翻转课堂

关系代数

Sailors(sid: integer, sname: string, rating: integer, age: real)
Boats(bid: integer, bname: string, color: string)
Reserves(sid: integer, bid: integer, day: data)

1.查找租用过103号船的所有水手的名字：有多种解决方法;

方法一： $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$

方法二： $\pi_{sname}((\pi_{sid}(\sigma_{bid=103}(Reserves))) \bowtie Sailors)$

方法三： $\pi_{sname}(\sigma_{bid=103}((\pi_{sid,sname}(Sailors)) \bowtie (\pi_{sid,bid}(Reserves))))$

2.查找租用过红色船只的水手的名字;

方法一： $\pi_{sname}(\sigma_{color='red'}(Boats \bowtie Reserves \bowtie Sailors))$

方法二： $\pi_{sname}(\pi_{sid}(\pi_{bid}(\sigma_{color='red'}(Boats)) \bowtie Reserves) \bowtie Sailors)$

3.查找租用过红色或绿色船只的水手的名字;

方法一： $\pi_{sname}(\sigma_{color='red' \cup color='green'}(Boats \bowtie Reserves \bowtie Sailors))$

方法二： $\pi_{sname}(\pi_{sid}(\pi_{bid}(\sigma_{color='red' \cup color='green'}(Boats)) \bowtie Reserves) \bowtie Sailors)$

4.查找租用过红色和绿色船只的水手的名字;

方法一： $(\pi_{sname}(\sigma_{color='red'}(Boats \bowtie Reserve \bowtie Sailors))) \cap$
$(\pi_{sname}(\sigma_{color='red'}(Boats \bowtie Reserves \bowtie Sailors)))$

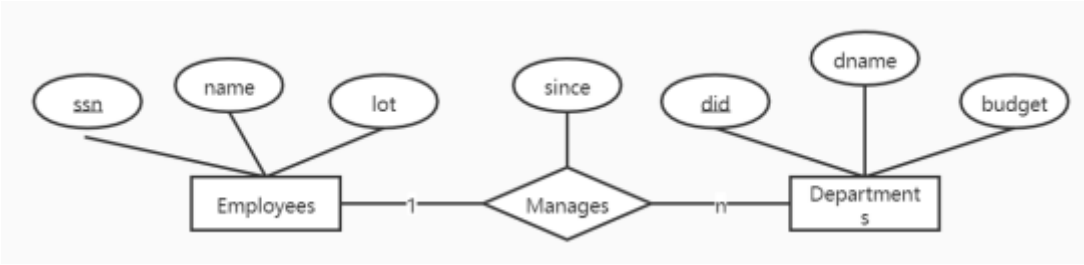
方法二： $(\pi_{sname}(\pi_{sid}(\pi_{bid}(\sigma_{color='red'}(Boats)) \bowtie Reserves) \bowtie (Sailors)) \cap$
$(\pi_{sname}(\pi_{sid}(\pi_{bid}(\sigma_{color='green'}(Boats)) \bowtie Reserves) \bowtie (Sailors))$

5.查找租用过所有船只的水手的名;

$\pi_{sname}(((\pi_{bid,sid}(Reserves)))/(\pi_{bid}(Boats)) \bowtie Sailors)$

一对多联系

基本E-R模型的转换——一对多联系



方法一：将一对一联系转换为一个独立的关系表

关系模式: Manages (did, ssn, since)
CREATE TABLE Manages(
ssn CHAR(11),
did INTEGER,
since DATE,
PRIMARY KEY (did),
FOREIGN KEY (ssn) REFERENCES Employees,
FOREIGN KEY (did) REFERENCES Departments)

为何方法一中primary key少了ssn?

primary key，即主码，主码是一个能唯一标识一个元组的属性。而因为雇员与部门之间的联系是一对多的，即不存在一个部门对应多个雇员的情况，所以雇员不需要作为主码属性。

