CIS 315 - Homework 5

Deadline:

1:00 pm (beginning of lab) on Wednesday, October 27th

How to submit:

When you are ready, within the directory `315hw5` on `nrs-labs.humboldt.edu` (and at the nrs-labs UNIX prompt, NOT inside `sqlplus`!) type:

```
~st10/315submit
```

...to submit these `.sql`, `.txt`, and `.pdf` files, using a homework number of `5`. (**Make sure** that you see the names of all of the files you wished to submit!)

Purpose:

To get experience converting an ERD into a database design/schema, and to get additional practice with nested selects/sub-selects as well as practice with `order by` clauses, `group by` clauses, and `having` clauses.

Additional notes:

- You are required to use the HSU Oracle `student` database for Problem 2.
- An example `hw5-results.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 2. If your `hw5-results.txt` matches this posted one, that doesn't guarantee that you wrote appropriate queries, but it is an encouraging sign...
  - Be careful, though -- in some cases, there is more than one way to write a query that gives the same results, but for some of this homework's problems you are required to write a query that gets those results using certain query approaches (and perhaps deliberately not using other query approaches). (And, of course, your queries always need to meet course SQL style guidelines.) You need to meet those specifications, meet those style guidelines, *and* get the desired results for such problems.

Problem 1:

NOTE: THIS PROBLEM DOES NOT USE ORACLE AT ALL!!

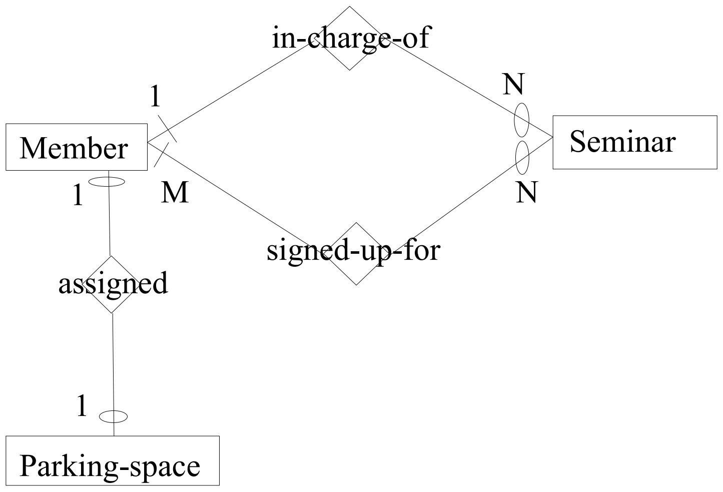
For this part, you will be converting a database model into a (partial) database design/schema. (Why partial? Because, for this assignment, we are not including domains or business rules, which are part of a database design/schema, also.)

Place your answers for this part in a file named `hw5-1.txt` within your `315hw5` directory.

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

* 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.)

The 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 homework again uses the tables created by the SQL script `hw3-setup.sql`. As a reminder, it created and populated a collection of tables that 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

Use `nano` (or `vi` or `emacs`) to create a file named `hw5-2.sql`:

```
nano hw5-2.sql
```

While within `nano` (or `vi` or `emacs`), type in the following:

- your name within a SQL comment
- `315 Homework 5 - Problem 2` as a SQL comment
- the date this file was last modified as a SQL comment
- use `spool` to start writing the results for this script's actions into a file `hw5-results.txt`
- 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 add in SQL statements for the following, **preceding each with a `prompt` command noting what question it is for**. (please notice, we're using a `prompt` command **INSTEAD** of a comment, now! I don't mind if you ALSO put a comment, but the `prompt` command should have a nice effect in your spooled output.)

Problem 2-1

Perform a relational selection of the rows of the `client` table, displaying the resulting rows in **increasing** order of client credit rating.

Then, perform another relational selection of the rows of the `client` table, but now displaying the resulting rows in **decreasing** order of client credit rating.

Problem 2-2

You might remember that, on Homework 5, Problem 3-8 asked you to write a query which projects one column in its result: this column, with heading "Rating: Movie", shows, for each movie, the rating for that movie, then a colon and a space, and then the title of that movie.

Rewrite this query so that, now, its results are ordered by increasing order of movie rating, and for rows with the same movie rating, they should be ordered by increasing order of movie title.

Problem 2-3

First, perform a projection of the *name* of a movie's category, the movie title, and the movie rating, for all movies, displaying the resulting rows in order of movie category *code*, and for movies with the same movie category code, in (secondary) order of movie rating, and for movies with the same category code and movie rating, in (tertiary) order of movie title.

