【复习】多表查询：

1）子查询

使用场景：如果题目给你的查询信息里涉及了多张表，且所在查询的字段信息只来自于一张表，就可以考虑使用子查询。

2）关联查询

使用场景：如果题目给你的查询信息里涉及了多张表，且需要同时显示多张表里的字段信息，就可以考虑使用关联查询

笛卡儿积运算:多表组合的方式 ——》from a,b
内联接：返回的是满足关联条件的查询结果集 inner join
外联接：涉及到需要完整的去展示某张表的所有数据 left/right/full join

重点：

from a,b——》a 表和 b 表做关联，会将 a 和 b 合成一张大的表，合并的方式是做笛卡儿积运算
——》这种运算会产生没有意义的垃圾数据——》可以通过添加关联条件的方式来消除垃圾数据

【作业】

请根据以下表间关系（如图所示）写出如下 select 语句：

每个业务日期的“状态为派单，送出和结账” 的订单金额合计值。

方法 1：

(1)求出状态名称为派单，送出和结账所对应的状态 id

```
select status_id from status where status_name in("派单","送出","结账");
select status_id from status where status_name="派单" or status_name="送出"
or status_name="结账";
```

（2）每个业务日期的“状态为派单，送出和结账” 的订单金额合计值

```
select order_date,sum(amount)
```

```
from order_header
where status_id in(select status_id from status where status_name in("派单","送出","结账"))
group by order_date;
```

方法 2：

```
select o.order_date,sum(o.amount)
from status s,order_header o
where s.status_id=o.status_id
and s.status_name in("派单","送出","结账")
group by o.order_date;
```

在表 A 中有如下列： program_id,program_start_time,qc_state,chanel_id,
请使用 sql 语句找出 chanel_id 为 10，qc_state 为 5 的 program_id,
并使其按照 program_start_time 进行排序；

```
select program_id
from a
where chanel_id=10 and qc_state=5
order by program_start_time asc;
```

在表 B 中有如下列： Ticket_id,program_id,device_id,
请使用 sql 语句找出表 B 中 program_id 与上一题查询结果相符合的 ticket_id 和 device_id

方法 1：

```
select ticket_id,device_id
from b
where program_id in(
    select program_id
    from a
    where chanel_id=10 and qc_state=5
    order by program_start_time asc
);
```

方法 2：

```
select b.ticket_id,b.device_id
from a,b
where a.program_id=b.program_id
and a.chanel_id=10 and a.qc_state=5;
```

【三表关联】：本质来说表之间两两合并，最终生成了一张大的表

假如需要查看的信息来自于三张表，例如要知道某个学员的名字，修的课程名，以及得分。

```
select *
from xsb,cjb,kcb
where xsb.xh=cjb.xh
and cjb.kch=kcb.kch;

select *
from xsb inner join cjb on xsb.xh=cjb.xh
inner join kcb on kcb.kch=cjb.kch;
```

【既有表的关联又有子查询的示例】

查询 java 语言课的学员的名字和成绩

（1）查询 java 语言课对应的课程号

```
select kch from kcb where kcm='Java 语言';
```

（2）查询 java 语言课的学员的名字和成绩

```
select xsb.xm,cjb.cj
from xsb,cjb
where xsb.xh=cjb.xh
and cjb.kch=(select kch from kcb where kcm='Java 语言');
```

MySQL 的高级特性（事务、存储过程、触发器）

1、事务(transaction)

(1)什么是事务？

例如：a 要给 b 转账 100 块，本质是对两个人的账户余额做更新操作：

```
update xx set ye=ye-100 where countname='a';
update xx set ye=ye+100 where countname='b';
```

需求：对这样的一组操作，要求这一组语句要么同时成功，要么同时失败，

否则会引发纠纷。

在这里，转账就是事务。事务指的是一组 DML 操作（update/delete/insert），只允许它们要么同时成功，要么同时失败。

（2）事务的 4 个特征:

1)原子性：事务中的所有操作（一组 SQL 语句）被看成一个整体，不可分割
2）隔离性：一个事务在其操作期间，别的事务不可以对其进行干扰
3）永久性：事务处理结束后，对数据的修改是永久的
4）一致性：在事务开始之前和事务结束之后，数据库的完整性没有被破坏

