**Chapter 2 연습문제 Solution (총 3 문제; 2.10, 2.12, 2.13)**

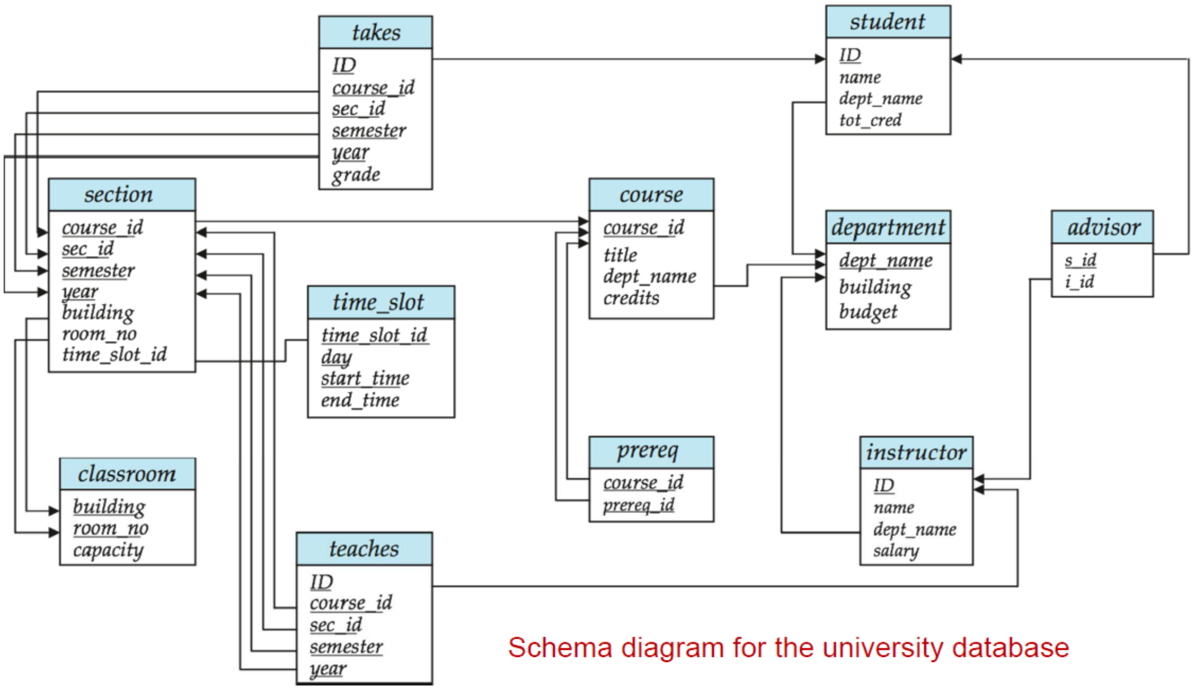
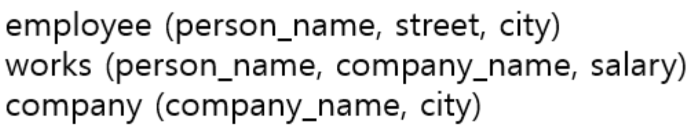


Figure 2.8. Schema diagram for the university database

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?

**Answer:** 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*.



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

Figure 2.14. Relational database for Exercise 2.12.

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 name of all employees who work for "First Bank Corporation".

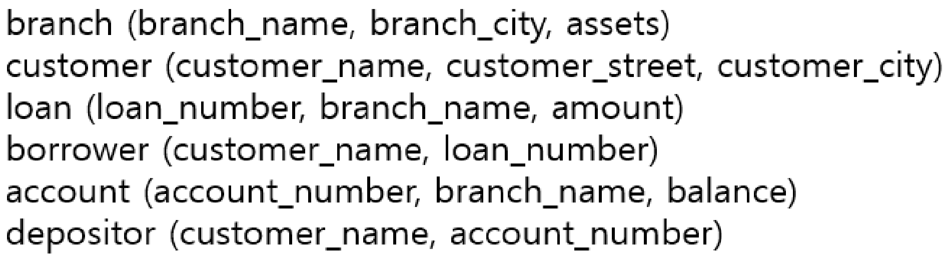
**Answer: PROJECTION** person_name (**SELECTION** company_name = “First Bank Corporation” (works))

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

**Answer: PROJECTION** person_name, city (employee **NATURAL JOIN** (**SELECTION** company_name = “First Bank Corporation” (works)))

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.

**Answer: PROJECTION** person_name, street, city (**SELECTION** (company_name = “First Bank Corporation” and salary >10000) (works **NATURAL JOIN** employee))



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. Relational database for Exercise 2.13.

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

a) Find all loan numbers with a loan value greater than $10,000.

