

MySQL

SQL语句DDL

DataGrip

```
1 #显示所有数据库
2 show databases ;
3
4 #创建数据库
5 create database database1;
6
7 #如果不存在database1就创建
8 create database if not exists database1;
9
10 #使用数据库
11 use database1;
12
13 #创建表
14 create table student(
15     sid int(11),
16     name varchar(255), #字符
17     age int(11)
18 );
19
20 # 删除数据库
21 drop database database1;
22
23 #如果存在database1就删除
24 drop database if exists database1;
25
26 #修改数据库编码
27 alter database database1 char set utf8;
28
29 desc database1.t_stu;
30
31 #修改表结构
32 alter table student add tel char(11);#添加一列
33 alter table student drop age;#删除类
34 alter table student modify column tel int(11);#修改列数据类型
35 alter table student change tel telephone char(11);#修改tel列为telephone
36
37 # 修改表名
38 rename table student to t_stu;
39
40 #查看建表语句
41 show create table t_stu;
42 CREATE TABLE `t_stu` (
43     `sid` int DEFAULT NULL,
44     `name` varchar(20) DEFAULT NULL,
45     `gender` varchar(20) DEFAULT NULL,
46     `birth` date DEFAULT NULL,
47     `address` varchar(20) DEFAULT NULL,
```

```

48 `score` double DEFAULT NULL,
49 `telephone` char(11) DEFAULT NULL
50 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
51
52 #查看建库语句
53 show create database database1;
54 CREATE DATABASE `database1` /*!40100 DEFAULT CHARACTER SET utf8mb3 */
55 /*!80016 DEFAULT ENCRYPTION='N' */
56
57 #插入一条数据
58 insert into database1.student (sid, `name`, age) values(0,"王五",24);

```

数据表的约束

1 **1. 主键约束**
2 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。
3 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。
4 主键约束基本语法：
5 字段名 数据类型 **primary key**；
6 设置主键约束的两种方式
7 1. **create table student (id int primary key);**
8 2. **create table student (id int , primary key (id));**
9 -----
10 **2. 非空约束**
11 非空约束即**not null**指的是字段到的值不能为空，
12
13 非空约束基本语法：
14 字段名 数据类型 **not null**；
15 -----
16 **3. 默认值约束**
17 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入默认值；
18
19 默认值约束语法：
20 字段名 数据类型 **default** 默认值；
21 -----
22 **4. 唯一性约束**
23 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：
24
25 唯一性约束语法：
26 字段名 数据类型 **unique**；
27 -----
28 **5. 外键约束**
29 外键约束即**foreign key**常用于多张表之间的约束
30
31 基本语法：
32 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)；
33 2. 创建数据表后指定：**alter table 表名 add constraint 外键名 foreign key**(从表外键字段) **references** (主键字段)；
34 -----
35 **create table if not exists student(**

```

36     sid int primary key auto_increment,
37     name varchar(20) unique ,
38     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 |

```

1 -- 查看当前数据所有的表
2 show tables;
3
4 -- 查看指定表的创建语句
5 show create table student;
6
7 CREATE TABLE `student` (
8     `sid` int NOT NULL AUTO_INCREMENT,
9     `name` varchar(20) DEFAULT NULL,
10    `age` int NOT NULL,
11    PRIMARY KEY (`sid`),
12    UNIQUE KEY `name` (`name`)
13 ) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb3
14
15 -- 查看表结构
16 desc student;
17
18 +-----+-----+-----+-----+-----+
19 | Field | Type      | Null | Key | Default | Extra          |
20 +-----+-----+-----+-----+-----+
21 | sid   | int       | NO   | PRI | NULL    | auto_increment |
22 | name  | varchar(20) | YES  | UNI | NULL    |                |
23 | age   | int       | NO   |     | NULL    |                |
24 +-----+-----+-----+-----+-----+
25
26 -- 删除表
27 drop table student;

```

对表结构的常用操作-修改表结构格式

```
1 -- 添加列
2 alter table student add dept varchar(20);
3 +-----+-----+-----+-----+
4 | Field | Type      | Null | Key | Default | Extra       |
5 +-----+-----+-----+-----+
6 | sid   | int       | NO   | PRI  | NULL    | auto_increment |
7 | name  | varchar(20) | YES  | UNI  | NULL    |               |
8 | age   | int       | NO   |       | NULL    |               |
9 | dept  | varchar(20) | YES  |       | NULL    |               |
10+-----+-----+-----+-----+
11
12
13 -- 修改列名和类型
14 alter table student change dept department varchar(30);
15 +-----+-----+-----+-----+
16 | Field      | Type      | Null | Key | Default | Extra       |
17 +-----+-----+-----+-----+
18 | sid        | int       | NO   | PRI  | NULL    | auto_increment |
19 | name       | varchar(20) | YES  | UNI  | NULL    |               |
20 | age        | int       | NO   |       | NULL    |               |
21 | department | varchar(30) | YES  |       | NULL    |               |
22+-----+-----+-----+-----+
23
24
25 -- 修改表删除列
26 alter table student drop department;
27 +-----+-----+-----+-----+
28 | Field | Type      | Null | Key | Default | Extra       |
29 +-----+-----+-----+-----+
30 | sid   | int       | NO   | PRI  | NULL    | auto_increment |
31 | name  | varchar(20) | YES  | UNI  | NULL    |               |
32 | age   | int       | NO   |       | NULL    |               |
33+-----+-----+-----+-----+
34
35
36 -- 修改表名
37 rename table student to stu;
38 +-----+
39 | Tables_in_database1 |
40 +-----+
41 | stu                 |
42 +-----+
```

数据库操作DML

```
1 -- 数据插入
2 insert into student (sid,name,age)values(5,'老七',19),(6,'老八',20);
3
4 insert into student(sid,age) values(100,20);
5
6
7 insert into student values(7,'老九',21),(8,'老十',22);
```

```
8
9  sid name age
10 1  张三  20
11 2  李四  20
12 3  王五  24
13 4  老六  18
14 5  老七  19
15 6  老八  20
16 7  老九  21
17 8  老十  22
18 100    20
19 -----
20
21 -- 数据修改
22 -- 将所有学生的年纪修改为20
23 update student set address ='武汉';
24
25 -- 将4的地址改为葛店
26 update student set address ='葛店' where sid =4;
27
28 update student set address ='长职' where sid >4;
29
30 -- 将id为3的学生地址修改为光谷 年龄修改为18
31 update student set address = '光谷',age=18 where sid =3;
32
33 -----
34 -- 数据删除
35 -- 删除sid为4的学生数据
36 delete from student where sid =4;
37
38 -- 删除表所有数据
39 delete from student;
40
41 -- 清空表数据
42 truncate table student;
43 truncate student;
44
45 -----
46 -- 案例
47 -- 创建员工表employee
48 -- id name gender salary
49 create table if not exists database1.employee(
50   id int,
51   name varchar(20),
52   gender varchar(10),
53   salary double
54 );
55
56 -- 插入数据
57 -- 1 张三 男 2000
58 -- 2 李四 男 1000
59 -- 3 王五 女 4000
60 insert into employee values(1,'张三','男',2000),(2,'李四','男',1000),(3,'王
5 五','女',4000);
61
62 -- 修改表数据
```

```
63 -- 将所有员工薪水修改为5000元
64 update employee set salary=5000;
65
66 -- 将姓名为张三的员工薪水修改为3000元
67 update employee set salary=3000 where name = '张三';
68
69 -- 将姓名为李四的员工薪水修改为4000元 gender改为女
70 update employee set salary=4000,gender='女' where name = '李四';
71
72 -- 将王五的薪水在原有的基础上增加1000元
73 update employee set salary=salary+1000 where name = '王五';
```

Mysql约束

主键约束

方式1-语法：

```
-- 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：

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

方式2-语法：

```
-- 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：

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

```
1 -- 联合主键
2 -- 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
3 -- primary key (字段1, 字段2....., 字段n)
4 create table emp3(
5     name varchar(20),
6     deptId int,
7     salary double,
8     constraint pk2 primary key(name,deptId)
9 );
```

```

10
11 insert into emp3 values('张三',10,5000);
12 insert into emp3 values('张三',20,5000);
13
14 -- 联合主键的各列 每一列都不能为空
15 insert into emp3 values(NULL,30,5000);
16 insert into emp3 values('赵六',NULL,5000);
17 insert into emp3 values(NULL,NULL,5000);
18 -----
19
20 -- 添加单列主键
21 create table emp4(
22 eid int,
23 name varchar(20),
24 deptId int,
25 salary double
26 );
27
28 alter table emp4 add primary key(eid);
29
30
31 -- 创建多列主键
32 create table emp5(
33 eid int,
34 name varchar(20),
35 deptId int,
36 salary double
37 );
38
39 alter table emp5 add primary key(name,deptId);
40 -----
41 -- 删除主键约束
42 alter table emp5 drop primary key;

```

自增长约束

