CS3101 – Databases

Assignment: P2 – Implementation

Deadline: 11 April 2022 *Credits*: 60% of coursework mark

MMS is the definitive source for deadline and credit details

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

Aim / Learning objectives

The aim of this assignment is to promote awareness of issues involved in using databases and database connections.

Requirements

You are required to produce a database and a simple graphical user interface for the scenario described in the Database Scenario section. You will need to create MariaDB tables, populate them, produce specified queries and procedures, enforce appropriate constraints, and produce a user interface for part of the system. Finally, you will compose a single word-processed document in PDF format, showing evidence of your practical work and justifying design and implementation decisions. The code, report and dump of the database must be submitted to MMS.

Details

1. **Create a MariaDB database and tables** corresponding to the schema described in the Relational Schema section and populate your tables with the sample data from the Sample Data section. You will need to choose appropriate SQL datatypes and integrity constraints, such as uniqueness, nullability and permissible values.
   The implementation must be on the School MariaDB servers. For information on how to connect, see the MariaDB & Host Services section.
2. **Specify each of the queries below in SQL** and run them against your database. Once you have finalised your queries, define views in the database to represent these queries and give them the names specified.
   1. List all courts with a grass surface. The query should return a single row per court, and should give the following information:
      Court number, Venue name, Address.
      The result should be ordered alphabetically by venue name, then court number.
      Save your query as a view named *view_grass_courts*.
   2. List the email addresses of all players along with their Elo ratings and the total number of matches they have won. The query should return a single row for each person, and should give the following information:
      Email address, Elo rating, Number of matches won.

    The result should be ordered by Elo rating in descending order, and then alphabetically by email address.
    Save your query as a view named *view_win_count*.
    Hint: you might want to create a function that calculates the winner of a match.
    3. List the names of everyone in the club, along with their email addresses and phone numbers. The query should return a single row per player, and should give the following information:

       Full name, Email address, Phone number(s).

       If a player has multiple phone numbers, they should all be included separated by commas. The type of number (home, mobile, work) should not be included. A player's *full name* is their first name, their middle names (if they have any) and their last name, concatenated together with spaces. For example, "Michael Colin Young" or "Frederick Threepwood". The result should be ordered alphabetically by surname, then by full name, then by email address.
       Save your query as a view named *view_contact_details*.
       Hint: the *CONCAT_WS* function might be useful for this query.
3. **Enforce each of the constraints below** using checks, triggers or procedures as appropriate.
    1. Ensure each *surface* attribute is equal to either "grass" or "clay" or "hardcourt". Only allow insertions into the *court* table that satisfy this condition.
    2. Ensure each phone number contains only digits from 0 to 9 and no other characters. Only allow insertions into the *player_phone* table that satisfy this condition.
    3. Ensure sets are valid, by preventing invalid sets from being added to the database. In each set, one player should win exactly 6 games, and the other player should win 4 games or fewer; or alternatively, one player should win 7 games and the other player should win 5 or 6 games. Sets should have set number 1, 2 or 3.
    4. Provide a procedure to add a new match together with its sets. The procedure should be called *proc_add_match*, and its inputs should be the email addresses of the two players, a date, a venue name, a court number, and the number of games won by each player in each of the 3 sets (if only two sets were played, the last two inputs should be NULL): 11 inputs in total. It should add the match to the *played_matches* table, then add the sets to the *played_sets* table, numbered 1, 2, and if appropriate, 3. Finally, it should update the Elo ratings of the two players involved, according to the formula given in the specification.
4. **Design and implement a user-friendly graphical interface** for your database. A user of your interface should not have to enter any SQL or read any SQL at any point. Users should only see relevant attributes. The interface should provide at least the following functionality:
    1. Permit the user to choose a venue and see all matches that have been played in that venue. Only matches played in that venue should be shown. Player names should be shown instead of player emails.
    2. Permit the user to create a new player in the database, with optional phone numbers, by entering the appropriate information in the GUI. If the submitted information violates the constraints of the database, an appropriate human-readable error message should be shown to the user.

