SQL语句DDL

DataGrip

```sql
#显示所有数据库
show databases ;

#创建数据库
create database database1;

#如果不存在database1就创建
create database if not exists database1;

#使用数据库
use database1;

#创建表
create table student(
    sid int(11),
    name varchar(255), #字符
    age int(11)
);

# 删除数据库
drop database database1;

#如果存在database1就删除
drop database if exists database1;

#修改数据库编码
alter database database1 char set utf8;

desc database1.t_stu;

#修改表结构
alter table student add tel char(11);#添加一列
alter table student drop age;#删除类
alter table student modify column tel int(11);#修改列数据类型
alter table student change tel telephone char(11);#修改tel列为telephone

# 修改表名
rename table student to t_stu;

#查看建表语句
show create table t_stu;
CREATE TABLE `t_stu` (
  `sid` int DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `gender` varchar(20) DEFAULT NULL,
  `birth` date DEFAULT NULL,
  `address` varchar(20) DEFAULT NULL,
  `score` double DEFAULT NULL,
  `telephone` char(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

```

#查看建库语句
show create database database1;
CREATE DATABASE `database1` /*!40100 DEFAULT CHARACTER SET utf8mb3 */
/*!80016 DEFAULT ENCRYPTION='N' */


#插入一条数据
insert into database1.student (sid, `name`, age) values(0,"王五",24);
```

数据表的约束

```
1.主键约束
主键约束即primary key用于唯一的标识表中的每一行。被标识为主键的数据在表中是唯一的且其值不能为空。
这点类似于我们每个人都有一个身份证号，并且这个身份证号是唯一的。
主键约束基本语法：
字段名  数据类型   primary key;
设置主键约束的两种方式
1. create table student (id int primary key);
2. create table student (id int , primary key (id));
-----------------------------------------------------------------
2.非空约束
非空约束即not null指的是字段到的值不能为空，

非空约束基本语法：
字段名  数据类型  not null;
-----------------------------------------------------------------
3.默认值约束
默认值约束即DEFAULT用于给数据表中的字段指定默认值，即当在表中插入一条新记录时若未给该字段赋值，那么，数据库系统会自动为这个字段插人默认值；

默认值约束语法：
字段名  数据类型  default 默认值;
-----------------------------------------------------------------
4.唯一性约束
唯一性约束即unique用于保证数据表中字段的唯一性，即表中字段的值不能重复出现，其基本的语法格式如下所示：

唯一性约束语法：
字段名  数据类型  unique;
-----------------------------------------------------------------
5.外键约束
外键约束即foreign key常用于多张表之间的约束

基本语法：
1. 创建数据表时指定：constraint  外键名  foreign key (从表外键字段) references  主表 （主键字段）;
2. 创建数据表后指定：alter table 表名  add constraint 外建名  foreign key(从表外键字段) references（主键字段);
-----------------------------------------------------------------
create table if not exists student(
    sid int primary key  auto_increment,
    name varchar(20) unique ,
    age int(11) not null
```

```
39 );
```

数值类型

整型

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 byte	(-128，127)	(0，255)	小整数值
SMALLINT	2 bytes	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 bytes	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 bytes	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 bytes	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 bytes	(-3.402 823 466 E+38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度 浮点数值
DOUBLE	8 bytes	(-1.797 693 134 862 315 7 E+308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度 浮点数值
DECIMAL		依赖于M和D的值	依赖于M和D的值	小数值

字符串

类型	大小	用途
CHAR	0-255 bytes	定长字符串
VARCHAR	0-65535 bytes	变长字符串
TINYBLOB	0-255 bytes	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 bytes	短文本字符串
BLOB	0-65 535 bytes	二进制形式的长文本数据
TEXT	0-65 535 bytes	长文本数据
MEDIUMBLOB	0-16 777 215 bytes	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 bytes	中等长度文本数据
LONGBLOB	0-4 294 967 295 bytes	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 bytes	极大文本数据

日期

类型	大小 (bytes)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 2147483647 秒，北京时间 2038-1-19 11:14:07，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

对表结构的常用操作-其他操作

功能	SQL
查看当前数据库的所有表名称	show tables;
查看指定某个表的创建语句	show create table 表名;
查看表结构	desc 表名
删除表	drop table 表名

