Student Access To Sixth Form

Contents

[Analysis 3](#_Toc482346036)

[Problem Identification 3](#_Toc482346037)

[Research 4](#_Toc482346038)

[Summary of Research 5](#_Toc482346039)

[Clients and Users 5](#_Toc482346040)

[Background Information 6](#_Toc482346041)

[User Requirements 6](#_Toc482346042)

[Analysis Objectives 7](#_Toc482346043)

[Design (Entity Relationship (ER) Model) 9](#_Toc482346044)

[Proposed Solution 9](#_Toc482346045)

[Software Requirements 9](#_Toc482346046)

[Hardware Requirements 9](#_Toc482346047)

[Data Volumes 10](#_Toc482346048)

[Design 11](#_Toc482346049)

[Hierarchy Chart 11](#_Toc482346050)

[Structure Diagram 11](#_Toc482346051)

[Class Diagram 12](#_Toc482346052)

[System Structure Flowchart 13](#_Toc482346053)

[Data Flow Diagrams 17](#_Toc482346054)

[Level 0 17](#_Toc482346055)

[Level 1 18](#_Toc482346056)

[Level 2 19](#_Toc482346057)

[Human-Computer Interaction (HCI) 20](#_Toc482346058)

[Queries 21](#_Toc482346059)

[System Security 21](#_Toc482346060)

[Technical Solution 22](#_Toc482346061)

[Final Solution 30](#_Toc482346062)

[Data Dictionary 44](#_Toc482346063)

[Sorting 46](#_Toc482346064)

[Testing 47](#_Toc482346065)

[Test Strategy 47](#_Toc482346066)

[Requirement Testing 48](#_Toc482346067)

[Functional Testing 53](#_Toc482346068)

[User Testing 61](#_Toc482346069)

[Evaluation 62](#_Toc482346070)

[User Feedback 66](#_Toc482346071)

[User Feedback Analysis 66](#_Toc482346072)

[Possible Extensions 66](#_Toc482346073)

# Analysis

## Problem Identification

The problem that I have been tasked to solve is creating a python program that could potentially replace the current system used by Beauchamp College to show the grades of GCSE level students and then determine whether or not that student will be allowed access to the Sixth Form centre. The students will have information stored about them, including their grade at GCSE level. If the student has achieved 5 A\* to C grades in any subject accompanied by a B in English Language, then the program will create a green mark/ flag to say to the user that the student has met the entry requirements to get into the sixth form centre at Beauchamp College. However, if the student has not achieved the entry requirements then the program will create a red mark/ flag which the user can then view. Also in the Graphical User Interface that is created by Tkinter the user will be able to view the students that have been inputted into the database. The user will also be able to search for a specific user and then view the grades that were achieved and all of the necessary attributes that would be required by the unique entity (student).

## Research

As my program is being designed for one specific person to use very often I plan to carry out my research through an Interview with my client.

I plan to research my project through the use of an interview:

* The Pros of using an interview to conduct research include:

1. An interview allows for clarification of the answers that have been given to the questions that were asked.
2. There is a high response rate to the questions that are asked so more questions can be asked in a shorter amount of time.
3. I would be able to gather in-depth information on the project that I have been asked to create. I would also be able to pursue any hunches that arise during the interview.
4. An interview would also give me a more personalised approach because I can tailor the line of discussion to my client.
5. An interview will also make it easier to ask open-ended questions and use probing questions to find out more information about the problem that I have been tasked with solving.
6. My client’s words will be recorded so there are no misconceptions of data.

* The Cons of using an interview to conduct research include:

1. There may be a Reactive Effect. This is where my presence and characteristics may bias my client’ results.
2. An interview is the slowest method of data collection and analysis so it will take a lot of time to complete as there may be a lot of information that needs to be collected.
3. The responses of the client may be less honest and thoughtful.
4. An interview will only reach a smaller sample so less data is collected at once.
5. The data that I collect may be difficult to analyse and therefore quantify the results.
6. Interviews can be very time consuming; setting up, interviewing, transcribing and analysing

The reason why I have chosen to conduct an interview as my main way to research my project is because my client/ user list is very short (only includes Miss Young and the senior staff of the school) so I am able to conduct an interview with my client face-to-face so I will be able to ask more open ended questions to get answers to the questions that I ask throughout. An interview will also allow me to:

1. Investigate any issues in an in depth way.
2. Deepen the understanding about how the program will work and explain any statistical data.

### Summary of Research

The interview that I conducted with my client enabled me to gain insight to what I was required to do. The main feature that I was required to produce was to enable ease of access so that anyone, even with no programming experience, can use the solution that I have created. Miss Young also wanted the program to store all student information in a way that is easily accessible so that it can be accessed and edited as, and when, the user’s wish. Miss Young also requested that the fields can be searched accurately so that the users can view any specific student’s information and then determine if the student can access The Beauchamp Sixth Form Centre.

## Clients and Users

The client of my program is Miss Young from the Science Department and the head of sixth form. She has asked me to create this program to allow ease of access to all student information that is relevant to their access to sixth form.

The users of my program include Miss Young and the rest of the senior staff at Beauchamp College. The users will be required to use the program yearly when results day for GCSE arrives so that they can view all internal students that will be entering Beauchamp College Sixth Form

## Background Information

For the coursework side of my Computing A-Level I had to find a client that had a solvable problem that I could tackle. For my client I talked to Miss Young from the Science department in Beauchamp College, who is also the head of sixth form. The problem that I was tasked with is to create a new version of the database that determines whether GCSE students are allowed access to Sixth Form based on their grades. If the student achieves five C - A\* in any subject (excluding English Language) and a B - A\* in English Language then they are allowed into the sixth form centre. To make it easier for the program to understand each Grade will be given a numerical value with C being given the numerical value of 4. With this number system in place the program will then add them up and compare it to the entry requirement of 25 point (Minimum of 5 Cs and 1 B). If the student achieves 25 points or more then the system will flag up that student to the user to say that they have achieved the entry requirement. Most likely by creating a green box next to the student’s name. If the student does not achieve the minimum entry of 25 points (or more) then the program will hopefully outline the student with a red box to tell the user that the requirements have not been met by the student.

To create the program I will create a relational database in the format of Standard Query Language (SQL). This will then be linked to a code that will be created in coding language Python. This will be the part of the program that won’t be visible to the user whilst the program is running. The third and final component of the program will be a front end Graphical User Interface (GUI) that will be created by using the python ran TKinter which creates the GUI from the command-line programming of python. This window will be fully customisable so that it suits the user’s needs. This window will include two different sections that will allow the user to view every student that has achieved the entry requirements and all of the students that didn’t achieve the grades. This will be done when the user presses a button labelled “Show all students”. There will also be a search bar that will allow the user to type in a specific students attributes (StudentID, FirstName and Surname). If there is a student that matches the searched attributes then their grades will appear along with a green or red box. However, if there is no student that matches the search criteria then the program will outline this and then ask if the user would like to add in a new student entity to the database.

## User Requirements

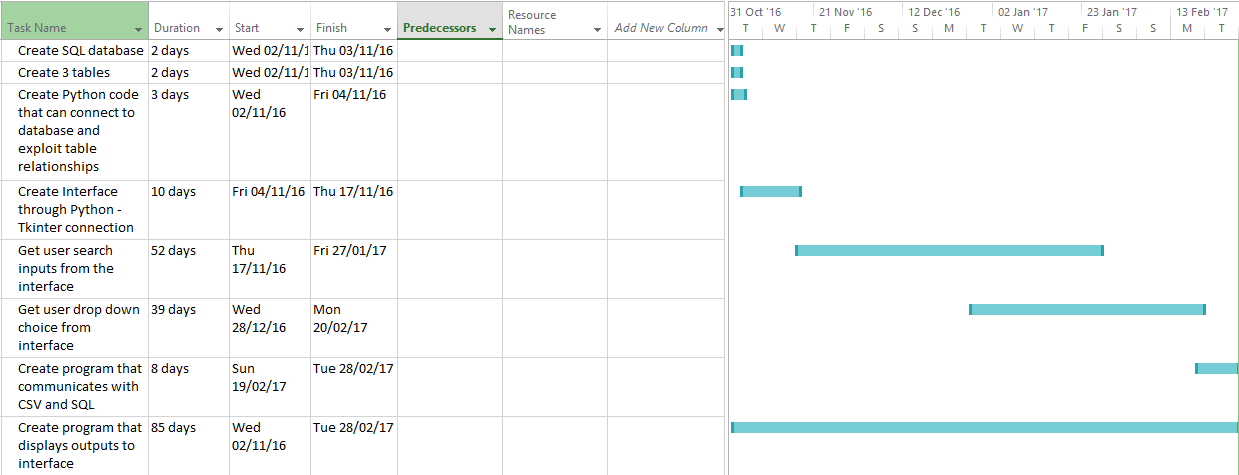
* **System requirements:**

1. Create a database to store student information
2. Create a Python – Tkinter program to communicate with the database and create a main interface for interaction
3. Allow searching through the list of fields in the database

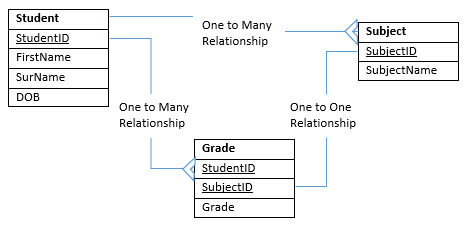
## Analysis Objectives

* **System requirements:**

1. Create a Standard Query Language (SQL) database that holds the component details
   1. Create three tables that will hold the necessary details
      1. Make reference to Entity Relationship Diagram below
      2. Create a relationship between the necessary tables that can then be used when reading from the database.
2. Create a link in the Python code that can connect with the SQL database
   1. Allow the program to exploit table relationships to allow user to view the required data
3. Create an interface in Python/Tkinter for the user to use effectively
4. Read any user inputs that are given
   1. Take the user’s search “Entry Year”
      1. Accepts valid digits e.g. “2016>”
      2. Rejects invalid digits e.g. “<2016”
      3. Rejects any numbers that aren’t whole
      4. Rejects negative inputs
      5. Informs user if input is invalid
   2. Take the user’s search “Last Name”
      1. Must be a string of characters
      2. Rejects numbers
      3. Informs user if input is invalid
      4. Input is case sensitive
   3. Take the user’s search “Initial”
      1. Accepts all letters e.g. “A”
      2. Rejects any number
      3. Rejects erroneous data
   4. Take the user’s search “Grade”
      1. Accepts valid inputs
         1. “A\* - F” and “U” and “9 – 0”
      2. Rejects invalid inputs
         1. “G – T” and “10>”
   5. Take the user’s search “Subject”
      1. Accepts valid inputs
         1. “Mathematics”
      2. Rejects invalid inputs
         1. “Maths”
5. Create drop down menus that the user can exploit
   1. Create menu “File”
      1. Create options “Load Students With Grades”, “Load all Students”, “Close”, “About”
      2. Create program that responds to a user click on desired option
   2. Create menu “Edit”
      1. Create options “Edit Student”, “ Edit Grades”
      2. Create program that responds to a user click on desired option
   3. Create menu “Delete”
      1. Create options “Delete Item”
         1. Deletes any selected item from the database and the interface
      2. Create program that responds to a user click on desired option
   4. Create menu “Insert”
      1. Create options “Insert Student details”, “Insert Student Grades”
      2. Create program that responds to a user click on desired option
   5. Create menu “Import”
      1. Create program that responds to a user click on desired option
      2. Reads information from CSV file and imports it into the database with the correct format
6. Construct outputs determined by the input
   1. Take the user search input from 4.
      1. Search through the table list using the correct parameters
         1. If result found then go to point 6.
         2. If result not found then display error message for user
7. Display the outputs
   1. Display any results in the table for the user to view



## Design (Entity Relationship (ER) Model)



## Proposed Solution

In this section I will specify the software and hardware requirements of the solution that I have created.

### Software Requirements

The program will require “Python (IDLE) 3.4.4.” to run the solution so that the user can interact with the database that I have created to run parallel to the code. The user will also need to install “SQL lite DB Browser” so that they are enabled to open the database directly through the windows interface and interact with it in that way if they so choose too. Also if they need to edit mistakes that were made that my solution does not account for. My solution also requires Microsoft Excel to be downloaded onto the system software so that it has the ability to bulk import from the CSV file. The user will also need to have an OS of Windows XP, Windows Vista, Windows 7 (32bit/64bit), Windows 8, Windows 10 with 1 GB RAM and a minimum of 850 MB free disk space for Python to be able to run efficiently

### Hardware Requirements

The user will need a monitor to view the outputs from the python program. The user will also need a keyboard and mouse to be able to input and interact with my final solution

## Data Volumes

The data that has been accounted for being entered into my system will be entered yearly but at a bulk import rate. This means that at least 1000 new students will be added each year based on the number of internal students. The solution that I have created should be able to cope with the input as I created a Bulk Import system inbuilt into the program that interacts with the CSV file that the user puts any details of students into. These details are then imported the SQL database after the program has edited the format of the data so that it corresponds to all data currently being held in the database. The solution that I have created, in theory, should not have the processing speed slow down as

# Design

## Hierarchy Chart

## Structure Diagram

## Class Diagram

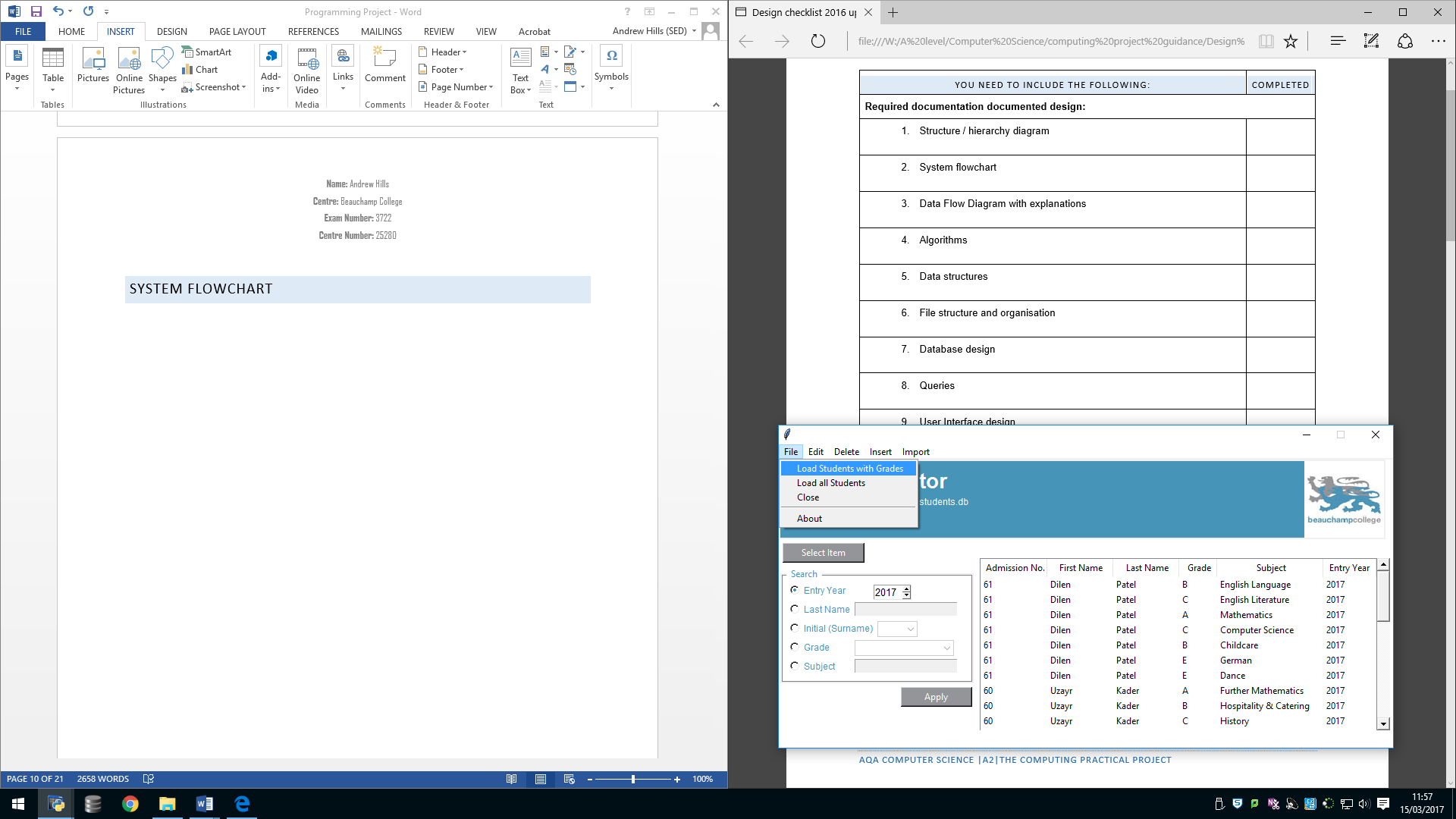
|  |
| --- |
| getStudentGrades |
| +myComboSubjectboxVal: String  +myComboGradeboxVal: String |
| +fnGetStudentGrades() |

|  |
| --- |
| getStudentDetails |
| +txtEntry1: String  +txtEntry2: String  +SpinDayVal: Integer  +SpinYrval: Integer  +SpinMonthVal: Integer  +SpinEntryVal: Integer |
| +fnGetStudentDetails() |

