Brandon Vo

CS Databases

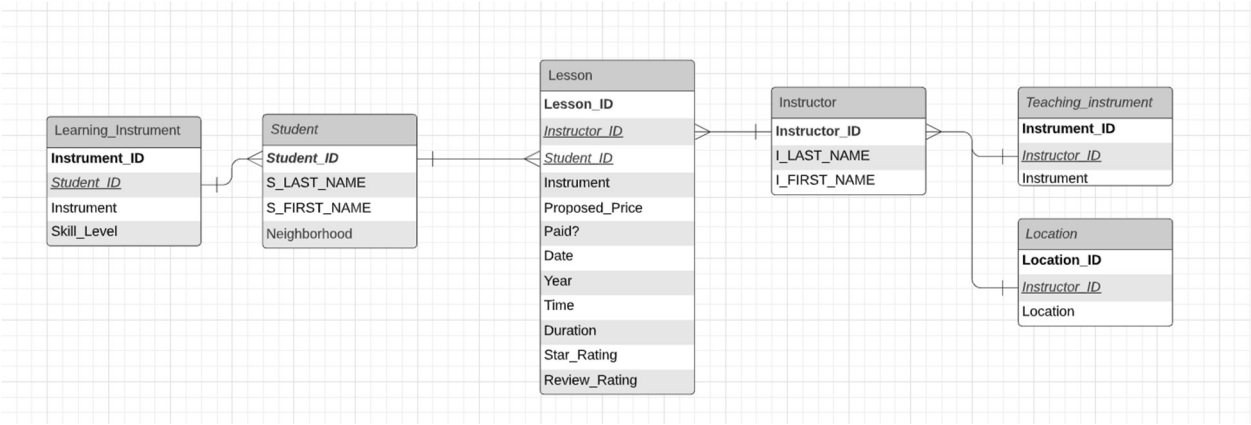
Problem Set #1

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.

(a) 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.



(b) 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.

(i) 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"

(ii) 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);
```

(iii) 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
```

(iv) 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;
```

(v) 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
```

Brandon Vo

```
GROUP BY (Student.student_ID)
HAVING count(Student.student_ID) >= 2;
```

Brandon Vo

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;
```

(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.

```
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);
```

(vi) 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');
```

