Mysql数据库

一,mysql数据库的常用命令

1.登录数据库

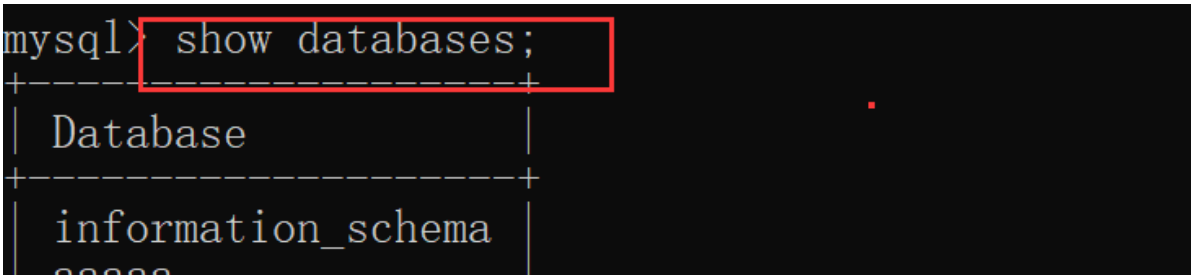
```
mysql -uroot -p+回车+密码+回车
```

2.退出数据库

```
quit
exit---是退出Dos界面
```

3.查看所有的数据库

```
show databases;
```



4.进入到指定的数据库

```
use 数据库名字;
```

5.查看所有的数据表

```
show tables;
```

```
mysql> use shop
Database changed
mysql> show tables;
+----------------+
| Tables_in_shop |
+----------------+
| ec_article     |
| ec_article_type|
| ec_item        |
| ec_order       |
```

6.创建数据库

- 直接创建一个数据库

```
create database gz202219;
```

```
mysql> create database gz202219;
Query OK, 1 row affected (0.00 sec)

mysql>
```

- 创建数据库是设置编码集

```
create database gz202219demo1 character set utf8;
```

```
mysql> create database gz202219demo1 character set utf8;
Query OK, 1 row affected (0.04 sec)

mysql>
```

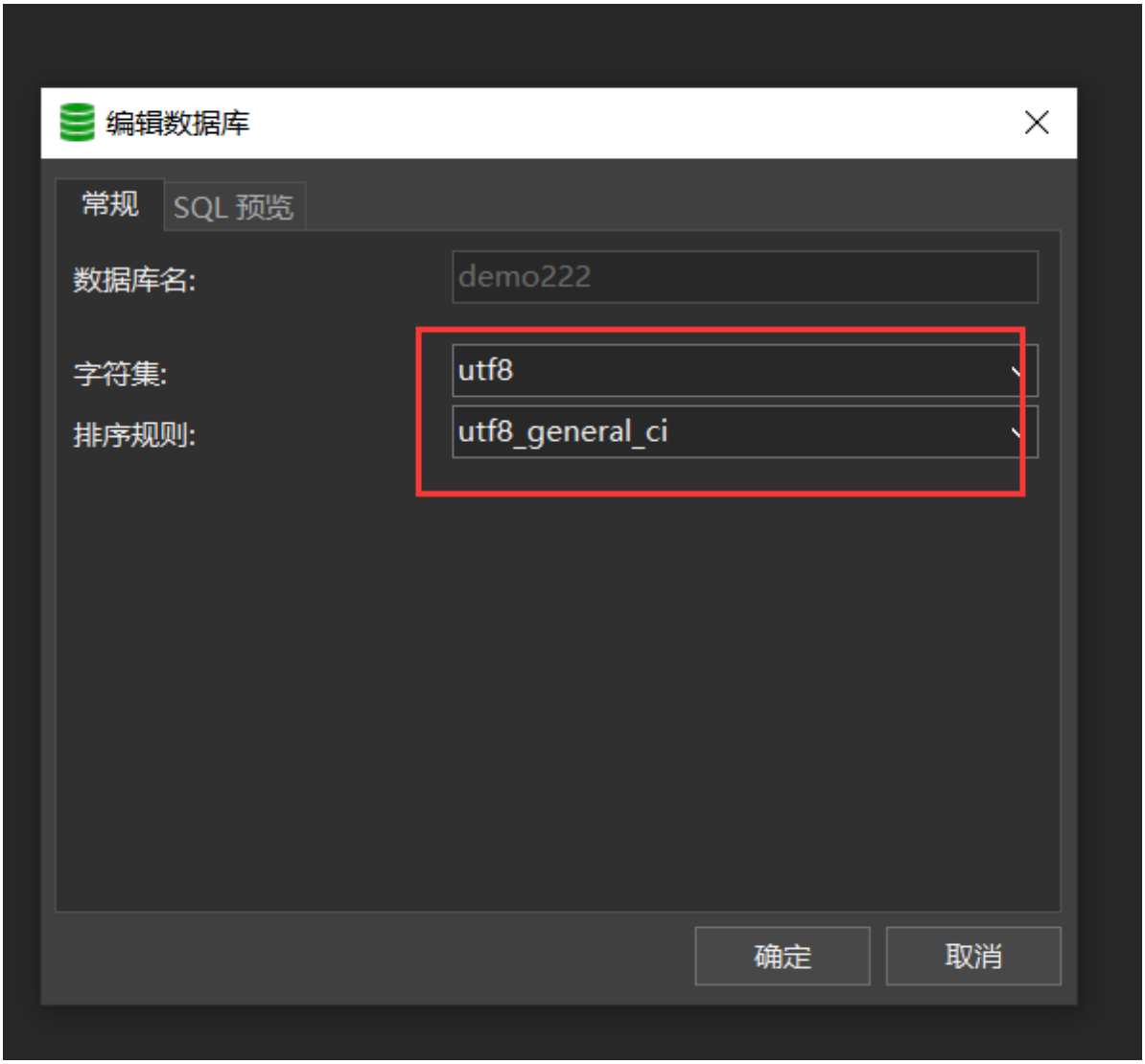
7.查询sql创建语句

```
show create database gz202219demo1;
```

```
mysql> show create database gz202219demo1;
+--------------+-----------------------------------------------------------------+
| Database     | Create Database                                                 |
+--------------+-----------------------------------------------------------------+
| gz202219demo1 | CREATE DATABASE `gz202219demo1` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+--------------+-----------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> show create database gz202219;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| gz202219 | CREATE DATABASE `gz202219` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

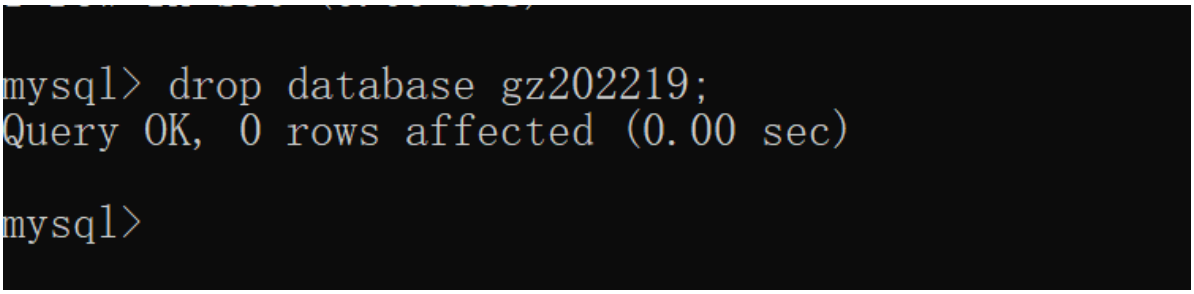
注意一般创建数据库都要设置编码集为utf-8，为了防止后续的中文乱码

注意：如果是用navicat创建数据库则，创建时要选择编码集如下



8.删除数据库

```
drop database 数据库名字;
如果没有这个数据库则会报错
```



- 第二种删除方式

```
drop database if exists 数据库名字;

特点：如果有数据库就执行删除操作，如果没有数据库就不执行任何操作
```

```
mysql> drop database gz202219;
ERROR 1008 (HY000): Can't drop database 'gz202219'; database doesn't exist
mysql> drop database if exists gz202219;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql>
```

二，mysql数据库的常用数据类型

如果我们要创建数据表，则必须要先知道每个字段的字段的数据类型是什么，这个数据类型和java的数据类型的意义一样，都是表示一个属性的类型

分类	数据类型	说明
数值类型	BIT(M) TINYINT [UNSIGNED] [ZEROFILL] BOOL，BOOLEAN SMALLINT [UNSIGNED] [ZEROFILL] INT [UNSIGNED] [ZEROFILL] BIGINT [UNSIGNED] [ZEROFILL] FLOAT[(*M*,*D*)] [UNSIGNED] [ZEROFILL] DOUBLE[(*M*,*D*)] [UNSIGNED] [ZEROFILL] Decimal(M,D)	位类型。M指定位数，默认值1，范围1-64 带符号的范围是-128到127。无符号0到255。 使用0或1表示真或假 2的16次方 2的32次方 2的64次方 M指定显示长度，d指定小数位数 表示比float精度更大的小数 定点数，存储货币等精度要求高的数据
文本、二进制类型	CHAR(size) char(20) VARCHAR(size) varchar(20) BLOB LONGBLOB TEXT(clob) LONGTEXT(longclob)	固定长度字符串 可变长度字符串 二进制数据 大文本
时间日期	DATE/DATETIME/TimeStamp	日期类型(YYYY-MM-DD) (YYYY-MM-DD HH:MM:SS)，TimeStamp表示时间戳，它可用于自动记录insert、update操作的时间

以上是mysql数据库的数据类型

常用的：

整数：int类型

小数：double类型

字符串：varchar

货币：Decimal

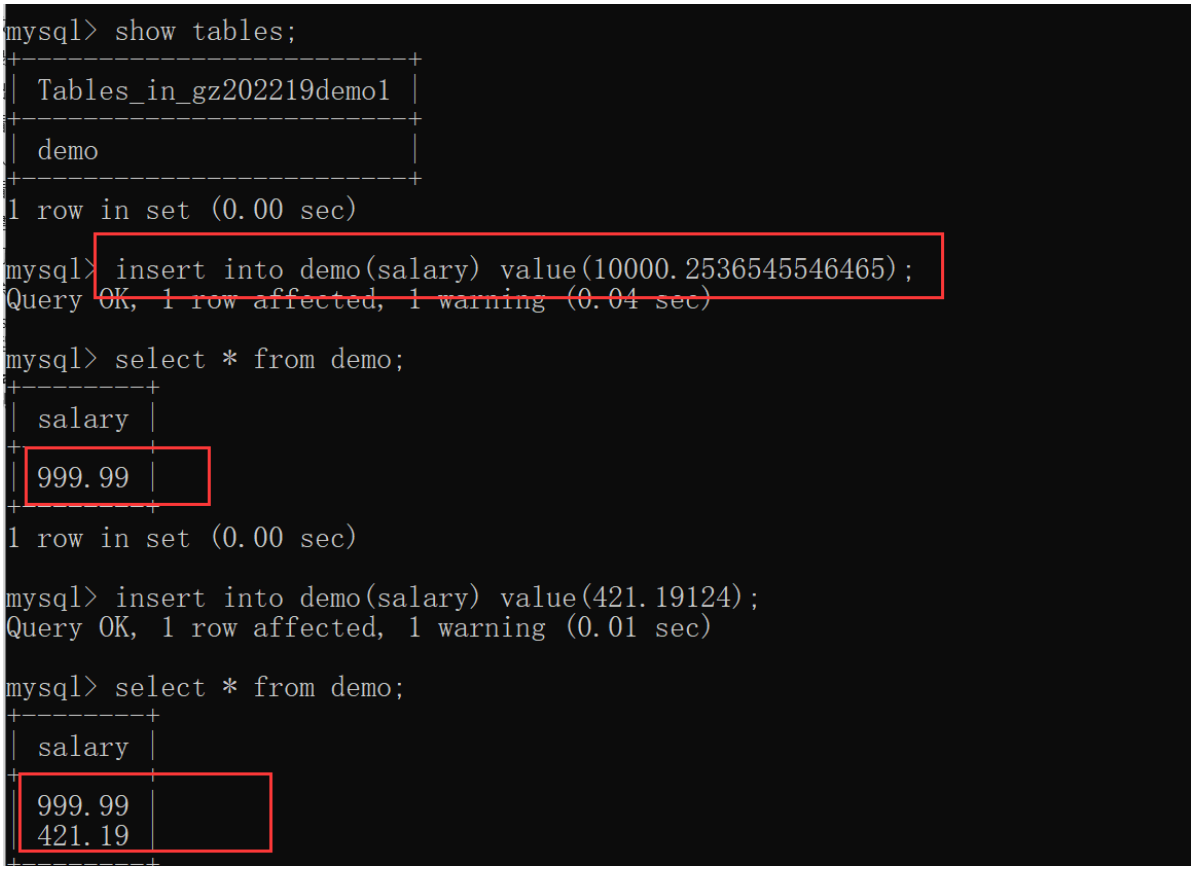
枚举类型：男/女

事件类型:date datetime

注意：sql语句的大小写不敏感，也就是一般不区分大小写。

在学习时要区别开：比如oracle就没有这么多类型，比如整数直接就一个类型：number

1.dicimal，float,double类型中的两个参数分别是代表什么



由上图可以看出：decimal中的两个参数：第一个：表示数据的总位数，包括小数位数，比如：（5,2）表示最大可以放5位的数包括小数位数，最大可以存放：999.99， 第二个参数，是小数的保留位数，默认是四舍五入

2.char类型和varchar类型的区别

char类型是定长的，比如：char(20)表示最大可以存放20个字符，如果只存放了1个字符，在内存中也会开辟20个字符大小的内存空间

varchar类是变长的，比如：varchar(20)表示最大可以存放20个字符,如果只存放了1个字符,则会在内存中也会开辟1个字符大小的内存空间

所以开发中一般使用varchar可变长度

三，标准的SQL语句类型

1.DDL：

DDL(Data Definition Language，数据定义语言)语句：主要由create、alter、drop和truncate四个关键字完成。包括创建创建数据库，数据表，视图，索引... alter注意是修改结构，不是修改数据表中的数据，drop删除的不是数据，是物理对象，库，视图，索引，表....

2.DML:

DML(Data Manipulation Language，数据操作语言)语句：主要由insert、update和delete三个关键字完成。
其实就是对数据表的添加，修改，删除，没有查询，主要是整数数据

3.DCL

