CS 325 - Homework 9

Deadline

11:59 pm on **Friday, November 12, 2021**.

Purpose

To read and think about some topics related to transaction management; to get more practice converting ER models into a (partial) database designs/schemas (including converting supertype/subtype entity classes); to practice writing queries involving joins of more than two tables; to practice more with SQL views; and to get more experience with some combinations of the SQL features we have discussed so far.

How to submit

Problem 1 is completed on the course Canvas site.

For Problem 2 onward:

Each time you wish to submit, within the directory `325hw9` on nrs-projects.humboldt.edu (and at the nrs-projects UNIX prompt, **NOT inside** `sqlplus`!) type:

```
~st10/325submit
```

...to submit your current files, using a homework number of `9`.

(**Make sure** that the files you intend to submit are listed as having been submitted!)

Additional notes:

- **Reminder**: CS 325 course style for **relation-structure form** includes:
  - Write all attributes making up a relation's primary key in all-uppercase
  - For foreign keys, list their attributes as usual in the parentheses, but then also write a SQL-style foreign key clause after the closing parenthesis.
  - For example:

```
Rental(RENTAL_NUM, client_num, vid_id, date_out, date_due, date_returned)
    foreign key (client_num) references client,
    foreign key(vid_id) references video
```

- You are required to use the HSU Oracle `student` database for **Problem 3** of this homework.
- **DB Reading Packets 8 and 9** and **SQL Reading Packet 7**, on the course Canvas site, and the Week 11 Asynchronous Materials, along with the posted in-class projections from the public course web site, are useful references for this homework.
  - (But since some of the queries also deliberately combine features we have discussed earlier, you may also find it useful to refer to previous SQL Reading Packets, also.)
- Now that we have covered the `order by` clause, you are expected to use it appropriately when an

explicit row ordering is specified. Queries for problems asking for *explicit* row ordering will be incorrect if they do not include a reasonable `order by` clause.

- Feel free to add additional `prompt` commands to your SQL scripts as desired to enhance the readability of the resulting output.
- An example `325hw9-out.txt` has been posted along with this homework handout, to help you see if you are on the right track with your queries for Problem 3. If your `325hw9-out.txt` matches this posted one, that doesn't guarantee that you wrote appropriate queries, but it is an encouraging sign.
- You are expected to follow **course style standards** for SQL `select` statements.
  - On the CS 325 public course web site, under "References", there are now some evolving lists of course style standards posted. There is also a link to these on the course Canvas home page.

Problem 1

Correctly complete the "HW 9 - Problem 1 - Reading Questions for DB Reading Packet 9 - Database Design, Part 2", on the course Canvas site.

Setup for Problems 2 onward

Use `ssh` to connect to `nrs-projects.humboldt.edu`, and create, protect, and go to a directory named `325hw9` on nrs-projects:

```
mkdir 325hw9
chmod 700 325hw9
cd 325hw9
```

Put all of your files for this homework in this directory. (And it is from this directory that you should type `~st10/325submit` to submit your files each time you want to submit the work you have done so far.)

Problem 2

Use `nano` (or `vi` or `emacs`) to create a file named `325hw9-db-designs.txt`:

```
nano 325hw9-db-designs.txt
```

