05 MySQL之多表查询

MySQL之多表查询

阅读目录

一 介绍

本节主题

- 多表连接查询
- 复合条件连接查询
- 子查询

首先说一下，我们写项目一般都会建一个数据库，那数据库里面是不是存了好多张表啊，不可能把所有的数据都放到一张表里面，肯定要分表来存数据，这样节省空间，数据的组织结构更清晰，解耦和程度更高，但是这些表本质上是不是还是一个整体啊，是一个项目所有的数据，那既然分表存了，就要涉及到多个表连接查询了，比如说员工信息一张表，部门信息一张表，那如果我想让你帮我查一下技术部门有哪些员工的姓名，你怎么办，单独找员工表能实现吗，不能，单独找部门表也无法实现，因为部门表里面没有员工的信息，对不对，所以就涉及到部门表和员工表来关联到一起进行查询了，好，那我们来建立这么两张表：

```
#建表#部门表
create table department(
id int,
name varchar(20)
);
#员工表，之前我们学过foreign key，强行加上约束关联，但是我下面这个表并没有直接加foreign key，这两个表我只是让它们在逻辑
create table employee(
id int primary key auto_increment,
name varchar(20),
sex enum('male','female') not null default 'male',
age int,
dep_id int
);

#给两个表插入一些数据
insert into department values
(200,'技术'),
(201,'人力资源'),
(202,'销售'),
(203,'运营'); #注意这一条数据，在下面的员工表里面没有对应这个部门的数据

insert into employee(name,sex,age,dep_id) values
('egon','male',18,200),
('alex','female',48,201),
('wupeiqi','male',38,201),
('yuanhao','female',28,202),
('liwenzhou','male',18,200),
('jingliyang','female',18,204) #注意这条数据的dep_id字段的值，这个204，在上面的部门表里面也没有对应的部门id。所以两者都会
;


#查看表结构和数据
```

```
mysql> desc department;
+-------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id | int(11) | YES | | NULL | |
| name | varchar(20) | YES | | NULL | |
+-------+-------------+------+-----+---------+-------+

mysql> desc employee;
+--------+----------------------+------+-----+---------+----------------+
| Field | Type | Null | Key | Default | Extra |
+--------+----------------------+------+-----+---------+----------------+
| id | int(11) | NO | PRI | NULL | auto_increment |
| name | varchar(20) | YES | | NULL | |
| sex | enum('male','female') | NO | | male | |
| age | int(11) | YES | | NULL | |
| dep_id | int(11) | YES | | NULL | |
+--------+----------------------+------+-----+---------+----------------+

mysql> select * from department;
+------+--------------+
| id | name |
+------+--------------+
| 200 | 技术 |
| 201 | 人力资源 |
| 202 | 销售 |
| 203 | 运营 |
+------+--------------+

mysql> select * from employee;
+----+------------+--------+------+--------+
| id | name | sex | age | dep_id |
+----+------------+--------+------+--------+
| 1 | egon | male | 18 | 200 |
| 2 | alex | female | 48 | 201 |
| 3 | wupeiqi | male | 38 | 201 |
| 4 | yuanhao | female | 28 | 202 |
| 5 | liwenzhou | male | 18 | 200 |
| 6 | jingliyang | female | 18 | 204 |
+----+------------+--------+------+--------+
```

二 多表连接查询

```
#重点：外链接语法

SELECT 字段列表
    FROM 表1 INNER|LEFT|RIGHT JOIN 表2
    ON 表1.字段 = 表2.字段;
```

