Registry Agents' Operations:

- Register a Birth
- Register a Marriage
- Renew a Registration
- Process a Bill of Sale
- Process a Payment
- Get a Driver Abstract

Traffic Officers' Operations:

- Issue a Ticket
- Find a Car Owner

This design document includes:

1. A general overview of the system with a small user guide.
2. A detailed design of the software with a focus on the components required to deliver the major functions of the application.
3. Our testing strategy.
4. Our group work break-down strategy.

General Overview & User Guide:

Project Name: Alpha

This program is used by Registry Agents and Traffic Officers as a virtual information registry for people in a small city. Registry Agents and Traffic Officers have operations that are specific to them, and are listed above. Entering a proper username and corresponding password (Stored in the users SQL data), allows the user to log into the main user interface. The user interface comprises of the various functions the user is able to do based on his profession. There is one number that corresponds with each function (Ex: If a registry agent logs in and inputs 1, he gets the Register a birth function as a result). Regardless if the user had successfully executed one of these functions correctly, they are always brought back to the main interface, where they are able to do various functions again indefinitely. Once the user has finished using the program, they are able to logout by inputting q to close the program, where they will return to the login screen, inputting q again will close the python program entirely.

Detailed Design of Software:

When analyzing our software in a more in-depth level, it is clear that all of the main functions are based off of one function in system.py. This function is the login_screen() function, in which it checks for a valid user in the database by requesting the user_id and password as inputs and searches the users table for a possible match. The reason why this function is so important is that it returns the type of user, this is very important for determining what to show the user based on the type he/she is, and what functions the user may be allowed to access. The functions that are available for the user is specified in the operation_choice() function, it takes in the type of user as a parameter and makes use of if statements to determine what to show to the user. This function is where we first get access to the main functions that alter the data tables in SQL. We will now go into further detail into all functions that serves the specifications of the assignment. For a registry agent, we have the following functions:

register_birth.register_a _birth(city):
This function comes from the register_birth file that we imported into system.py. It consists of two functions, the register_a_birth ensures if the user really wants to register a birth, and if so executes the cursor with the values returned by the getBirthInfo function. The getBirthInfo requests the first name, last name, gender, a properly formatted birth date, and birthplace as input, and all but the first and last name can be null values. Additionally, the program requests the father's full name along with the mothers, and should the names not be in persons table, we use the persons class to create a person through the createPerson method, that essentially asks the user to fill the values of the persons table. The getPerson method serves to check if a person with the given first name and last name as arguments is in the data, and returns a truth value. Then, the program will get the city of registry by the city argument passed in, it will then use the datetime module to get the current date for the registration date, and also gets a unique regno by using the getUnique() method from the persons class where it returns a uniquely generated registration number. Then we return the filled values back to register birth function and inserts the newly created birth into the data.

register_marriage.register_marriage(city):
This function comes from the register_marriage file that we imported into system.py It is a single function that ensures if the user really wants to register a birth. It requests the full names of both partners and checks to see if they are in the database, if either one is not present then we create the person in the same way we did for creating parents in register_birth, we also get the unique regno, regdate, and the city of registration in the same way as register_a_birth.

renew_registration.renew_registration()
This function comes from the renew_registration file. It is a single function that requests a valid regno, if it does not exist then the function ends, indicating that you did not enter a valid regno. If you want to renew the regno, inputting Y/y will compare the expiry date of the regdate and today's date, if the difference is less than 0 or equal then we set the new expiry date to today after a year, otherwise we extend the expiry date to a year. We then update the regno on the data table.

Bill_of_sale.bill_of_sale()
The function comes from the Bill_of_sale file. It is a single function that requests a valid vin, if there is corresponding data, the vin, fname, lname, regdate is taken. The function then requests the current owner's full name, and checks to see if the name matches the most recent registration. If there is no match, then the function returns false. The full name of the new owner is requested, if there is no match in the registration table, then we return false and print that there is no such owner in the database. A plate name is then requested, and if the user enters a string longer than 7, we return false, stating that they went over the limit of length. The current date is then acquired by the datetime module, and the date of the current date exactly a year from now is also brought, and the previous owner's expiry date is updated to be today, and a new row is inserted with the information gathered earlier (vin, owner full name, plate, etc…)

process_payment.process_payment()

