Section 4 - Prototype

Table of contents:

Features of the prototype

Chosen features

1. Booking:
    - 1.1. The system must allow the creation of records:
        - 1.1.1. The system must allow the following to stakeholders to create records:
            - 1.1.1.1. The customer/guest.
            - 1.1.1.2. The receptionist staff.
            - 1.1.1.3. The management staff.
    - 1.2. The system must allow the following stakeholders to read the booking table:
        - 1.2.1. The customer/guest will only be allowed to read their own record.
            - 1.2.1.1. This will be sent to the customer/guest via an automatic email once the booking is made.
        - 1.2.2. The management staff will be able to read the whole table.
            - 1.2.2.1. This will be outputted as a table on the system.
        - 1.2.3. The receptionist will be able to read the whole table.
            - 1.2.3.1. This will be outputted as a table on the system.
    - 1.3. The system must allow the following stakeholders to update records:
        - 1.3.1. Customer/guests.
        - 1.3.2. The management staff.
        - 1.3.3. The receptionists.
    - 1.4. The system must allow the following stakeholders to delete records:
        - 1.4.1. Customer/guest.
        - 1.4.2. Management staff.
        - 1.4.3. Receptionists.

The reason why I have only focused on the functional requirements for the ‘Booking’ entity is because it is the main entity of my system and it allows my prototype to be as simple as possible. The reason why I have chosen the above objectives as these are the common objectives which all entities have. These objectives are relatively simple and therefore the prototype will provide a good skeleton for my system. The objective 1.2.1.1 is included as it allows me to prototype my most common key output which is an email. These objectives also include simple addition and deletion of records to the table and I will be able to use this code within my final system. The data structure for the database also has to be created for the prototype to satisfy the objectives above.

Omitted features

2. Booking:
    2.1. The system must allow the following stakeholders to read the booking table:
        2.1.1. The customer/guest will only be allowed to read their own record.
            2.1.1.1. If an offer code that is not valid an email should be sent, automatically, to the customer.
        2.1.2. The management staff will be able to read the whole table.
            2.1.2.1. The management staff will be able to view the bookings as a tape chart. This will be outputted by the system on- screen.
            2.1.2.2. Management staff will be able to receive an output of number of customers staying at the hotel per month. This will be produced as a pie chart. This graph will be available to be outputted as a hard copy and on the system.
        2.1.3. The receptionist will be able to read the whole table.
            2.1.3.1. The receptionist will be able to view the bookings as a tape chart. This will be outputted by the system on- screen.
    2.2. The system must allow the following stakeholders to update records:
        2.2.1. The updated record will must be checked if it clashes with any other dates.
    2.3. The system must allow the following stakeholders to delete records:
        2.3.1. Customer/guest. They must only be allowed to delete their own records.

I have omitted the objectives relating to all other entities apart from the ‘Booking’ entity as I am attempting to make my prototype as simple as possible. The reason why I have chosen this entity is because it is the main entity of my system.

The above objectives are the functional objectives of the ‘Booking’ entity which I am not including in my prototype. The reason why I am not including the objectives 2.1.2.1, 2.1.2.2 and 2.1.3.1 is due to the fact that it requires the creation of graphs and therefore is to complex for the prototype as I am already creating automatic emails. I am omitting 2.1.1.1 because I will need to create a way to validate and generate the offer codes which will be too complex to do in my prototype. The objective 2.2.1 will not be included in my prototype as this would make my prototype more complex.