1、交叉连接：不适用任何匹配条件。生成笛卡尔积

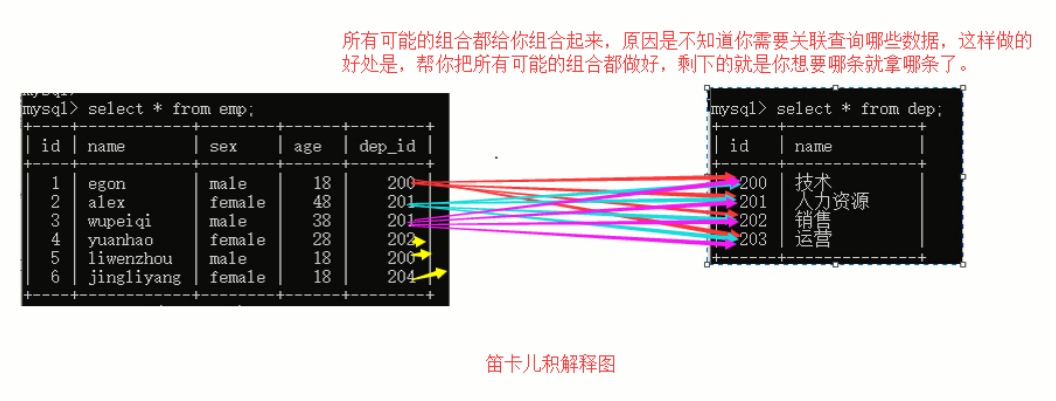
补充一点：select 查询表的时候，后面可以跟多张表一起查询：

```
mysql> select * from department,employee; #表用逗号分隔，看我查询时表的顺序，先department后employee，所以你看结果表
+------+--------------+----+------------+--------+------+--------+
| id   | name         | id | name       | sex    | age  | dep_id |
+------+--------------+----+------------+--------+------+--------+
| 200 | 技术         |  1 | egon       | male   |  18 |    200 |
| 201 | 人力资源     |  1 | egon       | male   |  18 |    200 |
| 202 | 销售         |  1 | egon       | male   |  18 |    200 |
| 203 | 运营         |  1 | egon       | male   |  18 |    200 |
| 200 | 技术         |  2 | alex       | female |  48 |    201 |
| 201 | 人力资源     |  2 | alex       | female |  48 |    201 |
| 202 | 销售         |  2 | alex       | female |  48 |    201 |
| 203 | 运营         |  2 | alex       | female |  48 |    201 |
| 200 | 技术         |  3 | wupeiqi    | male   |  38 |    201 |
| 201 | 人力资源     |  3 | wupeiqi    | male   |  38 |    201 |
| 202 | 销售         |  3 | wupeiqi    | male   |  38 |    201 |
| 203 | 运营         |  3 | wupeiqi    | male   |  38 |    201 |
| 200 | 技术         |  4 | yuanhao    | female |  28 |    202 |
| 201 | 人力资源     |  4 | yuanhao    | female |  28 |    202 |
| 202 | 销售         |  4 | yuanhao    | female |  28 |    202 |
| 203 | 运营         |  4 | yuanhao    | female |  28 |    202 |
| 200 | 技术         |  5 | liwenzhou  | male   |  18 |    200 |
| 201 | 人力资源     |  5 | liwenzhou  | male   |  18 |    200 |
| 202 | 销售         |  5 | liwenzhou  | male   |  18 |    200 |
| 203 | 运营         |  5 | liwenzhou  | male   |  18 |    200 |
| 200 | 技术         |  6 | jingliyang | female |  18 |    204 |
| 201 | 人力资源     |  6 | jingliyang | female |  18 |    204 |
| 202 | 销售         |  6 | jingliyang | female |  18 |    204 |
| 203 | 运营         |  6 | jingliyang | female |  18 |    204 |
+------+--------------+----+------------+--------+------+--------+
24 rows in set (0.12 sec)我们让employee表在前面看看结果，注意看结果表的字段
```

