26
7.3.10 获取部分行-----limit	26
7.3.11 分页.....	26
7.3.12 一个完整的 select 语句的执行顺序.....	27
7.3.13 mysql 语句执行顺序	27

8 MySQL 高级部分内容.....	29
8.1 连接查询.....	30
8.1.1 inner join: 内连接	30
8.1.2 left join: 左连接	31
8.1.3 right join: 右连接.....	32
8.2 更完整的 select 语句.....	32
8.3 自关联.....	33
8.4 视图.....	35
8.5 事务.....	36
8.5.1 事务的 4 个特性.....	36
8.5.2 事务需要注意的地方	36
8.5.3 事务的基本原理.....	36
8.5.4 事务操作语句.....	37
8.5.5 执行事务操作前提条件.....	37
9 与 python 交互.....	38
9.1 连接数据库.....	38
9.2 远程连接数据库-----当 mysql 服务器不在本机电脑上	39
9.3 创建数据库表.....	41
9.4 数据库插入数据-----“增”	42
9.5 数据库更新数据-----“改”	44
9.6 数据库删除操作-----“删”	45
9.7 数据库查询操作-----“查”	46
9.8 封装 mysql 语句为一个“类”	48

1 如何使用可视化工具 navicat 创建数据库、表

1.1 建立数据库

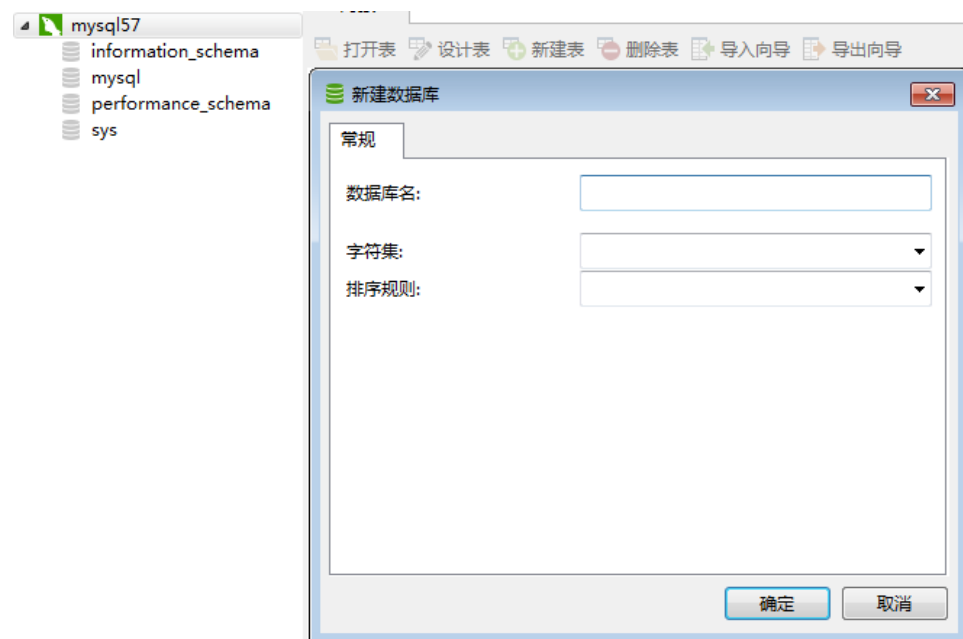


出现“连接成功”，即可

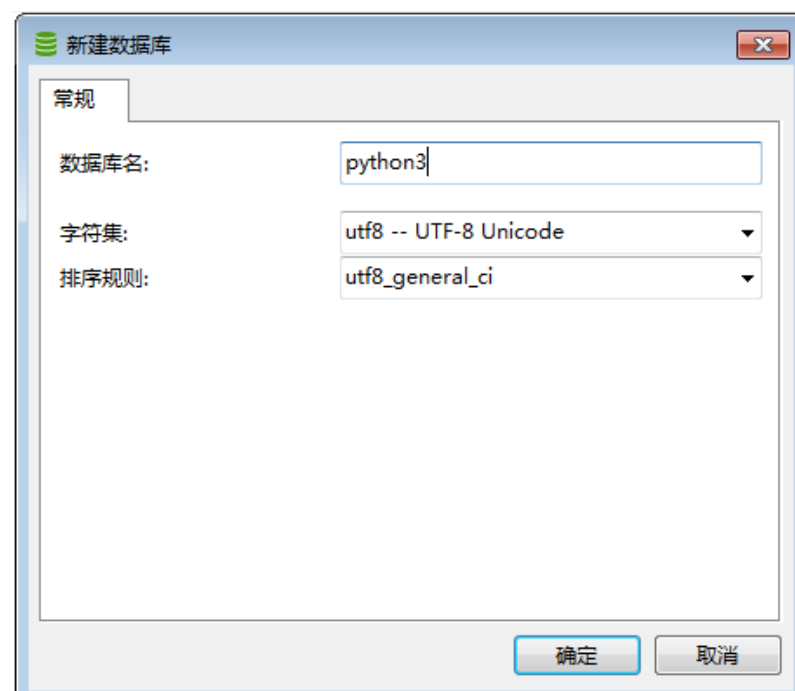
光标放在 mysql57 处→点击“右键”→点击“打开连接”→mysql 57 就高亮显示了



