CS143: Homework #3 (Advanced SQL)

In this homework, you will practice advanced SQL queries using a sample database. Your answers to this part of the homework must be submitted as a set of sql script files to GradeScope.

Database Schema

You are given the following set of tables that store university-related information:

- Classroom(building, room_number, capacity)
- Department(dept, building, budget)
- Class(id, title, dept, credits)
- Instructor(id, name, dept, salary)
- Section(class_id, sec_id, semester, year, building, room_number)
- Teaches(inst_id, class_id, sec_id, semester, year)
- Student(id, name, dept, tot_cred)
- Takes(stud_id, class_id, sec_id, semester, year, grade)
- Advisor(stud_id, inst_id)
- Prereq(class_id, prereq_id)

Hopefully, the meanings of each table and their attributes are clear from their names. The keys of the tables are underlined.

Sample Database Load Script

To help you write queries for this homework, we provide a sample MySQL script, hw3-load.sql, that creates tables according to the above schema and populates them with sample tuples. You can download the script and create the sample database by executing the following sequence of commands inside the `mysql-apache` container:

```
$ cd ~/shared
$ wget http://oak.cs.ucla.edu/classes/cs143/homework3/hw3-load.sql
$ mysql class_db < hw3-load.sql
```

Your Task

Your task is to write SQL queries that correspond to the statements given below. In writing your queries, you may assume that

- Every department offers at least one class.
- Every department has at least one instructor.
- Every department has at least one student.

Please make sure that your queries **do not return any duplicates,** but try to avoid using `DISTINCT` when it is not necessary

Query1: Find the instructors' average salary.

Query2: For each department, find the max credit among all classes offered by the department. Your answer should consist of tuples with two columns (dept, maximum_course_credit).

Query3: Return the department names that offer at least three classes.

Query4: Find the names of departments that offer only 3-credit classes.

Query5: Return the average credit of the courses that are offered by the 'Comp. Sci.' department.

Query6: As above, but display the average course credit per *every* department. Your answer should consist of two columns (dept, dept_average_course_credit) and every department should appear once in your result.

Query7: For every department, display the average course credit of the department **and** the *overall average* of the course credits (i.e., the average credit over *all* courses, not within a department). Your answer should consist of three columns (dept, dept_avg_course_credit, overall_avg_course_credit). In your answer, return one tuple per each department.

Remark: The overall course-credit average is 3.3846 in our dataset, so something is wrong with your query if you get a different number in the third column of your result.

Query8: Compute the average salary of all instructors and compare it against the average instructor salary within each department. Your answer must consist of two columns (dept, diff_avg_salary), where dept is a department name and diff_avg_salary is avg(all instructor salaries) - avg(the department's instructors' salaries). In your answer, return one tuple per each department.

Query9: 'Comp. Sci.' and 'Elec. Eng.' departments belong to 'Engineering' school and all other departments belong to 'L&S' (letter and science) school. For every department, return the school they belong to. Your answer must consist of two columns (dept, school).

Remark: There are many ways to write this query, but using the `CASE` function is likely to lead to the most succinct query.

Query10: For each school (i.e., Engineering or L&S), show the number of students and the number of instructors in the school. Your answer must consist of three columns (school, num_studs, num_insts).

Query11: Find the id's of the students who took more classes in 2009 than in 2010 (in terms of number of classes, not credits).

Remark: Note that some students took classes in 2009 but not in 2010. These students must be included in your answer. Using `OUTER JOIN` may lead to a succinct query.

Query12: For every student, return the year in which they obtained the maximum number of course credits. Your answer must consist of two columns, (stud_id, max_credit_year). If a student, say id 12345, obtained the maximum number of course credits in multiple years, say in 2009 and 2010, your answer must return one tuple per each year, like (12345, 2009) and (12345, 2010). If a student did not take any class at all, do not return the student.

Remark: There are many ways to write this query, but using *common table expression* may lead to a succinct query.

Query13: Return the id's of the top-4 students who have obtained the largest number of course credits so far. In case of a tie, you can break the tie arbitrarily (i.e., you may return any student(s) that tied).

Remark: MySQL does not support the SQL standard `FETCH FIRST` clause. You may have to use the `LIMIT` clause that is supported by MySQL for this query.

Query14: For every student, show their name and their advisor's name. Your answer must consist of two columns (student_name, advisor_name). If a student does not have any advisor, return NULL as the advisor's name.

Remark: There are many ways to write this query, but using `OUTER JOIN` may lead to a succinct query.

