SQL查询 - 复习

2020年2月6日 11:18

1. 准备

```
create database mydb2;
use mydb2;

create table user (id int, name varchar(20),age int, addr varchar(20));
insert into user values (1,'aa',19,'bj');
insert into user values (2,'bb',23,'sh');
insert into user values (3,'cc',26,'gz');
insert into user values (4,'dd',24,'bj');
insert into user values (5,'ee',31,'sz');
insert into user values (6,'ff',27,'sh');
insert into user values (7,'gg',26,'bj');

create table dept (id int ,name varchar(20));
insert into dept values (999,'行政部');
insert into dept values (888,'财务部');
insert into dept values (777,'销售部');
insert into dept values (666,'科技部');

create table emp (id int,name varchar(20),did int);
insert into emp values (1,'萨达姆',999);
insert into emp values (2,'哈利波特',888);
insert into emp values (3,'孙悟空',777);
insert into emp values (4,'朴乾',777);
insert into emp values (5,'本拉登',555);

create table emp2( id int, name varchar(20), salary int, did int, job varchar(20) );
insert into emp2 values (1,'aaa',3200,999,'it');
insert into emp2 values (2,'bbb',4500,888,'it');
insert into emp2 values (3,'ccc',7300,999,'saler');
insert into emp2 values (4,'ddd',3000,999,'saler');
insert into emp2 values (5,'eee',2800,777,'hr');
insert into emp2 values (6,'fff',5100,777,'it');
insert into emp2 values (7,'ggg',9100,777,'it');
insert into emp2 values (8,'hhh',9900,666,'saler');
```

2. 过滤查询

a. 语法结构

```
select from where
```

b. 案例:

i. 查询年龄大于20岁的用户

```
mysql> select * from user where age > 20;
+------+------+------+------+
| id   | name | age  | addr |
+------+------+------+------+
|    2 | bb   |   23 | sh   |
|    3 | cc   |   26 | gz   |
|    4 | dd   |   24 | bj   |
|    5 | ee   |   31 | sz   |
|    6 | ff   |   27 | sh   |
|    7 | gg   |   26 | bj   |
+------+------+------+------+
```

ii. 查询年龄在20-30岁的北京用户

```
mysql> select * from user where age between 20 and 30 and addr = 'bj';
+------+------+------+------+
| id   | name | age  | addr |
+------+------+------+------+
|    4 | dd   |   24 | bj   |
|    7 | gg   |   26 | bj   |
+------+------+------+------+
```

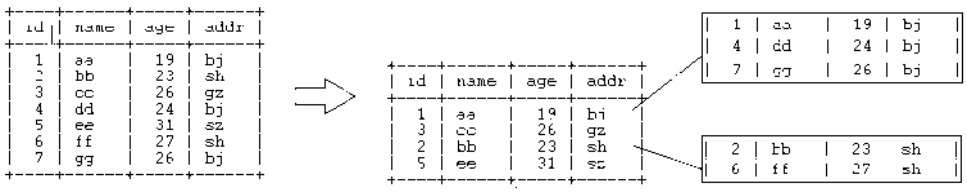
3. 分组查询

a. 语法结构

select from group by having

b. 分组查询的原理

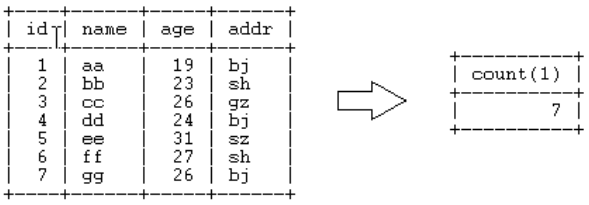
i. 数据进行分组后，并不是相同组只保留一条数据，而是同组数据合并为一条显示，同组的数据“摞在一起”



c. 聚合函数特点

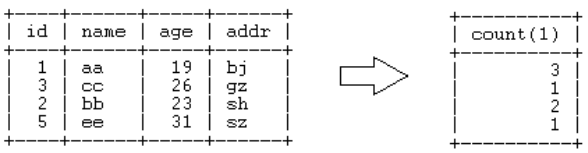
i. 如果没有分组操作，聚合函数作用在整个结果上

```
mysql> select count(1) from user;
```



ii. 如果有分组操作，聚合函数作用在组内

```
mysql> select count(1)from user group by addr;
```



iii. where 和having的区别

1) where是分组之前的过滤,having是分组之后的过滤

案例：将20-30岁的用户按地区分组求平均年龄，求这些平均年龄大于25的地区名称。

```
mysql> select addr,avg(age)
 from user where age between 20 and 30 group by addr having avg(age)>25;
+------+----------+
| addr | avg(age) |
+------+----------+
| gz   | 26.0000  |
+------+----------+
```