点击“右键”→点击“新建数据库”，则会出现以下窗口



数据库名：可按照需求随意设计；字符集、排序规则：按照以下来

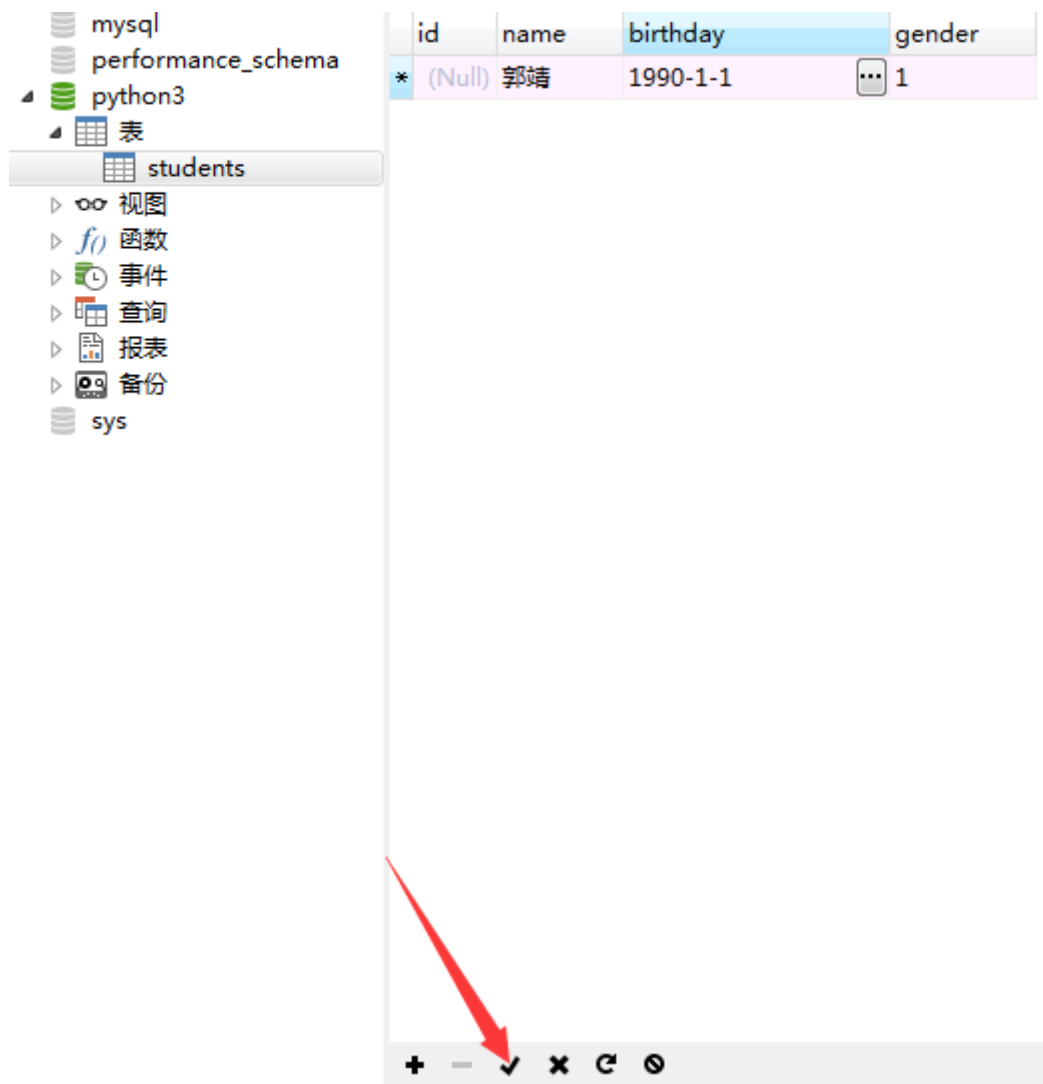


1.2 创建表

The screenshot illustrates the steps to create a table in MySQL Workbench:

- Database Structure:** The sidebar shows the hierarchy: `mysql57` > `python3` > `表` (Tables).
- Toolbar:** The `新建表` (New Table) button is highlighted with a red arrow.
- Table Definition:** The main area displays a table with the following columns:

名	类型	长度	小数点	不是 null	主键
id	int	11	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
name	varchar	10	0	<input type="checkbox"/>	<input type="checkbox"/>
birthday	datetime	0	0	<input type="checkbox"/>	<input type="checkbox"/>
gender	bit	1	0	<input type="checkbox"/>	<input type="checkbox"/>
- Properties Panel:** The bottom panel shows settings for the selected column:
 - 默认: NULL
 - 注释:
 - 字符集: utf8
 - 排序规则: utf8_general_ci
 - 键长度:
 - ☐ 二进制



1.3 一些常用命令

1.3.1 启动服务

net start mysql

1.3.2 停止服务

net stop mysql

1.3.3 连接数据

mysql -u root -p

1.3.4 退出登录

exit 或 quit

1.3.5 查看版本（连接后执行）

select version();

```
mysql> select version();
+-----+
| version() |
+-----+
| 5.7.15-log |
+-----+
1 row in set (0.03 sec)
```

1.3.6 显示当前时间

select now();

```
mysql> select now();
+-----+
| now() |
+-----+
| 2019-04-28 20:19:52 |
+-----+
1 row in set (0.04 sec)
```

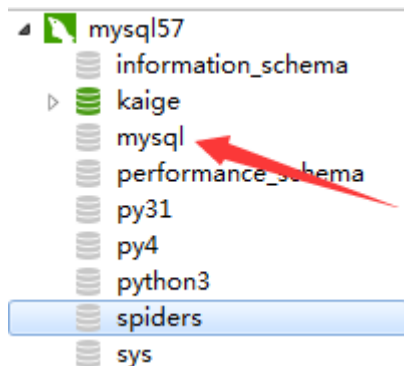
1.3.7 远程连接

格式: mysql -h ip 地址 -u 用户名 -p

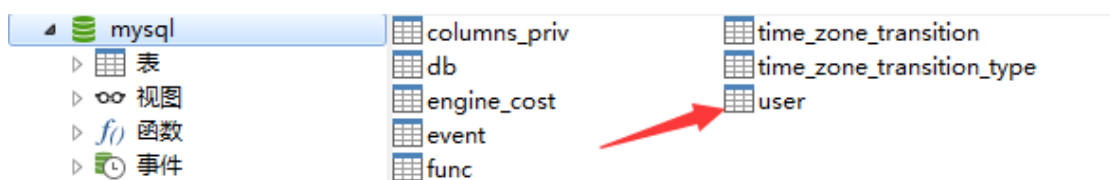
输入对方 mysql 密码

执行远程连接之前: 首先执行如下操作, 不然会报错

打开 mysql 系统库



打开其中的 user 表



将 root 左侧空格中的“localhost”替换为“%”

Host	User	Select_priv	Insert_priv	Update_priv
%	root	Y	Y	Y
localhost	mysql.sys	N	N	N

