CS3101 Databases
Practical 2 – Implementation

School of Computer Science
University of St Andrews

Due Friday 5th April, weighting 60% of coursework
MMS is the definitive source for deadlines and weightings.

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.

   a. Produce a table for conveniently looking up a species' scientific name by its common name. The query should return one row per common name, and should contain the following columns:

Common name, Scientific name

where "Scientific name" is the full scientific name formed from the genus name and species epithet. The result should be ordered alphabetically by common name. Save your query as a view named `view_common_name_lookup`.

b. List the *average* latitude and longitude of sightings by each user, to give an indication of whereabouts in the country they do most of their birding. The query should return one row per user, and should contain the following columns:

   Display name, Home latitude, Home longitude, Average latitude, Average longitude

   The result should be ordered by average latitude from north to south. Save your query as a view named `view_average_location`.

c. List the species of crow that have so far been spotted by users, along with the number of times they have been spotted. The query should return one row per spotted crow species, and should contain the following columns:

   Common name(s), Scientific name, Sightings

   If there are several common names, they should be listed together separated by a forward slash symbol. "Scientific name" should be the full scientific name formed from the genus name and species epithet. A crow is any species that belongs to the family "Crows". The result should be ordered by number of sightings from highest to lowest, then by scientific name. Save your query as a view named `view_crow_frequency`.

3. Enforce each of the following constraints. You may use checks, triggers, procedures or any other appropriate mechanism.

   a. Ensure that a user's email address is no more than 320 characters long and contains an `@` symbol.

   b. Ensure that latitude and longitude values are always in the appropriate range as described in the Database scenario section.

4. Provide the following functionality using functions and procedures.

   a. Provide a function to find the distance in kilometres between two locations. The function should be called `func_earth_distance`, and it should have precisely 4 inputs: the latitude ϕ_1 and longitude λ_1 of the first location, and the latitude ϕ_2 and longitude λ_2 of the second location.

      This is a rather complicated thing to work out mathematically, but it is possible using the Haversine formula. After converting all the inputs from degrees to radians, calculate the internal angle A using the formula

      $$A = (\sin((\phi_2 - \phi_1)/2))^2 + \cos(\phi_1)\cos(\phi_2)(\sin((\lambda_2 - \lambda_1)/2))^2.$$

      Then the distance is equal to

      $$2r \cdot \mathrm{atan2}(\mathrm{sqrt}(A), \mathrm{sqrt}(1 - A))$$

      where r is the average radius of the earth, 6371 kilometres.

   Note: This won't be very accurate for short distances, but if you use the DOUBLE type for intermediate variables in your function, it should be a reasonable approximation of long distances.

   Note: Recall the SQL built-in functions `PI`, `SQRT`, `SIN`, `COS` and `ATAN2`, any of which might be useful.

   b. Provide a procedure to add a new user to the `user` table. The procedure should be called `proc_add_user`, and it should have precisely 5 inputs: the person's email address, display name, home latitude and longitude, and the hash of their password (in this order). The user's salted hashed password should be computed and stored, as described in the Passwords section.

   c. Provide a function to verify a user's email and password when they attempt to log in. The function should be called `func_valid_credentials`, and it should have precisely 2 inputs: an email address and a hashed password. It should do any appropriate hashing and return a boolean that states whether that email/password combination represents a valid login.

5. Design and implement a user-friendly graphical interface for the system, which interacts with your database. A user of your interface should not have to enter any SQL or read any SQL at any point.

   Before doing anything else, a user must log in with a correct username and password. After successfully logging in, the user should be able to view records of all their own sightings, add new ones and delete existing ones. They should also be able to change their display name.

   A user should not be able to see any sightings belonging to other users.

   If any submitted information violates the constraints of the database, an appropriate human-readable error message should be shown to the user. The user should never have to think about the implementation of the database, or even know that it uses SQL.

   You **must** implement the GUI as a Java application. Some starter code is provided on StudRes in the `gui` directory, which you may alter as much as you wish. Usage instructions are included in the `README` file.

   It is very important that you include instructions on running your GUI from the command line. Update the instructions in the `README` if they are not sufficient for your program by the time it is finished.

   Note: While user-friendliness is important, the visuals of the program are not an important part of this submission. We're not interested in whether the UI looks good, so long as it has good functionality and error handling, presents the data clearly, and allows the user to conduct tasks without unnecessary extra work.