The function comes from the process payment file. It is a single function that requests a valid ticket number, the program stops if the user does not enter a ticket number in the database. The program then takes all of the existing payments from the payments table and adds up any previous payments together and stores it. If the total previous payments add up to the fine, the program states that the ticket is already paid and exits the program. Otherwise, the user is then prompted to pay the fine, if they pay too much for the fine the transaction does not happen, If they pay properly then the payment is processed into the payments table. If the fine is fully paid an indication is given, otherwise the remainder of the fine is showed.

get_drivers_abstract.get_drivers_abstract()
The function comes from the get_drivers_abstract file. It is a single function that requests a full name, if the name is in the database, then we select the count of distinct tickets and demerit notices using the joining of demeritNotices with registrations using the full name, then joining that with tickets using regno. We specify all the values where the full name is in it in order to count. This process is repeated similarly for gathering the count of distinct tickets and demerit notices over the span of 2 years, only that we specify the date being less than 2 years old. We then take the sum of the points of all the appropriate demerit notices with both time conditions being considered. Finally, we select the tno, vdate, violation, fine, regno, make, and model to display the specific ticket descriptions. Initially, we display the abstract of both the lifetime and the past 2 years. After that the first 5 recent tickets displayed, with the additional option of inputting y to see the rest of the tickets.

For the traffic officer, he/she has access to 2 functions specific to their role, issue_ticket() and find_car.

find_car_owner.find_car_owner()
The function comes from the find_car_owner file. It is a single function that requests the user to input the make, model, year, color, and plate of a vehicle that is in the database, however not all are needed in order to return results. The function then returns all matching vehicles with the inputted values, including a count of how many results, if the count is greater or equal to 4, only the make, model, year, plate, and color gets printed and a number is associated with each result. If one is to type in a number that corresponds with a certain row, then more details are given. If there are 3 or less matches then more details are given (regdate, expiry, etc…)

issue_ticket.issue_ticket()
The function comes from the issue_ticket file. It is a single function that requests a valid regno, and returns false if no match is to be found. Information of the row corresponding with the regno can be found is given, and the user is prompted to enter the date of offense, the offense, and the fine for the offense. A ticket number is automatically generated, and the ticket is recorded into the database.

Testing Strategy:
Our methods of testing came in various forms. One method occurred more frequently, which is when one of the members of our team reviewed another peer's code and asked questions based on the structure of the code. Specifically we would ask how it would be possible to run into an error, and if we discovered a possible scenario where the program would catch an error, the code gets revised to handle the scenario. This applied mainly on the times where we would ask the user to input, since we sometimes converted the input as an int in order to find a regno for example. Another form of testing was when all members met up in a computer lab and ran the program in its entirety for any other bugs or errors that we may have missed. Logging in with a valid username and password was tested here, as well as preventing injection, the functionalities for each user type was also tested, which includes testing our error handling methods. The sql data was also examined in this phase, ensuring that the sql executions were actually giving the results that we desired. Case insensitivity was also tested here as well. Typical bugs discovered were mainly around buggy SQL statements, improperly formed if statements and loops, input not being handled correctly, and injection attacks succeeding. Although an actual count of the occurrences of bugs were not recorded, generally speaking they did not happen too often.

Group Work Breakdown Strategy:
As a representation of our group work, we are uploading our GitHub link here:
https://github.com/pickabo/Cmput291assignments.git

This mini project was worked on by Jawad Rizvi, Nasif Hossain, and Yazan Al-Muhtaseb. Coordination was through a group chat on WhatsApp, where we discussed meetups, troubleshooting, and so on. The work was broken up by functions, and some broader tasks like testing and documentation were distributed amongst members:

- Yazan was responsible for the documentation, and created the rough sketch of functions 2, 3, and 5. He spent roughly 15-20 hours on the assignment.
- Nasif was taking care of testing and implementing functions 1, 2 and 6 of registry agents as well as countering the SQL injection. He spent roughly 20 hours of work on the assignment.
- Jawad managed data.sql, designed system.py, and implemented function 4 from registry agents as well as functions 1 and 2 from traffic officers. He spent roughly 20 hours of work on the assignment.