Database Exercises

Homework 1

2.9 Consider the bank database of Figure 2.15

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*)

Figure 2.15

a. What are the appropriate primary keys?

branch(<u>*branch name*</u>, *branch city*, *assets*)
customer (<u>*customer name*</u>, *customer street*, *customer city*)
loan (<u>*loan number*</u>, *branch name*, *amount*)
borrower (<u>*customer name*</u>, <u>*loan number*</u>)
account (<u>*account number*</u>, *branch name*, *balance*)
depositor (<u>*customer name*</u>, <u>*account number*</u>)

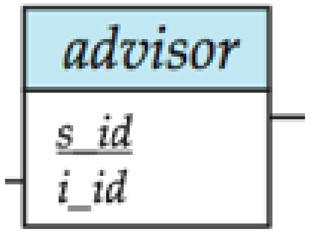
b. Given your choice of primary keys, identify appropriate foreign keys.

- For *loan*: *branch_name* referencing *branch*.
- For *borrower*: Attribute *customer_name* referencing *customer* and *loan_number* referencing *loan*
- For *account*: *branch_name* referencing *branch*.
- For *depositor*: Attribute *customer_name* referencing *customer* and *account_number* referencing *account*

Homework 1

2.10 Consider the advisor relation shown in Figure 2.8, with s_id as the primary key of advisor. Suppose a student can have more than one advisor. Then, would s_id still be a primary key of the advisor relation? If not, what should the primary key of advisor be?

No, *s_id* would not be a primary key, since there may be two (or more) tuples for a single student, corresponding to two (or more) advisors. The primary key should then be *s_id, i_id*.



Homework 1

2.12 Consider the relational database of Figure 2.14. Give an expression in the relational algebra to express each of the following queries:

a. **Find the names of all employees who work for "First Bank Corporation".**
b. **Find the names and cities of residence of all employees who work for "First Bank Corporation".**
c. **Find the names, street address, and cities of residence of all employees who work for "First Bank Corporation" and earn more than $10,000.**

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

Homework 1

2.12 Consider the relational database of Figure 2.14. Give an expression in the relational algebra to express each of the following queries:

- **a. Find the names of all employees who work for "First Bank Corporation".**
- **b. Find the names and cities of residence of all employees who work for "First Bank Corporation".**
- **c. Find the names, street address, and cities of residence of all employees who work for "First Bank Corporation" and earn more than $10,000.**

a. $\Pi_{person\_name}\,(\sigma_{company\_name\ =\ \text{“First Bank Corporation”}}\,(works))$

b. $\Pi_{person\_name,\,city}\,(employee \bowtie (\sigma_{company\_name\ =\ \text{“First Bank Corporation”}}\,(works)))$

c. $\Pi_{person\_name,\ street,\ city}\,(\sigma_{(company\_name\ =\ \text{“First Bank Corporation”}\ \wedge\ salary\ >\ 10000)}\,(works \bowtie employee))$

Homework 1

2.13 Consider the bank database of Figure 2.15. Give an expression in the relational algebra for each of the following queries:

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*)

Figure 2.15

a. Find all loan numbers with a loan value greater than $10,000.
b. Find the names of all depositors who have an account with a value greater than $6,000.
c. Find the names of all depositors who have an account with a value greater than $6,000 at the "Uptown" branch.

Homework 1

a. Find all loan numbers with a loan value greater than $10,000.
b. Find the names of all depositors who have an account with a value greater than $6,000.
c. Find the names of all depositors who have an account with a value greater than $6,000 at the "Uptown" branch.