```
-- 查看当前数据所有的表
show tables;

-- 查看指定表的创建语句
show create table student;

CREATE TABLE `student` (
  `sid` int NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  `age` int NOT NULL,
  PRIMARY KEY (`sid`),
  UNIQUE KEY `name` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb3

-- 查看表结构
desc student;

+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| sid   | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | YES  | UNI | NULL    |                |
| age   | int         | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+

-- 删除表
drop table student;
```

对表结构的常用操作-修改表结构格式

```
-- 添加列
alter table student add dept varchar(20);
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| sid   | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | YES  | UNI | NULL    |                |
| age   | int         | NO   |     | NULL    |                |
| dept  | varchar(20) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+


-- 修改列名和类型
alter table student change dept department varchar(30);
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| sid        | int         | NO   | PRI | NULL    | auto_increment |
| name       | varchar(20) | YES  | UNI | NULL    |                |
| age        | int         | NO   |     | NULL    |                |
| department | varchar(30) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+


-- 修改表删除列
alter table student drop department;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| sid   | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | YES  | UNI | NULL    |                |
| age   | int         | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+


-- 修改表名
rename table student to stu;
+---------------------+
| Tables_in_database1 |
+---------------------+
| stu                 |
+---------------------+
```

数据库操作DML

```
-- 数据插入
insert into student (sid,name,age)values(5,'老七',19),(6,'老八',20);

insert into student(sid,age) values(100,20);


insert into student values(7,'老九',21),(8,'老十',22);
```

```
sid name age
1   张三  20
2   李四  20
3   王五  24
4   老六  18
5   老七  19
6   老八  20
7   老九  21
8   老十  22
100     20
```

```
-----------------------------------------------------------------
```

```
-- 数据修改
-- 将所有学生的年纪修改为20
update student set address ='武汉';

-- 将4的地址改为葛店
update student set address ='葛店' where sid =4;

update student set address ='长职' where sid >4;

-- 将id为3的学生地址修改为光谷 年龄修改为18
update student set address = '光谷',age=18 where sid =3;
```

```
-----------------------------------------------------------------
```

```
-- 数据删除
-- 删除sid为4的学生数据
delete from student where sid =4;

-- 删除表所有数据
delete from student;

-- 清空表数据
truncate table student;
truncate student;
```

```
-----------------------------------------------------------------
```

```
-- 案例
-- 创建员工表employee
-- id name gender salary
create table if not exists database1.employee(
id int,
name varchar(20),
gender varchar(10),
salary double
);

-- 插入数据
-- 1 张三 男 2000
-- 2 李四 男 1000
-- 3 王五 女 4000
insert into employee values(1,'张三','男',2000),(2,'李四','男',1000),(3,'王五','女',4000);

-- 修改表数据
```

```sql
-- 将所有员工薪水修改为5000元
update employee  set salary=5000;

-- 将姓名为张三的员工薪水修改为3000元
update employee set salary=3000 where name = '张三';

-- 将姓名为李四的员工薪水修改为4000元 gender改为女
update employee set salary=4000,gender='女' where name = '李四';

-- 将王五的薪水在原有的基础上增加1000元
update employee set salary=salary+1000 where name = '王五';
```

Mysql约束

主键约束

方式1-语法：

```sql
-- 在 create table 语句中，通过 PRIMARY KEY 关键字来指定主键。
--在定义字段的同时指定主键，语法格式如下：
create table 表名(
    ...
    <字段名> <数据类型> primary key
    ...
)
```

方式1-实现：

```sql
create table emp1(
    eid int primay key,
    name VARCHAR(20),
    deptId int,
    salary double
);
```

方式2-语法：