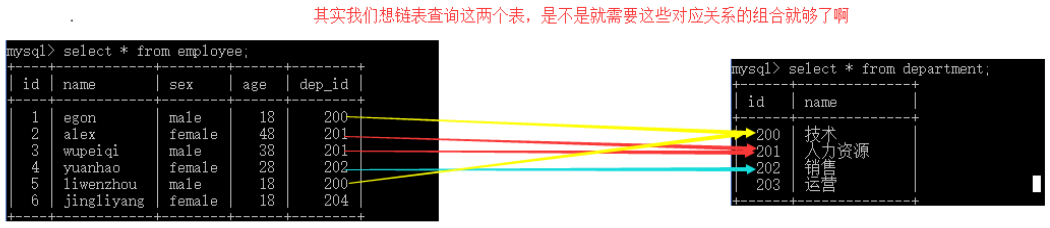
```
mysql> select * from employee,department;
+----+------------+--------+------+--------+------+--------------+
| id | name       | sex    | age  | dep_id | id   | name         |
+----+------------+--------+------+--------+------+--------------+
|  1 | egon       | male   |   18 |    200 |  200 | 技术         |
|  1 | egon       | male   |   18 |    200 |  201 | 人力资源     |
|  1 | egon       | male   |   18 |    200 |  202 | 销售         |
|  1 | egon       | male   |   18 |    200 |  203 | 运营         |
|  2 | alex       | female |   48 |    201 |  200 | 技术         |
|  2 | alex       | female |   48 |    201 |  201 | 人力资源     |
|  2 | alex       | female |   48 |    201 |  202 | 销售         |
|  2 | alex       | female |   48 |    201 |  203 | 运营         |
|  3 | wupeiqi    | male   |   38 |    201 |  200 | 技术         |
|  3 | wupeiqi    | male   |   38 |    201 |  201 | 人力资源     |
|  3 | wupeiqi    | male   |   38 |    201 |  202 | 销售         |
|  3 | wupeiqi    | male   |   38 |    201 |  203 | 运营         |
|  4 | yuanhao    | female |   28 |    202 |  200 | 技术         |
|  4 | yuanhao    | female |   28 |    202 |  201 | 人力资源     |
|  4 | yuanhao    | female |   28 |    202 |  202 | 销售         |
|  4 | yuanhao    | female |   28 |    202 |  203 | 运营         |
|  5 | liwenzhou  | male   |   18 |    200 |  200 | 技术         |
|  5 | liwenzhou  | male   |   18 |    200 |  201 | 人力资源     |
|  5 | liwenzhou  | male   |   18 |    200 |  202 | 销售         |
|  5 | liwenzhou  | male   |   18 |    200 |  203 | 运营         |
|  6 | jingliyang | female |   18 |    204 |  200 | 技术         |
|  6 | jingliyang | female |   18 |    204 |  201 | 人力资源     |
|  6 | jingliyang | female |   18 |    204 |  202 | 销售         |
|  6 | jingliyang | female |   18 |    204 |  203 | 运营         |
+----+------------+--------+------+--------+------+--------------+
24 rows in set (0.00 sec)
```

关于笛卡儿积：我们看一下上面的这些数据，有什么发现，首先看到这些字段都显示出来了，并且数据变得很多，我们来看一下，这么

关于笛卡儿积现象的解释图：



咱们为了更好的管理数据，为了节省空间，为了数据组织结构更清晰，将数据拆分到了不同表里面，但是本质上是不是还是一份数据，一份重复内容很多的很大的数据，所以我们即便是分表了，但是咱们是不是还需要找到一个方案把两个本来分开的表能够合并到一起来进行查询，那你是不是就可以根据部门找员工，根据员工找部门了，对不对，但是我们合并两个表的时候，如何合并，根据什么来合并，通过笛卡儿积这种合并有没有浪费，我们其实想做的是不是说我们的员工表中dep_id**这个字段中的数据和部门表里面的**id**能够对应上就可以了，因为我们知道我们设计表的时候，是通过这两个字段来给两个表建立关系的，对不对，看下图：**



我们的目标就是将两个分散出去的表，按照两者之间有关系的字段，能对应上的字段，把两者合并成一张表，这就是多表查询的一个本质。那么笛卡儿积干了什么事儿，就是简单粗暴的将两个表的数据全部对应了一遍，用处就是什么呢，它肯定就能保证有一条是对应准的，你需要做的事情就是在笛卡儿积的基础上只过滤出我们需要的那些数据就行了，笛卡儿积不是咱们最终要得到的结果，只是给你提供了一个基础，它不管对应的对不对，全部给你对应一遍，然后你自己去筛选就可以了，然后基于笛卡儿积我们来找一下对应的数据，看看能不能找到：

2、内连接：只连接匹配的行

```
#我们要找的数据就是员工表里面dep_id字段的值和部门表里面id字段的值能对应上的那些数据啊，所以你看下面的写法：
mysql> select * from employee,department where employee.dep_id=department.id;
+----+-----------+--------+------+--------+------+--------------+
| id | name      | sex    | age  | dep_id | id   | name         |
+----+-----------+--------+------+--------+------+--------------+
|  1 | egon      | male   |   18 |    200 |  200 | 技术         |
|  2 | alex      | female |   48 |    201 |  201 | 人力资源     |
|  3 | wupeiqi   | male   |   38 |    201 |  201 | 人力资源     |
|  4 | yuanhao   | female |   28 |    202 |  202 | 销售         |
```

```
| 5 | liwenzhou | male   |  18 |    200 |  200 | 技术       |
+----+-----------+--------+-----+--------+------+-------------+
5 rows in set (0.14 sec)
```