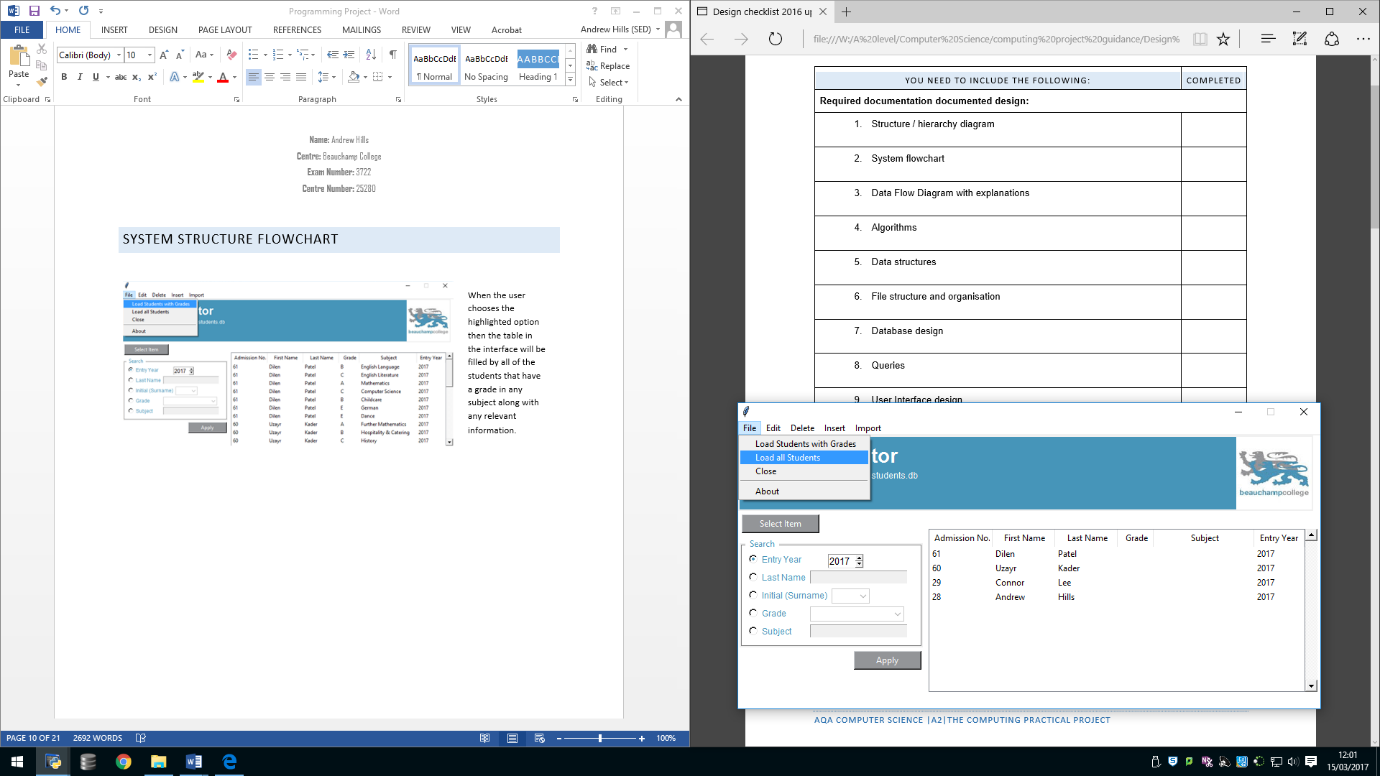
|  |
| --- |
| EditStudentDetails |
| +txtEntry1: String  +txtEntry2: String |
| +fnChangeStudentInfo() |

|  |
| --- |
| EditStudentGrades |
| +myComboGradeboxVal: String |
| +fnChangeStudentInfo() |

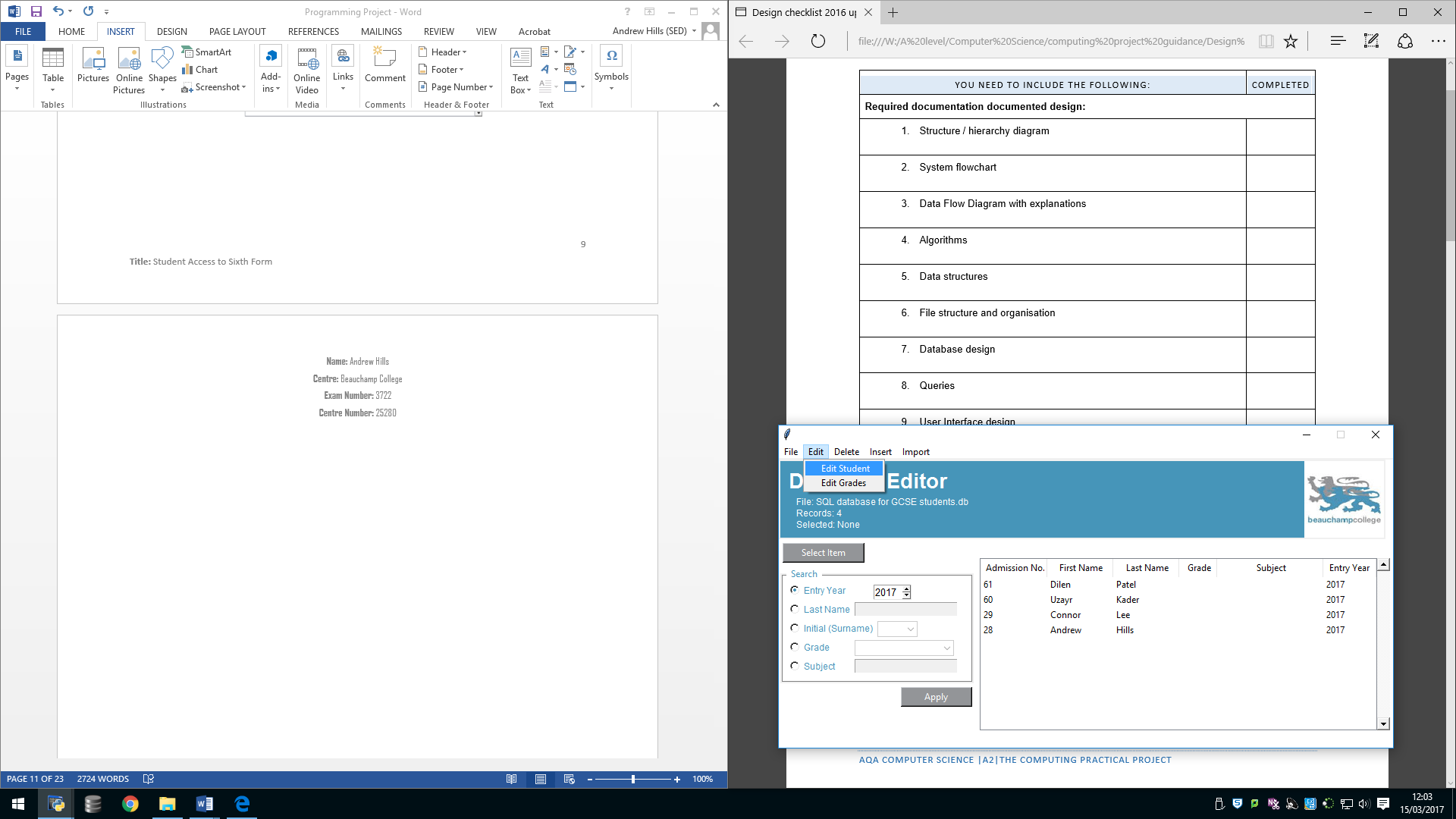
## System Structure Flowchart



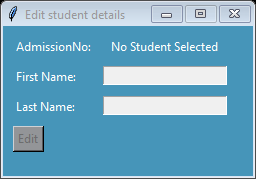
When the user chooses the highlighted option then the table in the interface will be filled by all of the students that have a grade in any subject along with any relevant information.

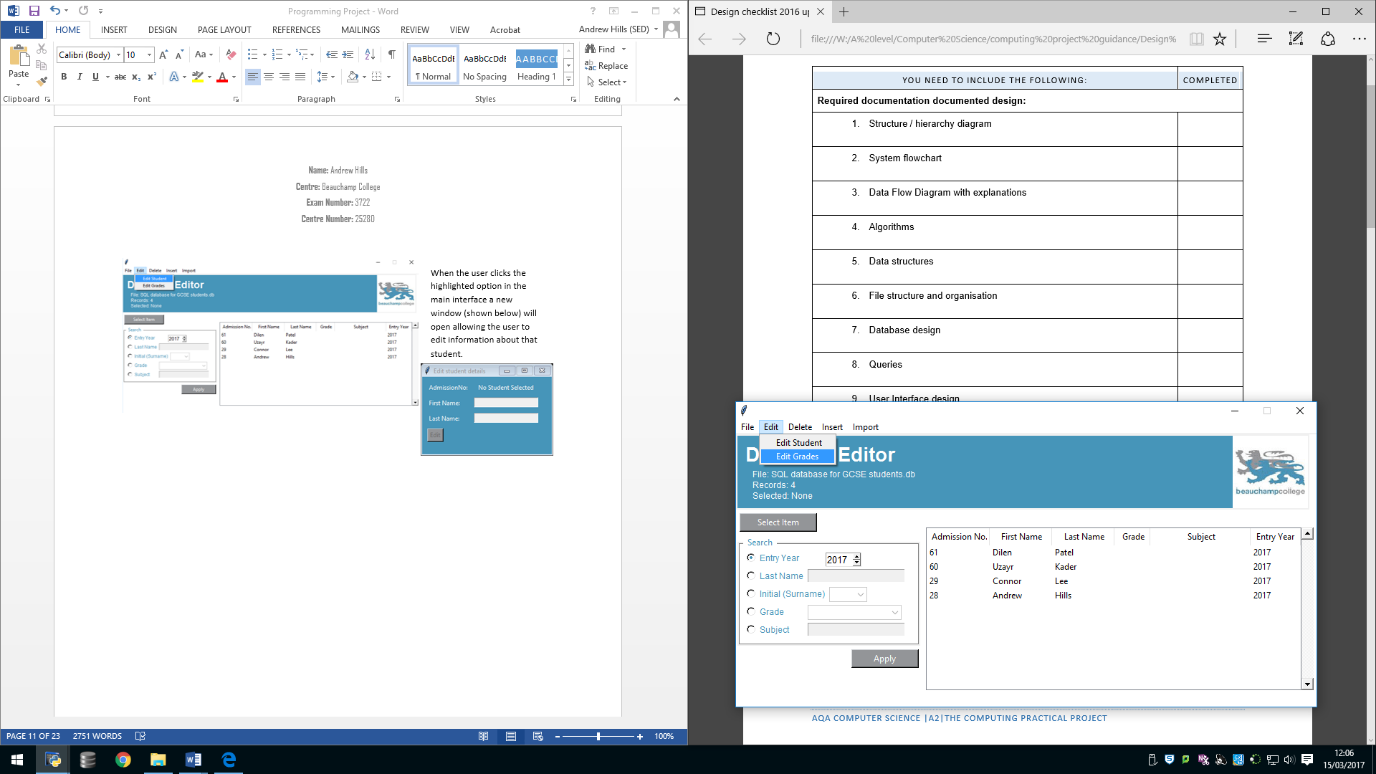


When the user chooses the highlighted option then the table in the interface will be filled by all of the students that have a record in the “Student” table in the database.

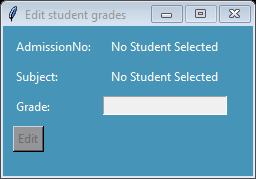


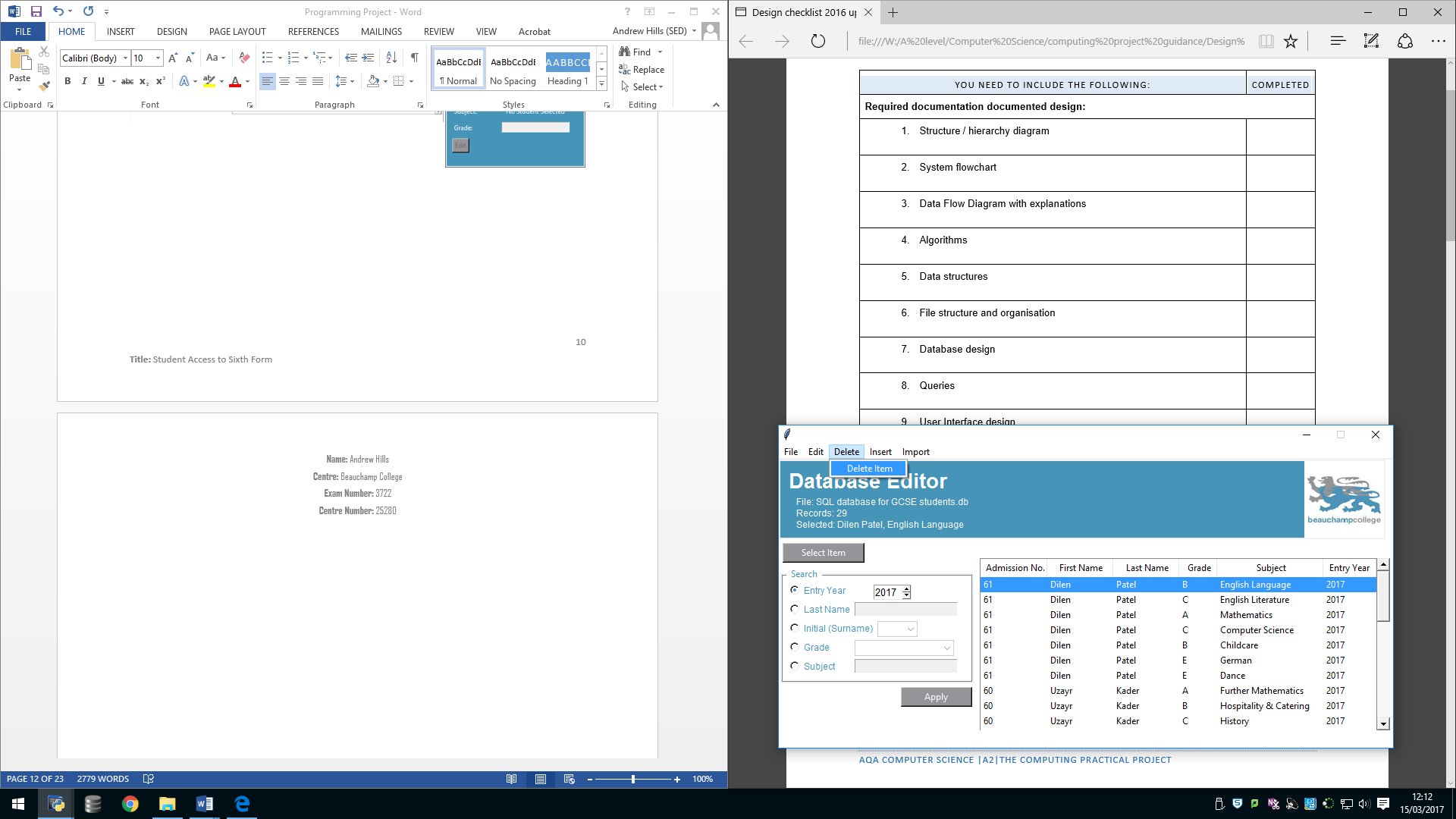
When the user clicks the highlighted option in the main interface a new window (shown below) will open allowing the user to edit information about that student.



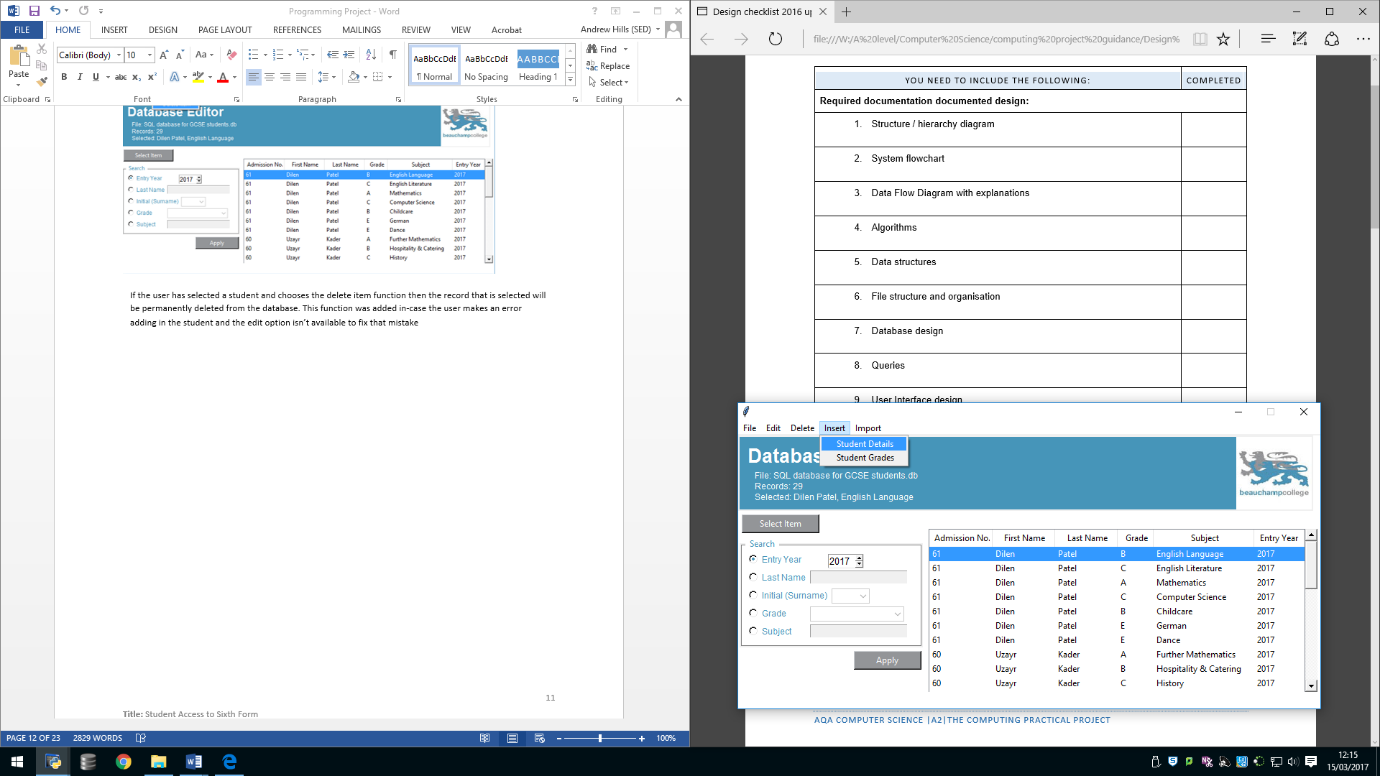


When the user clicks the highlighted option in the main interface a new window (shown below) will open allowing the user to edit information about that student’s grades

