**<201433707 이형욱 – HW3>**

**3.1 Write the following queries in SQL, using the university schema. (We suggest you actually run these queries on a database, using the sample data that we provide on the Web site of the book, db-book.com. Instructions for setting up a database ,and loading sample data ,are provided on the above Web site.)**

**a. Find the titles of courses in the Comp. Sci. department that have 3 credits.**

select title

from course

where dept name = ’Comp. Sci.’

and credits = 3

**b. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.**

This query can be answered in several different ways. One way is as follows.

select distinct student.ID

from (student join takes using(ID))

join (instructor join teaches using(ID))

using(course id, sec id, semester, year)

where instructor.name = ’Einstein’

**c. Find the highest salary of any instructor.**

select max(salary)

from instructor

**d. Find all instructors earning the highest salary (there may be more than one with the same salary).**

select ID, name

from instructor

where salary = (select max(salary) from instructor)

**e. Find the enrollment of each section that was offered inAutumn2009.**

select course id, sec id, count(ID)

from section natural join takes

where semester = ’Autumn’

and year = 2009

group by course id, sec id

Another way is to use a subquery in the select clause, as follows.

select course id, sec id,

(select count(ID)

from takes

where takes.year = section.year

and takes.semester = section.semester

and takes.course id = section.course id

and takes.section id = section.section id)

from section

where semester = ’Autumn’

and year = 2009

Note that if the result of the subquery is empty, the aggregate function count returns a value of 0.

**f. Find the maximum enrollment, across all sections, in Autumn 2009.**

select max(enrollment)

from (select count(ID) as enrollment

from section natural join takes

where semester = ’Autumn’

and year = 2009

group by course id, sec id)

As an alternative to using a nested subquery in the from clause, it is possible to use a with clause, as illustrated in the answer to the next part of this question. A subtle issue in the above query is that if no section had any enrollment, the answer would be empty, not 0. We can use the alternative using a subquery, from the previous part of this question, to ensure the count is 0 in this case.

**g. Find the sections that had the maximum enrollment in Autumn2009.**

with sec enrollment as (

select course id, sec id, count(ID) as enrollment

from section natural join takes

where semester = ’Autumn’

and year = 2009

group by course id, sec id)

select course id, sec id

from sec enrollment

where enrollment = (select max(enrollment) from sec enrollment)

It is also possible to write the query without the with clause, but the subquery to find enrollment would get repeated twice in the query.

**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)**

**3.2 Suppose you are given a relation grade points(grade,points),which provides a conversion from letter grades in the takes relation to numeric scores; for example an “A” grade could be speciﬁed to correspond to 4points,an“A−” to 3.7 points, a “B+” to 3.3 points, a “B” to 3 points, and so on. The grade points earned by a student for a course offering (section) is deﬁned as the number of credits for the course multiplied by the numeric points for the grade that the student received. Given the above relation, and our university schema, write each of the following queries in SQL. You can assume for simplicity that no takes tuple has the null value for grade.**

**a. Find the total grade-points earned by the student with ID12345,across all courses taken by the student.**

select sum(credits \* points)

from (takes natural join course) natural join grade points

whereID = ’12345’

One problem with the above query is that if the student has not taken any course, the result would not have any tuples, whereas we would expect to get 0 as the answer. One way of fixing this problem is to use the natural left outer join operation, which we study later in Chapter 4. Another way to ensure that we get 0 as the answer, is to the following query:

(select sum(credits \* points)

from (takes natural join course) natural join grade points

where ID = ’12345’)

union

(select 0

from student

where takes.ID = ’12345’ and

not exists ( select \* from takes where takes.ID = ’12345’))

As usual, specifying join conditions can be specified in the where clause instead of using the natural join operation or the join .. using operation.

**b. Find the grade-point average (GPA) for the above student, that is, the total grade-points divided by the total credits for the associated courses.**

select sum(credits \* points)/sum(credits) as GPA

from (takes natural join course) natural join grade points

where ID = ’12345’

As before, a student who has not taken any course would not appear in the above result; we can ensure that such a student appears in the result by using the modified query from the previous part of this question. However, an additional issue in this case is that the sum of credits would also be 0, resulting in a divide by zero condition. In fact, the only meaningful way of defining the GPA in this case is to define it as null. We can ensure that such a student appears in the result with a null GPA by adding the following union clause to the above query. union (select null as GPA from student where takes.ID = ’12345’ and not exists ( select \* from takes where takes.ID = ’12345’))