```
DCL(Data Control Language，数据控制语言)语句：主要由grant和revoke两个关键字完成
```

4.DQL

```
select 查询语句 主要是针对数据表的查询
```

四，数据表的操作

1.创建一个数据表

```
create table test1(age int,name varchar(20),birthday date,sex
enum('男','女'),salary decimal(10,2));
```

```
mysql> create table test1(age int,name varchar(20),birthday date,sex enum('男','女'),salary decimal(10,2));
Query OK, 0 rows affected (0.01 sec)
```

2.添加数据

添加单条数据

```
insert into test1(age,name,birthday,sex,salary)values('15','张三','1987-12-
14','男','15243.27');
```

```
mysql> insert into test1(age,name,birthday,sex,salary)values('15','张三','1987-12-14','男','15243.27');
Query OK, 1 row affected (0.01 sec)

mysql>
```

添加多条数据

```
mysql> insert into test1(age,name,birthday,sex,salary)values('15','张三','1987-
12-14','男','15243.27'),
    -> ('13','李四','1987-12-14','男','15243.27'),
    -> ('15','王五','1987-12-14','男','15243.27'),
    -> ('15','段誉','1987-12-14','男','15243.27');
```

```
mysql> insert into test1(age,name,birthday,sex,salary)values('15','张三','1987-12-14','男','15243.27'),
    -> ('13','李四','1987-12-14','男','15243.27'),
    -> ('15','王五','1987-12-14','男','15243.27'),
    -> ('15','段誉','1987-12-14','男','15243.27');
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql>
```

3.查询所有数据

```
select * from test1;
```

```
mysql> select * from test1;
+------+------+------------+------+----------+
| age  | name | birthday   | sex  | salary   |
+------+------+------------+------+----------+
|   15 | 张三 | 1987-12-14 | 男   | 15243.27 |
|   15 | 张三 | 1987-12-14 | 男   | 15243.27 |
|   13 | 李四 | 1987-12-14 | 男   | 15243.27 |
|   15 | 王五 | 1987-12-14 | 男   | 15243.27 |
|   15 | 段誉 | 1987-12-14 | 男   | 15243.27 |
+------+------+------------+------+----------+
```

4.通過id查询指定的数据，条件查询

```
select * from test1 where name='张三';
```

```
mysql> select * from test1 where name='张三';
+------+------+------------+------+----------+
| age  | name | birthday   | sex  | salary   |
+------+------+------------+------+----------+
|   15 | 张三 | 1987-12-14 | 男   | 15243.27 |
|   15 | 张三 | 1987-12-14 | 男   | 15243.27 |
+------+------+------------+------+----------+
2 rows in set (0.00 sec)
```

5.修改数据

将名字是张三的人的薪水修改为10000

```
update test1 set salary='10000' where name='张三';
```

```
mysql> update test1 set salary='10000' where name='张三';
Query OK, 2 rows affected (0.01 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from test1 where name='张三';
+------+------+------------+------+----------+
| age  | name | birthday   | sex  | salary   |
+------+------+------------+------+----------+
|   15 | 张三 | 1987-12-14 | 男   | 10000.00 |
|   15 | 张三 | 1987-12-14 | 男   | 10000.00 |
+------+------+------------+------+----------+
2 rows in set (0.00 sec)
```

6.删除数据

```
delete from test1 where name='李四';
```

```
mysql> delete from test1 where name='李四';
Query OK, 1 row affected (0.01 sec)

mysql>
```

练习：创建一个员工表

字段：

1.工号

2.姓名

3.省份证

4.入职日期

5.性别

6.薪水

7.工作岗位

要求：

1.设计出表，添加10条数据

2.将张三的性别修改为女

3.将所有低于1000元的员工薪水在原来的基础上添加10000员

4.查询出薪资高于10000的所有员工

5.删除工号为E1003的员工

五.对表对象操作

1.查看表的结构

```
desc 表名字;
```

```
mysql> desc test1;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| age      | int(11)         | YES  |     | NULL    |       |
| name     | varchar(20)     | YES  |     | NULL    |       |
| birthday | date            | YES  |     | NULL    |       |
| sex      | enum('男','女') | YES  |     | NULL    |       |
| salary   | decimal(10,2)   | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
5 rows in set (0.05 sec)
```

2.查看数据表的创建语句

```
show create table test1;
```

```
mysql> show create table test1;
+-------+------------------------------------------------------------------
-------------------------------------------------------------------------
---+
| Table | Create Table
  |
+-------+------------------------------------------------------------------
-------------------------------------------------------------------------
---+
| test1 | CREATE TABLE `test1` (
  `age` int(11) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `sex` enum('男','女') DEFAULT NULL,
  `salary` decimal(10,2) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-------+------------------------------------------------------------------
-------------------------------------------------------------------------
```

3.修改表结构

- 在表的最后面添加一个字段，用来表示头像

```
alter table test1 add image varchar(30);
```

```
mysql> alter table test1 add image varchar(30);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> _
```

age	name	birthday	sex	salary	image
15	张三	1987-12-14	男	10000.00	NULL
15	张三	1987-12-14	男	10000.00	NULL
15	王五	1987-12-14	男	25243.27	NULL
15	段誉	1987-12-14	男	15243.27	NULL

- 修改字段image的名字，修改为images

```
alter table test1 change column image images varchar(30);
```

```
mysql> alter table test1 change column image images varchar(30);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc test1;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| age      | int(11)         | YES  |     | NULL    |       |
| name     | varchar(20)     | YES  |     | NULL    |       |
| birthday | date            | YES  |     | NULL    |       |
| sex      | enum('男','女') | YES  |     | NULL    |       |
| salary   | decimal(10,2)   | YES  |     | NULL    |       |
| images   | varchar(30)     | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
6 rows in set (0.05 sec)
```

- 删除images这一列字段

```
alter table test1 drop images;
```

-

```
mysql> alter table test1 drop images;
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc test1;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| age      | int(11)         | YES  |     | NULL    |       |
| name     | varchar(20)     | YES  |     | NULL    |       |
| birthday | date            | YES  |     | NULL    |       |
| sex      | enum('男','女') | YES  |     | NULL    |       |
| salary   | decimal(10,2)   | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
5 rows in set (0.05 sec)
```

- 修改数据表的名字

```
rename table test1 to user;
```

```
mysql> rename table test1 to user;
Query OK, 0 rows affected (0.04 sec)

mysql> show tables;
+-------------------------+
| Tables_in_gz202219demo1 |
+-------------------------+
| demo                    |
| user                    |
+-------------------------+
2 rows in set (0.00 sec)
```

- 修改表的字符集

```
alter table user character set utf8;
```

```
mysql> alter table user character set utf8;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show create table user;
+-------+--------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------
--+
| Table | Create Table

  |
+-------+--------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------
--+
| user  | CREATE TABLE `user` (
  `age` int(11) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `sex` enum('男','女') DEFAULT NULL,
  `salary` decimal(10,2) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-------+--------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------
--+
1 row in set (0.00 sec)
```

- 修改数据库的编码集

```
alter database demo101 character set utf8;
```

```
mysql> show create database demo101;
+----------+--------------------------------------------------------------------+
| Database | Create Database                                                    |
+----------+--------------------------------------------------------------------+
| demo101  | CREATE DATABASE `demo101` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+--------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> alter database demo101 character set utf8;
Query OK, 1 row affected (0.00 sec)

mysql> show create database demo101;
+----------+------------------------------------------------------------------+
| Database | Create Database                                                  |
+----------+------------------------------------------------------------------+
| demo101  | CREATE DATABASE `demo101` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+------------------------------------------------------------------+
1 row in set (0.00 sec)
```

六.表单约束

在开发中设计表时，一定要遵循，约束越严格越好，越严格则垃圾数据越少

数据库约束

- 所有的关系型数据库都支持对数据表使用约束，通过约束可以更好地保证数据表数据的完整性。约束是在表上强制执行的数据校验规则，约束主要用于保证数据库里数据的完整性。
- 大部分数据库都支持下面5种完整性约束：
  - NOT NULL：非空约束，指定某列不能为空。
  - UNIQUE：唯一约束，指定某列或者几列组合不能为空。
  - PRIMARY KEY：主键，指定该列的值唯一地标识该条记录。
  - FOREIGN KEY：外键，指定该行记录从属于主表中的一条记录，主要用于保证参照完整性。
  - 自动增长 auto_increment
  - 默认值，default
  - CHECK：检查，指定一个布尔表达式，用于指定对应列的值必须满足该表达式。(MySQL数据库不支持)

其实枚举也是约束的一种

1.工号--->主键（特点：不能为空，唯一，主键一个表只能有一个，主键自动创建索引）自动增长

2.姓名--->非空

3.省份证--->非空，唯一

4.入职日期-->非空

5.性别-->枚举"男"/“女”

6.薪水--->非空

7.工作岗位--->非空，默认值，普工

```
create table emp(
    number int primary key auto_increment,
    username varchar(50) not null,
    code varchar(18) not null unique,
    indate date not null,
    sex enum('男','女'),
    salary decimal(10,2) not null,
    job varchar(50) not null default '普工');
```

```
mysql> create table emp(
    ->     number int primary key auto_increment,
    ->     username varchar(50) not null,
    ->     code varchar(18) not null unique,
    ->     indate date not null,
    ->     sex enum('男','女'),
    ->     salary decimal(10,2) not null,
    ->     job varchar(50) not null default '普工');
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> desc emp;
+----------+-----------------+------+-----+---------+----------------+
| Field    | Type            | Null | Key | Default | Extra          |
+----------+-----------------+------+-----+---------+----------------+
| number   | int(11)         | NO   | PRI | NULL    | auto_increment |
| username | varchar(50)     | NO   |     | NULL    |                |
| code     | varchar(18)     | NO   | UNI | NULL    |                |
| indate   | date            | NO   |     | NULL    |                |
| sex      | enum('男','女') | YES  |     | NULL    |                |
| salary   | decimal(10,2)   | NO   |     | NULL    |                |
| job      | varchar(50)     | NO   |     | 普工    |                |
+----------+-----------------+------+-----+---------+----------------+
7 rows in set (0.05 sec)
```

```
mysql> insert into emp(username,code,indate,sex,salary,job)values ('张三','524132541445142415','2019-10-12','男','10201.356','工程师');
Query OK, 1 row affected, 1 warning (0.04 sec)

mysql> select * from emp;
+--------+----------+------------------+------------+------+----------+--------+
| number | username | code             | indate     | sex  | salary   | job    |
+--------+----------+------------------+------------+------+----------+--------+
|      1 | 张三     | 524132541245142415 | 2019-10-12 | 男   | 10201.36 | 工程师 |
|      2 | 张三     | 524132541445142415 | 2019-10-12 | 男   | 10201.36 | 工程师 |
+--------+----------+------------------+------------+------+----------+--------+
2 rows in set (0.00 sec)
```

- 将所有员工的薪水都改为5000

```
update emp set salary=5000;
```

- 将姓名为张三的员工薪水改为2000

```
update emp set salary=3000 where username='张三';
```

- 将员工李四的薪水修改为4000且job修改为：总经理

```
update emp set salary=3000,job='总经理' where username='李四';
```

- 将王五的薪水在原来的基础上添加10000

```
update emp set salary=salary+1000 where username='王五';
```

drop 和delete 和 truncate区别

1. drop 是直接销毁整个表，删除后这个表就不存在了