Prototype Screenshots

Creating the data structure:

create_database.py - E:\NAE\create_database.py (3.7.4)

File Edit Format Run Options Window Help

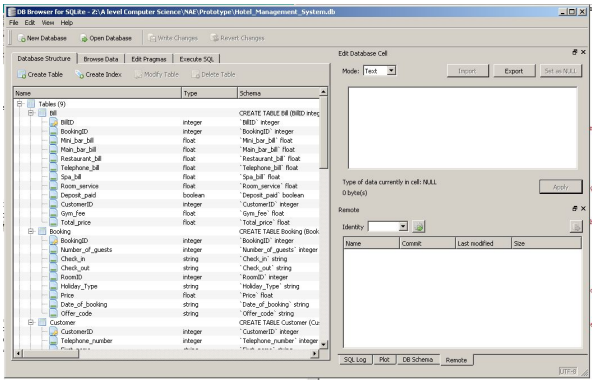
```
import sqlite3

def create_table(db_name,table_name,sql):

    with sqlite3.connect(db_name) as db:
        cursor = db.cursor()
        cursor.execute("select name from sqlite_master where name=?", (table_name,))
        cursor.execute(sql)
        db.commit()

def create_customer_table():
    sql = """create table Customer
             (CustomerID integer,
             Telephone_number integer,
             First_name string,
             Surname string,
             BookingID integer,
             BillID integer,
             Address string,
             Postcode string,
             Payment_Type string,
             Card_Number integer,
             Expiry_Date string,
             CVC_code integer,
             City string,
             Age integer,
             Allergies string,
             Email string,
             Username string,
             Password string,
             primary key(CustomerID),
             foreign key(BookingID) references Booking(BookingID),
             foreign key(BillID) references Bill(BillID))"""
    create_table(db_name, "Customer" ,sql)
```

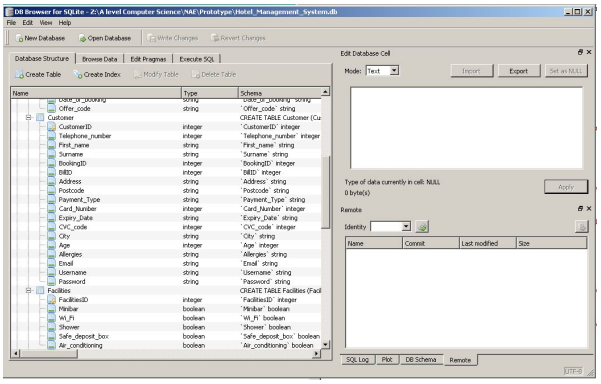
```
def create_room_type_table():
    sql = """create table Room_Type
             (Room_TypeID integer,
             Type_of_suite string,
             Balcony boolean,
             Pets boolean,
             Price float,
             primary key(Room_TypeID))"""
    create_table(db_name, "Room_Type" ,sql)

def create_staff_table():
    sql = """create table Staff
             (StaffID integer,
             Salary float,
             Frst_name string,
             Srn_name string,
             Email string,
             Postcode string,
             DOB string,
             contact_number integer,
             Username string,
             Password string,
             Job string,
             Access_level integer,
             primary key(StaffID))"""
    create_table(db_name, "Staff" ,sql)

def create_facilities_table():
    sql = """create table Facilities
             (FacilitiesID integer,
             Minibar boolean,
             Wi_Fi boolean,
             Shower boolean,
             Safe_deposit_box boolean,
             Air_conditioning boolean,
             primary key(FacilitiesID))"""
    create_table(db_name, "Facilities" ,sql)
```

