Charlie Bauchop - Documentation and Written Component

Purpose and End-Users:

The purpose of this database is to be usable for a school to handle all of the data that they need to use in order to create and manage their school systems. The end users are anyone who will need to use this database for their school systems, from the students and teachers themselves to any principals and school administrators who would use this database for any reasons necessary. In order to make this database suitable for the end user I need to allow easy access to the important commands and queries to the database, and output the results in a clear and systematic format. All the decisions I have made allow the end user to use the system in a clear and understandable manner.

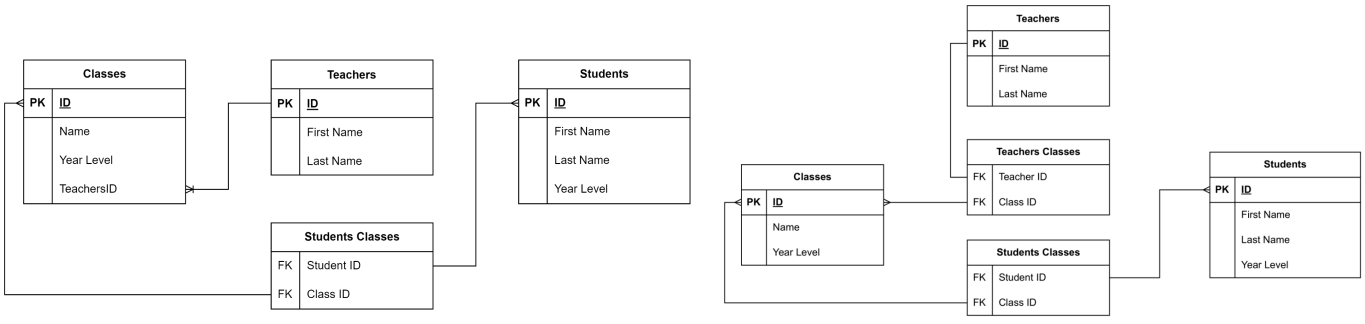
Design:

The design of the interface is a text based terminal using python user input and outputting using the tabulate package, which formats the data in a neat tidy way. In order to access the database you are presented with 2 or 3 layers of options, the first Add, Update, View, Delete are the 4 processes that you can apply to the database in adding data, updating information, viewing data and deleting data. After selecting which of those you would like it asks you to select from Students, Classes or Teachers which selects the table you are accessing to perform the requested task. If you are viewing, it asks whether you want to view from a different field (i.e a teacher from a given student or class), it also allows you to simply press enter which returns all the information in that table.

After gathering this request it calls the function which handles the request (Adding, Updating, Deleting or Viewing), making several requests to the database and asking for further user input if required.

Any outputs from the database are then outputted using the tabulate package which formats the data in a clear easy to understand format.

Structure of Data:



The image on the left shows an early framework for my database topology. After working on the concept and beginning the implementation I decided to remove the TeachersID column from the classes table and instead add a Teachers_Classes joining table to handle the relationship between teachers and the classes they teach. This maintained consistency with how I handled the Students_Classes and allowed me to use similar code to implement the same functionality for both relationships.

The program creates the tables from the provided data in the form of a csv. It then selects the first row of the csv to create the columns, and then scans through the data identifying the types of each column allowing it to create the tables with columns of the right type, for all tables in the database.
It then creates the joining tables by identifying the primary keys of the tables and inserting the data in the provided csv's to create the tables required for a many to many relationship

When provided with this dataset it creates 5 tables:
Classes:
The Classes table contains a primary key ID which is an auto increment integer, which is used for joins and selections. It also contains two additional fields, which are Name and Year Level, which contain the class name (e.g Maths) and the year level of the class (e.g. 9).

Students:
The Students table contains a primary key ID, which is an auto increment integer, which is used for joins and selections. It also contains 3 fields for the student's First Name, Last Name, and Year Level which can be accessed through queries.

Teachers:
The Teachers table contains a primary key ID, which is an auto increment integer, which is used for joins and selections. The Teachers table contains columns for the teacher's First Name and Last Name, which can be returned through queries

Teachers_Classes:
This joining table contains a column for the Teacher's ID and a column for the ClassID. By entering in rows with both these fields it enables the connection between the two tables through this joining table. As this is a one to many relationship each classID only appears once, whilst a teachers ID can appear many times. This enables one teacher to teach many classes.

