Info 290T: Human-in-the-loop Data Management **Spring 2020**

Assignment 1

Due Wednesday, Feb 19 at 11:59pm

General Instructions

- Feel free to talk to other members of the class in doing the homework. You should, however, write down your solutions yourself. *List the names of everyone you worked with at the top of your submission.*
- Keep your solutions brief and clear.
- Please use Piazza if you have questions about the homework but do not post answers. Feel free to use private posts or come to the office hours.
- Here is the UC Berkeley student code of conduct: `sa.berkeley.edu/code-of-conduct`. We will treat any violations of the code of conduct with the seriousness it deserves.

Homework Submission

- Late Policy: You are allowed up to four late days, to be used in any way, without any explanation necessary. Any submission after the deadline will be rounded up to the closest day (i.e., 24 hour window)—so even one hour late will count as the use of a full day. After four late days, students will lose 1% of the grade per day late.
- We will be using BCourses for collecting the homework assignments. Please submit your answers via BCourses. Hard copies are not accepted.
- Contact Doris Lee if you are having technical difficulties in submitting the assignment; attempt to submit well in advance of the due date/time.
- The homework must be submitted in **pdf** format. Scanned handwritten and/or hand-drawn pictures in your documents won't be accepted.
- Please do not zip the answer document (PDF) so that the instructors can read it directly on BCourses. You need to submit one answer document, named as **hw1_CalID.pdf**.

In this homework, you will be writing relational algebra and SQL queries. You will be operating on the following database about movies:

Information about movies:
Movie (MovieID, DirectorID, Name, Genre, Rating, GrossProfit)
Information about who acted in each movie:
MovieActor (MovieID, ActorID)
Information about directors:
Director (DirectorID, Name, Genre, Studio)
Information about actors:
Actor (ActorID, Name, Age, Agent)

Relational Algebra & SQL Basics (30 pts)

1. [**10pts**] Express the following query:
List the names of movies that either star "Tom Hanks" or have a rating above 90.

(a) In relational algebra:

ANSWER:

(b) In SQL:

ANSWER:

2. [**10pts**] Express the following query:
List the names of directors who have starred as an actor in a movie that they were also directing.

(a) In relational algebra:

ANSWER:

(b) In SQL:

ANSWER:

3. [**5pts**] Describe the following SQL query in a few sentences:

```
SELECT Director.Name, AVG(Movie.Rating)
FROM Movie, Director
WHERE Director.DirectorID = Movie.DirectorID
AND Director.Name LIKE 'Quentin%'
GROUP BY Director.Name
```

ANSWER:

4. [**5pts**] Express the following query in SQL:
What is the average age of actors who have acted in more than 10 movies?
ANSWER:

Rel. Algebra and SQL: A More Advanced Exercise (30 pts)

For this question only, we will use a different schema. This database contains information about students and whether they are research assistants or instructors.
Information about students (status = 'grad' if grad student, 'undergrad' otherwise):
Student(CalID, name, dept, status)
Information about students who are graduate research assistants (GRAs):
GRA(CalID, advisor, dept)
Information about students who are graduate student instructors (GSIs):
GSI(CalID, course, dept)

5. [**15pts**] Express the following query:
Find the names of all graduate students who are neither a GRA nor a GSI.

(a) In relational algebra:

ANSWER:

(b) In SQL:

ANSWER:

6. [**15pts**] Express the following query:
Find the names of all graduate students who are a GSI or a GRA in a department other than their own.

(a) In relational algebra:

ANSWER:

(b) In SQL:

ANSWER:

New Ways to Use Subqueries (25 pts)

7. [**10pts**] As we saw in class, subqueries are a convenient way of using a query result as part of an expression, either as a scalar or via `(NOT) EXISTS`. The latter treats the subquery result as a set, and checks if it is empty or not.

There are other ways to use subqueries as sets to determine whether some boolean condition is satisfied. One such way is the expression `<a> IN (S)`, where `S` is a subquery. This expression tests whether some value `<a>` is present in the set of values represented by `S`. (While `IN` can also be used when the subquery result `S` has multiple attributes, for this question we focus on the setting when `S` has a single attribute, but many rows.)

As an example, say we want to list movie directors who have directed action movies; we can do it using the following query.

```
SELECT Director.Name
FROM Director
WHERE DirectorID IN (SELECT DirectorID
                     FROM Movie
                     WHERE Genre = 'Action')
```

Now, express the following query in SQL, using the `IN` clause.
List the names of actors, who have at least one movie with a perfect rating (100%).
ANSWER:

8. [**15pts**] In class, we focused primarily on the use of subqueries within `WHERE` clauses. Subqueries can also be used in the `HAVING` clause, in a very similar manner to the `WHERE` clause. And they can be also used in the `FROM` clause. For example, we can say:

```
SELECT ...
FROM (SELECT Age
      FROM Actor
      WHERE Agent = 'Eva Smith') AS EvaActorAges
WHERE ....
```

where the `EvaActorAges` is the result of a subquery that computes the ages of all actors whose agent is Eva Smith.

Now, express the following query in SQL.
What is the name of the actors whose movies yield the highest average gross profit?
ANSWER:

Defining and Modifying Databases via SQL (15 pts)

Beyond the SELECT-FROM-WHERE queries that we are familiar with, SQL can also be used to create new relations or manipulate existing ones. Examples of SQL operators that are part of the Data Manipulation Language (DML) include INSERT, UPDATE, and DELETE that allow users to modify existing relations. Examples of SQL operators that are part of the Data Definition Language (DDL) include CREATE, ALTER, and DROP that allow users to modify the database schema.

9. [**5pts**] Assume that the initial database is empty (without any relations present), provide the SQL commands to create the following relation and add the associated tuple to the relation.

Movie

MovieID	DirectorID	Name	Genre	Rating	GrossProfit
1	1	Star Wars	SciFi	53	1,027,000,000

Answer:

10. [**5pts**] During data entry, you realize that there is more than one episode of the Star Wars movie. In order to add other Star Wars movies to the database, you first need to change the existing movie entry to "Star Wars: Episode I – The Phantom Menace". Express this modification as a single SQL statement.

Answer:

11. [**5pts**] Later on, we realize that the information that we were storing for `Movie` was not enough: we also need to store a new attribute, `Studio`, with a default value `'Universal'`. Modify the schema of the `Movie` relation to add this attribute.

Answer: