CS 325 - Homework 9

Deadline:

Problem 1 -- answering reading questions on Moodle for Reading Packet 10 -- needs to be completed by 10:45 am on **THURSDAY, DECEMBER 1**.

The remaining problems are due by **11:59 pm** on **Friday, December 2, 2016**

How to submit:

For Problem 2 onward:

Each time you wish to submit, within the directory `325hw9` 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 `9`.

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

Purpose:

To think a little about integrities, desired properties of database transactions, database recovery, and some of the concurrency control mechanisms discussed in lecture; to practice more with SQL views, including some more practice with SQL `update` and `delete` statements to show how changing the underlying table works smoothly with views; and to practice with some of the SQL*Plus commands useful for creating attractive and readable ASCII reports.

Additional notes:

- SQL Reading Packets 7 and 8 and "regular" Reading Packets 9 and 10 are useful references 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.
- You are expected to follow the SQL style standards noted in previous homework handouts and mentioned in class.
- You are required to use the HSU Oracle `student` database for this homework.
- Feel free to add additional `prompt` commands to your SQL scripts as desired to enhance the readability of the resulting spooled output.
- An example `hw9-3-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 3. If your `hw9-3-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.)

Problem 1

Have to correctly answer the Reading Questions for Reading Packet 10 - Transaction Management - part 2, on the course Moodle site, by 10:45 am on **THURSDAY, DECEMBER 1**.

Problem 2

Place your name and then your answers for this part in a file named `hw9-2.txt` within your `325hw9` directory.

Problem 2-1

We have discussed entity integrity, domain integrity, referential integrity, and transaction integrity, and how a DBMS may support (or enforce) them.

Give the most appropriate of these four types of integrity in answer to each of the following:

2-1 part a

This is being supported (to at least some degree) in the Oracle DBMS by refusal to allow the insertion of an attribute value that is not the given type for that attribute (based on the type declaration for that attribute).

(that is, this kind of integrity is being enforced when the Oracle DBMS will not permit the insertion of a row that includes a date value for a column declared to be of type integer)

2-1 part b

This is being supported when a DBMS does not permit the insertion of a row with the same primary key as an already-existing row in that table.

2-1 part c

This is being supported when a DBMS does not permit the insertion of a row with a foreign key value for which there is not a corresponding primary key value in the parent table.

2-1 part d

Oracle SQL supports this with its `COMMIT` and `ROLLBACK` statements.

2-1 part e

If a DBMS permitted you to insert duplicate rows within a table, it would not be supporting this.

2-1 part f

One way in which Oracle DBMS supports this is with its "on delete restrict" default for foreign keys --- that is, in Oracle, if you attempt to delete a parent row with a primary key matching children rows with that value for their foreign key, you are not permitted to do so.

2-1 part g

When a DBMS has automatic backup and recovery capabilities, it could be said to be providing support for

this (in terms of the desired property of durability).

2-1 part h

Oracle is providing additional support for this with the `check` clauses that one may use in a `create table` statement to restrict the allowed values in a particular column, which it then uses to only permit insertions of rows which have values for that column that satisfy the `check` clause.

Problem 2-2

We have discussed five desired properties for database transactions: atomicity, consistency, isolation, durability, serializability (ACIDS)

Give the most appropriate of these five desired properties in answer to each of the following:

2-2 part a

Which of these is particularly about the database maintaining its consistent state *between* transactions?

2-2 part b

Which of these do you have when the concurrent execution of transactions is equivalent to the case where the transactions executed serially in some arbitrary order?

2-2 part c

Which of these stipulates that a transaction has to transform a database from one consistent state to another consistent state?

2-2 part d

Which **two** of these is it the responsibility of the recovery subsystem to ensure?

2-2 part e

Which of these means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed (at least, the effect must be as if this was the case)?

2-2 part f

Which of these is the "all or nothing" property, requiring either that all operations/parts/steps of a transaction be completed in their entirety, or not performed at all?

Problem 2-3

You have a transaction log, in which all transaction operations since the last complete database backup are recorded in chronological order. A failure occurs at time X, and recovery is begun by starting with the last complete database state. In this system, recovery is done using **rollforward**, as discussed in lecture.

2-3 part a

In looking at the transaction log during recovery, it is discovered that transaction T1 had been rolled back/aborted before time X. Will its logged actions be re-done as part of the recovery process?