**Answer: PROJECTION** loan_number (**SELECTION** amount > 10000(loan))

b) Find the names of all depositors who have an account with a value greater than $6,000.

**Answer: PROJECTION** customer_name (**SELECTION** balance > 6000(depositor **NATURAL JOIN** account))

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

**Answer:** **PROJECTION** customer_name(**SELECTION** balance > 6000 and branch_name = "Uptown" (depositor **NATURAL JOIN** account))

**Chapter 3 연습문제 (총 7 문제; 3.11, 3.12, 3.14, 3.16, 3.20, 3.21, 3.24)**

3.11. Write the following queries in SQL, using the university schema

a) Find the names of all students who have taken at least on Comp. Sci. course; make sure there are no duplicate names in the result.

**Answer:**

```
select name
from student natural join takes natural join course
where course.dept = 'Comp. Sci.'
```

b) Find the IDs and names of all students who have not taken any course offering before Spring 2009.

**Answer:**

```
select id, name
from student
except
select id, name
from student natural join takes
where year < 2009
```

**참고:** Since the **except** operator eliminates duplicates, there is no need to use a **select distinct** clause, although doing so would not affect correctness of the query.

c) Find each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor.

**Answer:**

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

d) Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.

**Answer:**

```
select min(maxsalary)
from (select dept, max(salary) as maxsalary
      from instructor
      group by dept)
```

3.12. Write the following queries in SQL, using the university schema

a) Create new course “CS-001”, titled “Weekly Seminar”, with 0 credits.

**Answer:**

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

b) Create a section of this course in Autumn 2009, with *sec_id* of 1.

**Answer:**

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

**참고:** Note that the building, roomnumber and slot were not specified in the question, and I has set them to null. The same effect would be obtained if they were specified to default to null, and we simply omitted values for these attributes in the above insert statement. (Many database systems implicitly set the default value to null, even if not explicitly specified.)

c) Enroll every student in the Comp. Sci. department in the above section.

**Answer:**

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

d) Delete enrollments in the above section where the student’s name is Chavez.

**Answer:**

```
delete from takes
where course id= ’CS-001’ and section id = 1 and
    year = 2009 and semester = ’Autumn’ and
    id in (select id
        from student
        where name = ’Chavez’)
```

**참고:** Note that if there is more than one student named Chavez, all such students would have their enrollments deleted. If we had used = instead of **in**, an error would have resulted if there were more than one student named Chavez.

e) Delete the course CS-001. What will happen if you run this delete statement without first deleting offerings (sections) of this course.

**Answer:**

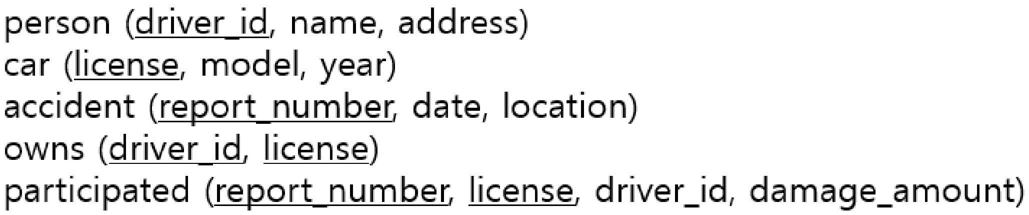
```
delete from takes
where course id = ’CS-001’
delete from section
where course id = ’CS-001’
delete from course
where course id = ’CS-001’
```

If we try to delete the course directly, there will be a foreign key violation because *section* has a foreign key reference to *course* (It is due to referential integrity constraints); similarly, we have to delete corresponding tuples from *takes* before deleting sections, since there is a foreign key reference from *takes* to *section*.

f) Delete all *takes* tuples corresponding to any section of any course with the word “database” as a part of the title; ignore case when matching the word with the title.

**Answer:**

```
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)

Figure 3.18. Insurance database for Exercise 3.14.

3.14. Consider the insurance database of Figure 3.18, where the primary keys are underlined. Construct the following SQL queries for this relational database.

a) Find the number of accidents in which the cars belonging to “John Smith” were involved.

**Answer:**

```
select count (*)
from accident
where exists
        (select *
         from participated, owns, person
         where owns.driver id = person.driver id
                and person.name = ’John Smith’
```

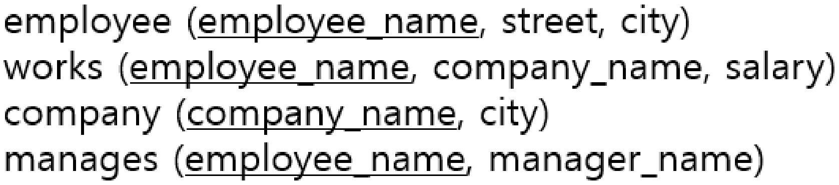
```
                    and owns.license = participated.license
                    and accident.report_number = participated.report number)
```

**참고:** The query can be written in other ways too; for example without a subquery, by using a join and selecting **count**(**distinct** *report number*) to get a count of number of accidents involving the car.

b) Update the damage amount for the car with the license number "AABB2000" in the accident with report number "AR2197" to $3000.

**Answer:**

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



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

Figure 3.20. Employee database for Exercise 3.16 and 3.20.

3.16. 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.

a) Find the names of all employees who work for "First Bank Corporation"

**Answer:**