```
drop table aaa;
```

2. delete 是清空表中的所有数据，表还存在

```
delete from aaa;
```

3. truncate 是删除表并且创建一个和原理表结构一模一样的新表

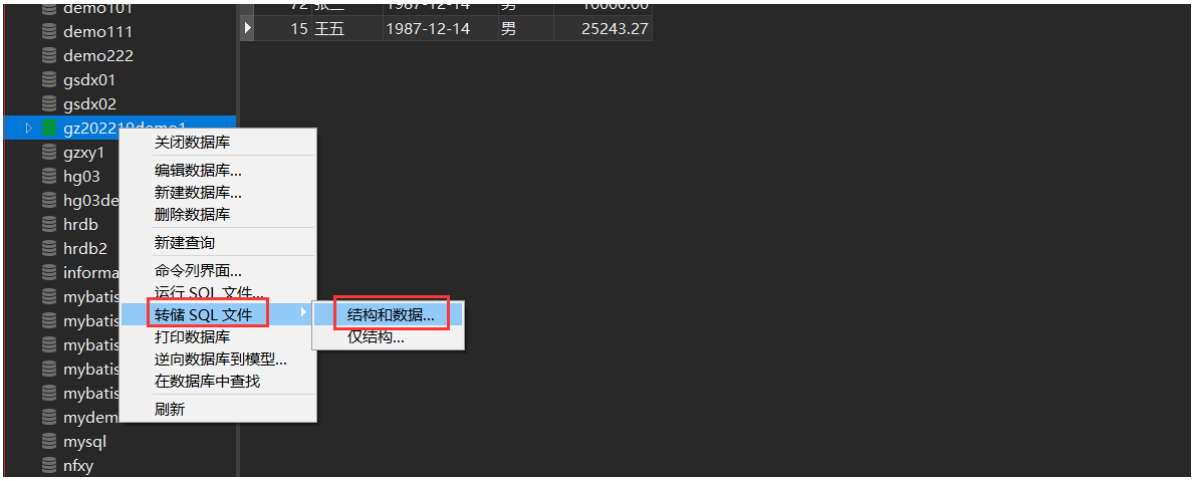
```
truncate table aaa;
```

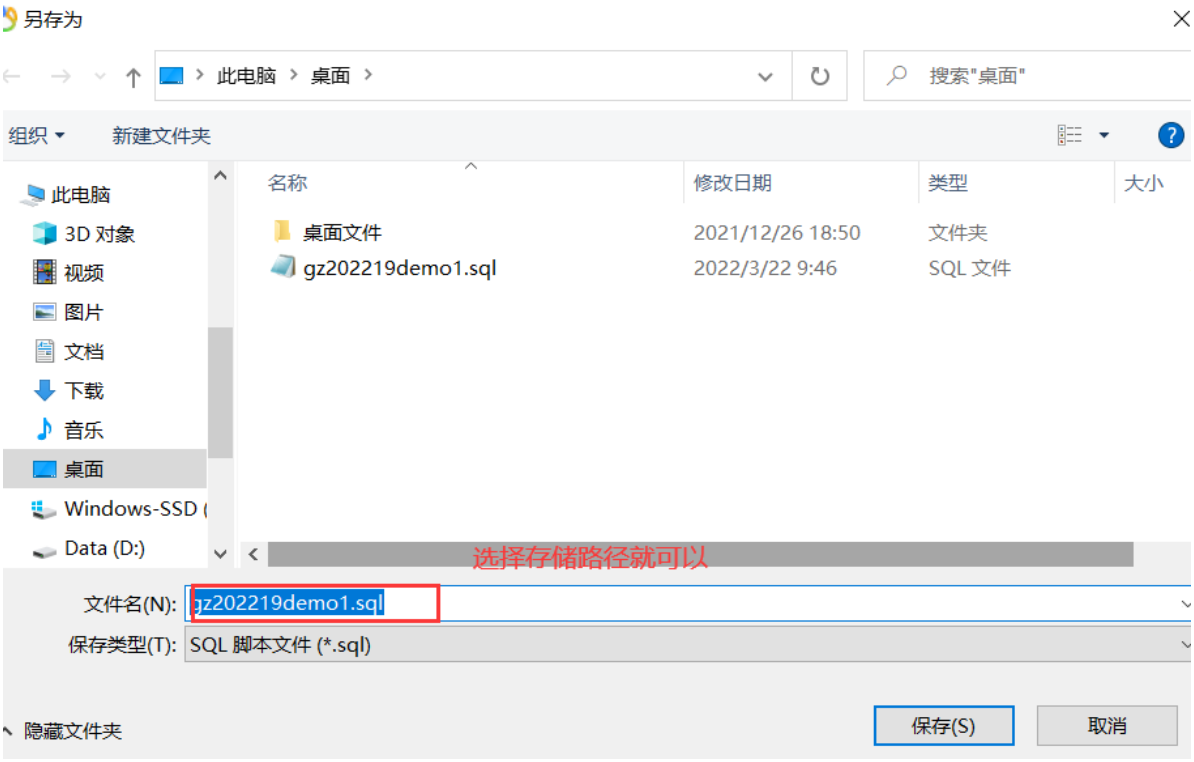
七.数据库备份和恢复

第一种:

直接用可视化工具备份

步骤:



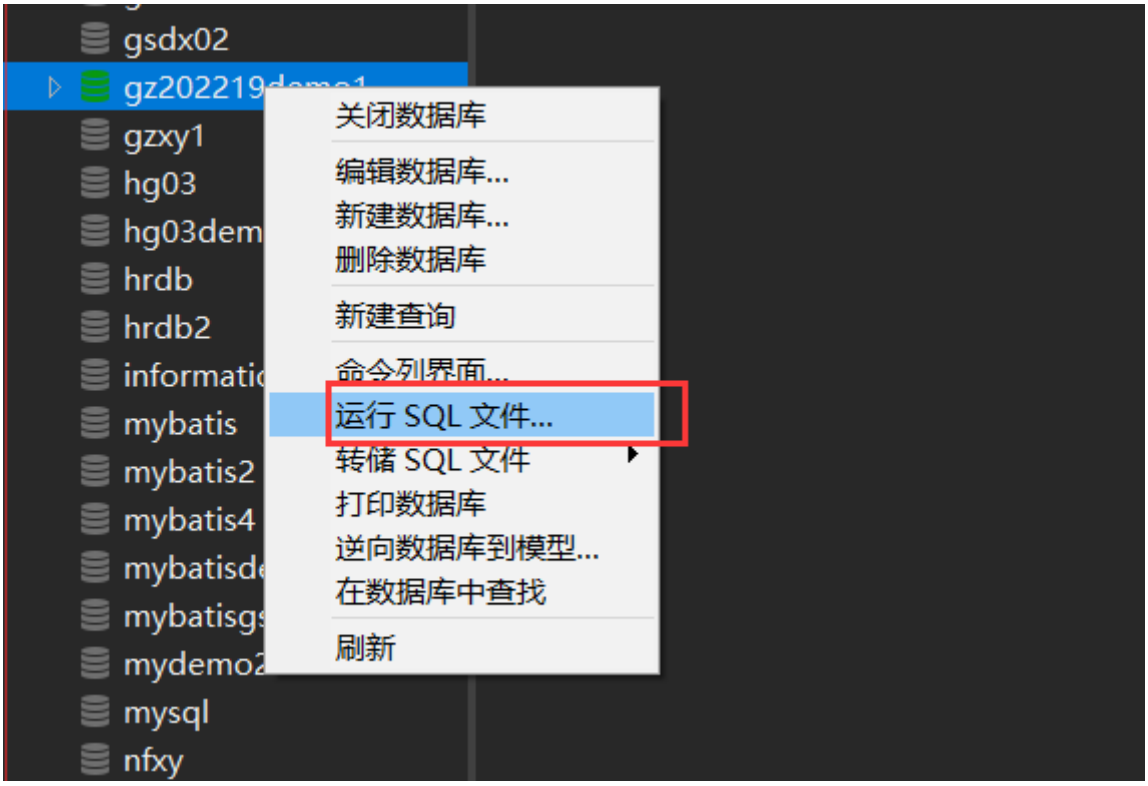


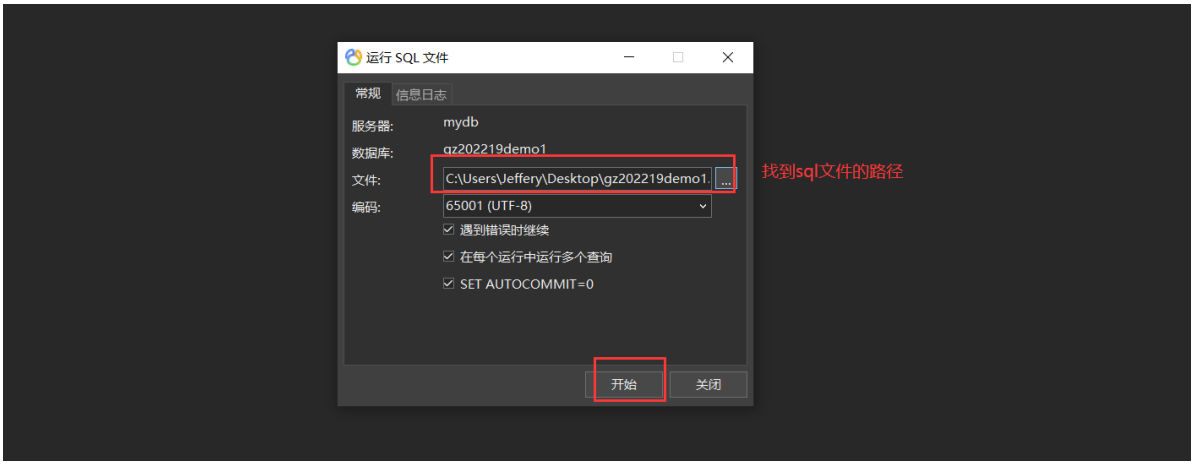
恢复数据，用工具：

演示：

先删除原来的数据库

drop database gz202219demo1;



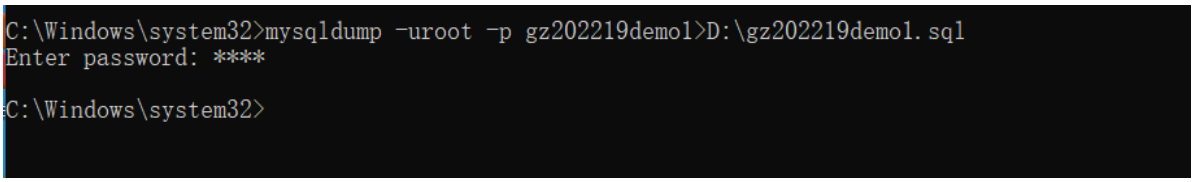


第二种:

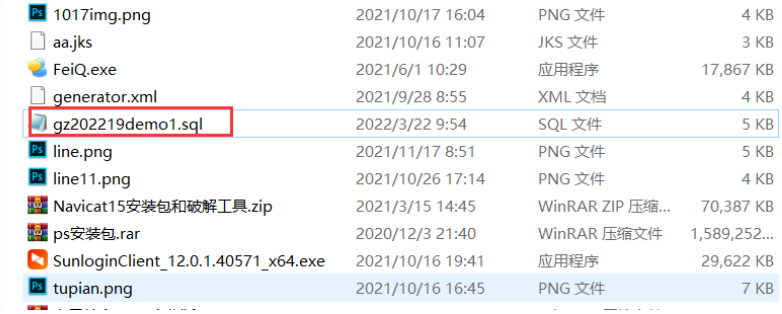
用dos命令完成

先完成备份

```
mysqldump -uroot -p gz202219demo1>D:\gz202219demo1.sql
```



备份成功后，在D盘中就会有一个sql文件

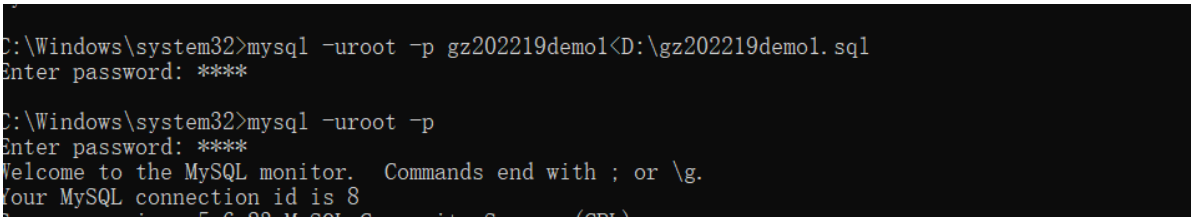


用dos命令恢复

先创建一个同名的数据库

恢复命令（1）

mysql -uroot -p gz202219demo1<D:\gz202219demo1.sql



也可以直接用执行sql文件命令恢复数据，注意这种方式是sql命令，要登录mysql数据才可以

```
mysql> source d:/gz202219demo1.sql;
```

要注意路径问题

八.单表查询

数据准备

```
create table stu(id int  primary key auto_increment,name varchar(20),chinese
int,english int,math int);


insert into stu(name,chinese,english,math)values('张三',45,50,65);


insert into stu(name,chinese,english,math)values('张三1',45,50,65);
insert into stu(name,chinese,english,math)values('李四2',46,51,66);
insert into stu(name,chinese,english,math)values('王五3',47,52,67);
insert into stu(name,chinese,english,math)values('马六4',48,53,68);
insert into stu(name,chinese,english,math)values('张三2',49,54,69);
insert into stu(name,chinese,english,math)values('李四3',50,55,70);
insert into stu(name,chinese,english,math)values('王五4',51,56,71);
insert into stu(name,chinese,english,math)values('马六5',52,57,72);
insert into stu(name,chinese,english,math)values('张三3',53,58,73);
insert into stu(name,chinese,english,math)values('李四4',54,59,74);
insert into stu(name,chinese,english,math)values('王五5',55,60,75);
insert into stu(name,chinese,english,math)values('马六6',56,61,76);
insert into stu(name,chinese,english,math)values('张三4',57,62,77);
insert into stu(name,chinese,english,math)values('李四5',58,63,78);
insert into stu(name,chinese,english,math)values('王五6',59,64,79);
insert into stu(name,chinese,english,math)values('马六7',60,65,80);
insert into stu(name,chinese,english,math)values('张三5',61,66,81);
insert into stu(name,chinese,english,math)values('李四6',62,67,82);
insert into stu(name,chinese,english,math)values('王五7',63,68,83);
insert into stu(name,chinese,english,math)values('马六8',64,69,84);
insert into stu(name,chinese,english,math)values('张三6',65,70,85);
insert into stu(name,chinese,english,math)values('李四7',66,71,86);
insert into stu(name,chinese,english,math)values('王五8',67,72,87);
insert into stu(name,chinese,english,math)values('马六9',68,73,88);
insert into stu(name,chinese,english,math)values('张三7',69,74,89);
insert into stu(name,chinese,english,math)values('李四8',70,75,90);
insert into stu(name,chinese,english,math)values('王五9',71,76,91);

select * from stu;
```

- 查询所有的学生

```
select * from stu;
```

- 查询出表中所有的姓名及英语成绩，只要两个字段

```
select name,english from stu;
```

信息 结果 1 剖析 状态

name	english
张三	50
张三1	50
李四2	51
王五3	52
马六4	53
张三2	54
李四3	55
王五4	56
马六5	57
张三3	58

- 过滤出表中重复的数据 distinct