```

1 -- 自增长约束
2 use database1;
3 create table t_user1(
4 id int primary key auto_increment,
5 name varchar(20)
6 );
7
8 INSERT into t_user1 values(null,'张三');
9 INSERT into t_user1(name) values('李四');
10
11 -- 创建表时指定
12 create table t_user2(
13 id int primary key auto_increment,
14 name varchar(20)
15 )auto_increment=100;
16
17 insert into t_user2 values(null,'张三');
18 insert into t_user2 values(null,'李四');
19 -----

```

```

20
21 -- 创建表之后指定
22 create table t_user3(
23 id int primary key auto_increment,
24 name varchar(20)
25 );
26
27 insert into t_user3 values(null,'张三');
28 insert into t_user3 values(null,'李四');
29 insert into t_user3 values(null,'王五');
30
31 alter table t_user3 auto_increment=100;
32 id name
33 1 张三
34 2 李四
35 100 王五
36 -----
37 delete和truncate在删除后自增列的变化
38
39 delete
40 use database1;
41 create table t_user1(
42 id int primary key auto_increment,
43 name varchar(20)
44 );
45
46 INSERT into t_user1 values(null,'张三');
47 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
48 id name
49 3 张三
50
51
52 truncate
53 create table t_user2(
54 id int primary key auto_increment,
55 name varchar(20)
56 )auto_increment=100;
57
58 insert into t_user2 values(null,'张三');
59 insert into t_user2 values(null,'李四');
60
61 truncate t_user2; -- truncate删除数据之后 自增长从1开始+1
62 id name
63 1 张三

```

非空约束(not null)

```

1 -- 非空约束
2 -- 创建表时指定
3 create table t_user6(
4 id int,
5 name varchar(20) not null,
6 address varchar(20) not null
7 );
8 +-----+-----+-----+-----+-----+

```

```

9 | Field   | Type      | Null | Key | Default | Extra |
10+-----+-----+-----+-----+-----+
11 | id      | int       | YES  |      | NULL    |      |
12 | name    | varchar(20) | NO   |      | NULL    |      |
13 | address | varchar(20) | NO   |      | NULL    |      |
14+-----+-----+-----+-----+-----+
15
16 -- 创建表之后指定
17 alter table t_user3 modify name varchar(20) not null;
18+-----+-----+-----+-----+
19 | Field | Type      | Null | Key | Default | Extra |
20+-----+-----+-----+-----+
21 | id    | int       | NO   | PRI  | NULL    | auto_increment |
22 | name  | varchar(20) | NO   |      | NULL    |      |
23+-----+-----+-----+-----+
24
25 -- 删除非空约束
26 alter table t_user6 modify address varchar(20);
27+-----+-----+-----+-----+
28 | Field | Type      | Null | Key | Default | Extra |
29+-----+-----+-----+-----+
30 | id    | int       | YES  |      | NULL    |      |
31 | name  | varchar(20) | NO   |      | NULL    |      |
32 | address | varchar(20) | YES  |      | NULL    |      |
33+-----+-----+-----+-----+

```

唯一约束

```

1 -- 唯一约束
2 -- 创建表时添加
3 create table t_user8(
4 id int,
5 name varchar(20),
6 phone_number varchar(20) unique
7 );
8+-----+-----+-----+-----+
9 | Field   | Type      | Null | Key | Default | Extra |
10+-----+-----+-----+-----+
11 | id      | int       | YES  |      | NULL    |      |
12 | name    | varchar(20) | YES  |      | NULL    |      |
13 | phone_number | varchar(20) | YES  | UNI  | NULL    |      |
14+-----+-----+-----+-----+
15
16 insert into t_user8 values(1001,'张三',null);
17 insert into t_user8 values(1002,'张三1',null);
18 id      name     phone
19 1001    张三     (NULL)
20 1002    张三1    (NULL)
21 在mysql中NULL和任何值都不相同 NULL不等于NULL
22
23 -- 创建表之后指定
24 alter table t_user6 add constraint UNI unique(address);
25+-----+-----+-----+-----+
26 | Field | Type      | Null | Key | Default | Extra |
27+-----+-----+-----+-----+

```

```

28 | id      | int       | YES  |      | NULL   |      |
29 | name    | varchar(20) | NO   |      | NULL   |      |
30 | address | varchar(20) | YES  | UNI  | NULL   |      |
31 +-----+-----+-----+-----+-----+
32
33 -- 删除唯一约束
34 alter table t_user6 drop index UNI;
35 +-----+-----+-----+-----+-----+
36 | Field  | Type     | Null | Key  | Default | Extra |
37 +-----+-----+-----+-----+-----+
38 | id     | int      | YES  |      | NULL   |      |
39 | name   | varchar(20) | NO   |      | NULL   |      |
40 | address | varchar(20) | YES  |      | NULL   |      |
41 +-----+-----+-----+-----+-----+

```

默认约束

```

1 -- 默认约束
2 -- 创建表时添加
3 create table t_user10(
4 id int,
5 name varchar(20),
6 address varchar(20) default '鄂州'
7 );
8 +-----+-----+-----+-----+-----+
9 | Field  | Type     | Null | Key  | Default | Extra |
10 +-----+-----+-----+-----+-----+
11 | id     | int      | YES  |      | NULL   |      |
12 | name   | varchar(20) | YES  |      | NULL   |      |
13 | address | varchar(20) | YES  |      | 鄂州   |      |
14 +-----+-----+-----+-----+-----+
15
16 -- 创建表之后指定
17 alter table t_user10 modify address varchar(20) default '长职';
18 +-----+-----+-----+-----+-----+
19 | Field  | Type     | Null | Key  | Default | Extra |
20 +-----+-----+-----+-----+-----+
21 | id     | int      | YES  |      | NULL   |      |
22 | name   | varchar(20) | YES  |      | NULL   |      |
23 | address | varchar(20) | YES  |      | 长职   |      |
24 +-----+-----+-----+-----+-----+
25
26 -- 删除默认约束
27 alter table t_user10 modify column address varchar(20) default null;
28 +-----+-----+-----+-----+-----+
29 | Field  | Type     | Null | Key  | Default | Extra |
30 +-----+-----+-----+-----+-----+
31 | id     | int      | YES  |      | NULL   |      |
32 | name   | varchar(20) | YES  |      | NULL   |      |
33 | address | varchar(20) | YES  |      | NULL   |      |
34 +-----+-----+-----+-----+-----+

```

零填充约束

```
1 -- 零填充约束
2 create table t_user12(
3 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
4 name varchar(20)
5 );
6 +-----+-----+-----+-----+
7 | Field | Type          | Null | Key | Default | Extra |
8 +-----+-----+-----+-----+
9 | id    | int(10) unsigned zerofill | YES  |     | NULL    |       |
10 | name   | varchar(20)           | YES  |     | NULL    |       |
11 +-----+-----+-----+-----+
12
13
```

DQL-基本查询

```
1 create table product(
2 pid int primary key auto_increment,
3 pname varchar(20) not null,
4 price double,
5 categroy_id varchar(20)
6 );
7
8 insert into product values(null,'海尔洗衣机',5000,'c001');
9 insert into product values(null,'美的冰箱',3000,'c001');
10 insert into product values(null,'格力电饭煲',5000,'c001');
11 insert into product values(null,'九阳电饭煲',5000,'c001');
12
13 -- 查询所有的商品
14 select * from product;
15
16 -- 查询商品名和商品的价格
17 select pname,price from product;
18
19 -- 别名查询.使用的关键字是as
20 -- 表别名
21 select * from product as p;
22 select *from product p;
23
24 select p.id,u.id from product p, user u;
25
26 -- 列别名
27 select pname as '商品名',price '商品价格' from product;
28
29 -- 去掉重复值
30 select distinct price from product;
31
32 -- 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
33 select pname, price +10 new_price from product;
34
35 -- 将所有商品的价格加价10%进行显示
```

```
36 select pname, price * 1.1 as new_price from product;
37
38 -- 查询商品名称为海尔洗衣机的商品所有信息
39 select * from product where pname = '海尔洗衣机';
40
41 -- 查询价格为5000商品
42 select * from product where price = 5000;
43
44 -- 查询价格不为5000的所有商品
45 select * from product where price != 5000;
46 select * from product where price <> 5000;
47 select * from product where not(price = 5000);
48
49 -- 查询价格大于3000元的所有商品信息
50 select * from product where price > 3000;
51
52 -- 查询价格在3000到5000之间的商品信息
53 select * from product where price >=3000 && price<=5000;
54 select * from product where price between 3000 and 5000;
55
56 -- 查询价格是3000或5000的所有商品
57 select * from product where price = 3000 || price =5000;
58
59 -- 查询含有'格力'的所有商品
60 select * from product where pname like '%格力%';           -- %为任意字符
61
62 -- 查询以'海'开头的所有商品
63 select * from product where pname like '海%';
64
65 -- 查询第二个字为'力'的所有商品
66 select * from product where pname like '_力%';           -- _下划线匹配单个字符
67
68 -- 查询category_id为Null的商品
69 select * from product where category_id is null;
70
71 -- 查询category_id不为null分类的商品
72 select * from product where category_id is not null;
73
74 -- 使用least求最小值
75 select least(10,20,30) as small_number;
76 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
77
78 -- 使用greatest求最大值
79 select greatest(50,20,40) as big_number;
80 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

```
1 -- 排序查询
2 -- 使用价格查询 降序
3 select * from product order by price desc;
4
5 -- 在价格降序的基础上 以分类降序
6 select * from product order by price desc,categroy_id desc;
7
8 -- 显示商品的价格去重 并降序
9 select distinct price from product order by price desc;
```