```sql
--在定义字段之后再指定主键，语法格式如下：
create table 表名(
    ...
    [constraint <约束名>] primary key [字段名]
);
```

方式2-实现：

```sql
create table emp2(
    eid INT,
    name VARCHAR(20),
    deptId INT,
    salary double,
    constraint  pk1 primary key(id)
);
```

```sql
-- 联合主键
-- 所谓的联合主键 就是这个主键是由一张表中多个字段组成的
-- primary key (字段1，字段2....，字段n)
create table emp3(
name varchar(20),
deptId int,
salary double,
constraint pk2 primary key(name,deptId)
);
```

```
insert into emp3 values('张三',10,5000);
insert into emp3 values('张三',20,5000);

-- 联合主键的各列 每一列都不能为空
insert into emp3 values(NULL,30,5000);
insert into emp3 values('赵六',NULL,5000);
insert into emp3 values(NULL,NULL,5000);
-----------------------------------------------------------------

-- 添加单列主键
create table emp4(
eid int,
name varchar(20),
deptId int,
salary double
);

alter table emp4 add primary key(eid);


-- 创建多列主键
create table emp5(
eid int,
name varchar(20),
deptId int,
salary double
);

alter table emp5 add primary key(name,deptId);
-----------------------------------------------------------------
-- 删除主键约束
alter table emp5 drop primary key;
```

自增长约束

```
-- 自增长约束
use database1;
create table t_user1(
id int primary key auto_increment,
name varchar(20)
);

INSERT into t_user1 values(null,'张三');
INSERT into t_user1(name) values('李四');

-- 创建表时指定
create table t_user2(
id int primary key auto_increment,
name varchar(20)
)auto_increment=100;

insert into t_user2 values(null,'张三');
insert into t_user2 values(null,'李四');
-----------------------------------------------------------------
```

```
-- 创建表之后指定
create table t_user3(
id int primary key auto_increment,
name varchar(20)
);

insert into t_user3 values(null,'张三');
insert into t_user3 values(null,'李四');
insert into t_user3 values(null,'王五');

alter table t_user3 auto_increment=100;
id  name
1   张三
2   李四
100 王五
-----------------------------------------------------------------
delete和truncate在删除后自增列的变化

delete
use database1;
create table t_user1(
id int primary key auto_increment,
name varchar(20)
);

INSERT into t_user1 values(null,'张三');
delete from t_user1; -- delete删除数据之后 自增长还是在最后一个值基础上+1
id  name
3   张三


truncate
create table t_user2(
id int primary key auto_increment,
name varchar(20)
)auto_increment=100;

insert into t_user2 values(null,'张三');
insert into t_user2 values(null,'李四');

truncate t_user2;  -- truncate删除数据之后 自增长从1开始+1
id  name
1   张三
```

非空约束(not null)

```
-- 非空约束
-- 创建表时指定
create table t_user6(
id int,
name varchar(20) not null,
address varchar(20) not null
);
+---------+-------------+------+-----+---------+-------+
```

```
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| address | varchar(20) | NO   |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+

-- 创建表之后指定
alter table t_user3 modify name varchar(20) not null;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+

-- 删除非空约束
alter table t_user6 modify address varchar(20);
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| address | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

唯一约束

```
-- 唯一约束
-- 创建表时添加
create table t_user8(
id int,
name varchar(20),
phone_number varchar(20) unique
);
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| id           | int         | YES  |     | NULL    |       |
| name         | varchar(20) | YES  |     | NULL    |       |
| phone_number | varchar(20) | YES  | UNI | NULL    |       |
+--------------+-------------+------+-----+---------+-------+

insert into t_user8 values(1001,'张三',null);
insert into t_user8 values(1002,'张三1',null);
id      name    phone
1001    张三     (NULL)
1002    张三1    (NULL)
在mysql中NULL和任何值都不相同 NULL不等于NULL

-- 创建表之后指定
alter table t_user6 add constraint UNI unique(address);
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
```

```
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| address | varchar(20) | YES  | UNI | NULL    |       |
+---------+-------------+------+-----+---------+-------+

-- 删除唯一约束
alter table t_user6 drop index UNI;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| address | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

默认约束

