第三章作业

3.11

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 = course.course_id and course.dept_name = 'Comp. Sci.'
```

b.

```
select ST.ID, ST.name
from student natural join takes as ST
where not exists (select T.course_id
                  from takes as T
                  where T.ID = ST.ID and year < 2009)
```

c.

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

d.

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

3.12

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', null, 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 ID in (select ID
             from student
             where name = 'Chavez')
    and course_id = 'CS-001'
    and sec_id = 1
    and semester = 'Autumn'
    and year = 2009
```

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'
```

如果在删除课程前，没有先删除这门课程的授课信息（课程段），因为section中的外码course_id指向course中的主码course_id，所以才删除失败，返回错误

f.

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

3.13

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。

```
create table person
(driver_id varchar(20),
name varchar(20),
address varchar(20),
primary key (driver_id))
```

```
create table car
(license varchar(20),
model varchar(20),
year int,
primary key (license))
```

```
create table accident
(report_number varchar(20),
date datetime,
location varchar(20),
primary key (report_number))
```

```
create table owns
(driver_id varchar(20),
license varchar(20),
primary key (driver_id),
foreign key (driver_id) references person,
foreign key (license) references car)
```

```
create table participated
(report_number varchar(20),
license varchar(20),
driver_id varchar(20),
damage_amount int,
primary key (report_number, license),
foreign key (report_number) references accident,
foreign key (license) references car,
foreign key (driver_id) references person)
```

3.14

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(*)
from accidentnatural join participated join person using driver_id
where name = 'John Smith'
```

b.

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

3.15

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.

```
select distinct customer_name
from depositor as D
where not exists ((select B.branch_name
                   from branch as B
                   where B.branch_city = 'Brooklyn')
                   except
                   (select AB.branch_name
                   from account natural join branch as AB
                   where AB.account_number = D.account_number))
```

b.

```
select sum(account)
from loan
```

c.

```
select A.branch_name
from branch as A
where A.assets > some(select B.assets
                      from branch as B
                      where B.branch_city = 'Brooklyn')
```

3.19

3. 19　证明在 SQL 中，**<> all** 等价于 **not in**。

充分性：对于任何一个SQL查询结果，若元素x <>all Q，则元素x不等于Q中任何一个数据，那么元素x必然不属于Q，即 x not in Q

必要性：对于任何一个SQL查询结果Q，若元素x not in Q，则元素x不属于Q，也就不存在Q中的任意一个元素等于元素x，即 x <>all Q

综上，<>all 等价于not in

3.23

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。

因为要查询的属性takes和student都有，再连接section新增的属性没有必要，所以不会改变查询结果