聚合查询

```
1 -- 聚合查询
2 -- 查询商品的总条目
3 select count(pid) from product;
4 select count(*) from product;
5
6 -- 查询价格大于200商品的总条目
7 select count(pid) from product where price > 4500;
8
9 -- 查询分类为'c001'的所有商品的总和
10 select sum(price) from product where categroy_id ='c001';
11
12 -- 查询商品的最大价格
13 select max(price) from product;
14
15 -- 查询商品的最小价格
16 select min(price) from product;
17
18 -- 查询分类为'c001'所有商品的平均价格
19 select avg(price) from product where categroy_id='c001';
20
21
22 -- 聚合查询null值的处理
23 create table test_null(
24 c1 varchar(20),
25 c2 int
26 );
27
28 insert into test_null values('aaa',3);
29 insert into test_null values('bbb',3);
30 insert into test_null values('ccc',null);
31 insert into test_null values('ddd',6);
32
33 select count(*), count(1), count(c2) from test_null; -- 4 4 3
34 select sum(c2), max(c2), min(c2), avg(c2) from test_null; -- 12 6 3 4
```

分组查询

```
1 -- 分组查询
2 -- select 字段1,字段2 from group by 分组字段 having 分组条件
3 -- 统计各个分类商品的个数
4 -- 分组之后 select的后边只能写分组字段和聚合函数
5 select categroy_id,count(pid) from product group by categroy_id;
6
7 -- where子句用来筛选from子句中指定的操作所产生的行
8 -- group by子句用来分组where子句的输出
9 -- having子句用来从分组的结果中筛选行
10 -- select 字段1,字段2 from 表名 group by 分组字段 having 分组条件;
11
12 -- 统计各个分类商品的个数 且只显示个数大于4的信息
13 select categroy_id,count(*) from product group by categroy_id having count(*) > 1 order by categroy_id;
```

分页查询

```
1 -- 分页查询
2 -- 显示前3条
3 select * from product limit 3;
4
5 -- 从第2条开始显示 显示3条
6 select * from product limit 2,3;
7
8 -- 分页显示
9 select * from product limit 0,60;
10 select * from product limit 60,60;
11 select * from product limit 120,60;
12 select * from product limit (n-1)*60,60;
```

insert into select语句

```
1 -- insert into select
2 select * from product;
3
4 create table product2(
5 pname varchar(20),
6 price double
7 );
8
9 insert into product2(pname,price) select pname,price from product;
10 select * from product2;
11
12 create table product3(
13 categroy_id varchar(20),
14 product_count int
15 );
16
17 insert into product3 select categroy_id , count(*) from product group by categroy_id;
18 select * from product3;
```

练习

```
1 -- 练习
2 create table student(
3     id int,
4     name varchar(20),
5     gender varchar(20),
6     chinese int,
7     english int,
8     math int
9 );
10
11 insert into student values(1, '张明', '男', 89, 78, 90);
12 insert into student values(2, '李进', '男', 67, 53, 95);
13 insert into student values(3, '王五', '女', 87, 78, 77);
14 insert into student values(4, '李一', '女', 88, 98, 92);
15 insert into student values(5, '李财', '男', 82, 84, 67);
16 insert into student values(6, '张宝', '男', 55, 85, 45);
17 insert into student values(7, '黄蓉', '女', 75, 65, 30);
18 insert into student values(7, '黄蓉', '女', 75, 65, 30);
19
20 -- 查询学生表中所有的信息
21 select * from student;
22
23 -- 查询表中所有学生的姓名和对应的英语成绩
24 select name,english from student;
25
26 -- 过滤表中重复数据
27 select distinct * from student;
28
29 -- 统计每个学生的总分
30 select name, chinese + english + math sum from student;
31
32 -- 在所有学生总分数上加10分特长分
33 select name, chinese + english + math +10 from student;
34
35 -- 使用别名表示学生的分数
36 select name, chinese '语文成绩', english '英语成绩', math '数学成绩' from
student;
37
38 -- 查询英语成绩大于90的同学
39 select name from student where english > 90;
40
41 -- 查询总分大于200分的同学
42 select name from student where (chinese + english + math) > 200;
43
44 -- 查询英语分数在80-90之间的同学
45 select name from student where english >= 80 && english <= 90;
46
47 -- 查询英语分数不在80-90之间的同学
48 select name from student where english <80 || english >90;
49
50 -- 查询数学分数为89 90 91的同学
51 select * from student where math in(89,90,91);
52 select name from student where math=89 || math=90 || math=91;
```

```
53
54 -- 查询数学分数不为89 90 91的同学
55 select * from student where math not in(89,90,91);
56 select * from student where not math in(89,90,91);
57
58 -- 查询所有姓李的学生的英语成绩
59 select name,english from student where name like '李%';
60
61 -- 查询数学分80并且语文分80的同学
62 select * from student where math=80 && chinese=80;
63
64 -- 查询英语80或者总分200的同学
65 select * from student where english=80 || (chinese + english + math)=200;
66
67 -- 对数学成绩降序
68 select * from student order by math desc;
69
70 -- 对总分降序排序
71 select * from student order by (chinese + english + math) desc;
72
73 -- 对姓李的学生成绩排序输出
74 select * from student where name like '李%' order by (chinese + english + math) desc;
75
76 -- 查询男生和女生分别又多少人 并将人数降序输出
77 select gender,count(*) as total_cnt from student group by gender order by total_cnt desc;
78
79 -- 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
80 select gender,count(*) as total_cnt from student group by gender having total_cnt >4 order by total_cnt desc;
81
82 -----
83 create table emp(
84 empno int,
85 ename varchar(50),
86 job varchar(50),
87 mgr int,
88 hiredate date,
89 sal int,
90 comm int,
91 deptno int
92 );
93
94 insert into emp values(7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20),
95 (7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300,30),
96 (7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30),
97 (7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20),
98 (7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30),
99 (7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30),
```

```
100      (7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10),
101      (7788,'SCOTT','ANALYST',7566,'1987-04-19',3000,NULL,20),
102          (7839,'KING','PRESIDENT',NULL,'1981-11-
103          17',5000,NULL,10),
104          (7844,'TURNER','SALESMAN',7698,'1981-09-
105          08',1500,0,30),
106          (7876,'ADAMS','CLERK',7788,'1987-05-23',1100,NULL,20),
107          (7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30),
108          (7902,'FORD','ANALYST',7566,'1981-12-
109          03',3000,NULL,20),
110          (7934,'MILLER','CLERK',7782,'1982-01-
111          23',1300,NULL,10);
112      -- 按员工编号升序排列不在10号部门工作的员工信息
113      select * from emp where deptno != 10 order by empno;
114
115      -- 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
116      -- ifnull(sal,0) 如果sal的值为null 则当作0 不为null 则还是原来的值
117      select * from emp where ename not like '_A%' and sal > 1000 order by
118          (12*sal+ifnull(comm,0)) desc;
119
120      -- 求每个部门的平均薪水
121      select deptno,avg(sal) from emp group by deptno;
122      select deptno,avg(sal) as avg_sal from emp group by deptno order by avg_sal
123      desc;
124
125      -- 求每个部门的最高薪水
126      select deptno,max(sal) from emp group by deptno;
127
128      -- 求每个部门每个岗位的最高薪水
129      select deptno,job,max(sal) from emp group by deptno,job order by deptno;
130
131      -- 求平均薪水大于2000的部门编号
132      select deptno,avg(sal) avg_sal from emp group by deptno having avg_sal >
133          2000;
134
135      -- 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
136      select deptno,avg(sal) avg_sal from emp group by deptno having avg_sal >
137          1500 order by avg_sal desc;
138
139      -- 选择公司中有奖金的员工姓名 工资
140      select ename,sal from emp where comm is not null;
141
142      -- 查询员工最高工资和最低工资的差距
143      select max(sal) - min(sal) as cha from emp;
```

| 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)oold' 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。 |

```

1 -- 正则表达式
2 -- ^ 在字符串开始处进行匹配
3 select 'abc' regexp '^a';
4 select * from student where name regexp '^张';
5
6 -- $ 在字符串末尾开始匹配
7 select 'abc' regexp 'a$';
8 select 'abc' regexp 'c$';
9 select * from student where name regexp '宝$';
10
11 -- . 匹配任意字符 可以匹配除换行符外的任意字符
12 select 'abc' regexp '.b';-- 1
13 select 'abc' regexp '.c';-- 1

