COMP 3005: Database Management Systems **(Due: Friday Oct. 15, 2021 (11:59 PM))**

Assignment #2

Instructor: Ahmed El-Roby *Name: , ID:*

Instructions: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- The accepted formats for your submission are: pdf, txt, and java. More details below.
- You can either write your solutions in the tex file (then build to pdf) or by writing your solution by hand or using your preferred editor (then convert to pdf). However, you are encouraged to write your solutions in the tex file. If you decide not to write your answer in tex, it is your responsibility to make sure you write your name and ID on the submission file.
- If you use the tex file, make sure you edit line 28 to add your name and ID. Only write your solution and do not change anything else in the tex file. If you do, you will be penalized.
- All questions in this assignment use the university schema discussed in class (available on Brightspace under Resources → University_Toy_Database), unless otherwise stated.
- For SQL questions, upload a text file with your queries in the format shown in the file "template.txt" uploaded on culearn. An example submission is in the file "sample.txt". You will be penalized if the format is incorrect or there is no text file submission.
- For programming questions, upload your .java file.
- Late submissions are allowed for 24 hours after the deadline above with a penalty of 10% of the total grade of the assignment. Submissions after more than 24 are not allowed.

Q 1: (3 points)

Consider the following DDL statements:

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

create table section
   (course_id          varchar(8),
        sec_id         varchar(8),
   semester            varchar(6)
       check (semester in ('Fall', 'Winter', 'Spring', 'Summer')),
   year                numeric(4,0)
       check (year > 1701 and year < 2100),
   building            varchar(15),
   room_number         varchar(7),
```

```
    time_slot_id        varchar(4),
    primary key (course_id, sec_id, semester, year),
    foreign key (course_id) references course
        on delete cascade,
    foreign key (building, room_number) references classroom
        on delete set null
    );
```

Now, consider the following SQL query:

```
select course_id, semester, year, sec_id, avg (tot_cred)
from takes natural join student
where year = 2017
group by course_id, semester, year, sec_id
having count (ID) >= 2;
```

Will appending **natural join** *section* in the **from** clause change the returned result? Explain why?
Adding a natural join with section would remove from the result each tuple in *takes* whose values for (*course_id*, *semester*, *year*, *sec_id*) do not appear in *section*. However, since *takes* has the constraint:

```
foreign key (course_id, semester, year, sec_id) references section
```

there cannot be a tuple in *takes* whose values for (*course_id*, *semester*, *year*, *sec_id*) do not appear in *section*.

Q 2: (2 points)

Write an SQL query using the university schema to find the names of each instructor who has never taught a course at the university. Do this using no subqueries and no set operations.

```
select name
from instructor left outer join teaches using (ID)
where course_id is null;
```

This query can also use **natural left outer join**, or using **on**.

Q 3: (2 points)

Rewrite the following query to replace the natural join with an inner join with **using** condition:

```
select *
from section natural join classroom;
```

```
select *
from section join classroom using (building, room_number);
```

Q 4: (2 points)

Consider the following relation definition:

```
create table manager(
emp_id          char(20),
manager_id      char(20),
primary key emp_id,
foreign key (manager_id) references manager(emp_id) on delete cascade)
```

The foreign key constraint means that every manager has to be an employee. Explain what is going to happen when a manager is deleted.

The manager will be deleted. All employees who this manager is a **direct or indirect** manager of will also be deleted.

Q 5: (5 points)

Using the university schema, define a view *tot_credits* (*year*, *num_credits*), giving the total number of credits taken in each year. Then, explain why insertions would not be possible into this view.

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

In this query, that will be due to the aggregate function (*sum*). Insertions will fail because there are many ways to insert values into the *credits* attribute for each tuple given a sum of all these values.

Q 6: (10 points)

Write a Java program that finds all prerequisites for a given course using JDBC. The program should:

- Takes a course id value as input using keyboard.
- Finds the prerequisites of this course through a SQL query.
- For each course returned, repeats the previous step until no new prerequisites can be found.
- Prints the results.

Don't forget to handle the case for cyclic prerequisites. For example, if course A is prerequisite to course B, course B is prerequisite to course C, and course C is prerequisite to course A, do not infinite loop.

```
import java.sql.*;
import java.util.Scanner;
import java.util.Arrays;
public class AllCoursePrereqs {
    public static void main(String[] args) {
        try (
        Connection con = DriverManager.getConnection(
            "jdbc:postgresql://localhost:5432/univ_db","userid","passwd");
        Statement s = con.createStatement();
        ){
            String q;
            String c;
            ResultSet result;
            int maxCourse = 0;
            q = "select count(*) as C from course";
            result = s.executeQuery(q);
            if (!result.next())
                System.out.println("Unexpected empty result.");
            else
                maxCourse = Integer.parseInt(result.getString("C"));
            int numCourse = 0, oldNumCourse = -1;
            String[] prereqs = new String [maxCourse];
            Scanner krb = new Scanner(System.in);
            System.out.print("Input a course id (number): ");
            String course = krb.next();
            String courseString = "" + '\'' + course + '\'';
            while (numCourse != oldNumCourse) {
                for (int i = oldNumCourse + 1; i < numCourse; i++) {
                    courseString += ", " + '\'' + prereqs[i] + '\'' ;
                }
                oldNumCourse = numCourse;
                q = "select prereq_id from prereq where course_id in ("
```

```
                + courseString + ")";
                result = s.executeQuery(q);
                while (result.next()) {
                    c = result.getString("prereq_id");
                    boolean found = false;
                    for (int i = 0; i < numCourse; i++)
                        found |= prereqs[i].equals(c);
                    if (!found) prereqs[numCourse++] = c;
                }
                courseString = "" + '\'' + prereqs[oldNumCourse] + '\'';
            }
            Arrays.sort(prereqs, 0, numCourse);
            System.out.print("The courses that must be taken prior to "
            + course + " are: ");
            for (int i = 0; i < numCourse; i++)
                System.out.print ((i==0?" ":", ") + prereqs[i]);
            System.out.println();
        } catch(Exception e){e.printStackTrace();
    }
}
```

Q 7: (5 points)

Consider the following schema:
employee(emp_name, street, city)
works(emp_name, company_name, salary)
Write a function *avg_sal* that takes a company name as input and finds the average salary of employees in the company. Then, write a SQL query that uses this function to find companies whose employees earn (on average) higher salary than the company "Losers Inc.".

```
create function avg_sal(cname varchar(15))
returns numeric(10, 2)
declare result numeric(10,2);
    select avg(salary) into result
    from works
    where works.company_name = cname
return result;
end

select company_name
from works
where avg_sal(company_name) > avg_sal('Losers Inc.')
```