CS 325 - Homework 10

Deadline

11:59 pm on **Friday, December 3, 2021**.

Purpose

To read and think some more about topics related to transaction management and concurrency control, to get more practice using views, to practice with some of the SQL*Plus commands useful for creating attractive and readable ASCII reports, and to get more experience with some combinations of the SQL features we have discussed so far.

How to submit

Problems 1, 2, 3, and 4 are completed on the course Canvas site.

For Problem 5 onward:

Each time you wish to submit, within the directory `325hw10` 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 `10`.

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

Additional notes:

- You are required to use the HSU Oracle `student` database for **Problem 5** of this homework.
- **DB Reading Packets 9 and 10** and **SQL Reading Packet 8**, on the course Canvas site, and the Week 13 Asynchronous Materials, along with the posted in-class projections from the public course web site, are useful references for this homework.
  - (But since some of the queries also deliberately combine features we have discussed earlier, you may also find it useful to refer to previous SQL Reading Packets, also.)
- 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.
- Feel free to add additional `prompt` commands to your SQL scripts as desired to enhance the readability of the resulting output.
- An example `325hw10-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 5. If your `325hw10-out.txt` matches this posted one, that doesn't guarantee that you wrote appropriate queries, but it is an encouraging sign.
- You are expected to follow **course style standards** for SQL `select` statements.

- Remember that, on the CS 325 public course web site, under "References", there is an evolving list of course style standards posted. There is also a link to these on the course Canvas home page.

Problem 1

Correctly complete the "HW 10 - Problem 1 - Reading Questions for DB Reading Packet 10 - Transaction Management, Part 2", on the course Canvas site.

Problem 2

Correctly complete the "HW 10 - Problem 2 - short-answer questions related to transaction logs", on the course Canvas site.

Problem 3

Correctly complete the "HW 10 - Problem 3 - short-answer questions related to levels of DBMS support for database integrity", on the course Canvas site.

Problem 4

Correctly complete the "HW 10 - Problem 4 - short-answer questions related to the five main database transaction properties", on the course Canvas site.

Setup for Problem 5 onward

Use `ssh` to connect to `nrs-projects.humboldt.edu`, and create, protect, and go to a directory named `325hw10` on nrs-projects:

```
mkdir 325hw10
chmod 700 325hw10
cd 325hw10
```

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 5

This problem again uses the tables created by the SQL script `movies-create.sql` and populated by `movies-pop.sql`. As a reminder, these tables 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 9, so you should **not** need to re-run `movies-create.sql` (*unless* you have been experimenting with table modifications).

Create a file named `325hw10.sql` and include the following within one or more SQL **comments**:

- your name
- `CS 325 - Homework 10 - Problem 5`
- the date this file was last modified

NOTE!!! READ THIS!!!

Now, within your file `325hw10.sql`, add in SQL statements for the following, **PRECEDING** EACH ***EXCEPT* FOR PROBLEM 5-1** with a SQL*Plus `prompt` command noting what problem part it is for.

Problem 5-1

(This ONE problem does NOT need to be preceded by a `prompt` command, for reasons that will hopefully become clear...!)

Because this script includes `update` and `delete` statements, this script should start with a "fresh" set of table contents each time it runs.

- Make a copy of `movies-pop.sql` in your `325hw10` directory.
  - Note that one of several ways to get this is to copy it from my home directory on nrs-projects. For example, assuming that you are currently in your `325hw10` directory,

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

    ...should accomplish this. (NOTE the space and the . at the end -- those are important! They say you are copying the file into your current directory, since `.` in Unix is a nickname for your current directory.)
- ***BEFORE* the** `spool` **command in** `325hw10.sql`, place a call executing `movies-pop.sql`. (That is, place the command you would type within `sqlplus` to run `movies-pop.sql` within your script `325hw10.sql` BEFORE it starts spooling to `325hw10-out.txt`)
  - (why? because I really don't need to see all of the row-inserted feedbacks in your results file... 8-))
  - (putting this within `set termout off` and `set termout on` would be fine, too!)
- use `spool` to **NOW** start writing the results for the REST of this script's actions into a file `325hw10-out.txt`
- put in a `prompt` command printing `Homework 10 Problem 5`
- 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 REST of the problems below BEFORE this `spool off` command!

Problem 5-2

(NOW start preceding each problem with a `prompt` command.)

Include SQL*Plus statements for each of the following within your script:

- explicitly clear any previously-set column headings, breaks, and computes
- create a **top title** and a **bottom title** (title contents of your choice)
- make the pagesize 35 lines and the linesize 75 characters.
- turn feedback off

Problem 5-3

Drop and create a view called `rental_history` which gives a view of which clients have rented which videos by including rows with the following 4 columns:

- in its first column, it has the last name of the client followed by the first name, with a comma and a blank separating the last name and the first name of each (e.g.:

```
Tuttle, Sharon
```

). Give this column the name `client_name` (**do NOT use double-quotes in specifying this column name, or ANY of the view column names given in this problem!**)
  - Hint: you *can* use concatenation in the `select` clause of the `select` statement defining a view.
- in its second column, it should have the movie title rented in that rental; make sure that, one way or another, the name of this column is `movie_title`
- in its third column, it should have the format of the video rented; make sure that, one way or another, the name of this column is `vid_format`
- in its fourth column, it should have the `vid_rental_price`; make sure that, one way or another, the name of this column is `vid_rental_price`
- One more hint: if a join involves four tables, make sure you have three appropriate join conditions!