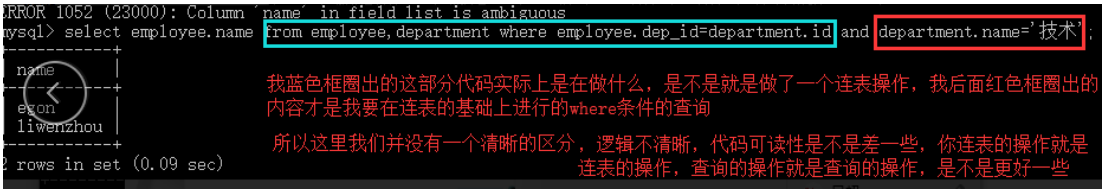
拿到了我们想要的结果。

但是你看，我们左表employee表中的dep_id为204的那个数据没有了，右表department表的id为203的数据没有了，因为我们现在要

```
#再看一个需求，我要查出技术部的员工的名字
mysql> select name from employee,department where employee.dep_id=department.id and department.name='技术';
ERROR 1052 (23000): Column 'name' in field list is ambiguous
#上面直接就报错了，因为select后面直接写的name，在两个表合并起来的表中，是有两个name字段的，直接写name是不行的，要加
mysql> select employee.name from employee,department where employee.dep_id=department.id and department.name=
+-----------+
| name      |
+-----------+
| egon      |
| liwenzhou |
+-----------+
2 rows in set (0.09 sec)
```

结果就没问题了

但是你看上面的代码有没有什么不太好的地方，虽然我们能够完成我们的事情，但是代码可读性不好，所以以后不要这么写，但是看图：



所以mysql为我们提供了一些专门做连表操作的方法，这些方法语义更加的明确，你一看就知道那些代码是连表的，那些代码是查询的，其实上面的连表也是个查询操作，但是我们为了区分明确，连表专门用连表的方法，查询就专门用查询的方法。那这些专门的方法都是什么呢，看后面的内容：

3、外链接之左连接：优先显示左表全部记录

```
#以左表为准，即找出所有员工信息，当然包括没有部门的员工
#本质就是：在内连接的基础上增加左边有右边没有的结果  #注意语法：
mysql> select employee.id,employee.name,department.name as depart_name from employee left join department on emp
+----+-------------+--------------+
| id | name        | depart_name  |
+----+-------------+--------------+
|  1 | egon        | 技术         |
|  5 | liwenzhou   | 技术         |
|  2 | alex        | 人力资源     |
|  3 | wupeiqi     | 人力资源     |
|  4 | yuanhao     | 销售         |
|  6 | jingliyang  | NULL         |
+----+-------------+--------------+
```

4、外链接之右连接：优先显示右表全部记录

```
#以右表为准，即找出所有部门信息，包括没有员工的部门
#本质就是：在内连接的基础上增加右边有左边没有的结果
mysql> select employee.id,employee.name,department.name as depart_name from employee right join department on er
+------+-----------+-------------+
| id   | name      | depart_name |
+------+-----------+-------------+
|    1 | egon      | 技术        |
|    2 | alex      | 人力资源    |
|    3 | wupeiqi   | 人力资源    |
|    4 | yuanhao   | 销售        |
|    5 | liwenzhou | 技术        |
| NULL | NULL      | 运营        |
+------+-----------+-------------+
```

5、全外连接：显示左右两个表全部记录

```
全外连接：在内连接的基础上增加左边有右边没有的和右边有左边没有的结果
#注意：mysql不支持全外连接 full JOIN
#强调：mysql可以使用此种方式间接实现全外连接
select * from employee left join department on employee.dep_id = department.id
union
select * from employee right join department on employee.dep_id = department.id
;
#查看结果
+------+------------+--------+------+--------+------+--------------+
| id   | name       | sex    | age  | dep_id | id   | name         |
+------+------------+--------+------+--------+------+--------------+
|    1 | egon       | male   |   18 |    200 |  200 | 技术         |
|    5 | liwenzhou  | male   |   18 |    200 |  200 | 技术         |
|    2 | alex       | female |   48 |    201 |  201 | 人力资源     |
|    3 | wupeiqi    | male   |   38 |    201 |  201 | 人力资源     |
|    4 | yuanhao    | female |   28 |    202 |  202 | 销售         |
|    6 | jingliyang | female |   18 |    204 | NULL | NULL         |
| NULL | NULL       | NULL   | NULL |   NULL |  203 | 运营         |
+------+------------+--------+------+--------+------+--------------+

#注意 union与union all的区别：union会去掉相同的纪录，因为union all是left join 和right join合并，所以有重复的记录，通过uni
```

