**CS Databases**

**Problem Set #1**

Problem 1. In this problem, you will design a relational schema for an online platform that connects music instructors to students who want to learn an instrument. Instructors can sign up with the platform by providing a name, and a short description of their background and qualifications. In addition, each instructor provides a list of instruments they are able to teach, their typical price per lesson (which may depend on the instrument and the length of a lesson, e.g., $45 for 45 minutes of guitar, $55 for 60 minutes of guitar, but $50 for 45 minutes of piano), and the neighborhoods to which they are willing to come to give a lesson. Students sign up with a name, the neighborhood where they live, the instrument(s) they are interested in learning, and a description of their skill level on each instrument. All lessons take place in the students’ homes. It is also assumed that students and instructors in the system can exchange messages, but these messages are not modeled in our relational schema. When an instructor and student agree on a lesson to be held at a particular time and date, for a certain length of time on a certain instrument at a certain price, the instructor will input the information about the lesson into the system. The system will then ask the student to confirm the particular lesson. After the lesson is completed, the student is asked to approve payment for the lesson, which consists of the agreed-upon price plus a certain fee (say, 10%) for the online platform. A student may also leave a rating from 1 to 5 stars and a short review for a lesson. For simplicity we assume that the information is stored for each lesson separately, even if a student and instructor have weekly lessons that are always at the same time and on the same day.

1. Design a relational database schema for this online application that supports all the functionality. You should state reasonable assumptions about your schema that you find necessary. You need to specify the primary keys and foreign keys for each table.

* Instructor has:
  + **Instructor\_ID** (Primary Key)
  + I\_First\_Name (Not Null)
  + I\_Last\_Name (Not Null)
* Teaching\_Instrument:
  + **Instrument\_ID** (Primary Key)
    - Not used for any queries. Only serves as a primary key.
  + Instrument\_Name (Not Null)
  + *Instructor\_ID* (Foreign Key references Instructor)
* Students have:
  + **Student\_ID** (Primary Key)
  + S\_First\_Name (Not Null)
  + S\_Last\_Name (Not Null)
  + Neighborhood they’re in (Not Null)
    - Assumes that a student will be in one neighborhood only.
* Learning\_Instrument:
  + **Instrument\_ID** (Primary Key)
    - Not used for any queries. Only serves as a primary key.
  + Instrument (Not Null)
  + Skill\_Level
  + *Student\_ID* (Foreign Key references Student)
* Location:
  + **Location\_ID** (Primary Key)
    - Not used for any queries. Only serves as a primary key.
  + *Instructor\_ID* (Foreign Key references Instructor)
  + Location (Not Null)
* Lessons table:
  + **Lesson\_ID** (Primary Key)
  + *Instructor\_ID* (Foreign Key references Instructor)
  + *Student\_ID* (Foreign Key references Student)
  + Instrument (Not Null)
  + Proposed\_Price (Not Null)
  + Paid – Boolean
  + Date (Not Null)
  + Year (Not Null)
  + Time (Not Null)
  + Duration (in minutes)
  + Star\_Rating
  + Review\_Rating
* In the diagram below, any row marked in bold is a primary key whereas any row underlined is a foreign key.

Table

Description automatically generated with medium confidence

1. Write SQL statements for the following queries. If your schema does not allow you to answer these queries, you should go back to (a) and revise your schema.
2. Output the names of all instructors who are willing to teach piano in the Park Slope neighborhood for less than $50 for 60 minutes.

**SELECT distinct** I\_Last\_Name, I\_First\_Name  
**FROM** instructor, teaching\_instrument, lesson, location   
**WHERE** Location = 'Park Slope' **and** proposed\_price < 50 **and** Duration = 60  
**and** Instructor.Instructor\_ID = Lesson.instructor\_Id   
**and** Instructor.Instructor\_ID = Teaching\_Instrument.Instructor\_ID  
**and** teaching\_instrument.instrument = "Piano"

1. Find the students who want to learn an instrument for which there are no instructors in their neighborhood. The result should contain the name and neighborhood of the student along with the instrument.