```

```
14 select 'abc' regexp 'a.';-- 1
15
16 -- [...] 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
17 select 'abc' regexp '[xyz]'; -- 0
18 select 'abc' regexp '[xaz]'; -- 1
19
20 -- [^...] 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
21 select 'a' regexp '[^abc]'; -- 0
22 select 'x' regexp '[^abc]'; -- 1
23 select 'abc' regexp '[^a]'; -- 1
24
25 -- 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
26 select 'stab' regexp '.ta*b'; -- 1
27 select 'stb' regexp '.ta*b'; -- 1
28 select '' regexp 'a*'; -- 1
29
30 -- a+ 匹配1个或者多个a 但是不包括空字符串
31 select 'stab' regexp '.ta+b'; -- 1
32 select 'stb' regexp '.ta+b'; -- 0
33
34 -- a? 匹配0个或者1个a
35 select 'stb' regexp '.ta?b'; -- 1
36 select 'stab' regexp '.ta?b'; -- 1
37 select 'staab' regexp '.ta?b'; -- 0
38
39 -- a1|a2 匹配a1或者a2
40 select 'a' regexp 'a|b'; -- 1
41 select 'b' regexp 'a|b'; -- 1
42 select 'b' regexp '^ (a|b)'; -- 1
43 select 'a' regexp '^ (a|b)'; -- 1
44 select 'c' regexp '^ (a|b)'; -- 0
45
46 -- a{m} 匹配m个a
47 select 'auuuuc' regexp 'au{4}c'; -- 1
48 select 'auuuuc' regexp 'au{3}c'; -- 0
49
50 -- a{m,} 匹配m个或者更多个a
51 select 'auuuuc' regexp 'au{3,}c'; -- 1
52 select 'auuuuc' regexp 'au{4,}c'; -- 1
53 select 'auuuuc' regexp 'au{5,}c'; -- 0
54
55 -- a{m,n} 匹配m到n个a 包含m和n
56 select 'auuuuc' regexp 'au{3,5}c'; -- 1
57 select 'auuuuc' regexp 'au{4,5}c'; -- 1
58 select 'auuuuc' regexp 'au{5,10}c'; -- 0
59
60 -- (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
61 select 'xababy' regexp 'x(abab)y'; -- 1
62 select 'xababy' regexp 'x(ab)*y'; -- 1
63 select 'xababy' regexp 'x(ab){1,2}y'; -- 1
64 select 'xababy' regexp 'x(ab){3}y'; -- 0
```

MySQL的多表操作

外键约束-一对多

```
1 -- 创建部门表 主表
2 create table if not exists dept(
3     deptno varchar(20) primary key,
4     name varchar(20)
5 );
6
7 -- 创建员工表 并创建dept_id 外键约束
8 create table if not exists emp(
9     eid varchar(20) primary key,
10    ename varchar(20),
11    age int,
12    dept_id varchar(20),
13    -- 创建表外键约束
14    constraint emp_fk foreign key (dept_id) references dept (deptno)
15 );
16
17 -- 创建表之后添加外键约束
18 create table if not exists dept2(
19     deptno varchar(20) primary key,
20     name varchar(20)
21 );
22
23 create table if not exists emp2(
24     eid varchar(20) primary key,
25     ename varchar(20),
26     age int,
27     dept_id varchar(20)
28 );
29
30 -- 创建外键约束
31 alter table emp2 add constraint dept_id_fk foreign key(dept_id) references
dept2(deptno);
32
33 -- 删除外键约束
34 alter table emp2 drop foreign key dept_id_fk;
```

外键约束-多对多

```
1 -- 学生表和课程表（多对多）
2 -- 创建学生表student(左侧主表)
3 create table if not exists student(
4     sid int primary key auto_increment,
5     name varchar(20),
6     age int,
7     gender varchar(20)
8 );
9
10 -- 创建课程表course (右侧主表)
11 create table course(
12     cid int primary key auto_increment,
```

```

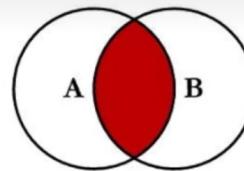
13  cidname varchar(20)
14  );
15
16  -- 修改和删除时 中间从表可以随便删除和修改 但是两边的主表受从表依赖的数据不能删除或者修改
17  -- 创建中间表student_course/score(从表)
18  create table score(
19    sid int,
20    cid int,
21    score double
22  );
23
24  -- 建立外键约束
25  alter table score add foreign key(sid) references student(sid);
26  alter table score add foreign key(cid) references course(cid);
27
28  -- 给学生表添加数据
29  insert into student values(1,'小龙女',18,'女'),(2,'阿紫',19,'女'),(3,'周芷若',20,'男');
30
31  -- 给课程表添加数据
32  insert into course values(1,'语文'),(2,'数学'),(3,'英语');
33
34  -- 给中间表添加数据
35  insert into score values(1,1,78),(1,2,79),(2,1,80),(2,3,81),(3,2,82),
(3,3,83);

```

多表联合查询-内连接查询

◆ 内连接查询

内连接查询求多张表的交集



```

SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key

```

➤ 格式

隐式内连接 (SQL92标准) : **select * from A,B where 条件;**

显示内连接 (SQL99标准) : **select * from A inner join B on 条件;**

➤ 操作

-- 查询每个部门的所属员工

```

select * from dept3,emp3 where dept3.deptno = emp3.dept_id;
select * from dept3 inner join emp3 on dept3.deptno =
emp3.dept_id;

```

```

1  -- 创建部门表
2  create table if not exists dept3(
3    deptno varchar(20) primary key,
4    name varchar(20)
5  );
6
7  -- 创建员工表
8  create table if not exists emp3(
9    eid varchar(20) primary key,

```

```
10    ename varchar(20),  
11    age int,  
12    dept_id varchar(20)  
13 );  
14  
15 -- 给dept3表添加数据  
16 insert into dept3 values('1001','研发部');  
17 insert into dept3 values('1002','销售部');  
18 insert into dept3 values('1003','财务部');  
19 insert into dept3 values('1004','人事部');  
20  
21 -- 给emp表添加数据  
22 insert into emp3 values('1','乔峰',20,'1001');  
23 insert into emp3 values('2','段誉',21,'1001');  
24 insert into emp3 values('3','虚竹',23,'1001');  
25 insert into emp3 values('4','阿紫',18,'1001');  
26 insert into emp3 values('5','扫地僧',85,'1002');  
27 insert into emp3 values('6','李秋水',33,'1002');  
28 insert into emp3 values('7','鸠摩智',50,'1002');  
29 insert into emp3 values('8','天山童姥',60,'1003');  
30 insert into emp3 values('9','慕容博',58,'1003');  
31 insert into emp3 values('10','丁春秋',71,'1005');  
32  
33 -- 交叉连接查询  
34 select * from dept3,emp3;  
35  
36 -- 内连接查询  
37 -- 隐式内连接 select * from A,B where 条件;  
38 -- 显示内连接 select * from A inner join B on 条件;  
39  
40 -- 查询每个部门的所属员工  
41 -- 隐式内连接  
42 select * from dept3,emp3 where dept3.deptno = emp3.dept_id;  
43 select * from dept3 a,emp3 b where a.deptno = b.dept_id;  
44  
45 -- 显式连接  
46 select * from dept3 inner join emp3 on dept3.deptno = emp3.dept_id;  
47 select * from dept3 a join emp3 b on a.deptno = b.dept_id;  
48  
49  
50 -- 查询研发部门的所属员工  
51 select * from dept3 a,emp3 b where a.deptno = b.dept_id;  
52 -- 隐式内连接  
53 select * from dept3 a,emp3 b where a.deptno = b.dept_id and name ='研发部';  
54  
55 -- 显示内连接  
56 select * from dept3 a join emp3 b on a.deptno = b.dept_id and name ='研发部';  
57  
58  
59 -- 查询研发部和销售部的所属员工  
60 -- 隐式内连接  
61 select * from dept3 a,emp3 b where a.deptno = b.dept_id and name in ('研发  
部','销售部');  
62  
63 -- 显示内连接
```

```

64 select * from dept3 a join emp3 b on a.deptno = b.dept_id and (name = '研发部'
or name = '销售部');
65 select * from dept3 a join emp3 b on a.deptno = b.dept_id and name in ('研发
部','销售部');
66
67
68 -- 查询每个部门的员工数 并升序排列
69 -- 隐式内连接
70 select a.name,a.deptno,count(1) from dept3 a,emp3 b where a.deptno =
b.dept_id group by a.deptno;
71
72 -- 显示内连接
73 select a.name,a.deptno,count(1) from dept3 a join emp3 b on a.deptno
=b.dept_id group by a.deptno;
74
75 -- 查询人数大于等于3的部门 并按照人数降序排序
76 -- 隐式内连接
77 select a.deptno,a.name,count(1) as total_cnt from dept3 a,emp3 b where
a.deptno = b.dept_id group by a.deptno,a.name having total_cnt >=3 order by
total_cnt desc;
78
79 -- 显示内连接
80 select a.deptno,a.name,count(1) as total_cnt from dept3 a join emp3 b on
a.deptno = b.dept_id group by a.deptno,a.name having total_cnt >=3 order by
total_cnt desc;