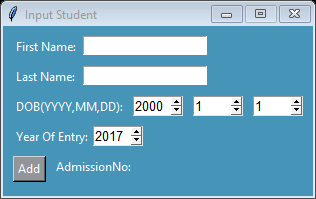


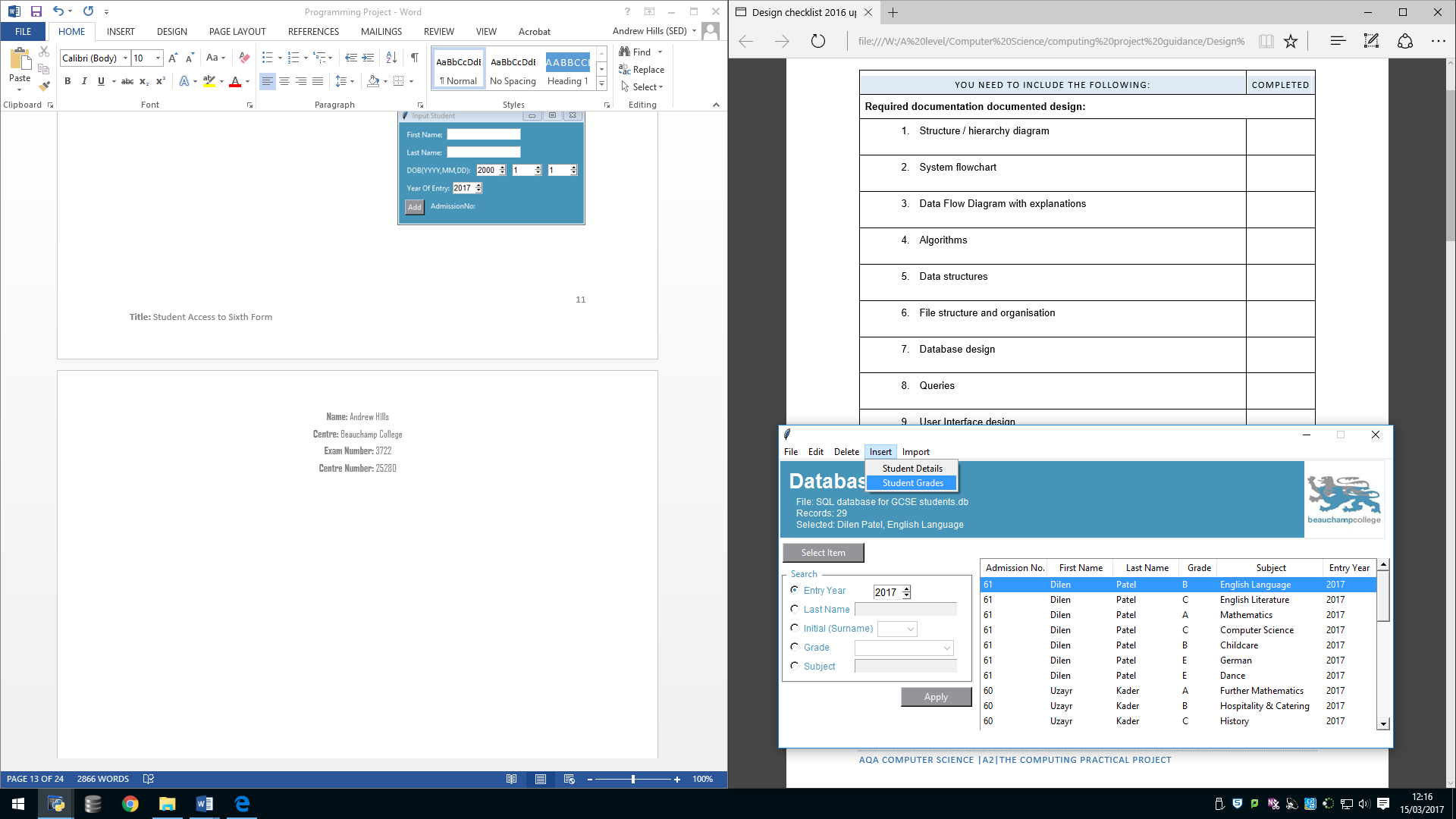


If the user has selected a student and chooses the delete item function then the record that is selected will be permanently deleted from the database. This function was added in-case the user makes an error adding in the student and the edit option isn’t available to fix that mistake

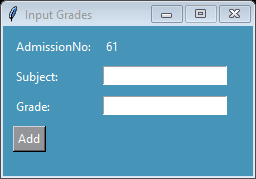


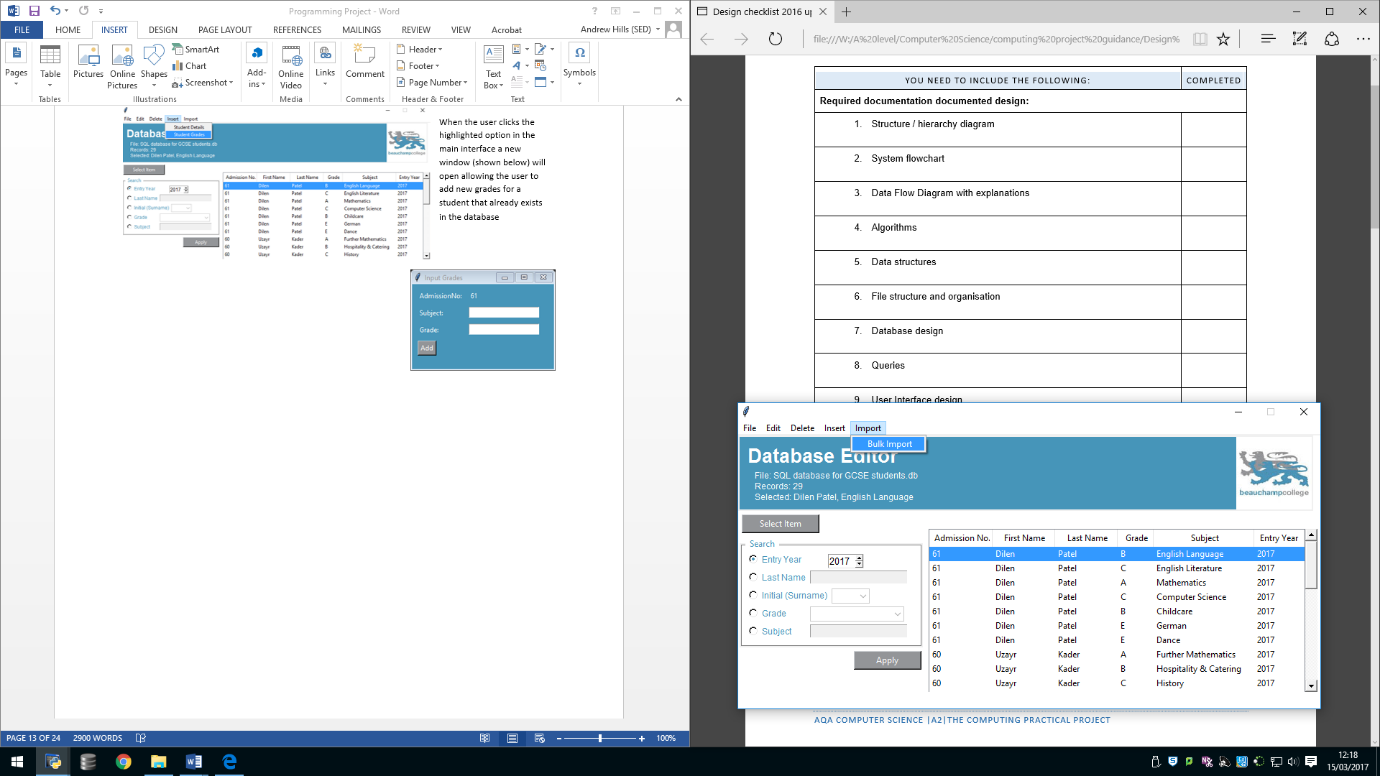
When the user clicks the highlighted option in the main interface a new window (shown below) will open allowing the user to add a new student to the database that can then be edited or built upon



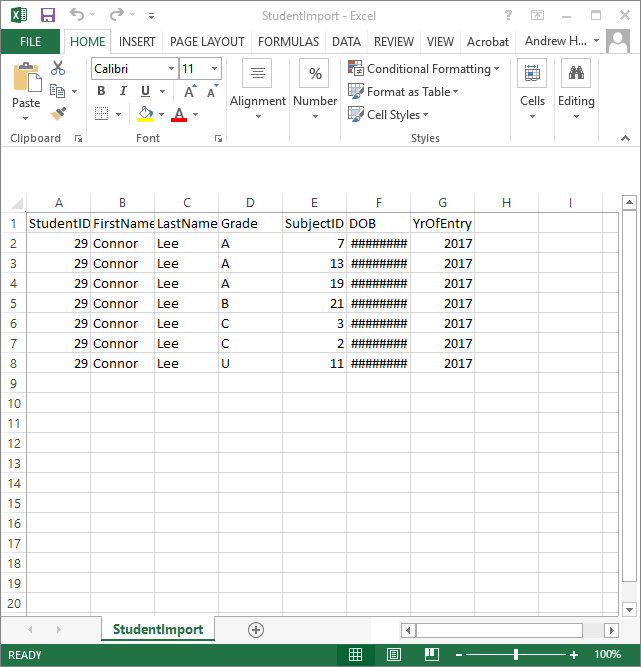


When the user clicks the highlighted option in the main interface a new window (shown below) will open allowing the user to add new grades for a student that already exists in the database



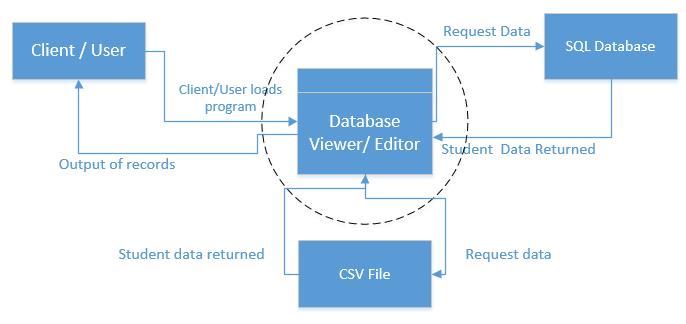


When the user clicks the highlighted option in the main interface the program will search through a CSV file (shown below) for any information that is being held there. If data is found then it will convert the format to SQL and then add the data into the relevant areas of the database

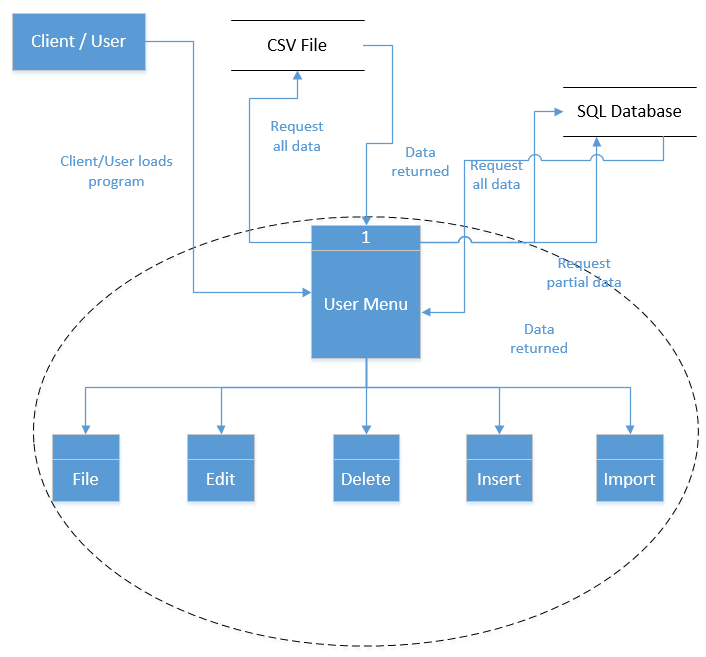


## Data Flow Diagrams

### Level 0



### Level 1



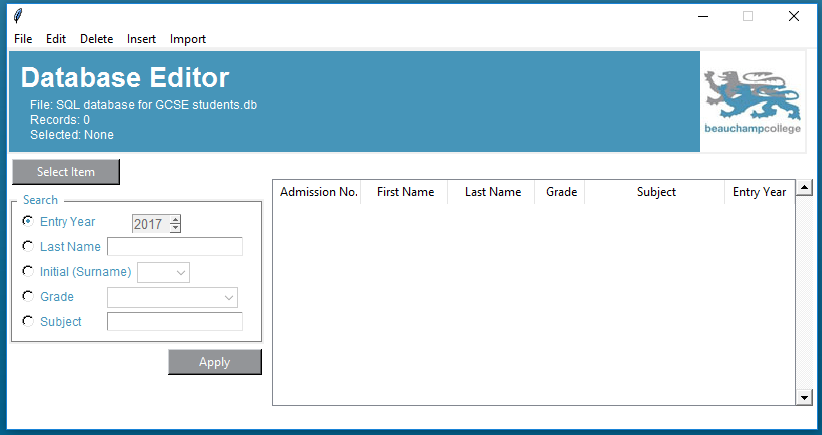
### Level 2

## Human-Computer Interaction (HCI)

Title

Shows the user how many records there are currently stored in the interface table. It also tells the user if the search criteria is invalid

Drop down menus that the user can interact with.



Database file name

Displays selected student

Logo

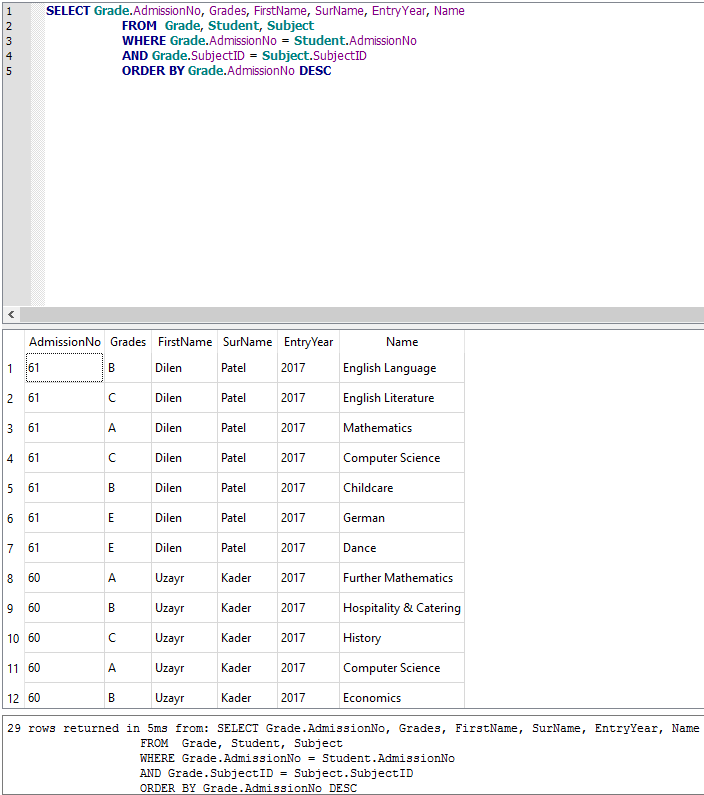
Allows the user to select an item in the table and then interact with it

These allow the user to search through the interface table on the right of the interface

Table that allows the user to view the records stored in the database. It also shows the output for any search criteria that the user chooses.

Allows the user to confirm the search criteria they have chosen and then searches through the table based on the search choice

## Queries



The query shown above enables the program to search through the database. In this example the code is querying the database for; AdmissionNo and Grades from the table Grade, FirstName, SurName and EntryYear from the table Student as well as Subject Name from the table Subject. The WHERE and AND statements tell the program where the foreign keys are linked to the primary keys. The ORDER BY statement allows the code to order the results from the query by the AdmissionNo retrieved from table Grade in a descending order.



The query shown above enables the program to search through the database. In this example the code is querying the database for every field that is currently being stored in the table Student



The query shown above enables the program to insert data into a database. In this case is allows the program to add AdmissionNo, Subject and Grade that were achieved by the student.

## System Security

The final solution that I have created does not require any implementation of additional security measures as the program and all of its components (database and excel files) are all stored locally on each individual users personal computer which only they have access to.

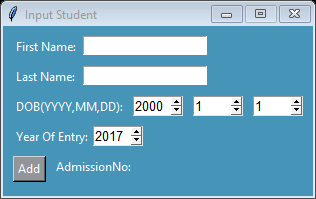
# Technical Solution

The code (above) creates an instantiation of a class. It includes validation where the user has to select a student from the main interface (below). If no student is selected then the program won’t allow the user to input any grades until a student is selected. The bracketed areas show how the code is retrieving the student’s admission number and then changing the input states to allow the user to input data.

The function (left) allows the program to import any data collected from the interface window to the SQL database.

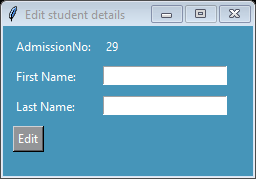
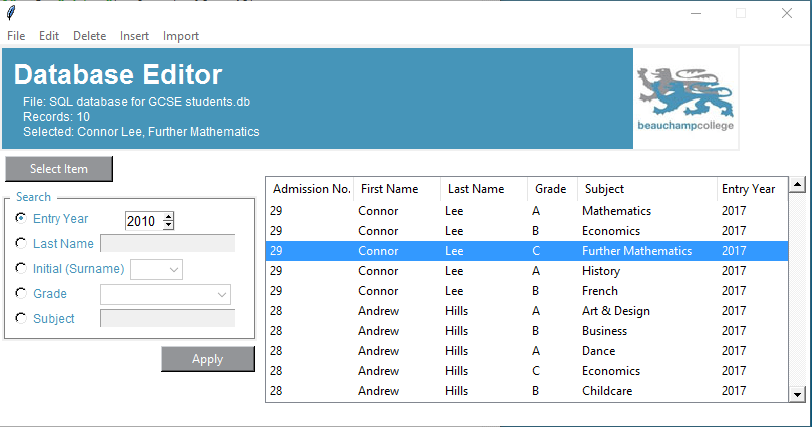
Example of an output interface from the code (above). Main interface with selected student (right)



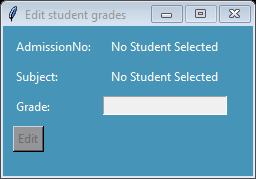


The code (above) creates another instantiation of a class. Although this is very different to the first as it allows the user to input a student’s details and then add those details to the database in the correct format.

The interface (left) is the result of the code being ran. This window is created from the drop-down menu “Input” in the main interface. Once the student has been added to the database the program will tell the user the students AdmissionNo so that they can then use the number to add grades and Subject options for that student



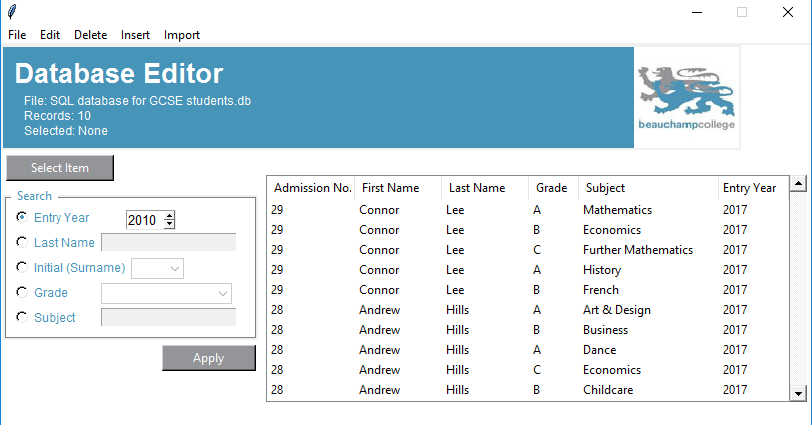
The code (above) creates a third instantiation of a class. This time it allows the user to select a student from the main interface (below) and then edit the name in an interface (left). The class also uses validation to make sure the user has selected a student as highlighted in the code above. The first bracket shows how the code retrieves and checks the selected item. The second shows how the code deals with the Boolean decision (the user can input if student is selected and can’t if no student is selected).

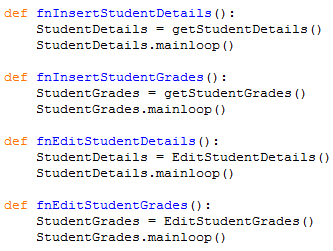


This code searches for whether the user has selected a record from the main interface

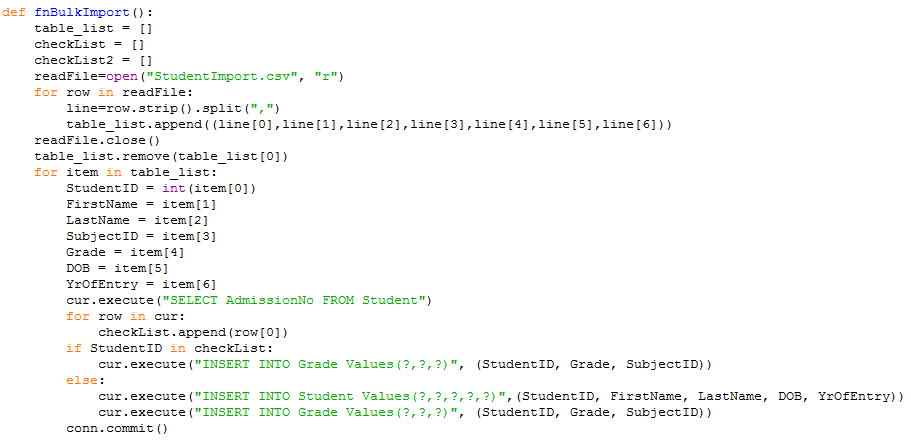
Checks what data is held in the two variables. If a student has been selected then the user can input data.