```
select distinct chinese from stu;
```

信息 结果 1 剖析 状态

chinese
45
49
50
51
52
53
54
55
56
57
58
59

- 将所有学生的语文成绩都加10查询出(是查询不是修改，查询是不会改变原始数据)

```
select name,chinese+10 from stu;
```

信息 结果 1 剖析 状态

name	chinese+10
张三	55
张三1	55
李四2	55
王五3	55
马六4	55
张三2	59
李四3	60
王五4	61
马六5	62
张三3	63
李四4	64

- 统计出每个学生的总成绩

```
select name,(chinese+math+english) from stu;
```

```
44
45 select name,(chinese+math+english) from stu;
```

信息 结果 1 剖析 状态

name	(chinese+math+eng
张三	160
张三1	160
李四2	162
王五3	164
马六4	166
张三2	172
李四3	175
王五4	178
马六5	181

- 给总成绩取个别名

```
select name,(chinese+math+english) as 总成绩 from stu;
select name,(chinese+math+english)  总成绩 from stu;
```

```
select name,(chinese+math+english) as 总成绩 from stu;
```

信息 结果 1 剖析 状态

name	总成绩
张三	160
张三1	160
李四2	162
王五3	164
马六4	166
张三2	172
李四3	175
王五4	178
马六5	181
张三3	184
李四4	187

- 模糊查询
- % 模糊匹配字符，_表示匹配个数
- 查询出班级中所有姓张的学生

```
select * from stu where name  like '张%';
```

信息 结果 1 剖析 状态

id	name	chinese	english	math
1	张三	45	50	65
2	张三1	45	50	65
6	张三2	49	54	69
10	张三3	53	58	73
14	张三4	57	62	77
18	张三5	61	66	81
22	张三6	65	70	85
26	张三7	69	74	89

- 查询出班级中所有姓张的学生,且只有两个字的

```
select * from stu where name  like '张_';
```

信息 结果 1 剖析 状态

id	name	chinese	english	math
1	张三	45	50	65

- 查询出所有名字中有3的学生

```
select * from stu where name  like '%3%';
```

信息 结果 1 剖析 状态

id	name	chinese	english	math
4	王五3	45	52	67
7	李四3	50	55	70
10	张三3	53	58	73

- where子句（条件查询）
- 查询出id为10的学生信息

```
57
58 select * from stu where id = 10;
```

信息 | 结果 1 | 剖析 | 状态

id	name	chinese	english	math
10	张三3	53	58	73

- 查询出语文成绩大于60分的同学

```
60 select * from stu where chinese>60;
```

信息 | 结果 1 | 剖析 | 状态

id	name	chinese	english	math
18	张三5	61	66	81
19	李四6	62	67	82
20	王五7	63	68	83
21	马六8	64	69	84
22	张三6	65	70	85
23	李四7	66	71	86
24	王五8	67	72	87
25	马六9	68	73	88
26	张三7	69	74	89
27	李四8	70	75	90
28	王五9	71	76	91

- 查询出语文成绩在50和60直接的

```
62 select * from stu where chinese>50&&chinese<60;
```

信息 | 结果 1 | 剖析 | 状态

id	name	chinese	english	math
8	王五4	51	56	71
9	马六5	52	57	72
10	张三3	53	58	73
11	李四4	54	59	74
12	王五5	55	60	75
13	马六6	56	61	76
14	张三4	57	62	77
15	李四5	58	63	78
16	王五6	59	64	79

- and 和or
  - and 且
  - or 或
  - 查询出语文成绩大于60且姓王的同学
  -

```
66 select * from stu where chinese>60 and name like '王%'
```

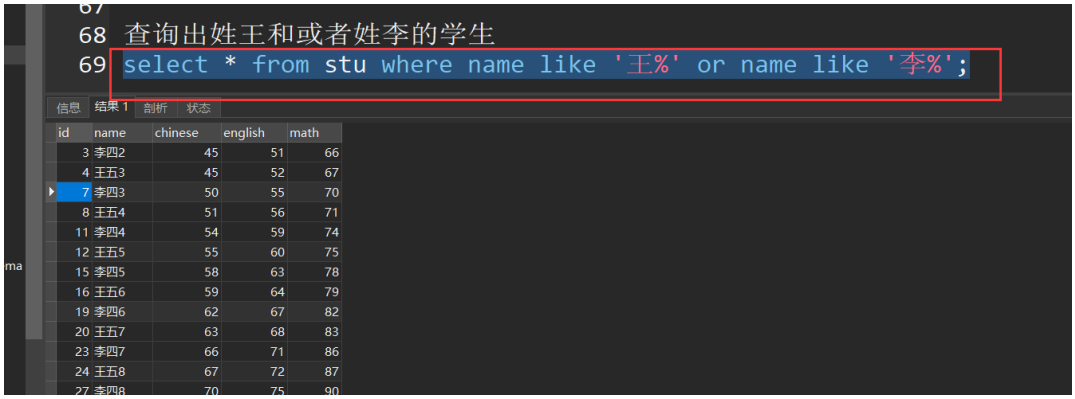
信息 | 结果 1 | 剖析 | 状态

id	name	chinese	english	math
20	王五7	63	68	83
24	王五8	67	72	87
28	王五9	71	76	91

  - 查询出姓王和或者姓李的学生

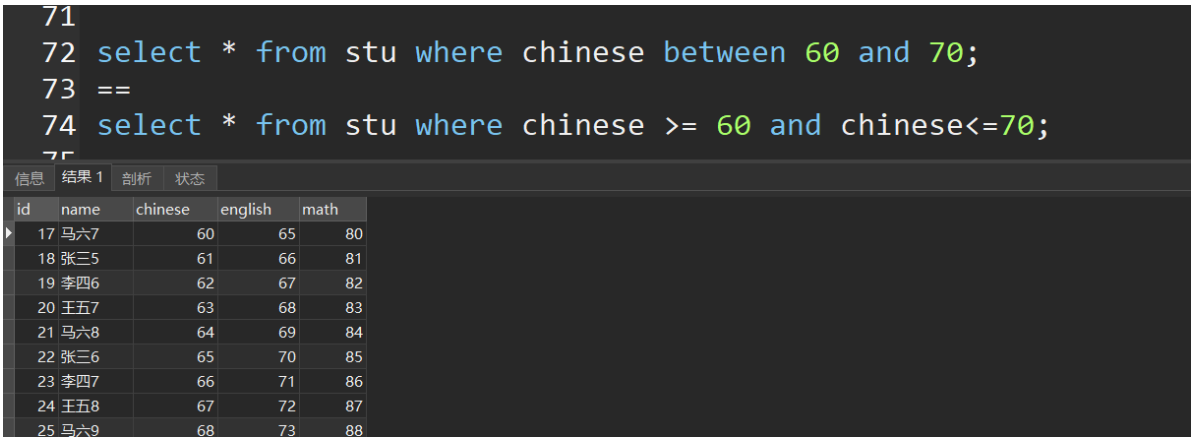
- 查询出姓王和或者姓李的学生

```
select * from stu where name like '王%' or name like '李%';
```



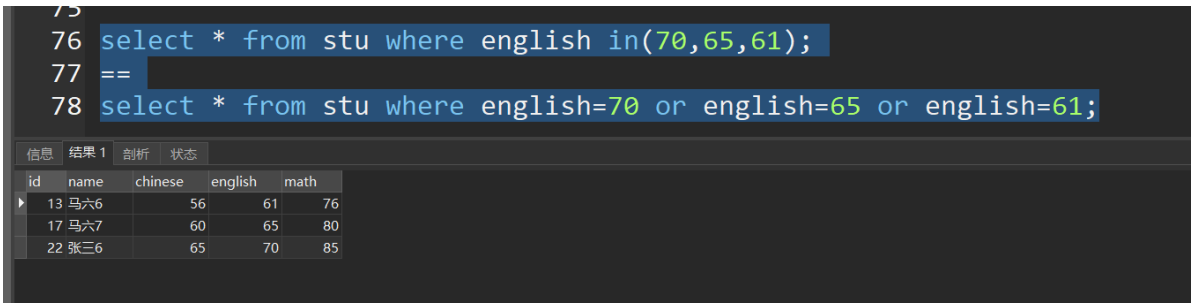
- in 和between and语句
- 查询出语文成绩在60-70之间的

```
select * from stu where chinese between 60 and 70;
==
select * from stu where chinese >= 60 and chinese<=70;
```



- 查询出英语成绩是61,65,48的学生

```
select * from stu where english in(70,65,61);
==
select * from stu where english=70 or english=65 or english=61;
```

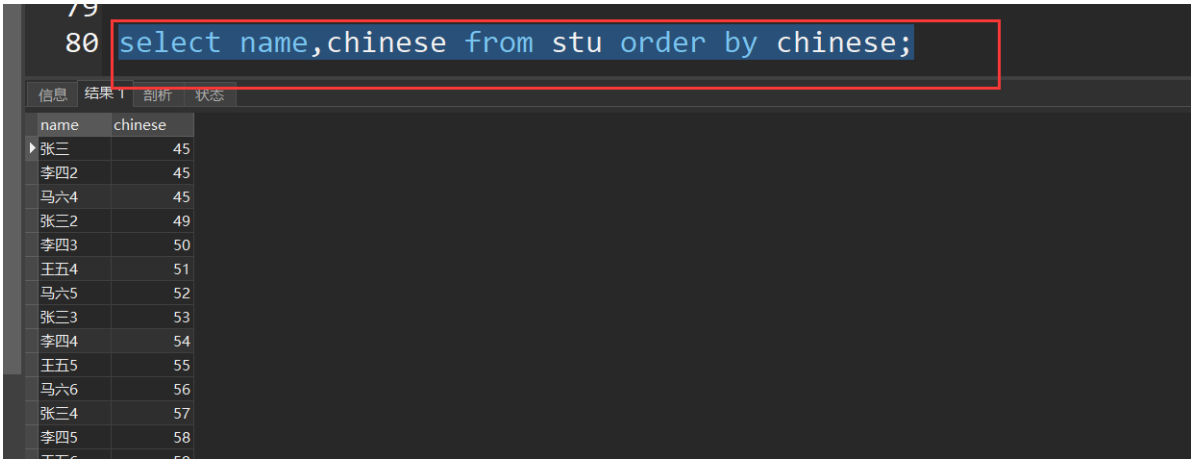


排序order by

升序：asc --从低到高

降序：desc---从高到低

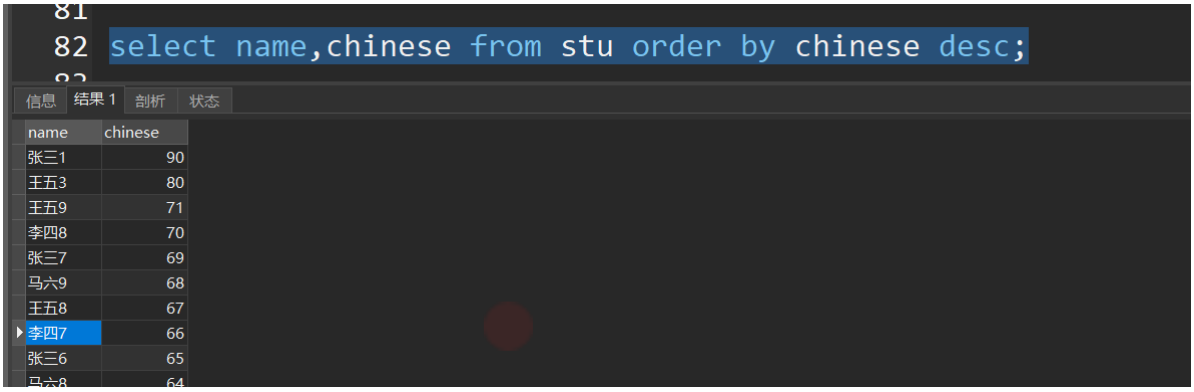
- 要求安装学生语文成绩从低到高排序，默认是升序




```
select name,chinese from stu order by chinese;
select name,chinese from stu order by chinese asc;
```

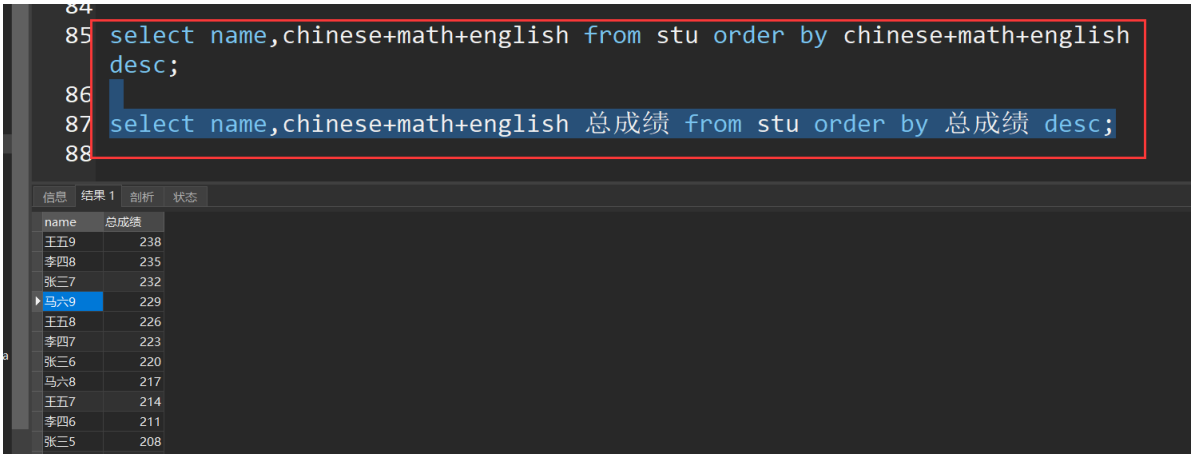
- 降序