2) where内部无法使用聚合函数,having内部可以使用聚合函数

原理：

i) where在分组之前执行，将每一条数据按where条件判断是否符合查询条件，确定过滤查询结果

未执行分组操作时执行的聚合函数作用于整个查询结果

where执行未完成，无法确定整个查询结果，条件不成立，聚合函数无法执行。

因此，where中无法使用聚合函数。

ii) having在分组操作之后执行

分组操作执行过后，聚合函数作用于组内

分组操作已经完成，组内数据是确定的，聚合函数可以执行。

因此，having中可以使用聚合函数。

iii) 如果没有分组操作，having等价于where,但不推荐这种用法

```
mysql> select * from user where age > 20;
+----+------+-----+------+
| id | name | age | addr |
+----+------+-----+------+
|  2 | bb   |  23 | sh   |
|  3 | cc   |  26 | gz   |
|  4 | dd   |  24 | bj   |
|  5 | ee   |  31 | sz   |
|  6 | ff   |  27 | sh   |
|  7 | gg   |  26 | bj   |
+----+------+-----+------+
6 rows in set

mysql> select * from user having age > 20;
+----+------+-----+------+
| id | name | age | addr |
+----+------+-----+------+
|  2 | bb   |  23 | sh   |
|  3 | cc   |  26 | gz   |
|  4 | dd   |  24 | bj   |
|  5 | ee   |  31 | sz   |
|  6 | ff   |  27 | sh   |
|  7 | gg   |  26 | bj   |
+----+------+-----+------+
6 rows in set
```

4. 排序查询

a. 语法结构

```
select from order by [asc|desc]
```

案例：按年龄排序所有用户

```
mysql> select * from user order by age desc;
+----+------+-----+------+
| id | name | age | addr |
+----+------+-----+------+
|  5 | ee   |  31 | sz   |
|  6 | ff   |  27 | sh   |
|  3 | cc   |  26 | gz   |
|  7 | gg   |  26 | bj   |
|  4 | dd   |  24 | bj   |
|  2 | bb   |  23 | sh   |
|  1 | aa   |  19 | bj   |
+----+------+-----+------+
7 rows in set
```

5. 分页查询

a. 语法结构

```
select from limit a,b
```

案例：查询年龄大于20，年龄由小到大的前3为用户

```
mysql> select * from user where age > 20 order by age limit 0,3;
+----+------+-----+------+
| id | name | age | addr |
+----+------+-----+------+
|  2 | bb   |  23 | sh   |
|  4 | dd   |  24 | bj   |
|  3 | cc   |  26 | gz   |
+----+------+-----+------+
3 rows in set
```