The code (above) creates the fourth instantiation of a class. This time it creates an interface where the user can edit the students Grade for a subject. If the user hasn’t selected a student from the main interface (below) then the program will return errors informing the user and preventing them from inputting data. However, if the student selects a student the program will run normally.

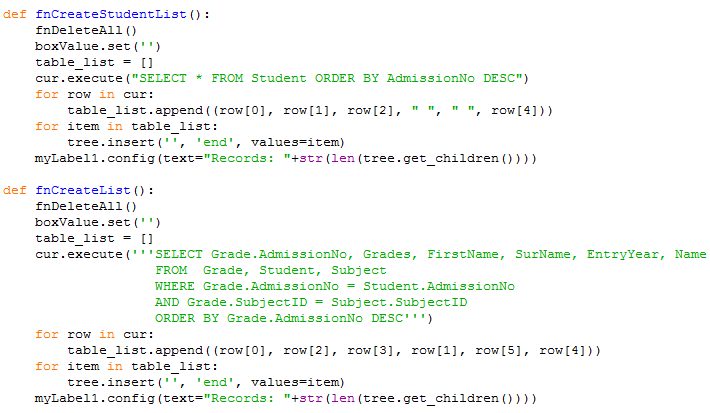




The code (left) will allow the user to call the specific classes (shown previously) to either add to the database or edit data currently stored in the database.

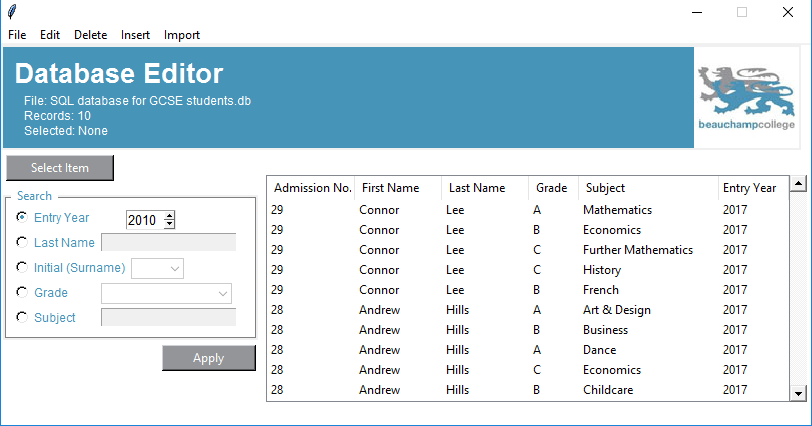


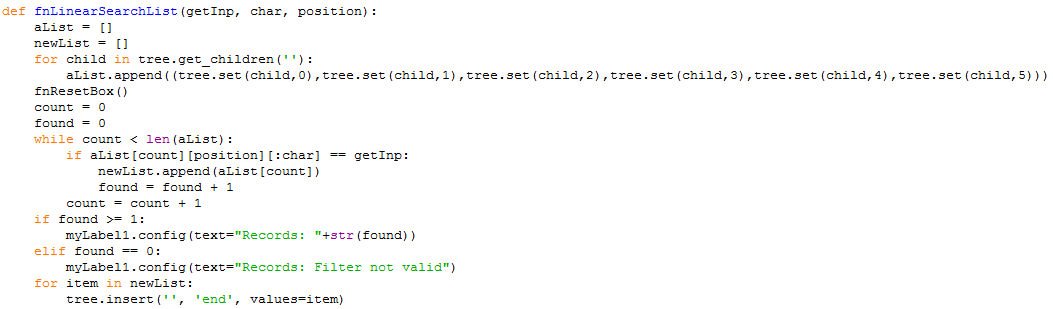
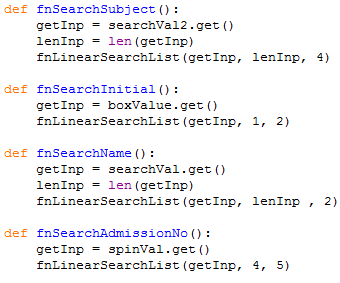
The code (above) allows the user to write data into a CSV file and then import that information into the Python program. The code then adjusts the format so that the data can be inputted into the SQL database.



The code (above) allows the user to view all of the necessary information that is stored about each student. It creates the table that can be seen in the main interface (below). This then allows the user to search through the table for the specific search functions that the user wants.

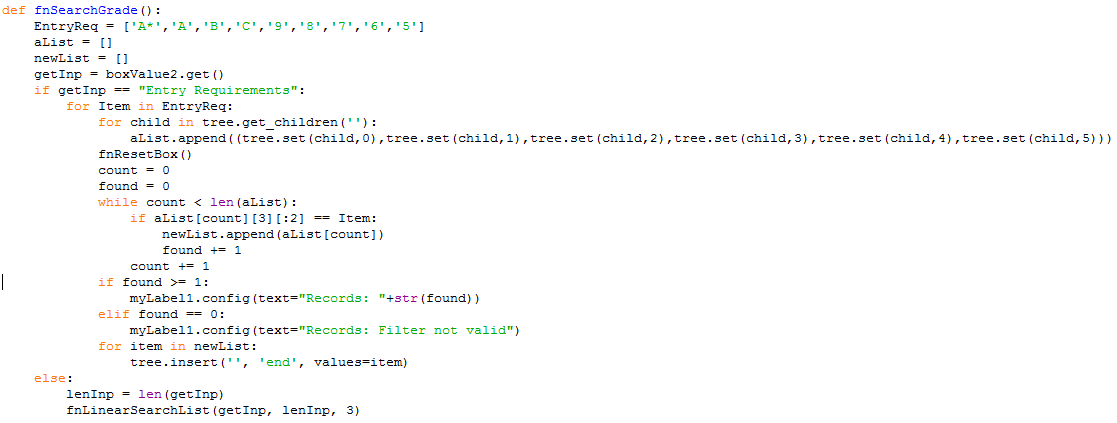
The code (left) allows the user to view all the student records that are stored in the database under the Student table.



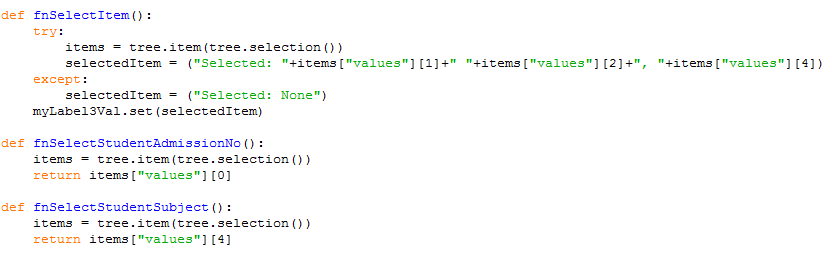


The code (left) enables the user to search through the produced table list and then create an output based on what has been found in the list. I decided to use a linear search as it can effectively search for multiple fields that have the same search criteria, compared to a binary search that would only search for one field and then stop the search.

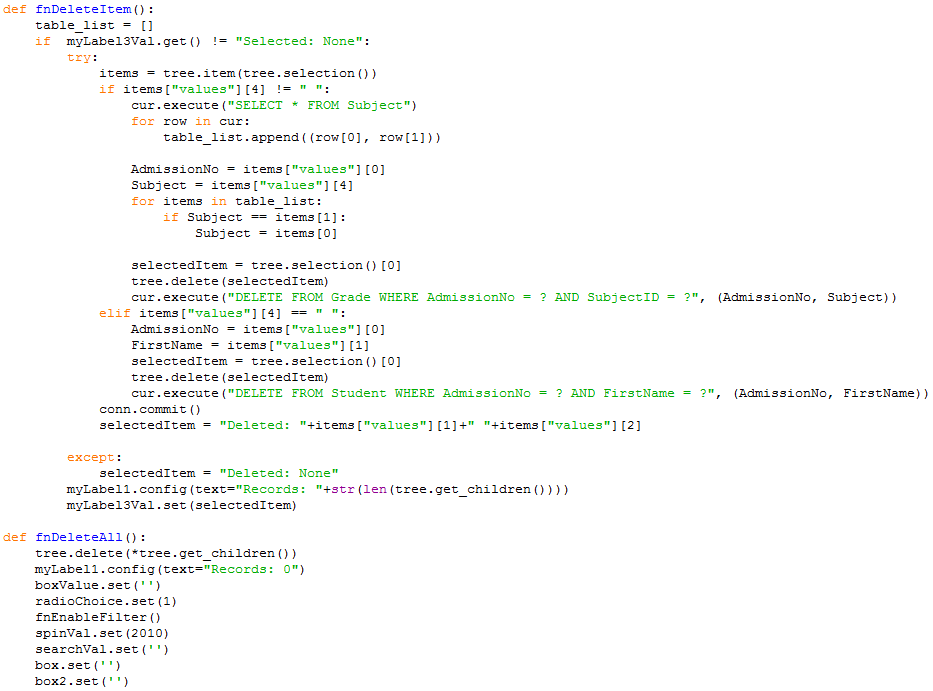
The code (left) takes the user’s search input and then passes the specific information to the function shown above to then search through the table list and produce an output.



The code (above) allows the user to search through the table list for specific grades or the entry requirements for the college. Depending on what the input is the code will either pass data to the linear search (shown earlier) if a specific grade is chosen. Or if the entry requirement input is chosen then a custom linear search will run which will search through the table list using data from the list specified at the start of the function. This will then create an output of all grades that match the list.



The code (above) allows the user to select a student. The first function creates an output for the user to view based on the users chosen student. The second function allows the program to pass the students AdmissionNo to the instantiations of classes that have been created earlier in the program. The third function allows the program to pass the students SubjectID to the instantiations of classes that have been created earlier in the program.



The code (left) allows the user to select a student from the table list and then delete that student from the database. I have included this feature in case the user inputs a student incorrectly and the field that is incorrect isn’t eligible for change through the main interface.

The code (left) clears the table list whenever a search function or a load function is ran. This prevents having multiple data in the table at once. It also helps remove superfluous data as well.

## Final Solution

1. **from** tkinter **import** \*
2. **import** tkinter.ttk **as** ttk
3. **import** sqlite3 **as** lite
4. **import** tkinter **as** tk
5. **from** collections **import** Counter
7. conn = lite.connect("SQL database for GCSE students.db")
8. cur = conn.cursor()
10. **class** getStudentGrades(tk.Tk):
11. **def** \_\_init\_\_(self):
12. tk.Tk.\_\_init\_\_(self)
13. self.config(bg="#4695b9")
14. self.geometry("250x150")
15. self.title("Input Grades")
16. self.resizable(0,0)
17. **try**:
18. self.SelectedItem = fnSelectStudentAdmissionNo()
19. **except**:
20. mylabel2 = tk.Label(self, text="No Student Selected", bg="#4695b9", fg="white").place(x=105,y=10)
21. self.SelectedItem = None
23. self.lblAdminNo = tk.Label(self, text="AdmissionNo:", bg="#4695b9", fg="white").place(x=10,y=10)
25. self.lblValidate = tk.Label(self, text=self.SelectedItem, bg="#4695b9", fg="white").place(x=100,y=10)
27. self.myComboSubjectboxVal = StringVar()
28. self.myComboSubjectbox = ttk.Combobox(self,textvariable=self.myComboSubjectboxVal, width=22,height=4, background="white", state=DISABLED)
29. self.myComboSubjectbox['values'] = ("Art & Design","Business","Childcare","Computer Science","Dance","Double Science","Drama","Economics","English Language",\
30. "English Literature","French","Further Mathematics","Geography","German","Graphic Communication","Health & Social Care",\
31. "History","Hospitality & Catering","ICT","Mandarin","Mathematics","Media","Performing Arts","P.E. (GCSE)","Product Design",\
32. "R.E.","Textiles & Fashion","Triple Science")
33. self.myComboSubjectbox.place(x=85,y=40)
34. self.lblSelect = tk.Label(self, text="Subject:", bg="#4695b9", fg="white").place(x=10,y=40)

37. self.myComboGradeboxVal = StringVar()
38. self.myComboGradebox = ttk.Combobox(self,textvariable=self.myComboGradeboxVal, width=6,height=4, background="white", state=DISABLED)
39. self.myComboGradebox['values'] = ("A\*","A","B","C","D","E","U","9","8","7","6","5","4","3","2","1","0")
40. self.myComboGradebox.place(x=85,y=70)
41. self.lblGrade = tk.Label(self, text="Grade:", bg="#4695b9", fg="white").place(x=10,y=70)
43. self.BtnAppend = tk.Button(self, text="Add", bg="#939598", fg="white",command=self.fnGetStudentGrades, state=DISABLED)
44. self.BtnAppend.place(x=10,y=100)

47. **if** self.SelectedItem == None:
48. self.myComboSubjectbox.config(state=DISABLED)
49. self.myComboGradebox.config(state=DISABLED)
50. self.BtnAppend.config(state=DISABLED)
51. **elif** self.SelectedItem != None:
52. self.myComboSubjectbox.config(state=NORMAL)
53. self.myComboGradebox.config(state=NORMAL)
54. self.BtnAppend.config(state=NORMAL)



59. **def** fnGetStudentGrades(self):
60. ('''This function takes the user input of a students grades and then converts it into SQL format
61. to be inputted into the database''')
62. AdmissionNo = self.SelectedItem
63. Subject = self.myComboSubjectbox.get()
64. Grade = self.myComboGradebox.get()
65. table\_list = []
66. Found = False
67. cur.execute("SELECT \* FROM Subject")
68. **for** row **in** cur:
69. table\_list.append((row[0], row[1]))
70. **for** item **in** table\_list:
71. **if** Subject == item[1]:
72. Subject = int(item[0])
73. Found = True
74. **if** Found == False:
75. self.lblValidate2 = tk.Label(self, text="Invalid", bg="#4695b9", fg="white").place(x=70,y=100)
76. **elif** Found == True:
77. cur.execute("INSERT INTO Grade Values(?,?,?)", (AdmissionNo,Subject,Grade))
78. conn.commit()
80. **class** getStudentDetails(tk.Tk):
81. **def** \_\_init\_\_(self):
82. tk.Tk.\_\_init\_\_(self)
83. self.config(bg="#4695b9")
84. self.geometry("310x170")
85. self.title("Input Student")
86. self.resizable(0,0)
88. self.txtEntry1 = tk.Entry(self)
89. self.lblFName = tk.Label(self, text="First Name:", bg="#4695b9", fg="white").place(x=10,y=10)
90. self.txtEntry1.place(x=80,y=10)
92. self.txtEntry2 = tk.Entry(self)
93. self.lblLName = tk.Label(self, text="Last Name:", bg="#4695b9", fg="white").place(x=10,y=40)
94. self.txtEntry2.place(x=80,y=40)
96. self.lblDOB = tk.Label(self, text="DOB(YYYY,MM,DD):", bg="#4695b9", fg="white").place(x=10,y=70)
97. self.SpinYrVal = tk.Spinbox(self, from\_=2000, to=2040,width=5, font=("",10),background="white")
98. self.SpinYrVal.place(x=130,y=70)
100. self.SpinMonthval = tk.Spinbox(self, from\_=1, to=12,width=5, font=("",10),background="white")
101. self.SpinMonthval.place(x=190,y=70)
103. self.SpinDayVal = tk.Spinbox(self, from\_=1, to=31,width=5, font=("",10),background="white")
104. self.SpinDayVal.place(x=250,y=70)
106. self.lblYrOfEntry = tk.Label(self, text="Year Of Entry:", bg="#4695b9", fg="white").place(x=10, y=100)
107. self.SpinEntryVal = tk.Spinbox(self, from\_=2017, to=2040,width=5, font=("",10),background="white")
108. self.SpinEntryVal.place(x=90,y=100)
110. self.lblAdminNo = tk.Label(self, text="AdmissionNo: ", bg="#4695b9", fg="white").place(x=50,y=130)
112. self.BtnAppend = tk.Button(self, text="Add", bg="#939598", fg="white", command=self.fnGetStudentDetails)
113. self.BtnAppend.place(x=10,y=130)

116. **def** fnGetStudentDetails(self):
117. ('''This function takes the user input of a students details and then converts it into SQL format
118. to be inputted into the database''')
119. table\_list = []
120. AdmissionNo = None
121. DOB = self.SpinDayVal.get()+"/"+self.SpinMonthval.get()+"/"+self.SpinYrVal.get()
122. YrOfEntry = self.SpinEntryVal.get()
123. **try**:
124. FirstName = str(self.txtEntry1.get())
125. SurName = str(self.txtEntry2.get())
126. cur.execute("INSERT INTO Student Values(?,?,?,?,?)", (AdmissionNo,FirstName,SurName,DOB,YrOfEntry))
127. conn.commit()
128. cur.execute("SELECT \* FROM Student")
129. **for** row **in** cur:
130. table\_list.append(row[0])
131. self.lblAdminNo = tk.Label(self, text=table\_list[len(table\_list)-1], bg="#4695b9", fg="white").place(x=130,y=130)
132. **except**:
133. self.lblValidate = tk.Label(self, text="Not a valid name").place(x=120,y=25)


137. **class** EditStudentDetails(tk.Tk):
138. **def** \_\_init\_\_(self):
139. tk.Tk.\_\_init\_\_(self)
140. self.config(bg="#4695b9")
141. self.geometry("250x150")
142. self.title("Edit student details")
143. self.resizable(0,0)
145. **try**:
146. self.SelectedItem = fnSelectStudentAdmissionNo()
147. **except**:
148. lblValidate1 = tk.Label(self, text="No Student Selected", bg="#4695b9", fg="white").place(x=105,y=10)
149. self.SelectedItem = None
151. self.lblAdminNo = tk.Label(self, text="AdmissionNo:", bg="#4695b9", fg="white").place(x=10,y=10)
152. self.lblValidate2 = tk.Label(self, text=self.SelectedItem, bg="#4695b9", fg="white").place(x=100,y=10)
154. self.txtEntry1 = tk.Entry(self, state=DISABLED)
155. self.lblFName = tk.Label(self, text="First Name:", bg="#4695b9", fg="white").place(x=10,y=40)
156. self.txtEntry1.place(x=100,y=40)
158. self.txtEntry2 = tk.Entry(self,state=DISABLED)
159. self.lblLName = tk.Label(self, text="Last Name:", bg="#4695b9", fg="white").place(x=10,y=70)
160. self.txtEntry2.place(x=100,y=70)
162. self.BtnEdit = tk.Button(self, text="Edit", bg="#939598", fg="white", command=self.fnChangeStudentInfo, state=DISABLED)
163. self.BtnEdit.place(x=10,y=100)
165. **if** self.SelectedItem == None:
166. self.txtEntry1.config(state=DISABLED)
167. self.txtEntry2.config(state=DISABLED)
168. self.BtnEdit.config(state=DISABLED)
169. **elif** self.SelectedItem != None:
170. self.txtEntry1.config(state=NORMAL)
171. self.txtEntry2.config(state=NORMAL)
172. self.BtnEdit.config(state=NORMAL)