Students_Classes:
This joining table contains a column for the student's ID and a column for the classID, with rows containing data for both these variables. It allows for the connection between the classes table and the students table. As this is a many to many relationship, each student can be a part of many classes and each class can have many students. This is represented by both classes and students being represented multiple times in this table.

The use of the joining tables Students_Classes and Teachers_Classes make the many to many relationship of students and classes and the one to many relationship of teachers to classes easier to manage. By splitting the relationships up into two one to manys or a one to many and a one to one relationship, which made making joining queries to search from one table to another easier
The different functions outlined in the design section make queries to these tables, and then process the results and user input to make changes to the tables, and output the desired results to the end user.

Relevant Implications:

Accessibility:
Accessibility is the relevant implication of how the program is accessible to those of all backgrounds. In order for a program to be accessible it needs to be clear and easy to understand, instructing the end user as it goes through each individual section to allow them to use the program in an easy and efficient manner. In order to make this database program accessible I gave instructions for all inputs, as well as what the input is needing i.e an ID, a name or an option (such as students, teachers or classes). I also made it clear to the end user what is going wrong throughout the program if they enter something that isn't expected by the system.

The program is written in python which is a platform independent programming language using the sqlite3 package to handle database queries. Because it is platform independent the program is able to be run on all devices and operating systems with python support allowing for accessibility on a wide range of devices.

Functionality:
The relevant implication of functionality, means that a database request software should have all the necessary features and capabilities to allow users to easily perform tasks like adding, deleting, updating, and viewing information in tables. It should work smoothly, provide accurate results, and make the data management tasks convenient and efficient for the user.

My program allows for the fundamental requests a database software needs to be able to perform, those being the addition, deletion, updating and viewing of information in tables which fulfils the relevant implication of functionality for the end users. Users can easily add records, remove records that are no longer needed, update existing information and retrieve specific data for viewing. The software ensures that all fundamental functionalities are readily accessible and user-friendly, enabling users to accomplish their tasks efficiently. Additionally the software supporters data integrity by implementing validation tests to prevent incorrect data types and also to prevent malicious sql injection attacks. It also supports the functionality of secondary checks allowing the user to retrieve all students taught by a specific teacher or any other combination of students, classes and teachers. By providing a wide range of functionalities while maintaining data accuracy and reliability, the software effectively meets the functional requirements of end users, empowering them to manage and utilise the database effectively for their specific needs.

Usability:
Useability is the measure of how usable the software is to the end user, for a program to be considered usable it should satisfy the useability, some of these are: consistency, real world matches, error prevention, flexibility, etc. By taking these into account we can ensure the program is usable for the end user.

My program allows for the addition, deletion, updating, and viewing of information stored within several different tables which satisfies several important implications of usability for the end user. Firstly, it is a clear text based interface which allows users to quickly grasp the software's functionalities, and perform tasks without extensive training. The software provides intuitive entries and clear instructions, which guide users through interacting with the database effectively. Users can easily add, delete, update, and view information in tables by selecting options pre provided to the end user, this reduces the time and effort required to perform these operations manually. The software also promotes error prevention and recovery by implementing validation checks to ensure the data entered is suitable for the system. It also presents the information to the end user in a clear simple manner using the tabulate python library, which formats the data into a table like format.

Another way I improved useability was when asking for full names if the user inputted more or less than two names it would ask for them to re enter allowing for two + first names and two + last names, allowing the user to input what names they wanted using the latin script.

Database Refinement:

Component descriptions:
The CreateDatabase file contains all the functions that are used to create the tables from the provided csv files. For the main 3 tables (Students, Teachers, Classes) in order to do this it reads the headers in the csv file to obtain the column names and then iterates through the individual data to identify the type the column should be (Integer, Text, etc). Then once this is all gathered it creates the table and inserts the values from the csv file into it. After this is complete it would then create the joining tables. It checks what the primary key of each table it is joining and then makes those the column names, and inserts the data from the csv to finish the joining table.

That code located in the CreateDatabase is only run whenever ‘Database.db’ is not already present in the folder to prevent duplication and any errors that result due to this. I had some issues with this because when I was importing functions from other files it would create the file but not run the creation of the tables resulting in an empty database, which is not useful. I fixed this by passing the connection through the functions as a parameter to all functions in other files, which removed the code snippets that caused these issues.

I have all the query functions for my code sorted into different files for legibility and ease of use when making changes.

