CS 325 - Homework 8

Deadline:

Problem 1 -- answering reading questions on Moodle for Reading Packet 9 -- needs to be completed by 10:45 am on Tuesday, November 15.

The remaining problems are due by **11:59 pm** on **Friday, November 18, 2016**

How to submit:

For Problem 2 onward:

Each time you wish to submit, within the directory `325hw8` 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 `8`.

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

Purpose:

To get experience converting an ER model involving supertype/subtype entity classes into a (partial) database design/schema, to practice with SQL `union`, `intersect`, and `minus` operators, to practice with SQL `update` and `delete` statements, to practice a little bit with SQL `commit` and `rollback` statements, and to practice with Oracle SQL sequences and SQL views.

Additional notes:

- SQL Reading Packets 6 and 7 and "regular" Reading Packet 8 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.
- Note that it is considered poor style to deliberately include a Cartesian product without adding the appropriate number of join conditions to make it an equi-join.

  (That is, if you have *n* tables in a `from` clause, you are expected to also have (*n*-1) join conditions.)
- You are expected to also follow the SQL style standards noted in previous homework handouts and mentioned in class.
- You are required to use the HSU Oracle `student` database for this homework.
- Feel free to add additional `prompt` commands to your SQL scripts as desired to enhance the readability of the resulting spooled output.
- An example `hw8-3-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 `hw8-3-out.txt` matches this posted one, that doesn't guarantee that you wrote appropriate queries, but it is an encouraging sign.

– (I added a few extra `prompt` commands near the beginning of this script to output some blank lines for slightly-prettier output.)

Problem 1

Have to correctly answer the Reading Questions for Reading Packet 9 - Transaction Management - part 1, on the course Moodle site, by 10:45 am on Tuesday, November 15.

Problem 2

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

Use `ssh` to connect to `nrs-projects.humboldt.edu`, and create a directory named `325hw8` on nrs-projects:

```
mkdir 325hw8
```

...and change this directory's permissions so that only you can read it:

```
chmod 700 325hw8
```

...and change your current directory to that directory (go to that new directory) to do this problem:

```
cd 325hw8
```

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.)

Now, use `nano` (or `vi` or `emacs`) to create a file named `hw8-2.txt`:

```
nano hw8-2.txt
```

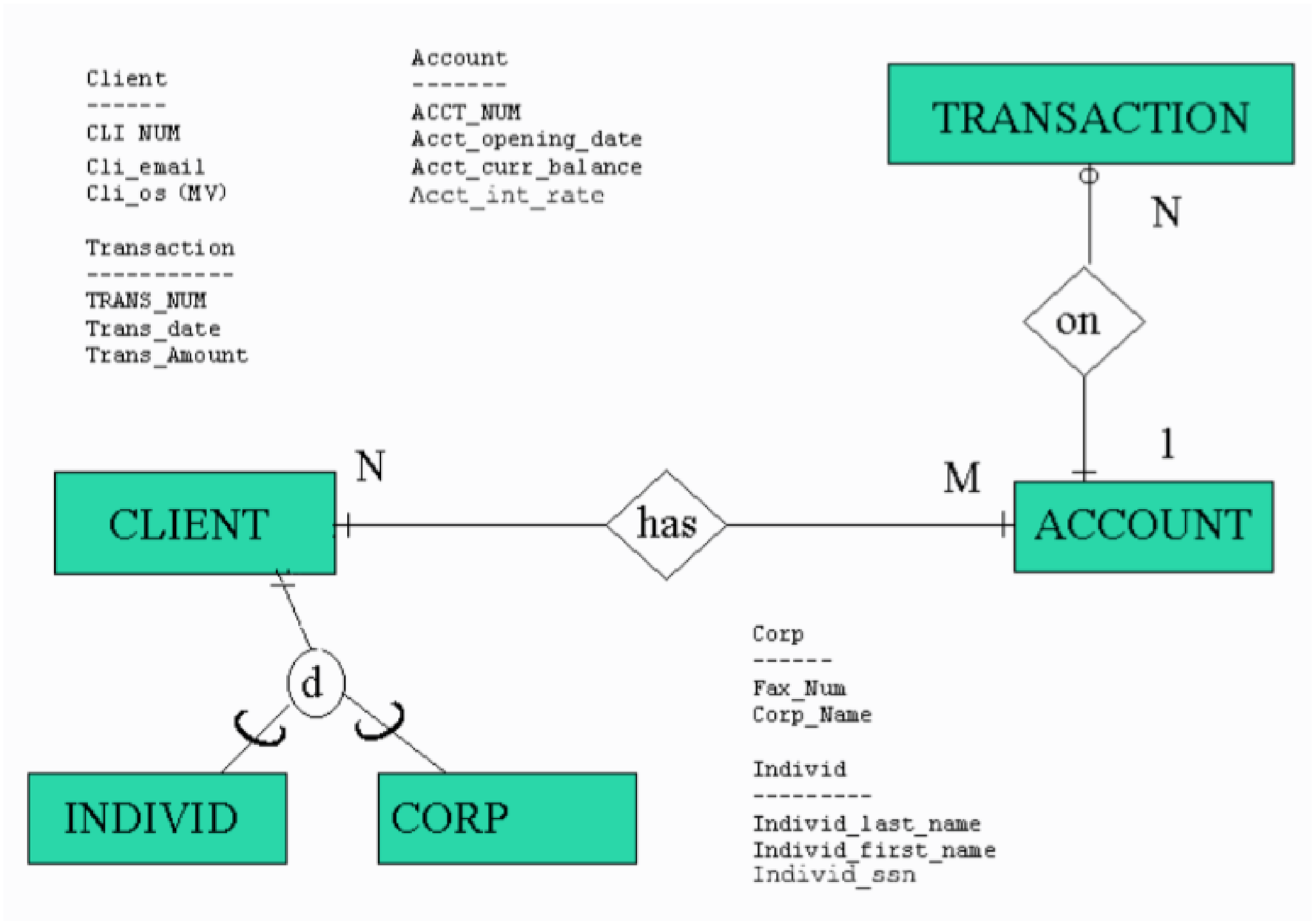
Place your answers for this problem into file `hw8-2.txt`