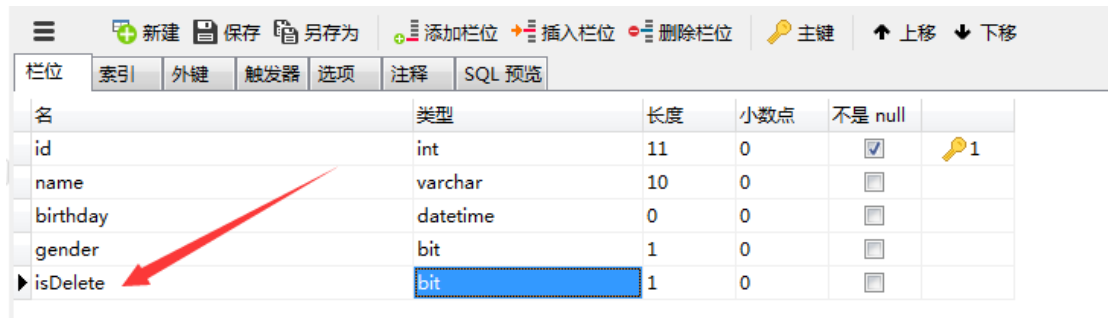
2 逻辑删除与物理删除

2.1 物理删除

直接从数据库中删除数据，删除以后就永久没有了。

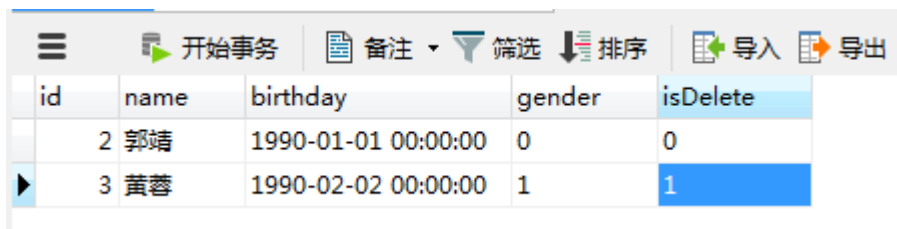
2.2 逻辑删除

不是真正的删除，isDelete=0 表示不删除；isDelete=1 表示删除。



名	类型	长度	小数点	不是 null	
id	int	11	0	<input checked="" type="checkbox"/>	1
name	varchar	10	0	<input type="checkbox"/>	
birthday	datetime	0	0	<input type="checkbox"/>	
gender	bit	1	0	<input type="checkbox"/>	
isDelete	bit	1	0	<input type="checkbox"/>	

设计原理：0 表示数据存在，1 表示数据没有，象征意义上的删除。当我们把 1 改为 0 后，又表示数据存在。



id	name	birthday	gender	isDelete
2	郭靖	1990-01-01 00:00:00	0	0
3	黄蓉	1990-02-02 00:00:00	1	1

3 数据库操作

3.1 显示当前所有的数据库

show databases;

3.2 创建数据库

格式：create database 数据库名 charset=utf8;

示例：create database python3 charset=utf8;

3.3 删除数据库

格式：drop database 数据库名;

示例：drop database python3;

3.4 切换数据库

格式：use 数据库名

示例：use python3;

3.5 查看当前选择了那个数据库

`select database();`

4 表操作

4.1 查看当前数据库中有哪些表

`show tables;`

4.2 创建表

注意默认值写法：default 1，而不是 default(1)

```
create table student(  
id int auto_increment primary key not null,  
name varchar(10) not null,  
gender bit default 1,  
birthday datetime);
```

4.3 查看表结构

格式：desc 表名;

示例：desc student;

```
mysql> desc students;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(10)	NO		NULL	
gender	bit(1)	YES		b'1'	
birthday	datetime	YES		NULL	
isDelete	bit(1)	YES		b'0'	

4.4 删除表

格式：drop table 表名;

示例：drop table student;

4.5 查看建表语句

格式：show create table 表名;

示例：show create table students;

```
! students ! CREATE TABLE `students` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `name` varchar(10) NOT NULL,  
  `gender` bit(1) DEFAULT b'1',  
  `birthday` datetime DEFAULT NULL,  
  `isDelete` bit(1) DEFAULT b'0',  
  PRIMARY KEY (`id`)  
> ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8 !
```

4.6 重命名表名

格式：rename table 原表名 to 新表名;

示例：rename table student to students;

4.7 修改表（这个在黑屏终端不要使用，了解就行）

注：一般表结构有了，数据也有了，不要轻易修改表结构，增加、删除、修改列

格式：alter table 表名 add|change|drop 列名 类型;

示例：alter table student add isDelete bit default 0;

5 数据操作

5.1 像表中添加数据-----增

5.2.1 全列插入（插入所有字段）

格式：

```
insert into 表名 values(...);
```

示例：

```
insert into student values(0,'郭靖',1,'1990-1-1',0);
```

注意：表结构中有 id 字段，虽然设置了主键、自动递增，mysql 可以自动维护，但是，在“**全列插入**”的时候，这个位置还必须写（通常写 0，插入后以实际为准），起到占位作用。

5.2.2 部分列插入（插入部分列字段）

格式：

```
insert into 表名(列 1,列 2,...) values(值 1,值 2,...);
```

示例：

```
insert into student(name) values('黄蓉');
```

```
insert into student(gender,name) values(0,'小龙女');
```

5.2.3 同时插入多条数据

格式：

```
insert into 表名 values('杨过'),('雕'),('郭襄');
```

示例：

```
insert into student(name) values(...),(...),(...) .....;
```

5.2 数据删除操作-----删

格式：（不写 where 条件，所有的数据都将删除）

```
delete from 表名 where 条件;
```

示例：

```
delete from student where id=3;
```

5.2.1 逻辑删除

注意：其实就是一个修改操作，以下两个语句配合使用

`update student set isDelete=1 where id=6;` #把不想看到数据 isDelete 设置为 1;

`select * from student where isDelete=0;` #然后显示的时候把 isDelete=1 的值过滤；

注意：delete 属于物理删除，一旦删除，数据就不存在，因此，重要数据建议用逻辑删除，不重要数据采用 delete 删除

5.3 数据修改操作-----改

格式：（不写 where 条件，相当于把一整列数据都修改了）

`update 表名 set 列 1=值 1,列 2=值 2,..... where 条件;`

示例：

`update student set birthday='1990-2-2' where id=2;`

`update student set gender=0,birthday='2019-4-24' where id=6;`

5.4 简单查询语句-----查

格式：select * from 表名;

示例：select * from student;

说明：查询表中的全部数据

```
mysql> select * from students;
```

id	name	gender	birthday	isDelete
1	郭靖	0	1990-01-01 00:00:00	
2	黄蓉	0	1990-02-02 00:00:00	
3	小龙女		1995-07-05 00:00:00	
4	杨过	0	2001-10-20 00:00:00	
6	郭襄		2017-02-13 00:00:00	0
7	黄药师	0	NULL	

6 数据备份与数据恢复

6.1 数据备份

注：数据备份与数据恢复（把数据从一个服务器迁移到另外一个服务器，先在原始服务器上备份数据，然后再新服务器上恢复数据）

6.2 原始服务器将数据导出到一个文件夹

```
C:\Program Files\MySQL\MySQL Server 5.7\bin>MySQLdump -u root -p -h  
127.0.0.1 python3 >C:\Users\Administrator\Desktop\c.sql  
Enter password: *****
```

6.3 现在数据库创建一个数据库

```
C:\Users\Administrator>create database py31 charset=utf8;
```

6.4 数据恢复

```
C:\Program Files\MySQL\MySQL Server 5.7\bin>mysql -u root -p py31 <  
C:\Users\Administrator\Desktop\c.sql  
Enter password: *****
```

6.5 查看数据恢复情况