```
-- 默认约束
-- 创建表时添加
create table t_user10(
id int,
name varchar(20),
address varchar(20) default '鄂州'
);
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | YES  |     | NULL    |       |
| address | varchar(20) | YES  |     | 鄂州    |       |
+---------+-------------+------+-----+---------+-------+

-- 创建表之后指定
alter table t_user10 modify address varchar(20) default '长职';
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | YES  |     | NULL    |       |
| address | varchar(20) | YES  |     | 长职    |       |
+---------+-------------+------+-----+---------+-------+

-- 删除默认约束
alter table t_user10 modify column address varchar(20) default null;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | YES  |     | NULL    |       |
| address | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

零填充约束

```
-- 零填充约束
create table t_user12(
id int zerofill,  -- zerofill默认为int(10) 当插入字段的值小于定义的长度时 会在该值的前面补上相应的0
name varchar(20)
);
+-------+--------------------------+------+-----+---------+-------+
| Field | Type                     | Null | Key | Default | Extra |
+-------+--------------------------+------+-----+---------+-------+
| id    | int(10) unsigned zerofill | YES  |     | NULL    |       |
| name  | varchar(20)              | YES  |     | NULL    |       |
+-------+--------------------------+------+-----+---------+-------+


```

DQL-基本查询

```
create  table product(
pid int primary key auto_increment,
pname varchar(20) not null,
price double,
categroy_id varchar(20)
);

insert into product values(null,'海尔洗衣机',5000,'c001');
insert into product values(null,'美的冰箱',3000,'c001');
insert into product values(null,'格力电饭煲',5000,'c001');
insert into product values(null,'九阳电饭煲',5000,'c001');

-- 查询所有的商品
select * from product;

-- 查询商品名和商品的价格
select pname,price from product;

-- 别名查询.使用的关键字是as
-- 表别名
select * from product as p;
select *from product p;

select p.id,u.id from product p, user u;

-- 列别名
select pname as '商品名',price '商品价格' from product;

-- 去掉重复值
select distinct price from product;

-- 查询结果是表达式(运算查询):将所有商品的价格加价10元进行显示
select pname, price +10 new_price from product;

-- 将所有商品的价格加价10%进行显示
```

```sql
select pname, price * 1.1 as new_price from product;

-- 查询商品名称为海尔洗衣机的商品所有信息
select * from product where pname = '海尔洗衣机';

-- 查询价格为5000商品
select * from product where price = 5000;

-- 查询价格不为5000的所有商品
select * from product where price != 5000;
select * from product where price <> 5000;
select * from product where not(price = 5000);

-- 查询价格大于3000元的所有商品信息
select * from product where price > 3000;

-- 查询价格在3000到5000之间的商品信息
select * from product where price >=3000 && price<=5000;
select * from product where price between 3000 and 5000;

-- 查询价格是3000或5000的所有商品
select * from product where price = 3000 || price =5000;

-- 查询含有'格力'的所有商品
select * from product where pname like '%格力%';         -- %为任意字符

-- 查询以'海'开头的所有商品
select * from product where pname like '海%';

-- 查询第二个字为'力'的所有商品
select * from product where pname like '_力%';           -- 下划线匹配单个字符

-- 查询category_id为Null的商品
select * from product where categroy_id is null;

-- 查询categroy_id不为null分类的商品
select * from product where categroy_id is not null;

-- 使用least求最小值
select least(10,20,30) as small_number;
select least(10,null,20);       -- 如果求最小值中 有个值为Null 则不会进行比较 结果直接为null

-- 使用greatest求最大值
select greatest(50,20,40) as big_number;
select greatest(10,null,20);       -- 如果求最大值中 有个值为Null 则不会进行比较结果直接为null
```

