CIS*2750
Assignment 4
Deadline: Thursday, April 7, 11:59pm
Weight: 25%

Introduction

This assignment has you explore some of the database concepts discussed in class and revisits some of the previous assignments, as well as key concepts covered in this course. Please note that, as discussed in class, the scope of this assignment has been changed due to technical issues with the cis2750 server. Assignment 4 is stand-alone, though it is still related to the topics of the previous assignments.

This assignment consists of two parts:

- In Part 1, you will need to create SQL code for creating and modifying tables with SVG file data. You do not need to connect the Web app created in A3 to the database, though you are welcome to explore this on your own.
- In Part 2, you will answer some general questions relating your A1-A3 code to some of the course learning outcomes and assignment requirements.

Part 1 – databases and SQL

1.1 Naming conventions

You **must not** change any of the names specified below. This requirement is intended simplify and speed up marking and allow for tables to be prepared with test data in advance. If you change specified table or column names, you **will lose marks**.

1.2 Tables

We are interested in storing data about files, as well as logging both changes to files and file downloads.. Every time a file is downloaded, we would store the date of the download and the host name of the downloader. Also, every time the file is modified, we store the date of the modification, as well as some additional details.

Thus, the schema for your database consists of three tables named `FILE`, `MODIFICATION`, and `DOWNLOAD`. Basic data about every unique file is stored in the `FILE` table. Change and download logs refer to the specific 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 name of the SVG file.
3. file_title: VARCHAR(256). Value of the title element of the SVG file. May be NULL.
4. file_description: VARCHAR(256). Value of the desc element of the SVG file. May be NULL.
5. 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.
6. file_size: INT, NOT NULL. File size. May be 0, but not NULL.

Table MODIFICATION

1. mod_id: INT, AUTO_INCREMENT, PRIMARY KEY.
2. mod_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. mod_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. mod_date: DATETIME, NOT NULL. The time and date of the change.
5. svg_id: INT, NOT NULL. The file that was modified. Must be a foreign key that references the svg_id column in the FILE table. Deleting the latter's row must automatically cascade to delete all its referencing modifications.

Table DOWNLOAD

1. download_id: INT, AUTO_INCREMENT, PRIMARY KEY.
2. d_time: DATETIME, NOT NULL. The time and date of the download.
3. d_hostname: VARCHAR(256), NOT NULL. Host name of the downloader
4. svg_id: INT, NOT NULL. The file that was downloaded. Must be a foreign key that references the svg_id column in the FILE table. Deleting the latter's row must automatically cascade to delete all its referencing downloads.

1.3. SQL code

You must submit the SQL code for creating and deleting each of these tables .

In addition, you must submit SQL code for four are required, three are up to you. 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 two required queries **are** overly simplistic, and are given to you as a warm-up.

You must supply SQL for creating and populating tables that contain all the necessary information that can be used to demonstrate your queries. In other words, if we create the tables using your statements, we must be able to execute any of the seven queries, and get meaningful - and non-empty - results.

NOTE: for the three custom queries, make sure you clearly describe what they are supposed to do. This description must be placed as a comment in the `query.sql` file (see Section 3 for submission details). For how to do comments in SQL, see online documentation (e.g. **https://www.w3schools.com/sql/sql_comments.asp**)

1. Display all columns from all files, sorted by file name **(required)**

2. Display all columns of all modifications, sorted by modification date (most recent first) **(required)**

3. Display names, sizes, and modification dates of all files modified between specific dates. You can hard-code specific start / end dates into your SQL code, but note that we may change these dates when we grade your code. The result must be sorted by file size. **(required)**

4. Display the name and download date of the most recently downloaded file. **(required)**

5. - 7. Up to you (see definition of non-simplistic queries above).

Part 1 questions will be worth 36 points in total. Grade breakdown summary:

- Creating tables: 6 marks
- Deleting tables: 2 marks
- Clearing tables: 2 marks
- Populating tables: 6 marks
- Queries, fully functional: 20 marks
  - Four required queries (2 marks each): 8 marks
  - Three additional queries (4 marks each): 12 marks

Part 2 - Coding Concepts Questions

Your answers to the following questions must be supported with examples from your own work in this course. Failure to provide examples from your own work will result in a grade of zero for the question.

2.1 Question 1

Select any two non-required functions (i.e. functions not explicitly listed as required in A1-A3 descriptions) of at least 20 lines of code from the code you have written for this course in this

semester. Your selections can be from any assignment. If you do not have two such functions in your work, then you should identify where you would refactor your existing code to include such functions and proceed as if the functions were written.

For each function:

- Provide the function signature
- Provide file name, line numbers, git repository name and branch to identify the function's location.
- Briefly describe the purpose of the function.
- Document a testing strategy for the function. You can describe what you did, or what you would do now.
    - As part of this documentation, describe the process of testing the function in terms of how you conducted the testing.
    - Detail any additional software you did or would write to accomplish the testing
    - Provide details about the test cases you used or would use.
    - Describe how you would record the testing results.
    - Describe how you would use the testing results.

Your answer to this question must be clear and concisely written in English. Spelling and grammar errors will not be penalized but lack of clarity will be penalized. If your answer cannot be understood easily you will lose marks. You may write in point form.

This question is worth 32 points, 16 points for each function.