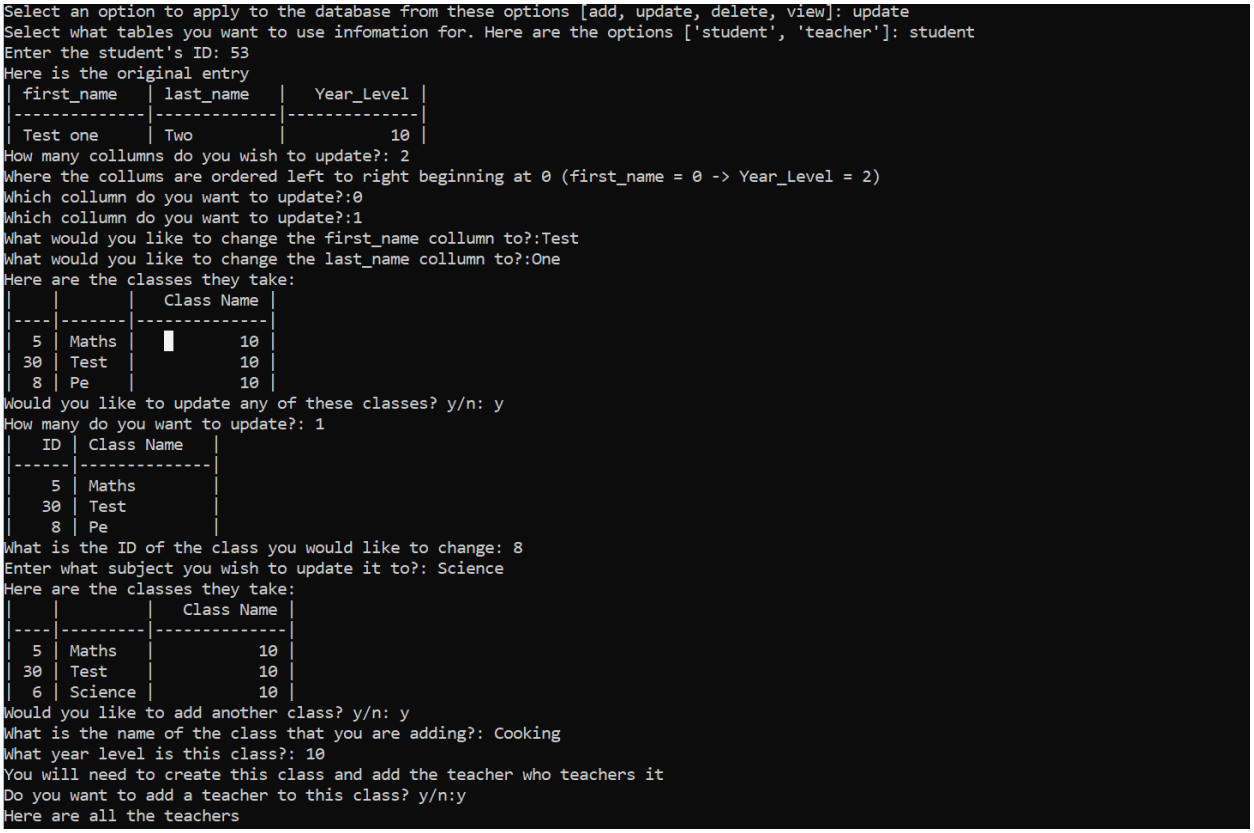
The addQueries file contains the code for adding students, teachers and classes to the database. They all work in similar ways, asking for information through user input, and then running the other commands to add classes that a student takes or a teacher teaches. It also adds to the joining tables to ensure that the other queries are able to function. I made changes to these functions as I tested them, adding the functionality to add students to the lowest populated class if there are duplicate classes, and also checks whether a class is already taught before adding a teacher. I also added functionality so that other functions notably the updating functions can call upon them to make changes but using premade information. So I added functionality to do this.

```
Select an option to apply to the database from these options [add, update, delete, view]: add
Select what tables you want to use infomation for. Here are the options ['student', 'class', 'teacher']: student
What is the students full name?: Test
['Test'] can't automatically be split into first and last name
Please enter the first name/names: Test One
Please enter the last name/names: Two
Enter the student's year level: 10
How many classes do they take?3
Enter what subject do they take?: Maths
Enter what subject do they take?: Test
You will need to create this class and add the teacher who teachers it
Do you want to add a teacher to this class? y/n:y
Here are all the teachers
  ID  First Name    Last Name
----  ------------  -----------
   1  Malcolm       Tremayne
   2  Mary          Kienzle
   3  Alfy          Spadaro
   4  Pall          Londors
   5  Pierrette     Eplate
   6  Rosamond      Arundale
   7  Isiahi        Perassi
   8  Holly         Norwell
   9  Fran          Phillips
  10  Frank         Clarke
Enter the ID of the teacher to add OR enter new to create a new teacher: new
What is the teachers full name?: Mr Teacher
How many classes do they teach?1
Addition of Teacher successful
Addition of Class successful
Enter what subject do they take?: Pe
Addition of Student successful
```