For this problem, you will be converting two database models 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.)

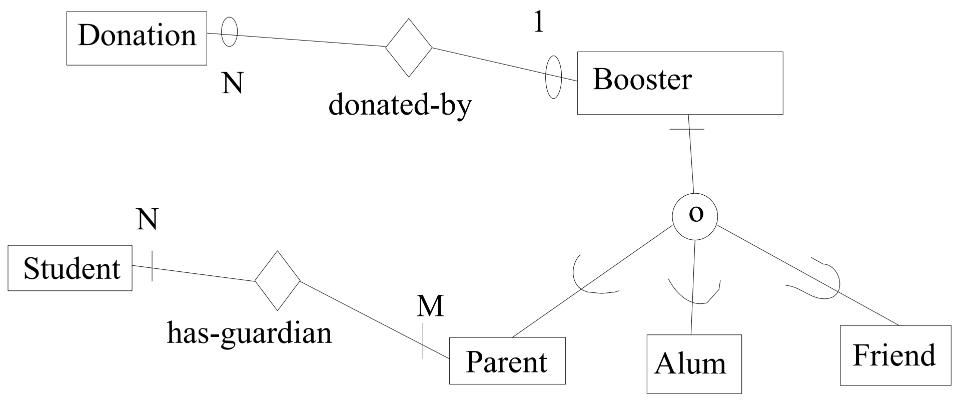
Consider the following ER models. Convert each into an appropriate corresponding (partial) design/schema, using the conversion rules discussed in lecture. Your resulting database designs/schemas needs to meet the following requirements:

* 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.)

2 part a's model:



2 part b'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

Now, you are back to using Oracle and SQL again.

This homework again uses the tables created by the SQL script `hw4-create.sql` and populated by `hw4-pop.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
```

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 `hw4-create.sql` or `hw4-pop.sql` unless you have been experimenting with insertions or other table modifications.)

Use `nano` (or `vi` or `emacs`) to create a file named `hw8-3.sql`:

```
nano hw8-3.sql
```

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

- your name within a SQL comment
- `CS 325 Homework 8 - Problem 3` within a SQL comment
- the date this file was last modified within a SQL comment

NOTE!!! READ THIS!!!

Now, within your file `hw8-3.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. (I am going to try to also have you practice with `commit;` and `rollback;` below, but I still want to play it safe a bit as you practice with modifying tables.)

- Make a copy of `hw4-pop.sql` in your `325hw8` 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 `325hw8` directory,

    ```
    cp ~st10/hw4-pop.sql .
    ```

    ...should accomplish this. (NOTE the space and the . at the end -- those are important! They say you are copying the file into your current directory, since `.` in Unix is a nickname for your current directory.)

- ***BEFORE* the `spool` command in** `hw8-3.sql`, place a call executing `hw4-pop.sql`. (That is, place the command you would type within `sqlplus` to run `hw4-pop.sql` within your script `hw8-3.sql` BEFORE it starts spooling to `hw8-3-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: Sky McKinley showed me a very useful SQL*Plus command I had not seen before:

    ```
    set termout off

    @ hw4-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 `hw4-pop.sql`, feel free!
- use `spool` to NOW start writing the results for the REST of this script's actions into a file `hw8-3-out.txt`
- put in a `prompt` command printing `Homework 8 - 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

(NOW start preceding each problem with a `prompt` command.)

Using `intersect` appropriately, project just the movie numbers of movies that have rating G intersected with the movie numbers of movies that are available on videos with the format DVD. (Yes, I know that you can do this with a Boolean `and` operator. However, you are not permitted to do so here, so that you can practice using `intersect`.)

Problem 3-3

Make a copy of your query for Problem 3-2, and rewrite it to now projects just the **titles** of movies that have rating G intersected with the titles of movies that are available on videos with the format DVD. You should still use `intersect`, but now the only Boolean `and`s that are permitted are to allow the join between the two tables in one of the sub-queries along with the selection criteria of which movies are available on the format DVD.

Problem 3-4

You've written queries regarding what videos have never been rented -- write a query that gives the titles of *movies* that have never been rented (movies for which none of their videos have been rented), using the `minus` operator appropriately in your answer.

(Hint: How can you simply project all movie titles? How can you project the titles of all movies that have been rented? From those, how can you use `minus` to get titles of movies that have never been rented?)

Problem 3-5

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. (Yes, I know that you can do this with a Boolean `or` operator. However, you are not permitted to do so here, so that you can practice using `union`.) Display the resulting rows in reverse order of `vid_rental_price`.

Problem 3-6

Using `minus` appropriately, show the client numbers of clients with credit rating higher than 3 minus the client numbers of clients with still-out/unreturned rentals.

(Thus, the result should be the client numbers of clients with credit rating higher than 3 who currently have no still-out/unreturned rentals.)

(Yes, I know that you can do this with `not exists`. However, you are not permitted to do so here, so that you can practice using `minus`.)

Problem 3-7

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 3-8

Commit the current state of your database -- we are about to make some hopefully-temporary changes over the next few problems.

Next, write a query projecting the client last names and credit ratings for all clients.

Then, write an `update` command to change the credit rating for client `'4444'` to `3.8`.

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

Problem 3-9

Write a query projecting the video ID's and video rental prices of non-DVD videos.

Then, write an `update` statement to decrease the video rental prices for non-DVD videos by 18%.

Finally, repeat the query projecting the video ID's and video rental prices of just those non-DVD videos.

Problem 3-10

What if it turned out that all of the rentals for client number `'3333'` were erroneous, and needed to be deleted from table `rental`?

First, write a query showing all of the contents of the `rental` table, displaying the rows in order of client number.

Then, write a single `delete` command that will indeed delete all of the rentals for client number `'3333'`.

Finally, repeat the query showing all of the contents of the `rental` table, again displaying the rows in order of client number.

Problem 3-11

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.

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

And, now that we are done with our `update` and `delete` practice, roll back the database to its state at the beginning of Problem 3-8.

Problem 3-12

Drop and create a view called `mini_action` which contains the columns `movie_num`, `movie_title`, and `movie_rating` from the `movie` table for rows with `category_code` of `200`. Follow that with a query showing all of the contents of this view, displaying the rows in order of `movie_title`.

Problem 3-13

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.

Follow that with a query showing all of the contents of this view, displaying the rows in order of movie rating, with a secondary ordering by movie title.

Problem 3-14

Drop and create a new table (not already used in `hw5-setup.sql` nor in lab) that has at least **three** columns.

Then, drop and create a **sequence** suitable for use in setting the primary key of this new table. Have your sequence start at a value other than 1.

Problem 3-15

Insert at least 3 rows into your table from Problem 3-14, using your **sequence** from Problem 3-14 to set each new row's primary key

Then, write a query showing the contents of this table.

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 `hw8-3-out.txt` -- at the nrs-projects prompt (the UNIX level, NOT in `sqlplus`!), type:

```
more hw8-3-out.txt
```

You should see that `hw8-3-out.txt` contains the query results you just saw within `sqlplus`.

When you are satisfied with these, then `hw8-3.sql` and `hw8-3-out.txt` are completed.