2.2 Question 2

Reacquaint yourself with the following two functions: `bool validateSVG(const SVG* img, const char* schemaFile);` (A2 module 1) and `char* rectListToJSON(const List *list);` (A2 module 3). Their specifications are given below for your convenience.

****

```
bool validateSVG(const SVG* img, const char* schemaFile);
```

This function takes an `SVG` struct and the name of a valid SVG schema file, and validates the contents of the `SVG` struct against an XSD file that represents the SVG standard. It also validated the contents against the constraints specified in `SVGParser.h`. It returns `true` if the `SVG` struct contains valid data and `false` otherwise.

There are two aspects to `SVG` struct validity. First, its contents must represent a valid SVG struct once converted to XML. This can be validated using a method similar to what do in `createValidSVG`.

The second aspect is whether the `SVG` violates any of the constraints specified in the `SVGParser.h`. Some of these constraints reflect the SVG specification. For example, the SVG documentation states that a circle radius cannot be negative. However, validating a libxml tree against the SVG schema file will not catch this violation - as long as the radius is a valid number, libxml will consider the underlying XML document to be valid and fully compliant with the schema.

In addition, there are constraints that enforce the internal consistency of the data structures in the `SVG` - for example, all pointers in an `SVG` must be initialized and must not be NULL.

This means that `validateSVG` must manually check the constraints of the struct against the specifications listed in `SVGParser.h` - ensure that the numbers are within valid ranges, lists are not NULL, etc.

rectListToJSON

```
char* rectListToJSON(const List *list);
```

This function will convert a list of Rectangles into a JSON string. You can - and should - use `rectToJSON` function defined above.

The function `rectListToJSON` must return a newly allocated string in the following format:

```
[RectString1,RectString2,…,RectStringN]
```

where every `RectString` is the JSON string returned by `rectToJSON`, and `N` is the number of rectangles in the original list. The order of `RectString`s must be the same as the order of attributes in the original list.

For example, given the list with two rectangles:

1. a `Rectangle` created from
   `<rect x="1cm" y="1cm" width="19cm" height="15cm" fill="none" stroke="blue" stroke-width="1" />`
2. a `Rectangle` created from
   `<rect width="2" height="2"/>`

The corresponding string would be

```
[{"x":1,"y":2,"w":19,"h":15,"numAttr":3,"units":"cm"},
{"x":0,"y":0,"w":2,"h":2,"numAttr":0,"units":""}]
```

As before, the string above has no newlines; it is spread over multiple lines for readability. The actual string will contain no newlines or spaces, and look like this (sorry for the teeny font):

```
[{"x":1,"y":2,"w":19,"h":15,"numAttr":3,"units":"cm"},
{"x":0,"y":0,"w":2,"h":2,"numAttr":0,"units":""}]
```

The format **must** be exactly as specified. Do not add any spaces or newlines.

Do not modify the order of elements in the original list. Also, do not make any assumptions about the length of the list - it can contain any number of elements.

This function must not modify its argument in any way.

If the argument `list` is NULL, or an empty list, the function must return the string `[]` (there is no space there - just two chars).

****

For each of these two functions:

- Provide file name, line numbers, git repository name and branch to identify the function's location in your solution. You may use any of your available assignment solutions for this offering of this course.
- Cut/paste your implementation of the function into your answer.
- Annotate the cut/paste source code to identify the location in your solution that corresponds to each requirement in the specification for that function.
    - As part of the annotation justify your selection of that approach by explaining why it is correct.
- If you did not complete the function, then use your partial solution and detailed annotations to indicate where you would change or improve the function to meet each specification.

Your answer to this question must be clear and concisely written in English. Spelling and grammar errors will not be penalized but lack of clarity will be penalized. If your answer cannot be understood easily you will lose marks. You may write in point form.

This question is worth 32 points, 16 points for each function.

3. Submission format

- For Part 1, you must submit the following `.sql` files. These files must be placed in a Zip archive that should begin with your login id. For example, if your login id is `iamme34`, a suitable Zip file name would be `iamme34Part1.zip` or `iamme34SQLAnswers.zip`.

  The files are
  - `createTables.sql` - contains the SQL code for creating your tables
  - `populate.sql` - contains the SQL code for populating your tables with the data necessary to run all seven queries and getting non-empty results
  - `query.sql` - contains the SQL code for running all the queries
  - `deleteTables.sql` - contains the SQL code for deleting all your tables
  - `clearTables.sql` - clears contents of all your tables without deleting the tables

- For Part 2, you must submit a single PDF file with your answers to the next two questions. The name of the pdf file should begin with your login id. For example, if your login id is `iamme34`, a suitable PDF file name would be `iamme34Part2.pdf` or `iamme34Part2Answers.pdf`.

NOTE: You can run the SQL code contained in the `.sql` files using the `mysql` command line environment to test them - which is generally a good idea. All you need to do to execute a local `.sql` script on the remote server is to provide the login credentials on the command line:

- Without password on the command line:

```
mysql -h dursley.socs.uoguelph.ca -u username -p db_name < sql_file
```

  If you do this, you will be asked to type in the password before the script is executed
  This is the preferred way, since it avoids displaying your password

- With the password on the command line:

```
mysql -h dursley.socs.uoguelph.ca -u username -pyour_password db_name < sql_file
```