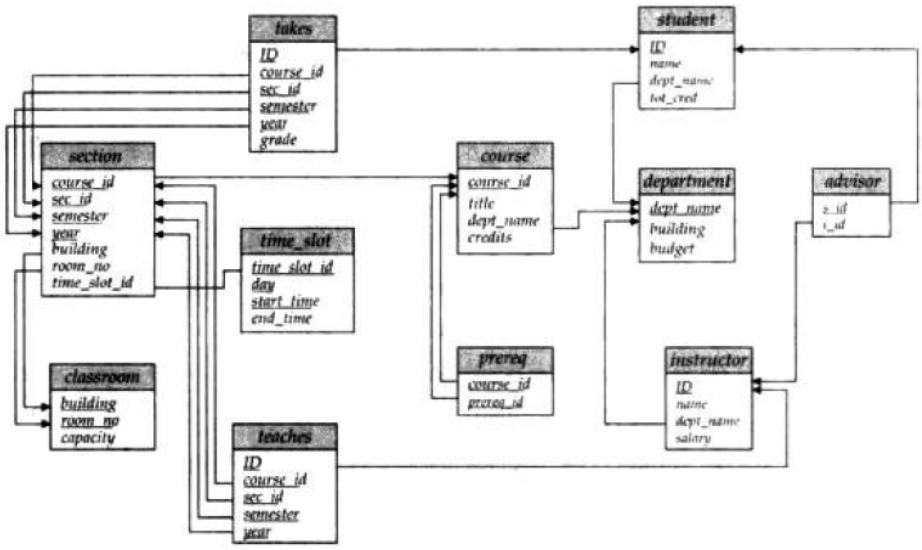

图 2-8　大学数据库的模式图

3.11　使用大学模式，用 SQL 写出如下查询。

a. 找出所有至少选修了一门 Comp. Sci. 课程的学生姓名，保证结果中没有重复的姓名。

b. 找出所有没有选修在 2009 年春季之前开设的任何课程的学生的 ID 和姓名。

c. 找出每个系教师的最高工资值。可以假设每个系至少有一位教师。

d. 从前述查询所计算出的每个系最高工资中选出最低值。

a.

```
select distinct name
from student, takes, course
where student.ID = takes.ID and takes.course_id = course.course_id
and dept_name = "Comp. Sci."
```

b.

```
select ID, name
from student
except
select ID, name
from student, takes, section
where student.ID = takes.ID and takes.course_id = section.course_id and year < 2009
```

c.

```
select max(salary), dept_name
from instructor
group by dept_name
```

d.

```
select min(max_salary)
from (select max(salary) as max_salary, dept_name
        from instructor
        group by dept_name)
```