**c. Find the ID and the grade-point average of every student**

select ID, sum(credits \* points)/sum(credits) as GPA

from (takes natural join course) natural join grade points

group by ID

Again, to handle students who have not taken any course, we would have to add the following union clause:

union

(select ID, null as GPA

from student

where not exists ( select \* from takes where takes.ID = student.ID))

**3.3 Write the following inserts, deletes or updates in SQL, using the university schema.**

**a. Increase the salary of each instructor in the Comp. Sci. department by 10%.**

**b. Delete all courses that have never been offered (that is, do not occur in the section relation).**

**c. Insert every student whose tot cred attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.**

Answer:

1. Increase the salary of each instructor in the Comp. Sci. department by 10%.

Update instructor

set salary = salary \* 1.10

where dept name = ’Comp. Sci.’

1. Delete all courses that have never been offered (that is, do not occur in the section relation). delete from course

where course id not in

(select course id from section)

1. Insert every student whose tot cred attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.

insert into instructor

select ID, name, dept name, 10000

from student

where tot cred > 100

**3.8 Consider the bank database of Figure3.19, where the primary keys are underlined. Construct the following SQL queries for this relational database.**

**a. Find all customers of the bank who have an account but not a loan.**

**b. Find the names of all customers who live on the same street and in the same city as “Smith”.**

**c. Find the names of all branches with customers who have an account in the bank and who live in “Harrison”.**

a. Find all customers of the bank who have an account but not a loan.

(select customer name

from depositor)

except (

select customer name

from borrower)

The above selects could optionally have distinct specified, without changing the result of the query.

b. Find the names of all customers who live on the same street and in the same city as “Smith”. One way of writing the query is as follows.

select F.customer name

from customer F join customer S using(customer street, customer city)

where S.customer name = ’Smith’

The join condition could alternatively be specified in the where clause, instead of using bf join .. using.

c. Find the names of all branches with customers who have an account in the bank and who live in “Harrison”.

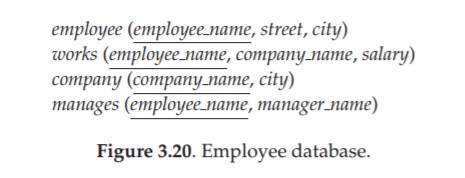
select distinct branch name

from account natural join depositor natural join customer

where customer city = ’Harrison’

As usual, the natural join operation could be replaced by specifying join conditions in the where clause.

**3.9 Consider the employee database of Figure3.20, where the primary keys are underlined. Give an expression in SQL for each of the following queries.**

****

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

**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 Exercises 3.9, 3.10, 3.16, 3.17, and 3.20.**

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

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

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

**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.**

**f. Find the company that has the most employees. g. Find those companies whose employees earn a higher salary, on average, than the average salary at “First Bank Corporation”.**

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

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. If people may work for several companies, the following solution will only list those who earn more than $10,000 per annum from “First Bank Corporation” alone.

select \*

from employee

where employee name in

(select employee name

from works

where company name = ’First Bank Corporation’ and salary > 10000)

As in the solution to the previous query, we can use a join to solve this one also.

c.Find all employees in the database who do not work for First Bank Corporation. The following solution assumes that all people work for exactly one company.

select employee name

from works

where company name 6= ’First Bank Corporation’

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’)

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’)

If people may work for several companies and we wish to consider the total earnings of each person, the problem is more complex. It can be solved by using a nested subquery, but we illustrate below how to solve it using the with clause.

with emp total salary as

(select employee name, sum(salary) as total salary

from works

group by employee name

)

select employee name

from emp total salary

where total salary > all

(select total salary

from emp total salary, works

where works.company name = ’Small Bank Corporation’ and

emp total salary.employee name = works.employee name

)

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. The simplest solution uses the contains comparison which was included in the original System R Sequel language but is not present in the subsequent SQL versions.

select T.company name

from company T

where (select R.city

from company R

where R.company name = T.company name)

contains

(select S.city

from company S

where S.company name = ’Small Bank Corporation’)

Below is a solution using standard SQL.

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))

f. Find the company that has the most employees.