Place your answers for this problem into file `325hw9-db-designs.txt`

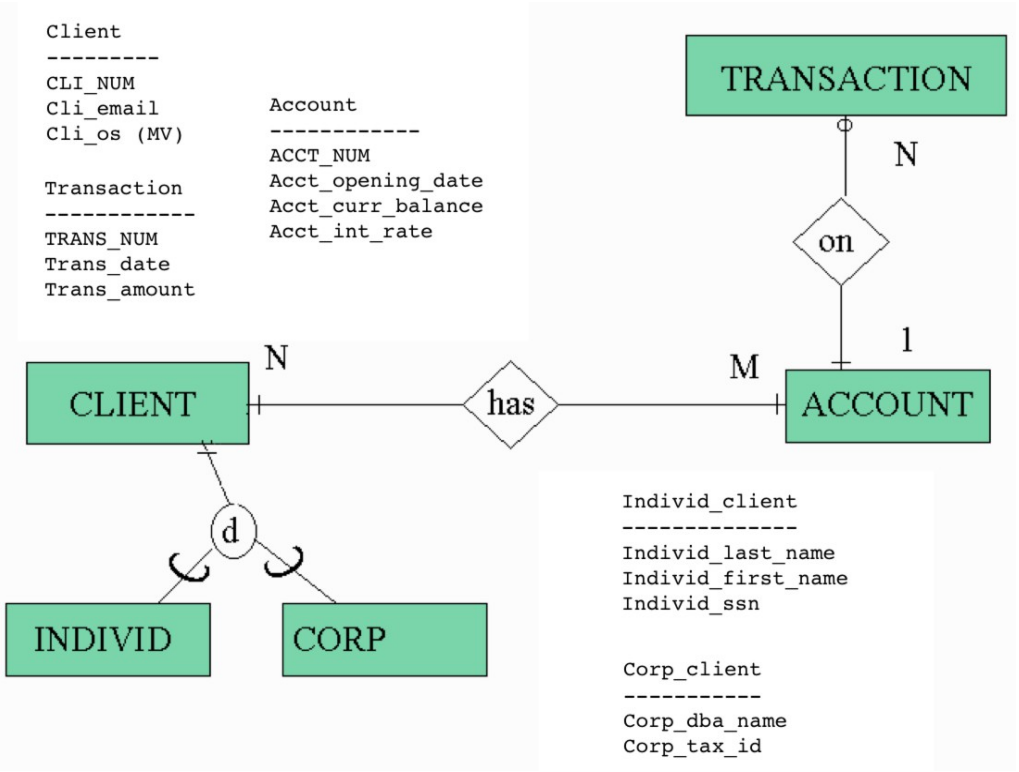
For this problem, you will be converting two database models into (partial) database designs/schemas. (Why partial? Because, again for this assignment, we are not including domains or business rules, which are part of a database design/schema, also.)

Consider the following ER models. Convert each into an appropriate corresponding (partial) design/schema, using the conversion rules discussed in lecture. Your resulting database designs/schemas needs to meet the following requirements:

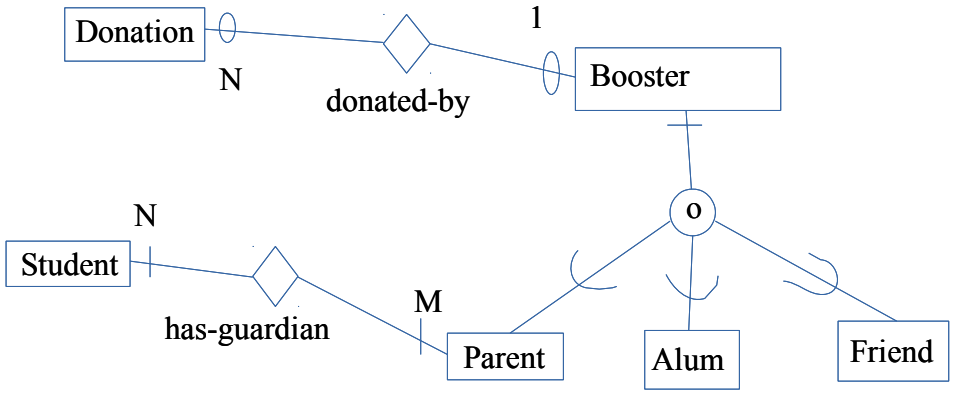
* for this problem, you will list your resulting tables in relation structure form, indicating foreign keys by writing SQL foreign key clauses after the relation structure.
* make sure, for each table, that you clearly indicate primary key attributes by writing them in all-uppercase (and by writing non-primary-key attributes NOT in all-uppercase).
* do not make ANY inferences/assumptions NOT supported by the given models or stated along with

them. (Assume that the models DO reflect the scenarios faithfully.)

Problem 2-1's model:



Problem 2-2's model:



Donation	Booster	Student	Parent	Alum	Friend
DON_NUM	BOOSTER_NUM	STU_ID	Vol_hrs	Year_left	Year_joined
Don_date	Booster_lname	Stu_lname	Prefd_phone		
Don_amount	Booster_fname	Stu_fname			
	Booster_email	Stu_grade			
		Stu_gpa			

Problem 3

This problem again uses the tables created by the SQL script `movies-create.sql` and populated by `movies-pop.sql`. As a reminder, these tables can be described in relation structure form as:

```
Movie_category(CATEGORY_CODE, category_name)

Client(CLIENT_NUM, client_lname, client_fname, client_phone,
client_credit_rtg,
       client_fave_cat)
   foreign key (client_fave_cat) references movie_category(category_code)

Movie(MOVIE_NUM, movie_title, movie_director_lname, movie_yr_released,
      movie_rating, category_code)
   foreign key(category_code) references movie_category

Video(VID_ID, vid_format, vid_purchase_date, vid_rental_price, movie_num)
   foreign key (movie_num) references movie

Rental(RENTAL_NUM, client_num, vid_id, date_out, date_due, date_returned)
   foreign key (client_num) references client,
   foreign key(vid_id) references video
```

And, again, for your convenience as a reference, a handout of these relation structures is posted along with this homework handout.