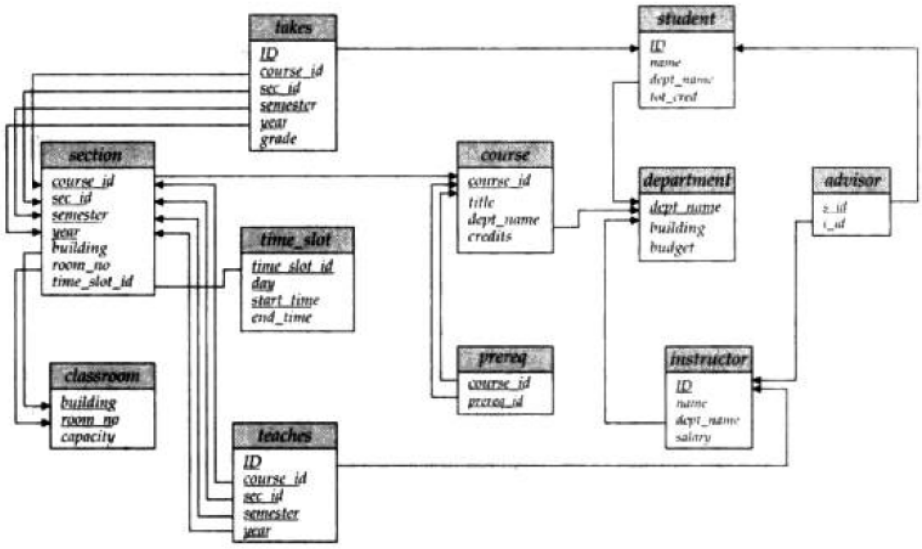

图 2-8　大学数据库的模式图

3. 12　使用大学模式，用 SQL 写出如下查询。

a. 创建一门课程“CS－001”，其名称为“Weekly Seminar”，学分为 0。

b. 创建该课程在 2009 年秋季的一个课程段，*sec_id* 为 1。

c. 让 Comp. Sci. 系的每个学生都选修上述课程段。

d. 删除名为 Chavez 的学生选修上述课程段的信息。

e. 删除课程 CS-001。如果在运行此删除语句之前，没有先删除这门课程的授课信息（课程段），会发生什么事情？

f. 删除课程名称中包含“database”的任意课程的任意课程段所对应的所有 *takes* 元组，在课程名的匹配中忽略大小写。

a.

```
insert into course
values("CS-001", "Weekly Seminar", "Comp. Sci.", 0)
```

b.

```
insert into section
values("CS-001", 1, "Autumn", 2009, null, null, null)
```

c.

```
insert into takes
select ID, "CS-001", 1, "Autumn", 2009, null
from student
where dept_name = "Comp. Sci."
```

d.

```
delete from takes
where course_id = "CS-001" and sec_id = 1 semester = and "Autumn" and year = 2009
and ID in (select ID
            from student
            where name = "Chavez")
```

e.

```
delete from takes
where course_id = "CS-001"
delete from section
where course_id = "CS-001"
delete from course
where course_id = "CS-001"
```

会发生外码错误，因为课程段有参照课程的外码。

f.

```
delete from takes
where course_id in (select course_id
                    from course
                    where lower(title) like "%database%")
```

person (*driver_id*, *name*, *address*)
car (*license*, *model*, *year*)
accident (*report_number*, *date*, *location*)
owns (*driver_id*, *license*)
participated (*report_number*, *license*, *driver_id*, *damage_amount*)

图 3-18　习题 3.4 和习题 3.14 的保险公司数据库

3.13　写出对应于图 3-18 中模式的 SQL DDL。在数据类型上做合理的假设，确保声明主码和外码。

```sql
create table person (
    driver_id varchar(20) not null,
    name varchar(20),
    address varchar(20),
    primary key(driver_id)
)
create table car (
    license varchar(20) not null,
    model varchar(20),
    year integer,
    primary key(license)
)
create table accident (
    report_number integer not null,
    date date,
    location varchar(20),
    primary key(report_number)
)
create table owns (
    driver_id varchar(20) not null,
    license varchar(20) not null,
    primary key(driver_id),
    foreign key(driver_id) references person,
    foreign key(license) references car
)
create table participated (
    report_number varchar(20) not null,
    license varchar(20) not null,
    driver_id varchar(20) not null,
    damage_amount integer,
    primary key(report_number, license)
    foreign key(report_number) references accident,
    foreign key(license) references car,
    foreign key(driver_id) references person
)
```

person (*driver_id*, *name*, *address*)
car (*license*, *model*, *year*)
accident (*report_number*, *date*, *location*)
owns (*driver_id*, *license*)
participated (*report_number*, *license*, *driver_id*, *damage_amount*)