方法二：给出ER到关系的另一个转换方法，只用2个表表示该ER图。

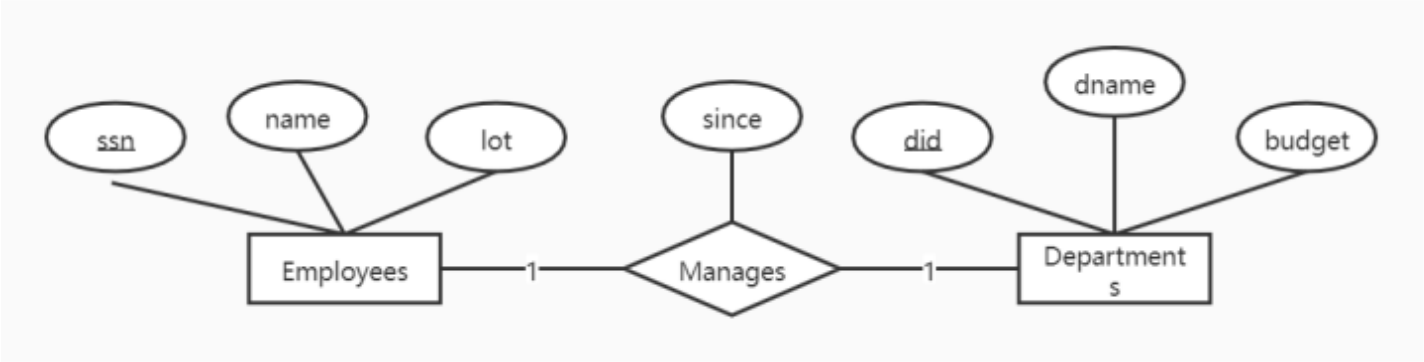
第一张表：Employees
关系模式: Employees (ssn, name, lot)
CREATE TABLE Employees (
ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn)
)

第二张表：Dept_Mgr
关系模式: Dept_Mgr (did, dname, budget, ssn, since)
CREATE TABLE Dept_Mgr (
did INTEGER,
dname CHAR(20),
budget REAL,
ssn CHAR(11),
since DATE,

```
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees
)
```

一对一联系

基本E-R模型的转换——一对一联系



方法一：将一对一联系转换为一个独立的关系表

```
CREATE TABLE Manages(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (did, ssn),
    FOREIGN KEY (ssn) REFERENCES Employees,
    FOREIGN KEY (did) REFERENCES Departments)
```