```
C:\Users\Administrator>mysql -u root -p  
Enter password: *****
```

```
mysql> use py31;
```

```
mysql> show tables;
```

7 数据查询-----详细“查询”语句

7.1 基本语法

格式：

```
select * from 表名;
```

说明：

- A、from 关键字后面是表名，表示数据是来源于这张表
- B、select 后面写的是表中的列名，如果是*表示在结果集中，显示表中的所有列
- C、在 select 的列名部分，可以使用 as 为列起别名，这个别名显示在结果集中
- D、要查询指定的多个列，之间使用逗号分隔

7.1.1 查看所有字段

```
select * from student;
```

7.1.2 查看特定字段

```
select id ,name from student;
```

```
select gender as "性别" from students;
```

```
mysql> select gender as "性别" from students;
+-----+
| 性别 |
+-----+
| 男   |
| 男   |
```

7.2 消除重复行-----distinct

示例：

```
select gender from student;
```

```
select distinct gender from student;
```

注意：

- A、在 select 后面、列前面，使用 distinct 可以消除重复的行。
- B、对于一个字段，“重复行”指的是每一行中相同的行。
- C、对于多个字段，“重复行”指的是所有字段完全一样的行，看下图。

name	gender
郭靖	男
黄蓉	女
小龙女	女
杨过	男
郭襄	女

红色部分虽然 gender 是一样的，但是 name 不一样，所以不能算重复行。注：distinct 是针对所有行的所有字段，每一行只要有一个字段不同，就属于不重复行。

7.3 条件查询-----where 数据筛选

语法：

```
select * from 表名 where 条件;
```

7.3.1 比较运算符

等于 =

大于 >

小于 <

大于等于 >=

小于等于 <=

不等于 !=或者<>

```
select * from student where id>3;
```

```
select name from student where isdelete=0;
```

7.3.2 逻辑运算符

且 and

或 or

非 not

注：not 用法不太熟悉

```
select ( from student where id>3 and gender=0;
```

```
select * from student where not id>3; # id 不大于 3 的学生;
```


7.3.3 模糊查询-----like

% 表示匹配一个或多个任意字符

```
select * from student where name like '黄%';
```

```
select * from student where name like '%龙%';
```

```
select * from student where name like '黄%' or name like '%靖%';
```

_ 表示匹配一个任意字符

```
select * from student where name like '黄_';
```

```
select * from student where name like '黄__';    # 两个下划线
```

7.3.4 范围查询-----in、 between...and...

非连续范围查询 in

```
select * from student where id in(1,2,3,8);
```

注：假如没有第 8 条数据，不会报错，只是没有返回值而已

连续范围查询 between...and...

```
select * from student where id between 3 and 8;
```

```
select * from student where id between 3 and 8 and gender=1;
```

注 1：假如没有第 8 条数据，不会报错，只是没有返回值而已。

注 2： between...and...和 and 同时运用，会优先计算 between...and...。

7.3.5 空判断-----is null

7.3.6 非空判断-----is not null

```
select * from student where birthday is null; # 查询没有生日信息的同学
```

注意： null 与 ' ' 的不同。null 就是空，在计算机中不占用任何内存； ' ' 为空字符串，需要占据一定内存。

7.3.7 优先级比较

小括号 —> not —> 比较运算符 —> 逻辑运算符 #优先级从高到低排列

注意： and 比 or 优先运算。如果同时出现并希望 or 优先运算，可以使用小括号。

7.3.8 聚合函数-----count(),max(),min(),sum(),avg()

A、 count(*) 表示计算总行数，括号中可以写“*”和“列名”

```
select count(*) from student where isdelete=0;
```

顺序：先 from student 拿到原始数据，再 where isdelete=0 进行筛选，后进行 count(*) 计算

B、 max(列) 表示求此列的最大值

```
select max(id) from student where gender=1;
```

C、min(列) 表示求此列的最小值

```
select min(id) from student where gender=1;
```

子查询：对于上述任务，我们非要查看最小 id 人的其他信息怎么办

```
select * from student where id=( select min(id) from student where gender=1);
```

D、sum(列) 表示求此列的和

```
select sum(id) from student where gender=1 and isdelete=0;
```

E、avg(列) 表示求此列的平均值

```
select avg(id) from student where gender=1 and isdelete=0;
```

7.3.9 分组-----group by

目的：进行分组的目的，就是为了进行聚合运算对数据进行统计。

语法：select 列 1,列 2,聚合...from 表名 group by 列 1,列 2,列 3...;

```
select count(*) from student group by gender;
```

```
mysql> select count(*) from student group by gender;
+-----+
| count(*) |
+-----+
|         2 |
|         4 |
+-----+
```

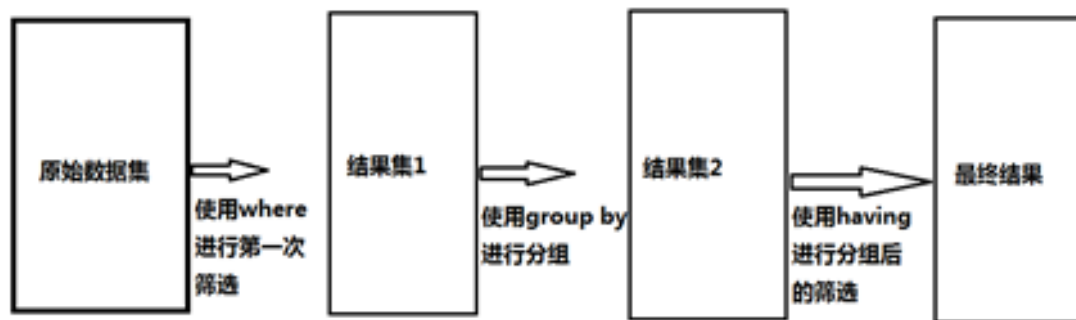
```
select gender,count(*) from student group by gender;
```

```
mysql> select gender,count(*) from student group by gender;
+-----+-----+
| gender | count(*) |
+-----+-----+
|       |         2 |
|  ♀    |         4 |
+-----+-----+
```

7.3.8 分组后的数据筛选-----having

语法：select 列 1,列 2,聚合...from 表名 group by 列 1,列 2,列 3...having 列 1,...
聚合...;

where 与 having 的区别：



注意：where 是对原始数据集的筛选，having 是对分组后的结果数据进行筛选。

注意：having 的运算符与 where 相同，像“逻辑运算符”、“比较运算符”

示例：

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

```
mysql> select gender,count(*) from students group by gender;
+-----+-----+
| gender | count(*) |
+-----+-----+
| 男     | 2       |
| 女     | 4       |
+-----+-----+
```

select gender,count(*) from student group by gender having gender=0;

```
mysql> select gender,count(*) from student group by gender having gender=0;
+-----+-----+
| gender | count(*) |
+-----+-----+
|       | 2       |
+-----+-----+
```

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

