Problem Set 1

Preliminaries

If you haven't already done so, you should complete Lab 0 before beginning this assignment.

In your work on this assignment, make sure to abide by the collaboration policies of the course. ***All of the problems in this assignment are individual-only problems that you must complete on your own.***

If you have questions, please come to office hours, post them on Piazza, or email `cs460-staff@cs.bu.edu`.

Make sure to submit your work on Gradescope, following the procedures found at the end of Part I and Part II.

Part I

due by 11:59 p.m. on Wednesday, February 6
30 points total

Creating the necessary file

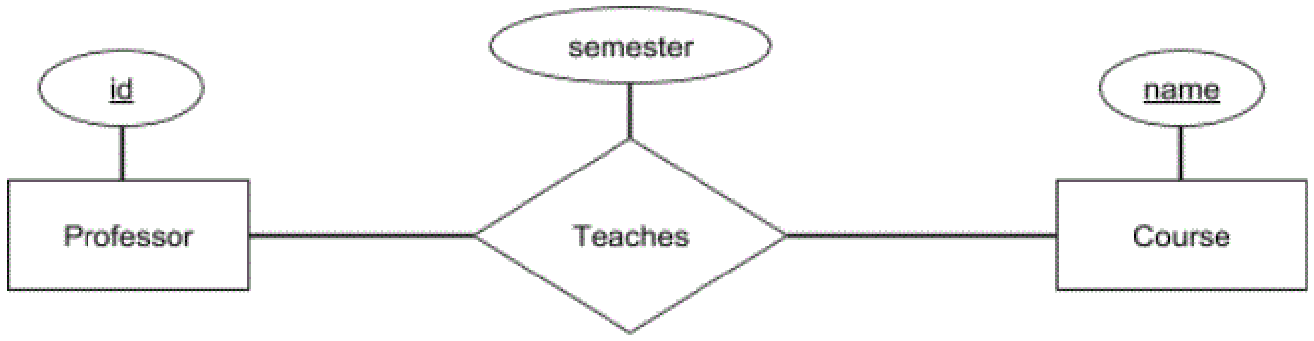
This part of the assignment will all be completed in a single PDF file. To create it, you should do the following:

1. Open the template that we have created for Part I in Google Docs: ps1_partI
2. Select *File->Make a copy...*, and save the copy to your Google Drive using the name `ps1_partI`.
3. Add your work for all of the problems from Part I to this file.
4. Once you have completed Part I, choose *File->Download as->PDF document*, and save the PDF file on your machine. The resulting PDF file (`ps1_partI.pdf`) is the one that you will submit. See the submission guidelines at the end of Part I.

Problem 1: ER diagram basics

8 points total; 2 pts. each part

The ER diagram shown below is part of the design of a database that includes information about professors and the courses they teach.



In the above version of the ER diagram, there are no constraints on the relationships between professors and courses.

In the `ps1_partI` template that we've provided on Google Drive (see above), we've included the beginnings of three separate versions of this ER diagram.

1. Edit the first version of the diagram, adding the connections needed to create an ER diagram whose only constraint is that every course is taught by *at least one* professor. To do so, you should:

- Click on the diagam and then click the *Edit* link that appears below the diagram.
- From the collection of eight connectors that we have provided below the diagram, select the appropriate connectors and use them to connect the two entity sets to the relationship set. Lengthen the connectors as needed to make the connections.
- Click the *Save & Close* button.

2. Edit the second version of the diagram, adding the connections needed to create an ER diagram whose only constraint is that every course is taught by *at most one* professor.
3. Edit the third version of the diagram, adding the connections needed to create an ER diagram whose only constraint is that every professor teaches *exactly one* course.
4. Consider your answer to part 2—the ER diagram for the situation in which every course is taught by *at most one* professor. If we converted that ER model to a relational schema, would the following be an acceptable schema for a relation used to capture the Teaches relationship set?