方法二：对于1：1联系的情况，是否可以做以下转换?
将2个实体和1个联系转换为一张大表。

不可以。原因如下：

1.一对一关系也包括两部分没有联系的，合在一起就浪费空间了;(主要原因）即双方存在未参与的

2.不便于管理（包括查询执行速度）

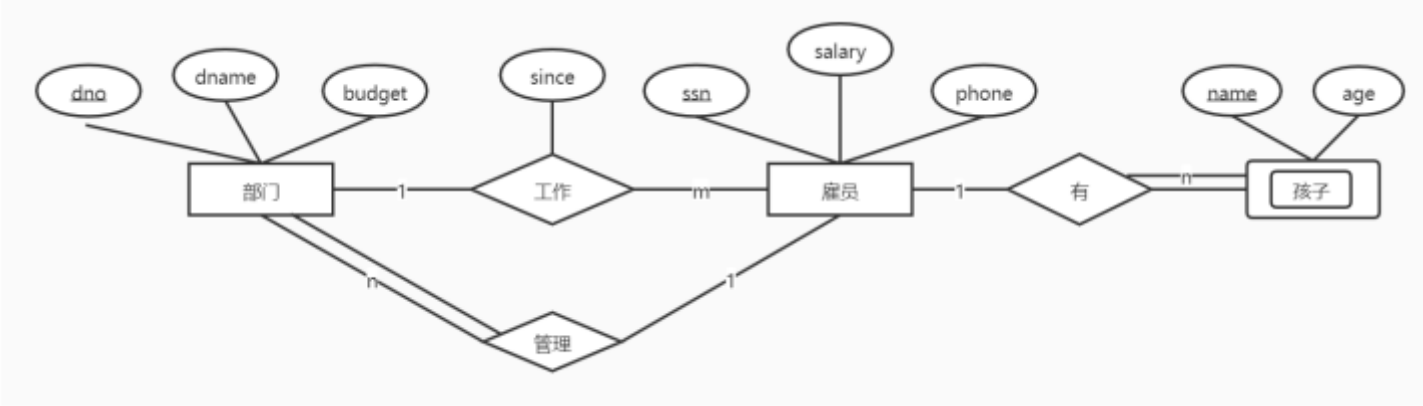
3.也就达不到关系型数据库的特性

4.可更好对业务进行事务隔离操作

弱实体、全参与

一个公司数据库需要存储雇员、部门和雇员小孩的信息。雇员工作在部门(一个雇员只能工作在一个部门)，每个部门由一个雇员管理，每个雇员小孩的名字是唯一的，假定小孩只有一个家长工作在这个公

司，而且我们不关心那些已经调离雇员的小孩情况。



部门与雇员工作联系(ssn, salary, phone, dno) 基本一对多的关系转换

部门与雇员管理联系(dno, dname, budget, ssn) 具有参加约束的n:1联系

雇员与小孩联系(ssn ,name, age) 具有弱实体集的转换

1.基本一对多的关系转换：

关系模式:
Emp_Works (ssn,salary,phone,don)

```
CREATE TABLE Emp_Works(
    ssn CHAR(11),
    salary INTEGER,
    phone CHAR(11),
    don INTEGER,
    PRIMARY KEY (ssn),
    FOREIGN KEY (don) REFERENCES Departments)
```

2.具有参加约束的1:n联系集的转换

关系模式:
Dept_Mgr (dno,dname,budget,ssn)

```
CREATE TABLE Dept_Mgr(
    dno INTEGER,
    dname CHAR(11),
    budget Integer,
    ssn CHAR(11) NOT NULL,
    PRIMARY KEY (dno),
    FOREIGN KEY (ssn) REFERENCES Employees)
```

3.具有弱实体集的转换

关系模式:
Dep_Policy (name,ssn,age)

```
CREATE TABLE Dep_Policy (
    name CHAR(20),
    ssn CHAR(11) NOT NULL,
    age INTEGER,
    PRIMARY KEY (name,ssn),
    FOREIGN KEY (ssn) REFERENCES Empolyees,
    ON DELETE CASCADE)
```

