CMPUT 291 - Mini Project 1 Design Document

1 Overview

The following python application is used to connect to an existing SQLite3 database storing enterprise data. The application is run through the terminal, and is operated with simple command line inputs. For information on connecting to a database and running the application, see *User Guide*.

Users with valid login credentials will be provided access to various functions to interact, view, and update the data stored in the database. Depending on their user type (determined by the database) the functionalities include: *Register a birth, Register a marriage, Renew a vehicle registration, Process a bill of sale, Process a payment, Get a driver abstract, Issue a ticket, Find a car owner*. For specifics on these functions and their operations, see *Software Design*.

1.1 User Guide

To launch the application, run the following command from the directory holding the source code:

```
    $ python3 db.py
```

To connect to a database, enter the path to the database when prompted (a):

```
    Enter path of database: path/to/database.db
```

Once the database connection is successfully established, the user will be prompted to enter their username and password (b), for example:

```
    Username: admin123
    Password: pwd12345
```

Depending in the user type (agent or officer), numerical options will appear on screen. Enter the number of the task you wish to perform, and follow the on-screen instructions. For further information regarding the options, see *Software Design*.

At the main user menu, the user can logout by entering the '0' key, which will return the application to the login screen. The application will remain connected to the database entered at launch, and other users are able to login from here.

The user can return to the main menu from any point within performing a task by hitting ctrl-d. Note: if a new entity was committed to the database or before hitting crtl-d, **this action will be saved in the database**, i.e. if a parent needs to be registered in order to register a birth, and this step is completed, and the user exits to the main menu before completing the birth registration process, the newly registered parent will remain in the database.

At the login in screen, the application can be exited by pressing crtl-d.

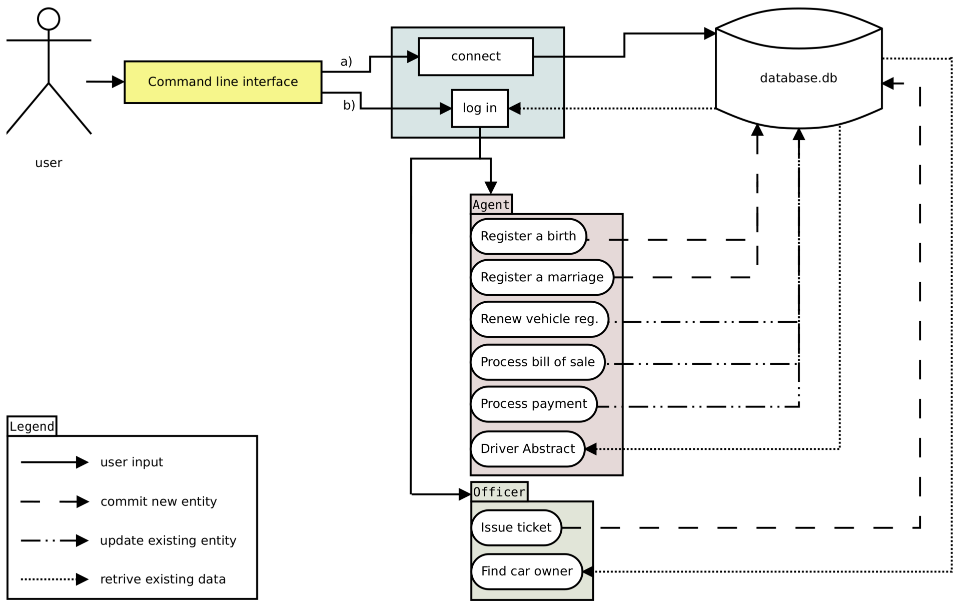

Figure 1: Diagram showing flow of data between application user, application functions, and the connected database.

2 Software Design

- User Menu [`display_menu(utype)`]

  After a successful log in, the user is provided a menu with an enumerated list of tasks they can perform, depending on their `utype`. By entering the number corresponding to a task and hitting the ENTER key, the program will display a new page with on-screen inputs and instructions on how to complete the task.

- Register a person [`insert_person(fname, lname)`]

  Registers a new person into *persons* table. If `fname, lname` are not provided as arguments, user can enter this information. Otherwise, names provided as arguments will be used.

