CIS 315 - Homework 6

Deadline:

NOTE - UNUSUAL DEADLINE! (because I forgot to post this handout earlier, and because of the Veteran's Day Holiday)

11:59 pm on FRIDAY, November 12th

How to submit:

When you are ready, within the directory `315hw6` 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 `6`. (**Make sure** that you see the names of all of the files you wished to submit!)

Purpose:

Practice with SQL `union`, `intersect`, and `minus` operators, practice with SQL `update` and `delete` statements, and practice with SQL sequences

Additional notes:

- You are required to use the HSU Oracle `student` database 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.
- An example `hw6-results.txt` has been posted along with this homework handout, to help you see if you are on the right track with your SQL statements. If your `hw6-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). (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.

The Problems:

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 `hw6.sql`:

```
nano hw6.sql
```

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

- your name within a SQL comment
- `315 Homework 6` 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 `hw6-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 EXCEPT FOR PROBLEM 1 with a `prompt` command noting what question it is for**. (I don't mind if you ALSO put a comment, but the `prompt` command improves your spooled output's readability.)

Problem 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 tables each time it runs.

- Make a copy of `hw3-setup.sql` in your `315hw6` directory
- **BEFORE** the `spool` command in `hw6.sql`, place a call executing `hw3-setup.sql`. (That is, place the command you would type within `sqlplus` to run `hw3-setup.sql` within your script `hw6.sql` BEFORE it starts spooling to `hw6-results.txt`)
  - (why? because we really don't need to see all of the row-inserted feedbacks in our result file...)

Problem 2

Using `intersect` appropriately, project just the movie numbers of movies that have rating G and 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

Make a copy of your query for Problem 2, and rewrite it to now projects just the **titles** of movies that have rating G and are available on videos with the format DVD. You should still use `intersect`, but now the only Boolean `and` that is permitted is 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 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.

Problem 5

Using `union` appropriately, project the video id's and vid_rental_prices of videos that have format HD-DVD (not regular DVD!) **or** 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 6

Using `minus` appropriately, show 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`.)

Be careful --- it is easy to "subtract" the wrong clients here! You want to end up with those clients with credit rating higher than 3 who currently have *no* still-out/unreturned rentals --- SO which clients do you need to remove/subtract from the set of all of those with credit rating higher than 3?

Problem 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:

* in addition to the video number, it also should project the title of the movie on that video, and the video format, before projecting the number of times 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 as follows: in alphabetical order by `movie_title`; for videos of the same movie, in order of video format.

Problem 8

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 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 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 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`.

Problem 12

Drop and create a new table (not already used in `hw3-setup.sql` nor in lab) that:

* has at least 4 columns
* has at least one column (that is not the primary key) declared to be `not null`
* has at least one column whose domain is further restricted using a `check` clause
* has at least one column with a `default` clause

Problem 13

Drop and create a sequence suitable for use in setting the primary key of the table you created for Problem 12. Have it start at a value other than 1.

Problem 14

Insert at least 3 rows into your table from Problem 12, being sure to:

* use your sequence from Problem 13 to set each new row's primary key
* have at least one `insert` that does not explicitly insert a value for the attribute declared using `default`, to show the default in action

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 `hw6-results.txt` -- at the nrs-labs prompt (the UNIX level, NOT in `sqlplus`!), type:

```
more hw6-results.txt
```

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

When you are satisfied with these, then `hw6.sql` and `hw6-results.txt` are ready to submit.