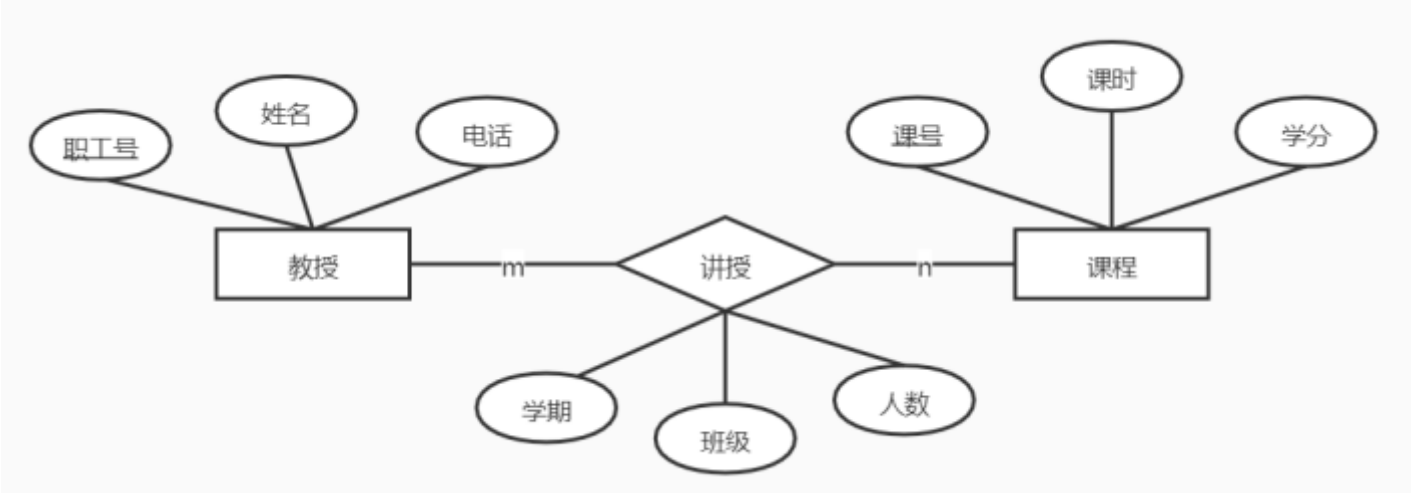
示例二

一个大学数据库包括教授、课程信息。教授讲授课程，下面几种情况都是描述有关讲授联系集的，对于每一种情况画ER图描述，并将其转换为关系模型。

1.教授可以在几个学期讲授同一门课程，但仅最近一次的讲授活动需被记录；

讲授（职工号，课号，学期，班级，人数）
教授（职工号，姓名，电话）
课程（课号，课时，学分）

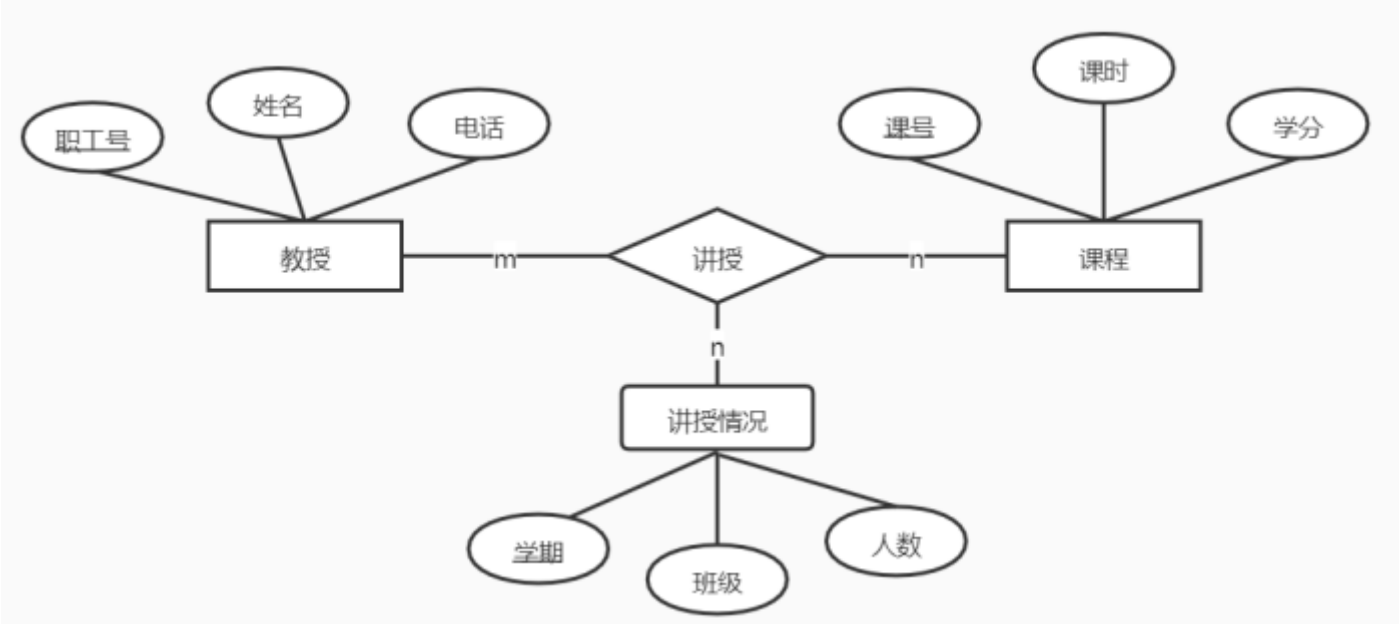
```
CREATE TABLE 讲授(
    职工号，
    课号,
    班级,
    学期，
    人数，
    PRIMARY KEY (职工号，课号),
    FOREIGN KEY (职工号) REFERENCES 教授,
    FOREIGN KEY (课号) REFERENCES 课程)
```