a. $\Pi_{loan\_number}\ (\sigma_{amount>\ 10000}(loan))$

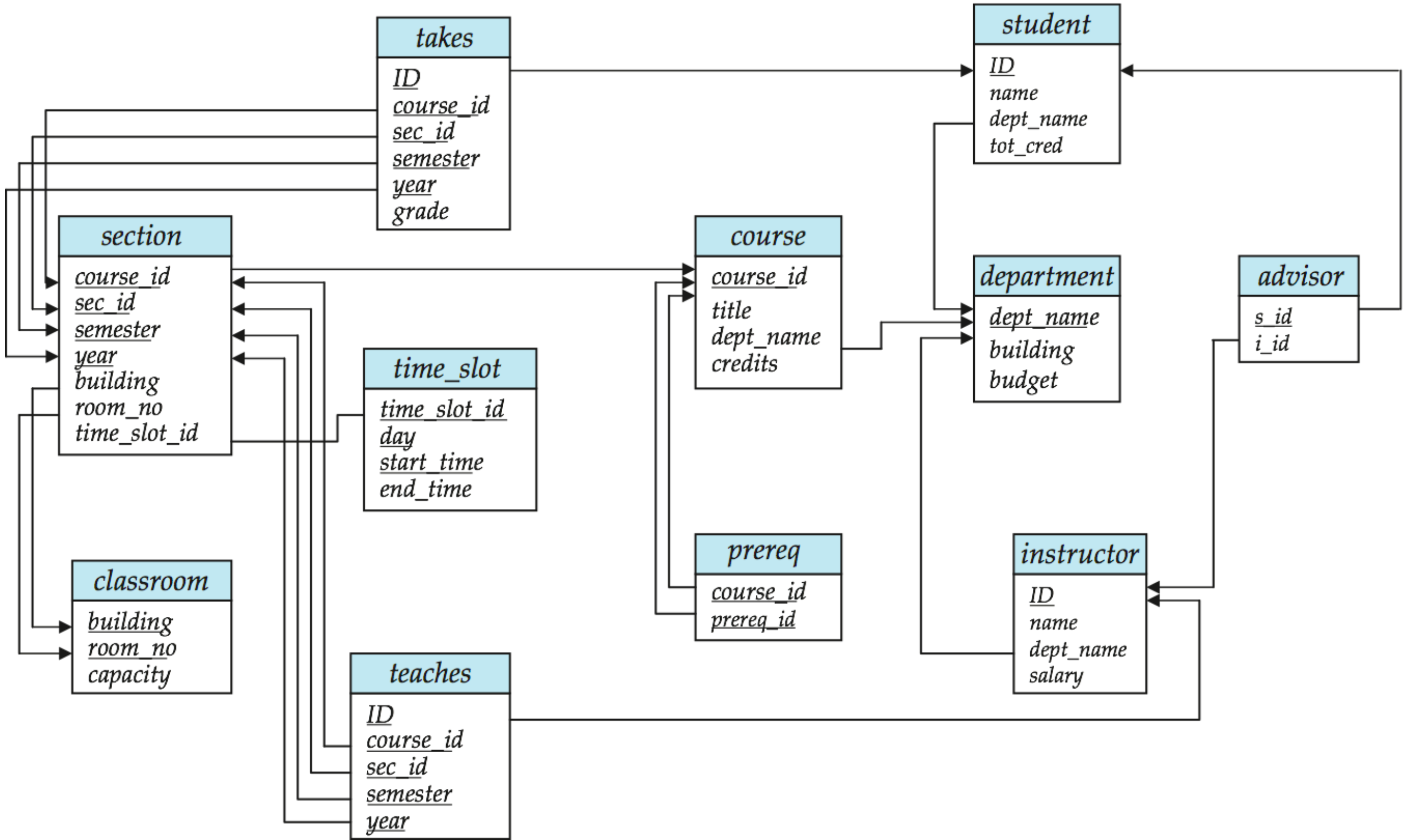
b. $\Pi_{customer\_name}\ (\sigma_{balance>\ 6000}\ (depositor \bowtie account))$

c. $\Pi_{customer\_name}\ (\sigma_{balance>\ 6000 \wedge\ branch\_name=\text{“Uptown”}}\ (depositor \bowtie account))$

Homework 2

6.10 Write the following queries in relational algebra, using the university schema.

a. Find the names of all students who have taken at least one Comp. Sci. course.
b. Find the IDs and names of all students who have not taken any course offering before Spring 2009.
c. For each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor.
d. Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query



Homework 2

a. $\Pi_{name}\ (student \bowtie takes \bowtie \Pi_{course\_id}(\sigma_{dept\_name\ =\ 'Comp.Sci.'}(course)))$
Note that if we join *student*, *takes*, and *course*, only students from the Comp. Sci. department would be present in the result; students from other departments would be eliminated even if they had taken a Comp. Sci. course since the attribute *dept_name* appears in both *student* and *course*.

b. $\Pi_{ID, name}\ (student) - \Pi_{ID, name}\ (\sigma_{year<2009}(\cancel{student \bowtie}\ takes)$ Note that Spring is the first semester of the year, so we do not need to perform a comparison on *semester*.

c. ${}_{dept\_name}\mathcal{G}_{\mathbf{max}(salary)}(instructor)$

d. $\mathcal{G}_{\mathbf{min}(maxsal)}({}_{dept\_name}\mathcal{G}_{\mathbf{max}(salary)\ \mathbf{as}\ maxsal}(instructor))$

Homework 2

6.11 Consider the relational database of Figure 6.22, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:

employee (<u>*person_name*</u>, *street*, *city*)
works (<u>*person_name*</u>, *company_name*, *salary*)
company (<u>*company_name*</u>, *city*)
manages (<u>*person_name*</u>, *manager_name*)

Figure 6.22

a. Find the names of all employees who work for "First Bank Corporation".
b. Find the names and cities of residence of all employees who work for "First Bank Corporation".
c. Find the names, street addresses, and cities of residence of all employees who work for "First Bank Corporation" and earn more than $10,000.
d. Find the names of all employees in this database who live in the same city as the company for which they work.
e. Assume the companies may be located in several cities. Find all companies located in every city in which "Small Bank Corporation" is located.