```
mysql> select gender,count(*) from student group by gender having count(*)>2;
+-----+-----+
| gender | count(*) |
+-----+-----+
| 女     | 4       |
+-----+-----+
```

mysql> select gender,count(*) as rs from student group by gender having count(*)>2;

```
mysql> select gender,count(*) as rs from student group by gender having count(*)>2;
+-----+-----+
| gender | rs      |
+-----+-----+
| 女     | 4       |
+-----+-----+
```

7.3.9 排序-----order by

语法: select * from 表名 order by 列 1 asc/desc,列 2 asc/desc...;

说明:

A、 将数据按照列 1 进行排序, 如果某些列 1 的值相同, 则按照列 2 进行排序。

B、 默认按照从小到大的顺序排序-----升序 asc。

C、 asc: 升序; desc: 降序。

示例:

```
select * from student
```

```
where isdelete=0 and gender=1
```

```
order by id desc;
```

id	name	gender	birthday	isDelete
7	黄药师	♂	NULL	
4	杨过	♂	NULL	
2	黄蓉	♀	1990-02-02 00:00:00	
1	郭靖	♂	1990-01-01 00:00:00	

7.3.10 获取部分行-----limit

语法: select * from 表名 limit start,count

说明: 从 start 开始, 获取 count 条数据, start 索引从 0 开始

```
select * from student limit 1,5;
```

id	name	gender	birthday	isDelete
2	黄蓉	♀	1990-02-02 00:00:00	
3	小龙女		NULL	
4	杨过	♂	NULL	
6	郭襄		2019-04-24 00:00:00	♂
7	黄药师	♂	NULL	

7.3.11 分页

已知: 每一页显示 m 条数据, 当前显示第 n 页

求总页数: 此段逻辑后面会在 python 中实现

步骤: 查询总条数 p1

使用 $p1$ 除以 m 得到 $p2$

如果整除，则 $p2$ 为总页数

如果不整除，则 $p2+1$ 为总页数

求第 n 页的数据

语法：

```
select * from students where isdelete=0 limit (n-1)*m,m
```

7.3.12 一个完整的 select 语句的执行顺序

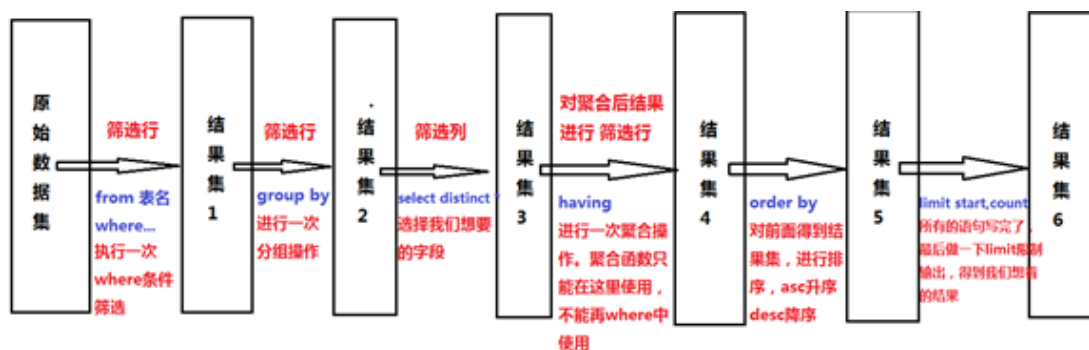
mysql 语句编写顺序

- ① select distinct *
- ② from 表名
- ③ where ...
- ④ group by ...having...
- ⑤ order by ...
- ⑥ limit start,count

注 1：① ②属于最基本语句，必须含有

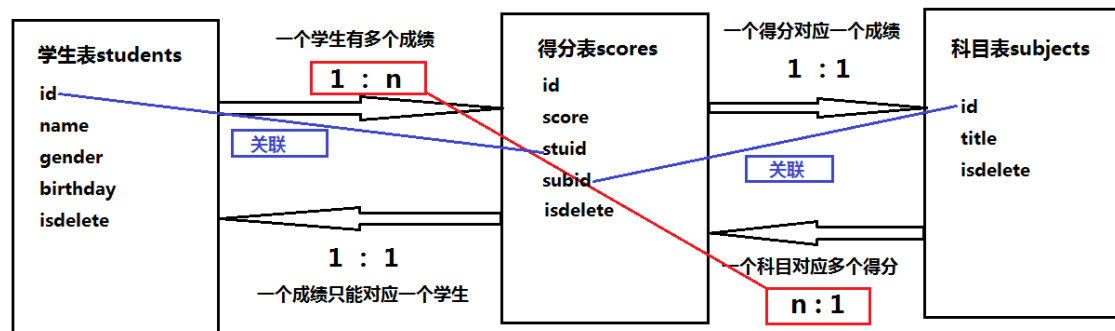
注 2：① ②可以与③ ④ ⑤ ⑥ 中任一搭配，也可以与③ ④ ⑤ ⑥ 中多个同时搭配

7.3.13 mysql 语句执行顺序



8 MySQL 高级部分内容

存储关系：1、先分析有没有关系，再确定是几对几的关系



特种重要：对于 1 对多的关系，关联字段像 **stuid** 和 **subid**，都放在“多”的那张表中，也就是 **scores** 中，同时还要在 **scores** 表中，设置外键约束。

建立关系表：2、将关系字段存下来

外键约束：3、为了保证数据的有效性，建立外键约束

对于外键约束的说明：可以看到 **subjects** 中 **id** 代表的是科目。采用 **subid** 关联 **subjects** 中 **id**，当我们朝 **scores** 中插入数据时，若设置外键约束后，插入与这些科目无关的，则会报错，表示无法插入；若不设置外键约束，虽然可以插入，但是插入的是无效数据，没啥用。

id	title	isdelete
1	python	0
2	linux	1
3	mysql	0
4	mongodb	0
5	redis	1

```
create table scores(  
id int auto_increment primary key not null,  
score decimal(4,1),  
stuid int,  
subid int,  
foreign key(stuid) references student(id),  
foreign key(subid) references subject(id));
```

8.1 连接查询

8.1.1 inner join: 内连接

inner join: 内连接（返回的是：能匹配上的数据）

inner join 不分顺序，以下结果都是一样的