2.教授可以在几个学期讲授同一门课程，每次讲授活动需被记录下来；

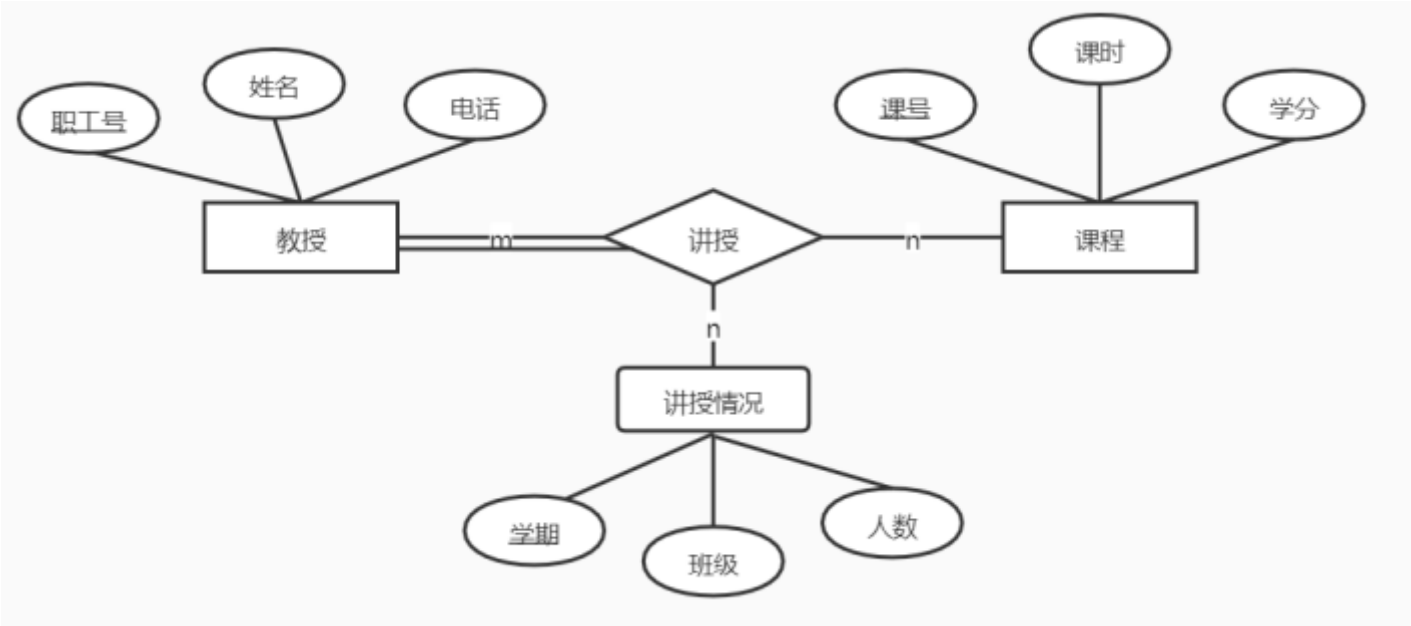
讲授（职工号，课号，学期）
讲授情况（学期，班级，人数）
教授（职工号，姓名，电话）
课程（课号，课时，学分）

CREATE TABLE 讲授 (
    职工号,
    课号,
    学期，
    PRIMARY KEY(职工号 , 课号，学期)，
    FOREIG KEY (职工号) REFERENCES 教授,
    FOREIG KEY (课号) REFERENCES 课程,
    FOREIGN KEY (学期) REFERENCE 讲授情况）



3.每个教授必须讲授课程；

CREATE TABLE 讲授 (
    职工号,
    课号,
    学期，
    PRIMARY KEY(职工号 , 课号，学期)，
    FOREIGN KEY (职工号) REFERENCES 教授,
    FOREIGN KEY (课号) REFERENCES 课程,
    FOREIGN KEY (学期) REFERENCE 讲授情况）