   Teaches(<u>professor</u>, <u>course</u>, semester)

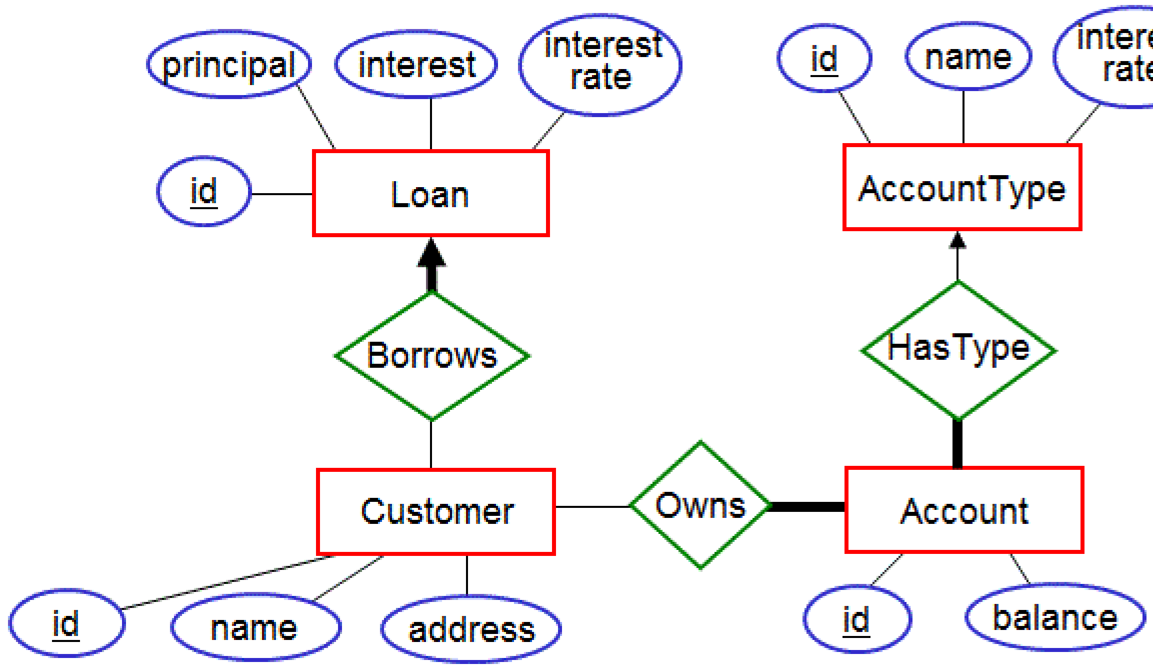
   where *professor* is a foreign key referring to *Professor(id)*, *course* is a foreign key referring to *Course(name)*, and the primary-key attributes of *Teaches* are underlined. Explain your answer briefly.

Problem 2: Database design

12 points total

The ER diagram shown below in Figure 2-1 represents information that is to be stored in a bank database.

Figure 2-1:



Customers borrow loans and own accounts, and accounts have account types.

1. (3 points) At least one of the relationship sets captures a many-to-one relationship. Which one(s)? In your answer, you should specify the direction of the relationship (e.g., ________ *is a many-to-one relationship from* ________ *to* __________).

2. (3 points) Describe all constraints on relationships that are specified by the diagram. Use words that describe the problem domain (e.g., *Each course meets in at most one room...*) rather than technical terminology.

3. (6 points) Transform this diagram into a relational schema by following the procedure discussed in lecture. Here are some additional guidelines:

   - Give the schema of each relation in the form *relation_name(attr_name1, attr_name2, ...)*.

   - When appropriate, you should combine relations as discussed in lecture. If you do combine two relations, you should briefly explain why it makes sense to do so.

   - Indicate the primary-key attribute(s) of each relation by putting an underscore character (_) on either side of the attribute name(s). For example, if you had a key called `id`, you would write it as `_id_`.

   - Specify each relation's foreign-key attribute(s) (if any) and state the associated referential-integrity constraints. For example, if you were working with the *MajorsIn* relation from the lecture notes, one of its referential-integrity constraints could be specified as follows: *Each value of the student attribute in MajorsIn must match a value of the id attribute from the Student relation.*

Problem 3: Combining relations

10 points total; 2 pts. each part

We will cover the material needed for this problem in lecture on Wednesday, January 28.