6. 多表查询

a. 多表设计

关系型数据库使用表来存储数据，使用表和表之间的关系来存储数据之间的关系。

表之间的关系通过外键维系。

表之间关系分为1对1，1对多，多对多三种，根据关系不同按照不同方式维系

表和表之间的关系

1对1

1 - 1

class

id	name	r_id
1902	向日葵班	506
1901	玫瑰花班	504
1812	牡丹花班	503
1811	菊花班	502

room

id	name	cid
506	梅屋	1902
504	兰屋	1901
503	竹屋	1812
502	菊屋	1811

1对多

1 - *

class

id	name	sid
1902	向日葵班	999/888
1901	玫瑰花班	
1812	牡丹花班	
1811	菊花班	

stu

id	name	cid
999	小新	1902
888	风间	1902
777	美芽	1901
666	小白	1812

多对多

* - *

stu

id	name	tid
999	小新	111/222
888	风间	
777	美芽	
666	小白	

teacher

id	name	sid
111	孙悟空	999/888
222	哈利波特	
333	灭霸	
444	朴乾	

stu_teacher

sid	tid
999	111
999	222
999	444
888	222
888	444
777	333
777	4444
666	111

b. 多表查询

i. 笛卡尔积查询

```
select * from dept,emp;
```

两张表相乘的结果，左表有m条数据，右表有n条数据，则得到m*n条数据

其中的结果中包含了大量错误的数据，原因在于没有考虑两张表之间的关联关系。

虽然这种查询方式很少直接使用，但它时其他多表查询技术的基础，有必要掌握。

```
mysql> select * from dept,emp;
+------+--------+------+----------+------+
| id   | name   | id   | name     | did  |
+------+--------+------+----------+------+
|  999 | 行政部 |    1 | 萨达姆   |  999 |
|  888 | 财务部 |    1 | 萨达姆   |  999 |
|  777 | 销售部 |    1 | 萨达姆   |  999 |
|  666 | 科技部 |    1 | 萨达姆   |  999 |
|  999 | 行政部 |    2 | 哈利波特 |  888 |
|  888 | 财务部 |    2 | 哈利波特 |  888 |
|  777 | 销售部 |    2 | 哈利波特 |  888 |
|  666 | 科技部 |    2 | 哈利波特 |  888 |
|  999 | 行政部 |    3 | 孙悟空   |  777 |
|  888 | 财务部 |    3 | 孙悟空   |  777 |
|  777 | 销售部 |    3 | 孙悟空   |  777 |
|  666 | 科技部 |    3 | 孙悟空   |  777 |
|  999 | 行政部 |    4 | 朴乾     |  777 |
|  888 | 财务部 |    4 | 朴乾     |  777 |
|  777 | 销售部 |    4 | 朴乾     |  777 |
|  666 | 科技部 |    4 | 朴乾     |  777 |
|  999 | 行政部 |    5 | 本拉登   |  555 |
|  888 | 财务部 |    5 | 本拉登   |  555 |
|  777 | 销售部 |    5 | 本拉登   |  555 |
|  666 | 科技部 |    5 | 本拉登   |  555 |
+------+--------+------+----------+------+
```

ii. 内连接查询

```
select * from dept,emp where dept.id = emp.did;
select * from dept inner join emp on dept.id = emp.did;
```

只有左边表和右边表都有数据的记录才会被查询出来。

```
mysql> select * from dept inner join emp on dept.id = emp.did;
+------+--------+------+----------+------+
| id   | name   | id   | name     | did  |
+------+--------+------+----------+------+
|  999 | 行政部 |    1 | 萨达姆   |  999 |
|  888 | 财务部 |    2 | 哈利波特 |  888 |
|  777 | 销售部 |    3 | 孙悟空   |  777 |
|  777 | 销售部 |    4 | 朴乾     |  777 |
+------+--------+------+----------+------+
```

iii. 外连接查询

左外连接查询

```
select * from dept left join emp on dept.id = emp.did;
```

在内连接的基础上增加了左边表有而右边表没有的数据

```
mysql> select * from dept left join emp on dept.id = emp.did;
+------+--------+------+----------+------+
| id   | name   | id   | name     | did  |
+------+--------+------+----------+------+
|  999 | 行政部 |    1 | 萨达姆   |  999 |
|  888 | 财务部 |    2 | 哈利波特 |  888 |
|  777 | 销售部 |    3 | 孙悟空   |  777 |
|  777 | 销售部 |    4 | 朴乾     |  777 |
|  666 | 科技部 | NULL | NULL     | NULL |
+------+--------+------+----------+------+
```

右外连接查询

```
select * from dept right join emp on dept.id = emp.did;
```

在内连接的基础上增加了右边表有而左边表没有的数据

```
mysql> select * from dept right join emp on dept.id = emp.did;
+------+--------+------+----------+------+
| id   | name   | id   | name     | did  |
+------+--------+------+----------+------+
|  999 | 行政部 |    1 | 萨达姆   |  999 |
|  888 | 财务部 |    2 | 哈利波特 |  888 |
|  777 | 销售部 |    3 | 孙悟空   |  777 |
|  777 | 销售部 |    4 | 朴乾     |  777 |
| NULL | NULL   |    5 | 本拉登   |  555 |
+------+--------+------+----------+------+
```

全外连接查询

```
select * from dept full join emp on dept.id = emp.did;
```

在内连接的基础上增加左边表有而右边表没有的记录和右边表有而左边表没有的记录

~mysql不支持full join关键字

~mysql可以通过union操作来间接实现全外连接查询

```
select * from dept left join emp on dept.id = emp.did
union
select * from dept right join emp on dept.id = emp.did;
```

```
mysql> select * from dept left join emp on dept.id = emp.did
    -> union
    -> select * from dept right join emp on dept.id = emp.did;
+------+--------+------+----------+------+
| id   | name   | id   | name     | did  |
+------+--------+------+----------+------+
|  999 | 行政部 |    1 | 萨达姆   |  999 |
|  888 | 财务部 |    2 | 哈利波特 |  888 |
|  777 | 销售部 |    3 | 孙悟空   |  777 |
|  777 | 销售部 |    4 | 朴乾     |  777 |
|  666 | 科技部 | NULL | NULL     | NULL |
| NULL | NULL   |    5 | 本拉登   |  555 |
+------+--------+------+----------+------+
```

