CIS 315 - Homework 9

Deadline:
NOTE - we're BACK to the "normal" WEDNESDAY deadline, here! ...by the BEGINNING of lab! (...so as to not be too close to the final project milestone deadline.)

1:00 pm on WEDNESDAY, December 8th

How to submit:
When you are ready, within the directory 315hw9 on nrs-labs.humboldt.edu (and at the nrs-labs UNIX prompt, NOT inside sqlplus!) type:
~st10/315submit
...to submit these .sql and .txt files, using a homework number of 9. (Make sure that you see the names of all of the files you wished to submit!)

Purpose:
To practice a little with outer joins and with PL/SQL triggers.

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 hw9-results.txt has been posted along with this homework handout, to help you see if you are on the right track with your SQL and SQL*Plus statements. If your hw9-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 hw9.sql:

nano hw9.sql

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

• your name within a SQL comment
• 315 Homework 9 as a SQL comment
• the date this file was last modified as a SQL comment
• don't start spooling yet
• (it will be specified below just where the spool off command should be placed, also...)

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 problem does not need to be preceded by a prompt command.)

Because this script includes insert statements, this script should start with a "fresh" set of tables each time it runs.

• Make a copy of hw3-setup.sql in your 315hw9 directory
• place a call executing hw3-setup.sql
• now start spooling to hw9-results.txt

**Problem 2**

(Now start preceding each problem with a prompt command.)

Write a break command to break on the vid_id and vid_rental_price columns, skipping one line after each such break.
Then write an outer join that will project `vid_id, vid_rental_price, and the date_out` for each rental of that video, set up such that all `vid_id`'s will appear even if they have never been rented. Order the resulting rows by `vid_id` and in secondary order by `date_out`.

Then, clear breaks -- we don't want this break in effect for the rest of the problems.

**Problem 3**

Write an outer join that will project two columns:

* the client last name concatenated with ', ' and the client first name, set up to display with the column heading `CLIENT`, and
* the category *name* for his/her favorite category, set up to display with the column heading `CATEGORY`,

...set up such that all client names will appear even if he/she has a null favorite category, and display the resulting rows in order of client last name.

**Problem 4**

Hey -- it turns out that you can use outer joins to write a query that gives, say, the number of times any video of a movie has been rented, projecting a count of 0 for movies whose videos have never been rented -- IF you are careful! And you can even combine different joins (for example, an outer join and an equi-join) in the same query...!

Consider:

* you can join more than two tables using the "new" ANSI join syntax using something along the lines of:

```sql
from table1 t1 join table t2 on t1.thing_id = t2.thing_id
    join table t3 on t2.thing_id = t3.thing_id
```

* when I played with it, I found that both of those DON'T have to be just `join` -- one of them CAN be, say, `left outer join`, for example;

* what if you then used a `group by` clause on the result? That's just fine, it turns out;

* and, what if you now projected the `count` of something on that group? Not using `count(*)` -- then every group would get at least a count of 1 -- but using `count` with an argument attribute that might be `null`, but ONLY for outer-joined rows that would not be there for an equi-join? (for example, counting the number of rows in each group with a non-null value of something like the rental table's `date_out`, which is never `null` for a rental, but then would be `null` for something outer-joined with rental that wouldn't appear in an equi-join with rental)

To see this for yourself, write an outer join that indeed projects, for each movie, just two columns: the movie title, and the number of times videos of that movie have been rented, such that movies that have never been rented appear with a count of 0 times rented. Give the number-of-rentals column in the result the heading `NUM_RENTALS`, and order the resulting rows in descending order of the number of rentals, and in secondary order by movie title (for movies with the same number of rentals).

This should not take more than a 5-6 line `select` statement, depending on whether you always put
Problem 5
Write a PL/SQL trigger named approve_rental that will executed before any insertion of a row into rental, that will prevent that rental from being inserted if the credit rating for the client involved in that rental is less than 1.5.

Problem 6
Write an insert statement for a rental for:
* client '7777',
* for a rental_num '0000025',
* for a video of your choice,
* using sysdate for the date_out,
* sysdate+3 for the date_due, and
* not filling the date_returned column.
(approve_rental should cause this insert to fail, as this client's credit rating is 0).

Problem 7
Write an insert statement for a rental for:
* client '6666',
* for a rental_num '0000025',
* for the same video,
* using sysdate for the date_out,
* sysdate+3 for the date_due, and
* not filling the date_returned column.
(approve_rental should permit this insertion.)

Problem 8
Write an update statement to update client '7777' to now have a credit rating of 1.6.
Then write an insert statement for a rental for:
* client '7777',
* for a rental_num '0000026',
* for a video of your choice,
* using sysdate for the date_out,
* sysdate+3 for the date_due, and
* not filling the date_returned column.

(approve_rental should approve this rental, as '7777''s credit rating is now high enough)

**Problem 9**

Finally, write a select statement displaying all of the rows of the rental table, displaying the rows in order of rental_num.

**Problem 10**

Now:
* turn off your spooling

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 hw9-results.txt -- at the nrs-labs prompt (the UNIX level, NOT in sqlplus!), type:

```
more hw9-results.txt
```

You should see that hw9-results.txt contains the query results you just saw within sqlplus. When you are satisfied with these, then hw9.sql and hw9-results.txt are ready to submit.