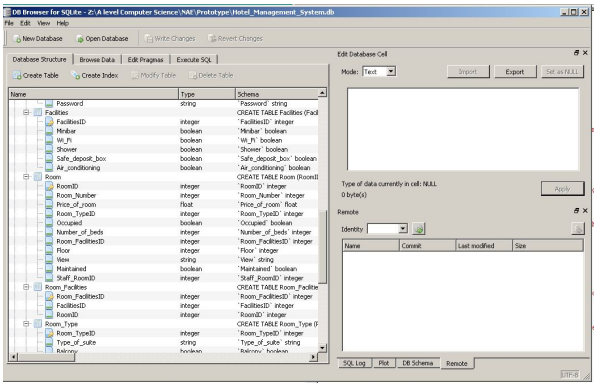
```
def create_staff_room_table():
    sql = """create table Staff_Room
             (Staff_RoomID integer,
             StaffID integer,
             RoomID integer,
             primary key(Staff_RoomID),
             foreign key(StaffID) references Staff(StaffID),
             foreign key(RoomID) references Room(RoomID))"""
    create_table(db_name, "Staff_Room" ,sql)

def create_room_facilities_table():
    sql = """create table Room_Facilities
             (Room_FacilitiesID integer,
             FacilitiesID integer,
             RoomID integer,
             primary key(Room_FacilitiesID),
             foreign key(FacilitiesID) references Facilities(FacilitiesID),
             foreign key(RoomID) references Room(RoomID))"""
    create_table(db_name, "Room_Facilities" ,sql)

def create_room_table():
    sql = """create table Room
             (RoomID integer,
             Room_Number integer,
             Price_of_room float,
             Room_TypeID integer,
             Occupied boolean,
             Number_of_beds integer,
             Room_FacilitiesID integer,
             Floor integer,
             View string,
             Maintained boolean,
             Staff_RoomID integer,
             primary key(RoomID),
             foreign key(Room_TypeID) references Room_Type(Room_TypeID),
             foreign key(Staff_RoomID) references Staff_Room(Staff_RoomID),
             foreign key(Room_FacilitiesID) references Room_Facilities(Room_FacilitiesID))"""
    create_table(db_name, "Room" ,sql)
```

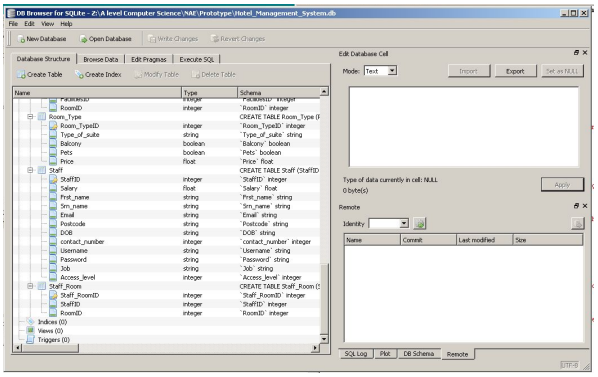
```
def create_bill_table():
    sql = """create table Bill
            (BillID integer,
            BookingID integer,
            Mini_bar_bill float,
            Main_bar_bill float,
            Restaurant_bill float,
            Telephone_bill float,
            Spa_bill float,
            Room_service float,
            Deposit_paid boolean,
            CustomerID integer,
            Gym_fee float,
            Total_price float,
            primary key(BillID),
            foreign key(CustomerID) references Customer(CustomerID),
            foreign key(BookingID) references Booking(BookingID))"""
    create_table(db_name, "Bill" ,sql)

def create_booking_table():
    sql = """create table Booking
            (BookingID integer,
            Number_of_guests integer,
            Check_in string,
            Check_out string,
            RoomID integer,
            Holiday_Type string,
            Price float,
            Date_of_booking string,
            Offer_code string,
            primary key(BookingID),
            foreign key(RoomID) references Room(RoomID))"""
    create_table(db_name, "Booking" ,sql)
```

```
if __name__=="__main__":
    db_name="Hotel_Management_System.db"
    create_customer_table()
    create_room_type_table()
    create_room_facilities_table()
    create_staff_table()
    create_facilities_table()
    create_staff_room_table()
    create_room_table()
    create_booking_table()
    create_bill_table()
```