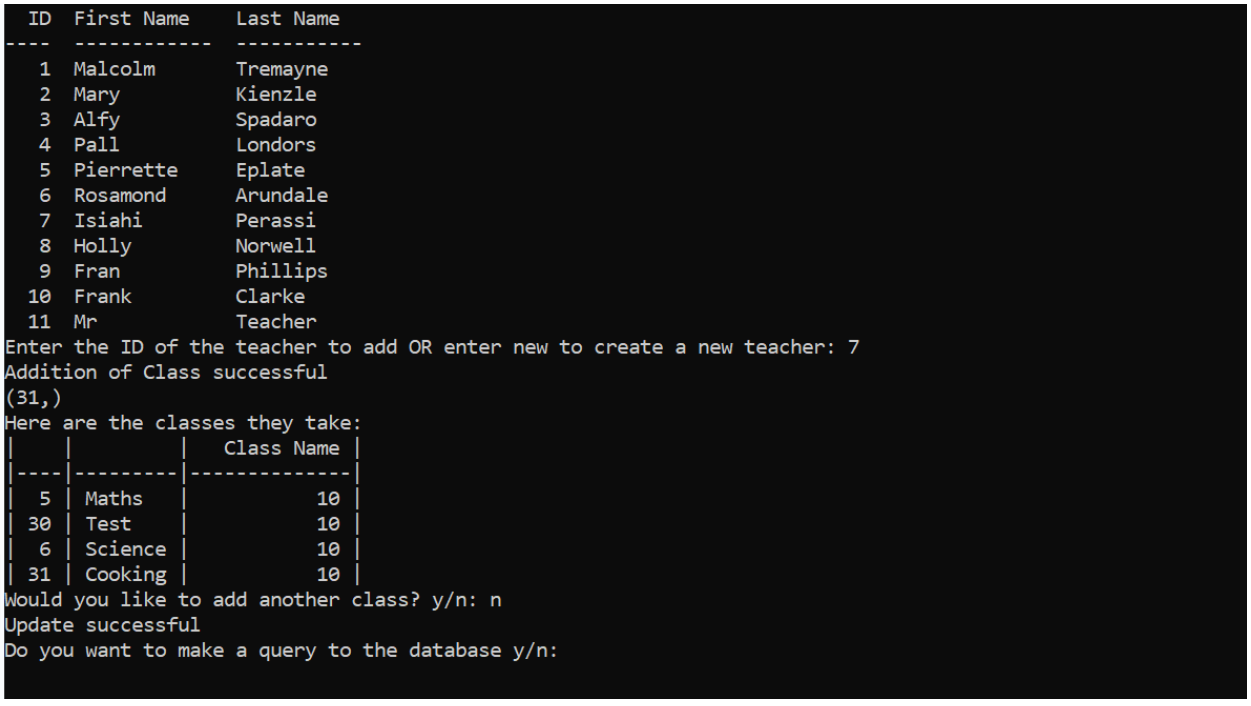
Here is an example of an adding query, in fact the add student function also calls the add class and add teacher if needed to, like in this example.

The updateQueries file contains the code for updating existing information for both students and teachers. It allows the changes of all columns (not including ID) then allows for the updating of classes that they take followed by the addition of new classes. It does this by making use of sqlite queries to retrieve information from the database and returns them to the user for user input to determine what changes will be applied. Throughout the design of these functions I made several changes to how they operate, from just adding existing classes, to calling the addition function to create new classes should the user require it, As well as preventing duplicates for students but not teachers (I had previously allowed duplicates for both). This is because in a typical school a student can not take say 2 year 9 maths classes, but a teacher