```
select students.name,subjects.title,scores.score  
from scores
```

```
inner join students on scores.stuid=students.id
```

```
inner join subjects on scores.subid=subjects.id;
```

```
select students.name,subjects.title,scores.score  
from students
```

```
inner join scores on scores.stuid=students.id
```

```
inner join subjects on scores.subid=subjects.id;
```

```
select students.name,subjects.title,scores.score  
from subjects
```

```
inner join scores on scores.subid=subjects.id
```

```
inner join students on scores.stuid=students.id;
```

	name	title	score
▶	郭靖	python	100
	郭靖	linux	90
	郭靖	mysql	80
	黄蓉	python	99
	黄蓉	linux	98
	黄蓉	mysql	97
	小龙女	python	90
	小龙女	linux	80
	小龙女	mysql	70

8.1.2 left join: 左连接

注意：以 left 左边的 A 表为准，left 左边表中数据在结果中都会显示，若 A 表中数据在 B 表中没有数据对应，则采用 Null 填充

select *

from students

left join scores on students.id=scores.stuid;

id	name	gender	birthday	isDelete	id1	score	stuid	subid	isdelete1
1	郭靖	1	1990-01-01 00:00:00	0	1	100	1	1	0
1	郭靖	1	1990-01-01 00:00:00	0	3	90	1	2	0
1	郭靖	1	1990-01-01 00:00:00	0	4	80	1	3	0
2	黄蓉	1	1990-02-02 00:00:00	0	5	99	2	1	0
2	黄蓉	1	1990-02-02 00:00:00	0	6	98	2	2	0
2	黄蓉	1	1990-02-02 00:00:00	0	7	97	2	3	0
3	小龙女	0	1995-07-05 00:00:00	0	8	90	3	1	0
3	小龙女	0	1995-07-05 00:00:00	0	9	80	3	2	0
3	小龙女	0	1995-07-05 00:00:00	0	10	70	3	3	0
4	杨过	1	2001-10-20 00:00:00	0	(Null)	(Null)	(Null)	(Null)	(Null)
6	郭襄	0	2017-02-13 00:00:00	1	(Null)	(Null)	(Null)	(Null)	(Null)
7	黄药师	1	(Null)	0	(Null)	(Null)	(Null)	(Null)	(Null)

select *

from scores

left join students on students.id=scores.stuid;

id	score	stuid	subid	isdelete	id1	name	gender	birthday	isDelete1
1	100	1	1	0	1	郭靖	1	1990-01-01 00:00:00	0
3	90	1	2	0	1	郭靖	1	1990-01-01 00:00:00	0
4	80	1	3	0	1	郭靖	1	1990-01-01 00:00:00	0
5	99	2	1	0	2	黄蓉	1	1990-02-02 00:00:00	0
6	98	2	2	0	2	黄蓉	1	1990-02-02 00:00:00	0
7	97	2	3	0	2	黄蓉	1	1990-02-02 00:00:00	0
8	90	3	1	0	3	小龙女	0	1995-07-05 00:00:00	0
9	80	3	2	0	3	小龙女	0	1995-07-05 00:00:00	0
10	70	3	3	0	3	小龙女	0	1995-07-05 00:00:00	0

8.1.3 right join: 右连接

select *

from scores

right join students on students.id=scores.stuid;

id	score	stuid	subid	isdelete	id1	name	gender	birthday	isDelete1
1	100	1	1	0	1	郭靖	1	1990-01-01 00:00:00	0
3	90	1	2	0	1	郭靖	1	1990-01-01 00:00:00	0
4	80	1	3	0	1	郭靖	1	1990-01-01 00:00:00	0
5	99	2	1	0	2	黄蓉	1	1990-02-02 00:00:00	0
6	98	2	2	0	2	黄蓉	1	1990-02-02 00:00:00	0
7	97	2	3	0	2	黄蓉	1	1990-02-02 00:00:00	0
8	90	3	1	0	3	小龙女	0	1995-07-05 00:00:00	0
9	80	3	2	0	3	小龙女	0	1995-07-05 00:00:00	0
10	70	3	3	0	3	小龙女	0	1995-07-05 00:00:00	0
(Null)	(Null)	(Null)	(Null)	(Null)	4	杨过	1	2001-10-20 00:00:00	0
(Null)	(Null)	(Null)	(Null)	(Null)	6	郭襄	0	2017-02-13 00:00:00	1
(Null)	(Null)	(Null)	(Null)	(Null)	7	黄药师	1	(Null)	0

select *

from students

right join scores on students.id=scores.stuid;

id	name	gender	birthday	isDelete	id1	score	stuid	subid	isdelete1
1	郭靖	1	1990-01-01 00:00:00	0	1	100	1	1	1 0
1	郭靖	1	1990-01-01 00:00:00	0	3	90	1	2	2 0
1	郭靖	1	1990-01-01 00:00:00	0	4	80	1	3	3 0
2	黄蓉	1	1990-02-02 00:00:00	0	5	99	2	1	1 0
2	黄蓉	1	1990-02-02 00:00:00	0	6	98	2	2	2 0
2	黄蓉	1	1990-02-02 00:00:00	0	7	97	2	3	3 0
3	小龙女	0	1995-07-05 00:00:00	0	8	90	3	1	1 0
3	小龙女	0	1995-07-05 00:00:00	0	9	80	3	2	2 0
3	小龙女	0	1995-07-05 00:00:00	0	10	70	3	3	3 0

8.2 更完整的 select 语句

1 select distinct *

2 from 表 1 inner|left|right join 表 2 on 表 1 与表 2 的关系...

3 where ...

4 group by ...having...

5 order by ...

6 limit start,count

注意：2 中可以是两个表的联系（两个 join 语句），也可以是三个表的联系（三

个 join 语句)

8.3 自关联

相关链接: <https://blog.csdn.net/hubingzhong/article/details/81277220>

```
create table areas(  
id int auto_increment primary key not null,  
title varchar(20),  
pid int,  
foreign key(pid) references areas(id)  
);
```

```
insert into areas  
values ('130000', '河北省', NULL),  
( '130100', '石家庄市', '130000'),  
( '130400', '邯郸市', '130000'),  
( '130600', '保定市', '130000'),  
( '130700', '张家口市', '130000'),  
( '130800', '承德市', '130000'),  
( '410000', '河南省', NULL),  
( '410100', '郑州市', '410000'),  
( '410300', '洛阳市', '410000'),  
( '410500', '安阳市', '410000'),  
( '410700', '新乡市', '410000'),  
( '410800', '焦作市', '410000');
```

自关联查询语句

```
select city.*  
from areas as city  
inner join areas as province on city.pid=province.aid
```

where province.title="陕西省"

8.4 视图

目的：就是将复杂的 select 语句进行封装，方便以后进行调用。说白了，就是 select 语句太长，每次使用起来不方便