三 符合条件连接查询

```
#示例1：以内连接的方式查询employee和department表，并且employee表中的age字段值必须大于25,即找出年龄大于25岁的员工i
select employee.name,department.name from employee inner join department
    on employee.dep_id = department.id
    where age > 25;

#示例2：以内连接的方式查询employee和department表，并且以age字段的升序方式显示
select employee.id,employee.name,employee.age,department.name from employee,department
    where employee.dep_id = department.id
```

```
    and age > 25
    order by age asc;
```

四 子查询

子查询其实就是将你的一个查询结果用括号括起来，这个结果也是一张表，就可以将它交给另外一个sql语句，作为它的一个查询依据来进行操作。

来，我们简单来个需求：技术部都有哪些员工的姓名，都显示出来：　1、看一下和哪个表有关，然后from找到两个表　2、进行一个连表操作　3、基于连表的结果来一个过滤就可以了

```
#我们之前的做法是：先连表
mysql> select * from employee inner join department on employee.dep_id = department.id;
+----+-----------+--------+------+--------+------+--------------+
| id | name      | sex    | age  | dep_id | id   | name         |
+----+-----------+--------+------+--------+------+--------------+
|  1 | egon      | male   |   18 |    200 |  200 | 技术         |
|  2 | alex      | female |   48 |    201 |  201 | 人力资源     |
|  3 | wupeiqi   | male   |   38 |    201 |  201 | 人力资源     |
|  4 | yuanhao   | female |   28 |    202 |  202 | 销售         |
|  5 | liwenzhou | male   |   18 |    200 |  200 | 技术         |
+----+-----------+--------+------+--------+------+--------------+
5 rows in set (0.10 sec)

#然后根据连表的结果进行where过滤，将select*改为select employee.namemysql> select employee.name from employee inne
```

```
+-----------+
| name      |
+-----------+
| egon      |
| liwenzhou |
+-----------+
2 rows in set (0.09 sec)
```

然后看一下子查询这种方式的写法：它的做法就是解决完一个问题，再解决下一个问题，针对我们上面的需求，你想，我们的需求是不是说找技术部门下面有哪些员工对不对，如果你直接找员工表，你能确定哪个dep_id的数值表示的是技术部门吗，不能，所以咱们是不是应该先确定一个技术部门对应的id号是多少，然后根据部门的id号，再去员工表里面查询一下dep_id为技术部门对应的部门表的那个id号的所有的员工表里面的记录：好，那我们看一下下面的操作

```
#首先从部门表里面找到技术部门对应的id
mysql> select id from department where name='技术';
+------+
| id   |
+------+
|  200 |
+------+
1 row in set (0.00 sec)

#那我们把上面的查询结果用括号括起来，它就表示一条id=200的数据，然后我们通过员工表来查询dep_id=这条数据作为条件来查询
mysql> select name from employee where dep_id = (select id from department where name='技术');
+-----------+
| name      |
```

```
+-----------+
| egon      |
| liwenzhou |
+-----------+
2 rows in set (0.00 sec)上面这些就是子查询的一个思路，解决一个问题，再解决另外一个问题，你子查询里面可不可以是多个表的查
```

```
子查询：#1：子查询是将一个查询语句嵌套在另一个查询语句中。
#2：内层查询语句的查询结果，可以为外层查询语句提供查询条件。
#3：子查询中可以包含：IN、NOT IN、ANY、ALL、EXISTS 和 NOT EXISTS等关键字
#4：还可以包含比较运算符：= 、 !=、 > 、<等
```