- Register a birth [`register_birth(user_info)`]

  Registers a new person into *persons* table and a new birth record in *births* table with a unique registration number generated by the program. Prompts user to enter child's first and last name, birth gender, mother's first and last name, father's first and last name, and the child's birth date and birth place. Registration place is automatically set to the user's city (from `user_info`), and registration date is set to today's date. Child inherits mother's address and phone number, which is obtained from a query within the function. In the case where the mother or father do not exist in the database, the user has the option to register them, which calls `insert_person(fname, lname)`

- Register a marriage [`register_marriage(user_info)`]

  Registers a new marriage record; registration numbers automatically generated by the program. User enters both partners first and last name. If partner is not found in database, user can register them first, calling `insert_person(fname, lname)`. Registration place is determined by `user_info`, and date is set to today's date.

- Renew a vehicle registration [`renew_reg()`]

  Receives a registration number from user input, then sets the registration date to the current date and the expiry to be one year after.

- Process a bill of sale [`bill_of_sale()`]

  Prompts user to enter a vin, current and new owner for a vehicle, the creates a registration for the new owner. The system also expires the old registration. The system allows the user to re-enter any incorrect information.

- Process a payment [`process_payment()`]

  Receives a ticket number and payment amount from the user. The system records the payment into the database. The user is not allowed to enter a payment amount that will exceed what is due on the ticket.

- Get a driver abstract [`get_driver_abstract()`]

  Prompts user to enter first and last name of a registered person to display their driver abstract. If the name entered is not found, the user has the option to renter the name, or return to the main menu. The driver abstract consists of: ticket counts for lifetime and last two years, demerit notices and points for lifetime and last two years. The agent has the option to view a detailed ticket history by entering the t/T key. Initially five tickets are displayed in order from most recent to oldest, with an option to display 5 more until the end of the history.

- Issue a ticket [`issue_ticket()`]

  Creates a new ticket entry in *tickets* table. User will be asked to enter a valid registration number; if number is invalid user will be returned to menu. If a valid registration number is entered, registration details are displayed to the user. From here, they enter violation date (defaults to today's date if field left blank), description, and fine amount. A unique ticket number is generated by the program by taking the largest existing number and incrementing it by one.

- Find a car owner [`find_car_owner()`]

  Prompts user to optionally enter the make, model, year, colour and plate of a vehicle. Matching vehicles are found and displayed with their make, model, year, colour and plate. The user may choose a vehicle to view the registration date, expiry date and the name of the owner. Alternatively, if there are less than 3 matches the additional information is displayed automatically.

3 Testing Strategy

Functions of the software were tested as they were developed, and then again when the entire development process was completed. For each of the functionalities, the following cases were tested:

- *Register a birth*: Both parents in db; only one parent in db; no parents in db; mother with Null values
- *Register a marriage*: partners not in db
- *Renew a vehicle registration*: regno not found
- *Process a bill of sale*: regno, current and/or new owner not in db
- *Process a payment*: invalid tno; invalid payment amount (i.e. over paying a ticket)
- *Get a driver abstract*: driver not found in db
- *Issue a ticket*: regno not found; violation date specified; violation date not specified
- *Find a car owner*: No info entered, less than 4 results, more than 4 results

4 Group Break-down Strategy

The project was developed using github to track progress and coordinate work on implementing the features outlined in the spec. A detailed breakdown of the group work on functions and features follows:

- main, database connection, user login, main menu [`main()/connect_to_DB()/get_login()/display_menu(utype)`]
  - Dalton & Harrison: collaborative, 1.5 hrs
- SQL injection prevention and non-visible password
  - Harrison & Dalton: collaborative, 0.5 hrs
- Register a birth [`register_birth(user_info)/insert_person()`]
  - Harrison & Dalton: collaborative, 1 hrs
- Register a marriage [`register_marriage(user_info)`]
  - Harrison & Dalton: collaborative, 0.5 hrs
- Renew a vehicle registration [`renew_reg()`]
  - Harrison & Dalton: collaborative, 0.5 hrs
- Bill of sale [`bill_of_sale()`]
  - Amir: 2 hrs
  - Harrison: format changes, 0.5 hours
- Process a payment [`process_payment()`]
  - Harrison & Dalton & Amir: collaborative, 1 hrs
- Get a driver abstract [`get_driver_abstract()/ticket_report()`]
  - Dalton: 3 hrs
- Issue a ticket [`issue_ticket()`]
  - Harrison & Dalton & Amir: collaborative, 1 hrs

- Find a car owner [`find_car_owner()`]
  - Amir: 8 hours
  - Harrison & Dalton & Amir: collaborative, 1.5 hrs
- ctrl-d escape
  - Harrison: 1 hrs
- Design Document
  - Dalton: format, flow diagram, overview, user guide, 3 hrs
  - Harrison & Dalton & Amir: collaborative, 1.5 hrs