CIS*2750
Assignment 4
Deadline: April, March 15, 11:59am

Weight: N/A
see COVID-19 update document for details

1. Introduction

This assignment builds on the SVG parser created in Assignments 1 and 2, and the Web app created in Assignment 3. This component requires you expand your A3 in the following ways:

- Add a UI panel for various database activities (see Sections 2 and 3)
- Use a database to track the changes to SVG images that we have been making in A3, as well as running some queries about them (see Sections 2 and 3)
- Create and maintain tables in an SQL database (see Sections 2 and 3)
- Add connectivity to a MySQL database (see Section 4)

2. Database Design

2.1 Naming conventions

You **must not** change the names of menus, menu commands, buttons, tables, columns, and the like that are specified below. Note that user-specified names in SQL (tables, columns) are case sensitive, but SQL keywords are not. This requirement is intended to 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**.

2.2 Tables

When your program executes, it must create these tables in your database - but only if they do not already exist. Remember to check for errors when creating tables. If creation fails with the message indicating that the table already exists, you are good to go. However, if creating a table fails for some other reason (e.g. invalid SQL syntax), you must fix the problem!

Another way to check if a table exists is with a select statement:
`select TABLE_NAME from INFORMATION_SCHEMA.TABLES where TABLE_NAME = 'MY_TABLE';`
where `MY_TABLE` is whatever table name you are looking for. If table exists, the query will return one row. Otherwise the result is empty.

Also, make sure that your program **does not drop** any tables. Your code can 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 files, routes, and points. Thus, the schema for your database consists of three tables named `FILE`, `CHANGE`, and `DOWNLOAD`. The idea is that every unique file is stored in the `FILE` table. Change and download logs refer to their source files by means of foreign keys.

Column names, data types, and constraint keywords are listed below. The foreign key constraints ensure that when we delete a SVG file, all of its changes and downloads are automatically deleted from their respective tables.

Table FILE

1. svg_id: INT, AUTO_INCREMENT, PRIMARY KEY. The AUTO_INCREMENT keyword gives MySQL the job of handing out a unique number for each file so your program doesn't have to do it.

2. file_name: VARCHAR(60), NOT NULL. The value of the SVG file.

3. file_title: VARCHAR(256). Value of the title element of the SVG image. May be NULL.

4. file_description: VARCHAR(256). Value of the desc element of the SVG image. May be NULL.

5. n_rect: INT, NOT NULL. Total number of rectangles in the file. May be 0, but not NULL.

6. n_circ: INT, NOT NULL. Total number of circles in the file. May be 0, but not NULL.

7. n_path: INT, NOT NULL. Total number of paths in the file. May be 0, but not NULL.

8. n_group: INT, NOT NULL. Total number of groups in the file. May be 0, but not NULL.

9. creation_time: DATETIME, NOT NULL. The time/date when the file was created. We count the time/date when the file was added to the database as creation time.

10. file_size: INT, NOT NULL. File size. May be 0, but not NULL.

Table CHANGE

1. change_id: INT, AUTO_INCREMENT, PRIMARY KEY.

2. change_type: VARCHAR(256), NOT NULL. Type of change. Contents can be things like "add attribute", "change attribute", "add circle", etc.. Exactly what you store here is up to you, but this information must support the search queries described in Section 3.

3. change_summary: VARCHAR(256), NOT NULL. Summary of the change, indicating what was done to the file, i.e. exactly what was changed or added - e.g. "change fill to blue for rectangle 3", "add circle at 2,3 with radius 6", etc..

4. change_time: DATETIME, NOT NULL. The time and date of the change.

5. svg_id: INT, NOT NULL. The file that was modified. FOREIGN KEY REFERENCES establishes a foreign key to the svg_id column in the FILE table. Deleting the latter's row will automatically cascade to delete all its referencing routes.

6. Additional constraint: FOREIGN KEY(svg_id) REFERENCES FILE(svg_id) ON DELETE CASCADE

Table DOWNLOAD

1. download_id: INT, AUTO_INCREMENT, PRIMARY KEY.

2. d_descr: VARCHAR(256). Additional information about the download, e.g. the downloader's IP address. What you put here is up to you. NULL if missing.