can teach multiple classes of the same year.



```
Select an option to apply to the database from these options [add, update, delete, view]: update
Select what tables you want to use infomation for. Here are the options ['student', 'teacher']: student
Enter the student's ID: 53
Here is the original entry
| first_name   | last_name   |   Year_Level |
|--------------|-------------|--------------|
| Test one     | Two         |           10 |
How many collumns do you wish to update?: 2
Where the collums are ordered left to right beginning at 0 (first_name = 0 -> Year_Level = 2)
Which collumn do you want to update?:0
Which collumn do you want to update?:1
What would you like to change the first_name collumn to?:Test
What would you like to change the last_name collumn to?:One
Here are the classes they take:
|    |       |   Class Name |
|----|-------|--------------|
|  5 | Maths |           10 |
| 30 | Test  |           10 |
|  8 | Pe    |           10 |
Would you like to update any of these classes? y/n: y
How many do you want to update?: 1
|   ID | Class Name   |
|------|--------------|
|    5 | Maths        |
|   30 | Test         |
|    8 | Pe           |
What is the ID of the class you would like to change: 8
Enter what subject you wish to update it to?: Science
Here are the classes they take:
|    |         |   Class Name |
|----|---------|--------------|
|  5 | Maths   |           10 |
| 30 | Test    |           10 |
|  6 | Science |           10 |
Would you like to add another class? y/n: y
What is the name of the class that you are adding?: Cooking
What year level is this class?: 10
You will need to create this class and add the teacher who teachers it
Do you want to add a teacher to this class? y/n:y
Here are all the teachers
```



```
  ID  First Name    Last Name
----  ------------  -----------
   1  Malcolm       Tremayne
   2  Mary          Kienzle
   3  Alfy          Spadaro
   4  Pall          Londors
   5  Pierrette     Eplate
   6  Rosamond      Arundale
   7  Isiahi        Perassi
   8  Holly         Norwell
   9  Fran          Phillips
  10  Frank         Clarke
  11  Mr            Teacher
Enter the ID of the teacher to add OR enter new to create a new teacher: 7
Addition of Class successful
(31,)
Here are the classes they take:
|    |         |   Class Name |
|----|---------|--------------|
|  5 | Maths   |           10 |
| 30 | Test    |           10 |
|  6 | Science |           10 |
| 31 | Cooking |           10 |
Would you like to add another class? y/n: n
Update successful
Do you want to make a query to the database y/n:
```

Here is an example of an updating query it allows not only the change of information but also the classes that they are taking/teaching

The deleteQueries file contains the code to handle deleting items from the database. It does this by asking for the id of what is being deleted through user input, and then deleting the rows with this ID from the table it is located within as well as the joining tables this is present in. I did this because if a class no longer exists, it can't contain any students anymore which logically makes sense.

```
Select an option to apply to the database from these options [add, update, delete, view]: delete
Select what tables you want to use infomation for. Here are the options ['student', 'class', 'teacher']: student
Enter the student's ID to delete: 2452
No data found with ID = 2452
Do you want to make a query to the database y/n:y
Select an option to apply to the database from these options [add, update, delete, view]: delete
Select what tables you want to use infomation for. Here are the options ['student', 'class', 'teacher']: student
Enter the student's ID to delete: 53
Deletion successful
```

Here is a demonstration of a deleting file, note how I entered an incorrect ID and how it informed the user that the ID was invalid.

