

SQL查询

SQL语句中对于程序员来讲最重要的就是DML，而在DML语句当中最常用的语句就是增删改查，而其中增删改基本上是千篇一律，唯独查询千变万化

查询语句的语法规范：

```
select 查询列 from 查询表

where 查询条件
group by 分组依据 having 分组条件
order by 排序列
limit 控制查询结果的数量
```

举例：创建实验用表

```
create table student(
    sid int primary key auto_increment,
    sname varchar(10) not null,
    cno varchar(20) not null,
    phone char(11) unique key not null,
    score float not null
);

insert into student(sname,cno,phone,score) values
('李四','2','1311111111',78.7),
('lily','2','1311111112',81.2),
('james','1','1311111113',64.1),
('lucy','2','1311111114',91)
```

1、基础查询

```
#查询所有数据
select * from student;

#查询指定列
select sname,score from student;

#查询所有学生姓名
select sname from student;

#查询学生姓名、手机号（前缀拼接）
select sname,concat('+86-',phone) from student

#同级数据的总行数
select count(*) from student;
```

```
#查询最高分
select max(score) from student;
#查询最低分
select min(score) from student;
#查询平均分]
select avg(score) from student;
#求所有学生分数总和
select sum(score) from student;
```

2、条件查询

```
#查询所有分数在80分以上的学生信息
select * from student where score >= 80;
#查询区间值（查询分数在70-90之间的学生信息）
select * from student where score >= 70 and score <= 90;
select * from student where score between 70 and 90;
#查询james、lily、lucy三名学生的成绩和姓名
select sname,score from student where sname='james' or
sname='lily' or sname='lucy';
select sname,score from student where sname='james' ||
sname='lily' || sname='lucy';
select sname,score from student where sname
in('james','lily','lucy');
#查询所有分数不等于80的学生信息
select * from student where score != 80;
select * from student where score <> 80;
#查询所有名字以‘l’开头的学生信息
select * from student where sname like 'l%';
#查询所有名字以‘j’开头同时后面跟了4个字符的学生信息
select * from student where sname like 'j_____';
#查询所有名字中包含‘e’的学生信息
select * from student where sname like '%e%';
#查询名字不是‘j’开头的学生信息
select * from student where sname not like 'j%';
```

模糊查询：

% 是一个占位符，标识任意字符长度

_ 是一个占位符，只标识一个字符长度

3、分组、排序、分页

```
/* 分组 */
#查询每个班的平均成绩和班级编号
select cno,avg(score) from student group by cno;
#查询每个班的总人数
select count(cno) '人数',cno from student group by cno;
#查询总人数超过4个人的班级的总人数
select count(cno) '人数',cno from student group by cno having
count(cno) > 4;

/* 排序 */
#查询所有的学生成绩，并升序显示(默认情况下就是升序，由低到高，asc)
select * from student order by score;
#查询所有的学生成绩，降序排列
select * from student order by score desc;
#查询每个班的平均分并降序排列
select cno,avg(score) from student group by cno order by
avg(score) desc;

/* 分页 */
#查询前3条数据
#查询分数排名前三的学生信息
#查询第二页数据，每页显示3条
```

4、去除重复行

在select语句中使用关键distinct去除重复行

```
select distinct cno from student;
```

注意事项:

distinct 需要放在所有列的前面，如果夹在中间会报错

5、空值参与运算

所有运算符与列值运算时如果遇到null，运算结果都是null

这里一定注意:

在mysql里面，空值是不等于空字符串，一个空字符串的长度是0，而一个空值的长度是空，所以在mysql当中空值是占空间的

6、着重号

```
create table `table`(  
)
```

我们需要保证表中的字段名，表名没有和保留字关键字、系统中常用方法名冲突，如果真的相同，在sql语句中可以使用 `` 着重号框起来

7、列的别名

```
select sname as '学生姓名' from student;
```

重命名一个列，方便计算，使用as关键字，可以省略不写

8、多表查询

多表查询也叫关联查询，一般指的是两张或两张以上的多张表一起完成查询操作

前提条件：

要实现多表查询，表与表之间需要形成关系，他们之间有关联的字段，这个关联字段可以建立外键，比如，员工表和部门表，这两个表依靠部门编号进行关联

#案例：查询员工姓名和部门名称

```
select ename,dname from emp,dept;
```

结果分析：

我们发现返回的结果并不是我们想要的一个结果，简单来说这里做了一个两列查询，结果返回了一个多重组合，这种情况我们称之为**笛卡尔积的错误**

9、笛卡尔积（交叉链接）的理解

笛卡尔积是一个数学运算，假设我们现在有两个集合x和y，那么x和y的笛卡尔积就是x和y的所有可能组合，组合的个数即为两个集合中元素个数的乘积

笛卡尔积的错误会在以下条件中产生：

- 省略了多个表的关联条件
- 关联条件无效

- 所有表种的所有行相互关联

为了避免笛卡尔积，可以在where后面加入有效的关联条件

语法：

```
select table1.column,table2.column  
from table1,table2  
where table1.column = table2.c1oumn
```

说明：

table1和table2表示你需要查询的表

把上面的例子做修改

```
SELECT  ename,dname  
FROM    emp,dept  
WHERE   emp.deptno = dept.deptno;
```

还可以简化用别名

```
SELECT  ename,dname  
FROM    emp e,dept d  
WHERE   e.deptno = d.deptno;
```

10、多表查询分类

所谓多表查询我们可以先简单理解成就是把多张表中的数据调出，然后通过一些对比的方式将多张表的数据拼成一张表返回给我们

10.1、等值链接

```
SELECT  *  
FROM    emp e,dept d  
WHERE   e.deptno = d.deptno;
```

多个链接条件

```
SELECT  *  
FROM    emp e,dept d  
WHERE   e.deptno = d.deptno and d.deptno = 10;
```

区别重命名

多个表有相同列名的时候，必须要在列名前面加上表名做前缀

```
SELECT *  
FROM emp,dept  
WHERE emp.deptno = dept.deptno and dept.deptno = 10;
```

链接多个表

```
SELECT e.ename, e.job, e.sal, e.hiredate, d.dname, s.grade  
FROM emp e, dept d, salgrade s      # e:员工表    d: 部门表    s:  
薪资等级表  
WHERE e.deptno = d.deptno and e.sal BETWEEN s.losal AND s.hisal
```

注意:

如果链接n个表，至少需要n-1个链接条件，比如链接三个表，至少需要2个链接条件

10.2、非等值链接

```
SELECT e.ename,e.job,e.sal,s.grade  
FROM emp e,salgrade s  
WHERE e.sal BETWEEN s.losal AND s.hisal;
```

11、内连接与外连接

链接查询的意义：在用户查看数据的时候，需要显示的数据来自多张表

11.1、内链接

内链接查询时最常见的链接查询，内连接可以查询两张或以上的表

内连接：inner join 从左表中取出‘所有’记录，去右表中与右表中的所有数据进行匹配，匹配必须要时某个条件在左表与右表中想相同才会最终保留下来，否则不保留

语法结构：

```
左表 inner join 右表 on 左表.字段 = 右表.字段
```

on表示链接条件，条件字段就时代表相同的业务含义

当两个表中存在相同意义的字段的时候，就可以通过该字段链接查询两个表，当该相同意义字段的值相同的时候就可以查询该记录

实验：

显示所有员工所在的部门信息

```
-- 内连接查询 SQL99
SELECT *
FROM emp e inner join dept d
ON e.deptno = d.deptno;
```

我们会发现上面的查询起始和等值连接的结果是一样的

SQL92与SQL99语法表种上对于连接拆线呢的表示不同：

- SQL92：把连接查询的链接条件与筛选条件都放在where的后面
- SQL99：引入了inner join、on关键字，将链接条件和筛选条件分离，使查询语句更加清晰简洁

显示所有员工的姓名，职位，月薪，入职时间，部门名，薪资等级

```
# e:员工表    d: 部门表    s: 薪资等级表
-- SQL92
SELECT e.ename, e.job, e.sal, e.hiredate, d.dname, s.grade
FROM emp e, dept d, salgrade s
WHERE e.deptno = d.deptno and e.sal BETWEEN s.losal AND s.hisal

-- SQL99
SELECT e.ename, e.job, e.sal, e.hiredate, d.dname, s.grade
FROM emp e
INNER JOIN dept d ON d.deptno = e.deptno
INNER JOIN salgrade s ON e.sal BETWEEN s.losal AND s.hisal;
```