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)

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

select company name

from works

group by company name

having avg (salary) > (select avg (salary)

from works

where company name = ’First Bank Corporation’)

**3.10 Consider the relational database of Figure 3.20. Give an expression in SQL for each of the following queries.**

**a. Modify the database so that “Jones” now livesin “Newtown”.**

The solution assumes that each person has only one tuple in the employee relation.

update employee

set city = ’Newton’

where person name = ’Jones’

**b. Give all managers of “First Bank Corporation” a 10 percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3 percent raise.**

b. Give all managers of First Bank Corporation a 10-percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3-percent raise.

update works T

set T.salary = T.salary \* 1.03

where T.employee name in (select manager name

from manages)

and T.salary \* 1.1 > 100000

and T.company name = ’First Bank Corporation’

update works T

set T.salary = T.salary \* 1.1

where T.employee name in (select manager name

from manages)

and T.salary \* 1.1 <= 100000

and T.company name = ’First Bank Corporation’

The above updates would give different results if executed in the opposite order. We give below a safer solution using the case statement.

update works T

set T.salary = T.salary ∗

(case

when (T.salary ∗ 1.1 > 100000) then 1.03

else 1.1

)

where T.employee name in (select manager name

from manages) and

T.company name = ’First Bank Corporation’

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

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

Select distinct name

From student natural join takes

Where course\_id in (select course\_id from course where dept\_name=’comp. sci.’;

**b. Find the ID sand names of all students who have not taken any course offering before Spring 2009.**

Select distinct id, name

From student natural join takes

Where course\_id not in(select course\_id from course where year<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.**

Select name, dept\_name,max(salary) as max\_salary

From instructor;

Group by dept\_name;

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

Select min(max\_sal)

From (select max(salary) as max\_sal from instructor group by dept\_name) as new\_table;

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

**a. Createanewcourse“CS-001”,titled“WeeklySeminar”,with0credits.**

Insert into course(course\_id,title,dept\_name,credits) values (‘CS-001’, ‘Weekly Seminar’, ‘Comp. Sci.’,0);

**b. Create a section of this course in Autumn 2009, with sec id of 1.**

Insert into section(course\_id, sec\_id, semester, year) values(‘CS-001’, 1,’Fall’, 2009);

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

Insert into takes(id, course\_id, sec\_id, semester, year)

Select id,’CS-001’,’1’,’Fall’, 2009

From student

Where dept\_name=’comp. sci.’;

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

Delete from takes

Where (course\_id=’cs-001’) and (sec\_id=’1’) and (semester=’fall’) and (year=2009) and (id is(select from student where name=’chavez’));

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

Delete from course

Where course\_id=’cs-001’

We can’t delete the course ‘cs-001’ because of the foreign key, but delete on cascade constraints let us can delete the course automatically.

**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.**

Delete from takes where course\_id in (select course\_id from course where title like ‘%database%’);

**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 3.19 Banking database for Exercises 3.8 and 3.15.**

**3.15 Consider the bank database of Figure3.19, where the primary keys are underlined. Construct the following SQL queries for this relational database.**

**a. Find all customers who have an account at all the branches located in “Brooklyn”.**

Select distinct customer\_name

From depositor natural join account natural join branch

Where branch\_city=’brooklyn’;

**b. Find out the total sum of all loan amounts in the bank.**

Select sum(amount)

From load natural join branch

Group by branch\_name;

**c. Find the names of all branches that have assets greater than those of at least one branch located in “Brooklyn”.**

Select branch\_name

From branch

Where assets>all in (select assets from branch where branch\_city=’brooklyn’) as ass

**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 Exercises 3.9, 3.10, 3.16, 3.17, and 3.20.**

**3.16 Consider the employee database of Figure3.20, 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”.**

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.**

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.**

Select e1.employee\_name

From employee e1, employee e2, manages m

Where e1.employee\_name=m.employee\_name and e2.employee\_name=m.manager\_name and e1.city=e2.city and e1.street=e2.street;

**d. Find all employees who earn more than the average salary of all employees of their company.**

select a.employee\_name

from works, (select x.company\_name, avg(x.salary) as avg\_sal from instructor x group by company\_name) as b

where a.company\_name=b.company\_name and a.salary>b.avg\_sal;

**e. Find the company that has the smallest payroll.**

Select company\_name,min(payroll)

From (select company\_name, sum(salary) payroll from works group by company\_name) as new\_table;

**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 Exercises 3.9, 3.10, 3.16, 3.17, and 3.20.**

**3.17 Consider the relational database of Figure 3.20. Give an expression in SQL for each of the following queries.**

**a. Give all employees of “First Bank Corporation” a 10 percent raise.**

Update works

Set salary=salary\*1.1

Where company\_name=’first bank corporation’;

**b. Give all managers of “First Bank Corporation” a 10 percent raise.**

Update works

Set salary=salary\*1.1

Where employee\_name in ( select manager\_name from manages) and company\_name=’first bank corporation’;

**c. Delete all tuples in the works relation for employees of “Small Bank Corporation”.**

Delete works

Where company\_name=’small bank corporation’;