Relation R has attributes a and b. Relation S has attributes c, b, and a. You are given the following instances of these relations:

Relation R

a	b
1	2
3	4
5	6

Relation S

c	b	a
3	2	1
5	4	3
9	8	7

For each of the following questions, use the *Insert->Table* menu option in Google Drive to insert an appropriately sized table for the answer, and then fill in the cells of the table with the correct values.

1. What is the Cartesian product of R and S?
2. What is the natural join of R and S?
3. What is the left outer join of R and S?
4. What is the right outer join of R and S?
5. What is the full outer join of R and S?

Submitting your work for Part I

1. Once you have completed Part I in Google Drive, choose *File->Download as->PDF document*, and save the resulting file (`ps1_partI.pdf`) on your machine.
2. Login to Gradescope by clicking the following link:

Gradescope

If you don't have a Gradescope account, you should create one now using your BU email address.

3. Once you are in logged in, click on the box for ***CS 460***. (If you don't see that box, email `cs460-staff@bu.edu` ASAP and ask to be added to the course on Gradescope.)

4. Click on the name ***Problem Set 1, Part I*** in the list of assignments. You should see a pop-up window labeled ***Submit Assignment***. (If you don't see it, click the ***Submit*** or ***Resubmit*** button at the bottom of the page.)

5. Choose the ***Submit PDF*** option, and then click the *Select PDF* button and find the `ps1_partI.pdf` that you created in step 1. Then click the *Upload PDF* button.

6. You should see an outline of the problems along with thumbnails of the pages from your uploaded PDF. For each problem in the outline:

   - Click the title of the problem.
   - Click the page(s) on which your work for that problem can be found.

   As you do so, click on the magnifying glass icon for each page and doublecheck that the pages that you see contain the work that you want us to grade.

7. Once you have assigned pages to all of the problems in the question outline, click the *Submit* button in the lower-right corner of the window.

8. You should see a box saying that your submission was successful. Click the `(x)` button to close that box.

9. You can use the ***Resubmit*** button at the bottom of the page to resubmit your work as many times as needed before the final deadline.

Important

- It is your responsibility to ensure that the correct version of a file is on Gradescope before the final deadline. ***We will not accept any file after the submission window for a given assignment has closed, so please check your submission carefully using the steps outlined above.***

- If you are unable to access Gradescope and there is enough time to do so, wait an hour or two and then try again. If you are unable to submit and it is close to the deadline, email your homework ***before the deadline*** to `cs460-staff@cs.bu.edu`

Part II

due by 11:59 p.m. on Wednesday, February 13
70 points total

We will cover the material needed for this part of the assignment beginning on Wednesday, January 28, and continuing into the week of February 4.

Our movie database

Our problem domain is movie trivia, and the database that we'll be using contains:

- the winners of the Academy Awards for Best Picture, Best Director, Best Actress, Best Actor, Best Supporting Actress, and Best Supporting Actor from the Oscars' inception in 1929 to the present, as well as the films for which the acting and directing awards were won
- the 200 top-grossing films of all time (as of September 18, 2018)—i.e., the films that have made the most money—as well as some former top-grossers that have since been demoted
- the five top-billed cast members and the director(s) of each film included in the database.

The source of our data is a great website called imdb.com, the Internet Movie Database.

The tables

Below are the schema of the five tables in our movie database. ***The names of the primary-key attributes are italicized.***

- The `Movie` table/relation stores information about movies. It has the following attributes:

attribute name	data type	description
	`CHAR(7)`	a unique id assigned to a movie
`name`	`VARCHAR(64)`	the name of the movie
`year`	`INTEGER`	the year the movie was released
`rating`	`VARCHAR(5)`	the MPAA rating of the movie; if the movie is unrated or has a non-standard rating, this value is `NULL`
`runtime`	`INTEGER`	the length of the movie in minutes
`genre`	`VARCHAR(16)`	the genre(s) of the film, formed by concatenating one-letter genre codes (e.g., A for action and D for drama)

attribute name	data type	description
earnings_rank	INTEGER	the earnings rank of the film. The film that has made the most money has an earnings rank of 1; the film that has made the second largest amount of money has an earnings rank of 2, etc. This attribute only has a non-NULL value for the 200 top-grossing movies. All other movies have a NULL value for their earnings_rank value.

