CS 325 - Homework 5

Deadline

11:59 pm on **Monday, November 14, 2022**.

How to submit

Problem 1 should be saved as a .pdf and submitted to Canvas.

For Problem 2 onward:

Each time you wish to submit, within the directory `325hw5` on nrs-projects.humboldt.edu (and at the nrs-projects UNIX prompt, **NOT inside** `sqlplus`!) type:

```
~ma548/325submit
```

...to submit your current files for problems 2 and 3, using a homework number of **`52 and 53`**.

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

Setup for Problems 2 onward

Use `ssh` to connect to `nrs-projects.humboldt.edu`, and create, protect, and go to a directory named `325hw5` on nrs-projects:

```
mkdir 325hw5
chmod 700 325hw5
cd 325hw5
```

Put all of your files for this homework in this directory. (And it is from this directory that you should type `~ma548/325submit` to submit your files each time you want to submit the work you have done so far.)

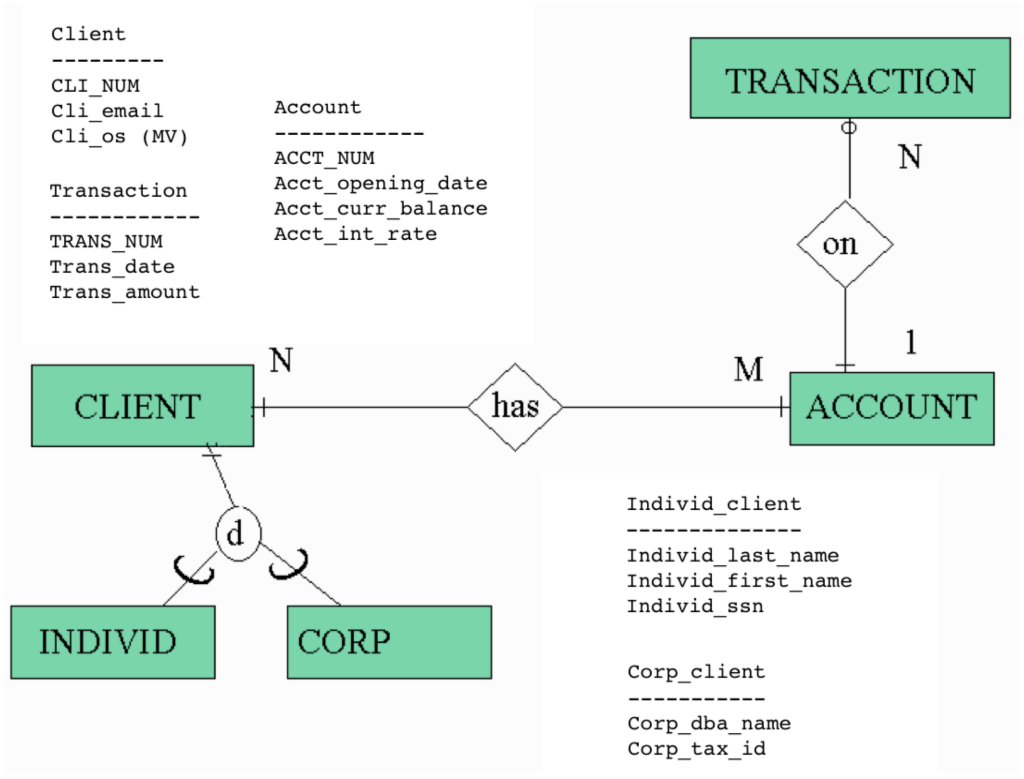
Problem 1

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.)

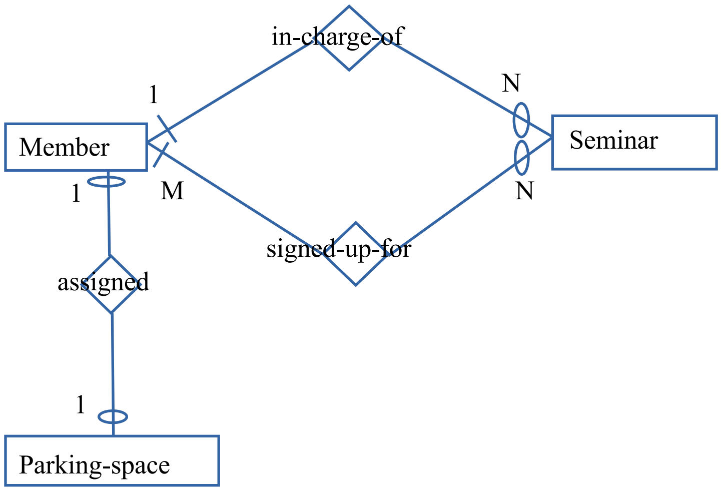
Consider the following ER models. Convert each into an appropriate corresponding (partial) design/schema, using the conversion rules discussed in lecture. Your resulting database designs/schemas needs to meet the following requirements:

* 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 1-1's model:



Problem 1-2's model:



```
Member        Seminar      Parking-space
-----------   ----------   -------------
Last-name     SEM-NUM      PARK-ID-NUM
MEM-NUM       Title        Garage-name
Email (MV)    Sem-date     Section-num
Date-joined   Time-begin   Space-num
              Time-end
```

Problem 2

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.