（3）事务的代码实现

——**MySQL 开启事务：**start transaction;
——**事务提交：**commit;
——**事务回滚：**rollback; (没有提交的事务允许回滚)

（4）事务的示例

数据准备：

```
create table account(
    id int auto_increment primary key,
    name varchar(20) not null,
    money float
);

insert into account(name,money) values('a',1000),('b',1000);

select * from account;
```

注意：MySQL 的默认设置，事务都是自动提交的，即执行 MySQL 语句之后会马上 commit，
如果你希望不自动提交，可使用 start transaction 来显式的开启事务，或者执行 set autocommit=0 来禁止自动提交。

```
update account set money=money-100 where name='a';
update account set money=money-100 where name='b';
```

如果不希望每次更新都自动将更新结果提交到服务器端，那么可以手动开启自动提交：

```
start transaction;
update account set money=money-100 where name='a';
```

```
update account set money=money+100 where name='b';
rollback;
commit;
```

2、存储过程

（1）需求：

比如遇到一些 sql 语句使用频繁、需要反复的使用，每次去编写比较耗费时间，比较好的一种做法是：给这组 sql 语句取个名字，以后需要使用 sql 语句时，通过这个名字对
它进行调用。

存储过程（procedure）的作用:将常用的 sql 语句存放起来，方便以后重复使用。
存储过程：是一段有名字的代码，用来完成一个特定的功能

（2）如何编写存储过程

语法格式：

create procedure 存储过程名([形式参数])
begin
常用的 sql 语句;
end

调用存储过程：

call 存储过程名([实际参数])

示例 1：不带参数的存储过程

数据准备：

```
create table student(
    id int primary key auto_increment,
    name varchar(20) not null,
    grade float,
    gender char(4)
);

insert into student(name,grade,gender)
values('jack',25,'男'),('rose',23,'女'),('Lucy',24,'女');

select * from student;
```

创建并调用存储过程：

```
create procedure p_stu()
begin
    select * from student;
end

call p_stu()
```

注意：数据库里的标识符命名规则：由字母、数字和下划线所组成，
其中数字不打头，不能是关键字。

示例 2：带参数的存储过程

存储过程支持输入参数和输出参数，参数存放在存储过程名字后面的小括号里。

需求：

用户在调用存储过程的时候 ，输入女生时，返回女生的数量，
输入是男生时，返回的是男生的数量。——》即根据不同的输入，得到不同的输出。

定义存储过程：

```
create procedure p_stu_cou(in s_gender char(4),out num int) #in 后面输入  out 输出
begin
    select count(*) into num from student where gender=s_gender;
End
```

注解: select 后面是统计的意思,into 是输出哪个参数,然后 from 是调用哪个表,where 是调用 student 内的 gender 传给 s_gender 内,最后调用

调用

```
call p_stu_cou('女',@num);
select @num  女生数量;
call p_stu_cou('男',@num);
select @num  男生数量;
```

3、触发器

需求：

我们希望当我们对一张表的内容做改动的时候能够触发另外一个表也做一些内容
的改动。

触发的过程就是触发器。

创建触发器的语法：

```
create trigger 触发器名 触发的时机 监视的事件 on 表名 for each row
触发的事件;
```

注解：

触发的时机的取值：before/after
触发的事件/监视的事件：insert/update/delete

示例：当我们往 student 表插入数据时，它会触发往 timelog 表里插入当前时间。

创建 timelog 表：

```
create table timelog(
    id int primary key auto_increment,
    savetime datetime
);

select * from timelog;
```

创建触发器：

```
create trigger t_insert before insert on student
for each ROW
insert into timelog(savetime) values(now());
```

触发器的效果：

当你执行 **student** 表的插入操作：

```
insert into student(name) values('wangyibo');
```

```
select * from timelog;
```

注意：

触发器是在一张表中做操作（插入、修改、删除，不能是查询操作，查询操作无法引发触发），引发另外一张表的操作。

补充:

主外键关联：通常是同名字段关联（主键和外键取名是一样的），但是需要注意的是，
外键和主键的名字并不是非得一样，即主键和外键的名字可以不同，也不是说非得在
两张不同的表里。

top-n 分析：

将成绩前三名的信息展示出来：

```
select * from cjb order by cj desc
limit 3;
```

也可以:

```
select * from cjb order by cj desc
limit 1 3; ===>表示从第二个开始数三个,默认第一个为 0
```