- The Person table/relation stores information about actors, actresses, and directors. It has the following attributes:

attribute name	data type	description
	CHAR(7)	a unique id assigned to a person
name	VARCHAR(128)	the person's full name (first name first)
dob	DATE	the person's date of birth, if known; NULL otherwise
pob	VARCHAR(128)	the person's place of birth, if known; NULL otherwise

- The Actor table/relation captures relationships between actors and the movies in which they have acted. It has the following attributes:

attribute name	data type	description
_	CHAR(7)	the id attribute of an actor; a foreign key that references Person(id)
_	CHAR(7)	the id attribute of a movie in which the actor appeared; a foreign key that references Movie(id)

- The Director table/relation captures relationships between directors and the movies they have directed. It has the following attributes:

attribute name	**data type**	**description**
_	CHAR(7)	the `id` attribute of a director; a foreign key that references `Person(id)`
_	CHAR(7)	the `id` attribute of a movie that he or she directed; a foreign key that references `Movie(id)`

- The `Oscar` table/relation stores information about Academy Awards. It has the following attributes:

attribute name	**data type**	**description**
`movie_id`	`CHAR(7)`	the `id` attribute of a movie that was awarded an Oscar; a foreign key that references `Movie(id)`
`person_id`	`CHAR(7)`	the id attribute of the actor, actress, or director who won the award; if type is `'BEST-PICTURE'` (see below), this attribute is `NULL`; a foreign key that references `Person(id)`
`type`	`VARCHAR(23)`	the type of Oscar won, which will be one of the following strings: `'BEST-PICTURE'`, `'BEST-DIRECTOR'`, `'BEST-ACTRESS'`, `'BEST-ACTOR'`, `'BEST-SUPPORTING-ACTRESS'`, `'BEST-SUPPORTING-ACTOR'`
`year`	`INTEGER`	the year in which the Oscar was won

(**Note**: We chose not to specify a primary key for the `Oscar` relation. One reason for our decision is that you need the combination of all four attributes in order to guarantee uniqueness. In addition, one of those four attributes (`person_id`) has `NULL` values in some tuples, and `NULL` values are not allowed in primary-key attributes. Instead of specifying a primary key, we use a `UNIQUE` clause to specify that the combination of the four attributes must be unique.)

Example of how the foreign keys are used
As noted above, the database uses foreign keys to capture relationships. For example, Tom Hanks won the Best Actor Oscar in 1994 for his performance in the movie *Philadelphia*, which was directed by Jonathan Demme. Hanks has an id of `0000158`, Demme has an id of `0001129`, and the movie has an id of `0107818`. To capture the necessary relationships, the following tuples appear in the database:

- `('0000158', '0107818')` in the `Actor` table, to capture the fact that Tom Hanks acted in *Philadelphia*

- `('0001129', '0107818')` in the `Director` table, to capture the fact that Jonathan Demme directed *Philadelphia*
- `('0107818', '0000158', 'BEST-ACTOR', 1994)` in the `Oscar` table, to capture the Oscar that Tom Hanks won for his performance in *Philadephia*.

In addition, there is a tuple for the movie itself in the `Movie` table, and there are tuples for Hanks and Demme in the `Person` table.

Downloading the necessary files

1. Go to sqlitebrowser.org.
2. Find the appropriate download button for your machine/operating system, and download that version of DB Browser for SQLite.
3. Run the installer. You should be able to use all of the default options.
4. Download the following files by right-clicking on each link:
   - `movie.sqlite`
   - `ps1_queries.py`

Using DB Browser for SQLite

1. Launch the program.
2. Click the *Open Database* button, and find and open the `movie.sqlite` database file that you downloaded above.
3. To explore the schema of the database, click on the *Database Structure* tab, and then click on the arrows to the left of the table names.
4. To explore the contents of the tables, click on the *Browse Data* tab, and then choose the appropriate table from the drop-down menu.
5. Use the *Execute SQL* tab to perform queries on the database. Enter your SQL command in the space provided, and press F5 or Ctrl-R to run it. (There is also a small button that you can click; it has a triangular shape that looks like the Play button of a music player.)

