CIS 315 - Homework 4

Deadline:

1:00 pm (beginning of lab) on Wednesday, October 20th

How to submit:

When you are ready, within the directory `315hw4` 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 `4`. (**Make sure** that you see the names of all of the files you wished to submit!)

Purpose:

To get more practice with database modeling (especially involving supertype/subtype entity classes), to think a bit more about the superkey/minimal key/candidate key/primary key definitions, to practice normalizing sets of relations into 1NF, 2NF, and 3NF, and to write more SQL queries (including queries with nested selects/subselects).

Additional notes:

- You are required to use the HSU Oracle `student` database for Problem 3.
- An example `hw4-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 3. If your `hw4-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). You need to meet those specifications *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 creating an Entity-Relationship Diagram (ERD) for a scenario.

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 ERD for the above scenario. Create a PDF of your final ERD in a file named `hw4erd.pdf`, and somehow get a copy of that file to `nrs-labs.humboldt.edu`. (Make sure to put it in the same directory as your `.sql` and `.txt` files on `nrs-labs.humboldt.edu.)` Then submit it using `~st10/315submit` 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. There should be ONLY 1 rectangle per entity class! (You can have multiple relationship class lines connected to a single entity class rectangle.)
- each **relationship class** 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 class.
  - (that is, you draw a line between related entity classes, and the diamond for the relationship class appears on that line…)
- the **maximum** cardinality of each relationship class 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 Week 5 Lecture reading packet, the PDF "slightly different version of Week 5 lecture material").
- 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 depicted in the Week 5 Lecture reading packet, the PDF "slightly different version of Week 5 lecture material"). (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.)
- under the finished ERD, or on the next page, should be listed the name of each entity class, and underneath it:
  - the attributes for that entity class

– for any attribute that can be multi-valued, write `(MV)` after that attribute

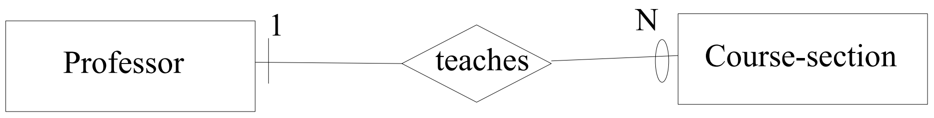
– underline **or** write in all capital letters the attribute(s), if any, that users in the scenario use to identify/distinguish between different instances of that entity (keeping in mind that not all entity classes have identifying attributes)

Do NOT write these lists of entity class attributes as relation structures -- these are **not** relations yet! They are **entities**, and each entity 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 alone -- they do NOT indicate relationships between entities. The relationship lines in the ERD do that! A useful rule of thumb: each attribute should only appear once, TOTAL, in this list of entities' 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 supertype/subtype entities, use the notation as illustrated in the Week 5 Lecture reading packet, the PDF "slightly different version of Week 5 lecture material".

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 proper for the database modeling phase --- the relationship between professors and courses is indicated at this phase by the line between their rectangles and by the minimum and maximum cardinalities shown on that line.

Problem 2:

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

If you haven't already done this as part of completing Problem 1, use `ssh` to connect to `nrs-labs.humboldt.edu`, and create a directory named `315hw4` on nrs-labs:

```
mkdir 315hw4
```

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

```
chmod 700 315hw4
```

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

```
cd 315hw4
```

Put all of your files for this homework in this directory. (And it is from this directory that you should type `~st10/315submit` to submit your files when you are done.)

Now, use `nano` (or `vi` or `emacs`) to create a file named `hw4-2.txt`:

```
nano hw4-2.txt
```

While within `nano` (or `vi` or `emacs`), type in your name, and then an answer for each of the following, preceding each answer with the number of the question being answered.

Consider the relation (from those relations set up and populated by `hw3_setup.sql`):

```
Video(VID_ID, vid_format, vid_purchase_date, vid_rental_price,
      movie_num)
        foreign key (movie_num) references movie
```

Problem 2-1

Is `(vid_id, vid_rental_price)` a **superkey** for this relation? Why or why not?

Problem 2-2

Is `(vid_format, vid_rental_price)` a **superkey** for this relation? Why or why not?

Problem 2-3

Is `(vid_id, vid_rental_price)` a **candidate key** for this relation? Why or why not?

Problem 2-4

Is `(vid_format, vid_rental_price)` a **candidate key** for this relation? Why or why not?

Consider the following relation structure and additional functional dependency information, and note that the attributes making up the primary key are capitalized. You should capitalize the primary key attributes in your answers as well.

And, as we have been doing in class, you are expected to indicate foreign keys using a SQL-style foreign key clause after your relation structure. For example,

```
MyTable1(THING_ID, Thing_name, Thing_color)

MyTable2(IT_ID, thing_id, it_size, it_shape)
   foreign key(thing_id) references MyTable1
```

Normalization Scenario 1

Here are some relations and additional functional dependency information:

```
Volunteer_Record(VOL_NUM, PROJ_NUM, vol_lname, vol_fname,
                 proj_name, hrs_worked, date_worked, task_type_code,
                 task_type_descr)
```

```
vol_num -> vol_lname, vol_fname
```

```
proj_num -> proj_name
```

```
task_type_code -> task_type_descr
```

Problem 2-5

Is this relation already in **1NF**? If not, do what is necessary so that it is; if so, leave it as-is.

Write the relation structure(s) for the 1NF relation(s) in either case. (ONLY make those changes, if needed, necessary to reach 1NF -- do **NOT** go beyond, yet!)

