CPSC 304 Project Instructions: Formal Specifications

Due Date: Monday, October 23, 2017 at 23:59

Last Update: October 17, 2017

Details

This part of the project will be worth 7% of your project grade.

The goal of this project checkpoint is for you to tell us what to expect from your final application.

Note that we expect your final project to have, or make use of, some sort of rudimentary Graphical User Interface (UI); but, it doesn't have to be fancy.

Hand in the following information **about the functionality that your application will have:**

1. What will the different users be able to do with your system? You don't have to write the queries in SQL yet, but describe the data and output that each user will see and work with. You'll want to provide different functionality for different users, even if it's all driven from the same menu or front-end in your application. Don't bother creating separate sign-on screens for different users; it is OK if you share the same main menu. Here are some examples of what we mean:
    - In a banking application, bank employees and customers can see and update different things. We might have one set of SQL statements for the bank employees, and one set for the customers. We don't expect different users to start with different logon IDs—the different users can just select or click on different menu options (like 1 for a withdrawal, 2 for a deposit, 3 to see a monthly statement, etc.)
    - In an airport application, the users might be passengers and airline employees. Employees might assign planes to gates, but passengers wouldn't do this.
    - In a medical application, the users might be patients, lab personnel, doctors, etc.
    - Even if your application doesn't naturally lend itself to different "classes" of users, then you can still have different features: different windows, different views, and different update activities that junior or less-privileged employees might see.

    Deliverables for this part:

    a) Deliverable 1: Identify the different kinds of users that your application will service.
    b) Itemize the **kinds of queries, reports, or changes to the database** that you expect these users to be able to do. You can add to, or modify, these queries in the future, but at this stage, we want to get a sense of what your application is going to do.

You don't need to provide the SQL, yet; but what you have learned about Relational Algebra or SQL so far in the course may help you decide on some of these queries.

- You need to have **at least 10 SQL data manipulation queries and changes**: some simple, and some complex. You don't have to create the SQL statements yet, but **describe in words what they will be**.
    - More about this requirement and its deliverables are listed below.
- We expect your application to have the ability to **update** at least some part of the database, *and* the ability to have some queries that depends on the user's input.
    - For example, in a banking domain, I might want the customer to be able to update the database by transferring money from one account to the other, and I might also want a bank employee (e.g., teller) to be able to ask for all of the account information for a given customer. In both cases, the customer number needs to be specified, along with any needed amounts, etc. Don't hard-code these parameters; the user will be entering them.
- You need to have two or more modification operations that each use a WHERE clause in their respective SQL statements. Describe each of these for your application:
    i. Deliverable 2: You need to have at least one INSERT statement.
    ii. Deliverable 3: You need to have at least one DELETE statement.
    iii. Deliverable 4: You need to have at least one UPDATE statement.
- Queries involving joins: We expect that some of your queries will be something *interesting*, that is, something beyond a simple select, project, or join from a single relation. In other words, you need to have some **meaningful queries that join multiple tables**. Describe them:
    i. Deliverable 5: At least one query must join 3 or more tables.
    ii. Deliverables 6 and 7: At least 2 other queries need to join 2 or more tables.
- Deliverable 8: At least one query must be an interesting GROUP BY query (aggregation). Describe it.
- Deliverables 9-11: Describe the other queries you plan to have (these can be simpler queries), so that you have at least 10 SQL statements overall.
- Deliverable 12: You need to have at least one view for your database, created using the CREATE VIEW statement in SQL. It should be a proper subset of a table. When the user wants to display all the data from this view, they will get all the columns specified by the view, but not all of the columns in the underlying table.
    - You don't have to give the SQL for the view yet; just indicate the subset of which table(s) that will be used, and for which kind of user.
- If you anticipate having any triggers or assertions, this would also be functionality that would be useful to present as part of your project; but if you don't have any triggers or assertions or procedures, don't worry about it because you won't be deducted marks.