(vii) 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
```

Brandon Vo

```
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;
```

Brandon Vo

(b) 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.

$\pi_{sid, sname, major}(\gamma_{sum(credits) > 30}$ (Course $\bowtie_{course.deptid = major.deptid}$ (student $\bowtie_{student.mid=major.mid}$ 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.

$\pi_{sname, sid}$ (student)
- $\pi_{sname,sid}$ ($\sigma_{semester="Fall" \wedge class.year=2021}$ $\bowtie$ (course $\bowtie_{course.deptid \neq major.deptid}$ (Major$\bowtie_{major.mid=student.mid}$ (student $\bowtie_{student.sid = enrolled.sid}$ enrolled)))
- $\pi_{sname,sid}$ ($\sigma_{semester="Spring" \wedge class.year=2022}$ $\bowtie$ (course $\bowtie_{course.deptid \neq major.deptid}$(Major$\bowtie_{major.mid=student.mid}$ (student $\bowtie_{student.sid = enrolled.sid}$ enrolled)))

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

$\pi_{fname, deptname}$ (Class $\bowtie_{faculty.fid = class.fid}$ (Faculty $\bowtie_{faculty.fid = department.fid}$(Department)) -
$\pi_{fname, deptname}$ ($\sigma_{time>14:00:00}$ (Class $\bowtie_{faculty.fid = class.fid}$(Faculty $\bowtie_{faculty.fid=Department.fid}$ 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

$\pi_{coursename}$(Course $\bowtie_{course.coursenumber=class.coursenumber}$ ($\sigma_{Year=2022 \wedge semester="Spring"}$ (Class $\bowtie_{class.classid=enrolled.classid}$ ($\sigma_{grade="F"}$ (Student $\bowtie_{student.sid=enrolled.sid}$ Enrolled)))))
- $\pi_{coursename}$(Course $\bowtie_{course.coursenumber=class.coursenumber}$ ($\sigma_{Year=2021 \wedge semester="Spring"}$ (Class $\bowtie_{class.classid=enrolled.classid}$ ($\sigma_{grade="F"}$ (Student $\bowtie_{student.sid=enrolled.sid}$ 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.

S_1<- $_{(sid, classid)}\gamma_{count}$(classid) **as** count_class ($\sigma_{semester="Spring" \wedge year="2021"}$ (Class $\bowtie_{class.classid=enrolled.classid}$ (Student $\bowtie_{student.sid=enrolled.sid}$ enrolled)))

S_2<-$_{(classid)}\gamma_{max}$(classid) **as** max_class(S_1)

($\Pi_{sid,sname}$($\sigma_{semester=max\_class=count\_class \wedge student.sid=s1.sid}$ (S_1 x S_2 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.

π_{sid}(
Department $\bowtie_{department.deptid=faculty.deptid}$ (Faculty $\bowtie_{faculty.fid=class.fid}$(Class $\bowtie_{class.classid = enrolled.classid}$ (Student $\bowtie_{student.sid=enrolled.sid}$ Enrolled)))
/($\sigma_{deptname="CS"}$ (Department $\bowtie_{department.deptid=faculty.deptid}$ (Faculty $\bowtie_{faculty.fid = class.fid}$ (Class $\bowtie_{class.classid =}$

Brandon Vo

$_{enrolled.classid}$ (Student $\bowtie_{student.sid=enrolled.sid}$ Enrolled)))))
)

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

S_1 <- $\sigma_{year=2020 \wedge semester="Spring"}$ (Class $\bowtie_{class.classid=enrolled.sid}$ (Student $\bowtie_{student.sid=enrolled.sid}$ Enrolled) as S_1

S_2 <- $\sigma_{year=2020 \wedge semester="Spring"}$ (Class $\bowtie_{class.classid=enrolled.sid}$ (Student $\bowtie_{student.sid=enrolled.sid}$ Enrolled) as S_2

$\pi_{sname1,sname2}(\sigma_{S1.sid \neq S2.sid \wedge s1.coursnumber=s2.coursenumber}$ (S_1 x S_2))

c) 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.

$$
\begin{aligned}
&\{< sname, sid >| \exists s \in student\{ \\
&\wedge \exists sem, y(< semester, year > \in semester \wedge (sem = \text{"Fall"} \wedge \text{y} = 2021) \\
&(\vee \text{sem} = \text{"}spring\text{"} \wedge y = 2022) \\
&\} \\
&\wedge \neg \exists u \in student\{ \\
&\quad \wedge \exists e1 \in enrolled(e1[sid] \wedge u[sid]) \\
&\quad \wedge \exists cl1 \in class\ (cl1[classid] = e1[classid] \wedge cl1[year] = 2021 \wedge cl1[semester] = Fall \\
&\quad \wedge \exists c1 \in course(\ c1[coursenumber] = cl1[coursenumber]) \\
&\quad \wedge \exists d1 \in department\ (d1[deptid] = course1[deptid] \wedge d1[deptid] \\
&\quad \wedge \exists m1 \in major\ (m1[mid] = u[sid] \wedge c1[deptid] \neq m1[deptid]) \\
&\} \\
\\
&\wedge \neg \exists u2 \in student\{ \\
&\quad \wedge \exists e2 \in enrolled(e2[sid] \wedge u2[sid]) \\
&\quad \wedge \exists cl2 \in class\ (cl2[classid] = e2[classid] \wedge cl2[year] = 2022 \wedge cl2[semester] = Spring \\
&\quad \wedge \exists c2 \in course(\ c2[coursenumber] = cl2[coursenumber]) \\
&\quad \wedge \exists d2 \in department\ (d2[deptid] = course2[deptid] \wedge d2[deptid] \\
&\quad \wedge \exists m2 \in major\ (m2[mid = u2[sid] \wedge c2[deptid]\ \neq m2[deptid]) \\
&\}
\end{aligned}
$$

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

$$
\begin{aligned}
&\{< fname, deptid > | \exists f \in faculty\{ \\
&\wedge \exists d \in department\ (f[deptid] = d[deptid]) \\
&\} \\
&\neg \exists f2 \in faculty\{ \\
&\wedge \exists d2 \in department\ (f2[deptid] = d2[deptid]) \\
&\wedge \exists c2 \in class\ (\ c2[fid] = f2[fid] \wedge c2[fid] > 14{:}00{:}00) \\
&\}
\end{aligned}
$$

(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.