1、带IN关键字的子查询

```
#查询员工平均年龄在25岁以上的部门名，可以用连表，也可以用子查询，我们用子查询来搞一下
select id,name from department
   where id in
      (select dep_id from employee group by dep_id having avg(age) > 25);
#连表来搞一下上面这个需求
select department.name from department inner join employee on department.id=employee.dep_id
   group by department.name
   having avg(age)>25;
总结：子查询的思路和解决问题一样，先解决一个然后拿着这个的结果再去解决另外一个问题，连表的思路是先将两个表关联在一起，

#查看技术部员工姓名
select name from employee
   where dep_id in
      (select id from department where name='技术');

#查看不足1人的部门名(子查询得到的是有人的部门id)
select name from department where id not in (select distinct dep_id from employee);
```

2、带比较运算符的子查询

```
#比较运算符：=、!=、>、>=、<、<=、<>
#查询大于所有人平均年龄的员工名与年龄
mysql> select name,age from emp where age > (select avg(age) from emp);
+---------+------+
| name | age |
+---------+------+
| alex | 48 |
| wupeiqi | 38 |
+---------+------+
2 rows in set (0.00 sec)


#查询大于部门内平均年龄的员工名、年龄
select t1.name,t1.age from emp t1
inner join
(select dep_id,avg(age) avg_age from emp group by dep_id) t2
on t1.dep_id = t2.dep_id
where t1.age > t2.avg_age;
```

3、**带EXISTS关键字的子查询**

EXISTS关字键字表示存在。在使用EXISTS关键字时，内层查询语句不返回查询的记录。而是返回一个真假值。True或False

当返回True时，外层查询语句将进行查询；当返回值为False时，外层查询语句不进行查询。还可以写not exists，和exists的效果就是反的

```
#department表中存在dept_id=203，Ture
mysql> select * from employee
    ->     where exists
    ->         (select id from department where id=200);
+----+------------+--------+------+--------+
| id | name       | sex    | age  | dep_id |
+----+------------+--------+------+--------+
|  1 | egon       | male   |   18 |    200 |
|  2 | alex       | female |   48 |    201 |
|  3 | wupeiqi    | male   |   38 |    201 |
|  4 | yuanhao    | female |   28 |    202 |
|  5 | liwenzhou  | male   |   18 |    200 |
|  6 | jingliyang | female |   18 |    204 |
+----+------------+--------+------+--------+

#department表中存在dept_id=205，False
mysql> select * from employee
    ->     where exists
    ->         (select id from department where id=204);
Empty set (0.00 sec)
```

练习：通过连表的方式来查询每个部门最新入职的那位员工

```
company.employee
    员工id      id              int
    姓名        emp_name        varchar
    性别        sex             enum
    年龄        age             int
    入职日期     hire_date       date
    岗位        post            varchar
    职位描述     post_comment    varchar
    薪水        salary          double
    办公室       office          int
    部门编号     depart_id       int



#创建表，只需要创建这一张表
create table employee(
id int not null unique auto_increment,
name varchar(20) not null,
sex enum('male','female') not null default 'male', #大部分是男的
age int(3) unsigned not null default 28,
```

```
hire_date date not null,
post varchar(50),
post_comment varchar(100),
salary double(15,2),
office int, #一个部门一个屋子
depart_id int
);


#查看表结构
mysql> desc employee;
+-------------+----------------------+------+-----+---------+----------------+
| Field       | Type                 | Null | Key | Default | Extra          |
+-------------+----------------------+------+-----+---------+----------------+
| id          | int(11)              | NO   | PRI | NULL    | auto_increment |
| name        | varchar(20)          | NO   |     | NULL    |                |
| sex         | enum('male','female') | NO   |     | male    |                |
| age         | int(3) unsigned      | NO   |     | 28      |                |
| hire_date   | date                 | NO   |     | NULL    |                |
| post        | varchar(50)          | YES  |     | NULL    |                |
| post_comment | varchar(100)        | YES  |     | NULL    |                |
| salary      | double(15,2)         | YES  |     | NULL    |                |
| office      | int(11)              | YES  |     | NULL    |                |
| depart_id   | int(11)              | YES  |     | NULL    |                |
+-------------+----------------------+------+-----+---------+----------------+

#插入记录
#三个部门：教学，销售，运营
insert into employee(name,sex,age,hire_date,post,salary,office,depart_id) values
('egon','male',18,'20170301','老男孩驻沙河办事处外交大使',7300.33,401,1), #以下是教学部
('alex','male',78,'20150302','teacher',1000000.31,401,1),
('wupeiqi','male',81,'20130305','teacher',8300,401,1),
('yuanhao','male',73,'20140701','teacher',3500,401,1),
('liwenzhou','male',28,'20121101','teacher',2100,401,1),
('jingliyang','female',18,'20110211','teacher',9000,401,1),
('jinxin','male',18,'19000301','teacher',30000,401,1),
('成龙','male',48,'20101111','teacher',10000,401,1),

('歪歪','female',48,'20150311','sale',3000.13,402,2),#以下是销售部门
('丫丫','female',38,'20101101','sale',2000.35,402,2),
('丁丁','female',18,'20110312','sale',1000.37,402,2),
('星星','female',18,'20160513','sale',3000.29,402,2),
('格格','female',28,'20170127','sale',4000.33,402,2),

('张野','male',28,'20160311','operation',10000.13,403,3), #以下是运营部门
('程咬金','male',18,'19970312','operation',20000,403,3),
('程咬银','female',18,'20130311','operation',19000,403,3),
('程咬铜','male',18,'20150411','operation',18000,403,3),
('程咬铁','female',18,'20140512','operation',17000,403,3)
;

#ps：如果在windows系统中，插入中文字符，select的结果为空白，可以将所有字符编码统一设置成gbk
```