175. **def** fnChangeStudentInfo(self):
176. ('''This function reads a users input and then changes a specific students details in the SQL database''')
177. AdmissionNo = self.SelectedItem
178. FirstName = self.txtEntry1.get()
179. LastName = self.txtEntry1.get()
180. cur.execute("UPDATE Student SET FirstName = ?, SurName = ? WHERE AdmissionNo = ?", (FirstName,LastName,AdmissionNo))
181. conn.commit()

184. **class** EditStudentGrades(tk.Tk):
185. **def** \_\_init\_\_(self):
186. tk.Tk.\_\_init\_\_(self)
187. self.config(bg="#4695b9")
188. self.geometry("250x150")
189. self.title("Edit student grades")
190. self.resizable(0,0)
192. **try**:
193. self.SelectedNumber = fnSelectStudentAdmissionNo()
194. self.SelectedSubject = fnSelectStudentSubject()
195. **except**:
196. lblValidate1 = tk.Label(self, text="No Student Selected", bg="#4695b9", fg="white").place(x=105,y=10)
197. lblValidate2 = tk.Label(self, text="No Student Selected", bg="#4695b9", fg="white").place(x=105,y=40)
198. self.SelectedNumber = None
199. self.SelectedSubject = None
201. self.lblAdminNo = tk.Label(self, text="AdmissionNo:", bg="#4695b9", fg="white").place(x=10,y=10)
202. self.lblSubject = tk.Label(self, text="Subject:", bg="#4695b9", fg="white").place(x=10,y=40)
203. self.lblValidate3 = tk.Label(self,text=self.SelectedNumber, bg="#4695b9", fg="white").place(x=100,y=10)
204. self.lblValidate4 = tk.Label(self,text=self.SelectedSubject, bg="#4695b9", fg="white").place(x=100,y=40)
206. self.myComboGradeboxVal = StringVar()
207. self.myComboGradebox = ttk.Combobox(self,textvariable=self.myComboGradeboxVal, width=6,height=4, background="white", state=DISABLED)
208. self.myComboGradebox['values'] = ("A\*","A","B","C","D","E","U","9","8","7","6","5","4","3","2","1","0")
209. self.myComboGradebox.place(x=100,y=70)
210. self.lblGrade = tk.Label(self, text="Grade:", bg="#4695b9", fg="white").place(x=10,y=70)
212. self.BtnEdit = tk.Button(self, text="Edit", bg="#939598", fg="white", command=self.fnChangeStudentInfo, state=DISABLED)
213. self.BtnEdit.place(x=10,y=100)
215. **if** self.SelectedNumber == None **and** self.SelectedSubject == None:
216. self.myComboGradebox.config(state=DISABLED)
217. self.BtnEdit.config(state=DISABLED)
218. **elif** self.SelectedNumber != None **and** self.SelectedSubject != None:
219. self.myComboGradebox.config(state=NORMAL)
220. self.BtnEdit.config(state=NORMAL)
222. **def** fnChangeStudentInfo(self):
223. ('''This function reads a users input and then changes a specific students grades in the SQL database''')
224. Subject\_list = []
225. Grades = self.myComboGradebox.get()
226. AdmissionNo = self.SelectedNumber
227. SubjectID = self.SelectedSubject
228. cur.execute("SELECT \* FROM Subject")
229. **for** row **in** cur:
230. Subject\_list.append((int(row[0]), row[1]))
231. **for** item **in** Subject\_list:
232. **if** SubjectID == item[1]:
233. SubjectID = int(item[0])
234. cur.execute("UPDATE Grade SET Grades = ? WHERE AdmissionNo = ? AND SubjectID = ?", (Grades, AdmissionNo, SubjectID))
235. conn.commit()
237. **def** fnInsertStudentDetails():
238. StudentDetails = getStudentDetails()
239. StudentDetails.mainloop()
241. **def** fnInsertStudentGrades():
242. StudentGrades = getStudentGrades()
243. StudentGrades.mainloop()
245. **def** fnEditStudentDetails():
246. StudentDetails = EditStudentDetails()
247. StudentDetails.mainloop()
249. **def** fnEditStudentGrades():
250. StudentGrades = EditStudentGrades()
251. StudentGrades.mainloop()
253. **def** fnBulkImport():
254. ('''This function allows the program to read from a CSV file and then input all data stored in the file into
255. the SQL database''')
256. table\_list = []
257. checkList = []
258. checkList2 = []
259. readFile=open("StudentImport.csv", "r")
260. **for** row **in** readFile:
261. line=row.strip().split(",")
262. table\_list.append((line[0],line[1],line[2],line[3],line[4],line[5],line[6]))
263. readFile.close()
264. table\_list.remove(table\_list[0])
265. **for** item **in** table\_list:
266. StudentID = int(item[0])
267. FirstName = item[1]
268. LastName = item[2]
269. SubjectID = item[3]
270. Grade = item[4]
271. DOB = item[5]
272. YrOfEntry = item[6]
273. cur.execute("SELECT AdmissionNo FROM Student")
274. **for** row **in** cur:
275. checkList.append(row[0])
276. **if** StudentID **in** checkList:
277. cur.execute("INSERT INTO Grade Values(?,?,?)", (StudentID, Grade, SubjectID))
278. **else**:
279. cur.execute("INSERT INTO Student Values(?,?,?,?,?)",(StudentID, FirstName, LastName, DOB, YrOfEntry))
280. cur.execute("INSERT INTO Grade Values(?,?,?)", (StudentID, Grade, SubjectID))
281. conn.commit()
282. readFile.close()
284. **def** fnMergeSort(alist):
285. ('''This function allows the program to sort the data retrieved into order by the students AdmissionNo''')
286. **if** len(alist)>1:
287. mid = len(alist)//2
288. lefthalf = alist[:mid]
289. righthalf = alist[mid:]
290. fnMergeSort(righthalf)
291. fnMergeSort(lefthalf)
292. i=0
293. j=0
294. k=0
295. **while** i < len(lefthalf) **and** j < len(righthalf):
296. **if** lefthalf[i] < righthalf[j]:
297. alist[k]=lefthalf[i]
298. i=i+1
299. **else**:
300. alist[k]=righthalf[j]
301. j=j+1
302. k=k+1
303. **while** i < len(lefthalf):
304. alist[k]=lefthalf[i]
305. i=i+1
306. k=k+1
307. **while** j < len(righthalf):
308. alist[k]=righthalf[j]
309. j=j+1
310. k=k+1
312. **def** fnCreateStudentList():
313. ('''This function allows the program to retrieve all data from the database stored in the Student table''')
314. fnDeleteAll()
315. boxValue.set('')
316. table\_list = []
317. cur.execute("SELECT \* FROM Student")
318. **for** row **in** cur:
319. table\_list.append((row[0], row[1], row[2], " ", " ", row[4]))
320. fnMergeSort(table\_list)
321. **for** item **in** table\_list:
322. tree.insert('', 'end', values=item)
323. lblTblLength.config(text="Records: "+str(len(tree.get\_children())))
325. **def** fnCreateList():
326. ('''This function allows the program to retrieve all data from the database that shows the user the
327. required information''')
328. fnDeleteAll()
329. boxValue.set(" ")
330. table\_list = []
331. cur.execute('''SELECT Grade.AdmissionNo, Grades, FirstName, SurName, EntryYear, Name
332. FROM  Grade, Student, Subject
333. WHERE Grade.AdmissionNo = Student.AdmissionNo
334. AND Grade.SubjectID = Subject.SubjectID''')
335. **for** row **in** cur:
336. table\_list.append((row[0], row[2], row[3], row[1], row[5], row[4]))
337. fnMergeSort(table\_list)
338. **for** item **in** table\_list:
339. tree.insert('', 'end', values=item)
340. lblTblLength.config(text="Records: "+str(len(tree.get\_children())))
342. **def** fnLinearSearchList(getInp, char, position):
343. ('''This function allows the program to search through the table list on the interface and then output
344. based on the records that are found''')
345. aList = []
346. newList = []
347. **for** child **in** tree.get\_children(''):
348. aList.append((tree.set(child,0),tree.set(child,1),tree.set(child,2),tree.set(child,3),tree.set(child,4),tree.set(child,5)))
349. fnResetBox()
350. count = 0
351. found = 0
352. **while** count < len(aList):
353. **if** aList[count][position][:char] == getInp:
354. newList.append(aList[count])
355. found = found + 1
356. count = count + 1
357. **if** found >= 1:
358. lblTblLength.config(text="Records: "+str(found))
359. **elif** found == 0:
360. lblTblLength.config(text="Records: Filter not valid")
361. **for** item **in** newList:
362. tree.insert('', 'end', values=item)

365. **def** fnSearchGrade():
366. ('''This function calls the LinearSearch function but calling for a specific grade''')
367. EntryReq = ['A\*','A','B','C','9','8','7','6','5']
368. aList = []
369. newList = []
370. AdminList = []
371. getInp = boxGradeVal.get()
372. count = 0
373. **if** getInp == "Entry Requirements":
374. **for** Item **in** EntryReq:
375. **for** child **in** tree.get\_children(''):
376. aList.append((tree.set(child,0),tree.set(child,1),tree.set(child,2),tree.set(child,3),tree.set(child,4),tree.set(child,5)))
377. fnResetBox()
378. count = 0
379. found = 0
380. **while** count < len(aList):
381. **if** aList[count][3][:2] == Item:
382. newList.append(aList[count])
383. found += 1
384. count += 1
385. **if** found >= 1:
386. lblTblLength.config(text="Records: "+str(found))
387. **elif** found == 0:
388. lblTblLength.config(text="Records: Filter not valid")
389. ##        print("Count for 28: ", newList[1].count('28'))
390. ##        cnt = Counter()
391. ##        for AdminNo in newList:
392. ##            AdminList.append(AdminNo[0])
393. ##        for item in AdminList:
394. ##            cnt[item] += 1
395. ##        print(cnt)
396. **for** item **in** newList:
397. tree.insert('', 'end', values=item)
398. **else**:
399. lenInp = len(getInp)
400. fnLinearSearchList(getInp, lenInp, 3)

403. **def** fnSearchSubject():
404. ('''This function calls the LinearSearch function but calling for a specific subject''')
405. getInp = boxSubject.get()
406. lenInp = len(getInp)
407. fnLinearSearchList(getInp, lenInp, 4)
409. **def** fnSearchInitial():
410. ('''This function calls the LinearSearch function but calling for a specific student's initial''')
411. getInp = boxValue.get()
412. fnLinearSearchList(getInp, 1, 2)
414. **def** fnSearchName():
415. ('''This function calls the LinearSearch function but calling for a specific surname''')
416. getInp = searchVal.get()
417. lenInp = len(getInp)
418. fnLinearSearchList(getInp, lenInp , 2)
420. **def** fnSearchAdmissionNo():
421. ('''This function calls the LinearSearch function but calling for a specific student''')
422. getInp = spinVal.get()
423. fnLinearSearchList(getInp, 4, 5)
425. **def** fnApplyFilter():
426. **if** checkChoice1.get() == 1:
427. fnSearchAdmissionNo()
428. **if** checkChoice2.get() == 1:
429. fnSearchName()
430. **if** checkChoice3.get() == 1:
431. fnSearchInitial()
432. **if** checkChoice4.get() == 1:
433. fnSearchGrade()
434. **if** checkChoice5.get() == 1:
435. fnSearchSubject()
436. lblSelectVal.set("Selected: None")
437. fnClearSearch()

440. **def** fnSelectItem():
441. ('''This function enables the user to select a student from the interface and then manipulate the data
442. stored on that student as they wish''')
443. **try**:
444. items = tree.item(tree.selection())
445. selectedItem = ("Selected: "+items["values"][1]+" "+items["values"][2]+", "+items["values"][4])
446. **except**:
447. selectedItem = ("Selected: None")
448. lblSelectVal.set(selectedItem)
450. **def** fnSelectStudentAdmissionNo():
451. items = tree.item(tree.selection())
452. **return** items["values"][0]
454. **def** fnSelectStudentSubject():
455. items = tree.item(tree.selection())
456. **return** items["values"][4]
458. **def** fnDeleteItem():
459. ('''This function enables the user to delete a student's information from the database permanently''')
460. table\_list = []
461. **if**  lblSelectVal.get() != "Selected: None":
462. **try**:
463. items = tree.item(tree.selection())
464. **if** items["values"][4] != " ":
465. cur.execute("SELECT \* FROM Subject")
466. **for** row **in** cur:
467. table\_list.append((row[0], row[1]))
469. AdmissionNo = items["values"][0]
470. Subject = items["values"][4]
471. **for** items **in** table\_list:
472. **if** Subject == items[1]:
473. Subject = items[0]
475. selectedItem = tree.selection()[0]
476. tree.delete(selectedItem)
477. cur.execute("DELETE FROM Grade WHERE AdmissionNo = ? AND SubjectID = ?", (AdmissionNo, Subject))
478. **elif** items["values"][4] == " ":
479. AdmissionNo = items["values"][0]
480. FirstName = items["values"][1]
481. selectedItem = tree.selection()[0]
482. tree.delete(selectedItem)
483. cur.execute("DELETE FROM Student WHERE AdmissionNo = ? AND FirstName = ?", (AdmissionNo, FirstName))
484. conn.commit()
485. selectedItem = "Deleted: "+items["values"][1]+" "+items["values"][2]
487. **except**:
488. selectedItem = "Deleted: None"
489. lblTblLength.config(text="Records: "+str(len(tree.get\_children())))
490. lblSelectVal.set(selectedItem)
492. **def** fnDeleteAll():
493. tree.delete(\*tree.get\_children())
494. lblTblLength.config(text="Records: 0")
495. boxValue.set('')
496. fnEnableFilter()
497. spinVal.set(2017)
498. searchVal.set('')
499. box.set('')
500. boxGrade.set('')
501. boxSubject.set('')
502. fnClearSearch()
504. **def** fnResetBox():
505. tree.delete(\*tree.get\_children())
506. lblTblLength.config(text="Records: 0")
508. **def** fnEnableFilter():
509. **if** checkChoice1.get() == 1:
510. spinbox.config(state=NORMAL)
511. **if** checkChoice2.get() == 1:
512. searchBox.config(state=NORMAL)
513. **if** checkChoice3.get() == 1:
514. box.config(state = "readonly")
515. **if** checkChoice4.get() == 1:
516. boxGrade.config(state = "readonly")
517. **if** checkChoice5.get() == 1:
518. boxSubject.config(state = "readonly")
520. **def** fnquitProgram():
521. myExit =messagebox.askyesno(title="Quit",message="Are you sure you want to quit?")
522. **if** myExit > 0:
523. root.destroy()
524. **return**
526. **def** fnaboutProgram():
527. myAboutMessage =messagebox.showinfo("DataBase Editor Version 1.0", "Beauchamp College | AH 2016")
528. **return**
530. **def** fnClearSearch():
531. ('''This function enables to reselect the search criteria on the main interface''')
532. checkChoice1.set(0)
533. checkChoice2.set(0)
534. checkChoice3.set(0)
535. checkChoice4.set(0)
536. checkChoice5.set(0)
537. searchVal.set('')
538. spinVal.set(2017)
539. box.set('')
540. boxGrade.set('')
541. boxSubject.set('')
542. searchBox.config(state=DISABLED)
543. spinbox.config(state=DISABLED)
544. box.config(state=DISABLED)
545. boxGrade.config(state=DISABLED)
546. boxSubject.config(state=DISABLED)

549. root = Tk()
550. root.config(bg="white")
552. table\_header = [' Admission No.', ' First Name',' Last Name',' Grade',' Subject','Entry Year']
553. container = Frame()
554. container.place(x=265,y=130)
555. tree = ttk.Treeview(columns=table\_header,show="headings")
556. vsb = Scrollbar(orient="vertical", command=tree.yview)
557. tree.configure(yscrollcommand=vsb.set)
558. tree.grid(column=0, row=0, in\_=container)
560. vsb.grid(column=1, row=0, sticky='ns', in\_=container)
561. container.grid\_columnconfigure(0, weight=1)
562. container.grid\_rowconfigure(0, weight=1)
564. tree.column(table\_header[0],width=87)
565. tree.column(table\_header[1],width=87)
566. tree.column(table\_header[2],width=87)
567. tree.column(table\_header[3],width=50)
568. tree.column(table\_header[4],width=140)
569. tree.column(table\_header[5],width=70)
571. **for** col **in** table\_header:
572. tree.heading(col, text=col.title())
574. backCanvas= Canvas(root, background = "#4695b9")
575. backCanvas.place(x=0,y=0, width = 800, height = 105)
577. Photo1 = PhotoImage(file="BClogo.gif")
578. myPhoto1 = backCanvas.create\_image(745,53,image=Photo1)