    You may implement the GUI as a web interface or as a standalone application. You should use one of the following sets of technologies:
    1. HTML, CSS, javascript (optionally), and PHP
    2. HTML, CSS, javascript (optionally), and Node.js
    3. Java using Swing or JaxaFX

    You are permitted to use an ORM or similar, but you should document and justify its usage.
    If you want to use a different programming language and/or framework, **please ask us first**.

5. **Write a report** discussing your design and implementation. Try to concentrate on why you did something rather than merely on what you did. The document should explain and justify any decisions at the design and implementation stages and should contain headed sections for the following 4 elements:
    1. *Compilation, Execution & Usage Instruction*: A clear statement of what languages and frameworks have been used to implement the user interface. Instructions for compiling, running and using your interface.
    2. *Overview*: A short overview summarising the functionality of your implementation, including any extensions and interesting features, indicating the level of completeness with respect to the requirements listed above.
    3. *Database Implementation*: A summary listing the MariaDB tables, triggers, views, queries, functions and procedures in the database, along with a short description of the purpose, interesting features and implementation decisions associated with each. For each of the queries and views, include screenshots or textual output from executing them on your database.
    4. *GUI Implementation*: A summary listing the files for your GUI stating the purpose of each file, along with a **clear indication** of which files or code fragments you wrote from scratch, which you modified from examples that were given in class, and which you sourced from elsewhere.
       Include a discussion of the interesting features, design and implementation decisions you made when implementing your GUI.

   Please ensure that your screenshots are legible in the document you submit.

Extensions

Once you have built a basic working solution, fulfilling the functional requirements listed above, you may choose to design and implement one or more of the following extension elements:

1. Implement additional views, functions, procedures and triggers to improve the data consistency and ease of use of the database. For any such additional feature, explain the purpose and justify why it is interesting and useful, i.e. what features does it add to the system and what SQL features does it demonstrate that have not already been shown in your other queries.
2. Extend the GUI to include additional pages and/or functionality. Clearly document what additional features have been added and any changes that were required to the database structure.

Word Limit

An advisory word limit of 2000 words excluding references, figures and appendices applies to the report. A word count must be provided at the end of the document.

Submission

A *zip* file containing the contents listed below must be submitted electronically via MMS to the P2 assignment slot by the deadline. Submissions in any other format may be rejected.

- All the source files for your interface.
- A dump file for your MariaDB database.
    - For information on how to generate this see section MariaDB & Host Services.
- Your word-processed report as **PDF**.
    - Remember, the report should concentrate on why you did something as well as stating what you did. It should explain and justify any decisions at the design and implementation stages and should contain headed sections as described in section Details.

Assessment Criteria

Marking will follow the guidelines given in the school student handbook (see link in next section).

The following aspects will also be considered:

- Completeness of the implementation with respect to the requirements, including database, GUI and report.
- Quality of the database implementation – appropriate tables, attributes, datatypes, constraints, functions, procedures, views, etc.
- Quality of the GUI implementation – decomposition into appropriate functions/methods, classes; in-code documentation; etc.
- Quality of the report – depth and clarity of explanations; justification of design and implementation decisions for database and GUI; etc.
- Additional functionality or features

Policies and Guidelines

Marking
See the standard mark descriptors in the School Student Handbook:
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness penalty
The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good academic practice
The University policy on Good Academic Practice applies:
https://www.st-andrews.ac.uk/students/rules/academicpractice/

MariaDB & Host Services

The School provides every user with access to a host server (*username*.host.cs.st-andrews.ac.uk) running a MariaDB service. To use the service, you must ssh into your host server[1], retrieve your MariaDB credentials, create a database, and select it for use.
You can find detailed information about how to do this, and about the MariaDB service more generally, on the Systems wiki at:

https://systems.wiki.cs.st-andrews.ac.uk/index.php/Linux_Host_service#MariaDB

You can also find a summary of how to connect to and use your MariaDB database in the CS3101 Lab sheet:

https://discourse.cs.st-andrews.ac.uk/t/exercise-4-lab-session-sql/1310

as well as in the first demo recording:

https://discourse.cs.st-andrews.ac.uk/t/lecture-13-database-connections/1300/6

To generate a dump file for a database you must ssh into your host server, change to the directory you wish to create the dump file in, then run the following command:

```
mysqldump --routines -p database_name > filename.sql
```

with `database_name` replaced with the name of the database you want to dump (e.g. mct25_CS3101_P2), and `filename.sql` replaced with the name of the file you want to dump

[1] Note: logging into a lab machines is **not** the same as logging into your host server. You must ssh into your host server *username*.host.cs.st-andrews.ac.uk, where *username* is your username.

the database to (e.g. mct25_CS3101_P2_dump.sql). You will be prompted to enter you MariaDB password.