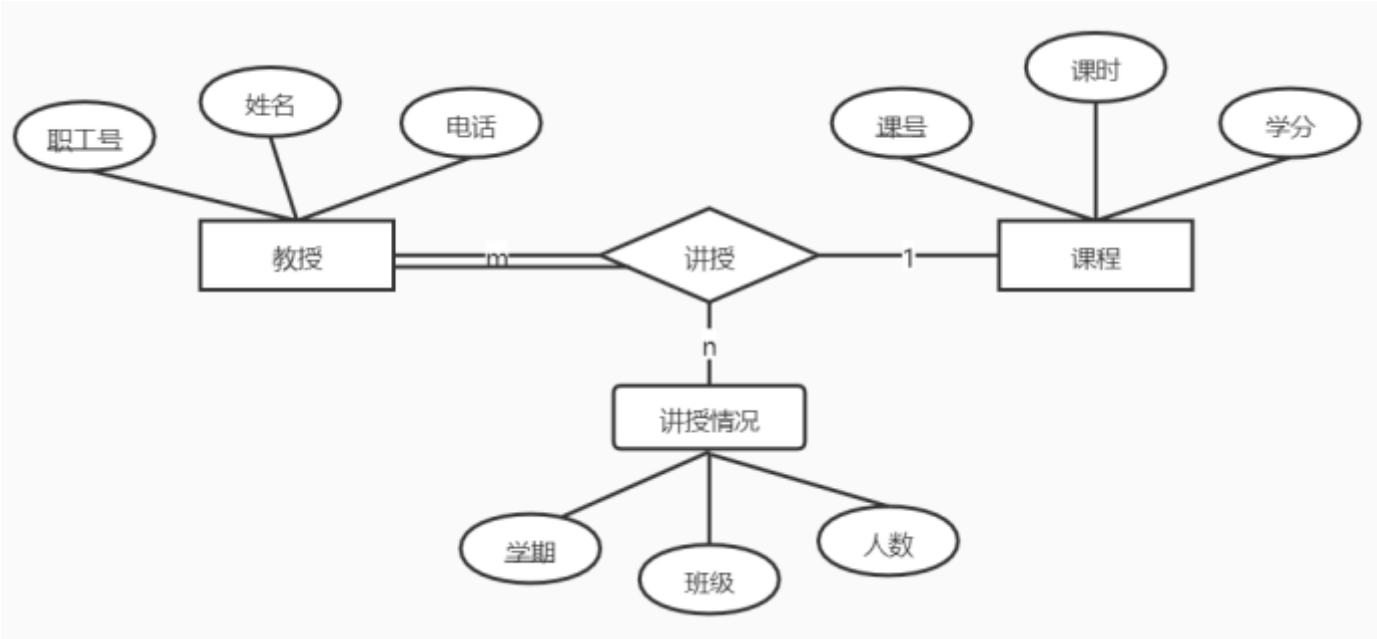


4.每个教授只讲授一门课程；

CREATE TABLE 讲授 (
职工号,
课号,
学期，
PRIMARY KEY(职工号 , 学期)，
FOREIGN KEY (职工号) REFERENCES 教授,
FOREIGN KEY (课号) REFERENCES 课程,
FOREIGN KEY (学期) REFERENCE 讲授情况）

5.每个教授只讲授一门课程，每门课程可有几位教授讲授；
讲授（职工号，课号，学期）
讲授情况（学期，班级，人数）
教授（职工号，姓名，电话）
课程（课号，课时，学分）