查询出ACCOUNTING部门中最高的月薪

#内连接查询 --SQL99

```
SELECT max(e.sal), d.dname
FROM emp e
INNER JOIN dept d ON d.deptno = e.deptno and d.dname =
'ACCOUNTING'
```

#等值连接查询 --SQL92

```
SELECT max(e.sal), d.dname
FROM emp e,dept d
WHERE d.deptno = e.deptno and d.dname = 'ACCOUNTING'
```

查询出每个部门的最高月薪以及部门名称

#内连接查询

```
SELECT max(e.sal), d.dname
FROM emp e
INNER JOIN dept d ON d.deptno = e.deptno GROUP BY d.deptno
```

#等值连接查询

```
SELECT max(e.sal), d.dname
FROM emp e,dept d
WHERE d.deptno = e.deptno
GROUP BY d.deptno
```

查询出每个部门的平均月薪，并按照平均薪资从高到低排序

#等值连接

```
SELECT avg(e.sal),d.dname
FROM emp e,dept d
WHERE d.deptno = e.deptno GROUP BY d.deptno ORDER BY avg(e.sal)
desc
```

#内连接

```
SELECT avg(e.sal),d.dname
FROM emp e
INNER JOIN dept d ON d.deptno = e.deptno
GROUP BY d.deptno
ORDER BY avg(e.sal) desc
```

11.2、外连接

以某张表为主，取出里面的所有记录，然后每条与另外一张表进行连接，不管能崩匹配上，最后都会保留，能正确匹配的正确保留，不能匹配上的设置为null

左连接： left join，以左表为主表

语法结构：

```
from 左表 left join 右表 on 左表.字段 = 右表.字段
```

左表不管能不能匹配上条件，最终都会保留，能匹配正确保留，不能匹配，右表的字段都设置成null

右连接： right join 以右表为主表

语法结构：

```
from 左表 right join 右表 on 左表.字段 = 右表.字段
```

右表不管能不能匹配上条件，最终都会保留，能匹配正确保留，不能匹配，左表的字段都设置成null

举例：

查询出所有员工的信息以及部门信息，同时显示没有部门的员工信息

```
SELECT e.*,d.*  
FROM emp e  
LEFT JOIN dept d ON d.deptno = e.deptno;
```

12、子查询

通过我们在查询SQL中嵌套查询，称为子查询，子查询通常会使复杂的查询变的简单，但是相关的子查询要对基础表的每一条数据都进行子查询的动作，所以当表单中数据过大的时候，一定谨慎使用

带IN关键字的子查询

使用in关键字可以在原表中特定列的值与子查询返回的结果进行值比较，如果某行的特定列存在，则在select语句的查询结果中包含这一行

查询RESEARCH部门的员工信息（单行单列的子查询）

```

SELECT *
FROM emp
WHERE deptno=(
    SELECT deptno
    FROM dept
    WHERE dname = 'RESEARCH'
)

```

查询ACCOUNTING和RESEARCH的员工信息（多行子查询）

```

SELECT *
FROM emp
WHERE deptno in( #deptno = 20 or deptno = 10
    SELECT deptno
    FROM dept
    WHERE dname in('ACCOUNTING','RESEARCH')
)

```

查询出与 'ALLEN' 同一个部门， 同一个岗位的所有员工

```

SELECT *
FROM emp e,(
    SELECT deptno,job
    FROM emp
    WHERE ename = 'ALLEN'
) t
WHERE e.deptno = t.deptno and e.job = t.job

```

将一个子查询的结果当作一个临时表（虚拟表）作为外层查询的依据

查询ACCOUNTING中比RESEARCH部门中任何一个员工工资都高的员工信息

```

SELECT e.*
FROM emp e,dept d
WHERE e.deptno = d.deptno and d.dname = 'ACCOUNTING' and e.sal >
(
    SELECT max(e.sal)
    FROM emp e,dept d
    WHERE e.deptno = d.deptno and d.dname = 'RESEARCH'
)

```