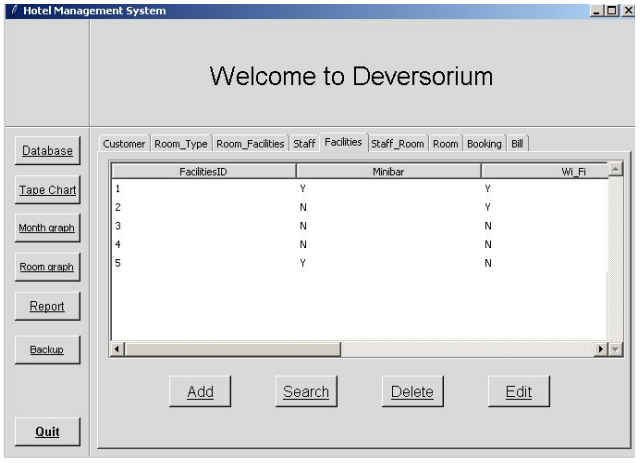




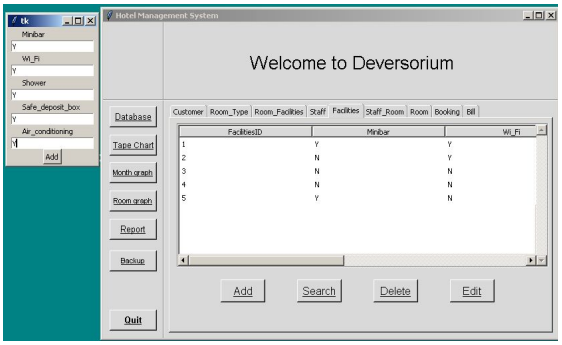
Adding Records:

Management (Adding to the Facilities Table):
Step 1:
The user will select the ‘Database’ button.

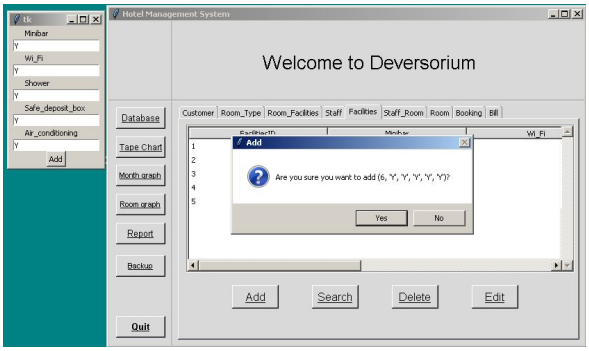
Step 2:
The user will select the ‘Add’ button.



Step 3:
The user will type in the data they would like to add in the ‘pop-up’ window and select the ‘Add’ button in this window.

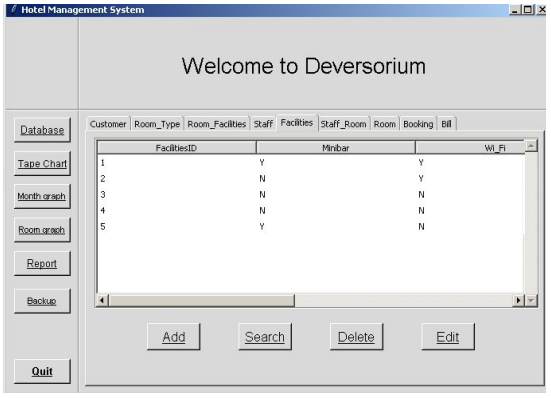


Step 4:
The user will proofread the data to be entered and will hit yes.

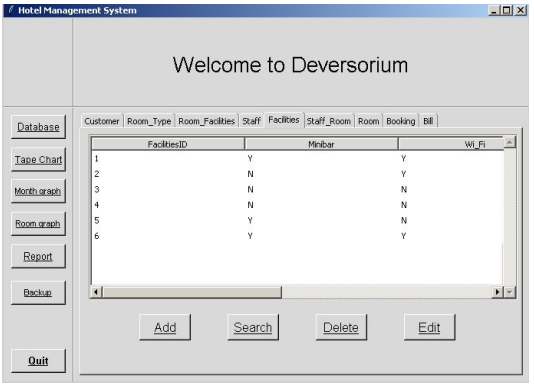


Below is shows the before and after screenshots of my system:

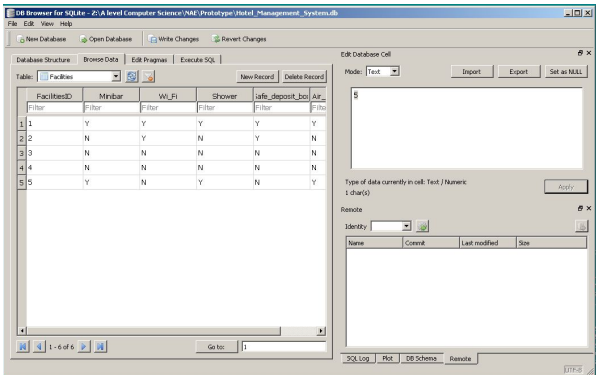
Before:



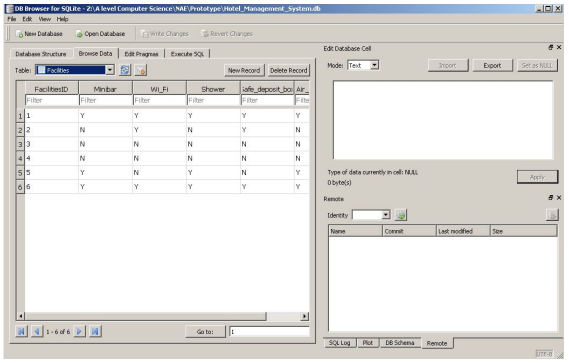
After:



Below is the before and after screenshots of the database shown in the db browser software:
Before:



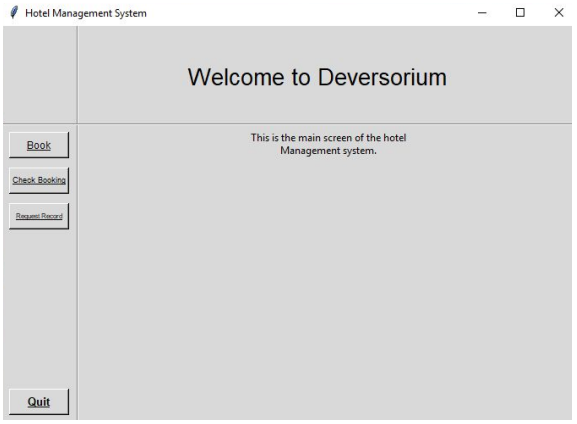
After:



Customer (booking a holiday):

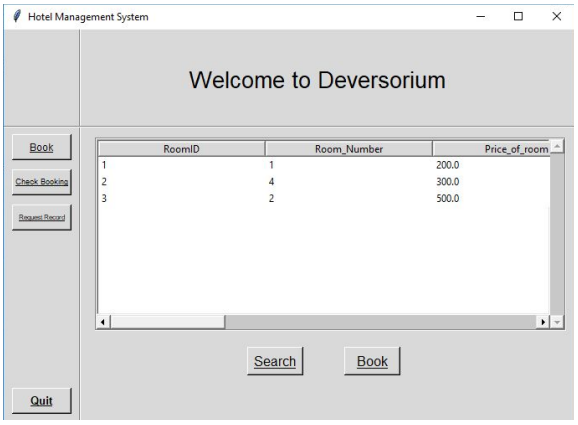
Step 1:

The user will select the ‘Book’ button:



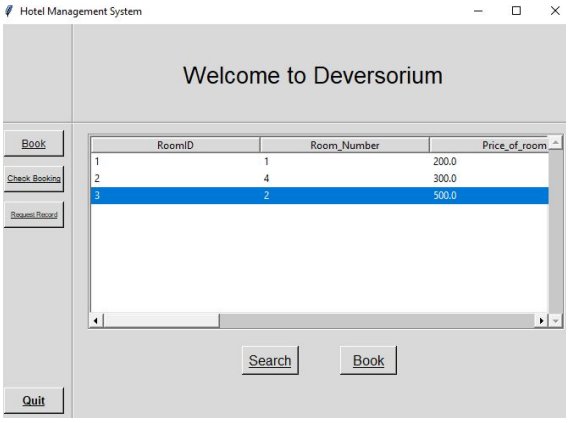
Step 2:

The user will select a room from the table shown.