答案：

```
SELECT
    *
```

```
FROM
    emp AS t1
INNER JOIN (   #和虚拟表进行连表
    SELECT
        post,
        max(hire_date) as max_date  #给这个最大的日期取个别名叫做max_date，先将每个部门最近入职的最大的日期的信息筛选出
    FROM
        emp
    GROUP BY
        post
) AS t2 ON t1.post = t2.post  #给虚拟表取个别名叫做t2
WHERE
    t1.hire_date = t2.max_date;  #然后再通过where来过滤出，入职日期和最大日期相等的记录，就是我们要的内容
```

五 综合练习

表结构为

班级表：class

cid	caption
1	三年二班
2	一年三班
3	三年一班

学生表：student

sid	sname	gender	class_id
1	钢蛋	女	1
2	铁锤	女	1
3	山炮	男	2

老师表：teacher

tid	tname
1	波多
2	苍空
3	饭岛

课程表：course

cid	cname	tearch_id
1	生物	1
2	体育	1
3	物理	2

成绩表：score

sid	student_id	corse_id	number
1	1	1	60
2	1	2	59
3	2	2	100

```
#创建表及插入记录
CREATE TABLE class (
  cid int(11) NOT NULL AUTO_INCREMENT,
  caption varchar(32) NOT NULL,
  PRIMARY KEY (cid)
) ENGINE=InnoDB CHARSET=utf8;

INSERT INTO class VALUES
(1, '三年二班'),
(2, '三年三班'),
(3, '一年二班'),
(4, '二年九班');

CREATE TABLE course(
  cid int(11) NOT NULL AUTO_INCREMENT,
  cname varchar(32) NOT NULL,
  teacher_id int(11) NOT NULL,
  PRIMARY KEY (cid),
```

```
  KEY fk_course_teacher (teacher_id),
  CONSTRAINT fk_course_teacher FOREIGN KEY (teacher_id) REFERENCES teacher (tid)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO course VALUES
(1, '生物', 1),
(2, '物理', 2),
(3, '体育', 3),
(4, '美术', 2);

CREATE TABLE score (
  sid int(11) NOT NULL AUTO_INCREMENT,
  student_id int(11) NOT NULL,
  course_id int(11) NOT NULL,
  num int(11) NOT NULL,
  PRIMARY KEY (sid),
  KEY fk_score_student (student_id),
  KEY fk_score_course (course_id),
  CONSTRAINT fk_score_course FOREIGN KEY (course_id) REFERENCES course (cid),
  CONSTRAINT fk_score_student FOREIGN KEY (student_id) REFERENCES student(sid)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO score VALUES
(1, 1, 1, 10),
(2, 1, 2, 9),
(5, 1, 4, 66),
(6, 2, 1, 8),
(8, 2, 3, 68),
(9, 2, 4, 99),
(10, 3, 1, 77),
(11, 3, 2, 66),
(12, 3, 3, 87),
(13, 3, 4, 99),
(14, 4, 1, 79),
(15, 4, 2, 11),
(16, 4, 3, 67),
(17, 4, 4, 100),
(18, 5, 1, 79),
(19, 5, 2, 11),
(20, 5, 3, 67),
(21, 5, 4, 100),
(22, 6, 1, 9),
(23, 6, 2, 100),
(24, 6, 3, 67),
(25, 6, 4, 100),
