CIS*2750 Assignment 4
Deadline: Friday, March 30, 11:59pm
Weight: 12.5%

1. Introduction

For this assignment you will expand your A3 in the following ways:
- Add UI for Database activities
- Add UI for interacting with the database
- Create and maintain tables in an SQL database
- Connect the app to a MySQL database (see Section 4)

The UI is described in Section 2. Section 3 describes the SQL table layout. Section 4 provides various implementation details.

You do not need to modify any of the existing A3 UI, since none of it relies on the database backend. All A4 functionality is an addition to existing A3 functionality, not a replacement for it.

2. User Interface Additions

2.1 Database functionality

To interact with the genealogy database, add a **Database** UI section with the items described below. This section can be placed on in `index.html` below the A3 functionality. You can provide a header for A4 functionality to make it more visible.

Each item is only active when it is logically meaningful. After every command, except **Execute Query**, print in the Console Panel a status line based on a count of each table's rows: "Database has N files and N individuals".

- **Store All Files**: This command is used to store the data from - and about - all .ged files on the server into your database. This should be disabled if there are no files on the server.

  You will need to create a table that will store the information displayed in the File Log Panel. You will need to create a table `FILE` and store the summary data in it. The layout of this table is described in Section 3.

  In addition, you will need to a table `INDIVIDUAL` containing data about individuals stored in each file. This is essentially the data shown in the GEDCOM View Panel, as well as the name of the file for that individual. The layout of this table is described in Section 3.

  For every .ged file, obtain all necessary data, and store it in the table. If a file has no individuals, its information is stored in the `FILE` table, but you don't need to add any new individuals to `INDIVIDUALS` table.

- **Clear All Data**: Delete all rows from all the tables. This may have to be done in a certain order due to the foreign keys. This is only active if any table is not zero length. Do not drop the tables themselves. You do not need to update the File View Panel, or GEDCOM View panel. That functionality does not rely on the database, and does not need to be modified.

- **Display DB Status**: This displays in the Console Panel panel the status line described above: "Database has N files and N individuals".

- **Execute Query**: this UI item is used to query the database. It should contain a way to submit one of the five standard queries discussed below. In addition, it contains the following two panels:

  - **Query**: These controls are used to make ad hoc queries on the database. The user must be able to select one of several standard queries, which the user can fill in and submit.

    This panel contains a text area initialized with the word "SELECT" where the user can finish typing an arbitrary SELECT statement. It also has a Submit button to send the chosen standard query or SQL statement to the database.

There is also a Help button which displays a small table showing the output of "DESCRIBE FILE; DESCRIBE INDIVIDUAL" queries, giving precise table and column names--information the user will need to correctly format a query.

- **Results**: The results from the submitted query are displayed here in a scrollable table.

2.2 Standard queries

The **Execute Query** menu item displays five standard queries, with some parameters that the user may fill in. There should be some way to select which query is submitted. The queries must be displayed in simple English, but your program will generate the underlying SQL statements incorporating any filled-in variables. The intended user of this functionality is someone who wants to access the genealogy database, so think about what they might want to know, and how to make it easier for them to provide the information necessary to execute the query.

Two required queries are given below. You should come up with three additional different ones that are not overly simplistic. For our purposes, "overly simplistic" means anything that doesn't have conditions, a join, a nested query, and/or aggregate functions. The first required query **is** overly simplistic, and is given to you as a warm-up. Your non-trivial queries can rely on the data in `shakespeare.ged`. In other words, if we upload `shakespeare.ged`, we can execute any of the five queries, and get meaningful, non-empty results.

Queries:

1. Display all individuals sorted by last name. **(required)**

2. Display the individuals from a specific file. How you sort the individuals is up to you. **(required)**

3. - 5. Up to you.

3. Database Design

3.1 Naming conventions

You **must not** change the names of menus, menu commands, buttons, tables, columns, and the like that are specified below. This requirement is intended simplify and speed up marking and allow, for example, for tables to be prepared with test data in advance. If you change specified names, you **will lose marks**. Note that user-specified names of SQL tables and columns are case sensitive, but SQL keywords are not.

3.2 Tables

When your program executes, it must create these tables in your database if they do not already exist. Remember to check for errors when crating tables. If creation fails with the message indicating that the table already exists, you're good to go. However, if creating a table fails for some other reason (e.g. invalid SQL syntax), you must fix the problem! Also, make sure that your program **does not drop** any tables. Your code an create, edit, and clear tables, but it cannot drop them. If you need to drop a table, do so manually through the `mysql` command line tool.

We are interested in storing data about individuals and files containing the original data. The schema for your database consists of two tables named `FILE` and `INDIVIDUAL`. The data about every source file is stored in the `FILE` table, while the individuals refer to their source files by means of foreign keys.