2. Deliverable 13: **What will the division of labour be like among your group?** We want to know how you're planning on splitting up the work. You should plan for approximately equal number of tasks, or an equal amount of time required, for each member of your group. Certain members of your group might want to go over and above this for their own area of responsibility. That's fine; however, don't do another member's work. For example, you might have these tasks (and possibly others):

    a) Create the tables and other database objects. Be sure to save your SQL scripts, as you are very likely to DROP and re-create your work. In fact, we may ask you to do this, during your demo.
        1. Tutorial #6 (and its reference material) provides some examples of SQL DDL.
    b) Create data for the tables. Then, populate (load) the tables. Be sure to save your scripts. You might want to create a bunch of SQL INSERT statements to load the data. You might even want to write a short program that loops and creates those SQL INSERT statements based on data being read from a file, or that generates the data from scratch (possibly via random number generators).
    c) Code each set of queries and test them in SQL (e.g., Oracle's SQL*Plus or another DBMS of your choice).
    d) Embed the SQL statements in a program, and code the programming logic. Use a graphical user interface. Format the output appropriately (e.g., in the form of a results window, an HTML page, etc.)
        1. The application logic is likely to take up more of the time than any other individual task.
        2. All of your group members must take part in this talk because embedded SQL in a host language should be practiced by everyone. However, it is OK if some group members do more programming than others, providing those other group members do more of the other tasks.
        3. Be prepared to DROP your tables and re-create them, if something significant goes wrong during the UPDATE, INSERT, and DELETE operations.
    e) Test each set of queries. Determine how errors will be handled. Take appropriate action. For example, what happens if a user inserts a duplicate key?
    f) Document the project.
    g) All group members must be present to demo your application to a TA. This will take place near the end of the term.

If you wish to update your schema at this checkpoint, or even your ER-diagram, that's OK. After thinking about your queries and reports, you might have a better idea of what's missing.

If you think something's going to be too much work, ask your TA before committing to it. But, please don't ask your TA to pre-mark your project deliverables. We won't be able to answer questions like, "Is this correct?" "Is our solution worth full marks?"

If you want to see a *preview* of both the formal specifications (and the ER diagram which you already handed in), take a look at the following Bookstore (UBStore) example. (You may need to copy-and-paste the following links, if you can't get there by clicking on the links.)

- https://www.ugrad.cs.ubc.ca/~cs304/2017W1/project/p1/p1-desc.html
- https://www.ugrad.cs.ubc.ca/~cs304/2017W1/project/p1/p1-solution.html
- Note that this example is based on a previous offering of this course—so, be sure that you follow the *current* instructions given to you earlier in this course. You may *not* model any previous UBC example data (e.g., discussed in lectures—although the music/radio case is fine), employee supervision (discussed in a textbook), a bookstore (project topic in this example), MP3 storage (may be discussed in class), Motor Vehicles Branch application, or a project given to you in the tutorials. Additionally, this must be a *new* project—you may *not* reuse a pre-existing project like something you got from someone else, a book, the Internet, a co-op term, etc.

What to Turn In

- A cover page: https://www.ugrad.cs.ubc.ca/~cs304/2017W1/project/CoverPage.html.
- Your specifications in the form of a PDF file. Do not submit a photo; instead, paste it into Word (or other application) and generate a PDF file. If you hand-draw them, or write them by hand, that's OK; but turn them into PDF before submitting them via `handin`.
- Keep your PDF files to under 1 MB each.
- A README.txt can be included to list any special instructions or comments to be given to the marker. Be sure to include your teammates' names in this file.
- Summary: Submit 2-3 things in all: 1 PDF file (specs, required); a README.txt file (if necessary); and your cover page (required).

How to Use Handin

One group member should be the person doing all the electronic `handin` submissions. This will simplify things when the TAs have to check off the deliverables and associate them with the correct group. This person should be the same person who submitted the Project Proposal and the ER-diagram.

To submit your Cover Page and your PDF files, perform the following steps:

- On an undergraduate machine (e.g., using `ssh` on `remote.ugrad.cs.ubc.ca`), copy the file(s) that you want to hand in, to the directory `~/cs304/project_specs` (note that it will wind up in your home directory). You can create this directory using:
    - `mkdir ~/cs304/project_specs`
- Copy your file(s) into this directory.
- Then, from your home directory, run:
    - `handin cs304 project_specs`
- **Take a screen shot of your successful submission, in case any problems exist.** Without this, it may be difficult for us to give part marks if something went wrong.

Only one group member should submit the assignment. It should be the same group member as for previous project deliverables. Group members: verify with your group that the submission has actually taken place!