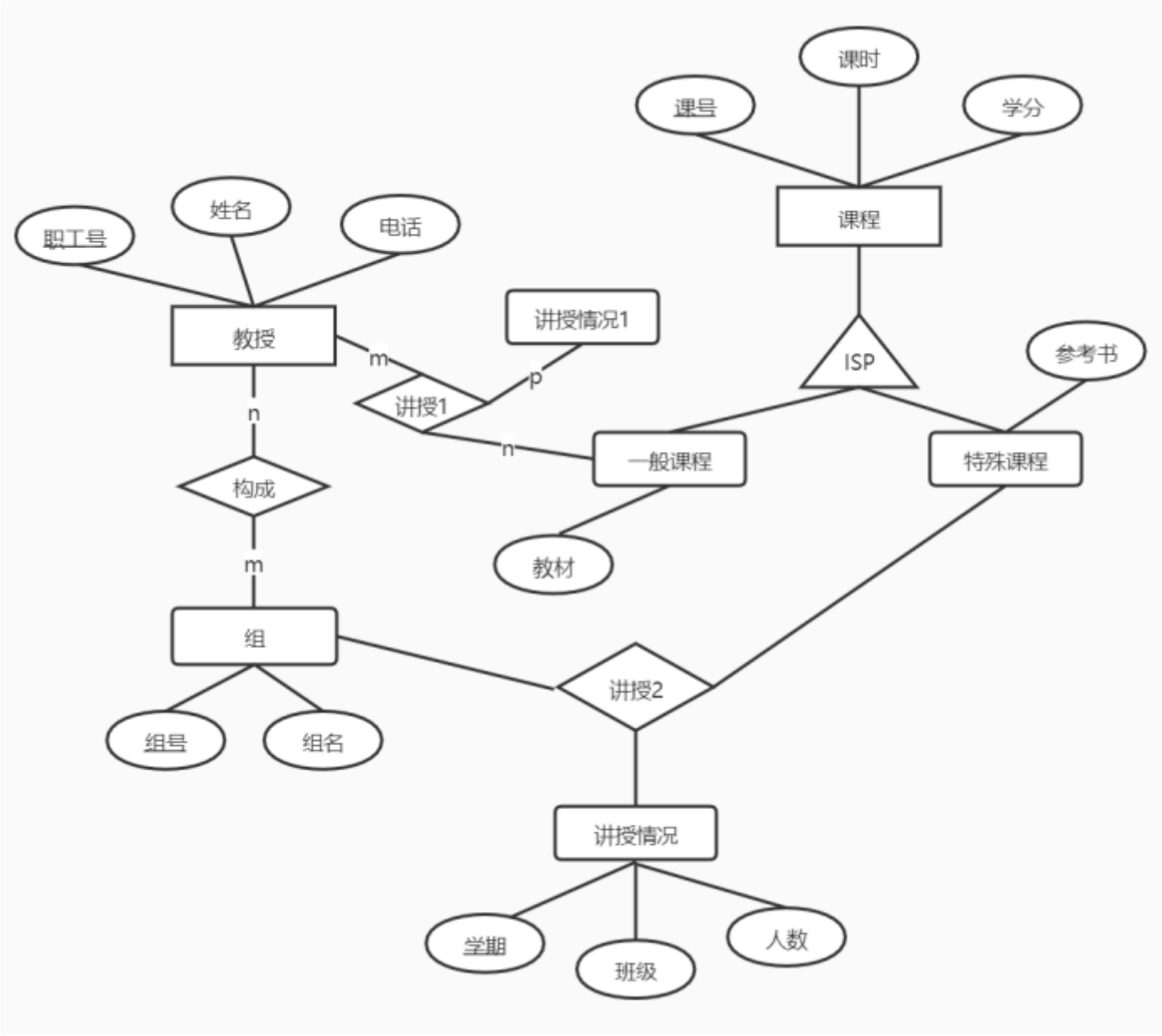
CREATE TABLE 讲授 (
职工号,
课号,
学期，
PRIMARY KEY(职工号 , 学期)，
FOREIGN KEY (职工号) REFERENCES 教授,
FOREIGN KEY (课号) REFERENCES 课程,
FOREIGN KEY (学期) REFERENCE 讲授情况）



6.假定一些课程可由一组教授联合讲授；

7.假定一些特定课程只能由一组教授联合讲授，且这些教授中的任一位不可能独立讲授这门课程。

教授(职工号, 姓名, 电话)
组（组号，组名）
构成（职工号，组号）
课程(课号, 课时, 学分)
一般课程（课号，教材）
特定课程（课号，参考书）
讲授情况1(学期，班级，人数)
讲授1（职工号,一般课号,学期）
讲授情况2（学期，班级，人数，组号，特定课号）
讲授2（组号,特定课号,学期）