```

多表联合查询-外连接查询

◆ 外连接查询

外连接分为左外连接(left outer join)、右外连接(right outer join), 满外连接(full outer join)。

注意: oracle里面有full join, 可是在mysql对full join支持的不好。我们可以使用union来达到目的。

➤ 格式

左外连接: left outer join

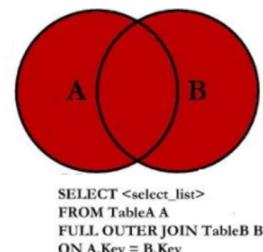
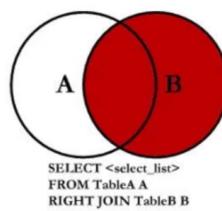
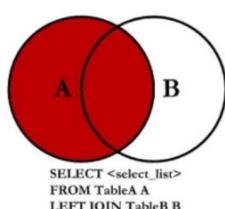
select * from A left outer join B on 条件;

右外连接: right outer join

select * from A right outer join B on 条件;

满外连接: full outer join

select * from A full outer join B on 条件;



```

1 -- 外连接查询
2 -- 查询哪些部门有员工 哪些部门没有员工
3 select * from dept3 left outer join emp3 on dept3.deptno = emp3.dept_id;
4
5 select * from A
6 left join B on 条件1
7 left join C on 条件2
8 left join D on 条件3;

```

```

9
10 -- 查询哪些员工有对应的部门 哪些没有
11 select * from dept3 right outer join emp3 on dept3.deptno = emp3.dept_id;
12
13 select * from A
14 right join B on 条件1
15 right join C on 条件2
16 right join D on 条件3;
17
18 -- 使用union关键字实现左外连接和右外连接的并集
19 select * from dept3 left outer join emp3 on dept3.deptno = emp3.dept_id
20 union
21 select * from dept3 right outer join emp3 on dept3.deptno = emp3.dept_id;

```

多表联合查询-子查询

◆ 子查询

➤ 介绍

子查询就是指的在一个完整的查询语句之中，嵌套若干个不同功能的小查询，从而一起完成复杂查询的一种编写形式，通俗一点就是包含select嵌套的查询。



➤ 特点

子查询可以返回的数据类型一共分为四种：

- 1.单行单列：返回的是一个具体列的内容，可以理解为一个单值数据；
- 2.单行多列：返回一行数据中多个列的内容；
- 3.多行单列：返回多行记录之中同一列的内容，相当于给出了一个操作范围；
- 4.多行多列：查询返回的结果是一张临时表

```

1 -- 查询年龄最大的员工信息 显示信息包含员工号 员工名字 员工年龄
2 -- 1.查询最大年龄:
3 select max(age) from emp3;
4
5 -- 2.让每一个员工的年龄和最大年龄进行比较 相等则满足条件
6 select * from emp3 where age = (select max(age) from emp3); -- 单行单列 可以作为一个值来用
7
8 -- 查询研发部和销售部得到员工信息 包含员工号 员工名字
9 -- 方式1-关联查询
10 select * from dept3 a join emp3 b on a.deptno = b.dept_id and (name = '研发部'
11 or name = '销售部');
12
13 -- 方式2-子查询
14 -- 先查询研发部和销售部的部门号 deptno
15 select deptno from dept3 where name = '研发部' or name = '销售部';
16
17 -- 查询哪个员工的部门号是100 或者 1002
18 select * from emp3 where dept_id in (select deptno from dept3 where name =
19 '研发部' or name = '销售部'); -- 多行单列多个值
20
21 -- 查询研发部20岁以下的员工信息 包括员工号 员工名字 部门名字
22 -- 方式1 关联查询
23 select * from dept3 a join emp3 b on a.deptno = b.dept_id and (name = '研发部'
24 and age < 20);
25
26 -- 方式2 子查询

```

```

24 -- 在部门表中查询研发部信息
25 select * from dept3 where name = '研发部'; -- 一行多列
26
27 -- 在员工表中查询年龄小于20岁的员工信息
28 select * from emp3 where age < 30;
29
30 -- 将以上两个查询的结果进行关联查询
31 select * from (select * from dept3 where name = '研发部') t1 join (select *
from emp3 where age < 30) t2 on t1.deptno = t2.dept_id; -- 多行多列

```

多表联合查询-子查询关键字

子查询关键字-ALL

◆ 子查询关键字-ALL

➤ 格式

```

select ...from ...where c > all(查询语句)
--等价于:
select ...from ... where c > result1 and c > result2 and c > result3

```

➤ 特点

- ALL: 与子查询返回的所有值比较为true 则返回true
- ALL可以与=、>、>=、<、<=、<>结合使用，分别表示等于、大于、大于等于、小于、小于等于、不等于其中的所有的数据。
- ALL表示指定列中的值必须要大于子查询集的每一个值，即必须要大于子查询集的最大值；如果是小于号即小于子查询集的最小值。同理可以推出其它的比较运算符的情况。

➤ 操作

```

-- 查询年龄大于'1003'部门所有年龄的员工信息
select * from emp3 where age > all(select age from emp3 where dept_id = '1003');
-- 查询不属于任何一个部门的员工信息
select * from emp3 where dept_id != all(select deptno from dept3);

```

```

1 -- 子查询关键字All
2 -- 查询年龄大于'1003'部门所有年龄的员工信息
3 select * from emp3 where age >all(select age from emp3 where dept_id =
'1003');
4
5 -- 查询不属于任何一个部门的员工信息
6 select * from emp3 where dept_id !=all(select deptno from dept3);

```

子查询关键字-any和some

◆ 子查询关键字-ANY和SOME

➤ 格式

```
select ...from ...where c > any(查询语句)
--等价于:
select ...from ... where c > result1 or c > result2 or c > result3
```

➤ 特点

- ANY:与子查询返回的任何值比较为true 则返回true
- ANY可以与=、>、>=、<、<=、<>结合是来使用， 分别表示等于、大于、大于等于、小于、小于等于、不等于其中的任何一个数据。
- 表示制定列中的值要大于子查询中的任意一个值，即必须要大于子查询集中的最小值。同理可以推出其它的比较运算符的情况。
- SOME和ANY的作用一样， SOME可以理解为ANY的别名

➤ 操作

-- 查询年龄大于'1003'部门任意一个员工年龄的员工信息

```
select * from emp3 where age > any(select age from emp3 where dept_id = '1003');
```

```
1 -- 子查询关键字-any和some
```

```
2 -- 查询年龄大于'1003'部门任意一个员工年龄的员工信息
```

```
3 select * from emp3 where age > any(select age from emp3 where dept_id =
  '1003') and dept_id != '1003';
```

子查询关键字-in

◆ 子查询关键字-IN

➤ 格式

```
select ...from ...where c in(查询语句)
--等价于:
select ...from ... where c = result1 or c = result2 or c = result3
```

➤ 特点

- IN关键字，用于判断某个记录的值，是否在指定的集合中
- 在IN关键字前边加上not可以将条件反过来



➤ 操作

-- 查询研发部和销售部的员工信息，包含员工号、员工名字

```
select eid,ename,t.name from emp3 where dept_id in (select deptno from dept3
where name = '研发部' or name = '销售部') ;
```

```
1 -- 子查询关键字-in
```

```
2 -- 查询研发部和销售部的员工信息 包含员工号 员工名字
```

```
3 select eid,ename from emp3 where dept_id in (select deptno from dept3 where
name ='研发部' or name ='销售部');
```

子查询关键字-EXISTS

子查询关键字-EXISTS

➤ 格式

```
select ...from ...where exists(查询语句)
```

➤ 特点

- 该子查询如果“有数据结果”（至少返回一行数据），则该EXISTS()的结果为“true”，外层查询执行
- 该子查询如果“没有数据结果”（没有任何数据返回），则该EXISTS()的结果为“false”，外层查询不执行
- EXISTS后面的子查询不返回任何实际数据，只返回真或假，当返回真时 where条件成立
- 注意，EXISTS关键字，比IN关键字的运算效率高，因此，在实际开发中，特别是大数据量时，推荐使用EXISTS关键字

➤ 操作

```
-- 查询公司是否有大于60岁的员工，有则输出
select * from emp3 a where exists(select * from emp3 b where a.age > 60);

-- 查询有所属部门的员工信息
select * from dept3 a where exists(select * from emp3 b where a.deptno =
b.dept_id);
```

```
1 -- 子查询关键字-EXISTS
2 -- select ... from ...where exists(查询语句)
3 select * from emp3 where exists(select 1); -- 全表输出
4 select * from emp3 where exists(select * from emp3); -- 全表输出
5
6 -- 查询公司是否有大于60岁的员工 有则输出
7 select * from emp3 a where exists ( select * from emp3 where a.age > 60);
8 select * from emp3 a where eid in ( select eid from emp3 where a.age > 60);
9
10 -- 查询有所属部门的员工信息
11 select * from emp3 a where exists ( select * from dept3 b where a.dept_id =
b.deptno);
12 select * from emp3 a where dept_id in ( select deptno from dept3 b where
a.dept_id = b.deptno);
```

