CS 325 - Homework 4

Deadline:

11:59 pm on **Friday, October 7, 2016**

How to submit:

Each time you wish to submit, within the directory `325hw4` 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 `4`.

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

Purpose:

To practice with more "basic" SQL `select` statements, and to get more practice with database modeling (especially involving supertype/subtype entity classes).

Additional notes:

- You are required to use the HSU Oracle `student` database for this homework.
- SQL Reading Packet 3 and "regular" Reading Packet 5, on the course Moodle site, are useful references for this homework.
- An example `hw4-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 1. If your `hw4-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.)
- Feel free to add additional `prompt` commands to your SQL scripts as desired to enhance the readability of the resulting spooled output.
- NOTE the following course style standards for SQL `select` statements:
  - In a SQL script, put a blank line BEFORE and AFTER each `select` statement, for better readability.
  - A `select` statement's `from` clause should ALWAYS start on a new line.
  - If a `select` statement has a `where` clause, it should always start on a new line.
  - If a clause is longer than one line, INDENT the continuation on the next by at least three spaces (so it is clear which clause it "belongs" to).

Setup for Problem 1

Use `ssh` to connect to `nrs-projects.humboldt.edu`, and create a directory named `325hw4` on nrs-projects:

```
mkdir 325hw4
```

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

```
chmod 700 325hw4
```

...and change your current directory to that directory (go to that new directory) to do Problem 3 of this homework:

```
cd 325hw4
```

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 1

On the public course web page, along with this assignment, you will find a SQL script `hw4-create.sql`. It creates 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
```

(Think of it as a quaint, historical scenario... 8-/)

For your convenience and reference, a handout of these relation structures is posted along with this homework handout.

And, SQL script `hw4-pop.sql` initially populates these tables (because it is better style to separate table creation and initial table population).

In your `325hw4` directory on nrs-projects, create a copy of the scripts `hw4-create.sql` and `hw4-pop.sql`, either by pasting them from the public course web page, or by using these commands at the nrs-projects UNIX prompt while in your directory `325hw4`:

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

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

Run these SQL scripts `hw4-create.sql` and `hw4-pop.sql` in `sqlplus`; these 5 tables should be created, and then initially populated. Look them over; check out their contents.

Then, use `nano` (or `vi` or `emacs`) to create a file named `hw4.sql`:

```
nano hw4.sql
```

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

- your name within a SQL comment
- `CS 325 Homework 4` within a SQL comment
- the date this file was last modified within a SQL comment
- use `spool` to start writing the results for this script's actions into a file `hw4-out.txt`
- put in a `prompt` command printing `Homework 4 Problem 1`
- 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!

NOTE!!! READ THIS!!!

Now, within your file `hw4.sql`, add in SQL statements for the following, **PRECEDING** EACH with a SQL*Plus `prompt` command noting what problem part it is for.

Problem 1-1

Perform a **relational selection** of rows of the `client` table for clients with a client credit rating less than 3.4.

Problem 1-2

Perform a **"pure"/"true" relational projection** of the movie rating and year released columns from the `movie` table.

Problem 1-3

Perform an **equi-join** of the `client` and `movie_category` tables. (Consider: what would be a suitable join condition for an equi-join between these two tables?)

Problem 1-4

Note that, in this scenario, a rental's date returned attribute is empty/null until the rented video has been returned.

Project just the video IDs and their date due for rentals that have not yet been returned.

Problem 1-5

Project just the video IDs, formats, and rental prices for videos that do not have the format DVD.

Problem 1-6

Project just the movie category NAME (**not** the movie category code!), client's last name, and client's credit rating from the equi-join of the `movie_category` and `client` tables.

Problem 1-7

Perform a relational selection of videos with a purchase date between July 15, 2008 and December 1, 2011, inclusive. For full credit, appropriately use the `between` operator in your query.

Problem 1-8

Perform a relational selection of videos which have a rental price greater than or equal to $3.99 and have a purchase date on or after January 1, 2011.

Problem 1-9

Project only the movie title and the movie rating for all movies containing the string `'the'` anywhere in their title (where the case IS significant -- only all-lowercase `'the'` is desired). For full credit, appropriately use the `like` operator in your query.

Problem 1-10

Project only the video ID, format, and what the rental price would be if it were decreased by 20%, for videos whose format is not DVD. Use a column alias so that the third column, the rental price if decreased by 20%, appears in the result with the column heading `PRICE_IF_REDUCED`.

(That is, you will have three columns in your result -- yes, you should project a computed column for that 3rd column. Remember: you are not actually changing the video table, you are simply seeing what the rental prices for the selected rows would look like IF such a change were made.)

Problem 1-11

Project only the movie title and movie rating for all movies with ratings of PG-13 or R or A only. For full credit, appropriately use the `in` operator in your query.

Problem 1-12

Project the current total number of videos and the average rental price for a video. (Do not worry about the many decimal places in the result!) Use column aliases so that the result's column headings appear as "`Num Vids`" and "`Avg Rental`" (with the blanks and mixed case as shown).

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 `hw4-out.txt` -- at the nrs-projects prompt (the UNIX level, NOT in `sqlplus`!), type:

```
more hw4-out.txt
```

You should see that `hw4-out.txt` contains the query results you just saw within `sqlplus`.

When you are satisfied with these, then `hw4.sql` and `hw4-out.txt` are completed.

Problem 2

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

For this part, you will be creating an entity-relationship diagram including entity attribute lists for a scenario.

How are you going to create this? You have several choices:

- you may use Word's or OpenOffice's or NeoOffice's or LibreOffice's drawing tools to draw it
  - **NOTE**: several students report liking `draw.io`, a browser-based drawing tool, available at https://www.draw.io/
- you can use other drawing software if you have it,
- you can draw it by hand,
- you can produce part of it using software, and then finish it by hand; etc.

Then, you need to convert your result into PDF format using some means, into a file named `hw4-erd.pdf`;

- you might be using software with an option to save as PDF,
- you can scan it and save it as a PDF,
- you can take a (legible!) photo of it and save it as or convert it to PDF, etc.

You will submit your resulting `hw4-erd.pdf` for this part. (Make sure to put it in the same directory as your `.sql` and `.txt` files on `nrs-projects.humboldt.edu`.)

The Scenario:

Consider the following. In a particular on-line financial institution, clients have a unique client number, and the institution keeps their one preferred e-mail address, and whichever operating systems (yes, there might be more than one) that they like to use to access their account(s). Clients may be individuals or corporations -- individual clients also need their last name, first name, and social security number on-record, while corporate clients need their one preferred doing-business-as (DBA) name and their one preferred fax number on-record.

A client can have 1 or more accounts at this institution; they must have at least one account. Accounts are related to at least one client, but may be related to more than one client as well (that is, joint or multiple-owner accounts are possible); each account has a unique account number, the date it was opened, its current balance, and its current interest rate. An account is identified by its account number. Finally, transactions are made on accounts, and the institution cares about the following information from each transaction: the transaction number, the date of the transaction, and the amount of the transaction (which is a single monetary amount, and which may be positive or negative). Each transaction is uniquely identified by a transaction number, and each transaction is only on a single account. An account may have many transactions, but it does not have to have any transactions.

Your Task:

Develop and draw an appropriate entity-relationship diagram including entity attribute lists for the above scenario. Create a PDF of your final entity-relationship diagram including entity attribute lists named `hw4-erd.pdf`, transfer a copy of `hw4-erd.pdf` to `nrs-projects.humboldt.edu`, and submit it using `~st10/325submit` from there.

Be sure to meet the following style standards:

- each **entity class** is represented by a **rectangle**, with the name of the entity class within the rectangle.
  - Remember: a particular entity class rectangle appears ONLY once within an ERD!

  - (If an entity class is involved in multiple relationships, you just have multiple relationship lines connected to that entity class rectangle.)
- each **relationship** is represented by a diamond on a line connecting the associated entity classes, labeled on or above or below the diamond with a descriptive (and preferably unique) name for that relationship.
  - (that is, you draw a line between related entity classes, and the diamond for the relationship appears on that line…)
- the **maximum** cardinality of each relationship must be depicted by writing the 1, M, or N, whichever is appropriate, near each end of the relationship line near the appropriate entity class rectangle (as depicted in the Reading Packet 4 - Entity-Relationship Modeling, Part 1, available from the course Moodle site, under "CS 325 Reading Packets".)
- the **minimum** cardinality of each relationship class must be depicted by drawing either an oval or a line, whichever is appropriate, on each end of the relationship line (near the entity class rectangle) (as also depicted in the Reading Packet 4 - Entity-Relationship Modeling, Part 1).
  - (Remember: you draw an oval on the relationship line near the entity class rectangle if the relationship is optional at that end, and you draw a line across the relationship line near the entity class rectangle if the relationship is mandatory on that end.)
- for **supertype/subtype** entity classes, use the notation illustrated in the Reading Packet 5 - Entity-Relationship Modeling, Part 2
  - there are lines between the supertype and the subtype entity classes leading to a circle
  - each line between this circle and a subtype entity class has a ⊂ on it (subset-ish symbol)
  - the line between this circle and the supertype entity class has the appropriate minimum cardinality marking to indicate whether a supertype entity instance must also be one of the subtype entity instances or not
  - the circle has within it either **d** (for **disjoint**), **o** (for **overlapping)**, or **u** (for **union**)
- remember to include as part of the ERD -- they can be at the bottom, to the side, or on the next page -- the entity attribute lists for each entity class, given as follows: write the name of each entity class, and underneath it put:
  - the attributes for that entity class
  - for any attribute that can be multi-valued, write `(MV)` after that attribute
  - write in all-uppercase the attribute(s), if any, that users in the scenario use to identify/distinguish between different instances of that entity (keeping in mind that **not** all entity classes have identifying attributes)
  - for the attribute lists for supertype and subtype entity classes, be careful that attributes all entity instances of those entity classes have are listed JUST in the supertype entity class's attribute list,

    and those particular to just one of the subtype entity classes are listed JUST in that subtype entity class's attribute list

  Do NOT write these lists of entity class attributes as relation structures -- entity classes are **NOT** relations! Each entity class will eventually be transformed into **one or more** relations during the database design phase, which FOLLOWS the modeling phase.

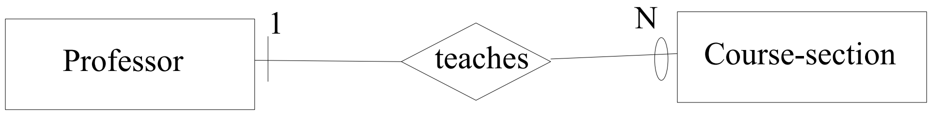
  Also remember: in the modeling phase, the attributes in these lists are attributes of that entity class alone

-- **they do NOT indicate relationships between entity classes**. The relationship lines in the ERD do that!

A useful rule of thumb: each attribute should only appear once, TOTAL, in this list of all entity class's attributes.

- Including a set of **business rules** is **not required** for this homework problem, unless you would like to to justify/explain some of your choices (for example, for whether attributes can be multi-valued).
  - Note that you may **not** change anything in the scenario, however.

For example, the following meets the above standards:



```
Professor                      Course-section
-----------------              ------------------
PROF_SSN                       Course_name
Prof_lname                     COURSE_ID
Prof_email (MV)                Course_length
```

In the above example, given the maximum and minimum cardinalities shown, a professor does not HAVE to teach any course-sections, but may teach many course-sections. A course-section MUST have an associated professor, and may have at most one professor associated with it -- that is, there is exactly one professor associated with each course-session.

Note that the Professor entity attributes do not include any indication of which courses that professor teaches, nor do the Course-section entity attributes include any indication of which professor teaches that course-section -- this is NOT an error. This is **desired** for the database modeling phase -- the relationship between professors and courses is shown at this phase by the line between their rectangles and by the minimum and maximum cardinalities shown on that line.