Column names, data types, and constraint keywords are listed below:

Table `FILE`

1. `file_id: INT, AUTO_INCREMENT, PRIMARY KEY.` The `AUTO_INCREMENT` keyword gives MySQL the job of handing out a unique number for each organizer so your program doesn't have to do it.
2. `file_Name: VARCHAR(60), NOT NULL`. The value of the GEDCOM file.
3. `source`: `VARCHAR(250), NOT NULL`. The value of the source for that file.
4. `version`: `VARCHAR(10), NOT NULL`. GEDCOM version.
5. `encoding`: `VARCHAR(10), NOT NULL`. GEDCOM encoding.
6. `sub_name`: `VARCHAR(62), NOT NULL`. Submitter encoding.
7. `sub_addr`: `VARCHAR(256)`. Submitter address. NULL if missing.
8. `num_individials: INT`. Number of individuals in the file (`0` if none).
9. `num_families: INT.` Number of families in the file (`0` if none).

Table `INDIVIDUAL`

1. `ind_id: INT, AUTO_INCREMENT, PRIMARY KEY.`
2. `surname: VARCHAR(256), NOT NULL`. Individual's surname. For the sake of consistency with the parser library, store an empty string here if the surname is missing in the original file.
3. `given_name: VARCHAR(256), NOT NULL`. Individual's given name name. Again, store an empty string here if the surname is missing in the original file.
4. `sex: VARCHAR(1)`. The sex of the individual. NULL you did not implement this functionality in A3.
5. `fam_size: INT`. The individual's family size. Store `0` here is you did not implement this functionality in A3.
6. `source_file: INT`. The key of the source file for this individual. The `FOREIGN KEY REFERENCES` constraint establishes a foreign key to the `file_id` column over in the `FILE` table, and deleting the latter's row will automatically cascade to delete all its referencing individuals.
7. Additional constraint: `FOREIGN KEY(source_file) REFERENCES FILE(file_id) ON DELETE CASCADE`

4. Connecting to the SOCS database

4.1 Connection details

Our official SoCS MySQL server has the hostname of `dursley.socs.uoguelph.ca`. Your username is the same as your usual SoCS login ID, and your password is your 7- digit student number. A database has been created for you with the same name as your username. You have permission to create and drop tables within your own named database.

To access the database from home, connect to the school network using a VPN (Cisco AnyConnect). You will have three options:

1. Login from home using the `mysql` command line tool, if assuming you have it installed in your machine or VM. See Lecture 16 for details.

2. Login to linux.socs.uoguelph.ca, and use the `mysql` command line tool from there (it is installed)

3. Execute JavaScript code with SQL queries from your machine and look at the output.

Week 10 notes and examples have details for how to connect to a MySQL server, create/drop tables, insert/delete data, and run queries. This includes a simple JavaScript program that connects to the SoCS MySQL server and runs some queries (`dbTest.js`) .

4.2 Implementation details

The assignment solution must placed in the directory `assign4`, which must use the exact same structure as Assignment 3. The compilation process is the same as Assignment 3.

All the new UI functionality goes into `index.html` and `index.js`. All code for connecting to the DBMS server and interacting with the database must be placed into `app.js`. Your GUI client will interact with the server to load the data and run the queries. You will need to create additional routes on the server for this.

You will need to install the mysql for Node.js: `npm install mysql --save`. Make sure you include the `--save` flag - it will automatically update `packages.json` to include a reference to the `mysql` package. Again, remember that you must submit A4 without the `node_modules` directory, and we will install the Node modules by typing `npm install`.

You will develop your assignment using your own credentials and database, but part of the grading process will involve connecting your code to our database. This means that you cannot simply hardcode your credentials into `app.js`.

Your UI must include a pop-up a window that will ask the user to enter the username, password, and database name, and will attempt to create a connection. If the connection fails, your program will prompt the user to re-enter the username, DB name, and password.

5. Grade breakdown

• Correct database tables and database connection, obtaining the user's credentials:		15 marks
• Usability of UI additions, including error handling:		15 marks
• Each of the first three elements of the **Database** UI section, fully functional:		40 marks
• **Store All Files**:	30 marks	
• **Clear All Data**:	5 marks	
• **Display DB**:	5 marks	
• **Execute Query**, fully functional:		30 marks
• UI, two required queries:	15 marks	
• Three additional queries:	15 marks	

Deductions

As always, test your code on the SoCS servers using NoMachine and Firefox 52.6.0 - this is where it will be graded.

Any compiler errors will result in an automatic grade of **zero** (**0**) for the assignment.

Additional deductions include:

• Any compiler warnings:	**-15 marks**
• Any additional failures to follow assignment requirements:	**up to -25 marks**

- This includes creating an archive in an incorrect format, having your Makefile place the `.so` file in a directory other than `assign4/`, modifying the assignment directory structure, creating additional `.js` and `.html` files, etc.

6. Submission

Submit all your C and JavaScript code, along with the necessary Makefile. File name must be be `A4FirstnameLastname.zip`.

Late submissions: see course outline for late submission policies.

This assignment is individual work and is subject to the University Academic Misconduct Policy. See course outline for details)