Important guidelines

1. Open `ps1_queries.py` in a text editor, saving it as a plain-text file. This file is a Python file, so you could use a Python IDE to edit it, but a regular text editor like TextEdit or Notepad++ would also be fine.
2. Construct the SQL commands needed to solve the problems given below. Test each SQL command in DB Browser for SQLite to make sure that it works.

3. Once you have finalized the SQL command for a given problem, copy the command into your `ps1_queries.py` file, putting it between the triple quotes provided for that problem's variable. We have included a sample query to show you what the format of your answers should look like.
4. **Make sure that every query ends with a semi-colon.**
5. Unless otherwise stated, each of the problems must be solved by means of **a single query** (i.e., each query should have a single semi-colon). Use nested subqueries as needed.
6. Your queries should *not* use a `LIMIT` clause.
7. Make sure that the results produced by your queries contain exact answers to the problems. We should not have to infer the answer from the results. For example, if we ask you how many films meet a given criterion, we want a number, not a list of the films. In addition, your results should not include any extraneous information.

Problem 4: Favourite stars

5 points

Make sure to read the important guidelines above before you get started!

Emma Stone and Rachel Weisz have both been nominated for Best Supporting Actress for their performances in *The Favourite*. Write a single query to find the places of birth and dates of birth for these two people. The result of the query should be tuples of the form (name of person, place of birth, date of birth).

Hint: If your initial query does not produce any results, you may want to reconsider the logical operator (AND, OR, NOT) that you are using in your WHERE clause.

Problem 5: Long movies

5 points

Write a query that determines the number of movies in the database that have a runtime of more than 200 minutes. The result of the query should be a single number.

Problem 6: Oscars won by Christian Bale

5 points

Christian Bale has been nominated for a Best Actor Oscar for his performance in *Vice*. If he wins, it will be his second Oscar. Write a query to find the name of the movie for which he won his earlier Oscar, the type of award that he won, and the year of the award. The result of your query should be a tuple of the form (movie name, award type, year won).

Problem 7: Foreign-born directors of foreign-born actors

5 points

Write a query that finds the number of foreign-born directors who have directed one or more movies in which a foreign-born actor or actress appeared. A foreign-born person is someone who was born outside of the US, and you may assume that all people who *were* born in the US have the word ‘USA’ at the end of their place of birth. *Hint:* You may find it helpful for your FROM clause to include multiple instances of at least one of the tables in the database.

Problem 8: Average runtimes per year

5 points

How does the average runtime of movies change over time? Write a query that determines, for every year from 2000 until the present, the average runtime of movies from that year. The result of your query should be tuples of the form (year, average runtime for that year). Order the results from oldest to most recent.

Problem 9: Sam He Is

5 points

Sam Rockwell is nominated for Best Supporting Actor for his work in *Vice*. Write a query to determine the names and dates of birth of all people in the database whose first name is Sam. Use pattern-matching as needed, and make sure that you only request people whose first name is exactly ‘Sam’. *Hint:* Think about how to construct a pattern that obtains people whose first name is ‘Sam’ without getting people whose first name begins with ‘Sam’ (e.g., Samuel) or people who have ‘Sam’ as all or part of their last name.

Problem 10: Shorter than all Best Pictures

5 points

Write a query that determines the names of the movies in the database that are shorter than all of the movies that have been named Best Picture? *Hint:* You will need a subquery.

Problem 11: Money-making directors

5 points

Which directors have been most successful at the box office? Write a query that finds all directors who have directed **at least four** of the 200 top-grossing films. The result of the query should consist of tuples of the form (director, number of top grossers). List them in order of the number of top-grossers they have directed, from the highest number to the lowest. You may assume that all directors in the database have a unique name.

Problem 12: Earnings gold *and* Oscar gold?

5 points

Do the Oscars reward top-grossing films? Write a query that determines the number of Oscars won by each of the top 25 grossing films. The results of your query should contain tuples of the form (earnings rank, movie name, number of Oscars). Sort the tuples by earnings rank.

Problem 13: Actors with no top grossers

5 points

Write a query that determines the number of actors and actresses in the database who have *not* acted in any of 200 top-grossing movies. The result of the query should be a single number, and the count should *not* include people who are only in the database as a director.