**WITH** M as (SELECT student.student\_id  
 **FROM** Student, learning\_instrument, Instructor join Location, teaching\_instrument  
 **WHERE** learning\_instrument.Instrument = teaching\_instrument.Instrument  
 **and** teaching\_instrument.instructor\_id = instructor.instructor\_id  
 **and** learning\_instrument.student\_id = student.student\_id  
 **and** Student.Neighborhood = Location.Location  
 **and** Instructor.Instructor\_ID = Location.Instructor\_ID)  
**SELECT** **distinct** S\_LAST\_NAME, S\_FIRST\_NAME, Neighborhood, learning\_instrument.instrument  
**FROM** Student, learning\_instrument  
**WHERE** learning\_instrument.student\_id = student.student\_id  
**not in** (**Select** \*  
 **FROM** M  
 **WHERE** M.Student\_ID = Student.Student\_ID);

1. For each neighborhood and each instrument, output the average price of a 45-minute lesson for the instrument offered in the neighborhood.

**SELECT** **avg**(Proposed\_price), student.neighborhood   
**FROM** Lesson, Student, Instructor, learning\_instrument, teaching\_instrument, location  
**WHERE** Duration = 45  
**and** Lesson.Instructor\_ID = Instructor.Instructor\_ID  
**and** Lesson.Student\_ID = Student.Student\_ID  
**and** teaching\_instrument.instructor\_id = instructor.instructor\_id  
**and** learning\_instrument.student\_id = student.student\_id  
**and** location.instructor\_id = instructor.instructor\_id;  
**group by** student.neighborhood

1. For each instructor, output their name and the average rating of all the lessons they taught during 2020 (for those lessons that received a rating).

**SELECT** I\_LAST\_NAME, I\_FIRST\_NAME, **AVG**(star\_rating)  
**FROM** Instructor,Lesson  
**WHERE** Lesson.Instructor\_ID = Instructor.Instructor\_ID  
**AND** Year = 2020  
**GROUP BY** instructor.instructor\_id;

1. Output the names of all students who have refused payment for two or more lessons that they confirmed.

**SELECT** S\_LAST\_NAME, S\_FIRST\_NAME  
**FROM** STUDENT, Lesson   
**WHERE** student.student\_id = lesson.student\_id  
**and** paid = false  
**GROUP BY** (Student.student\_ID)  
**HAVING** count(Student.student\_ID) >= 2;

Problem 2. Suppose you have a database modeling a course selection platform, which is given by the following schema:

Student(sid, sname, year, birthday, mid)  
 mid references Major(mid)  
 year: freshman, sophomore, junior, senior, graduate

Department(deptid, deptname, location, school)

Major(mid, majorname, deptid)  
 deptid references Department(deptid)

Faculty(fid, fname, deptid, office\_location)

deptid references Department(deptid)

Course(coursenumber, coursename, credits, coursedescripton)

Class(classid, coursenumber, semester, year, fid, time, classroom)  
 fid references Faculty(fid)

coursenumber references Course(coursenumber)

Enrolled(sid, classid, grade)  
 sid references Student(sid)   
 classid references Class(classid)

This schema is a simplified course selection system. Students can select a major from

any department depending on their preference. (A student can select only one major.) A

course can be taught by different professors in the same semester and year with

different classids. A professor can also teach the same course several times in different

semesters, or in the same semester.

(a) Write SQL statements for the following queries:

(i) Output the id, name and major of the graduate students who have already enrolled in more than 30 credits in total.

**SELECT** student.sid, sname, majorname  
**FROM** student, major, enrolled, course join class  
**WHERE** student.mid = major.mid  
**and** enrolled.sid = student.sid  
**and** class.classid = enrolled.classid  
**and** course.coursenumber = class.coursenumber  
**group by** student.sid  
**Having** (**sum**(credits) > 30)

(ii) Get the name and id of students who have never selected any course outside the department they are majoring in the semester Fall 2021 and Spring 2022.

This query was written based off the instructor answer to a question in Piazza: *Every student has selected a major(constant throughout the semesters) which belongs to a specific department. For the two specific semesters, list all students who have selected a course outside the department.*

As such, this question was written with the assumption that we’re supposed search within the two semesters for students that took courses outside their semester.

**select** sname, sid  
**from** student, major, department, course  
**where** student.mid = major.mid  
**and** major.deptid = department.deptid  
**and** sid not in  
(**select** sid  
**from** student, major, department, course, class  
**where** student.mid = major.mid  
**and** major.deptid = department.deptid  
**and** course.deptid = department.deptid  
**and** class.coursenumber = course.coursenumber  
**and** semester = "Fall"  
**and** class.year = 2021  
)  
**and** sid **not in**   
(**select** sid  
**from** student, major, department, course, class  
**where** student.mid = major.mid  
**and** major.deptid = department.deptid  
**and** course.deptid = department.deptid  
**and** class.coursenumber = course.coursenumber  
**and** semester = "Spring"  
**and** class.year = 2022  
)  
**GROUP BY** sname

(iii) Output the name and department of professors who have never taught class after 2 PM.