3. svg_id: INT, NOT NULL. The file that was modified. FOREIGN KEY REFERENCES establishes a foreign key to the svg_id column in the FILE table. Deleting the latter's row will automatically cascade to delete all its referencing routes.

4. Additional constraint: `FOREIGN KEY(svg_id) REFERENCES FILE(svg_id) ON DELETE CASCADE`

3. Database functionality and UI additions

To interact with the SVG database, add a **Database** UI section with the items described below. This section can be placed on the Web page below the A3 functionality. You can provide a separate sub-header for A4 section of the Web interface, to make it more visible. A proper Web app would use a less clunky solution, but we are trying to keep things simple.

Each UI item is only active when it is logically meaningful. For example, we cannot run queries if the tables are not created. We also cannot run queries if there are no SVG files on the server. After every command, except **Execute Query**, display up an alert with the status line based on a count of each table's rows, e.g.: "Database has N1 files, N2 changes, and N3 downloads" (how you get this information is up to you).

- **Login**: Your UI must ask the user to enter the username, password, and database name, and will attempt to create a connection. If the connection fails, your program must display an error (as an alert) and prompt the user to re-enter the username, DB name, and password.

  Until the user has logged in, all A3/A4 functionality should be disabled.

- **Store All Files**: This command is used to insert all the data from the files displayed in the File Log Panel into your database. This is active / visible only if the File Log Panel contains at least one file - i.e. there is at least one valid SVG file on the server in the `uploads/` directory. For every file, go through the following steps:

  - Obtain all necessary data that should be stored. By now, you shouldn't have to write any new C code, and can just use the functions from Assignments 2 and 3.

  - Check if the file name is already in the `FILE` table. If not, insert a new record for this file into the `FILE` table,.

  - Keep in mind that you cannot have changes or downloads with no files, so if you ever end up with an empty `FILE` table and non-empty `CHNAGE` or `DOWNLOAD` - something definitely went wrong!

- **Track downloads:** every time the user clicks on the download link for a file using the A3 interface, insert a record into the `DOWNLOAD` table. Make sure that the new record's `svg_id` correctly references the relevant in the `FILE` table.

- **Track file creation:** every time the user creates a new SVG file using the A3 interface, on the download link for a file, insert a new record record into the `FILE` table with appropriate data.

- **Track changes:** add logging of all changes to the file that you can make through the A3 interface: adding/ modifying attributes, modifying title or description, and adding new circles and rectangles. Every time the user performs one of these tasks using the A3 interface, insert a record into the `CHNAGE` table. Make sure that the new record's `svg_id` correctly references the relevant in the `FILE` table.

  Also, update the file size in `FILE` when the file is modified. You can reuse the A3 code you used to find the file size.

- **Clear All Data**: Delete all rows from all the tables. This may have to be done in a specific order due to the foreign keys. This is only active if tables have data in them. Do not drop the tables themselves.

  You do not need to update the File View Panel, or SVG View panel, since the files themselves are not deleted from disk.

- **Display DB Status**: This displays an alert with the status line described above: "Database has N1 files, N2 changes, and N3 downloads". Again, how you get this information is up to you - there are multiple SQL queries that you can use here.

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

Queries

The **Execute Query** menu item displays four 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 a SVG image database/editor, 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.

The result of the each query must be displayed as a table. The required queries are given below.

Queries:

1. Display all files. Include all data from `FILE` except `svg_id`. Your UI must allow files to be sorted either by name or by size.

2. Display all files created between specific dates. Include all data from `FILE` except `svg_id`. Allow the user to sort the results by file name, size, or creation date.

3. Display all files modified between specific dates. Include the most recent modification date, number of changes made to the file during that time, file name, and file size. Allow the user to sort the results by file name, size, or most recent modification date.

4. Display all files with shape counts in a specific range, e.g. 0-10 paths, 1-4 circles, etc.. Include shape counts in the results. The user must specify the type of shape and desired shape count range. Allow the user to sort the results by file name, size, or shape count.

5. Display N most frequently downloaded files, along with their summaries, download counts, and most recent download dates. Allow the user to specify N. Also allow the user to sort the results by file name, download count, or most recent download date.