Homework 2

a. $\Pi_{person\_name}\ (\sigma_{company\_name\ =\ \text{“First Bank Corporation”}}\ (works))$

b. $\Pi_{person\_name,\ city}\ (employee\ \bowtie$
$(\sigma_{company\_name\ =\ \text{“First Bank Corporation”}}\ (works)))$

c. $\Pi_{person\_name,\ street,\ city}$
$(\sigma_{(company\_name\ =\ \text{“First Bank Corporation”}\ \wedge\ salary\ >\ 10000)}$
$works\ \bowtie\ employee)$

d. $\Pi_{person\_name}\ (employee\ \bowtie\ works\ \bowtie\ company)$

e. Note: Small Bank Corporation will be included in each answer.
$\Pi_{company\_name}\ (company\ \div$
$(\Pi_{city}\ (\sigma_{company\_name\ =\ \text{“Small Bank Corporation”}}\ (company))))$

Homework 3

3.9 Consider the employee database of Figure 3.20, where the primary keys are underlined. Give an expression in SQL for each of the following queries.

employee (<u>*employee_name*</u>, *street, city*)
works (<u>*employee_name*</u>, *company name, salary*)
company (<u>*company_name*</u>, *city*)
manages (<u>*employee_name*</u>, *manager name*)

Figure 3.20. Employee database.

a. Find the names and cities of residence of all employees who work for First Bank Corporation.

select *e.employee_name, city*
from *employee e, works w*
where *w.company_name* = 'First Bank Corporation' **and**
w.employee_name = *e.employee_name*

Homework 3

b. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

employee (<u>*employee_name*</u>, *street*, *city*)
works (<u>*employee_name*</u>, *company_name*, *salary*)
company (<u>*company_name*</u>, *city*)
manages (<u>*employee_name*</u>, *manager_name*)

```
select *
from employee
where employee_name in
    (select employee_name
    from works
    where company_name = 'First Bank Corporation' and salary > 10000)
```

```
select employee_name, streat, city
from employee natural join works
where company_name = 'First Bank Corporation' and salary > 10000
```

Homework 3

c. Find all employees in the database who do not work for First Bank Corporation.

select *employee_name*
from *works*
where *company_name* ≠ 'First Bank Corporation'

employee (<u>*employee_name*</u>, *street*, *city*)
works (<u>*employee_name*</u>, *company_name*, *salary*)
company (<u>*company_name*</u>, *city*)
manages (<u>*employee_name*</u>, *manager_name*)

If one allows people to appear in the database (e.g. in *employee*) but not appear in *works*, or if people may have jobs with more than one company, the solution is slightly more complicated.

select *employee_name*
from *employee*
where *employee_name* **not in**
  (**select** *employee_name*
  from *works*
  where *company_name* = 'First Bank Corporation')

(**select** *employee_name*
from *employee*)
except
(**select** *employee_name*
from *works*
where *company_name* = 'First Bank Corporation')

Homework 3

d. Find all employees in the database who earn more than each employee of Small Bank Corporation.

The following solution assumes that all people work for at most one company.

select *employee name*
from *works*
where *salary* > **all**
(**select** *salary*
from *works*
where *company name* = ’Small Bank Corporation’)

employee (<u>*employee_name*</u>, *street*, *city*)
works (<u>*employee_name*</u>, *company_name*, *salary*)
company (<u>*company_name*</u>, *city*)
manages (<u>*employee_name*</u>, *manager_name*)

select *employee name*
from *works*
where *salary* > (**select max**(*salary*)
from *works*
where *company name* = ’Small Bank Corporation’);

Homework 3

e. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

```
select S.company_name
from company S
where not exists ((select city
            from company
            where company_name
                    = 'Small Bank Corporation')
        except
          (select city
            from company T
            where S.company_name = T.company_name))
```

employee (<u>*employee_name*</u>, *street*, *city*)
works (<u>*employee_name*</u>, *company_name*, *salary*)
company (<u>*company_name*</u>, *city*)
manages (<u>*employee_name*</u>, *manager_name*)

Homework 3

f. Find the company that has the most employees.