**SELECT** **distinct** fname, deptname  
**FROM** Faculty, Department, Class  
**WHERE** faculty.fid = class.fid  
**and** department.deptid = faculty.deptid  
**and** faculty.fid **not in** (**select** faculty.fid

**from** Faculty, Department, Class  
 **where** faculty.fid = class.fid  
 **and** department.deptid = faculty.deptid  
 **and** class.time <= '14:00:00');

(iv) Output the name of the courses where no students have “F” grade in Spring 2021 but at least one student with an “F” grade in Spring 2022

**With** M **as** (**Select** course.coursenumber  
 **FROM** Class, enrolled, student, course  
 **WHERE** grade = "F"  
 **and** course.coursenumber = class.coursenumber  
 **and** class.classid = enrolled.classid  
 **and** student.sid = enrolled.sid  
 **and** semester = "Spring"  
 **and** class.year = "2022")

**SELECT** course.coursename  
**FROM** Course, enrolled, class **inner join** M  
**WHERE** class.year = 2021  
**and** Semester = "Spring"  
**and** enrolled.classid = class.classid  
**and** class.coursenumber = Course.coursenumber   
**and not exists** (Select course.coursename  
 **FROM** Class, enrolled, student, course  
 **WHERE** grade = "F"  
 **and** course.coursenumber = class.coursenumber  
 **and** class.classid = enrolled.classid  
 **and** student.sid = enrolled.sid  
 **and** semester = "Spring"  
 **and** class.year = "2021")  
**and** M.coursenumber = course.coursenumber  
**GROUP BY** course.coursename;

1. Find the students who enrolled in the most classes in Spring 2021, and output the students’ id and names together with the all classes they took in Spring 2021.

**SELECT** student.sid, sname, coursename  
**FROM** student, class, enrolled, course  
**WHERE** semester = "SPRING"  
**and** class.year = 2021  
**and** enrolled.classid = class.classid  
**and** enrolled.sid = student.sid  
**and** class.coursenumber = course.coursenumber  
**GROUP BY** student.sid  
**HAVING** **count**(\*) = (**SELECT** count(enrolled.sid)  
**FROM** student, class, enrolled, course  
**WHERE** semester = "SPRING"  
**and** class.year = 2021  
**and** enrolled.classid = class.classid  
**and** enrolled.sid = student.sid  
and class.coursenumber = course.coursenumber  
**GROUP BY** enrolled.sid  
**ORDER BY** enrolled.sid **ASC**  
**LIMIT** 1);

1. Output the ID of any student who has taken at least one class with every faculty member in the department of CS.

select student.sid  
from student, enrolled, class, course, department  
where student.sid = enrolled.sid  
and enrolled.classid = class.classid  
and course.coursenumber = class.coursenumber  
and course.deptid = department.deptid  
and deptname = 'Computer Science and Engineering'  
group by student.sid  
having count(distinct class.fid) = (select count(distinct faculty.fid)

from faculty, department

where department.deptid = faculty.deptid

and deptname = 'Computer Science and Engineering');

1. Output pairs of students (student1, student2) that took exactly the same courses in the Spring 2020 semester.

**With** s1 **as** (select student.sid, course.coursenumber  
 FROM student, enrolled, class, course  
 where class.year = 2020  
 and student.sid = enrolled.sid  
 and course.coursenumber = class.coursenumber  
 and semester = "spring"  
 and enrolled.classid = class.classid),  
 s2 as (select student.sid, course.coursenumber  
 FROM student, enrolled, class, course  
 where class.year = 2020  
 and student.sid = enrolled.sid  
 and course.coursenumber = class.coursenumber  
 and semester = "spring"  
 and enrolled.classid = class.classid)

select s1.sid, s2.sid  
from s1, s2  
where s1.sid<>s2.sid  
and s1.coursenumber = s2.coursenumber  
and (s1.sid, s2.sid) not in (select s1.sid, s2.sid from s1, s2  
 where s1.coursenumber <> s2.coursenumber  
 and s1.sid <>s2.sid)  
Order by s1.sid;

1. Write statements in Relational Algebra for all queries. Use basic RA whenever possible, and extended RA otherwise.

(i) Output the id, name and major of the graduate students who have already enrolled in more than 30 credits in total.