(26, 7, 1, 9),
(27, 7, 2, 100),
(28, 7, 3, 67),
(29, 7, 4, 88),
(30, 8, 1, 9),
(31, 8, 2, 100),
(32, 8, 3, 67),
(33, 8, 4, 88),
(34, 9, 1, 91),
(35, 9, 2, 88),
(36, 9, 3, 67),
(37, 9, 4, 22),
(38, 10, 1, 90),
(39, 10, 2, 77),
(40, 10, 3, 43),
(41, 10, 4, 87),
(42, 11, 1, 90),
```

```
(43, 11, 2, 77),
(44, 11, 3, 43),
(45, 11, 4, 87),
(46, 12, 1, 90),
(47, 12, 2, 77),
(48, 12, 3, 43),
(49, 12, 4, 87),
(52, 13, 3, 87);


CREATE TABLE student(
  sid int(11) NOT NULL AUTO_INCREMENT,
  gender char(1) NOT NULL,
  class_id int(11) NOT NULL,
  sname varchar(32) NOT NULL,
  PRIMARY KEY (sid),
  KEY fk_class (class_id),
  CONSTRAINT fk_class FOREIGN KEY (class_id) REFERENCES class (cid)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO student VALUES
(1, '男', 1, '理解'),
(2, '女', 1, '钢蛋'),
(3, '男', 1, '张三'),
(4, '男', 1, '张一'),
(5, '女', 1, '张二'),
(6, '男', 1, '张四'),
(7, '女', 2, '铁锤'),
(8, '男', 2, '李三'),
(9, '男', 2, '李一'),
(10, '女', 2, '李二'),
(11, '男', 2, '李四'),
(12, '女', 3, '如花'),
(13, '男', 3, '刘三'),
(14, '男', 3, '刘一'),
(15, '女', 3, '刘二'),
(16, '男', 3, '刘四');

CREATE TABLE teacher(
  tid int(11) NOT NULL AUTO_INCREMENT,
  tname varchar(32) NOT NULL,
  PRIMARY KEY (tid)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO teacher VALUES
(1, '张磊老师'),
(2, '李平老师'),
(3, '刘海燕老师'),
(4, '朱云海老师'),
(5, '李杰老师');
```


！！！重中之重：练习之前务必搞清楚sql逻辑查询语句的执行顺序

链接：https://www.cnblogs.com/clschao/articles/9995517.html

1、查询所有的课程的名称以及对应的任课老师姓名

2、查询学生表中男女生各有多少人

3、查询物理成绩等于100的学生的姓名

4、查询平均成绩大于八十分的同学的姓名和平均成绩

5、查询所有学生的学号，姓名，选课数，总成绩

6、 查询姓李老师的个数

7、 查询没有报李平老师课的学生姓名

8、 查询物理课程比生物课程高的学生的学号

9、 查询没有同时选修物理课程和体育课程的学生姓名

10、查询挂科超过两门(包括两门)的学生姓名和班级

11 、查询选修了所有课程的学生姓名

12、查询李平老师教的课程的所有成绩记录

13、查询全部学生都选修了的课程号和课程名

14、查询每门课程被选修的次数

15、查询之选修了一门课程的学生姓名和学号

16、查询所有学生考出的成绩并按从高到低排序（成绩去重）

17、查询平均成绩大于85的学生姓名和平均成绩

18、查询生物成绩不及格的学生姓名和对应生物分数

19、查询在所有选修了李平老师课程的学生中，这些课程(李平老师的课程，不是所有课程)平均成绩最高的学生姓名

20、查询每门课程成绩最好的前两名学生姓名

21、查询不同课程但成绩相同的学号，课程号，成绩

22、查询没学过“叶平”老师课程的学生姓名以及选修的课程名称；

23、查询所有选修了学号为1的同学选修过的一门或者多门课程的同学学号和姓名；

24、任课最多的老师中学生单科成绩最高的学生姓名

参考答案:https://www.cnblogs.com/clschao/articles/9995776.html