- Make a copy of `movies-pop.sql` in your `325hw5` directory -- one way to do so is using:

```
cp ~ma548/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 `325hw5-Problem2.sql` and include the following within one or more SQL **comments**:

- your name
- `CS 325 - Homework 5 - Problem 2`
- the date this file was last modified

Then:

- use `spool` to start writing the results for this script's actions into a file `325hw5-Problem2-out.txt`
- put in a `prompt` command printing `Homework 5 Problem 2`
- 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 `325hw5-Problem2.sql`, add in SQL statements for the following, **PRECEDING** EACH with a SQL*Plus `prompt` command noting what problem part it is for.

Problem 2-1

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 2-2

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 2-3

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 2-4

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 2-5

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 2-6

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

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.

(These tables should **still exist** in your database from Homework 4, so you should **not** need to re-run `movies-create.sql` unless you have been experimenting with insertions or other table modifications.)

Use `nano` (or `vi` or `emacs`) to create a file named `325hw5-Problem3.sql`:

```
nano 325hw5-Problem3.sql
```

While within `nano` (or whatever), type in the following within one or more SQL **comments**:

- your name
- `CS 325 - Homework 5 - Problem 3`
- the date this file was last modified

NOTE!!! READ THIS!!!

Now, within your file `325hw5-Problem3.sql`, add in SQL statements for the following, **PRECEDING** EACH ***EXCEPT* FOR PROBLEM 3-1** with a SQL*Plus `prompt` command noting what problem part it is for.

Problem 3-1

(This ONE problem does NOT need to be preceded by a `prompt` command, for reasons that will hopefully become clear...!)

Because this script experiments with `update` and `delete` statements, this script should start with a "fresh" set of table *contents* each time it runs.

- Make a copy of `movies-pop.sql` in your `325hw5` directory.
  - Note that one of several ways to get this is to copy it from my home directory on nrs-projects.

– For example, assuming that you are currently in your `325hw5` directory,

```
cp ~ma548/movies-pop.sql .
```

...should accomplish this. (REMEMBER the space and the . at the end!)

***BEFORE* the `spool` command in** `325hw5-Problem3.sql`, place a call executing `movies-pop.sql`. (that is, place the command you would type within `sqlplus` to run `movies-pop.sql` within your script `325hw5-Problem3.sql` BEFORE it starts spooling to `325hw5-Problem3-out.txt`).

Use `spool` to **NOW start writing the results for the REST of this script's actions into a file `325hw5-Problem3-out.txt`

- put in a `prompt` command printing `Homework 5 - 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 REST of the problems below BEFORE this `spool off` command!

Problem 3-2

Using `minus` appropriately, project the movie titles of all movies **minus** the movie titles that have been rented at least once, ordering the resulting rows in reverse alphabetical order by movie title. (To receive credit for this problem, you must appropriately use the `minus` operator.)

Note 1: Do you see that this is one of several ways to determine the titles of movies that have never been rented?

Note 2: There are several reasonable ways to write the second sub-query (the second operand of `minus`) for this problem. Pick your favorite!

Problem 3-3

Using `union` appropriately, project the video ids and vid_rental_prices of videos that have format HD-DVD (not regular DVD!) union'ed with the video ids and vid_rental_prices of videos that have never been rented, ordering the resulting rows in reverse order of `vid_rental_price`. (To receive credit for this problem, you must appropriately use the `union` operator.)

Problem 3-4

Write a query that shows, for all videos, how many times each has been rented. However, note the following important characteristics required of your result:

- it should project just two columns: the video id, and the number of times that video has been rented
- it needs to include rows for videos never rented, with a count of `0` for the number-of-times-rented (hint: `union` can be useful for this!)
- order the rows in reverse order of number of times rented, and for videos rented the same number of times, order them in order of video id.

Problem 3-5

Write a query projecting the client last names and credit ratings for all clients in order of credit rating.

Then, write an `update` command to increase the credit rating by 10% for those clients whose credit rating is both less than 4.0 and greater than the average client credit rating.

Finally, repeat the query projecting the client last names and credit ratings for all clients in order of credit rating.

Problem 3-6

First, write a query that will simply project how many rows are currently in the `video` table.

Then, write a single `delete` command that will delete all rows from the `video` table for videos that have never been rented.

Then, follow that with a query showing all of the contents of the `video` table, displaying the rows in order of `vid_id`.

Finally, now do a SQL `rollback;` command (to undo the database contents changes).

Submit your `sql` and `txt files`.