多表联合查询-自关联查询

◆ 自关联查询

➤ 概念

MySQL有时在信息查询时需要进行对表自身进行关联查询，即一张表自己和自己关联，一张表当成多张表来用。注意自关联时表必须给表起别名。

➤ 格式

```
select 字段列表 from 表1 a , 表1 b where 条件;
或者
select 字段列表 from 表1 a [left] join 表1 b on 条件;
```

➤ 操作

```
-- 创建表，并建立自关联约束
create table t_sanguo(
    eid int primary key ,
    ename varchar(20),
    manager_id int,
    foreign key (manager_id) references t_sanguo (eid) -- 添加自关联约束
);
```

```
1 -- 多表联合查询-自关联查询
```

```

2  -- 创建表 并建立自关联查询
3  create table t_sanguo(
4    eid int primary key,      -- 主键列
5    ename varchar(20),
6    manager_id int,        -- 外键列
7    foreign key(manager_id) references t_sanguo(eid)  -- 添加自关联约束
8  );
9
10 -- 添加数据
11 insert into t_sanguo values(1,'刘备',NULL);
12 insert into t_sanguo values(2,'关羽',1);
13 insert into t_sanguo values(3,'张飞',2);
14 insert into t_sanguo values(4,'曹操',1);
15 insert into t_sanguo values(5,'孙权',1);
16 insert into t_sanguo values(6,'许褚',5);
17 insert into t_sanguo values(7,'典韦',5);
18 insert into t_sanguo values(8,'周瑜',8);
19 insert into t_sanguo values(9,'鲁肃',8);
20 insert into t_sanguo values(10,'诸葛亮',8);
21
22 -- 进行关联查询
23 -- 查询每个三国人物及他的上级信息 如 关羽 刘备
24 select * from t_sanguo a,t_sanguo b where a.manager_id=b.eid;
25 select a.ename,b.ename from t_sanguo a,t_sanguo b where a.manager_id=b.eid;
26 select a.ename,b.ename from t_sanguo a join t_sanguo b on a.manager_id=b.eid;
27
28 -- 查询所有人物及上级
29 select a.ename,b.ename from t_sanguo a left join t_sanguo b on
30 a.manager_id=b.eid;
31 -- 查询所有任务 上级 上上级
32 select
33   a.ename,b.ename,c.ename
34   from t_sanguo a
35   left join t_sanguo b on a.manager_id=b.eid
36   left join t_sanguo c on b.manager_id=c.eid;

```

练习

```

1  -- 创建部门表
2  create table dept(
3    deptno int primary key,
4    dname varchar(14),
5    loca varchar(13)
6  );
7
8  insert into dept values(10,'accounting','new york');
9  insert into dept values(20,'research','dallas');
10 insert into dept values(30,'sales','chicago');
11 insert into dept values(40,'operations','boston');
12
13 -- 创建员工表
14 create table emp(
15   empno int primary key,
16   ename varchar(10),

```

```
17    jno varchar(9),  
18    mgr int,  
19    hiredate date,  
20    sal double,  
21    comm double,  
22    deptno int  
23 );  
24  
25 -- 添加 部门和员工 之间的主外键关系  
26 alter table emp add constraint foreign key emp(deptno) references  
dept(deptno);  
27  
28 insert into emp values(7369, 'smith' , 'clerk',7902,'1980-12-  
17',800,null,20);  
29 insert into emp values(7499, 'allen' , 'salesman' ,7698,'1981-02-  
20',1600,300,30);  
30 insert into emp values(7521, 'ward' , 'salesman',7698,'1981-02-  
22',1250,500,30);  
31 insert into emp values(7566, 'jones' , 'manager',7839,'1981-04-  
02',2975,null,20);  
32 insert into emp values(7654, 'martin' , 'salesman',7698,'1981-09-  
28',1250,1400,30);  
33 insert into emp values(7698, 'blake' , 'manager' ,7839,'1981-05-01  
' ,2850,null,30);  
34 insert into emp values(7782, 'clark' , 'manager',7839,'1981-06-  
09',2450,null,10);  
35 insert into emp values(7788, 'scott' , 'analyst',7566,'1987-07-03' ,  
3000,null,20);  
36 insert into emp values(7839, 'king' , 'president' , null,'1981-11-  
17',5000,null,10);  
37 insert into emp values(7844, 'turner' , 'salesman' ,7698,'1981-09-  
08',1500,0,30);  
38 insert into emp values(7876, 'adams' , 'clerk',7788,'1987-07-  
13',1100,null,20);  
39 insert into emp values(7900, 'james' , 'clerk',7698,'1981-12-03'  
,950,null,30);  
40 insert into emp values(7902, 'ford' , 'analyst' ,7566,'1981-12-  
03',3000,null,20);  
41 insert into emp values(7934, 'miller' , 'clerk',7782,'1981-01-  
23',1300,null,10);  
42  
43 -- 创建工资等级表  
44 create table salgrade(  
45 grade int,  
46 losal double,  
47 hisal double  
);  
48  
49  
50 insert into salgrade values(1,700,1200);  
51 insert into salgrade values(2,1201,1400);  
52 insert into salgrade values(3,1401,2000);  
53 insert into salgrade values(4,2001,3000);  
54 insert into salgrade values(5,3001,9999);  
55  
56 -- 返回拥有员工的部门名 部门号
```

```
57 select distinct d.dname,d.deptno from dept d join emp e on d.deptno =
e.deptno;
58
59 -- 工资水平多于smith的员工信息
60 select * from emp where sal > (select sal from emp where ename = 'smith');
61
62 -- 返回员工和所属经理的姓名
63 select a.ename,b.ename from emp a,emp b where a.mgr = b.empno;
64
65 -- 返回雇员的雇佣日期早于其经理雇佣日期的员工及其经理姓名
66 select a.ename,a.hiredate,b.ename,b.hiredate from emp a,emp b where a.mgr =
b.empno and a.hiredate < b.hiredate;
67
68 -- 返回员工姓名及其所在的部门名称
69 select e.ename,d.dname from emp e,dept d where e.deptno = d.deptno;
70
71 -- 返回从事clerk工作的员工姓名和所在部门名称
72 select e.ename,d.dname,e.job from emp e,dept d where e.deptno =d.deptno and
e.job = 'clerk';
73
74 -- 返回部门号及其本部门的最低工资
75 select deptno,min(sal) from emp group by deptno;
76
77 -- 返回销售部(sales)所有员工的姓名
78 select e.ename from emp e,dept d where e.deptno = d.deptno and d.dname =
'sales';
79
80 -- 返回工资水平多于平均工资的员工
81 select * from emp where sal > (select avg(sal) from emp);
82
83 -- 返回与scott从事相同工作的员工
84 select * from emp where job = (select job from emp where ename = 'scott')
and ename != 'scott';
85
86 -- 返回工资高于30部门所有员工工资水平的员工信息
87 select * from emp where sal > all(select sal from emp where deptno=30);
88
89 -- 返回员工工作及其从事此工作的最低工资
90 select job,min(sal) from emp group by job;
91
92 -- 计算出员工的年薪 并且以年薪降序排列
93 select ename,(sal *12 +ifnull(comm,0)) as year_sal  from emp order by (sal
*12 +ifnull(comm,0)) desc;    -- 非空ifnull
94
95 -- 返回工资处于第四级别的员工的姓名
96 select ename from emp where sal between (select losal from salgrade where
grade = 4) and (select hisal from salgrade where grade =4);
97
98 -- 返回工资为二级别的职员名字 部门所在地
99 select * from dept d join emp e on e.deptno = d.deptno join salgrade s on
grade =2 and e.sal >= s.loSal and e.sal <=s.hisal;
100
101 select * from dept d,emp e,salgrade s where e.deptno = d.deptno and grade =
2 and e.sal >=s.loSal and e.sal <= s.hisal;
```

MySQL的函数

mysql的函数-聚合函数

MySQL的函数-聚合函数

◆ 概述

- 在MySQL中，聚合函数主要由：count,sum,min,max,avg,这些聚合函数我们之前都学过，不再重复。这里我们学习另外一个函数:group_concat()，该函数用户实现行的合并
- group_concat()函数首先根据group by指定的列进行分组，并且用分隔符分隔，将同一个分组中的值连接起来，返回一个字符串结果。

◆ 格式

```
group_concat([distinct] 字段名 [order by 排序字段 asc/desc] [separator '分隔符'])
```