employee (*employee_name*, *street*, *city*)
works (*employee_name*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*employee_name*, *manager_name*)

```
select company_name
from works
group by company_name
having count (distinct employee_name) >= all
        (select count (distinct employee_name)
        from works
        group by company_name)
```

```
with company_emp_num as
(select company_name, count(distinct employee_nmae) as num
from works
group by company_name)
select company_name
from company_emp_num
where num  =  (select max(num)
               from company_emp_num ) ;
```

Homework 3

g. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

employee (*employee_name*, *street*, *city*)
works (*employee_name*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*employee_name*, *manager_name*)

```
select company_name
from works
group by company_name
having avg (salary) > (select avg (salary)
                       from works
                       where company_name = 'First Bank Corporation')
```

Homework 4

4.6 Complete the SQL DDL definition of the university database to include the relations student, takes, advisor, and prereq.

create table *student*
    (*ID*   **varchar** (5),
    name   **varchar** (20) **not null**,
    dept_name   **varchar** (20),
    tot_cred   **numeric** (3,0) **check** (*tot_cred* >= 0),
    primary key (*ID*),
    foreign key (*dept_name*) **references** *department*
        on delete set null);

Homework 4

4.6 Complete the SQL DDL definition of the university database to include the relations student, takes, advisor, and prereq.

```
create table takes
    (ID             varchar (5),
     course_id      varchar (8),
     section_id     varchar (8),
     semester       varchar (6),
     year           numeric (4,0),
     grade          varchar (2),
     primary key (ID, course_id, section_id, semester, year),
     foreign key (course_id, section_id, semester, year) references section
                    on delete cascade,
     foreign key (ID) references student
                    on delete cascade);
```

Homework 4

4.6 Complete the SQL DDL definition of the university database to include the relations student, takes, advisor, and prereq.

```
create table advisor
    (i_id            varchar (5),
    s_id             varchar (5),
     primary key (s_ID),
     foreign key (i_ID) references instructor (ID)
                     on delete set null,
     foreign key (s_ID) references student (ID)
                     on delete cascade);
 create table prereq
    (course_id       varchar(8),
     prereq_id       varchar(8),
     primary key (course_id, prereq_id),
     foreign key (course_id) references course
                     on delete cascade,
     foreign key (prereq_id) references course);
```

Homework 4

4.8 As discussed in Section Section 4.4.7Complex Check Conditions and Assertion we expect the constraint “an instructor cannot teach sections in two different classrooms in a semester in the same time slot” to hold.

a. **Write an SQL query that returns all (*instructor*, *section*) combinations that violate this constraint.**

select ID, *name*, ~~*section_id*~~, *semester*, *year*, *time_slot_id*,
count(**distinct** *building*, *room_number*)
from *instructor* **natural join** *teaches* **natural join** *section*
group by (ID, *name*, ~~*section_id*~~, *semester*, *year*, *time_slot_id*)
having **count**(*building*, *room_number*) > 1

Homework 4

4.8 As discussed in Section Section 4.4.7Complex Check Conditions and Assertion we expect the constraint "an instructor cannot teach sections in two different classrooms in a semester in the same time slot" to hold.

b. Write an SQL assertion to enforce this constraint (as discussed in Section Section 4.4.7 Complex Check Conditions and Assertionssubsection.4.4.7, current generation database systems do not support such assertions, although they are part of the SQL standard).

create assertion check not exists
    (**select** ID, *name*, ~~*section_id*~~, *semester*, *year*, *time_slot_id*,
        count(**distinct** *building*, *room_number*)
    from *instructor* **natural join** *teaches* **natural join** *section*
    group by (ID, *name*, ~~*section_id*~~, *semester*, *year*, *time_slot_id*)
    having **count**(*building*, *room_number*) > 1)

Homework 4

4.14 Show how to define a view tot_credits (year, tot_credits), giving the total number of credits taken by students in each year.

create view *tot_credits*(*year*, *tot_credits*)
as
    (**select** *year*, **sum**(*credits*)
    from *takes* **natural join** *course*
    group by *year*)

Homework 4

4.16 Referential-integrity constraints as defined in this chapter involve exactly two relations. Consider a database that includes the relations shown in Figure 4.12. Suppose that we wish to require that every name that appears in address appears in either salaried worker or hourly worker, but not necessarily in both.

salaried_worker (*name*, *office*, *phone*, *salary*)
hourly_worker (*name*, *hourly_wage*)
address (*name*, *street*, *city*)

Figure 4.12 Employee database

a. Propose a syntax for expressing such constraints.
b. Discuss the actions that the system must take to enforce a constraint of this form.

a. **foreign key** (*name*) **references** *salaried worker* **or** *hourly worker*

b. To enforce this constraint, whenever a tuple is inserted into the *address* relation, a lookup on the *name* value must be made on the *salaried worker* relation and (if that lookup failed) on the *hourly worker* relation (or vice-versa).