CREATE TABLE 讲授2 (

组号,

课号,

学期，

PRIMARY KEY(组号, 学期, 课号)，

FOREIGN KEY (组号) REFERENCES 组,

FOREIGN KEY (课号) REFERENCES 课程,

FOREIGN KEY (学期) REFERENCE 讲授情况）

CREATE TABLE 构成 （

组号,

职工号，

PRIMARY KEY(组号, 职工号)，

FOREIGN KEY (组号) REFERENCES 组，
FOREIGN KEY (职工号) REFERENCES 教授）

SQL部分

学生表Student (Sno, Sname, Ssex, Sage, Sdept)
学生（学号，姓名，性别，年龄，所在系）
课程表Course (Cno, Cname, Cpno, Ccredit)
课程（课程号，课程名，先行课，学分）
学生选课表SC (Sno, Cno, Grade)
选课（学号，课程号，成绩）

1.查询选修了课程名为信息系统的学生学号和姓名

第一种方法：

```
select Sno, Sname
from Students
where Sno in(
            select Sno
            from SC
            where Cno in(
                        select Cno
                        from Courses
                        where Cname='信息系统'));
```

第二种方法：

```
select Sno, Sname
from Student join SC join Course
where Cname='信息系统';
```

2.带有比较运算符的子查询：子查询结果为单值的情况

例子：查询与刘晨在一个系学习的学生

```
select *
from Students
where Sdept in (select Sdept
                from Students
                where Sname='刘晨');
```

给出用in和=的对比实验结果。

结果相同，因为SQL允许子查询出现在返回单个值的表达式能够出现的任何地方，这是标量子查询

3.用集函数实现上述查询(效率比any和all高)

```
select Sname, Sage
from Students
where Sage < all(select Sage
                 from Students
                 where Sdept=‘IS’)
    and Sdept!=‘IS’ ;


select Sname, Sage
from Students
where Sage < (select min(Sage)
                 from Students
                 where Sdept=‘IS’)
    and Sdept!=‘IS’ ;
```

4.带有exists的子查询：子查询结果为true或false

查询没有选修了1号课程的学生姓名

```
select sname
from students, sc
where students.sno=sc.sno and sc.cno<>1


select Sname
     from Students
    where not exists (select *
                      from SC
                      where Sno=Students.Sno and Cno=‘1’);


select Sname
     from Students
     where exists (select *
                   from SC
                   where Sno=Students.Sno and
                   Cno != ‘1’);
```

上述查询是否正确?实验对比上面3种查询的结果

第一个不正确，因为有可能有学生既选了1号课也选了其他课

第二个正确

第三个不正确，理由同1

5.Sailors(sid , sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

查找租用过103号船只的水手的名字

```
select S.sname
from Sailors S
where S.sid in (select R.sid
                from Reserve R
                where R.bid=103);
```

不使用嵌套查询该怎么做?

```
select sname
from Sailors join Reserve
Where bid=103;
```

6.查找租用过红船和绿船的水手编号和姓名

错误的方法:

```
Select S.sid，S.sname
from Sailors S, Reverse R, Boats B
where S.sid=R.sid and R.bid=B.bid and B.color=‘red’ and B.color=‘green’
```

正确的方法:

```
Select S.sid，S.sname
from Sailors S, Reverse R1, Boats B1, Reverse R2, Boats B2
where S.sid=R1.sid and R1.bid=B1.bid and S.sid=R2.sid and R2.bid=B2.bid and B1.color=‘re
```

实验对比2个查询的结果?
第一个查询不到，因为不存在颜色即是红色也是绿色的船;
第二个可以查询到正确结果，分别查询出租用红色和绿色船的船员然后取交集。

7.intersect

```
select S.sid，S.sname
from   Sailors S, Reverse, R Boats B
where S.sid=R.sid and R.bid=B.bid and B.color=‘red’
intersect
select S2.sid，S2.sname
from Sailors S2, Reverse R2, Boats B2
where S2.sid=R2.sid and R2.bid=B2.bid and B2.color=‘green’
```

第二部分的from子句和第一部分的from子句是否可以相同?

可以