Overview

Complete the following problems. You should turn in a carefully written solution showing your work and **explaining your answer**.

This homework will be graded anonymously via Moodle's anonymous marking feature, so *do not* include your name anywhere in your submission.

I highly recommend learning LaTeX, which is the standard tool in theoretical computer science for communicating technical material, and it is valuable to get the practice with it. If you have no experience with LaTeX at all, Getting Something out of LaTeX is a fine place to start. LaTeX Tutorials — A Primer is a more detailed tutorial. Text Processing using LaTeX is a great reference for the language. I choose to prepare my LaTeX documents using the Emacs text editor, but you may choose another text editor or environment if you prefer. I've had good luck with both ShareLaTeX and Overleaf also, both of which are web-based. If you prefer, you can use a different word processing tool. You may also submit your work written out by hand, but be careful to write it out carefully and neatly if you do. You'll also need to scan it in order to submit it electronically.

You should submit your assignment as a PDF via Moodle.

Collaboration policy: As with all non-programming assignments in this course, you may collaborate on the homework assignments to the extent of formulating ideas as a group, but you may not collaborate in the actual writing of solutions. *In particular, you may not work from notes taken during collaborative sessions.* You *must* cite all sources, including others in the class from whom you obtained ideas. You may not consult any materials from any previous offerings of this course or from any other similar course offered elsewhere. You are required to completely understand any solution that you submit, and, in case of any doubt, you must be prepared to orally explain your solution to me. *If you have submitted a solution that you cannot verbally explain to me, then you have violated this policy.*

Problem 1

Complete exercise 2.10 in your textbook.

Problem 2

A relation schema consists of a *set* of attributes. A relation instance consists of a *set* of tuples. Since these are sets, there is no specific ordering associated with either the attributes or the tuples. However, if I wish to put into print a sample relation instance, I need to pick an arbitrary ordering. For example, look at Figure 2.5 in your book, which shows a relation instance for the *department* relation. Even though conceptually there is no order associated with the tuples or the attributes, an order had to be chosen in order to display it on the page.

How many different ways (considering orders of tuples and attributes) are there to represent a relation instance in print if that instance has...

(a) Three attributes and three tuples?

(b) Four attributes and five tuples?

(c) n attributes and m tuples?

Clarification: page 46 of your textbook says "It is customary to list the primary key attributes of a relation schema before the other attributes." True, it is customary, but not mandatory. For purposes of answering this problem, don't assume that the primary key needs to be ordered in some special way. Said differently, don't worry about primary keys at all for the above problem.

(*Thanks to Ullman and Widom.*)

Problem 3

Suppose that we have the following movie database schema, where for each relation, the underlined attributes jointly form the primary key:

`Movie(`<u>`title`</u>`, `<u>`year`</u>`, length, fgenre, studioName, producerCertNum)`
`StarsIn(`<u>`movieTitle`</u>`, `<u>`movieYear`</u>`, `<u>`starName`</u>`)`
`MovieStar(`<u>`name`</u>`, address, gender, birthdate)`
`MovieExec(name, address, `<u>`certNum`</u>`, netWorth)`
`Studio(`<u>`name`</u>`, address, presidentCertNum)`

(Note on 9/14, 3:55 pm: I added an underline to `year` in `Movie`)

(We're making up that every non-acting movie exec/producer/etc in the movie industry has a "certificate number," which is referred to in a number of the tables above.)

Express each of the following constraints by suggesting a foreign key that, if put in place, would enforce that constraint. The answer for each of these problems should be of the form

"The attribute X in relation R should be a foreign key, referencing attribute Y in relation S."

If the specified constraint is not possible to be done as a foreign key constraint, explain why.

(a) The producer of a movie must be someone mentioned in `MovieExec`.

(b) A movie that appears in `StarsIn` must also appear in `Movie`.

(c) A star appearing in `StarsIn` must also appear in `MovieStar`.

(d) Every movie in the relation `Movie` must appear with at least one star in `StarsIn`.

(*Thanks to Ullman and Widom.*)

Problem 4

The MusicBrainz project is an open music encyclopedia that collects data about music: artists, titles, release dates, and so on. There is lots of detailed information about its database schema. I was looking around at it because it seemed like it might be a fun example, and came across

some interesting details they had to work their way through: specifically, artist credits caught my attention. Read the artist credits information page, and then look at the entire database schema. We haven't looked at these diagrams in detail yet, but you should be able to make some sense out of focusing on the relations `artist_credit`, `artist_credit_name`, and `artist`.

There are three examples on the MusicBrainz Schema page (scroll down to the "Artist Credit" section) that describe situations where the artist credit concept is relevant. Specifically, those examples are:

- "Queen & David Bowie" – two artists ("Queen" and "David Bowie"), no name variations, joined with "&"
- "Jean-Michel Jarre" – one artist ("Jean Michel Jarre"), name variation "Jean-Michel Jarre"
- "Tracy W. Bush, Derek Duke, Jason Hayes and Glenn Stafford" – four artists, no name variations, joined with commas and an "and".

  Draw relation instances for the `artist_credit`, `artist_credit_name`, and `artist` relations with data in it that captures the above examples. For the `artist` relation, you only need to include the attributes `id` and `name` for purposes of this exercise. For the `artist_credit` relation, you only need to include the attributes `id`, `name`, and `artist_count`. You should include all attributes shown in the schema diagram for `artist_credit_name`. If the examples don't give you the information that you need to fill in some of the values, make something up.

  For the join_phrase attribute, you may need to sometimes have a value that means there is no join phrase. You can leave the value blank in that case, or alternatively put in NULL. In the pure relational model, there is no such thing as "no value," but essentially any real database system allows you to use NULL as a value.

  (*You could likely accomplish this task by downloading the actual MusicBrainz database and looking at the raw data. Don't do that. The point of this exercise is to get you thinking about design considerations.*)