Remember to indicate foreign keys, if any, using the SQL-like foreign key notation given above.

Problem 2-6

Are your relation(s) resulting from Problem 2-5 in **2NF**? If not, do what is necessary so that is is/they are; if so, leave it/them as-is.

Write the relation structure(s) for the 2NF relation(s) in either case (**including** those that are unchanged, if any.) (ONLY make those changes, if needed, necessary to reach 2NF -- do **NOT** go beyond, yet!)

Again, remember to indicate any foreign keys.

Problem 2-7

Are your relation(s) resulting from Problem 2-6 in **3NF**? If not, do what is necessary so that is is/they are; if so, leave it/them as-is.

Write the relation structure(s) for the 3NF relation(s) in either case (**including** those that are unchanged, if any.) (ONLY make those changes, if needed, necessary to reach 3NF -- do **NOT** go beyond, yet!)

Again, remember to indicate any foreign keys.

Normalization Scenario 2

Now consider this second scenario -- note that this time, this is deliberately generic, so that I can see if you are getting the normalization steps without any extraneous semantic information accidentally getting in the way. All of the information that you need is below *if* you understand the normalization process as described in lecture.

```
R1 (A, B, C, D, e, f, g, h, i, j, k, l, m, n, o)
```

```
(e, f) -> (k, l)
(a, c) -> o
 h -> m
 d -> n
```

- NOTE --- "Whoops!" An "error" was made above --- even though, as written above, we know that it should be true that (a, b, c, d) --> g, it is actually the case that attribute g can have multiple values for a particular set of values (a, b, c, d). (That is, g is a multi-valued attribute.) (Note that it needs all 4 to determine those multiple values of g, however.) Rhetorical question (you do not have to answer in your homework file): what does that then imply about R1 above?

Problem 2-8

Is this relation already in **1NF**? If not, do what is necessary so that it is; if so, leave it as-is.

Write the relation structure(s) for the 1NF relation(s) in either case. (ONLY make those changes, if needed, necessary to reach 1NF -- do **NOT** go beyond, yet!)

Remember to indicate foreign keys, if any, using the SQL-like foreign key notation given above.

Problem 2-9

Are your relation(s) resulting from Problem 2-8 in **2NF**? If not, do what is necessary so that is is/they are; if so, leave it/them as-is.

Write the relation structure(s) for the 2NF relation(s) in either case (**including** those that are unchanged, if any.) (ONLY make those changes, if needed, necessary to reach 2NF -- do **NOT** go beyond, yet!)

Again, remember to indicate any foreign keys.

Problem 2-10

Are your relation(s) resulting from Problem 2-9 in **3NF**? If not, do what is necessary so that is is/they are; if so, leave it/them as-is.

Write the relation structure(s) for the 3NF relation(s) in either case (**including** those that are unchanged, if any.) (ONLY make those changes, if needed, necessary to reach 3NF -- do **NOT** go beyond, yet!)

Again, remember to indicate any foreign keys.

Problem 3:

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 `hw4-3.sql`:

```
nano hw4-3.sql
```

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

- your name within a SQL comment
- `315 Homework 4 - Problem 3` 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 `hw4-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 3-1

Using a **nested** select statement, **and using NO join or product operations**, project the **video id's only** of all videos that cost less to rent than the average price to rent a video.

Problem 3-2

Using a **nested** select statement, **and using NO join or product operations**, project the client numbers **only** for clients involved in rentals of videos that cost less to rent than the average price to rent a video. Do not worry, in this case, about whether or not there are duplicate rows in the result.

Problem 3-3

Using a **nested** select statement, **and using NO join or product operations**, project the last names and the client credit ratings of clients involved in rentals of videos that cost less to rent than the average price to rent a video. (If done correctly, there is no way that you can get duplicate rows in this query's results, even without the keyword that prevents them --- do you understand why? You do not have to answer as part of this homework, but you should know...)

Problem 3-4

Write a query which will project the **average rental price** and **number of such videos** for videos of Classic movies --- BUT! You must write this without explicitly using the category code for Classic movies; you must use the name `'Classic'` in your query instead. Rename the columns to be "Avg Cost - Classic" and "# Videos - Classic", respectively, using precisely these spaces and case.

Problem 3-5

Using a **join**, **and NOT using ANY nesting or sub-selects**, project the last names, first names, and date the video was due for clients who have ever rented the video with ID `'130012'`.

Problem 3-6

Using a **nested** select, **and using NO join or product operations**, project the last names and first names only (no date due this time) for clients who have ever rented the video with ID `'130012'`.

Problem 3-7

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`.)

Problem 3-8

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.

Problem 3-9

Using **EXISTS** or **NOT EXISTS**, write a query that will project the last names, phone numbers, and credit ratings of clients that have rented a video and not returned it yet (those who have an unreturned video rental, whether overdue or not). (This will not be accepted as correct unless it properly uses EXISTS or NOT EXISTS.)

Problem 3-10

Now, write a query that will do the same thing as Problem 3-8's query, **except** it should **not** use EXISTS or NOT EXISTS. (Any non-EXISTS approach that meets class style standards is fine.)

Problem 3-11

Using **EXISTS** or **NOT EXISTS**, write a query that will project the number of videos which have never been rented and their average rental price. (This will likewise not be accepted as correct unless it properly uses EXISTS or NOT EXISTS.)

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-results.txt` -- at the nrs-labs prompt (the UNIX level, NOT in `sqlplus`!), type:

```
more hw4-results.txt
```

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

When you are satisfied with these, then `hw4-3.sql` and `hw4-results.txt` are ready to submit.