```
select name,chinese from stu order by chinese desc;
```



- 总成绩降序

```
select name,chinese+math+english from stu order by chinese+math+english desc;


select name,chinese+math+english 总成绩 from stu order by 总成绩 desc;
```



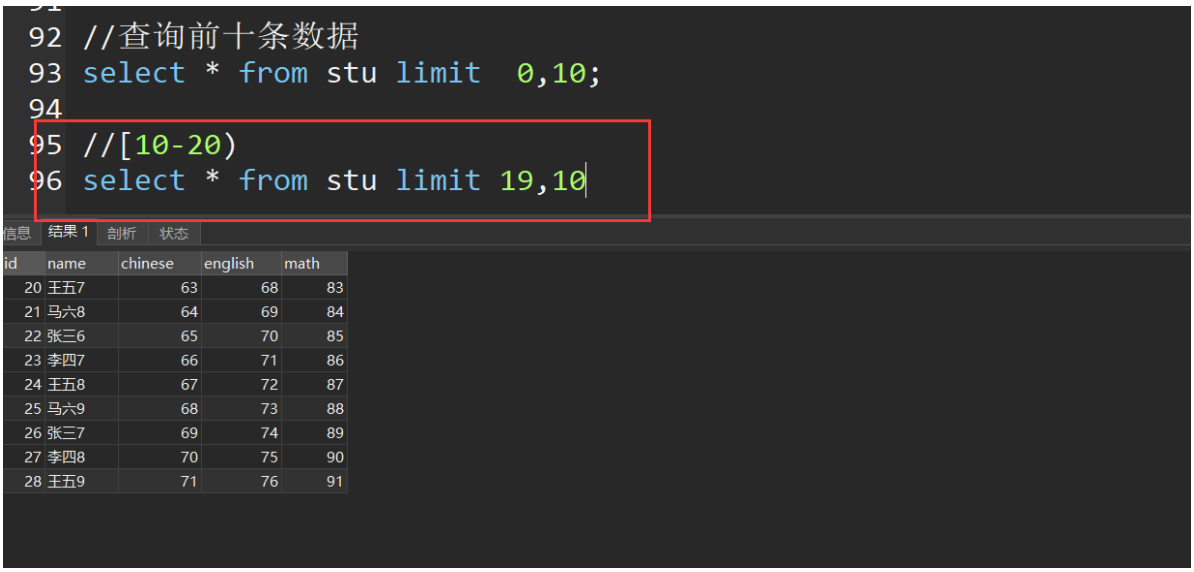
九.分页语句 limit

```
//查询前十条数据
select * from stu limit  0,10;
```

0--->表示查询的起始位置

10--->表示查询的条数，注意不是查询的结束位置

如果想用查询[10-20)条数据：select * from stu limit 9,10;



常用函数

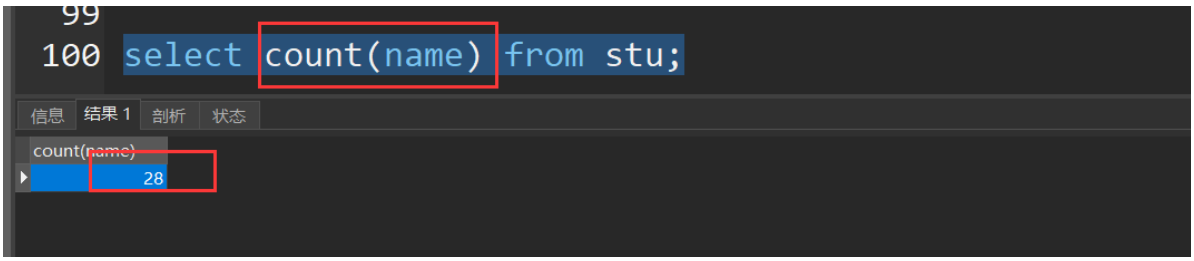
统计函数 count

- 查询班级中有多少个人

```
select count(*) from stu;
```

count函数不统计null

*如果所有的字段都为空就不统计



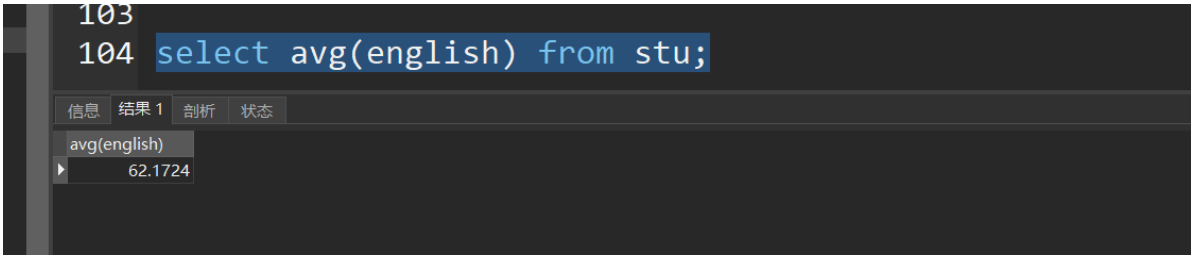
sum加总函数,平均值avg

- 查询出语文班级的总成绩

```
select sum(chinese) from stu;
```

- 查询班级中的英语平均分

```
select avg(english) from stu;
```



最大值最小值

总成绩第一名和倒数第一名

```
select max(english+math+chinese) from stu;

select min(english+math+chinese) from stu;
```

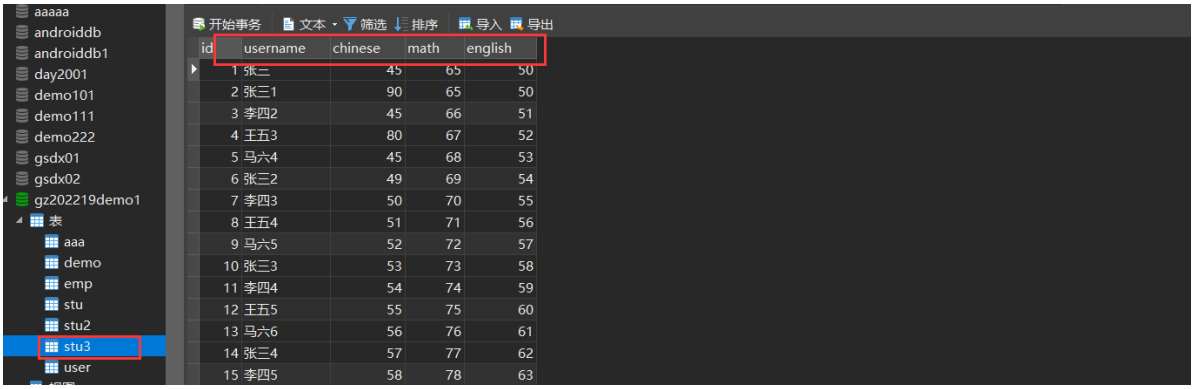
复制表

有时候需要将一张表的数据和结构复制到新表，可以通过CREATE TABLE ... [as] SELECT... 实现

如：将dept表数据复制到新表dept_2,dept_2字段名改为id,name,location

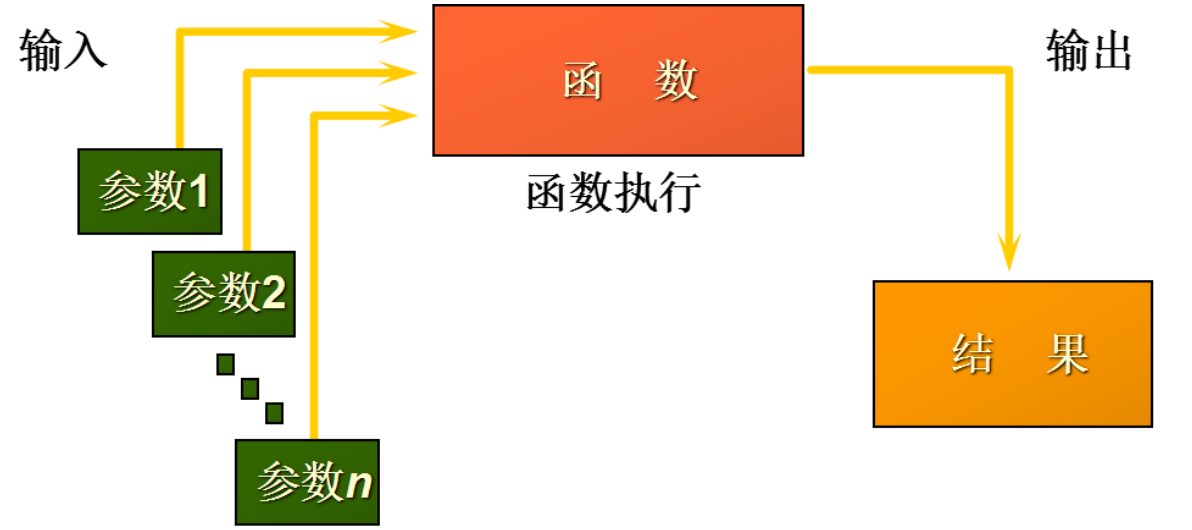
```
CREATE TABLE dept_2 SELECT
deptno id,
dname NAME,
loc location
FROM
    dept;
```

```
    create table stu3 select
    id id,
    name username,
    chinese chinese,
    math math,
    english english
        from stu;
```

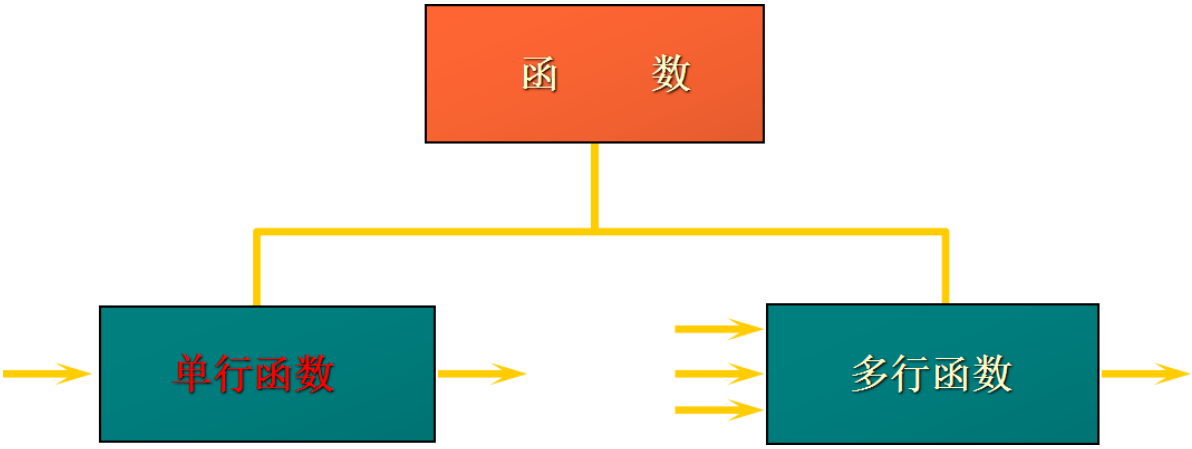


如：将dept表结构复制到新表dept_3，不复制数据

```
CREATE TABLE dept_3 SELECT
*
FROM
    dept
WHERE
    1=2;
```

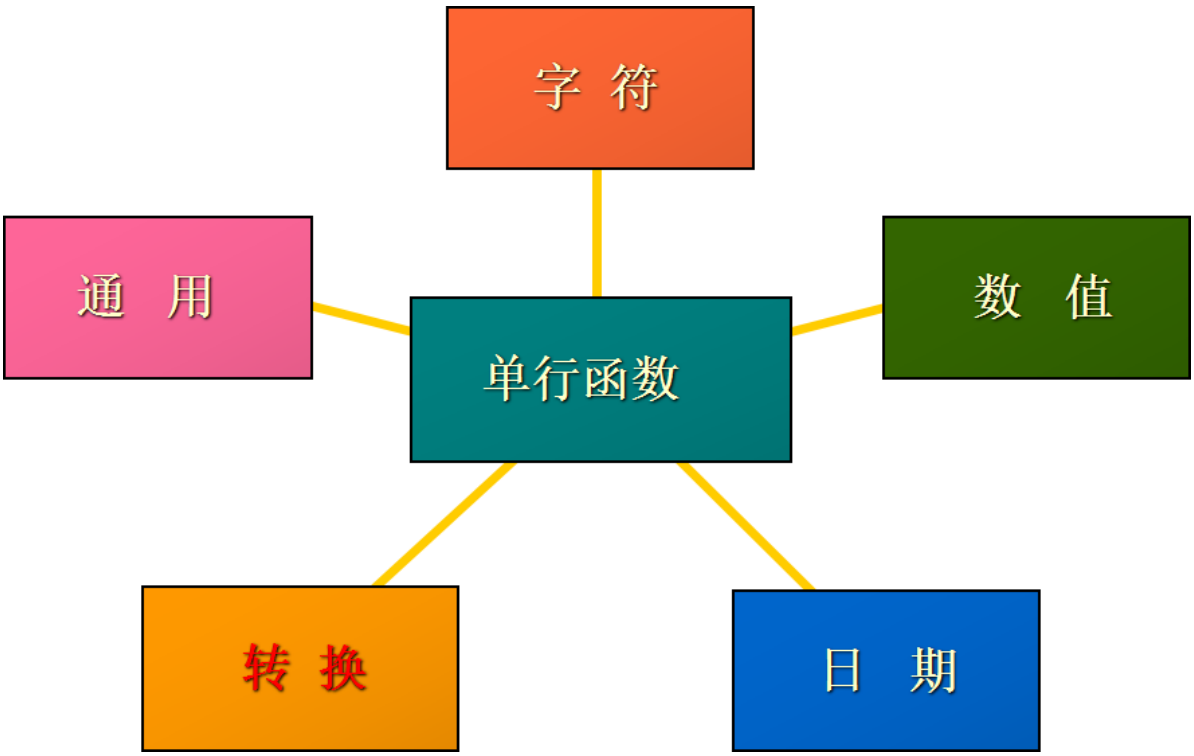


y = f(x1,...,x2)



单行函数

操作数据对象,接受参数返回一个结果,只对一行进行变换,每行返回一个结果



length