581. lblTitle = Label(root,text = "Database Editor", bg="#4695b9", fg="white", font=("Helvetica", 20, "bold"))
582. lblTitle.place(x=10,y=10)
584. lblDbTitle = Label(root,text = "File: SQL database for GCSE students.db", bg="#4695b9", fg="white", font=("Open\_sansregular",9))
585. lblDbTitle.place(x=20,y=45)
587. lblTblLength = Label(root,text = "Records: 0", bg="#4695b9", fg="white", font=("Open\_sansregular",9))
588. lblTblLength.place(x=20,y=60)
590. lblSelectVal = StringVar()
591. lblSelectVal.set('Selected: None')
592. lblSelect = Label(root,textvariable=lblSelectVal, bg="#4695b9", fg="white", font=("Open\_sansregular",9))
593. lblSelect.place(x=20,y=75)
595. frameCanvas= Canvas(root)
596. frameCanvas.config(bg="white")
597. frameCanvas.place(x=2,y=150, width = 255, height = 145)
598. frameCanvas.create\_rectangle(1, 1, 253, 143, fill='',width = 3,outline="grey")
600. lblSearch = Label(root,text = " Search ", bg="white", fg="#4695b9")
601. lblSearch.place(x=10,y=140)
603. BtnSlctItm = tk.Button(root, text = "Select Item", bg="#939598", fg="white", width = 14,command = fnSelectItem)
604. BtnSlctItm.place(x=5,y=110)
606. searchVal = StringVar()
607. searchBox = Entry(root,textvariable=searchVal, width=22, background="white", state=DISABLED)
608. searchBox.place(x=100,y=188)
609. searchVal.set('')
611. spinVal = StringVar()
612. spinbox = Spinbox(root, from\_=2017, to=2040,width=5, font=("",10), textvariable=spinVal, background="white", state=DISABLED)
613. spinbox.place(x=125,y=165)
615. boxValue = StringVar()
616. box = ttk.Combobox(root,textvariable=boxValue, width=5, height = 4, background="white", state=DISABLED)
617. box['values'] = ('A','B','C','D','E','F','G','H','I','J','K','L','M','N',\
618. 'O', 'P', 'Q','R','S','T','U','V','W','X','Y','Z')
619. box.place(x=130,y=213)
621. boxGradeVal = StringVar()
622. boxGrade = ttk.Combobox(root,textvariable=boxGradeVal, width=18, height = 4, background="white", state=DISABLED)
623. boxGrade['values'] = ('Entry Requirements','A\*','A','B','C','D','E','F','U','9','8','7','6','5','4','3','2','1','0')
624. boxGrade.place(x=95,y=238)
626. boxSubjectVal = StringVar()
627. boxSubject = ttk.Combobox(root,textvariable=boxSubjectVal, width=22,height=4, background="white", state=DISABLED)
628. boxSubject['values'] = ("Art & Design","Business","Childcare","Computer Science","Dance","Double Science","Drama","Economics","English Language",\
629. "English Literature","French","Further Mathematics","Geography","German","Graphic Communication","Health & Social Care",\
630. "History","Hospitality & Catering","ICT","Mandarin","Mathematics","Media","Performing Arts","P.E. (GCSE)","Product Design",\
631. "R.E.","Textiles & Fashion","Triple Science")
632. boxSubject.place(x=95,y=263)
634. checkChoice1 = IntVar()
635. checkChoice2 = IntVar()
636. checkChoice3 = IntVar()
637. checkChoice4 = IntVar()
638. checkChoice5 = IntVar()
639. Checkbutton(root, text="Entry Year", variable=checkChoice1,command = fnEnableFilter, bg="white", fg="#4695b9", font=("Open\_sansregular",9)).place(x=10, y=160)
640. Checkbutton(root, text="Last Name", variable=checkChoice2,command = fnEnableFilter, bg="white", fg="#4695b9", font=("Open\_sansregular",9)).place(x=10, y=185)
641. Checkbutton(root, text="Initial (Surname)", variable=checkChoice3,command = fnEnableFilter, bg="white", fg="#4695b9", font=("Open\_sansregular",9)).place(x=10, y=210)
642. Checkbutton(root, text="Grade", variable=checkChoice4,command = fnEnableFilter, bg="white", fg="#4695b9", font=("Open\_sansregular",9)).place(x=10, y=235)
643. Checkbutton(root, text="Subject", variable=checkChoice5,command = fnEnableFilter, bg="white", fg="#4695b9", font=("Open\_sansregular",9)).place(x=10, y=260)
645. BtnApply = tk.Button(root, text = "Apply", bg="#939598", fg="white",width = 12,command = fnApplyFilter)
646. BtnApply.place(x=161,y=300)
648. BtnClear = tk.Button(root, text = "Clear Search", bg="#939598", fg="white",width = 12, command = fnClearSearch)
649. BtnClear.place(x=3,y=300)
651. MenuBar = Menu(root)
652. FileMenu = Menu(MenuBar,tearoff=0)
653. DeleteMenu = Menu(MenuBar, tearoff=0)
654. InsertMenu = Menu(MenuBar, tearoff=0)
655. EditMenu = Menu(MenuBar, tearoff=0)
656. ImportMenu = Menu(MenuBar, tearoff=0)
658. MenuBar.add\_cascade(label="File", menu=FileMenu)
659. FileMenu.add\_command(label="Load Students with Grades", command=fnCreateList)
660. FileMenu.add\_command(label="Load all Students", command=fnCreateStudentList)
661. FileMenu.add\_command(label="Close", command=fnquitProgram)
662. FileMenu.add\_separator()
663. FileMenu.add\_command(label="About", command=fnaboutProgram)
665. MenuBar.add\_cascade(label="Edit",menu=EditMenu)
666. EditMenu.add\_command(label="Edit Student",command=fnEditStudentDetails)
667. EditMenu.add\_command(label="Edit Grades",command=fnEditStudentGrades)
669. MenuBar.add\_cascade(label="Delete", menu=DeleteMenu)
670. DeleteMenu.add\_command(label="Delete Item", command=fnDeleteItem)
672. MenuBar.add\_cascade(label="Insert", menu=InsertMenu)
673. InsertMenu.add\_command(label="Student Details", command=fnInsertStudentDetails)
674. InsertMenu.add\_command(label="Student Grades", command=fnInsertStudentGrades)
676. MenuBar.add\_cascade(label="Import", menu=ImportMenu)
677. ImportMenu.add\_command(label="Bulk Import", command=fnBulkImport)
678. root.config(menu=MenuBar)

681. w = 810
682. h = 380
683. sw = root.winfo\_screenwidth()
684. sh = root.winfo\_screenheight()
685. x = (sw - w)/2
686. y = (sh - h)/2
687. root.geometry('%dx%d+%d+%d' % (w,h,x,y))
688. root.title("")
689. root.resizable(0,0)
690. root.protocol("WM\_DELETE\_WINDOW", fnquitProgram)
691. root.mainloop()

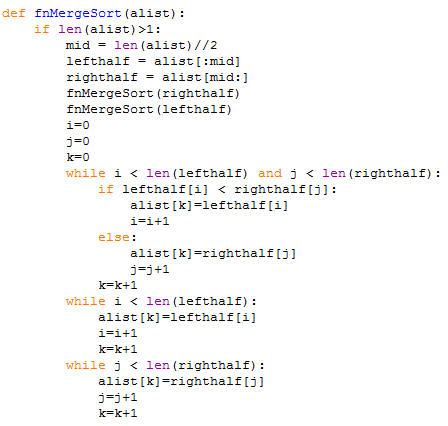
## Data Dictionary

|  |  |  |
| --- | --- | --- |
| Global Variable Name | Variable Data Type | Variable Description |
| radioChoice | INTEGER | To store the choice of the user to search through the database. It is based on the Tkinter variable class to provide an INTEGER variable connected to radioChoice widget |
| boxSubjectVal | STRING | To store all subjects that the database will recognise. It is based on the Tkinter variable class to provide an STRING variable connected to boxSubjectVal widget |
| boxGradeVal | STRING | To store any grades that the student stored in the database can achieve. It is based on the Tkinter variable class to provide an STRING variable connected to boxGradeVal widget |
| boxValue | STRING | To store all letters of the alphabet that the user can then use to search through the database by initials. It is based on the Tkinter variable class to provide an STRING variable connected to boxValue widget |

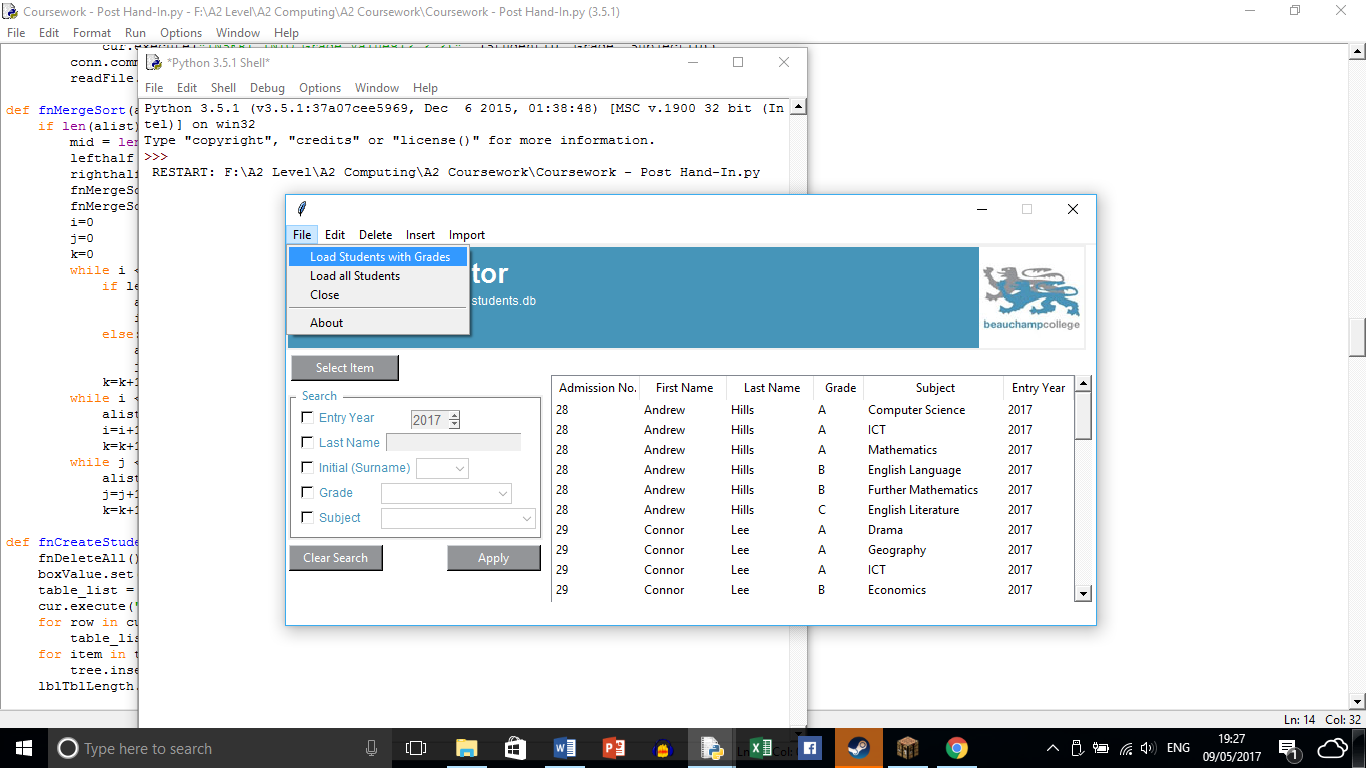
|  |  |  |
| --- | --- | --- |
| Local Variable Name | Variable Data Type | Variable Description |
| table\_list | List | This allows the program to store all data that is retrieved from the database and then output it to the interface for the user to interact with |
| StudentID | Integer | To store the primary identifier of each student |
| FirstName | String | To store the firstname of each student held in the database |
| LastName | String | To store the lastname of each student held in the database |
| DOB | Date/Time | To store the date of birth of each student held in the database |
| YrOfEntry | Real | Stores the year that the student entered the Sixth Form Centre |

## Sorting

In my final solution I decided that when the program is creating the list in the main interface it should sort the data that has been pulled from the database so that it is easily understandable for the user. To do this I created a merge sort (shown below).



This allows the program to sort all the incoming data and then output it to the user.



**Source:** <http://interactivepython.org/courselib/static/pythonds/SortSearch/TheMergeSort.html>

# Testing

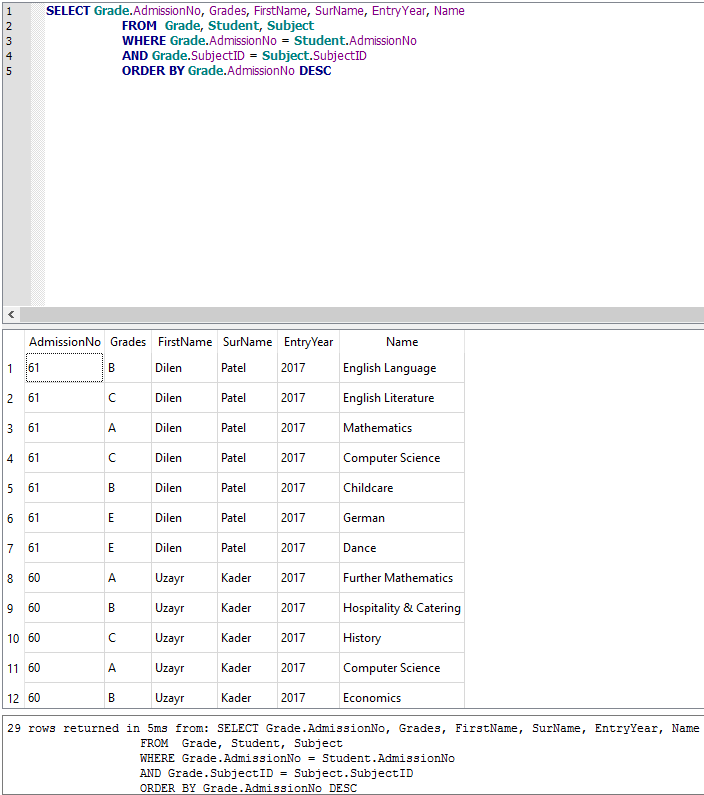
## Test Strategy

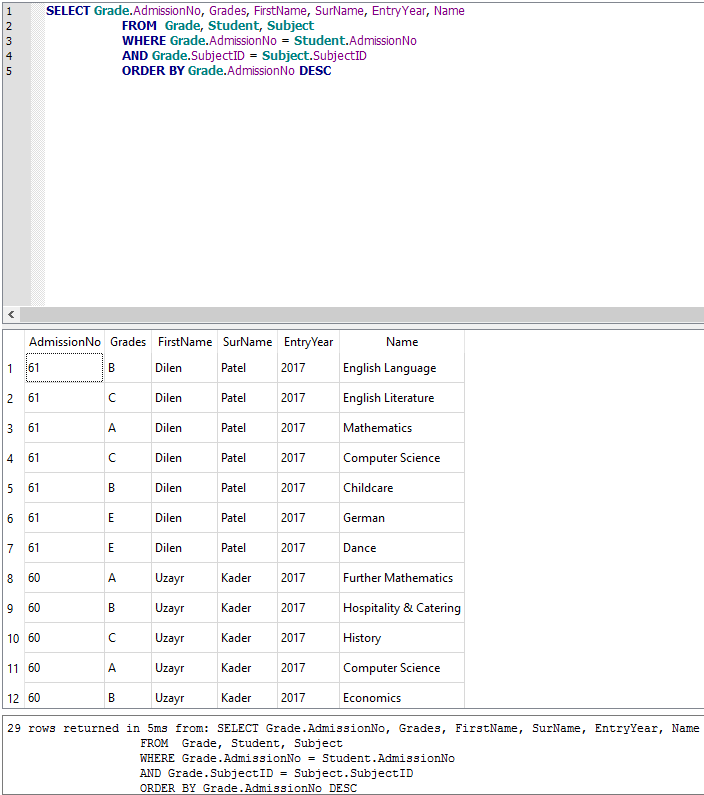
To make sure that the program runs as expected by the client, the following strategy will be followed:

1. Requirement testing. This will be used to make sure that all requirements are met successfully and that they perform the way that the client intended. This testing will be carried out by myself. Every test carried out will identify the system requirement being tested.
2. Functional Testing. This will be done in two ways. First will be done within the requirement testing. Second will be done during implementation of the program (testing as solution was created).
3. User testing. This is where I will ask my client to test the solution that I have created and then give feedback on how to improve the program that I have created. The feedback that I ask for will be based on; the efficiency of the program, how easy the solution is to use, have the client user requirements been met and any conditional features that could potentially improve the solution (GUI improvements).
4. I will also be using the following types of test data:
   1. Typical Erroneous Extreme. Where “typical” data is accepted by the program, “Erroneous” is rejected by the validation in the program and “Extreme” is testing the boundary data e.g. The user can input “1” and “12” in reference to months but cannot input “0”, “13”, “E”, “£”.

### Requirement Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **System Requirement** | **Test purpose** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **1.** | Create a Standard Query Language (SQL) database that holds the component details | To test the relationships between 3 separate tables | The foreign keys and primary keys connect the relevant tables in the correct way and output all of the required data | The primary keys and foreign keys connected correctly. All necessary data was outputted | Pass | None |



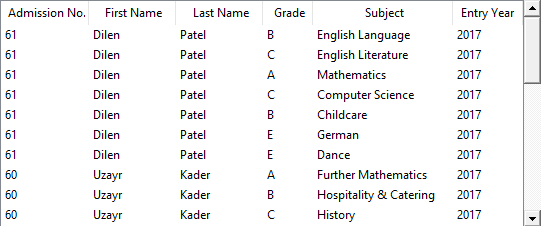
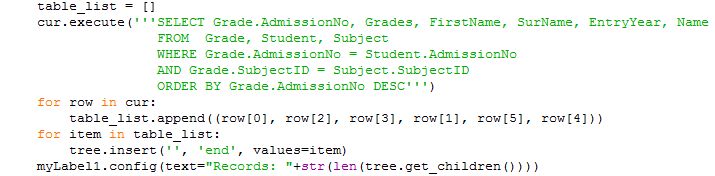


Above is a screenshot of the SQL statement that is needed by the program to retrieve the main information from the database. On the right is the actual outcome of the statement. It has retrieved all of the necessary information that can then be formatted by the Python program to be viewed by the user. See page 17 for other evidence of meeting the requirement.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **System Requirement** | **Test purpose** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **2.** | Create a link in the Python code that can connect with the SQL database | To test that the Python program can communicate with the SQL database | The Python program retrieves the records stored in the database | The Python program retrieves the records stored in the database | Pass | None |

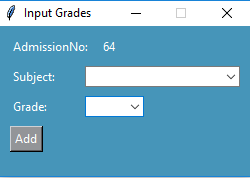


These two lines of code allow the program to set up a connection with the database that can then be exploited



The code above allows the program to exploit the link between Python and SQL and retrieve records from the database. On the right is the output table as a direct result of the Python code above. It outputs all of the necessary data to the interface for the user to then view.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **System Requirement** | **Test purpose** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **3.** | Read any user inputs that are given | To make sure that the program can handle any input that is given. | If the input is invalid then the user is given an error message. If the input is valid then the correct output is displayed. | Any search inputs are handled correctly. However, when the grades are inputted a number can be inputted that is <0 and >9. | Search Criteria (Pass). Student Input (Fail) | Edit the code so that the user cannot input a number <0 and >9 when adding grades for a particular student. |



NOTE: Post-testing I have changed the code so that the user can input the student’s grades within the valid range. The drop down box highlighted by blue only allows an input of A\* to E and U or 1 to 9.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **System Requirement** | **Test purpose** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **4.** | Create drop down menus that the user can exploit | To make sure that the user can navigate around the program quickly and efficiently | The drop-down menus work properly and create the desired outcome that the user can then use how they want | The drop-down menus work properly and create the desired outcome that the user can then use how they want | Pass | None |