2-3 part b

In looking at the transaction log during recovery, it is discovered that transaction T2 had been committed before time X. Will its logged actions be re-done as part of the recovery process?

2-3 part c

In looking at the transaction log during recovery, it is discovered that transaction T3 had not yet been committed before time X. Will its logged actions be re-done as part of the recovery process?

For Problems 2-4 through 2-6...

Assume that A, B, C, D, and E are data items and that T1, T2, T3, T4, T5, and T6 are transactions.

Problem 2-4

Assume that your DBMS is using binary locks for concurrency control. Currently, T1 has a binary lock on data item A, and T2 has a binary lock on data item C. All other data items are unlocked.

2-4 part a

T3 requests a binary lock on B. Will T3 obtain the lock at this point, or will T3 have to wait?

2-4 part b

T4 requests a binary lock on C. Will T4 obtain the lock at this point, or will T4 have to wait?

Problem 2-5

Assume that your DBMS is using shared/exclusive locks for concurrency control. Currently T1 has an exclusive/write lock on data item A, and T2 has a shared/read lock on data item C. All other data items are currently unlocked.

2-5 part a

Consider the operations Read and Write. Which of these can T1 do to A, given its lock (read, write, neither read nor write, or both read and write)?

2-5 part b

Consider the operations Read and Write. Which of these can T2 do to C, given its lock (read, write, neither read nor write, or both read and write)?

2-5 part c

T3 requests an exclusive lock on D. Will T3 obtain the lock at this point, or will T3 have to wait?

2-5 part d

T4 requests a shared lock on C. Will T4 obtain the lock at this point, or will T4 have to wait?

2-5 part e

T5 requests an exclusive lock on C. Will T5 obtain the lock at this point, or will T5 have to wait?

2-5 part f

T6 requests a shared lock on B. Will T6 obtain the lock at this point, or will T6 have to wait?

Problem 2-6

Transaction T1 has an exclusive lock on data item A, and has requested, and is waiting for, a shared lock on data item B. But, T2 has an exclusive lock on data item B, and has requested, and is waiting for, an exclusive lock on data item A. What is this situation an illustration of?

Problem 3

Now, you are back to using Oracle and SQL again.

This problem again uses the tables created by the SQL script `hw4-create.sql` and populated by `hw4-pop.sql`. (And now `hw4-pop.sql` INCLUDES appropriate `delete` commands so that it unpopulates the tables before trying to insert rows...!)

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` *unless* you have been experimenting with insertions or other table modifications, or unless any of these table names happen to conflict with your project tables...!)

Use `nano` (or `vi` or `emacs`) to create a file named `hw9-3.sql`:

```
nano hw9-3.sql
```

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

- your name within a SQL comment
- `CS 325 Homework 9 - Problem 3` within a SQL comment
- the date this file was last modified within a SQL comment
- **don't start spooling yet**
  - (it will be specified below just where the spool off command should be placed, also...)

NOTE!!! READ THIS!!!

Now, within your file `hw9-3.sql`, add in SQL statements for the following, **PRECEDING** EACH ***EXCEPT* FOR PROBLEM 3-1** with a SQL*Plus `prompt` command noting what problem part it is for.

Problem 3-1

(This problem does **NOT** need to be preceded by 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 **two line** top title and a **two line** bottom title (title contents of your choice)
- make the pagesize 35 lines and the linesize 75 characters.
- turn feedback off

Because this script experiments with `update` and `delete` statements, this script should start with a "fresh" set of table contents each time it runs. (I am going to try to also have you practice with `commit;` and `rollback;` below, but I still want to play it safe a bit as you practice with modifying tables.)

- Make a copy of `hw4-pop.sql` in your `325hw9` 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 `325hw9` directory,

    ```
    cp ~st10/hw4-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** `hw9-3.sql`, place a call executing `hw4-pop.sql`. (That is, place the command you would type within `sqlplus` to run `hw4-pop.sql` within your script `hw9-3.sql` BEFORE it starts spooling to `hw9-3-out.txt`)
  - (why? because I really don't need to see all of the row-inserted feedbacks in your results file... 8-))
  - IF YOU'D LIKE TO TRY SOMETHING NEAT: Sky McKinley showed me a very useful SQL*Plus command I had not seen before:

    ```
    set termout off
    @ hw4-pop
    set termout on
    ```

    This turns terminal output off, then you run the commands you don't really want to see all the output

from (here resetting up tables in a script you KNOW is fine), then you turn terminal input on again.