```
#会列出ename和ename的字符串长度
select ename,length(ename) from emp;
```

concat

concat是字符串连接函数，语法是：

```
concat(char1, char2,...)
```

用于返回两个字符串连接后的结果，两个参数char1、char2是要连接的两个字符串。

```
# 查询所有的员工姓名，按'员工的姓名是xxx'格式显示
select concat('员工的姓名是',ename) ename from emp;
```

连接三个字符串

```
# 连接emp表中的ename列和sal列，显示格式为:xxx的工资是xxx
SELECT CONCAT(ename, '的工资是', sal) FROM emp;
```

「课堂练习」

查询出所有的员工工号，并连接ename和job，显示格式为xxx work is xxx，并给该连接显示结果取别名为detail

```
SELECT empno,CONCAT(ename, ' work is ', job) detail FROM emp;
```

upper、lower

这两个函数全部是英文的大小写转换函数，用来转换字符的大小写：

- `UPPER(char)`用于将字符转换为大写形式

```
        将name字段的值转换成大写输出
        select upper(name) from stu;
```

- `LOWER(char)`用于将字符转换为小写形式

一般用来查询数据表中不确定大小写的情况。

```
#查询员工Jones的信息
select * from emp where ename='Jones';
```

将字符数据转换为小写或大写(如果不知道姓名的大小写形式 , 可以使用 lower/upper函数 , 忽略大小写)

```
select * from emp where lower(ename)='jones';

select * from emp where upper(ename)='JONES';
```

trim、ltrim、rtrim

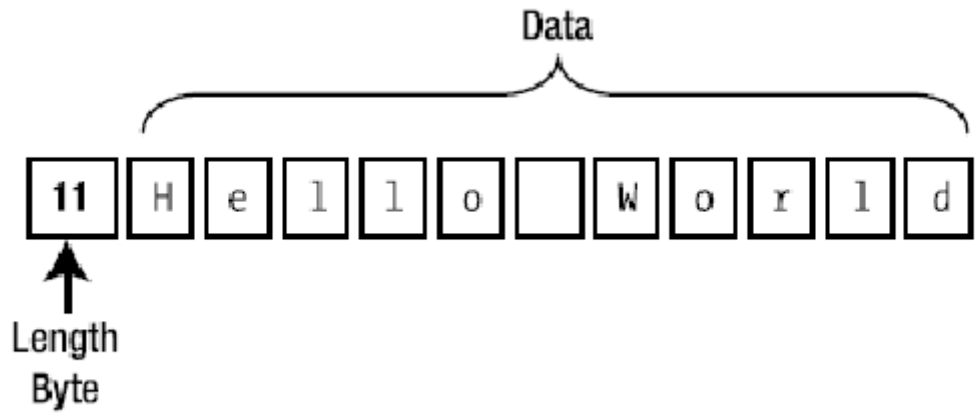
```
#dual是临时表（虚拟的表） 去除前后的字符'甜'
SELECT trim('甜' from '甜蜜生活是那么的甜') from dual;

#去掉前后空格
SELECT trim(" he llo ") from dual;

#去掉左空格
SELECT ltrim(" he llo ") from dual;

#去掉右空格
SELECT rtrim(" he llo ") from dual;
```

案例：测试char和varchar的区别

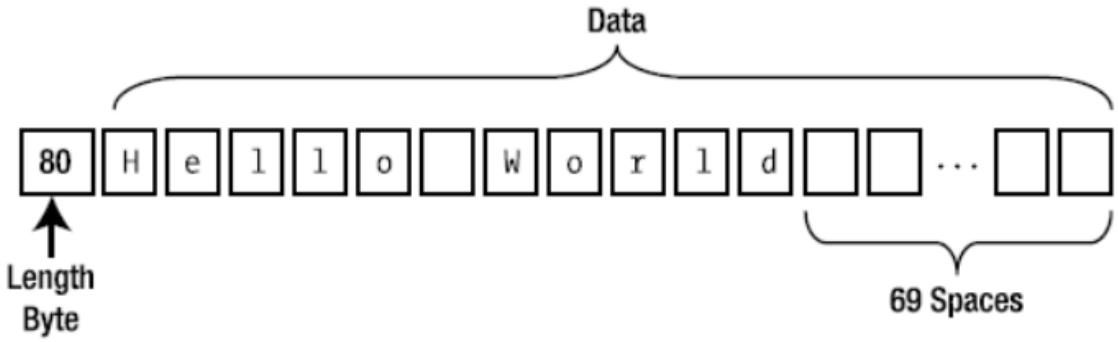
测试mysql常见数据类型的时候提到char和varchar的特点:

- char(n) 和 varchar(n) 中括号中 n 代表字符的个数，并不代表字节个数
- char类型是固定长度，存储数据不够长度时会用空格填充。varchar是不固定长度，不会用空格填充。
- char类型取值时会自动去掉自动填充的空格，因此，用length()方法时不会是括号中的长度。

如果一个包含"hello world"的varchar(80), 存储结构如图所示:



如果在char(80)中存储同样的信息，存储结构如图所示：



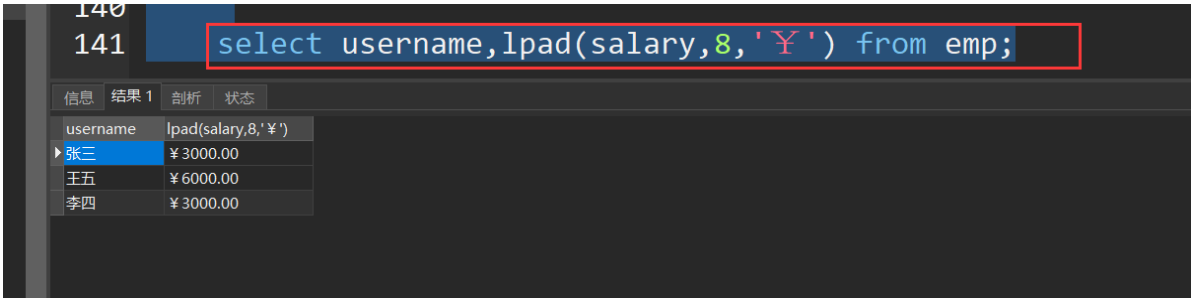

```
#测试char和varchar区别
#创建一个表，这个表里面包含两个字段，d_char和d_varchar，设定初始的字符长度都为4
CREATE TABLE data_type (
    d_char CHAR ( 4 ),
    d_varchar VARCHAR ( 4 )
)

#查看表结构
desc data_type;

#插入两条记录，每条记录都是'ab  '，注意，ab后面有2个空格
INSERT INTO data_type
VALUES
    ( 'ab  ', 'ab  ' );

#char  会自动使用trim 去除 空格  varchar则不会
SELECT
    LENGTH( d_char ),       #结果2
    LENGTH( d_varchar )     #结果4
FROM
    data_type;

#通过concat 进行字符连接，给每条记录的左右分别添加小括号  ，d_char的ab后面的空格被取消掉了，
而d_varchar后面的空格还依旧存在
SELECT
    CONCAT( '(', d_char, ')' ),
    CONCAT( '(', d_varchar, ')' )
FROM
    data_type;
```

LPAD、RPAD

PAD意即补丁，LPAD和RPAD两个函数都叫做补位函数，LPAD表示LEFT PAD，在左边打补丁，RPAD表示RIGHT PAD，在右边打补丁。

```
SELECT LPAD('hi',6,'abc')

SELECT RPAD('hi',6,'abc')

SELECT LPAD('1000',5,'￥')
#例：显示emp的sal，如果不满7位的，用￥补齐7位
select ename,lpad(sal,7,'￥') from emp
```

案例：



substr

substr表示在一个字符串中截取子串：

```
substr(char, m, n)

substr(char, m)
```

用于返回char中从m位开始取n个字符的子串，字符串的首位计数从1开始。

参数作用：

如果m取正数，则从首字符开始，如果m取负数，则从尾部开始

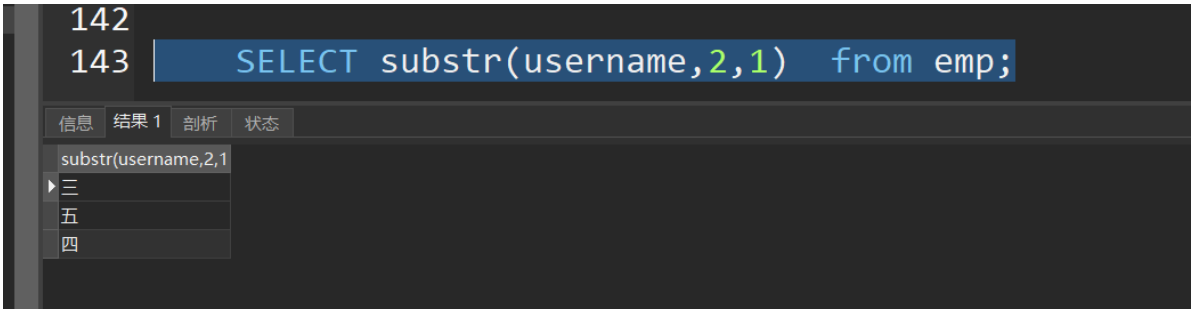
如果没有设置n，或者n的长度超过了char的长度，则取到字符串末尾为止

```
SELECT substr('i like oracle',3,4) ;
```

	SUBSTR('ILIKEORACLE',3,4)
1	like

```
SELECT substr('i like oracle',-11,4) ;
```

	SUBSTR('ILIKEORACLE',-11,4)
1	like



常用函数之数值操作

数值操作函数

数值函数指参数是数值类型的函数。常用的有`round`、`abs`、`mod`、`ceil`和`floor`。

round

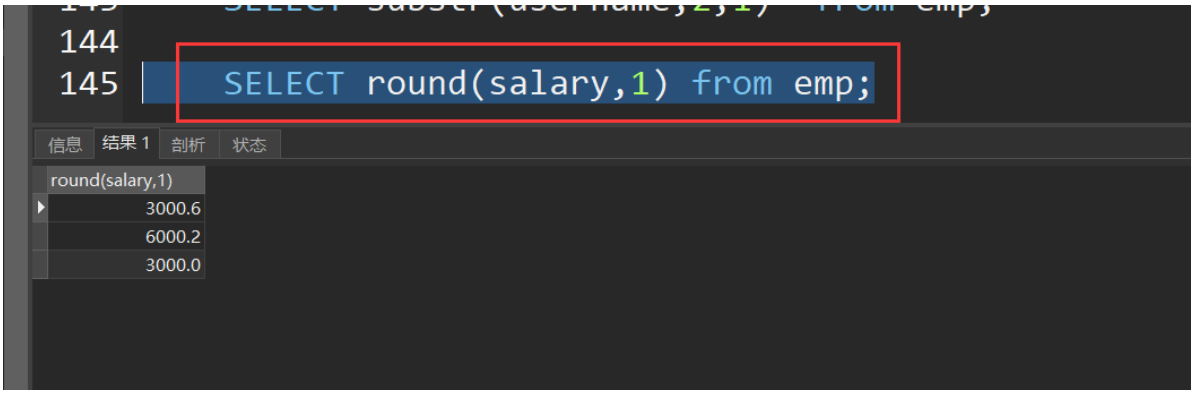
round(n, m) 用于将参数n按照m的数字要求四舍五入。

参数中的n可以是任何数字，指要被处理的数字

- m必须是整数
- m取正数则四舍五入到小数点后第m位
- m取负数，则四舍五入到小数点前m位
- m取0值则四舍五入到整数位
- m缺省，默认值是0

例：

```
SELECT round(1234.5678, 2) #1234.57

SELECT round(1234.5678, -1) #1230

SELECT round(1234.5678, 0) #1235

SELECT round(1234.5678) #1235
```



abs

abs(n)的功能是处理数字n得到正整数

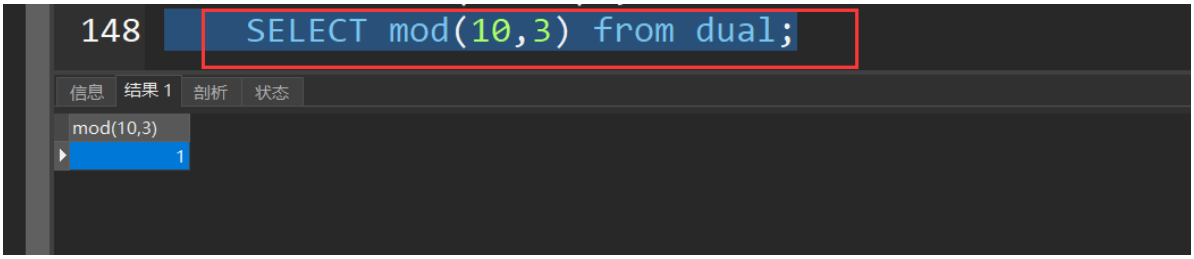
正数的绝对值是它本身，负数的绝对值是正数