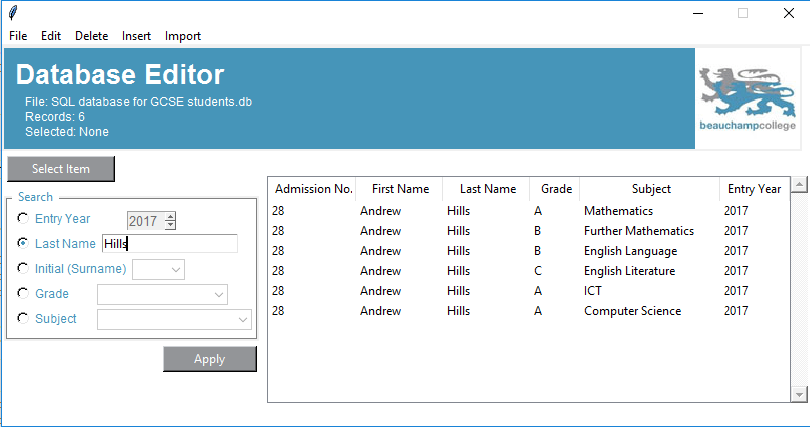
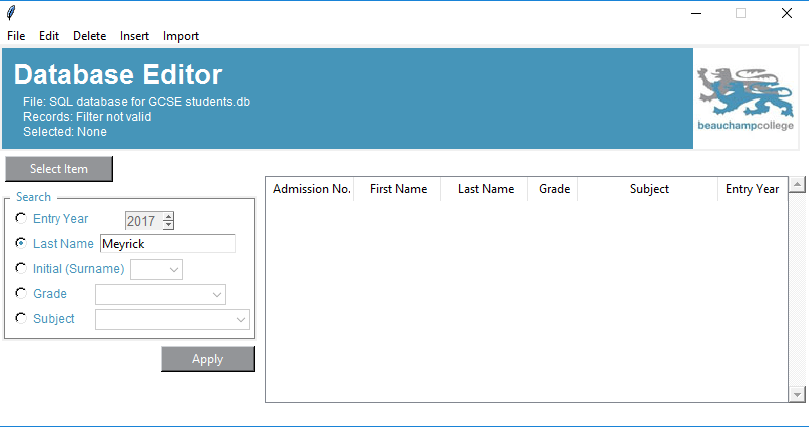
NOTE: To view screenshots of what output is created on each drop-down menu see System Structure Flowchart.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **System Requirement** | **Test purpose** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **5.** | Construct outputs determined by the input and produce an output | To make sure that the program creates the correct output based on what the user has chosen to input | The program can handle all inputs that are given effectively. If the input is invalid then the program gives the user feedback | The program can handle all inputs that are given effectively. If the input is invalid then the program gives the user feedback | Pass | None |

**VALID INPUT**

**INVALID INPUT**

Error message given

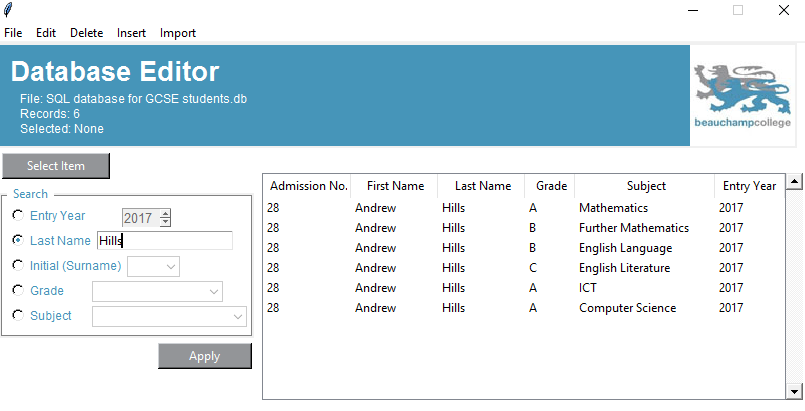


Output created from valid search

No outputs created from invalid search

### Functional Testing

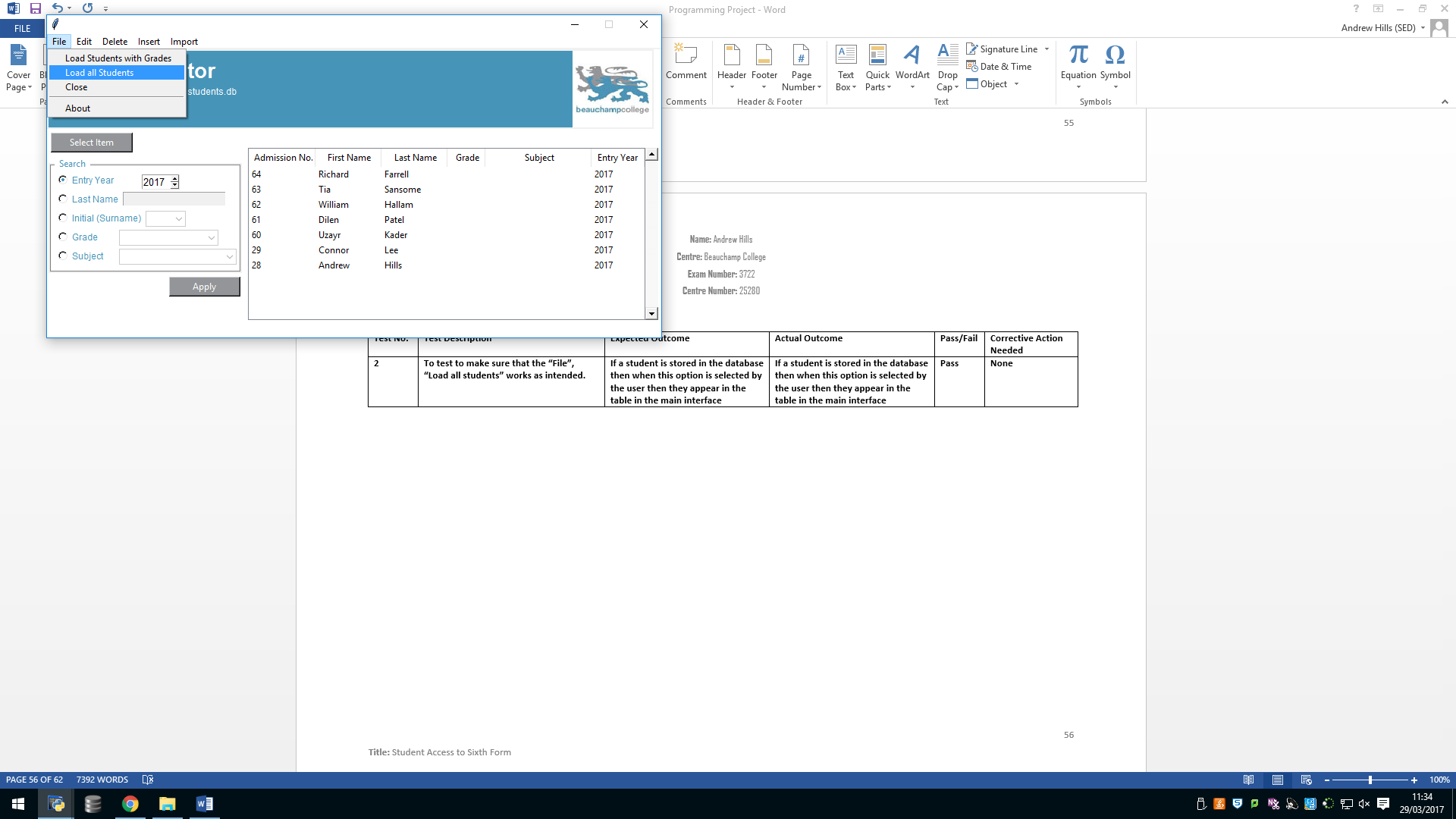
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **1** | **To test that the search “Last Name” feature works** | **If the student is present in the database then their data is presented in the output** | **If the student is present in the database then their data is presented in the output** | **Pass** | **None** |



When the user searches by a name that is stored in the database the information is displayed to the user.

I have tested the other search features in a similar fashion to “Last Name” and they all work the way that they were intended to.

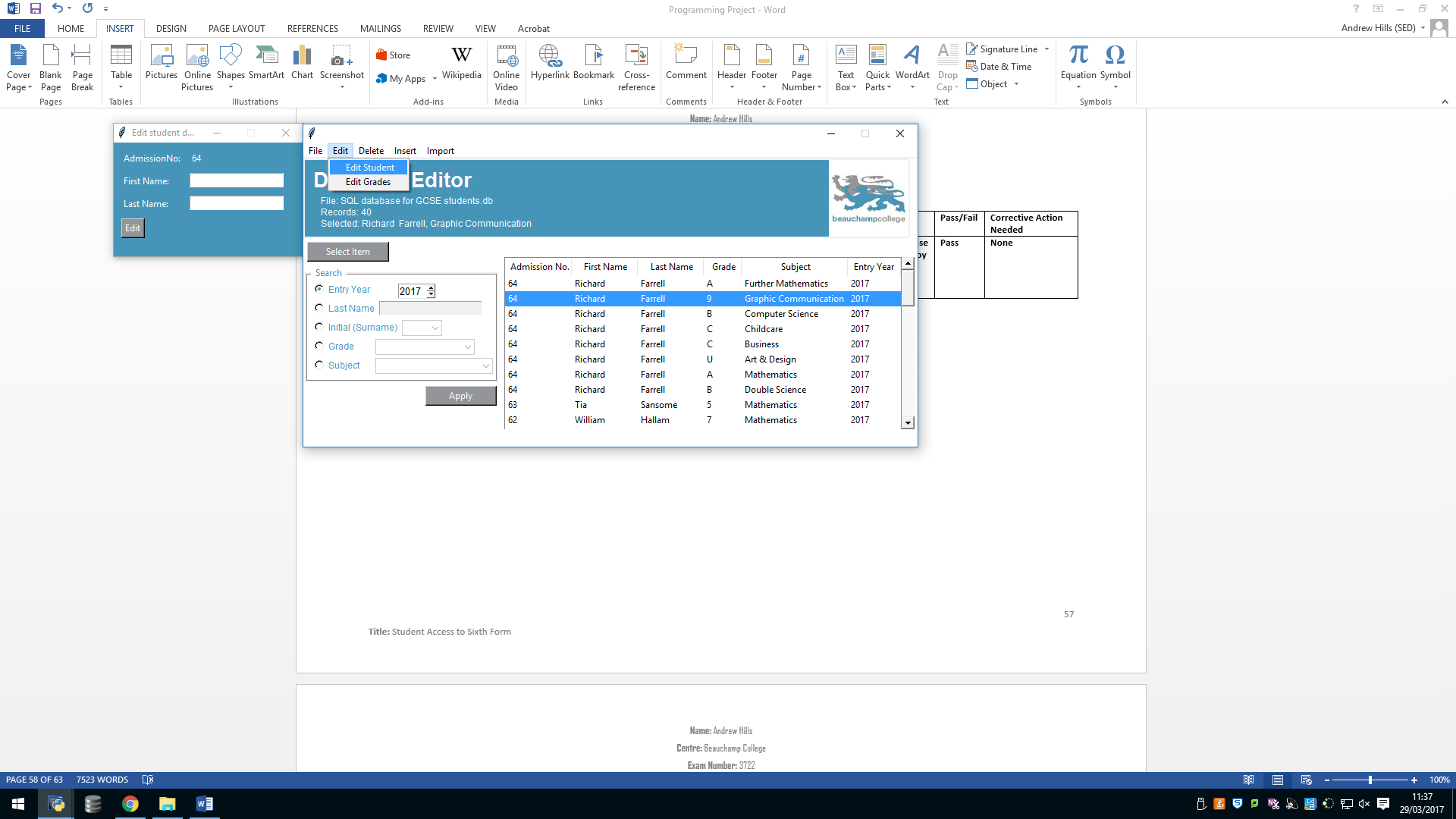
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **2** | **To test to make sure that the “File”, “Load all students” works as intended.** | **If a student is stored in the database then when this option is selected by the user then they appear in the table in the main interface** | **If a student is stored in the database then when this option is selected by the user then they appear in the table in the main interface** | **Pass** | **None** |



When the user selects the “Load all Students” option then all students are loaded into the interface.

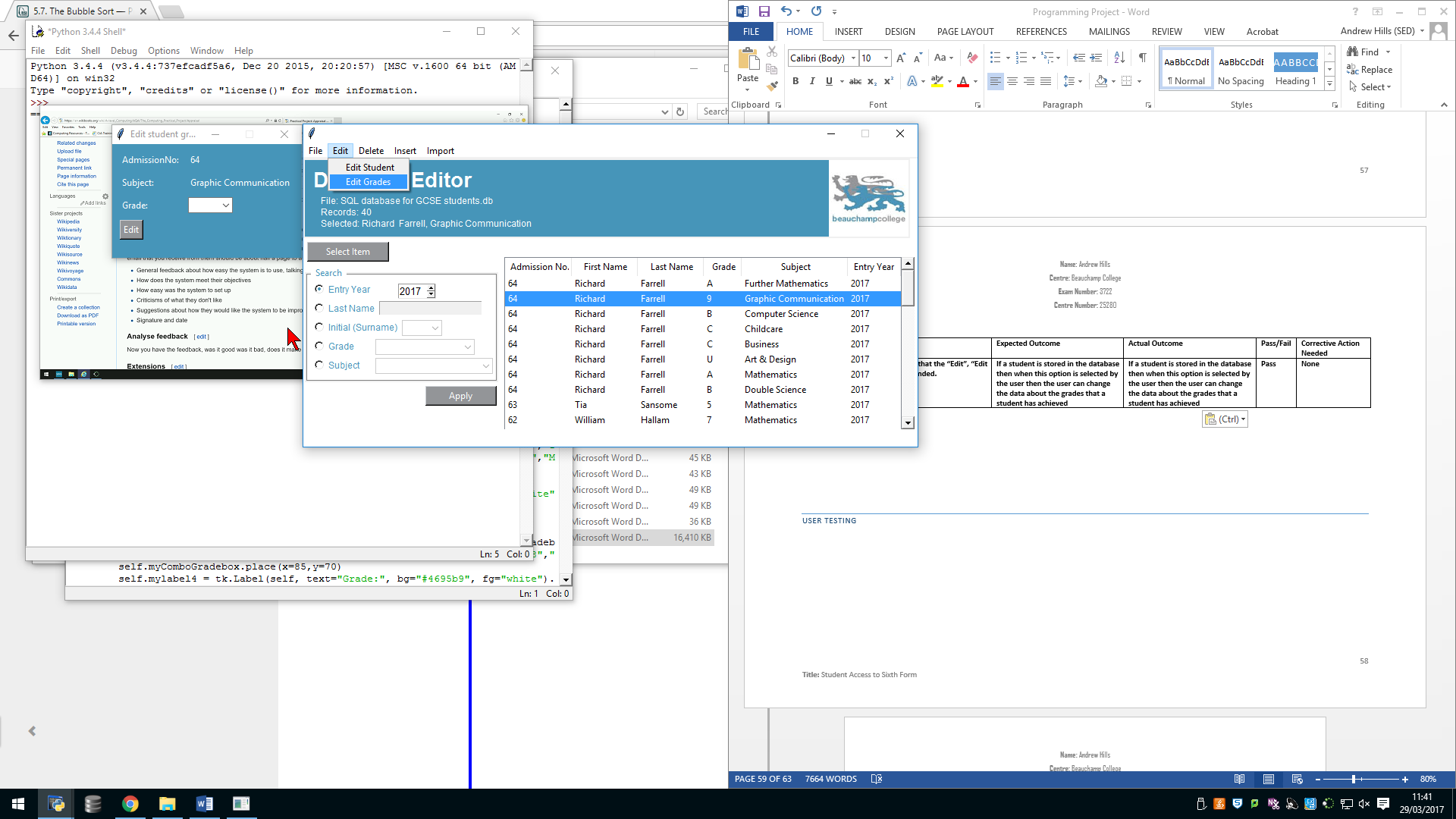
I have tested the other load features in a similar fashion and they also output the required data that the user can then manipulate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **3** | **To test to make sure that the “Edit”, “Edit student” works as intended.** | **If a student is stored in the database then when this option is selected by the user then the user can change the data about the student that they have selected** | **If a student is stored in the database then when this option is selected by the user then the user can change the data about the student that they have selected** | **Pass** | **None** |



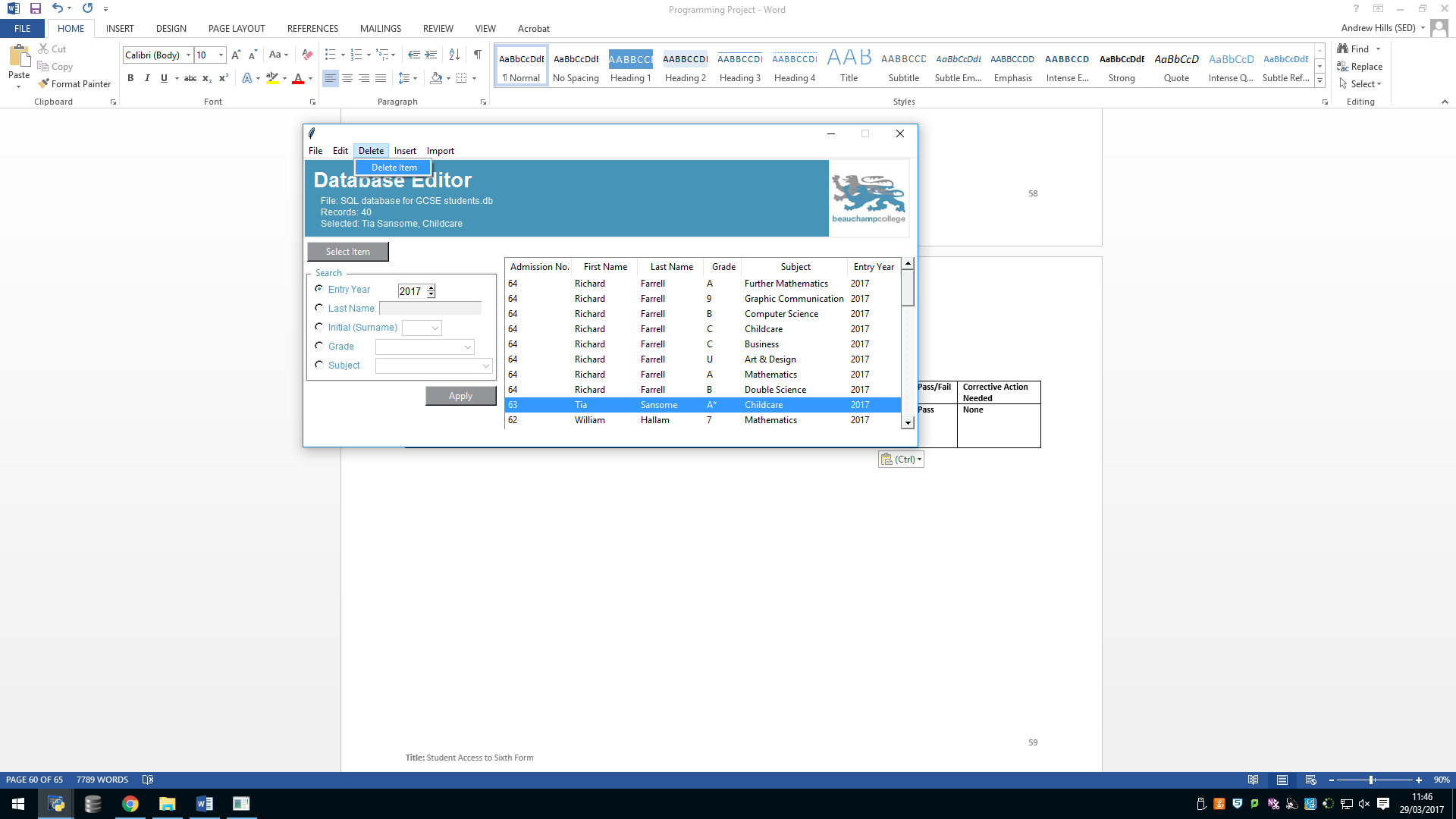
When the user selects the “Edit Student” option and have selected a student from the main table then they can then proceed to edit the name of that student. However, if the user hasn’t selected a student the program blocks them from editing anything until they select a student.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **4** | **To test to make sure that the “Edit”, “Edit Grade” works as intended.** | **If a student is stored in the database then when this option is selected by the user then the user can change the data about the grades that a student has achieved** | **If a student is stored in the database then when this option is selected by the user then the user can change the data about the grades that a student has achieved** | **Pass** | **None** |



