CIS*2750
Assignment 4

1. Introduction

This assignment builds on the iCalendar 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 Section 2)
- 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 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'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 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 calendar files, events, and alarms. Thus, the schema for your database consists of three tables named `FILE`, `EVENT` and `ALARM`. The idea is that every unique file is stored in the `FILE` table. Events refer to their source files by means of foreign keys. Similarly, alarms refer to their source events 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 calendar file, all of its events and alarms are automatically deleted from their respective tables.

Table `FILE`

1. `cal_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 iCalendar file.

3. `version: INT, NOT NULL`. iCalendar version for that file. Since our `CalendarToJSON` truncates the calendar version into an int, we might as well use INT data type here.

4. `prod_id: VARCHAR(256), NOT NULL`. The PRODID for that calendar (i.e. the tool that created it).

Table `EVENT`

1. `event_id: INT, AUTO_INCREMENT, PRIMARY KEY.`
2. `summary: VARCHAR(1024)`. The value from the SUMMARY property. NULL if missing.
3. `start_time: DATETIME, NOT NULL`. The value from the DTSTART property, converted into MySQL's datetime format.
4. `location: VARCHAR(60)`. The value from the LOCATION property. NULL if missing.
5. `organizer: VARCHAR(256)`. The value of the ORGANIZER property. NULL if missing.
6. `cal_file: INT, NOT NULL.` The file that originally contained this event. FOREIGN KEY REFERENCES establishes a foreign key to the `cal_id` column in the FILE table. Deleting the latter's row - will automatically cascade to delete all its referencing events.
7. Additional constraint: `FOREIGN KEY(cal_file) REFERENCES FILE(cal_id) ON DELETE CASCADE`

Table `ALARM`

1. `alarm_id: INT, AUTO_INCREMENT, PRIMARY KEY.`
2. `action: VARCHAR(256), NOT NULL`. The value from the ACTION property.
3. `trigger: VARCHAR(256), NOT NULL`. The value from the TRIGGER property.
4. `event: INT, NOT NULL`. This is a foreign key referring to the `event_id` column in the EVENT table. Deleting the latter's row will automatically cascade to delete all its referencing alarms.
5. Additional constraint: `FOREIGN KEY(event) REFERENCES EVENT(event_id) ON DELETE CASCADE`

3. Database functionality and UI additions

To interact with the calendar 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 is the tables are not created, or store events if there are no ical files on the server. After every command, except **Execute Query**, print in the Console Panel panel a status line based on a count of each table's rows, e.g.: "Database has N1 files, N2 events, and N3 alarms".

- **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 will prompt the user to re-enter the username, DB name, and password.
- **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 iCal 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, and add the appropriate records for its events and alarms into the `EVENT` and `ALARM` tables. Because of the foreign key constraints, you have to do this in the appropriate order. Use `NULL` for any missing fields.

- Keep in mind that you cannot have events with no files, or alarms with no events, so if you ever end up with an empty `FILE` table and non-empty `EVENT` - or empty `EVENT` and a non-empty `ALARM` - something definitely went wrong!

- **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 Calendar 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 N1 files, N2 events, and N3 alarms".

- **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.

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 a calendar file database (or just a calendar), 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.

Three 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.

You must supply an iCalendar demo file `A4demo.ics` that contains all the necessary calendar information that can be used to demonstrate your queries. In other words, if we upload `A4demo.ics`, we can execute any of the queries, and get meaningful (and non-empty) results for all of them. Failure to provide this file will result in loss of marks for Queries 4 - 6.

Queries:

1. Display all events sorted by start date. **(required)**

2. Display the events from a specific file. This query fetches the start dates and summaries of all events. How you sort the events is up to you. **(required)**

3. Display all events that might conflict with each other, because they happen at the same time. Include their summaries and organizers, if those fields are not null. **(required)**

4. - 6. Your choice, but at least one of them must be related to alarms. Remember, these must not be overly simplistic (see above)

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 15 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 9 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 `CalendarApp/` 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 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`. As mentioned in Section 2, your UI **must** ask the user to enter the username, password, and database name, and will attempt to create a connection.

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
- **Store All Files**: 30 marks
- **Clear All Data**: 5 marks
- **Display DB Status**: 5 marks
- **Execute Query**, fully functional: 30 marks
  - Three required queries: 15 marks
  - Three additional queries: 15 marks

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

Deductions

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 client 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 `CalendarApp/`, 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 `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)