7. 子查询

a. 非关联子查询

子查询先于主查询执行，查询结果作为主查询的一部分使用。

i. 标量子查询

子查询结果为单一值，作为主查询中单值使用

案例：查询所有收入大于员工aaa的员工

```
mysql> select * from emp2 where salary > (select salary from emp2 where
 name='aaa');
+------+------+--------+------+-------+
| id   | name | salary | did  | job   |
+------+------+--------+------+-------+
|    2 | bbb  |   4500 |  888 | it    |
|    3 | ccc  |   7300 |  999 | saler |
|    6 | fff  |   5100 |  777 | it    |
|    7 | ggg  |   9100 |  777 | it    |
|    8 | hhh  |   9900 |  666 | saler |
+------+------+--------+------+-------+
```

案例：查询所有部门编号及全公司平均薪水

```
mysql> select did,(select avg(salary) from emp2) from emp2;
+------+-----------------------------------+
| did  | (select avg(salary) from emp2)    |
+------+-----------------------------------+
|  999 |                        5612.5000  |
|  888 |                        5612.5000  |
|  999 |                        5612.5000  |
|  999 |                        5612.5000  |
|  777 |                        5612.5000  |
|  777 |                        5612.5000  |
|  777 |                        5612.5000  |
|  666 |                        5612.5000  |
+------+-----------------------------------+
```

ii. 列子查询

子查询结果为多行一列值，作为主查询中集合使用

案例：查询所有有it员工的部门的平均薪资

```
mysql> select did,avg(salary) from emp2 where did in (select distinct d
id from emp2 where job='it') group by did;
+------+-------------+
| did  | avg(salary) |
+------+-------------+
|  777 |  5666.6667  |
|  888 |  4500.0000  |
|  999 |  4500.0000  |
+------+-------------+
```

案例：查询所有工资高于任意it员工的saler员工

```
mysql> select * from emp2 where job='saler' and salary > any(select salary from emp2 where job = 'it');
+------+------+--------+------+-------+
| id   | name | salary | did  | job   |
+------+------+--------+------+-------+
|    3 | ccc  |   7300 |  999 | saler |
|    8 | hhh  |   9900 |  666 | saler |
+------+------+--------+------+-------+
```

案例：查询所有工资高于全部it员工的saler员工

```
mysql> select * from emp2 where job='saler' and salary > all(select salary from emp2 where job = 'it');
+------+------+--------+------+-------+
| id   | name | salary | did  | job   |
+------+------+--------+------+-------+
|    8 | hhh  |   9900 |  666 | saler |
+------+------+--------+------+-------+
```

iii. 行子查询(表子查询)

子查询结果为一行多列或多行多列，作为主查询中的一张表使用

案例：查询所有部门平均工资大于5000元的部门中平均工资最低的部门

```
mysql> select did,min(avgs) from (select did,avg(salary) as avgs from e
mp2 group by did having avgs>5000) as atab;
+------+------------+
| did  | min(avgs)  |
+------+------------+
|  666 | 5666.6667  |
+------+------------+
```

b. 关联子查询

主查询先于子查询执行，之后将主查询的结果依次应用于子查询，通常配合exists关键字对对主查询的结果进行进一步的过滤。

案例：查询所有工资大于销售平均工资的it员工的姓名（此题也可以用非关联子查询中标量子查询实现）

```
mysql> select name from emp2 as atab where job='it' and exists(select *
 from emp2 where job='saler' group by job having atab.salary>avg(salary
));
+------+
| name |
+------+
| ggg  |
+------+
```

8. 查询语句编写技巧

a. sql语句关键字书写顺序

select from where group by having order by limit

b. sql语句关键字执行顺序

from where group by having select order by limit

c. 编写sql语句的技巧

先根据需求确定需要哪些关键字，依次写下来

再根据sql语句关键字执行顺序因此填空

涉及到多张表关联子查询，先关联两张表，再将关联结果理解为一张虚拟的表，再关联第三张表，如此，依次关联所有表。