**Introduction**

The goal of this assignment is to improve your skills of writing declarative queries on a relational database, in general, and also to improve your fluency in SQL (and SQLite).

You have been provided with the following relational schema.

* *users(email, name, pwd, city, gender)*
* *products(pid, descr)*
* *sales(sid, lister, pid, edate, descr, cond, rprice)*
* *bids(bid, bidder, sid, bdate, amount)*
* *items(sid, no, pid, descr)*
* *reviews(reviewer, reviewee, rating, rtext, rdate)*
* *previews(rid, pid, reviewer, rating, rtext, rdate)*

The tables are derived from the specification of Assignment 1 and the names of the tables and columns should give the semantics, except minor differences which are explicit in table definitions, insert statements or queries.

**Creating the database**

Using [the SQL statements provided](https://eclass.srv.ualberta.ca/pluginfile.php/5496430/mod_page/content/54/a2-tables.sql?time=1581019600784), create the above tables in SQLite3 on Lab machines with some data. Here is [a small initial data](https://eclass.srv.ualberta.ca/pluginfile.php/5496430/mod_page/content/54/a2-initial-data.sql?time=1580521100641) to get you started.

**(90 marks) Queries**

Write down the following queries in SQL and run them in SQLite3 over the database created. You will be writing ONE SQL statement for every query (here One SQL statement starts with a SELECT and ends with a semicolon but may include multiple select statements combined in the form of subqueries and/or using set operations). Your SQL queries for questions 1-3 cannot use any of aggregation, grouping, or nesting (set operations are ok).

1. Find the email and name of users who have had a bid on their own sales.
2. List sid, lister, condition and the reserved price of all sales which mention either *ticket* or *voucher* in their descriptions and have at least one bidder from *Edmonton*.
3. List sid and lister of all sales that end over the next 3 days but have no bids with an amount reaching or exceeding the reserved price. *Hint*: Check out date and time functions in SQLite.
4. List all sales that are active (i.e., the end date is in future), and for each such sale list sid, description, condition, the number of bids, the highest bid amount and the number of days left until the sale ends. Your listing must include sales that have no bids. *Hint*: You may find outer join useful here and some of the subsequent queries.
5. Find all undervalued sales that have the term *xbox* in their descriptions; a sale *s* is undervalued if it is associated to a product *p* and *p* is associated with another sale *s'* and the highest bid for *s'* is more than twice the amount of the highest bid for *s*. Matching of the keyword *xbox* must be case-insensitive. For each qualifying sale *s*, list sid, lister of the sale, pid of the product associated with the sale and the maximum bid amount offered for the sale.
6. Find sid of sales that have the text *PS4* in their descriptions, and limit the results to only those that are listed by users who have at least 3 reviews with an average review rating greater than 4. The match for *PS4* must be case-insensitive.
7. For each user who has had some bids, list the user email, the total number of distinct sales the user has had bids on, the number of winning bids, and the total dollar amount of the wining bids. The user has a winning bid if the sale is over (i.e., the end date is in the past), the user has the highest bid and the bid amount is not less than the reserved price. The result should include users who have no winning bids. *Hint*: subqueries can be used in the from clause.
8. For users who have received at least 3 reviews, find the email and the average review rating. Limit the result to only top 5 users with the highest average review ratings, and sort the results in a descending order of average review ratings. *Hint*: check out the *limit* function in Sqlite.
9. Create a view called *product\_info* with columns pid, descr, revcnt, rating, rating6, and salecnt. The view includes for each product the product id, the description, the number of reviews, the average rating, the average rating based on the reviews written within the past 6 months, and the number of different sales that are either associated to the product or have an item that is associated to the product. Include products with no reviews or sales in the output with zero counts (if applicable) or null values.
10. Using the view created in the previous question, find users whose sales are all associated to hot products. A product is hot if its average rating is larger than 4 and its sale count is larger than the average sale count.

**(upto 5 bonus marks for the first 3 people sharing adequate data) Preparing test data**

Written queries should be tested for correctness and bug fixes, very much like programs written in any programming language. For testing, you need to have enough data in your tables such that all your queries are meaningful and non-trivial (e.g. the returned answers are not empty). You are encouraged to share your data with your classmates or use data prepared by them. *To make this collaboration happen, there will be up to 5 bonus marks (at the instructor's discretion) to the first 3 people who prepare a test data and share it with the rest of the class. Make sure your data is correct and covers all queries and boundary cases (as discussed above), and fix it if you find problems after your posting. Otherwise, posting an incomplete or incorrect test data early on does not place you in top 3, and others can post their test data for the bonus mark. We will be roughly testing the coverage of your test data that is shared with class.  
  
If you are sharing your test data, please post it to the course discussion forum.* Put all your *insert* statements in a file called *a2-data.sql*. Make sure to put down your name, email and a date when it is published or revised at the beginning of the file as a comment line (e.g. -- Data prepared by <firstname lastname>, <email address>, and published on <date>). If you are using data prepared by someone else, leave the identification line unchanged.

**(10 marks) Testing and report**

Starting from scratch, create your database as

*sqlite3 a2.db <a2-tables.sql*

and populate your tables using data file a2-data.sql (prepared in the previous step) as

*sqlite3 a2.db <a2-data.sql*

Put all your SQL queries in a file named a2-queries.sql; Add the following line at the beginning of the file

*.echo on*

and the following line before each SQL query (replacing X with the query number).

--Question X

Run your queries on your data file as

*sqlite3 a2.db <a2-queries.sql >a2-script.txt*

You will be submitting both a2-data.sql and a2-script.txt electronically as described in the instructions for submissions.  
  
You queries are expected to run on the lab machines. If you are developing your queries on your machine, ensure they are running smoothly on the lab machines to pass our tests.

**Instructions for Submissions**

We will make use of some automated tools in testing your queries. Thus it is important that you follow the following instructions closely.

1. Your queries will be tested under a TA account with the provided tables. Do not use any table or column names other than those provided.
2. Write each query in a separate file. Your solution must have **one SQL statement for each query**. In other words, you cannot use views or temporary tables unless you are explicitly asked to do so. The first query must be saved in a file named 1.sql, the second query in a file named 2.sql, and so on until the tenth query, which is to be saved in a file called 10.sql (**the names are important!**).
3. The first line of each query file must have the command:

.print Question *X* - CC*ID*

where *X* is the number of the query and CCI*D* is your CCID. For example, the first line of the third query file for the user with ccid 'drafiei' will be:

.print Question 3 - drafiei

The rest of each file must contain the SQL query you are submitting and nothing else.

1. Include with your submission a README.txt file that has your name, ccid, lab section, and the list of people you collaborated with (as much as it is allowed within the course policy) or the line "I declare that I did not collaborate with anyone in this assignment". A submission without a README.txt file or with missing information will lose 5% of the total mark.
2. Bundle all your queries, insert statements (a2-data.sql) and scripts (a2-script.txt) into a single tarfile by executing the Unix command (everything should be on one line):

tar -czf a2.tgz README.txt a2-data.sql a2-script.txt 1.sql 2.sql 3.sql 4.sql 5.sql 6.sql 7.sql 8.sql 9.sql 10.sql

1. Submit the file *a2.tgz* at the [submission page](https://eclass.srv.ualberta.ca/mod/assign/view.php?id=3927411) after logging into eclass.

Eclass does not support versioning (unfortunately) and each new submission replaces your previous one. This makes last minute submissions somewhat risky. Avoid last minute submissions as much as you can, and check your submissions after an upload to make sure the right content is uploaded. A common mistake is to use a wrong tar command and submit a corrupt file.