INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, KALYANI

Database Management Systems

Class Test I - Solutions

31st August 2016 - 19th September 2016

Maximum Marks $15 \times 2 = 30$

This test will contribute 3% to the total marks

1. Consider the following University database. Write the following queries in SQL without using $WITH$ clause.

   - $student(\underline{ID}, name, dept\_name, tot\_credit)$: this means a student has an unique ID and this table stores the ID, his/her name, department name and total credit taken so far by him/her.
   - $department(\underline{dept\_name}, building, budget)$: this means a department has an unique name and this table stores this name along with the building name where it is situated and the budget of the department.
   - $instructor(\underline{ID}, name, dept\_name, salary)$: this table stores all information about an instructor; an instructor has an unique ID, also stored are name, his/her department name and salary.
   - $classroom(\underline{building}, \underline{room\_no}, capacity)$: the information about the classrooms are stored here; building, room_no uniquely identifies a classroom, each classroom also has a capacity.
   - $course(\underline{course\_id}, title, dept\_name, credit)$: information of a course can be found from this table; course_id uniquely identifies a course tuple, also stored are title of the course, corresponding department name and credit of the course.
   - $prereqsit(\underline{course\_id}, \underline{prereq\_id})$: this table stores the prerequisites of course_id, which are given by the set of prereq_id of different courses.
   - $advisor(\underline{s\_id}, i\_id)$: identifies the advisor of student s_id as instructor i_id.
   - $time\_slot(\underline{time\_slot\_id}, \underline{day}, \underline{start\_time}, end\_time)$: specifies that a time-slot identified by time_slot_id is on day, start_time to end_time; for example, a time-slot may be given by id A and represents Monday 9-10, Tuesday 10-11 and Wednesday 12-1.
   - $section(\underline{course\_id}, \underline{section\_id}, \underline{semester}, \underline{year}, building, room\_no, time\_slot\_id)$: specifies a section for course course_id for the given semester and year takes place in the given building and room during time-slot given by time_slot_id.
   - $takes(\underline{ID}, \underline{course\_id}, \underline{section\_id}, \underline{semester}, \underline{year}, grade)$: a row of this table identifies student with the given ID takes the given course at section given by section_id during given semester and year and has obtained the given grade.
   - $teaches(\underline{ID}, \underline{course\_id}, \underline{section\_id}, \underline{semester}, \underline{year})$: identifies instructor with given id, takes the course given by course_id for section, semester and year given.

   (a) Find the highest salary of instructors.

   ```
   select max(salary)
   from instructor
   ```

   (b) Find instructor names and course identifiers he/she teaches with column of ”name” for instructor name renamed as ”instructor_name”.

   ```
   select name as instructor_name, course_id
   from instructor natural join teaches
   ```

   (c) Find the second highest salary of instructors.

   ```
   select max(salary)
   ```

```
from instructor
where salary <> (select max(salary)
from instructor)
```

(d) Find the sections that had maximum enrollments in Autumn 2009.

```
select course_id, sec_id
from (
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)
where enrollment = (select max(enrollment) from (
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))
```

(e) Insert every student whose *tot_cred* attribute is greater than 100 as an instructor in the same department, with a salary of 100000.

```
insert into instructor
select ID, name, dept name, 100000
from student
where tot_cred > 100;
```

(f) Check that semester values are among ”Spring” and ”Autumn”.

```
check semester in ("Spring", "Autumn")
```

(g) Design a trigger that, whenever a tuple is inserted into the takes relation, updates the tuple in the student relation for the student taking the course by adding the number of credits for the course to the students total credits.

```
create trigger credits_earned after update of takes
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> F and nrow.grade is not null
and (orow.grade = F or orow.grade is null)
begin atomic
update student
set tot_cred= tot_cred+
(select credits
from course
where course.course id= nrow.course id)
where student.id = nrow.id;
end;
```

(h) Display a list of all instructors, showing their ID, name, and the number of sections that they have taught. Make sure to show the number of sections as 0 for instructors who have not taught any section. Your query should use an outer join, and should not use scalar subqueries.

```
select ID, name,
count(course_id, section_id, year,semester) as Number of sections
from instructor natural left outer join teaches
group by ID, name;
```

with scalar subquery

```
select ID, name,
(select count(*) as Number of sections
from teaches T where T.id = I.id)
from instructor I;
```

(i) Display the list of all departments, with the total number of instructors in each department, without using scalar subqueries. Make sure to correctly handle departments with no instructors.

```
select dept_name, count(ID)
from department natural left outer join instructor
group by dept_name
```

(j) Rank students based on their GPA from the schema $GPA(\underline{s\_id, course\_id}, GPA)$.

```
select ID, rank() over (order by (GPA) desc) as s rank
from GPA
order by s rank;
```

OR

```
select ID, (1 + (select count(*)
from student grades B
where B.GPA > A.GPA)) as s rank
from GPA A
order by s rank;
```

2. Consider the following bank database with primary keys underlined. Give an expression in SQL for each of the following queries without using $WITH$ caluse.

- $employee(\underline{employee\_name}, street, city)$.
- $works(\underline{employee\_name}, company\_name, salary)$.
- $company(\underline{company\_name}, city)$.
- $manages(\underline{employee\_name}, manager\_name)$.

(a) 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)
```

(b) 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))
```

(c) 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 *
(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
```

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

```
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
```

(e) Describe how to create the *manages* table with foreign key and on delete and update cascade.

```
create table manages (
employee_name varchar(30),
manager_name varchar(30),
primary key (employee_name),
foreign key employee_name references employee(employee_name) on delete cascade,
foreign key manager_name references employee(employee_name) on delete cascade
);
```

-END-