```
    select employee_name
    from works
    where company name = 'First Bank Corporation'
```

b) Find all employees in the database who live in the same cities as the companies for which they work.

**Answer:**

```
select e.employee name
from employee e, works w, company c
where e.employee_name = w.employee_name and e.city = c.city and
      w.company_name = c.company_name
```

c) Find all employees in the database who live in the same cities and on the same streets as do their managers.

**Answer:**

```
select P.employee name
from employee P, employee R, manages M
where P.employee name = M.employee name and
     M.manager name = R.employee name and
     P.street = R.street and P.city = R.city
```

3.20. Give an SQL schema definition for the employee database of Figure 3.20. Choose an appropriate domain for each attribute and an appropriate primary key for each relation schema.

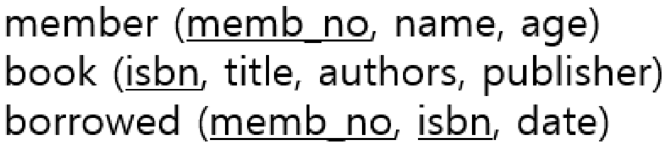
**Answer:**

```
create table employee
(employee name varchar(20),
 street char(30),
 city varchar(20),
 primary key (employee name))

create table works
(employee name person names,
 company name varchar(20),
 salary numeric(8, 2),
 primary key (employee name))
```

```
create table company
(company name varchar(20),
 city varchar(20),
 primary key (company name))

create table manages
(employee name varchar(20),
 manager name varchar(20),
 primary key (employee name))
```



member (memb_no, name, age)
book (isbn, title, authors, publisher)
borrowed (memb_no, isbn, date)

Figure 3.21. Library database for Exercise 3.21.

3.21. Consider the library database of Figure 3.21. Writhe the following queries in SQL

a) Print the names of members who have borrowed any book published by "McGraw-Hill".

**Answer:**

```
select name
from member m, book b, borrowed l
where m.memb_no = l.memb_no
and l.isbn = b.isbn and
   b.publisher = 'McGrawHill'
```

b) Print the names of members who have borrowed all books published by "McGraw-Hill".

**Answer:**

```
select distinct m.name
from member m
where not exists
    ((select isbn
    from book
    where publisher = ’McGrawHill’)
    except
    (select isbn
     from borrowed l
     where l.memb no = m.memb_no))
```

c) For each publisher, print the names of members who have borrowed more than five books of that publisher.

**Answer:**

```
    select publisher, name
    from (select publisher, name, count (isbn)
         from member m, book b, borrowed l
         where m.memb_no = l.memb_no
         and l.isbn = b.isbn
         group by publisher, name) as
         membpub(publisher, name, count_books)
    where count_books > 5
```

**참고:** The above query could alternatively be written using the **having** clause.

d) Print the average number of books borrowed per member. Take into account that if an member does not borrow any books, then that member does not appear in the *borrowed* relation at all.

**Answer:**

```
    with memcount as
        (select count(*)
         from member)
    select count(*)/memcount
    from borrowed
```

**참고:** Note that the above query ensures that members who have not borrowed any books are also counted. If we instead used **count**(**distinct** *memb_no*) from *borrowed,* we would not account for such members.

3.24. Consider the query:

```
with dept_total (dept_name, value) as
    (select dept_name, sum(salary)
    from instructor
    group by dept_name),
dept_total_avg(value) as
    (select avg(value)
    from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

Rewrite this query without using the **with** construct.

**Answer:**

There are several ways to write this query. One way is to use subqueries in the where clause, with one of the subqueries having a second level subquery in the from clause as below.

```
select distinct dept name d
from instructor i
where
    (select sum(salary)
     from instructor
     where department = d)
     >=
    (select avg(s)
     from
        (select sum(salary) as s
         from instructor
         group by department))
```

**참고:** Note that the original query did not use the *department* relation, and any department with no instructors would not appear in the query result. If we had written the above query using *department* in the outer **from** clause, a department without any instructors could appear in the result if the condition were <= instead of >=, which would not be possible in the original query.

As an alternative, the two subqueries in the where clause could be moved into the from clause, and a join condition (using >=) added.