The searchQueries file contains the functions that search the database and returns them to the user based on their requests. It contains functions for all dual combinations of students, teachers and classes as well as an additional function for purely selecting all from the required fields. It does this by running an sql query using the joining tables to add the information from multiple tables, to allow the selection of one table from information in another table.

```
Do you want to make a query to the database y/n:y
Select an option to apply to the database from these options [add, update, delete, view]: view
Select what tables you want to use infomation for. Here are the options ['students', 'classes', 'teachers']: students
Select an option to use as an additional parameter (i.e classes if you want to view _ from a class, you can press enter to return all items) ['class', 'teacher'
|    | First Name   | Last Name    |   Year Level |
|----|--------------|--------------|--------------|
|  1 | Carroll      | Keenleyside  |            9 |
|  2 | Velma        | Brissenden   |            9 |
|  3 | Yule         | Hellmore     |            9 |
|  4 | Prentiss     | Duetsche     |            9 |
|  5 | Fidelio      | Denys        |            9 |
|  6 | Levi         | Gorce        |            9 |
|  7 | Corabelle    | Manach       |            9 |
|  8 | Kate         | Sallowaye    |            9 |
|  9 | Carlina      | Vardie       |            9 |
| 10 | Clive        | Patullo      |            9 |
| 11 | Rodolph      | Cafe         |            9 |
| 12 | Travers      | Lowman       |            9 |
| 13 | Zebadiah     | Vickarman    |            9 |
| 14 | Flory        | Haldon       |            9 |
| 15 | Rosene       | Jolliff      |            9 |
| 16 | Gaultiero    | Sowerby      |            9 |
| 17 | Katine       | Scripps      |            9 |
| 18 | Kayla        | Francino     |            9 |
| 19 | Salli        | Ricardot     |            9 |
| 20 | Asia         | Sickling     |            9 |
| 21 | Urbano       | Del Castello |            9 |
| 22 | Any          | Niese        |            9 |
| 23 | Waly         | Simonelli    |            9 |
| 24 | Yorke        | Ziems        |            9 |
| 25 | Davis        | France       |           10 |
| 26 | Daniel       | Bellison     |           10 |
| 27 | Edsel        | Larcher      |           10 |
| 28 | Tobe         | Shewsmith    |           10 |
| 29 | Dar          | McGivena     |           10 |
| 30 | Luce         | Dormon       |           10 |
| 31 | Sidonnie     | Mews         |           10 |
| 32 | Kitty        | Robinet      |           10 |
| 33 | Bruno        | Gowen        |           10 |
| 34 | Alikee       | Morrel       |           10 |
| 35 | Emmaline     | Kuhwald      |           10 |
| 36 | Mahalia      | Lowey        |           10 |
| 37 | Dorian       | Gadd         |           10 |
| 38 | Rafa         | De Coursey   |           10 |
| 39 | Veronique    | Zupone       |           10 |
| 40 | Katya        | Wraight      |           10 |
| 41 | Phyllys      | Spada        |           10 |
| 42 | Katey        | Poulden      |           10 |
| 43 | Randy        | Cook         |           10 |
| 44 | Babara       | Rive         |           10 |
| 45 | Belvia       | Worters      |           10 |
| 46 | Selestina    | Stroobant    |           10 |
| 47 | Any          | Blaxall      |           10 |
| 48 | Ara          | Gebuhr       |           10 |
| 49 | Derek        | Gabb         |           10 |
| 50 | Enriqueta    | Greystock    |           10 |
```

Here is an example of a search query. It returns all the students in the database and formats it nicely using the tabulate package.

The main file contains the menu function which allows the user to select which queries shall be used. It contains a dictionary which has multilayered keys which correspond to lambda functions that call the individual functions within the other files to run the query. It uses user input which corresponds to each of the keys and subkeys to filter through the functions to allow the user to easily select which key it wants to use. It provides all the keys in advance for the user to select from, which makes it easy for the user to ensure that they are entering valid options for keys, and in the case of an invalid option it will inform the user and ask them to re-enter their options. It also asks in advance whether they want to make a request and loops until the user doesn't wish to make any further requests. This makes it so that the user doesn't need to constantly rerun the program to make multiple requests.

I used iterative development to design and work on this program using the above sections as smaller chunks of the program which I worked upon in isolation from each other testing and trailing each of them individually until it was to an acceptable standard, and then moving on to the next iteration being the next type of query or function. I also then tested the snippets in relation to each other such as adding a new class when updating a student.

This process of linking code together brought a few issues. At one point I was inserting the data into the wrong fields which lead to incorrect results as it would still return results if classID and teacherID were switched, by testing this I discovered the issue and was able to fix it. Another Issue I had was making the addition of new classes work from inside an updating function. I had to ensure the data types were consistent when passing values between functions for the creation. An example of when this went wrong was when I had accidentally passed the year level variable as a tuple (10,) instead of an integer 10. This caused issues as it wasn't the type it had been expecting. By testing throughout the design I discovered and corrected many of these issues that arose.

For more information and examples see the commit history of my github repository.

I filmed a test of the program on Thursday 8th of June using some of the functions to show how they work. See ExampleTest.mkv, you can also run the code yourself to test the programs on your own. For example in this test I found that the last name of the teacher was not being capitalised hence not returning any values when the name was being entered. To address this I added these lines of code

```
for i in range(len(TeacherName)-1):
        TeacherName[i] = TeacherName[i].capitalize()
```

To address this issue.

Another issue I had was with hyphenated double barreled names which would throw issues due to my sql injection preventions. In order to allow these I added the '-' character to my regex of accepted characters which resolved this issue.

In conclusion I designed a database structure that allows for an end user to add, delete, update, and view items from a database. I have used iterative development to organise my development of the database and resultant code. I have also designed the program to allow for these queries to be selected in an organised structured way, and for the data to be presented to the end user in a clear and systematic way.