  - SO if you'd also like to use this pair of statements around the execution of `hw4-pop.sql`, feel free!

- use `spool` to NOW start writing the results for the REST of this script's actions into a file `hw9-3-out.txt`
- put in a `prompt` command printing `Homework 9 - Problem 3`
- 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 3-2

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

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
```

). Name this column `client_name` (**do NOT use double-quotes in specifying this column name, or ANY of the view column names given in this problem**)

- 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`

Follow that with a query doing a relation selection of this view, displaying the rows in order of most expensive video rental price to least expensive video rental price, with a secondary ordering by client_name, and with a third ordering by movie_title.

Problem 3-3

Consider the `rental_history` view from Problem 3-2.

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` a 2-line heading, with `Video` on the first line and `Format` on the second (and each in camel-case as shown), and format it so that its entire heading shows
- give column `vid_rental_price` a 2-line heading, with `Rental` on the first line and `Price` on the second (and each in camel-case as shown), 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 3-2, doing a relational selection of the view `rental_history`, displaying the rows in order of most expensive video rental price to least expensive video rental price, with a secondary ordering by `client_name`, and with a third ordering by `movie_title`.

Problem 3-4

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 3-5

Consider the `category_stats` view from Problem 3-4.

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 a 2-line heading, with `Average` on the first line and `Price` on the second (and each in camel-case as shown), 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 3-4, 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 3-6

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

3-6 part a

Write an `update` command to decrease the rental prices of all videos with format Blu-Ray by 0.99. Then display the current contents of the view `category_stats` again, using the same row-ordering as you did at the end of Problems 3-4 and 3.5. (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` command, which we do **NOT** want redone...!)

3-6 part b

Write a query, using the view `rental_history` only, 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`.

3-6 part c

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

3-6 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 3-6.

Problem 3-7

Do the following:

- change the pagesize to 45 lines
- write a `break` command to suppress repeated movie titles, 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 titles of

movies that have been rented and the names of the clients who rented them, displaying the rows in order of movie title and in secondary order by client name

Problem 3-8

Do the following:

- write a `break` command to suppress repeated category names and movie ratings, putting **one blank line** between each set of such category names (but **NOT** between each set of such movie ratings)
- write a `column` command to set the `category_name`'s column to have a display heading of `Category` (in camel-case as shown)
- write a `column` command to set the `movie_rating`'s column to have a display heading of `Rating` (in camel-case as shown) with a format wide enough that the entire heading shows
- write a query to project the movie category name, the movie rating, and the movie title for each movie, displaying the rows in order of category name, in secondary order by `movie_rating`, and in third order by `movie_title`

Problem 3-9

Aha -- one of the computations that `compute` can do is indeed `count`.

Consider Problem 3-8. Write a `compute` command that will cause how many `movie_titles` are within each set of consecutive category names to be displayed, and then use `/` to redo the previous query, from Problem 3-8, which should now include these counts.

Problem 3-10

First, clear computes. Then:

- clear computes
- write a `break` command to suppress repeated category names and movie titles, putting **one blank line** between each set of such category names (but **NOT** between each set of such movie titles)
- write a `compute` command that will cause the average video rental price for each set of such category names to be displayed
  - (I did use the `label` feature of `compute` change the average's default label in the example solution used to generate the posted example output -- it is up to you if you wish to also do so.)
- write a `column` command that will set the `vid_rental_price`'s column to have a display heading of `Cost` (in camel-case as shown) with a format that always displays with 2 fractional places, and wide enough that the entire heading and the prices show.
- write a query to project the category name, the movie title, and the video rental price for each video rental, displaying the rows in order of category name, in secondary order by movie title, and in third order by video rental price.

Problem 3-11

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

When you think the results of all of these problems look correct, this would also be a good time to look at the contents of `hw9-3-out.txt` -- at the nrs-projects prompt (the UNIX level, NOT in `sqlplus`!), type:

```
more hw9-3-out.txt
```

You should see that `hw9-3-out.txt` contains the problem results you just saw within `sqlplus`.

When you are satisfied with these, then `hw9-3.sql` and `hw9-3-out.txt` are completed.