说明：

- (1) 使用distinct可以排除重复值；
- (2) 如果需要对结果中的值进行排序，可以使用order by子句；
- (3) separator是一个字符串值，默认为逗号。

```
1 create table emp(
2     emp_id int primary key auto_increment comment '编号',
3     emp_name char(20) not null default '' comment '姓名',
4     salary decimal(10,2) not null default 0 comment '工资',
5     department char(20) not null default '' comment '部门'
6 );
7
8 insert into emp(emp_name,salary,department) values('张晶晶',5000,'财务部'),('王
9 飞飞',5800,'财务部'),('赵刚',6200,'财务部'),
10 ('刘小贝',5700,'人事部'),('王大鹏',6700,'人事部'),('张小斐',5200,'人事部'),('刘云
11 云',7500,'销售部'),('刘云鹏',7200,'财务部'),('刘云鹏',7800,'财务部');
12
13 -- 将所有员工的名字合并成一行
14 select GROUP_CONCAT(emp_name) from emp;
15
16 -- 指定分隔符合并
17 select GROUP_CONCAT(emp_name separator ';') from emp;
18
19 -- 指定排序方式和分隔符
20 select department,group_concat(emp_name separator ';') from emp group by
department;
21 select department,group_concat(emp_name order by salary desc separator ';')
from emp group by department;
```

mysql函数-数学函数

MySQL的函数-数学函数

| 函数名 | 描述 | 实例 |
|------------------------------------|-----------------|--|
| ABS(x) | 返回 x 的绝对值 | 返回 -1 的绝对值： SELECT ABS(-1) -- 返回1 |
| CEIL(x) | 返回大于或等于 x 的最小整数 | SELECT CEIL(1.5) -- 返回2 |
| FLOOR(x) | 返回小于或等于 x 的最大整数 | 小于或等于 1.5 的整数： SELECT FLOOR(1.5) -- 返回1 |
| GREATEST(expr1, expr2, expr3, ...) | 返回列表中的最大值 | 返回以下数字列表中的最大值： SELECT GREATEST(3, 12, 34, 8, 25); -- 34 返回以下字符串列表中的最大值： SELECT GREATEST("Google", "Runoob", "Apple"); -- Runoob |
| LEAST(expr1, expr2, expr3, ...) | 返回列表中的最小值 | 返回以下数字列表中的最小值： SELECT LEAST(3, 12, 34, 8, 25); -- 3 返回以下字符串列表中的最小值： SELECT LEAST("Google", "Runoob", "Apple"); -- Apple |

```
1 -- 数学函数
2 -- 绝对值
3 select abs(-10);
4 select abs(表达式或者字段) from 表;
5
6 -- 向上取整
7 select ceil(1.1); -- 2
8 select ceil(1.0); -- 1
9
10 -- 向下取整
11 select floor(1.1); -- 1
12 select floor(1.9); -- 1
13
14 -- 取列表最大值
15 select greatest(1,2,3); -- 3
16
17 -- 取列表最小值
18 select least(1,2,3); -- 1
19
20 -- 求余数
21 select mod(5,2); -- 1
22
23 -- 取x的y次方
24 select power(2,3); -- 8
25
26 -- 取随机数
27 -- 0~100之间
28 select floor(rand()*100);
29
30 -- 取小数的四舍五入
31 select round(3.5415);
32 -- 保留三位小数
33 select round(3.5415,3)
```

mysql函数-字符串函数

MySQL的函数-字符串函数

| 函数 | 描述 | 实例 |
|--------------------------|---|--|
| CHAR_LENGTH(s) | 返回字符串 s 的字符数 | 返回字符串 RUNOOB 的字符数 SELECT CHAR_LENGTH("RUNOOB") AS LengthOfString; |
| CHARACTER_LENGTH(s) | 返回字符串 s 的字符数 | 返回字符串 RUNOOB 的字符数 SELECT CHARACTER_LENGTH("RUNOOB") AS LengthOfString; |
| CONCAT(s1,s2...sn) | 字符串 s1,s2 等多个字符串合并为一个字符串 | 合并多个字符串 SELECT CONCAT("SQL ", "Runoob ", "Google ", "Facebook") AS ConcatenatedString; |
| CONCAT_WS(x, s1,s2...sn) | 同 CONCAT(s1,s2,...) 函数，但是每个字符串之间要加上 x, x 可以是分隔符 | 合并多个字符串，并添加分隔符： SELECT CONCAT_WS("-", "SQL", "Tutorial", "is", "fun!") AS ConcatenatedString; |
| FIELD(s,s1,s2...) | 返回第一个字符串 s 在字符串列表(s1,s2...)中的位置 | 返回字符串 c 在列表值中的位置： SELECT FIELD("c", "a", "b", "c", "d", "e"); |

| 函数 | 描述 | 实例 |
|-----------------------------|--|---|
| RIGHT(s,n) | 返回字符串 s 的后 n 个字符 | 返回字符串 runoob 的后两个字符： SELECT RIGHT('runoob',2) -- ob |
| RTRIM(s) | 去掉字符串 s 结尾处的空格 | 去掉字符串 RUNOOB 的末尾空格： SELECT RTRIM("RUNOOB ") AS RightTrimmedString; -- RUNOOB |
| STRCMP(s1,s2) | 比较字符串 s1 和 s2，如果 s1 与 s2 相等返回 0，如果 s1>s2 返回 1，如果 s1<s2 返回 -1 | 比较字符串： SELECT STRCMP("runoob", "runoob"); -- 0 |
| SUBSTR(s, start, length) | 从字符串 s 的 start 位置截取长度为 length 的子字符串 | 从字符串 RUNOOB 中的第 2 个位置截取 3 个字符： SELECT SUBSTR("RUNOOB", 2, 3) AS ExtractString; -- UNO |
| SUBSTRING(s, start, length) | 从字符串 s 的 start 位置截取长度为 length 的子字符串 | 从字符串 RUNOOB 中的第 2 个位置截取 3 个字符： SELECT SUBSTRING("RUNOOB", 2, 3) AS ExtractString; -- UNO |

```
1 -- 字符串函数
2 -- 获取字符串字符个数
3 select char_length('hello'); -- 5
4 select char_length('长职'); -- 2
5
6 -- length 取长度 返回的单位是字节
7 select length('hello'); -- 5
8 select length('长职'); -- 6
9
10 -- 字符串合并
11 select concat('hello', 'world');
12 select concat(c1,c2) from table_name;
13
14 -- 指定分隔符进行字符串合并
15 select concat_ws('-', 'hello', 'world');
16
17 -- 返回字符串在列表中第一次出现的位置
18 select field('aa', 'aa', 'bb', 'cc');
19
20 -- 去除字符串左边空格
21 select ltrim(' aaaa');
```

```

22 -- 去除字符串右边空格
23 select rtrim(' aaaa      ');
24 -- 去除两端空格
25 select trim(' aaaaa ');
26
27 -- 字符串截取
28 select mid("helloworld",2,3); -- 从第二个字符开始截取 截取长度为3
29
30 -- 获取字符串A在字符串中出现的位置
31 select position('abc' in 'hellabcoworld');
32
33 -- 字符串替换
34 select replace('helloaaaworld','aaa','bbb');
35
36 -- 字符串反转
37 select reverse('abc');
38
39 -- 返回字符串后几个字符
40 select right('hello',3); -- 返回后3个字符
41
42 -- 字符串比较
43 select strcmp('hello','world'); -- -1 world比hello大
44
45 -- 字符串截取
46 select substr('hello',2,3); -- 从第二个字符开始截取 截取三个字符
47 select substring('hello',2,3); -- 从第二个字符开始截取 截取三个字符
48
49 -- 将小写转大写
50 select ucase('helloworld');
51 select upper('helloworld');
52
53 -- 将大写转为小写
54 select lcase('HELLOWORLD');
55 select lower('HELLOWORLD');

```

mysql函数-日期函数

MySQL的函数-日期函数

| 函数名 | 描述 | 实例 |
|---|------------------------------|--|
| UNIX_TIMESTAMP() | 返回从1970-01-01 00:00:00到当前毫秒值 | select UNIX_TIMESTAMP() -> 1632729059 |
| UNIX_TIMESTAMP(DATE_STRING) | 将制定日期转为毫秒值时间戳 | SELECT UNIX_TIMESTAMP('2011-12-07 13:01:03'); |
| FROM_UNIXTIME(BIGINT UNIXTIME[, STRING FORMAT]) | 将毫秒值时间戳转为指定格式日期 | SELECT FROM_UNIXTIME(1598079966,'%Y-%m-%d %H:%i:%s'); (1598079966,'%Y-%m-%d %H:%i:%s'); -> 2020-08-22 15-06-06 |
| CURDATE() | 返回当前日期 | SELECT CURDATE(); -> 2018-09-19 |
| CURRENT_DATE() | 返回当前日期 | SELECT CURRENT_DATE(); -> 2018-09-19 |