Query15: Some of you may have noticed that the tot_cred value for students do not match the credits from the courses they have taken. For every student, show this discrepency. Your answer must consist of two columns, (stud_id, credit_discrepency), where credit_discrepency is tot_cred - sum(credits of the courses taken by the student).

Remark: Note that some students may not have taken any classes. Your answer must include those students as well. You may find the `COALESCE()` function helpful in writing the query.

Query16: Return all class id's that a student must take before taking 'BIO-399', including its prerequisites and the prerequisites of its prerequisites, etc. Do not include 'BIO-399' in your answer.

Remark: You will need to write a recursive query to compute the answer since we do not know the length of BIO-399's prerequisite chain in advance.

CS143: Homework #3 (Advanced SQL)

1.
```
SELECT Avg(salary)
FROM   instructor;
```

2.
```
SELECT dept,
       Max(credits)
FROM   class C
GROUP  BY dept
```

3.
```
SELECT dept
FROM   class C
GROUP  BY dept
HAVING Count(*) >= 3;
```

4.
```
(SELECT dept
 FROM   department)
EXCEPT
(SELECT dept
 FROM   class
 WHERE  credits <> 3);
```

5.
```
SELECT Avg(credits)
FROM   class
WHERE  dept = 'Comp. Sci.';
```

6.
```
SELECT dept,
       Avg(credits)
FROM   class
GROUP  BY dept;
```

7.
```
SELECT DISTINCT dept,
                Avg(credits)
                  OVER(
                    partition BY dept),
                Avg(credits)
                  OVER()
FROM   class;
```

8.
```
SELECT DISTINCT dept,
                Avg(salary)
                  OVER() - Avg(salary)
                             OVER(
                               partition BY dept)
FROM   instructor;
```

9.
```
SELECT dept,
       CASE
         WHEN ( dept = 'Comp. Sci.'
                OR dept = 'Elec. Eng.' ) THEN 'Engineering'
         ELSE 'L&S'
       end school
FROM   department;
```

10.
```
WITH school
     AS (SELECT dept,
                CASE
                  WHEN ( dept = 'Comp. Sci.'
                         OR dept = 'Elec. Eng.' ) THEN 'Engineering'
                  ELSE 'L&S'
                END school
         FROM   department)
SELECT school,
       Count(DISTINCT S.id),
       Count(DISTINCT I.id)
FROM   school,
       student S,
       instructor I
WHERE  S.dept = school.dept
       AND S.dept = I.dept
GROUP  BY school;
```

11.
```
SELECT T9.stud_id
FROM   (SELECT stud_id,
               Count(*) count
        FROM   takes
        WHERE  year = 2009
        GROUP  BY stud_id) T9
       LEFT OUTER JOIN (SELECT stud_id,
                               Count(*) count
                        FROM   takes
                        WHERE  year = 2010
                        GROUP  BY stud_id) T10
                    ON T9.stud_id = T10.stud_id
WHERE  T9.count > T10.count
        OR T10.stud_id IS NULL;
```

12.
```
WITH credits
     AS (SELECT stud_id,
                year,
                Sum(credits) credits
         FROM   takes,
                class
         WHERE  class_id = id
         GROUP  BY stud_id,
                   year)
SELECT C.stud_id,
       C.year
FROM   credits C,
       (SELECT stud_id,
               Max(credits) credits
        FROM   credits
        GROUP  BY stud_id) MC
```

```
WHERE C.stud_id = MC.stud_id
      AND C.credits = MC.credits;
```

13.
```
SELECT stud_id
FROM   takes T,
       class C
WHERE T.class_id = C.id
GROUP BY stud_id
ORDER BY Sum(credits) DESC
LIMIT 4;
```

14.
```
SELECT S.name,
       I.name
FROM   (student S
        LEFT JOIN advisor A
              ON S.id = A.stud_id)
       LEFT JOIN instructor I
             ON A.inst_id = I.id;
```

15.
```
SELECT S.id,
       S.tot_cred - Coalesce(Sum(credits), 0)
FROM   (student S
        LEFT JOIN takes T
              ON S.id = T.stud_id
       )
       LEFT JOIN class C
             ON T.class_id = C.id
GROUP BY S.id;
```

16.
```
WITH recursive Pres AS (
    (SELECT prereq_id FROM Prereq WHERE class_id='BIO-399')
    UNION
    (SELECT P2.prereq_id FROM Pres P1, Prereq P2 WHERE P1.prereq_id = P2.class_id)
)
SELECT DISTINCT *
FROM Pres;
```