Because we modified these tables' contents in Homework 8, for Homework 9 you might want to make sure they have been restored to those populated by `movies-pop.sql`:

- Make a copy of `movies-pop.sql` in your `325hw9` directory -- one way to do so is using:

  ```
  cp ~st10/movies-pop.sql .
  ```

  (REMEMBER the space and the . at the end!)

- Now enter `sqlplus` and run your `movies-pop.sql` copy.

Create a file named `325hw9.sql` and include the following within one or more SQL **comments**:

- your name
- `CS 325 - Homework 9 - Problem 3`
- the date this file was last modified

Then:

- use `spool` to start writing the results for this script's actions into a file `325hw9-out.txt`
- put in a `prompt` command printing `Homework 9 Problem 3`
- put in a `prompt` command printing your name
- include a `spool off` command, at the BOTTOM/END of this file. Type your answers to the problems below BEFORE this `spool off` command!

Now, within your file `325hw9.sql`, add in SQL statements for the following, **PRECEDING** EACH with a SQL*Plus `prompt` command noting what problem part it is for.

Problem 3-1

To practice a join of more than two tables: write a query that projects, for each movie category **name**, the

number of videos of movies in that category, displaying the results in reverse order of the number of videos of movies in that category. Give the second column the column alias `NUM_VIDEOS`.

HINTS:

- in an equi-join of *n* tables, make sure you have *n-1* join conditions
- look carefully at the foreign keys of tables involved to determine what those join conditions should be, looking for columns with common domains where their being equal in the Cartesian product has useful meaning
- remember what `select` clause is needed when you want to project a column along with an aggregate function result...

Problem 3-2

The store would like to give counter employees a very limited view of client information (they will be `grant`ed access to this view instead of to the `client` table itself).

Drop and create a view called `counter_client_info`, based on the `client` and `movie_category` tables, which contains only the client's last name and the *name* of the client's favorite category. Write this such that the name of the second column in this view is `fave_category`.

Problem 3-3

Write a query that **uses JUST** the `counter_client_info` view to project all of the columns all of the rows of the `counter_client_info` view, displaying the rows in order of client last name.

Then write another query that **uses JUST** the `counter_client_info` view, but this time projecting the `fave_category` column first and then the client last name column, now displaying the rows in order of the name of the client's favorite category.

Problem 3-4

Drop and create a view called `movie_list` of the `movie` and `movie_category` tables, containing only the category name, movie rating, and movie title for each movie.

Problem 3-5

Write a query that **uses JUST** the `movie_list` view to project all of the columns of all of the rows of the `movie_list` view, displaying the rows in order of category name, with a secondary ordering by movie rating, and a third-ordering by movie title.

Problem 3-6

Now, **using ONLY the view** `movie_list`, write a query projecting two columns: the name of a movie category, and the number of movies in that category, giving this second column the column heading `CATEGORY_QUANT`, and displaying the rows in order of decreasing number of movies.

Problem 3-7

Because there is often more than one way to write a SQL query to answer some questions, on some previous assignments you have been asked to use a particular SQL approach or feature rather than another, to get practice with that SQL approach or feature.

But, that might give the mistaken impression that, for example, for features such as joins and sub-selects, it is always a choice of one OR the other, when actually there are queries that can usefully involve both used together!

Write a `select` statement that will project the last names, favorite movie category **names**, and credit ratings for clients who have credit ratings higher than the average credit rating for all clients. (**NOTE**: I am not asking you to project the `client_fave_cat` --- I am asking you to project the **name** of the category corresponding to the `client_fave_cat`.)

HINTS:

- a single query can definitely use both equi-joins AND sub-selects
- a `select` clause can only project attributes from relation(s) in its corresponding `from` clause

Problem 3-8

Continuing with the theme that a query can involve BOTH joins AND sub-selects:

Write a `select` statement that will project the video ids, the title of the movie on that video, and the format of that video, for all videos that have never been rented.

Submit your files `325hw9.sql` and `325hw9-out.txt`.