```
SELECT abs(-2) ; #2
```

mod

mod(m, n)是取模函数，返回m除以n后的余数，如果n为0则直接返回m。

```
SELECT mod(5,3) from dual;
SELECT mod(5,-3) from dual;
SELECT mod(-5,3) from dual;
SELECT mod(-5,-3) from dual;
```



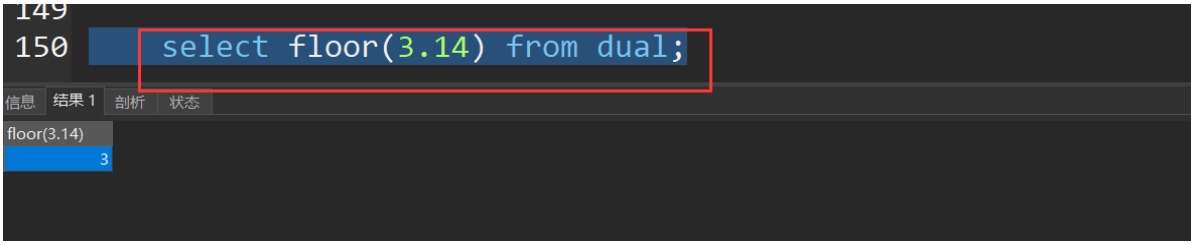
ceil和floor

ceil向上取整，

floor向下取整

ceil(n)、floor(n)这两个函数顾名思义，一个是天花板，就是取大于或等于n的最小整数值，一个是地板，就是取小于或等于n的最大整数值。比如数字n= 4.5，那么它的ceil是5.0，它的floor是4.0

```
select ceil(3.14) from dual;
select floor(3.14) from dual;

select ceil(-3.14) from dual;
select floor(-3.14) from dual;
```



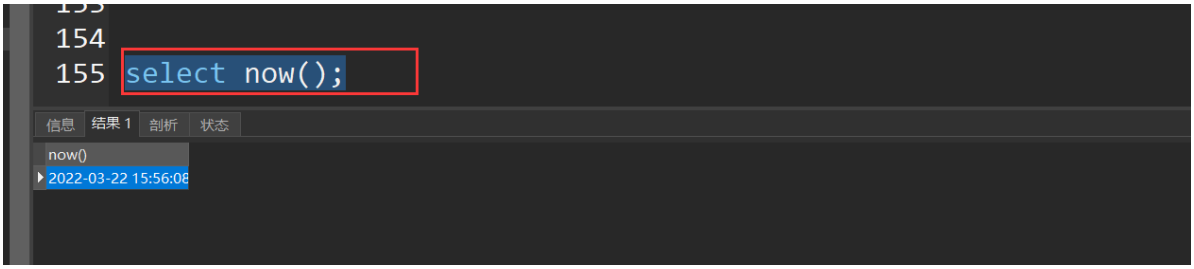
常用函数之日期处理

获得当前日期时间 函数

```
select now();
```

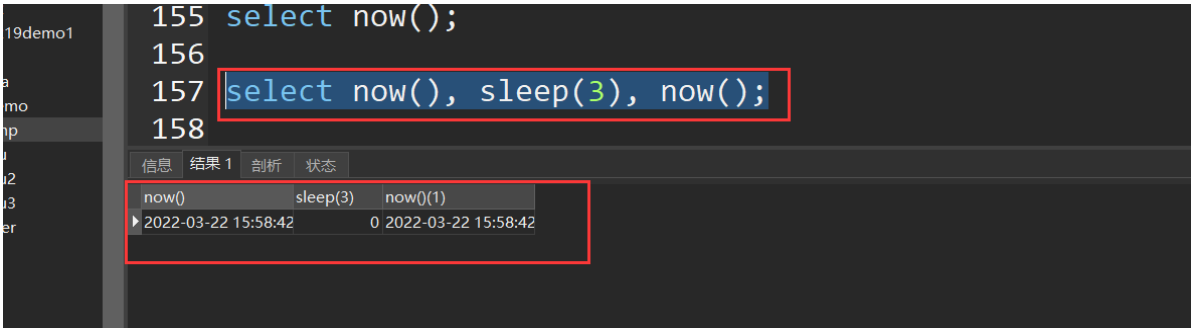
比如入职设计直接写成当前时间

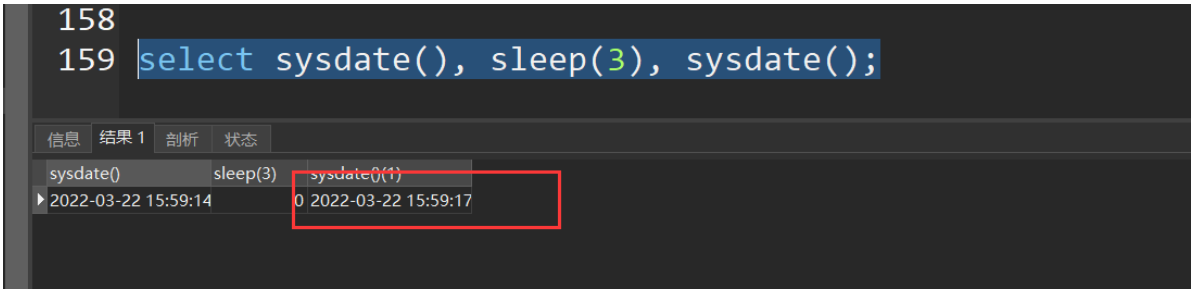
```
INSERT INTO `gz202219demo1`.`emp`( `username`, `code`, `indate`, `sex`, `salary`, `job`) VALUES ( '李四', '524132541445942465', now(), '男', 3000.01, '总经理');
```



sysdate() 日期时间函数跟 now() 类似，不同之处在于：now() 在执行开始时值就得到了， sysdate() 在函数执行时动态得到值。看下面的例子：

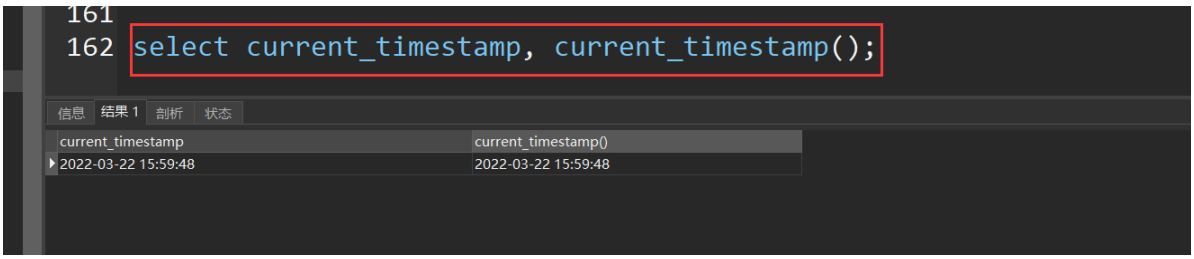
```
select now(), sleep(3), now();

select sysdate(), sleep(3), sysdate();
```





获得当前时间戳函数

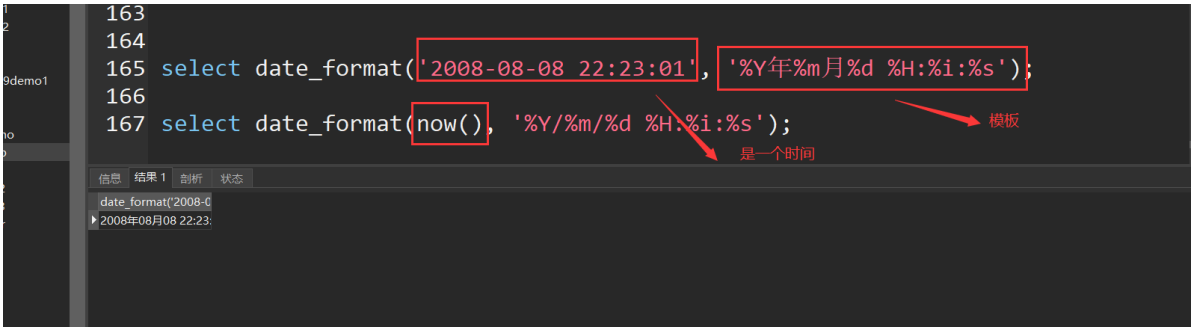
```
select current_timestamp, current_timestamp();
```



日期转换函数、时间转换函数

日期转换为字符串函数:date_format

```
select date_format('2008-08-08 22:23:01', '%Y-%m-%d %H:%i:%s');
select date_format(now(), '%Y-%m-%d %H:%i:%s');
```



字符串转换为日期函数：str_to_date(str, format)

```
select str_to_date('08/09/2008', '%m/%d/%Y'); # 2008-08-09
select str_to_date('08/09/08' , '%m/%d/%y'); # 2008-08-09
select str_to_date('08.09.2008', '%m.%d.%Y'); # 2008-08-09
select str_to_date('08:09:30', '%H:%i:%s'); # 08:09:30
select str_to_date('08.09.2008 08:09:30', '%m.%d.%Y %H:%i:%s'); # 2008-08-09
08:09:30
```

可以看到，str_to_date(str,format) 转换函数，可以把一些杂乱无章的字符串转换为日期格式。另外，它也可以转换为时间。“format” 可以参看 MySQL 手册。

为日期增加一个时间间隔date_add

例如：

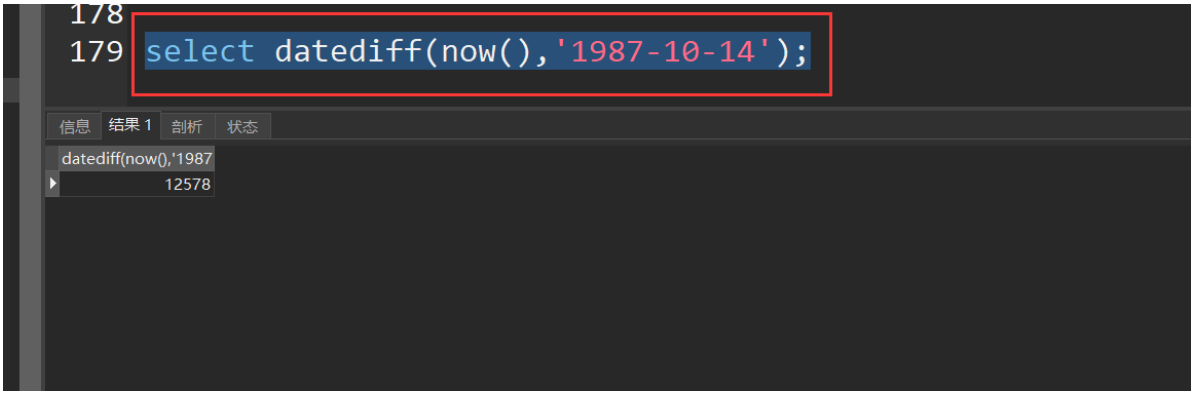
```
set @dt = now();

select date_add(@dt, interval 1 day); # add 1 day
select date_add(@dt, interval 1 hour); # add 1 hour
select date_add(@dt, interval 1 minute); # ...
select date_add(@dt, interval 1 second);
select date_add(@dt, interval 1 week);
select date_add(@dt, interval 1 month);
select date_add(@dt, interval 1 quarter);
select date_add(@dt, interval 1 year);

select date_add(@dt, interval -1 day); # sub 1 day
```

日期、时间相减函数datediff

```
MySQL datediff(date1,date2)：两个日期相减 date1 - date2，返回天数。
```

例如：

```
select datediff('2008-08-08', '2008-08-01'); # 7
select datediff('2008-08-01', '2008-08-08'); # -7
```



练习：

要求设计一个员工类，字段和约束自己设计？

字段：

1. 工号
2. 姓名
3. 出生日期
4. 入职日期
5. 薪水
6. 性别

1.

   数据准备：

```
create table test1(
number int primary key auto_increment,
name varchar(50) not null,
birthday date not null,
inday date not null,
sex enum('男','女'),
salary decimal(10,2) not null
);

insert into test1(name,birthday,inday,sex,salary)values('老八','1970-12-
1','2000-2-2','男','10000'),
('张三','1999-2-1','2022-2-23','男','1000'),
('李四','1988-1-1','2017-1-2','男','8000'),
('王五','1990-3-1','2017-3-2','男','8000'),
('老刘','1987-4-1','2008-2-2','男','8000'),
('吴琦','1986-5-1','2008-2-2','男','10000'),
('路霸','1990-6-1','2015-2-2','男','3000'),
('高斯','1991-7-1','2016-2-2','男','5000'),
('盖亚','1992-8-1','2017-2-2','男','10000'),
('拉克丝','1993-9-1','2018-2-2','女','10000');


insert into test1(name,birthday,inday,sex,salary)values ('张三','1995-12-
11','2018-10-2','男','8888.55'),
('李四','1996-12-11','2017-10-2','男','8854.55'),
('梦姑','1998-10-11','2016-10-2','女','9888.55'),
('小张','1982-6-11','2010-10-2','男','12888.55'),
('王五','1988-5-1','2012-12-2','男','1188.55'),
('吴红','1985-2-16','2010-10-2','女','15888.55'),
('小红','1968-2-1','2001-10-2','女','20888.55');
```

要求：

1. 将50岁以上的员工开除。

```
 delete from test1 where date_add(now(), interval -50 year)>birthday;
```

2. 将所有90后员工的工资涨1000员

```
 UPDATE test1 set salary = salary+1000 where birthday>'1990-1-1';
```

3. 得到公司30-40岁的员工有多少人

```
 SELECT count(*) from test1  WHERE  date_add(now(), interval -30
year)>birthday and date_add(now(), interval -40 year)<birthday;

 select count(*) from test1 where birthday between
date_add(now(),interval -40 year) and date_add(now(),interval -30 year);
```

4. 查询出公司年龄最大的人谁

```
select * from test1 order by birthday limit 0,1;
    嵌套查询
    select * from test1 where birthday in(select min(birthday) from
test1);
```

5. 查询出公司年龄最小的人是谁

```
    select * from test1 order by birthday desc limit 0,1 ;
    嵌套查询
    select * from test1 where birthday in(select max(birthday) from
test1);
```

6. 查询出公司总人数多少