```
select *  
  
from scores  
  
inner join students on scores.stuid=students.id  
  
inner join subjects on scores.subid=subjects.id;
```

id	score	stuid	subid	isdelete	id1	name	gender	birthday	isDelete1	id2	title	isdelete2
1	100	1	1	0	1	郭靖	1	1990-01-01 00:00:00	0	1	python	0
3	90	1	2	0	1	郭靖	1	1990-01-01 00:00:00	0	2	linux	1
4	80	1	3	0	1	郭靖	1	1990-01-01 00:00:00	0	3	mysql	0
5	99	2	1	0	2	黄蓉	1	1990-02-02 00:00:00	0	1	python	0
6	98	2	2	0	2	黄蓉	1	1990-02-02 00:00:00	0	2	linux	1
7	97	2	3	0	2	黄蓉	1	1990-02-02 00:00:00	0	3	mysql	0
8	90	3	1	0	3	小龙女	0	1995-07-05 00:00:00	0	1	python	0
9	80	3	2	0	3	小龙女	0	1995-07-05 00:00:00	0	2	linux	1
10	70	3	3	0	3	小龙女	0	1995-07-05 00:00:00	0	3	mysql	0

对于上述语句：

我们在做查询时，可能会经常使用此语句，该语句很长，每次写起来肯定不方便。

于是，我们可以通过：创建视图将该语句封装起来。

这样做对于以后修改很方便：

上述语句，以后在不同的 select 语句中会多次用到，假如每次都像上述一个个敲出来，一旦以后由于某种原因需要修改上述代码。可能每一个 select 语句都要修改。假如封装为一个视图，我们只需要修改保存好的这一个视图即可，其余使用了这个视图的 select 语句，相当于自动修改了

创建视图代码如下：

```
create view v_stu_sco_sub as  
  
select stu.*,sco.score,sub.title  
  
from scores as sco  
  
inner join students as stu on sco.stuid=stu.id  
  
inner join subjects as sub on sco.subid=sub.id;
```

红色：v_stu_sco_sub 表示将创建的视图命名，前面最后加一个前缀 v，让我们知道这是一个视图

蓝色：as 千万别忘记写了

8.5 事务

8.5.1 事务的 4 个特性

- 1 一组操作要么都成功，要么都失败，这一组操作是不可拆分的-----→原子性
- 2 在所有操作没有执行完毕之前，其它会话/窗口不能够看见中间数据的改变过程，只是当前窗口可以看见数据改变过程-----→隔离性
- 3 事务发生前、发生后，数据总额仍然是匹配的。eg: 模拟一个人给另外一个人转账来说，转账之前，2 人的金额总和为 400；转账以后，2 人的金额总额仍为 400-----→一致性
- 4 一旦 commit 提交后，事务产生的影响就不能够撤销了，已经实实在在把数据修改了-----→持久性

8.5.2 事务需要注意的地方

- 1 默认情况下，mysql 的事务是自动提交的。
- 2 set autocommit=0 取消 mysql 的事务自动提交功能。
set autocommit=1 设置 mysql 的事务自动提交功能。
- 3 commit 提交操作，一旦提交完成，数据就真正的改变了。
- 4 rollback 回滚操作，如果事务回滚，那么事务就会返回到最开始的状态。
- 5 mysql 中有一个叫做还原点的概念，操作失败后（eg: 断电导致电脑关机、操作失误），采用 rollback，可以回到开始的位置。

8.5.3 事务的基本原理

只有当执行增、删、改操作时（没有“查”），才可以执行事务操作。

有事务和无事务：执行增、删、改语句，他们保存的结果是不一样的。无事务时，执行操作后的结果在“数据表文件中显示”，增、删、改语句一旦执行完毕，实际上是做了两件事：1、语句执行 2、提交。有事务时，当没有 commit 操作，操作过程记录会先保存在“事务的日志文件中”，因此，执行的操作过程，在其他窗口不能看见。只有执行了 commit 操作，操作的结果会在“数据表文件中显示”，此时，其他窗口才可以看见。假如在执行过程中出现意外情况，我就不会执行 commit 操作，而会执行 rollback 操作，此时，操作则会回到最开始的位置。

8.5.4 事务操作语句

开启事务：start transaction 或 begin 或 set autocommit=0

回滚：rollback。当执行过程中，出现错误时

提交操作：commit。在执行过程中，没有出现错误，只有执行 commit 操作，数据才算是真正被修改了，此时，其他窗口可以看见修改后的结果。

8.5.5 执行事务操作前提条件

表的类型必须是 innodb 或 bdb 类型，才可以对此表执行事务操作。

```
use py4;

create table account(
id int auto_increment primary key not null,
username varchar(10),
money int);

insert into account values(null,'张旭',1000),
(null,'文德',1000),
(null,'程伟',1000);
```

9 与 python 交互

9.1 连接数据库

```
Import pymysql

# 连接数据库

# 参数 1 : mysql 服务器所在的主机 IP

# 参数 2 : 用户名

# 参数 3 : 密码

# 参数 4 : 连接的 mysql 主机的端口, 默认是 3306

# 参数 5 : 连接的数据库名

# 参数 6 : 通信采用的编码方式, 默认是'gb2312', 要求与数据库创建时指定的
          编码一致, 否则中文会乱码

db = pymysql.connect(host='localhost' , user='root' , password='123456' ,
port=3306 ,db='spiders' , charset='utf8')

# 创建一个 cursor 对象

cursor = db.cursor()

sql = 'select version()'

# 执行 sql 语句

cursor.execute(sql)

# 获取返回的信息

data = cursor.fetchone()

print('database version: ',data)

# 断开连接

db.close()
```

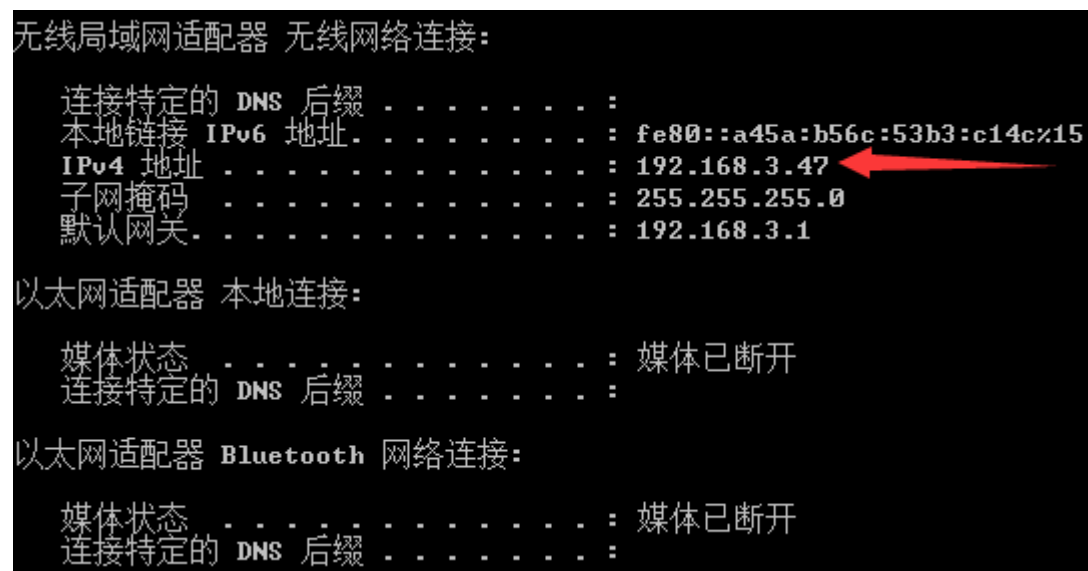

9.2 远程连接数据库-----当 mysql 服务器不在本机电脑上

其他代码不变，变的只是以下代码：