πsid, sname, major ( sum(credits) > 30 (Course ⋈ course.deptid = major.deptid (student ⋈ student.mid=major.mid major)))

(ii) Get the name and id of students who have never selected any course outside the department they are majoring in the semester Fall 2021 and Spring 2022.

πsname, sid (student)  
- πsname,sid (σsemester=”Fall” ∧class.year=2021 ⋈ (course ⋈course.deptid≠major.deptid (Major⋈major.mid=student.mid (student ⋈ student.sid = enrolled.sid enrolled)))  
- πsname,sid (σsemester=”Spring” ∧class.year=2022 ⋈ (course ⋈course.deptid≠major.deptid(Major⋈major.mid=student.mid (student ⋈ student.sid = enrolled.sid enrolled)))

(iii) Output the name and department of professors who have never taught class after 2 PM.

πfname, deptname (Class ⋈ faculty.fid = class.fid (Faculty ⋈ faculty.fid = department.fid(Department)) -  
πfname, deptname (σtime>14:00:00 (Class ⋈ faculty.fid = class.fid(Faculty ⋈ faculty.fid=Department.fid Department)))

(iv) Output the name of the courses where no students have “F” grade in Spring 2021 but at least one student with an “F” grade in Spring 2022

πcoursename(Course ⋈course.coursenumber=class.coursenumber (σYear=2022 ∧ semester=”Spring” (Class ⋈ class.classid=enrolled.classid (σgrade=”F” (Student ⋈student.sid=enrolled.sid Enrolled)))))  
- πcoursename(Course ⋈ course.coursenumber=class.coursenumber (σYear=2021 ∧ semester=”Spring” (Class ⋈class.classid=enrolled.classid (σgrade=”F” (Student ⋈ student.sid=enrolled.sid Enrolled)))))

(v) Find the students who enrolled in the most classes in Spring 2021, and output the students’ id and names together with the all classes they took in Spring 2021.

S1<- (sid, classid) count­(classid) **as** count\_class (σsemester=”Spring”∧year=”2021” (Class ⋈class.classid=enrolled.classid (Student ⋈ student.sid=enrolled.sid enrolled)))

S2<-(classid)max(classid) **as** max\_class(S1)

(Πsid,sname(σsemester=max\_class=count\_class ∧student.sid=s1.sid (S1 x S2 x student)))

(vii) Output the ID of any student who has taken at least one class with every faculty member in the department of CS.

πsid(  
Department⋈department.deptid=faculty.deptid (Faculty ⋈faculty.fid=class.fid(Class ⋈ class.classid = enrolled.classid (Student ⋈student.sid=enrolled.sid Enrolled)))  
/(σdeptname=”CS” (Department⋈department.deptid=faculty.deptid (Faculty ⋈faculty.fid = class.fid (Class ⋈ class.classid = enrolled.classid (Student ⋈student.sid=enrolled.sid Enrolled)))))  
)

(viii) Output pairs of students (student1, student2) that took exactly the same courses in the Spring 2020 semester.

S1 <- σyear=2020 ∧ semester=”Spring” (Class ⋈class.classid=enrolled.sid (Student ⋈ student.sid=enrolled.sid Enrolled) as S1  
S2 <- σyear=2020 ∧ semester=”Spring” (Class ⋈ class.classid=enrolled.sid (Student ⋈student.sid=enrolled.sid Enrolled) as S2

πsname1,sname2(σS1.sid ≠ S2.sid∧s1.coursnumber=s2.coursenumber (S1 x S2))

1. Write statements in (Domain or Tuple) Relational Calculus for query (i), (ii), (iii), and (v), or explain why it is not possible to do so.

(i) Output the id, name and major of the graduate students who have already enrolled in more than 30 credits in total.

**This schema would require having to acquire an aggregate sum of all credits for a particular student. This is not possible to represent in domain relational calculus, making it not possible to show this schema.**

(ii) Get the name and id of students who have never selected any course outside the department they are majoring in the semester Fall 2021 and Spring 2022.

(iii) Output the name and department of professors who have never taught class after 2 PM.

(v) Find the students who enrolled in the most classes in Spring 2021, and output the students’ id and names together with all the classes they took in Spring 2021

**This cannot be shown in relational calculus because this requires finding the aggregate output of all classes taken by a student and the maximum classes taken from each student. Because we cannot the sum of all classes, it is not possible to display this schema in domain relational calculus.**