| 函数名 | 描述 | 实例 |
|-----------------------------------|-------------------|---|
| TIMEDIFF(time1, time2) | 计算时间差值 | SELECT TIMEDIFF("13:10:11", "13:10:10"); -> 00:00:01 |
| DATE_FORMAT(d,f) | 按表达式 f的要求显示日期 d | SELECT DATE_FORMAT('2011-11-11 11:11:11','%Y-%m-%d %r') -> 2011-11-11 11:11:11 AM |
| STR_TO_DATE(string, format_mask) | 将字符串转变为日期 | SELECT STR_TO_DATE("August 10 2017", "%M %d %Y"); -> 2017-08-10 |
| DATE_SUB(date,INTERVAL expr type) | 函数从日期减去指定的时间间隔。 ↳ | Orders 表中 OrderDate 字段减去 2 天： SELECT OrderId,DATE_SUB(OrderDate,INTERVAL 2 DAY) AS OrderPayDate FROM Orders |

```

1 -- 日期函数
2 -- 获取时间戳(毫秒值)
3 select unix_timestamp();
4
5 -- 将一个日期字符串转为毫秒值
6 select unix_timestamp('2023-10-1 08:08:08');
7
8 -- 将时间戳毫秒值转为指定格式的日期
9 select from_unixtime(1696118888, '%Y-%m-%d %H:%i:%s');
10
11 -- 获取当前的年月日
12 select curdate();
13 select current_date();
14
15 -- 获取当前的时分秒
16 select current_time();
17 select curtime();
18
19 -- 获取年月日 和 时分秒
20 select current_timestamp();
21
22 -- 从日期字符串中获取年月日
23 select date('2023-9-30 12:59:59');
24
25 -- 获取日期之间的差值
26 select datediff(current_date(), '2008-08-08');
27
28 -- 获取时分秒之间的差值
29 select timediff('23:08:09', '13:01:22');
30
31 -- 日期格式化
32 select date_format('2021-1-1 01:12:01', '%Y-%m-%d %h:%i:%s');
33
34 -- 将字符串转为日期
35 select str_to_date('2021-12-13 11:11:11', '%Y-%m-%d %H:%i:%s');
36
37 -- 将日期进行减法
38 select date_sub('2021-10-01', interval 2 day);
39 select date_sub('2021-10-01', interval 2 month);
40
41 -- 将日期进行加法
42 select date_add('2021-10-01', interval 2 day);
43 select date_add('2021-10-01', interval 2 month);
44
45 -- 从日期中获取小时

```

```

46 select extract(hour from '2021-12-13 11:12:13');
47 select extract(year from '2021-12-13 11:12:13');
48 select extract(month from '2021-12-13 11:12:13');

49
50 -- 获取给定日期所在月的最后一天
51 select last_day('2021-8-13');

52
53 -- 获取指定年份和天数的日期
54 select makedate('2021',53);

55
56 -- 根据日期获取年月日 时分秒
57 select year('2021-12-13 11:12:13');
58 select month('2021-12-13 11:12:13');
59 select minute('2021-12-13 11:12:13');
60 select quarter('2021-12-13 11:12:13'); -- 获取季度

61
62 -- 根据日期获取信息
63 select monthname('2021-12-13 12:12:13');
64 select dayname('2021-12-13 12:12:13'); -- 获取周几 Monday
65 select dayofmonth('2021-12-13 12:12:13'); -- 当月的第几天
66 select dayofweek('2021-12-13 12:12:13'); -- 1周日 2周一
67 select dayofyear('2021-12-13 12:12:13'); -- 获取一年的第几天

68
69 select week('2021-12-13 12:12:13');
70 select week('2021-01-01 12:12:13');
71 select week('2021-12-31 12:12:13');

72
73 select yearweek('2021-3-01');

74
75 select now();

```

MySQL函数-控制流函数

MySQL的函数-控制流函数

◆ if逻辑判断语句

| 格式 | 解释 | 案例 |
|----------------------|---|--|
| IF(expr,v1,v2) | 如果表达式 expr 成立，返回结果 v1；否则，返回结果 v2。 | SELECT IF(1 > 0,'正确','错误') ->正确 |
| IFNULL(v1,v2) | 如果 v1 的值不为 NULL，则返回 v1，否则返回 v2。 | SELECT IFNULL(null,'Hello Word') ->Hello Word |
| ISNULL(expression) | 判断表达式是否为 NULL | SELECT ISNULL(NULL); ->1 |
| NULLIF(expr1, expr2) | 比较两个字符串，如果字符串 expr1 与 expr2 相等 返回 NULL，否则返回 expr1 | SELECT NULLIF(25, 25); -> |

| 格式 | 解释 | 操作 |
|---|--|--|
| <pre>CASE expression WHEN condition1 THEN result1 WHEN condition2 THEN result2 ... WHEN conditionN THEN resultN ELSE result END</pre> | <p>CASE 表示函数开始，END 表示函数结束。如果 condition1 成立，则返回 result1，如果 condition2 成立，则返回 result2，当全部不成立则返回 result，而当有一个成立之后，后面的就不执行了。</p> | <pre>select case 100 when 50 then 'tom' when 100 then 'mary'else 'tim' end;</pre> <pre>select case when 1=2 then 'tom' when 2=2 then 'mary' else'tim' end ;</pre> |

```

1 -- 控制流函数
2 -- if
3 select if(5>3,'大于','小于');
4 select name,if(chinese >= 80 , '优秀','良好') flag from student;
5
6 -- ifnull
7 select ifnull(5,0);
8 select ifnull(NULL,0);
9 select *,ifnull(comm,0) comm_flag from emp;
10
11 -- ifnull
12 select isnull(5);
13 select isnull(NULL);
14
15 -- nullif
16 select nullif(12,12); -- null
17 select nullif(12,13); -- 12
18
19 -- case when
20 select
21   case 5
22     when 1 then 'hello'
23     when 2 then 'world'
24     when 5 then '!!!!'
25     else
26       'other'
27   end as info;
28
29 select
30 case
31   when 2>1 then 'hello'
32   when 2<1 then 'world'
33   when 5>3 then '!!!!'
34   else
35     'other'
36   end as info;
37
38 -- 创建表
39 create table orders(
40   oid int primary key,
41   price double,
42   payType int -- 1微信 2支付宝 3银行卡 4其他
43 );
44
45 insert into orders values(1,1200,1);
46 insert into orders values(2,1000,2);
47 insert into orders values(3,200,3);

```

```

48  insert into orders values(4,3000,1);
49  insert into orders values(5,1500,2);
50
51 -- 方式1
52 select
53  *,
54  case payType
55    when 1 then '微信'
56    when 2 then '支付宝'
57    when 3 then '银行卡'
58    else
59      '其他支付方式'
60    end as payTypeStr
61  from orders;
62
63 -- 方式2
64 select
65  *,
66  case
67    when payType>0 then '微信'
68    when payType>1 then '支付宝'
69    when payType>2 then '银行卡'
70    else
71      '其他支付方式'
72    end as payTypeStr
73  from orders;

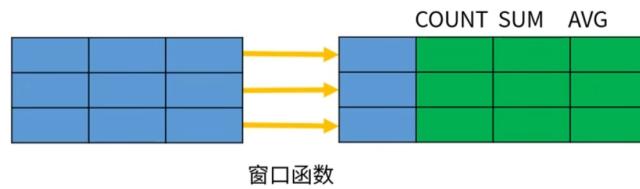
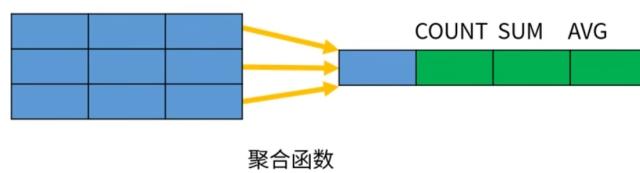
```

MySQL函数-窗口函数

MySQL的函数-窗口函数

◆ 介绍

- MySQL 8.0 新增窗口函数，窗口函数又被称为开窗函数，与Oracle 窗口函数类似，属于MySQL的一大特点。
- 非聚合窗口函数是相对于聚函数来说的。聚合函数是对一组数据计算后返回单个值（即分组），非聚合函数一次只会处理一行数据。窗口聚合函数在行记录上计算某个字段的结果时，可将窗口范围内的数据输入到聚合函数中，并不改变行数。



DOS

```
1 mysql> show databases;
2 +-----+
3 | Database           |
4 +-----+
5 | database1          |
6 | information_schema |
7 | mysql               |
8 | performance_schema |
9 | sys                 |
10+-----+
11 5 rows in set (0.00 sec)
12
13 mysql> use database1
14 Database changed
15 mysql> show tables;
16 +-----+
17 | Tables_in_database1 |
18 +-----+
19 | student              |
20 +-----+
21 1 row in set (0.00 sec)
22
23 mysql> desc student;
24 +-----+-----+-----+-----+-----+-----+
25 | Field   | Type      | Null | Key | Default | Extra       |
26 +-----+-----+-----+-----+-----+-----+
27 | sid     | int        | NO   | PRI | NULL    | auto_increment |
28 | name    | varchar(20) | YES  | UNI | NULL    |             |
29 | age     | int        | NO   |     | NULL    |             |
30 +-----+-----+-----+-----+-----+-----+
31
32
```