```
db = pymysql.connect(host='localhost' , user='root' , password='123456' ,  
port=3306 ,  
  
db='spiders' , charset='utf8')
```

步骤如下：

1、要知道要连接数据库主机 IP。怎么查看自己电脑主机 IP？打开 cmd 黑屏终端，输入 ipconfig，找到以下这个东西。



2、打开 navicat，点击 mysql 数据库（这个是系统数据库），打开 user 数据表，将 root 左边单元格的 localhost 替换为%。

Host	User	Select_priv	Insert_priv	Update_priv
%	root	Y	Y	Y
localhost	mysql.sys	N	N	N

3、再次打开 cmd 黑屏终端，先关闭数据库：net stop mysql57，再重新打开数据库 net start mysql57。

4、最后，将代码中 host='localhost'替换为 host='192.168.3.47'，运行即可

9.3 创建数据库表

```
Import pymysql
```

```
db = pymysql.connect(host='192.168.3.47' , user='root',  
password='123456' , port=3306 , db='spiders' , charset='utf8')
```

```
cursor = db.cursor()
```

```
# 检查表是否存在，如果存在删除
```

```
cursor.execute('drop table if exists students')
```

```
# 创建表
```

```
sql = 'create table students(id int auto_increment primary key not null,  
name varchar(10) not null,age int not null)'
```

```
cursor.execute(sql)
```

```
db.close()
```

注：以后用代码创建表的机会并不多，表一般都是我们提前创建好的。

9.4 数据库插入数据----- “增”

```
import pymysql
```

```
db = pymysql.connect(host='192.168.3.47' , user='root',  
password='123456' , port=3306 , db='spiders' , charset='utf8')
```

```
cursor = db.cursor()
```

```
# 插入数据
```

```
sql = 'insert into students(name,age) values(%s,%s)'
```

```
try:
```

```
    cursor.execute(sql,('孙悟空',100000))
```

```
    db.commit()
```

```
except:
```

```
print("插入失败")

db.rollback()

db.close()
```

注 1: 插入数据一定要用 **try...except...** 语句, 因为万一没插入成功, 其余代码都无法执行。

注 2: **import pymysql**, 此模块是默认开启 **mysql** 的事务功能的, 因此, 进行“增”、“删”、“改”的时候, 一定要使用 **db.commit()** 提交事务, 否则就看不见所插入的数据

9.5 数据库更新数据----- “改”

```
Import pymysql
```

```
db = pymysql.connect(host='192.168.3.47' , user='root',  
password='123456' , port=3306 , db='spiders' , charset='utf8')
```

```
cursor = db.cursor()
```

```
# 更新数据
```

```
sql = 'update students set age =%s where name=%s'
```

```
try:
```

```
    cursor.execute(sql,(30,"郭卫华"))
```

```
    db.commit()
```

```
except:
```

```
    print("插入失败")
```

```
    db.rollback()
```

```
db.close()
```

9.6 数据库删除操作----- “删”

注：针对表中数据而言

```
import pymysql
```

```
db = pymysql.connect(host='192.168.3.47' , user='root',  
password='123456' , port=3306 , db='spiders' , charset='utf8')
```

```
cursor = db.cursor()
```

```
# 删除数据
```

```
sql = 'delete from students where age=100000'
```

```
try:
```

```
    cursor.execute(sql)
```

```
    db.commit()
```

```
except:
```

```
    print("插入失败")
```

```
    db.rollback()
```

```
db.close()
```

9.7 数据库查询操作----- “查”

fetchone()

功能：获取下一个查询结果集，结果集是一个对象。

fetchall()

功能：接收全部返回的行。

rowcount：是一个只读属性，返回 **execute()**方法影响的行数。

```
import pymysql
```

```
db = pymysql.connect(host='192.168.3.47' , user='root',  
password='123456' , port=3306 , db='spiders' , charset='utf8')
```

```
cursor = db.cursor()
```

```
# 删除数据
```

```
sql = 'select * from students where age>60'
```

```
try:
```

```
    cursor.execute(sql)
```

```
    reslist = cursor.fetchall()
```

```
    for row in reslist:
```

```
        print("%d--%d" %(row[0],row[1],...row[n]))
```

```
except:
```

```
print("插入失败")
```

```
db.rollback()
```

```
db.close()
```


9.8 封装 mysql 语句为一个“类”

注：把下面类写在 studentsql 文件中

```
import pymysql

class StudentsSql():

    def __init__(self,host,user,port,dbname,charset):

        self.host = host

        self.user = user

        self.port = port

        self.dbname = dbname

        self.charset = charset


    def connet(self):

        self.db = pymysql.connect(self.host, self.user, self.port,

                                   self.dbname, self.charset)


    def close(self):

        self.cursor.close()

        self.db.close()


    def get_one(self,sql):

        res = None

        try:
```

```
        self.connect()

        self.cursor.execute(sql)

        res = self.cursor.fetchone()

        self.close()

    except:

        print("查询失败")

    return res
```

```
def get_all(self,sql):

    res = None

    try:

        self.connect()

        self.cursor.execute(sql)

        res = self.cursor.fetchall()

        self.close()

    except:

        print("查询失败")

    return res
```

```
def inset(self,sql):
```

```
    return self.__edit(sql)
```

```
def update(self,sql):
```

```

        return self.__edit(sql)

def delete(self,sql):

    return self.__edit(sql)


def __edit(self,sql):

    count = 0

    try:

        self.connect()

        count = self.cursor.execute(sql)

        self.db.commit()

        self.close()

    except:

        print("事务提交失败")

        self.db.rollback()

    return count

```

上述类封装成功，以后只需要调用即可

```
from studentsql import StudentsSql
```

```
s = StudentsSql("host='192.168.3.47' , user='root',
password='123456' , port=3306 , db='spiders' , charset='utf8'")
```

```
res = s.get_all('select * from students where age>60')
```

```
for row in res:
```

```
    print("%d--%d" %(row[0],row[1],...row[n]))
```