图 3-18　习题 3.4 和习题 3.14 的保险公司数据库

3.14　考虑图 3-18 中的保险公司数据库，其中加下划线的是主码。对这个关系数据库构造如下的 SQL 查询：

a. 找出和“John Smith”的车有关的交通事故数量。

b. 对事故报告编号为“AR2197”中的车牌是“AABB2000”的车辆损坏保险费用更新到 3000 美元。

a.

```
select count(report_number)
from paricipated, owns, person
where participated.license = owns.license and owns.driver_id = person.driver_id
and name = "John Simith"
```

b.

```
update participated
set damage_amount = 3000
where report_number = "AR2197" and license = "AABB2000"
```

branch(*branch_name*, *branch_city*, *assets*)
customer (*customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*customer_name*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*customer_name*, *account_number*)

图 3-19　习题 3.8 和习题 3.15 的银行数据库

3.15　考虑图 3-19 中的银行数据库，其中加下划线的是主码。为这个关系数据库构造出如下 SQL 查询：

a. 找出在“Brooklyn”的所有支行都有账户的所有客户。
b. 找出银行的所有贷款额的总和。
c. 找出总资产至少比位于 Brooklyn 的某一家支行要多的所有支行名字。

a.

```
with branchcount as
(select count(*)
from branch
where branch_city = "Brooklyn"
)
select customer_name
from customer c1
where branchcount = (select count(distinct branch_name)
                     from branch, account, customer c2
                     where branch.branch_name = account.branch_name
                     and account.account_number = c2.account_number
                     and c1.customer_name = c2.customer_name)
```

b.

```
select sum(amount)
from loan
```

c.

```
select branch_name
from branch
where assets > some (select assets
                     from branch
                     where branch_name = "Brooklyn")
```

3.19 证明在 SQL 中，<> all 等价于 not in。

`x <>all S` 即一元素x与一子查询S中任意元素都满足不等，当且仅当x不在S中即 `x not in S`。

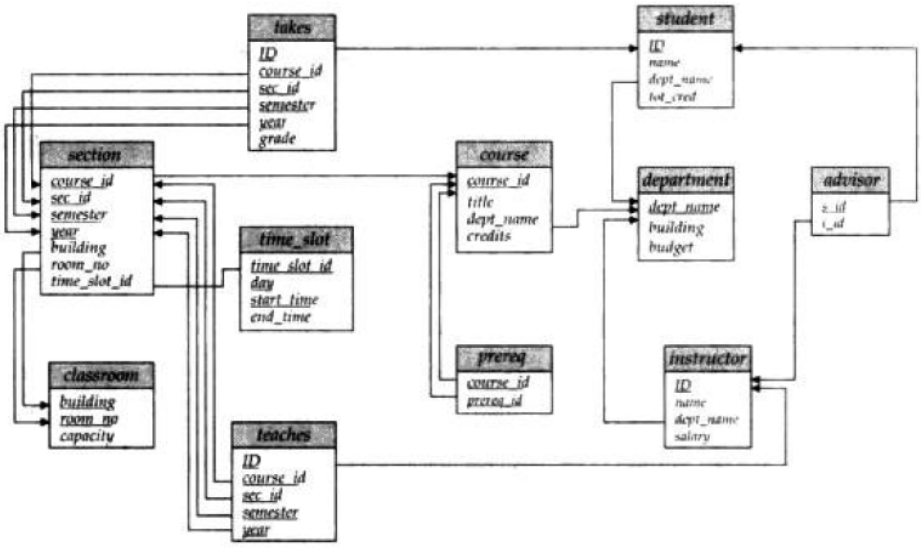

图 2-8 大学数据库的模式图

3.23 考虑查询：

```
select course_id, semester, year, sec_id, avg (tot_cred)
from takes natural join student
where year = 2009
group by course_id, semester, year, sec_id
having count (ID) >= 2;
```

解释为什么在 **from** 子句中还加上与 *section* 的连接不会改变查询结果。

`year` `corse_id` `semester` `year` `sec_id` 是 `takes` 参照 `sction`。所以每个 `takes` 可以匹配至多一个 `section`，所以每个组不会有更多元组。另外，这些属性是 `section` 的主码，不能是 `null`，因而自然连接后每个组不会丢失元组，所以结果相同。