Then write a query doing a relation selection of this view, displaying the rows in order of `client_name`, with a secondary ordering (for rows with the same client name) from most expensive video rental price to least expensive video rental price, and with a third ordering by `movie_title`.

Problem 5-4

Consider the `rental_history` view from Problem 5-3.

Write `column` commands for each of the following:

- give column `client_name` the heading `Client` (camel-case, uppercase for the first letter and lowercase for the rest),and format it so that it is:
  - **narrower** than its default width,

  - but **wide enough** for all of the last-name-first name combinations currently there,
  - but **narrow enough** that all of the columns "fit" on 1 line without wrapping in the subsequent query results involving this column
- give column `movie_title` the heading `Movie Title` (camel-case, including the blank between the 2 words), and format it so that it, too, is:
  - **narrower** than its default width,
  - but **wide enough** for all of the movie titles currently there,
  - but **narrow enough** that all of the columns "fit" on 1 line without wrapping in the subsequent query results involving this column
- give column `vid_format` the heading `Format` (camel-case, uppercase for the first letter and lowercase for the rest), and format it so that its entire heading shows
- give column `vid_rental_price` the heading, with `Rental Price` (camel-case, including the blank between the 2 words), and format it so that:
  - it contains a $
  - it always displays to 2 fractional places (to 2 decimal places)
  - its entire heading shows

AFTER these `column` commands, then use / to **rerun** the previous query, from Problem 5-3, doing a relation selection of this view, displaying the rows in order of `client_name`, with a secondary ordering (for rows with the same client name) from most expensive video rental price to least expensive video rental price, and with a third ordering by `movie_title`.

Problem 5-5

Consider what you get when you perform an equi-join of the `movie`, `video`, and `movie_category` tables -- now you have the corresponding movie details for each video's movie. Drop and create a view called `category_stats` which groups the videos by movie category **name** and contains only the category **name**, the number of videos in that category, and the average rental price of videos in that category. (Remember to rename the column names corresponding to function calls; choose appropriate column names.)

Follow that with a query doing a relational selection of this view, displaying the rows in order of most number of videos to least number of videos, with a secondary ordering by highest average rental price to lowest average rental price.

Problem 5-6

Consider the `category_stats` view from Problem 5-5.

First, change the pagesize to 20 lines.

Write `column` commands for each of the following:

- give `category_stats`' first column the heading `Category` (in camel-case as shown)
- give `category_stats`' second column the heading `# Videos` (camel-case, including the blank

between the 2 parts), and format it so that it is wide enough to show the whole heading

- give `category_stats`' third column the heading `Avg Price` (camel-case, including the blank between the 2 parts), and format it so that:
  - it contains a $
  - it always displays to 2 fractional places (to 2 decimal places)
  - its entire heading shows

AFTER these column commands, then use / to **rerun** the previous query, from Problem 5-5, doing a relational selection of the view `category_stats`, displaying the rows in order of most number of videos to least number of videos, with a secondary ordering by highest average rental price to lowest average rental price.

Problem 5-7

Commit the current state of your database -- we are about to make some hopefully-temporary changes over the next few problems.

5-7 part a

Write an `update` statement to **decrease** the rental prices of all videos with format Blu-Ray by 0.25.

Then display the current contents of the view `category_stats` again, using the same row-ordering as you did at the end of Problems 5-5 and 5-6. (Note that / will **NOT** work for redoing *this* query -- it causes the last SQL statement to be redone, which in this case is the `update` statement, which we do **NOT** want redone...!)

5-7 part b

Write a query, **using ONLY the view `rental_history`**, performing a **"true" relational projection** of just the names of those clients who have rented 'Gone with the Wind', displaying the rows in order of `client_name`.

5-7 part c

Write a single `delete` statement that will delete all rows from `rental` that involve the client with client number '5555'. Then repeat your query from part b.

(Again, note that / will **NOT** work for redoing *this* query -- it causes the last SQL statement to be redone, which in this case is the `delete` statement, not the preceding `select`.)

5-7 part d

Write a query, using the view `rental_history` only, to project each client's name and the number of rentals they have made, displaying the rows in order from highest to lowest number of rentals, with a secondary ordering by client name.

And, now that we are done with using `update` and `delete` to show how views nicely reflect changes in their underlying tables, **roll back** the database to its state at the beginning of Problem 5-7.

Problem 5-8

Do the following:

- change the pagesize to **45** lines
- write a `break` command to suppress repeated client names, putting **one blank line** between each set of such rows
- write a query, **using the view `rental_history`**, to do a **"pure" relational projection** of the names of clients who have rented movies and the titles of movies they have rented, displaying the rows in order of client name and in secondary order by movie title

Problem 5-9

Recall that one of the computations that `compute` can do is `avg`.

Change the pagesize to **60** lines.

Write a `compute` command that will determine the average `vid_rental_price` for each set of consecutive client names.

Then, write a relational selection **of the view `rental_history`**, displaying the rows in order of `client_name` and in secondary order by `movie_title`.

Problem 5-10

Now:

- Turn off your spooling
- Either call `cleanup.sql` (available with this homework handout) or put in the SQL*Plus commands to at least:
  - clear columns, breaks, and computes,
  - reset feedback to its default value,
  - reset pagesize and linesize to their default values,
  - turn off the top and bottom titles

Submit your files `325hw10.sql` and `325hw10-out.txt`.