CS 325 - Homework 6

Deadline:

11:59 pm on **Friday, October 28, 2016**

How to submit:

Each time you wish to submit, within the directory `325hw6` 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 `6`.

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

Purpose:

To practice normalizing sets of relations into 1NF, 2NF, and 3NF, to practice a bit with concatenation in SQL, and to practice some with nested selects/sub-selects using the predicates `EXISTS` and `NOT EXISTS`.

Additional notes:

- 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).
  - Nested selects (sub-selects) should be indented within their outer select, STILL with their `from` and `where` clauses each on their own line -- for example, both of the following meet class style standards:

```
select empl_last_name, salary
from   empl
where  dept_num IN
       (select dept_num
        from   dept
        where  dept_loc = 'Dallas');

select empl_last_name, salary
from   empl
where  dept_num IN (select dept_num
                    from   dept
                    where  dept_loc = 'Dallas');
```

- You are required to use the HSU Oracle `student` database for this homework.

- SQL Reading Packet 4 and "regular" Reading Packet 6 are useful references for this homework.
- An example `hw6-2-out.txt` has been posted along with this homework handout, to help you see if you are on the right track with your queries. If your `hw6-2-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.)
  - 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.
- Feel free to add additional `prompt` commands to your SQL scripts as desired to enhance the readability of the resulting spooled output.

Homework Setup:

Use `ssh` to connect to `nrs-projects.humboldt.edu`, and create a directory named `325hw6` on nrs-projects:

```
mkdir 325hw6
```

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

```
chmod 700 325hw6
```

...and change your current directory to that directory (go to that new directory) to do the problems for this homework:

```
cd 325hw6
```

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

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

Now, use `nano` (or `vi` or `emacs`) to create a file named `hw6-1.txt`:

```
nano hw6-1.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.

Normalization Scenario 1

This is a scenario in which volunteers work on different projects, and those within the scenario keep track of the work volunteers do for different projects. Here is a relation 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 1-1

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 SQL foreign key syntax directly under the relation structure that has that foreign key.

Problem 1-2

Are your relation(s) resulting from Problem 1-1 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 1-3

Are your relation(s) resulting from Problem 1-2 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 and in Reading Packet 6.

Here is a relation and additional functional dependency information:

```
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 1-4

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 SQL foreign key syntax directly under the relation structure that has that foreign key.

Problem 1-5

Are your relation(s) resulting from Problem 1-4 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 1-6

Are your relation(s) resulting from Problem 1-5 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 2

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 `hw6-2.sql`:

```
nano hw6-2.sql
```

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

- your name within a SQL comment
- `CS 325 Homework 6 - Problem 2` 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 `hw6-2-out.txt`
- put in a `prompt` command printing `Homework 6 - 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 problems below BEFORE this `spool off` command!

NOTE!!! READ THIS!!!

Now, within your file `hw6-2.sql`, add in SQL statements for the following, **PRECEDING** EACH with a SQL*Plus `prompt` command noting what problem part it is for.

Problem 2-1

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 2-2

Write another query that projects ONE column in its result: this column, with heading "`Movies`", shows, for each movie, the title for that movie, a space, and then, within a set of parentheses, the year that movie was released.

Problem 2-3

Using `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 any unreturned video rental -- we don't care whether it happens to be overdue or not). (This will not be accepted as correct unless it properly uses `EXISTS`.)

Problem 2-4

Using `NOT EXISTS`, write a query that will project the titles of movies that are not available in the format Blu-Ray. (This will not be accepted as correct unless it properly uses `NOT EXISTS`.)

Problem 2-5

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

Problem 2-6

Using `NOT EXISTS`, 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. (This will not be accepted as correct unless it properly uses `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 `hw6-2-out.txt` -- at the nrs-projects prompt (the UNIX level, NOT in `sqlplus`!), type:

```
more hw6-2-out.txt
```

You should see that `hw6-2-out.txt` contains the query results you just saw within `sqlplus`.

When you are satisfied with these, then `hw6-2.sql` and `hw6-2-out.txt` are completed.