排序查询

```sql
-- 排序查询
-- 使用价格查询 降序
select * from product order by price desc;

-- 在价格降序的基础上 以分类降序
select * from product order by price desc,categroy_id desc;

-- 显示商品的价格去重 并降序
select distinct price from product order by price desc;
```

聚合查询

```sql
-- 聚合查询
-- 查询商品的总条目
select count(pid) from product;
select count(*) from product;

-- 查询价格大于200商品的总条目
select count(pid) from product where price > 4500;

-- 查询分类为'c001'的所有商品的总和
select sum(price) from product where categroy_id ='c001';

-- 查询商品的最大价格
select max(price) from product;

-- 查询商品的最小价格
select min(price) from product;

-- 查询分类为'c001'所有商品的平均价格
select avg(price) from product where categroy_id='c001';


-- 聚合查询Null值的处理
create table test_null(
c1 varchar(20),
c2 int
);

insert into test_null values('aaa',3);
insert into test_null values('bbb',3);
insert into test_null values('ccc',null);
insert into test_null values('ddd',6);

select count(*), count(1), count(c2) from test_null;  -- 4 4 3
select sum(c2), max(c2), min(c2), avg(c2) from test_null;  -- 12 6 3 4
```

分组查询

```
-- 分组查询
-- select 字段1,字段2 from group by 分组字段 having 分组条件
-- 统计各个分类商品的个数
-- 分组之后 select的后边只能写分组字段和聚合函数
select categroy_id,count(pid) from product group by categroy_id;

-- where子句用来筛选from子句中指定的操作所产生的行
-- group by子句用来分组where子句的输出
-- having子句用来从分组的结果中筛选行
-- select 字段1,字段2 from 表名 group by 分组字段 having 分组条件;

-- 统计各个分类商品的个数 且只显示个数大于4的信息
select categroy_id,count(*) from product group by categroy_id having count(*) > 1 order by categroy_id;
```

分页查询

```
-- 分页查询
-- 显示前3条
select * from product limit 3;

-- 从第2条开始显示 显示3条
select * from product limit 2,3;

-- 分页显示
select * from product limit 0,60;
select * from product limit 60,60;
select * from product limit 120,60;
select * from product limit (n-1)*60,60;
```

insert into select语句

```
-- insert into select
select * from product;

create table product2(
pname varchar(20),
price double
);

insert into product2(pname,price) select pname,price from product;
select * from product2;

create table product3(
categroy_id varchar(20),
product_count int
);

insert into product3 select categroy_id , count(*) from product group by categroy_id;
select * from product3;
```