If successful, the dump file should contain all the SQL DDL statements required to recreate your database, including definitions for tables, view, procedures, functions, triggers, as well as statements to load all the data. You can check this by opening the file in any text editor and examining the contents or by loading the dump file into a new database.

Database Scenario

A local tennis club wants you to build a system to manage their data. It should include all information about their players, venues, courts, leagues and matches, as described below.

Tennis is a two-player racquet sport, similar to squash or badminton. A tennis *match* consists of several *sets*. The first player to win 2 sets wins the match. The results of all club matches are recorded and stored, along with the date played, court used and league the match was a part of. The club only plays *singles* games (one player vs one player).

Each set consists of several *games*. The two players play successive games, and the first player to win 6 games wins the set. However, the rules state that a player cannot win 6–5, so if this score is reached the set continues until one of the players reaches 7. If the final score of the set is 7–6, then this is known as a "tie-break set". The system does not record any details of the points scored in individual games.

A match is played on a *court*, an outdoor rectangle with a net and markings on the floor, designed especially for tennis. The surface of a court can be made of grass, clay or hardcourt; the different surfaces cause the ball to bounce differently. The tennis club allows matches to be played at several *venues*, each of which has an address and several courts numbered starting at 1. Players choose where they want to play each match by mutual agreement.

The club keeps a list of members (*players*) who are allowed to participate in its matches. The club insists on having an email address for each player, as well as their full name and date of birth. The club may also have one or more phone numbers for each player, to make it easier for members to contact each other to arrange matches. Phone numbers are marked as being for home, work or mobile, so that players can judge an appropriate time to call.

To help judge how strong a player is, the club wishes to use an *Elo rating system*. Each player has an *Elo rating*, which is an integer that starts at 1000 for new members. If a player wins matches, their rating goes up; and if they lose matches, their rating goes down. To reward players more for beating players with a higher rating, the change in rating after a match is defined by a formula that uses the current ratings of both players. After a match, the ratings change C is given by:

C = 40 * (1 - 1/(1+10^((L-W)/400)))

where L is the current rating of the losing player, and W is the current rating of the winning player. After C is calculated, the winning player gains C rating points and the losing player loses C rating points (rounded to the nearest integer). The tennis club has never used this system before, but is keen to use it now that a database is being created.

Relational Schema

This is the relational schema which you should follow when implementing the tennis club's database. You should choose appropriate data types and constraints in SQL when converting, and consider carefully what the foreign key constraints (marked with *) should reference.

- player = (
  email: String,
  forename: String,
  middlenames: String,
  surname: String,
  date_of_birth: Date,
  elo: Integer
)
- player_phone = (
  email*: String,
  phone_number: String,
  phone_type: "home" or "mobile" or "work"
)
- played_match = (
  id: Integer,
  p1_email*: String,
  p2_email*: String,
  date_played: Date,
  court_number*: Integer,
  venue_name*: String
)
- played_set = (
  match_id*: Integer,
  set_number: Integer,
  p1_games_won: Integer,
  p2_games_won: Integer
)
- venue = (
  name: String,
  address: String
)
- court = (
  number: Integer,
  venue_name*: String,
  surface: "grass" or "clay" or "hardcourt"
)

Sample Data

For several years, all the information for the club has been stored in a shared spreadsheet which everyone can edit (tennis_club_data.ods). The information has been recorded successfully in the spreadsheet, but it's not normalised or organised very well, so you will need to manipulate, normalise and sanitise the data prior to inserting into your database.

You can find the spreadsheet here:

https://studres.cs.st-andrews.ac.uk/CS3101/Practicals/P2/tennis_club_data.ods

Note: All the rows in the spreadsheet showing matches have players listed using only their first name – this is okay so far, because everyone in the club at the moment has a unique name, but in the future this might not be the case, so the club has asked you to use email addresses to identify individual people instead.