# CS 325 - Homework 7

## Deadline

**11:59 pm** on **Thursday, November 31, 2024**.

## Purpose

To continue reading and thinking about converting ER models into database designs, to get some practice converting an ER model into a database designs, and to get more practice writing SQL statements, including those making use of set-theoretic operators, an `update` statement, and a `delete` statement.

## How to submit

Zip the 325hw7 folder (contains `325hw7-db-design.txt, 325hw7.sql, 325hw7-out.txt`), submit the 325hw7.zip on Canvas.

## 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 2** of this homework.
- **DB Reading Packet 7** and **SQL Reading Packet 6**, on the course Canvas site, along with the posted slides from the course Canvas site, are useful references for this homework.
- 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 `325hw7-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 2. If your `325hw7-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 course Canvas site, under "References", there are now some evolving lists of course style standards posted.

## Problem 1

## Setup

Use `ssh` to connect to `nrs-projects.humboldt.edu`, and create, protect, and go to a directory named `325hw7` on nrs-projects:

```
mkdir 325hw7
chmod 700 325hw7
cd 325hw7
```

Put all your files for this homework in this directory.

## Problem 1

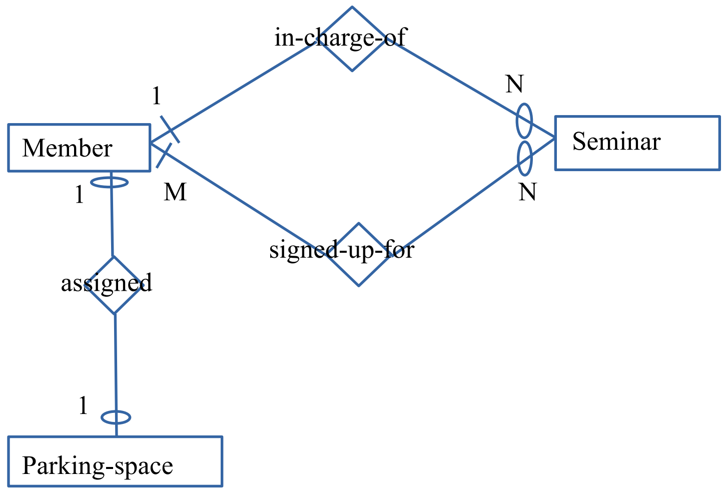
Place your answer for this problem into file `325hw7-db-design.txt`

For this problem, 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.)

Consider the following ER model. Convert it into an appropriate corresponding (partial) design/schema, using the conversion rules discussed in lecture. Your resulting database design/schema 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'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
```

Submit your file `325hw7-db-design.txt`.

## 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.

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

Use `vim` to create a file named `325hw7.sql`:

```
vim 325hw7.sql
```

While within `vim`, type in the following within one or more SQL **comments**:

- your name
- `CS 325 - Homework 7 - Problem 2`
- the date this file was last modified

### *NOTE!!! READ THIS!!!*

Now, within your file `325hw7.sql`, add in SQL statements for the following, **PRECEDING** EACH ***EXCEPT* FOR PROBLEM 2-1** with a SQL*Plus `prompt` command noting what problem part it is for.

### *Problem 2-1*

(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 `325hw7` directory.
  - Note that one of several ways to get this is to copy it from your local machine into nrs-projects using **sftp** and **put** command.
- ***BEFORE* the** `spool` **command in** `325hw7.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 `325hw7.sql` BEFORE it starts spooling to `325hw7-out.txt`)

- (why? because I really don't need to see all of the row-inserted feedbacks in your results file... 8-) )
- IF YOU'D LIKE TO TRY SOMETHING NEAT: a very useful SQL*Plus command:

```
set termout off
@ movies-pop
set termout on
```

This turns terminal output off, then you run the commands you don't really want to see all the output from (here resetting up tables in a script you KNOW is fine), then you turn terminal input on again.

- SO if you'd also like to use this pair of statements around the execution of `movies-pop.sql`, feel free!

- use `spool` to **NOW** start writing the results for the REST of this script's actions into a file `325hw7-out.txt`
- put in a `prompt` command printing `Homework 7 - 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 REST of the problems below BEFORE this `spool off` command!

## *Problem 2-2*

Using `intersect` appropriately, project the movie titles of movies that have rating G **intersected** with the movie titles of movies that are available on videos with the format DVD. (To receive credit for this problem, you must appropriately use the `intersect` operator.)

## *Problem 2-3*

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 2-4*

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 2-5*

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 2-6*

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 2-7*

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`.

Fibally, now do a SQL `rollback;` command (to undo the database contents changes from Problems 2-6 and 2-7, so they do not possibly lead to confusion on Homework 9...!)

Submit your files `325hw7.sql` and `325hw7-out.txt`.