练习

```sql
-- 练习
create table student(
id int,
name varchar(20),
gender varchar(20),
chinese int,
english int,
math int
);

insert into student values(1,'张明','男',89,78,90);
insert into student values(2,'李进','男',67,53,95);
insert into student values(3,'王五','女',87,78,77);
insert into student values(4,'李一','女',88,98,92);
insert into student values(5,'李财','男',82,84,67);
insert into student values(6,'张宝','男',55,85,45);
insert into student values(7,'黄蓉','女',75,65,30);
insert into student values(7,'黄蓉','女',75,65,30);

-- 查询学生表中所有的信息
select * from student;

-- 查询表中所有学生的姓名和对应的英语成绩
select name,english from student;

-- 过滤表中重复数据
select distinct * from student;

-- 统计每个学生的总分
select name,chinese + english + math sum from student;

-- 在所有学生总分数上加10分特长分
select name,chinese + english + math +10 from student;

-- 使用别名表示学生的分数
select name,chinese '语文成绩', english '英语成绩', math '数学成绩' from
student;

-- 查询英语成绩大于90的同学
select name from student where english > 90;

-- 查询总分大于200分的同学
select name from student where (chinese + english + math) > 200;

-- 查询英语分数在80-90之间的同学
select name from student where english >= 80 && english <= 90;

-- 查询英语分数不在80-90之间的同学
select name from student where english <80 || english >90;

-- 查询数学分数为89 90 91的同学
select * from student where math in(89,90,91);
select name from student where math=89 || math=90 || math=91;
```

```sql

-- 查询数学分数不为89 90 91的同学
select * from student where math not in(89,90,91);
select * from student where not math in(89,90,91);

-- 查询所有姓李的学生的英语成绩
select name,english from student where name like '李%';

-- 查询数学分80并且语文分80的同学
select * from student where math=80 && chinese=80;

-- 查询英语80或者总分200的同学
select * from student where english=80 || (chinese + english + math)=200;

-- 对数学成绩降序
select * from student order by math desc;

-- 对总分降序排序
select * from student order by (chinese + english + math) desc;

-- 对姓李的学生成绩排序输出
select * from student where name like '李%' order by (chinese + english +
math) desc;

-- 查询男生和女生分别又多少人 并将人数降序输出
select gender,count(*) as total_cnt from student group by gender order by
total_cnt desc;

-- 查询男生和女生分别又多少人 并将人数降序输出 查询出人数大于4的性别人数信息
select gender,count(*) as total_cnt from student group by gender having
total_cnt >4 order by total_cnt desc;

----------------------------------------------------------------
create table emp(
empno int,
ename varchar(50),
job varchar(50),
mgr int,
hiredate date,
sal int,
comn int,
deptno int
);

insert into emp values(7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20),

(7499,'ALEN','SALESMAN',7698,'1981-02-20',1600,300,30),

(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30),

(7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20),

(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30),

(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30),
```

```
(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10),

(7788,'SCOTT','ANALYST',7566,'1987-04-19',3000,NULL,20),
                        (7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10),
                        (7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30),
                        (7876,'ADAMS','CLERK',7788,'1987-05-23',1100,NULL,20),
                        (7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30),
                        (7902,'FORD','ANALYST',7566,'1981-12-03',3000,NULL,20),
                        (7934,'MILLER','CLERK',7782,'1982-01-23',1300,NULL,10);

-- 按员工编号升序排列不在10号部门工作的员工信息
select * from emp where deptno != 10 order by empno;

-- 查询姓名第二个字母不是A且薪水大于1000元的员工信息 按年薪降序排序
-- ifnull(sal,0) 如果sal的值为null 则当作0 不为Null 则还是原来的值
select * from emp where ename not like '_A%' and sal > 1000 order by (12*sal+ifnull(comn,0)) desc;

-- 求每个部门的平均薪水
select deptno,avg(sal) from emp group by deptno;
select deptno,avg(sal) as avg_sal from emp group by deptno order by avg_sal desc;

-- 求每个部门的最高薪水
select deptno,max(sal) from emp group by deptno;

-- 求每个部门每个岗位的最高薪水
select deptno,job,max(sal) from emp group by deptno,job order by deptno;

-- 求平均薪水大于2000的部门编号
select deptno,avg(sal) avg_sal from emp group by deptno having avg_sal > 2000;

-- 将部门平均薪水大于1500的部门编号列出来 按部门平均薪水降序排列
select deptno,avg(sal) avg_sal from emp group by deptno having avg_sal > 1500 order by avg_sal desc;

-- 选择公司中有奖金的员工姓名 工资
select ename,sal from emp where comn is not null;

-- 查询员工最高工资和最低工资的差距
select max(sal) - min(sal) as cha from emp;
```

运算符

比较运算符	说明
=	等于
< 和 <=	小于和小于等于
> 和 >=	大于和大于等于
<=>	安全的等于，两个操作码均为NULL时，其所得值为1；而当一个操作码为NULL时，其所得值为0
<> 或!=	不等于
IS NULL 或 ISNULL	判断一个值是否为 NULL
IS NOT NULL	判断一个值是否不为 NULL
LEAST	当有两个或多个参数时，返回最小值
GREATEST	当有两个或多个参数时，返回最大值
BETWEEN AND	判断一个值是否落在两个值之间
IN	判断一个值是IN列表中的任意一个值
NOT IN	判断一个值不是IN列表中的任意一个值
LIKE	通配符匹配
REGEXP	正则表达式匹配