Problem 14: Denzel

5 points

Denzel Washinton appears in the database as both an actor and a director. Write a query that lists all of the movies in the database with which he is associated and the function(s) that he played in each case. If he served as both an actor and a director for a given film, you should have two entries for that film in the results – one for actor and one for director. Sort the tuples so that multiple entries for a given movie will appear together. The comparable results table for Woody Allen looks like this:

```
+--------------------------+----------+
| movie                    | function |
+--------------------------+----------+
| Annie Hall               | actor    |
| Annie Hall               | director |
| Blue Jasmine             | director |
| Bullets Over Broadway    | director |
| Hannah and Her Sisters   | director |
| Mighty Aphrodite         | director |
| Vicky Cristina Barcelona | director |
+--------------------------+----------+
```

Hints:

- You can create tuples that include a constant-valued attribute by including the constant value in the SELECT clause. For example: SELECT Foo.bar, ‘hello’ ... will produce tuples whose second value is always ‘hello’.
- You may find it helpful to use one of the set operations available in SQL.
- Use aliases to obtain the correct column headings.

Problem 15: The most recent G-rated Best Picture

5 points

Write a query to determine the last (i.e., most recent) time that a G-rated movie won Best Picture. Your answer should be a single tuple of the form (movie name, year in which it won Best Picture). *Hint:* “Most recent” is equivalent to having the largest year.

> Important
>
> For full credit, your answer *must* use a subquery. Although SQLite provides a way to solve this problem without a subquery, that technique is not standard SQL, and thus it should not be used as part of your answer.

Problem 16: Adding a rating

5 points

The PG-13 rating was created in 1984 in response to concerns by parents that some movies with PG ratings were inappropriate for children. One of those movies was *Indiana Jones and the Temple of Doom*, which was never officially re-rated. Write a single SQL command that changes the rating of this movie to PG-13. *Hint*: This is the only question that does *not* call for a SELECT command. You will need to figure out the correct type of SQL command to use.

Problem 17: Relational-algebra queries

10 points total

Your work on this problem should go in a separate PDF file called _ . . See below for how to create it.

Write a relational-algebra query for each of the following problems. You already solved these problems using SQL; now you will solve them using relational algebra instead.

1. (3 points) problem 4 (Favourite stars)

2. (3 points) problem 6 (Oscars won by Christian Bale)

3. (4 points) problem 12 (Earnings gold *and* Oscar gold?) ***Instead of producing counts, your relational-algebra query should produce tuples of the form (earnings rank, movie name, award type)***. If a movie has won multiple awards, it should have multiple tuples in the results, one for each award. If a movie has won no awards, it should have a single tuple with a `NULL` value for the award type. You do *not* need to sort the results.

To create your PDF file, you should:

1. Open the template that we have created for it in Google Docs.

2. Select *File->Make a copy...*, and save the copy to your Google Drive using the name `ps1_problem17`.

3. Add your relational-algebra queries to this file.

To simplify your answers, you may:

- Replace the relational-algebra symbols with appropriate words (e.g., `JOIN` for ⋈, `SELECT` for σ, etc.) and you should surround with curly braces any subscripts that accompany the operators (e.g., `JOIN{id = student}`). For example, here is how you could write a query to get the name and year of all movies in the database with an R rating:

  ```
  PROJECT{name, year}(SELECT{rating = 'R'}(Movie))
  ```

- Perform a sequence of two or more queries in which the results of a given query are assigned to a variable using the assignment operator (`<--`), and then that variable is used in a subsequent query. For example, here is an alternate way of writing the above query:

  ```
  Rmovies <-- SELECT{rating = 'R'}(Movie)
  PROJECT{name, year}(Rmovies)
  ```

  Note that using a sequence of queries is *not* allowed in your SQL answers, but it is allowed here.

4. Once you have added all of your queries to the file, choose *File->Download as->PDF document*, and save the PDF file on your machine. The resulting PDF file (`ps1_problem17.pdf`) should be submitted with your other work from Part II.

Submitting your work for Part II

These instructions will be added soon!