When the user selects the “Edit Student” option and have selected a student from the main table then they can then proceed to edit the grade that the student achieved in that subject. However, if the user hasn’t selected a student the program blocks them from editing anything until they select an item from the main interface

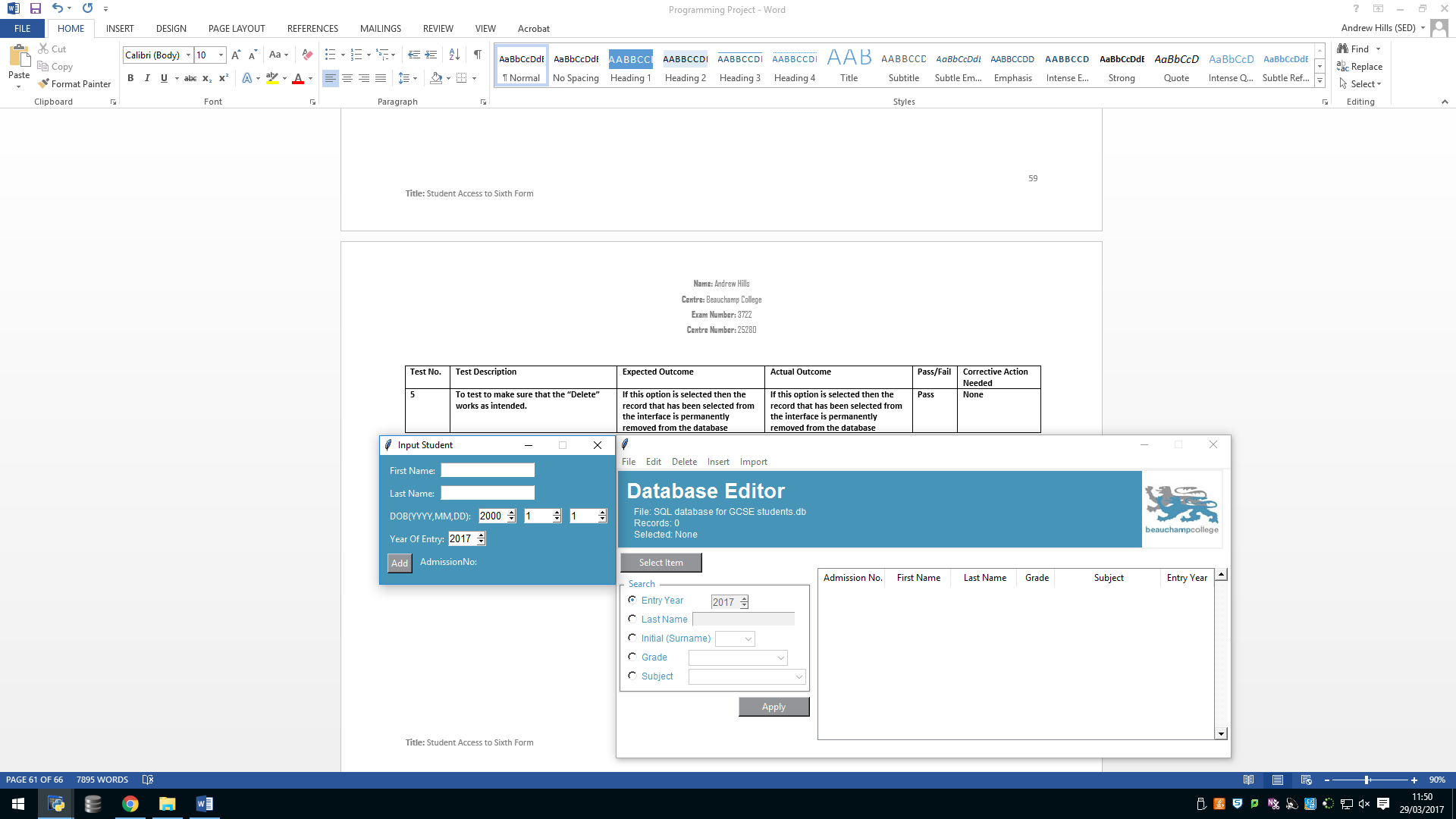
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **5** | **To test to make sure that the “Delete” works as intended.** | **If this option is selected then the record that has been selected from the interface is permanently removed from the database** | **If this option is selected then the record that has been selected from the interface is permanently removed from the database** | **Pass** | **None** |



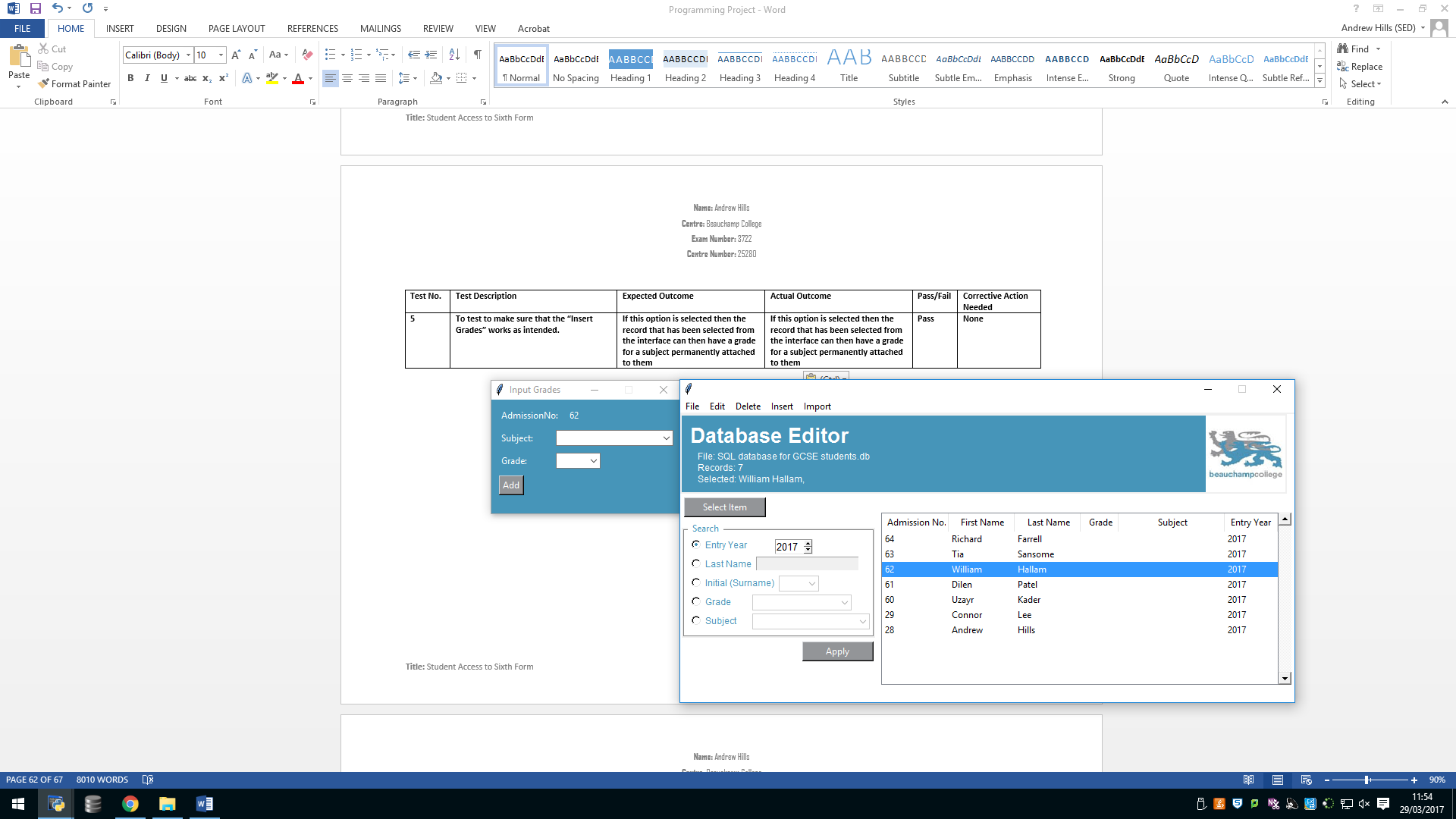
When the user selects the “Delete” option then the selected option (in the screenshot Tia Sansome) will have their subject and grade permanently deleted from the database. The information about the student, however, remains stored in the database

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **5** | **To test to make sure that the “Insert Student” works as intended.** | **If this option is selected then the user can create a new student record** | **If this option is selected then the user can create a new student record** | **Pass** | **None** |

When the user selects the “Insert student” option then a new window is created that allows the user to create a new record for a student. Then when the user clicks add then the student is added to the database to then be edited by the user.

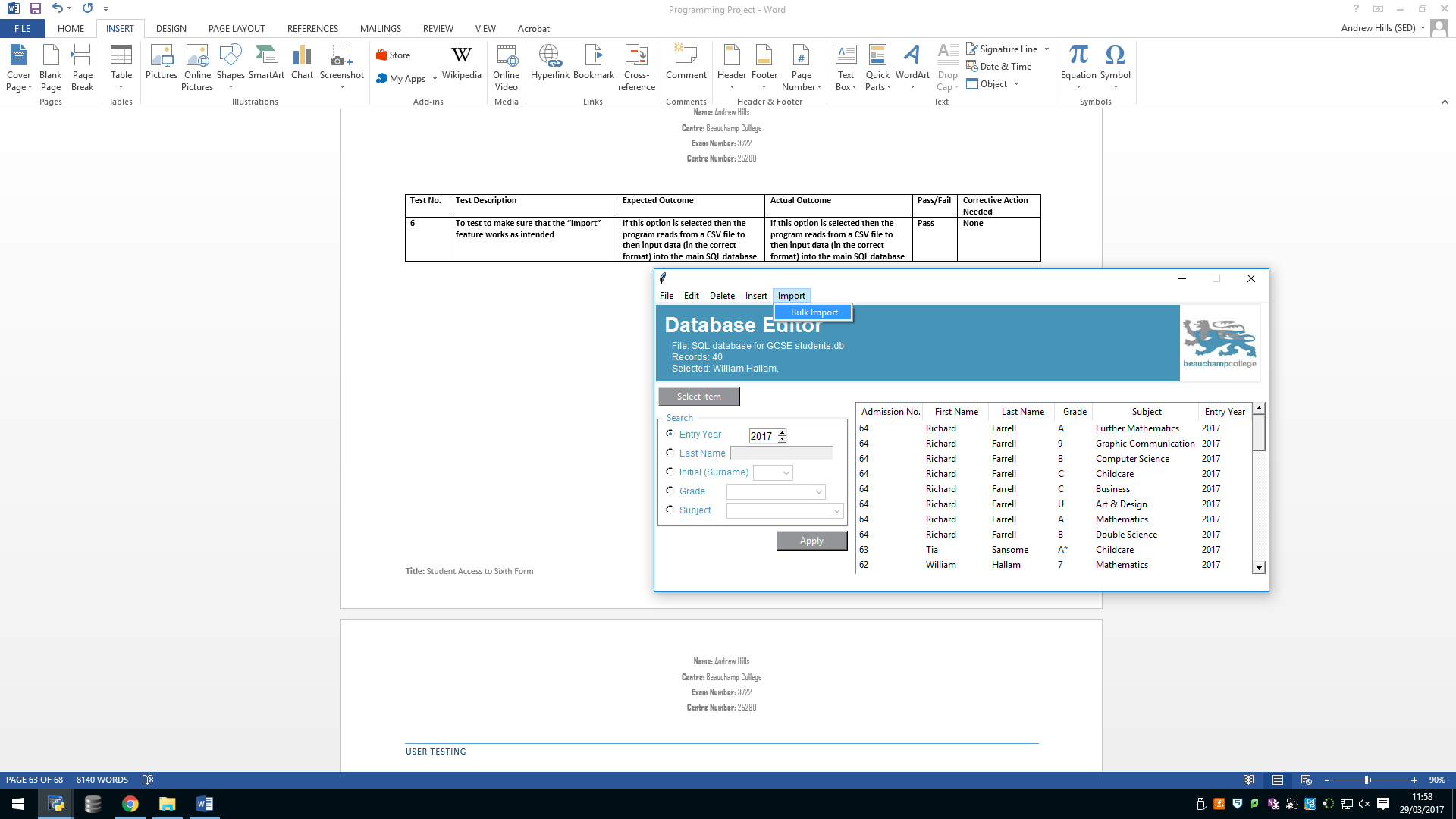


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **5** | **To test to make sure that the “Insert Grades” works as intended.** | **If this option is selected then the record that has been selected from the interface can then have a grade for a subject permanently attached to them** | **If this option is selected then the record that has been selected from the interface can then have a grade for a subject permanently attached to them** | **Pass** | **None** |



When the user selects the “Insert Grades” while a student record is selected then the program will allow them to add a subject and the achieved grade to the database. However, if no student is selected then the program will prevent the user from inputting until a student has been chosen.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Description** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** | **Corrective Action Needed** |
| **6** | **To test to make sure that the “Import” feature works as intended** | **If this option is selected then the program reads from a CSV file to then input data (in the correct format) into the main SQL database** | **If this option is selected then the program reads from a CSV file to then input data (in the correct format) into the main SQL database** | **Pass** | **None** |



When the user selects the “Bulk Import” the program will read from a CSV file to the change the format of the stored data and then add that data to the database for the user to edit how they wish.

### User Testing

When my client was given my solution to “break apart” they couldn’t find any errors in the interface or the main program which shows that my own personal testing (see above) has eliminated all errors encountered so that it runs as smoothly as possible for my client and any future users. This however, does not mean all bugs/ errors have been eliminated. Much more rigorous testing would be required into the solution by my client in their day-to-day usage

# Evaluation

1. **Create a Standard Query Language (SQL) database that holds the component details**
   1. **Create three tables that will hold the necessary details**
      1. **Make reference to Entity Relationship Diagram below**
      2. **Create a relationship between the necessary tables that can then be used when reading from the database.**

Objective 1 was completed when I created the final table that would store any data that the user would need at some point during the run-time of the program.

1. **Create a link in the Python code that can connect with the SQL database**
   1. **Allow the program to exploit table relationships to allow user to view the required data**

Objective 2 was completed by creating a connection when the program is run so that the user can exploit the database as much as they want. See **Figure 1**. All pieces of the program that require the use of the connection that is created can use it effectively to create the desired effect in the program.

1. **Create an interface in Python/Tkinter for the user to use effectively**

Objective 3 was completed when I finished the Graphical User Interface (GUI) that the user could then use to their advantage to read from the database and then manipulate the data. See **Figure 2**

1. **Read any user inputs that are given**
   1. **Take the user’s search “Entry Year”**
      1. **Accepts valid digits e.g. “2016>”**
      2. **Rejects invalid digits e.g. “<2016”**
      3. **Rejects any numbers that aren’t whole**
      4. **Rejects negative inputs**
      5. **Informs user if input is invalid**
   2. **Take the user’s search “Last Name”**
      1. **Must be a string of characters**
      2. **Rejects numbers**
      3. **Informs user if input is invalid**
      4. **Input is case sensitive**
   3. **Take the user’s search “Initial”**
      1. **Accepts all letters e.g. “A”**
      2. **Rejects any number**
      3. **Rejects erroneous data**
   4. **Take the user’s search “Grade”**
      1. **Accepts valid inputs**
         1. **“A\* - F” and “U” and “9 – 0”**
      2. **Rejects invalid inputs**
         1. **“G – T” and “10>”**
   5. **Take the user’s search “Subject”**
      1. **Accepts valid inputs**
         1. **“Mathematics”**
      2. **Rejects invalid inputs**
         1. **“Maths”**

Objective 4 was met effectively because the user can efficiently search through the data table to view any specific grade. The “entry requirement” works as well, showing all students who got a C or above in any subject. However, after user feedback I could improve my solution so that only the students who achieved 6 C’s or above or 5 and above are shown. This would mean any student who didn’t achieve 6 passes wouldn’t be shown in the output.

1. **Create drop down menus that the user can exploit**
   1. **Create menu “File”**
      1. **Create options “Load Students With Grades”, “Load all Students”, “Close”, “About”**
      2. **Create program that responds to a user click on desired option**
   2. **Create menu “Edit”**
      1. **Create options “Edit Student”, “ Edit Grades”**
      2. **Create program that responds to a user click on desired option**
   3. **Create menu “Delete”**
      1. **Create options “Delete Item”**
         1. **Deletes any selected item from the database and the interface**
      2. **Create program that responds to a user click on desired option**
   4. **Create menu “Insert”**
      1. **Create options “Insert Student details”, “Insert Student Grades”**
      2. **Create program that responds to a user click on desired option**
   5. **Create menu “Import”**
      1. **Create program that responds to a user click on desired option**
      2. **Reads information from CSV file and imports it into the database with the correct format**

Objective 5 was met after I completed all of the drop-down menus at the top of the GUI (See **Figure 2** for reference). The user can now use all menus effectively to input, edit and delete data from the database. I added a delete function to the program so that if the user cannot edit a specific piece of information then they have the option to delete it from the database and then rewrite it correctly.

1. **Construct outputs determined by the input**
   1. **Take the user search input from 4.** 
      1. **Search through the table list using the correct parameters**
         1. **If result found then go to point 6.**
         2. **If result not found then display error message for user**

Objective 6 was met when the program could handle all inputs from the user. Whether they be invalid or valid and then construct an output based on those inputs. If the input was invalid then the user is notified and then the input can be corrected. However, if the input is valid then the program will construct a relevant output from all data stored in the database.

1. **Display the outputs**
   1. **Display any results in the table for the user to view**

Objective 7 was met when the GUI could output any data type to its relevant position to then be exploited by the user. If the input is invalid then the user is notified accordingly (**Figure 3 and Figure 4).** Where figure 3 shows the output created if the user hasn’t selected a student but then tries to progress into the input grades and **Figure 4** shows what the program does if the user tries to search through the table data with an incorrect search criteria.

|  |  |
| --- | --- |
| **Figure 1** |  |
| **Figure 2** |  |
| **Figure 3** |  |
| **Figure 4** |  |

## User Feedback

Signed:

## User Feedback Analysis

The client feedback that I received was mostly positive. The only negative feedback that I received was when the program did not do everything that the client had in mind. For example, when I created the entry requirement search function any student who had a required grade would be displayed. What was wanted by the user, however, was for the program to only list students if they had 6 or more grades that met the entry requirements. This feature, therefore, can be added to a list of possible extensions for my solution that I will work on afterwards so that when my client does use the program then they can have all features they require. My client could not find any faults in the program when I gave it to them to test for themselves. This is very positive feedback as it shows the solution I have created is adequate for use by people who may not have prior knowledge of python.

## Possible Extensions

Prior to hand-in for my solution there is a list of possible extensions that I could complete. The first of which is to allow the user to search by multiple criteria at once. This would allow the user to search for a specific user and their equivalent grades simultaneously instead of one after the other. Another extension would be to allow the user to clear any search criteria that they have filled out so that they can restart the search through the list created on the interface