Then, perform a projection of the *name* of a movie's category, the movie title, and the movie rating, for all movies, displaying the resulting rows in order of movie rating, and for movies with the same rating, in *reverse* alphabetical order of move category name, and for movies with the same rating and category name, in order of movie_title.

Problem 2-4

Project the video ID only of all rented videos that were returned AFTER the date that they were due.

Problem 2-5

Using a nested select statement, and using NO join or product operations, project the movie numbers only of videos involved in rentals that were returned AFTER the date that they were due. Do not worry, in this case, about whether or not there are duplicate rows in the result.

Problem 2-6

Using a nested select statement, and using NO join or product operations, project the titles and ratings of movies that have videos that were involved in rentals that were returned AFTER the date that they were due. Display the resulting rows in order of movie rating, and, for rows with the same movie rating, in secondary order by movie title.

Problem 2-7

Project the client's last name, telephone number, and credit rating for clients whose credit rating is equal to or higher than the average client credit rating, displaying the resulting rows in reverse order of credit rating.

Problem 2-8

For videos that have never been rented, project the video id's, the title of the movie on that video, and the format of that video, displaying the resulting rows in order of video format, and for rows with the same video format, in order of movie title.

Problem 2-9

From the video table, for each video format, project the video format, the number of videos with that format, and the average video rental price for videos with that format. (Do not worry about the ugly formatting of the average video rental price.)

Problem 2-10

Rewrite Problem 2-9's query, this time displaying the resulting rows in increasing order of the number of videos with that format.

Problem 2-11

From the video table, for each video rental price, project the video rental price and the number of videos with that rental price, displaying the resulting rows in decreasing order of video rental price.

Problem 2-12

Rewrite Problem 2-11's query, except this time include only those video rental prices that are the prices of at least 5 (5 or more) videos. (That is, project the video rental price and number of videos with that rental price ONLY for video rental prices that are the prices of 5 or more videos.)

Problem 2-13

From the movie and movie_category tables, for each movie category, project the movie category *name*, and the number of *movies* in that category. Display the resulting rows in reverse order of the number of movies in each category(that is, display the row with the category with the most movies first).

Problem 2-14

For each movie category, now project the movie category *name*, and the number of *videos* of movies in that category. (Be sure you are clear: in Problem 2-13, you are projecting the number of *movies* in each category; in this problem, you are projecting the number of *videos* of movies in each category.) Display the resulting rows in reverse order of the number of videos in each category(that is, display the row with the category with the most videos first).

Problem 2-15

Consider your query from Problem 2-14. You decide that you are really only interested in the movie categories for which there are fewer than 7 videos in that category. For this problem, project the movie category *name* and the number of videos of movies in that category only for movie categories with fewer than 7 videos. Display the resulting rows in reverse order of the number of videos in each category(from highest to lowest).

Problem 2-16

Project the average number of videos per video rental price. (This is a single value, note.) (Hint: use an aggregate function with an aggregate function as its argument...)

Problem 2-17

You decide that you are really only interested in the video rental prices for which the number of videos per video rental price is **above average** for such number of videos. Now project the video rental price and the number of videos with that video rental price only for video rental prices whose number of videos is above average for the number of videos.

(Hint: it is indeed the case that a `having` clause *can* have a sub-select/nested select...)

Problem 2-18

There can be more than one video of a given movie. One might want to know how many times a *movie* has been rented (how many times any of its videos have been rented).

Write a query that projects the movie title and the number of rentals of videos of that movie, displaying the resulting rows in decreasing order of number of rentals, and for movies with the same number of rentals, displaying them in order of movie_title.

(Note: for now, do not include movies whose videos have never been rented in your result. This is actually not asking you to do anything extra -- if you take the expected approach to this problem, such movies just don't show up in the results. To include movies whose videos have never been rented actually takes more effort, and we'll be discussing approaches to this a bit later.)

When you think the results of all of these queries look correct, this would also be a good time to look at the contents of `hw5-results.txt` -- at the nrs-labs prompt (the UNIX level, NOT in `sqlplus`!), type:

```
more hw5-results.txt
```

You should see that `hw5-results.txt` contains the query results you just saw within `sqlplus`.

When you are satisfied with these, then `hw5-2.sql` and `hw5-results.txt` are ready to submit.