6. Display all changes of a specific type to a specific file between specific dates. The output must contain all information relevant to the query - file name, change type, change summary, and change date. The user must have at least three different ways of sorting the query results by change date: change type, recent change fist, recent change last.

Helpful hints:
- You can use SQL to sort query results in the SQL query itself (lecture 14a). You can also do it on the JavaScript side - remember, a call to `connection.execute` returns an array of row data, which can be sorted using a predicate function. Pick whatever you think is easier for you.
- You might find that some queries can be done using either a single SQL complex statement, or multiple simple statements - e.g. run SQL statement 1, and based in its results run SQL statement 2. Again, pick whatever you think is easier for you to do.
- Make sure you carefully review Lecture 14b and the posted code example as you implement your new app.js routes, to make sure you use the Node.js `mysql2` package correctly. Pay close attention to the use of await keyword, especially you need to sequence multiple queries in the same function.

- For tracking downloads, the `onclick` attribute of the `<a>` HTML element is your friend, since you can assign a JavaScript function to it. You can make small changes to `index.html` to integrate this as long as the changes don't break anything.

4. Connecting to the SOCS database

4.1 Connection details

Our official 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)**, and

- Login from home using the `mysql` command line tool, if assuming you have it installed in your machine or VM. See Lecture 14b for details.

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

You can also login to linux.socs.uoguelph.ca, and use the `mysql` command line tool from there (it is installed)

Week 8 - 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 simple JavaScript programs that connect to the SoCS MySQL server and run some queries .

4.2 Implementation details

The A4 solution must be added to your `SVGApp/` directory.

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 endpoints on the server for this.

You will need to install the `mysql2` for Node.js: `npm install mysql2 --save` . Make sure you include the `--save` flag - it will automatically update `packages.json` to include a reference to the `mysql2` 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`. As mentioned in Section 2, your UI **must** ask the user to enter the username, password, and database name, before it attempts to create a connection.

5. Grade breakdown

- Correct database tables and database connection, obtaining the user's credentials: 10 marks
- Usability of UI additions, including error handling: 17 marks
- **Store All Files**: 15 marks
- **Logging downloads** 8 marks
- **Logging of changes** 15 marks
- **Clear All Data**: 3 marks
- **Display DB Status**: 3 marks

- **Execute Query**, fully functional: 29 marks
  - Required queries 1-2: 2 @ 4 marks each
  - Required queries 3-5: 3 @ 5 marks each
  - Required query 6: 1 @ 6 marks each

The UI will be expected to connect to a functional database backend.

Deductions

Marks will be deducted for buggy or broken functionality. To get a passing grade, your code must connect to the database, update tables, and run the queries. If the UI is not connected to the database backend, only minimal marks will be given for the UI that is not connected to the database. The usability marks will apply only if the UI is connected to the database. Unprofessional language is also included in this category, and will be penalized.

Usability

Your assignment is expected to comply with the usability principles that are covered in the lectures. Examples include, but are not limited to:

- Making sure the UI is consistent, and does not break the user's expectations
- Handling user errors in the UI
- Making UI output readable and formatted for a general user - error messages are not just raw SQL / JSON string dumps, etc..
- Making sure that the UI, where possible, prevents the user from performing invalid actions, or entering invalid data. For example, if a set of valid values is known in advance (e.g. file names), present them as a drop-down list, instead of forcing the user to type them in - and potentially allowing them to make mistakes.
- Using colours and fonts that are legible - avoid small fonts, make sure there's enough contracts between background and text, etc..
- Making sure the user always knows what happened, for any success or failure of a query - i.e. don't just display nothing if, for example, the query returns no results.

When in doubt, use the UI principles from the lectures (user interfaces, part 2).

As always, test your code on the SoCS servers. As with A3, the server for A4 must be developed and tested on is `cis2750.socs.uoguelph.ca`. This is where we will run it. The Web app will be accessed from Firefox on `linux.socs.uoguelph.ca`.

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 `SVGApp/`, modifying the assignment directory structure, creating additional `.js` and `.html` files, etc.
  - Unprofessional language is also included in this category, and will be penalized.

6. Submission

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

Late submissions: see course outline for late submission policies.

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