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* **natural join** *takes*, *course*
where *takes.course_id* = *coutse.course_id* **and** *course.dept_name* = 'Comp. Sci.' ;

b. **select** *distinct ID*, *distinct name*
from *student*
where *ID* **not in** (**select** *student.ID*
from *student* **natural join** *takes*
where *year* < 2009 ;

c. **select max**(*salary*)
from *instructor*
group by *dept_name* ;

d. **select min**(*salary*)
from (**select max**(*salary*)
from *instructor*
group by *dept_name*)
as *max_salary*(*salary*) ;

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. 删除课程名称包含"database"的任意课程的任意课程段所对应 takes 元组，在课程名的配中忽略大小写。

a. **insert into** *course* (*course_id*, *title*, d*ept_name*, *credit*)
values ('CS-001', 'Weekly Seminar', 'Comp. Sci. ', 0) ;

b. **insert into** *section*(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_no*, *time_slot_id*)
values ('CS-001', 1, 'Fall', 2009, null, null, null) ;

c. **insert into** *takes*(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
select *ID*, 'CS-001', 1, 'Fall', 2009, null
from *student*
where *dept_name* = 'Comp. Sci' ;

d. **delete from** *takes*

where *course_id* = 'CS-001' **and** *ID* **in** (**select** *ID*
from *student*
where *name* = 'Chavez') ;

e. **delete from** *course*
where *course_id* = 'CS-001' ;

delete from *takes*
where *course_id* = 'CS-001' ;

delete from *section*
where *course_id* = 'CS-001' ;

可能会不让删除，可能 *takes* 中 *course_id* 为'CS-001'的每个元组都被删除，可能 *takes* 中 *course_id* 为'CS-001'的每个元组的 *course_id* 被改为 null。

f. **delete from** *takes*
where (*course_id*, *sec_id*, *semester*, *year*) **in**
(**select** *section.course_id*, *section.sec_id*, *section.semester*, *section.year*
from *course* **natural join** *section*
where lower(*course.title*) **like** '%database%') ;

3.13 写出对应图 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。

create table *person*
(*driver_id* **varchar**(10),
name **varchar**(10),
address **varchar**(20),
primary key (*driver_id*));

create table *car*
(*license* **varchar**(10),
model **varchar**(10),
year **int**,
primary key (*license*));

create table *accident*
(*report_number* **varchar(10)**,
date **varchar**(10),
location **varchar**(20).
primary key (*report_number*));

create table *owns*

(*driver_id* **varchar**(10).
license **varchar**(10)
primary key (*driver_id*),
foreign key (*driver_id*) **references** *person*,
foreign key (*license*) **references** *car*);

create table *participated*
(*report_number* **varchar(10)**,
license **varchar**(10),
driver_id **varchar**(10).
damage_amount **int**,
primary key (*report_number*, *license*)
foreign key (*report_number*) **references** *owns*,
foreign key *(license*) r**eferences** *car*,
froeign key (*driver_id*) **references** *person*);

3.14 考虑图 3-18 中的保险公司数据库，其中加下划线的主码，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的数量。
b. 对事故报告编号为“AR2197”中的车牌是“AABB2000”的车牌损坏保险费用更新到 3000 美元。

a. **select count**(*)
from *person* **natural join** p*articipated*, accident
where *participated.report_number* = *accident. report_number* **and**
person.name = ‘John Smith’ ;

b. **update** *participated*
set *damage_amount* = 3000
where *report_number* = ‘AR2197’ **and** *license* = ‘AABB2000’

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. 找出银行的所有货款额的总和。
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. **select** *account_number*
from (**select** a*ccount_number*, *branch_name*
from *branch* **natural join** *account*
where *branch_city* = ‘Brooklyn’)

group by *account_number*
having count(*branch_name*) = (**select count**(*branch_name*)
from *branch*
where *branch_city* = 'Brooklyn') ;

b. **select sum**(*amount*)
from *branch* **natural join** *loan*
group by *branch*_name

c. **select** *T.branch_name*
from *branch* **as** T
where *T.assets* > **some**(**select** *S.assets*
from *branch* **as** *S*
where *S.branch_city* = 'Brooklyn') ;

3.19 证明在 SQL 中，**<>all** 等价于 **not in**。

<>all 表示所有的某属性值都不等于，那么就是不包含的意思，所以**<>all** 等价于 **not in**。

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。

因为(*course_id*, *semester*, *yearm*, *sec_id*)是 *section* 的主码，而加上与 *section* 的连接恰好是根据这个主码来连接，所以连接后还是可以唯一确定哪个同学选了哪个课程段，不会出现多的课程段或是少了某个课程段，只是每行多了一些属性而已。