6. 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:

   a. **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.

b. **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.

c. **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.

d. **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. There is no need to mention the starter code.

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 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: 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?

Extending the GUI to include additional user-facing functionality is not recommended. It would be better to ensure that the required features of your GUI are implemented cleanly, with a simple user-friendly interface, excellent error handling and elegant code.

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 (see MariaDB & Host services below).
- Your word-processed report as a PDF.

Remember, the report should concentrate on why you did something as well as stating what you did. It should explain and justify important decisions at the design and implementation stages and should contain headed sections as described in the Details section.

Assessment criteria

Marking will follow the guidelines given in the school student handbook (see link in the Marking 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: https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness penalty

The standard penalty for late submission applies (Scheme A: 1 mark per 24-hour period, or part thereof): https://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.teaching.cs.st-andrews.ac.uk) running a MariaDB service. To use the service, you must ssh into your host server, 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 School wiki at https://wiki.cs.st-andrews.ac.uk/index.php?title=Teaching_Service#MariaDB

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 the database to (e.g. `mct25_CS3101_P2_dump.sql`). You will be prompted to enter your 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 small online community of bird enthusiasts wishes to set up a simple birdspotting application, with a connected database, to store and browse information on users and the birds they have spotted.

For verification and reference, the system stores information about all 268 species of British birds. Each species has a unique **scientific name**, which consists of two parts: the **genus name** and the **species epithet**. For example, in the scientific name *Falco peregrinus*, the genus name is *Falco* and the species epithet is *peregrinus*. Each species also has one or more **common names**, which non-scientists are more likely to use. Each bird belongs to one of the 55 bird **families**, such as "Finches" or "Gulls and terns".

Users should of course be tracked in the system. Each user has a unique **email address**, which can be used to contact them with news and updates. A user also has a **display name**, which can be shown to other users; and a **password**, which is used along with their email when they log into the system's GUI. Finally, a user can set their **home location**.

The main purpose of the system is that users can log birds that they have spotted. Each **sighting** records the email address of the user that spotted it, the scientific name of the bird identified, and the date and location that the sighting took place.

All locations are stored as two numbers: a **latitude** and **longitude**. Latitude is in degrees north of the equator, and longitude is in degrees east of the prime meridian. Each value can be negative, and must fall in the range

$$-90 < x \leq 90.$$

For privacy purposes, the database will only store these values with a precision of 2 decimal places.

Passwords

It is considered bad practice to store plain-text passwords in a database, since a data breach would reveal a user's password, which they might have reused for other services. Instead, *hashing* is used so that the database only stores a value computed from the password in a non-reversible way.

When a user enters their password p into the GUI, the application should **hash** the password using the SHA-256 algorithm, to produce a string h which is then sent to the database. A method for this has been provided in the starter code.

To make things even more secure, the database does not even store the hashed password h. Instead, it stores a **salted** version of it: the database takes h, appends an arbitrary string such as `BIRDS` to the end of it, and then hashes it again with the SHA-256 algorithm to produce a salted hash s which is stored. This way, a data breach would not reveal the user's password or even its hashed version. When a user logs in with their hashed password h, the process can be repeated and the salted version s can be compared with the value for s stored in the database.

Stored functions and procedures should be included in the database to make this easier.

Relational schema

This is the relational schema which you should follow when implementing the birding 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.

Make sure to use the **exact names shown** for tables and attributes in your database.

Primary key attributes are shown in **bold**.

family = (**name**: String)

species = (**genus_name**: String, **species_epithet**: String, family_name*: String)

common_name = (**genus_name***: String, **species_epithet***: String, **common_name**: String)

user = (**email**: String, display_name: String, home_latitude: Number, home_longitude: Number, salted_password_hash: String)

sighting = (**id**: Integer, user_email*: String, genus_name*: String, species_epithet*: String, date_spotted: Date, latitude: Number, longitude: Number)

Sample data

Until now, the users of the system have been storing all their data in a single spreadsheet (`all-data.ods`) to which they all have write access. This spreadsheet contains a total of 5 sheets, which includes everything you need to populate the database with.

The information has been recorded successfully in the spreadsheet, but it's not normalised or organised very well, and there are various notes and unnecessary pieces of information that aren't needed for the database. 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/Coursework/P2/all_data.ods