正则表达式

模式	描述
^	匹配输入字符串的开始位置。
$	匹配输入字符串的结束位置。
.	匹配除 "\n" 之外的任何单个字符。
[...]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。
[^...]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'。
p1|p2|p3	匹配 p1 或 p2 或 p3。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood" 或 "food"。

模式	描述
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,m}	m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。

```
-- 正则表达式
-- ^ 在字符串开始处进行匹配
select 'abc' regexp '^a';
select * from student where name regexp '^张';

-- $ 在字符串末尾开始匹配
select 'abc' regexp 'a$';
select 'abc' regexp 'c$';
select * from student where name regexp '宝$';

-- . 匹配任意字符 可以匹配除换行符外的任意字符
select 'abc' regexp '.b';-- 1
select 'abc' regexp '.c';-- 1
```

```
select 'abc' regexp 'a.';-- 1

-- [...] 匹配括号内的任意单个字符  正则表达式的任意字段是否在前边的字符串中出现
select 'abc' regexp '[xyz]';  -- 0
select 'abc' regexp '[xaz]';  -- 1

-- [^...] 注意^符合只有在[]内才是取反的意思 在别的地方都是表示开始处匹配
select 'a' regexp '[^abc]';  -- 0
select 'x' regexp '[^abc]';  -- 1
select 'abc' regexp '[^a]';  -- 1

-- a* 匹配0个或多个a 包括空字符串 可以作为占位符使用  .有没有指定字符都可以匹配到数据
select 'stab' regexp '.ta*b'; -- 1
select 'stb' regexp '.ta*b';  -- 1
select '' regexp 'a*';  -- 1

-- a+ 匹配1个或者多个a 但是不包括空字符
select 'stab' regexp '.ta+b';  -- 1
select 'stb' regexp '.ta+b';  -- 0

-- a? 匹配0个或者1个a
select 'stb' regexp '.ta?b';  -- 1
select 'stab' regexp '.ta?b'; -- 1
select 'staab' regexp '.ta?b'; -- 0

-- a1|a2 匹配a1或者a2
select 'a' regexp 'a|b';  -- 1
select 'b' regexp 'a|b';  -- 1
select 'b' regexp '^(a|b)'; -- 1
select 'a' regexp '^(a|b)'; -- 1
select 'c' regexp '^(a|b)'; -- 0

-- a{m} 匹配m个a
select 'auuuuc' regexp 'au{4}c'; -- 1
select 'auuuuc' regexp 'au{3}c'; -- 0

-- a{m,} 匹配m个或者更多个a
select 'auuuuc' regexp 'au{3,}c'; -- 1
select 'auuuuc' regexp 'au{4,}c'; -- 1
select 'auuuuc' regexp 'au{5,}c'; -- 0

-- a{m,n} 匹配m到n个a 包含m和n
select 'auuuuc' regexp 'au{3,5}c'; -- 1
select 'auuuuc' regexp 'au{4,5}c'; -- 1
select 'auuuuc' regexp 'au{5,10}c'; -- 0

-- (abc) abc作为一个序列匹配 不用括号括起来都是用单个字符去匹配 如果要把多个字符作为一个整体去匹配就需要用到括号 所以括号适合上面的所有情况
select 'xababy' regexp 'x(abab)y';  -- 1
select 'xababy' regexp 'x(ab)*y';   -- 1
select 'xababy' regexp 'x(ab){1,2}y'; -- 1
select 'xababy' regexp 'x(ab){3}y'; -- 0
```

DOS

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| database1          |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> use database1
Database changed
mysql> show tables;
+---------------------+
| Tables_in_database1 |
+---------------------+
| student             |
+---------------------+
1 row in set (0.00 sec)

mysql> desc student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| sid   | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | YES  | UNI | NULL    |                |
| age   | int         | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+


```