```
    select count(*) from test1;
```

7. 分别查询出公司男姓和女姓的人数

```
select count(*) from test1 where sex='男';

select count(*) from test1 where sex='女';
```

8. 查询出工资大于1000的员工，要求保留小数点1位四舍五入

```
select name,round(salary,1) from test1 where salary>1000
```

9. 查询出工龄大于10年的所有员工

```
select * from test1 where inday < date_add(now(),interval -10 year)
```

10. 将公司姓吴的员工，且80后出生的人工资涨1000

```
update test1 set salary = salary+1000 where name like '吴%' and
birthday>'1980-1-1';
```

外键

创建一个主表：

```
create table dept (id int primary key auto_increment,name varchar(20));

insert into dept(name)value('教学部');
insert into dept(name)value('行政部');
insert into dept(name)value('人事部');
insert into dept(name)value('销售部');
```

创建一个从表添加外键

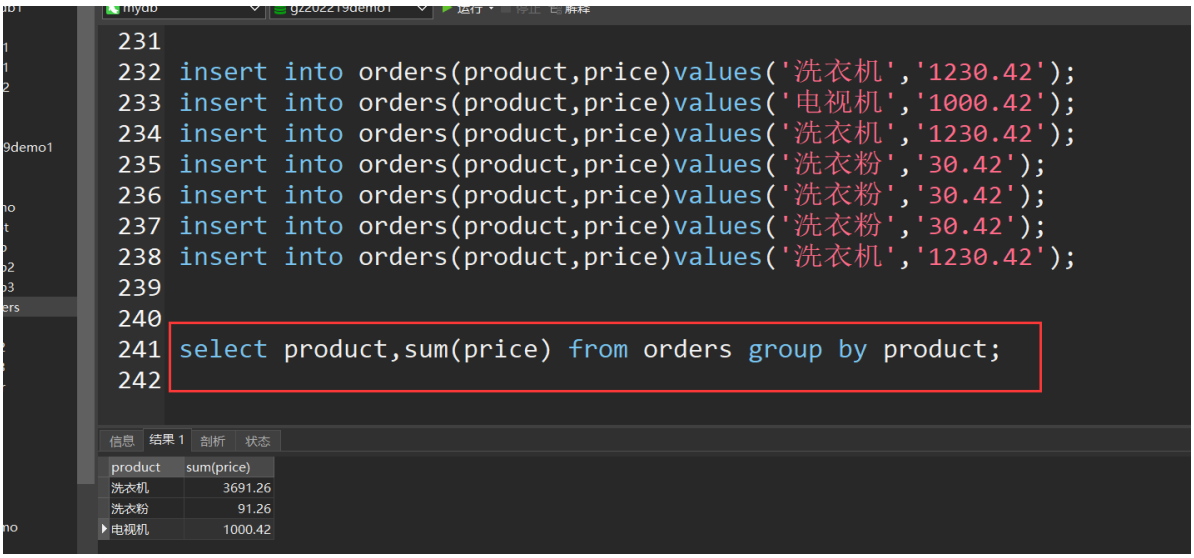
```
create table emp3(
id int primary key auto_increment,
name varchar(20),
dept_id int,
constraint dept_id_FK foreign key(dept_id) references dept(id));

drop table emp3;

insert into emp3(name,dept_id)values('张三','1');
insert into emp3(name,dept_id)values('李四','3');

insert into emp3(name,dept_id)values('李四','12');

select emp2.id,emp2.name,dept.name 部门 from emp2,dept where
emp2.dept_id=dept.id;
```

注意：外键列的字段数据类型，要和主表中的字段类型一致

group by语句 聚合函数

有一个订单表

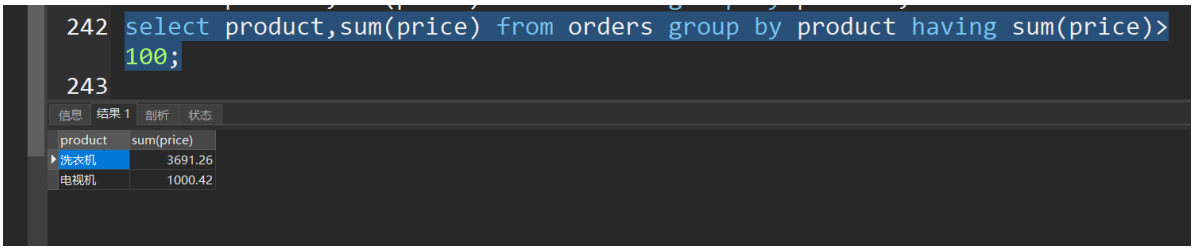
```
create table orders(id int primary key auto_increment,product varchar(100),price
float(10,2));

insert into orders(product,price)values('洗衣机','1230.42');
insert into orders(product,price)values('电视机','1000.42');
insert into orders(product,price)values('洗衣机','1230.42');
insert into orders(product,price)values('洗衣粉','30.42');
insert into orders(product,price)values('洗衣粉','30.42');
insert into orders(product,price)values('洗衣粉','30.42');
insert into orders(product,price)values('洗衣机','1230.42');


select product,sum(price) from orders group by product;
```



要求总和大于100元的才查询出来，group by 不能使用where，要用having

添加条件要用having

```
select product,sum(price) from orders group by product having sum(price)>100;
```



多表查詢

数据准备

```
//用户表
create table users(
id int primary key auto_increment,
name varchar(50),
sex varchar(5)
);
```

```
insert into users(name,sex)values('乔峰','男');
insert into users(name,sex)values('段誉','男');
insert into users(name,sex)values('阿朱','女');
insert into users(name,sex)values('虚竹','男');
insert into users(name,sex)values('梦姑','女');


//消息表
create table message(
id int primary key auto_increment,
title varchar(100),
info varchar(200),
user_id int
)

//添加外键
alter table message add constraint user_id_FK foreign key(user_id) references
users(id);

//添加数据
insert into message(title,info,user_id)values('瞎聊','朱朱你好吗','1');
insert into message(title,info,user_id)values('瞎聊','峰峰我很好','3');
insert into message(title,info,user_id)values('瞎聊','你吃饭了没','2');insert into
message(title,info,user_id)values('瞎聊','我还没有吃，没有你的日子吃不下','5');
insert into message(title,info,user_id)values('瞎聊','那我马上回来','2');
insert into message(title,info,user_id)values('瞎聊','那就在家等你','5');
insert into message(title,info,user_id)values('瞎聊','你做好饭','2');
insert into message(title,info,user_id)values('瞎聊','好的，已经最好大补汤','5');
```

多表连接查询

- 很多时候，需要选择的数据并不是来自一个表，而是来自多个数据表，这就需要使用多表连接查询。
- 多表连接查询有两种规范，较早的SQL92规范中支持如下几种多表连接查询：
  - 等值连接
  - 非等值连接
  - 外连接
  - 广义笛卡尔积
- SQL99规则提供了可读性更好的多表连接语法，并提供更多类型的连接查询，SQL99支持如下几种多表连接查询：
  - 交叉连接
  - 自然连接
  - 使用using子句的连接
  - 使用on子句的连接
  - 全外连接或者左、右外连接

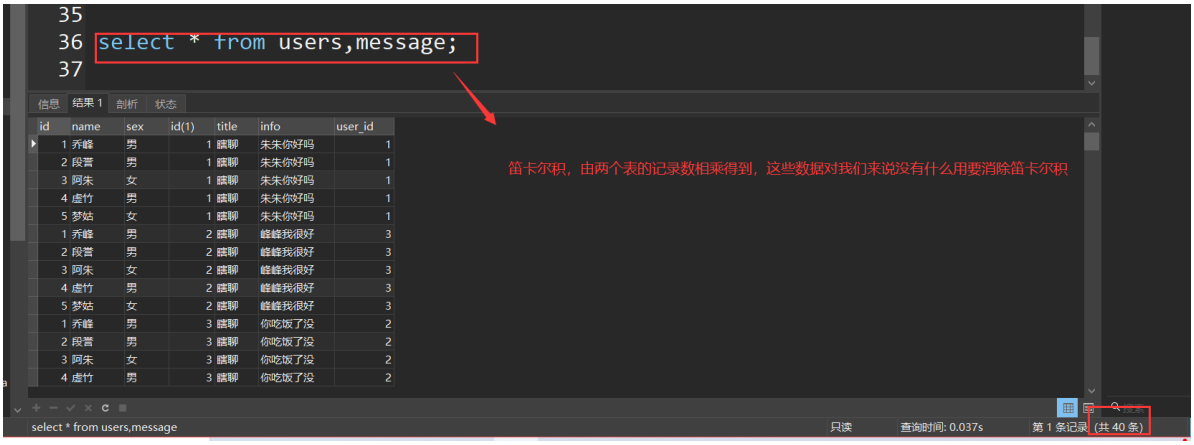
交叉连接

交叉连接

- 交叉连接效果就是**SQL92**的广义笛卡儿积，所以交叉连接无须任何连接条件

```
select s.* , teacher_name
# SQL 99 多表连接查询的 from 后只有一个表名
from student_table s
# cross join 交叉连接，相当于广义笛卡儿积
cross join teacher_table t;
```

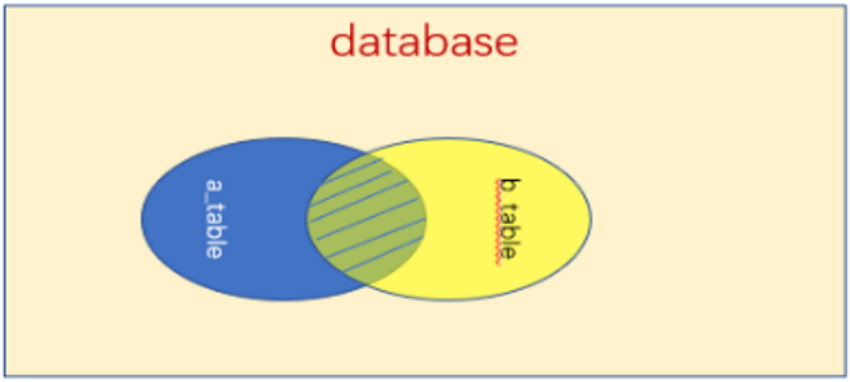
笛卡尔的数据是冗余的，所以这样的数据是无用的，其实我们的多表查询就是一个消除笛卡尔积的过程



内连接

内连接（ inner join ）

- 内连接指的是把表连接时表与表之间匹配的数据行查询出来，就是两张表之间数据行匹配时，要同时满足ON语句后面的条件才行。



隐式内连接

```
#隐式内连接
select m.id,m.title,m.info,u.name from users u,message m where u.id=m.user_id;
```

显示内连接

```
# 显示内内连接 就是吧inner join 写出来
select m.id,m.title,m.info,u.name from users u inner join message m  on
u.id=m.user_id;
```

外连接

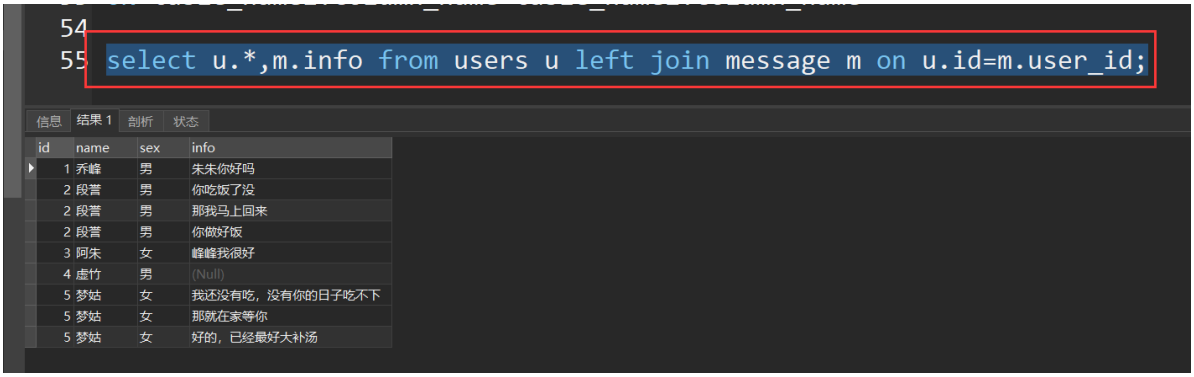
外连接（outer join）

- 外连接（**outer join**）：与内连接的区别在于，外连接不仅返回匹配的行，也会返回不匹配的行。
- （1）左外连接（left [outer] join）：结果集包括 LEFT OUTER子句中指定的左表的所有行，而不仅仅是连接列所匹配的行。如果左表的某行在右表中没有匹配行，则在结果集的行中右表的所有选择列表列均为空值（null）。
- （2）右外连接（right [outer] join）：与左连接相反。结果集包括 right OUTER子句中指定的右表的所有行，而不仅仅是连接列所匹配的行。如果右表的某行在左表中没有匹配行，则在结果集的行中左表的所有选择列表列均为空值（null）。

左外连接

left join--->特点，左边的表全部显示，右边的表有则显示，无则显示null

```
select u.*,m.info from users u left join message m on u.id=m.user_id;
```



在左外连接添加一个聚合分组

```
select u.*,m.info from users u left join message m on u.id=m.user_id group by
u.id;

//得到每个人发过几次消息
select u.name,count(m.info) from users u left join message m on u.id=m.user_id
group by u.id;
```

右外连接

right join--->特点，右边的表全部显示，左边的表有则显示，无则显示null

```
select u.*,m.info from users u right join message m on u.id=m.user_id ;
```

如果将表的顺序颠倒左连接就是右连接

```
select u.*,m.info from  message m left join users u on u.id=m.user_id ;
```

嵌套查询(子查询)

查询语句里面嵌套查询语句

要求查询出发过言的人

```
select distinct u.name from users u,message m where u.id=m.user_id;
用嵌套查询来完成相同的功能
select name from users where id in(select user_id from message);
```

事务，视图，触发器