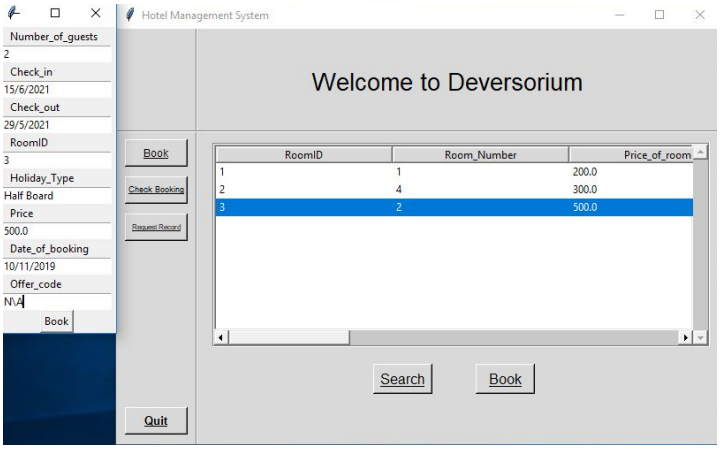


Step 3:

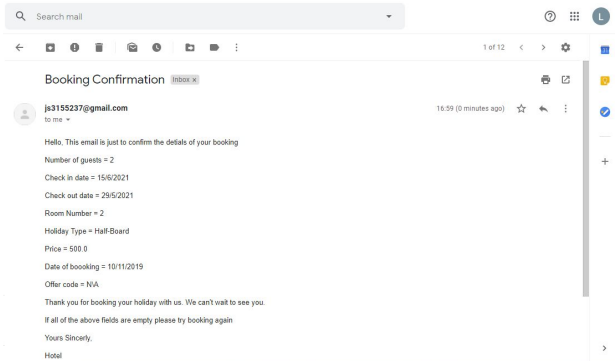
The user will select the ‘book’ button below the table.



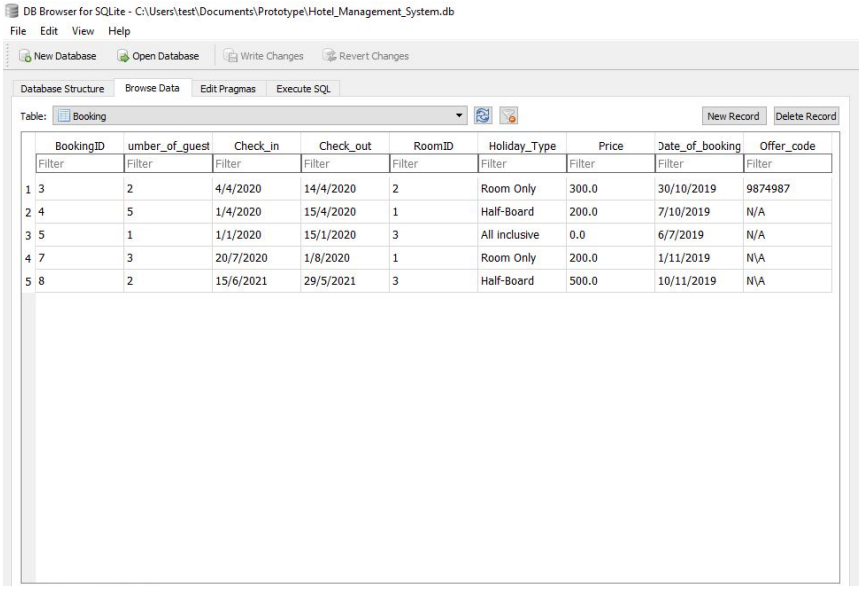
Step 4:
The user will enter the data into the ‘pop-up’ window and then select the ‘Book’ button in the ‘pop-up’ window. The ‘RoomID’ entry widget will already be filled depending on the room selected.



A confirmation email will be sent to the customer:



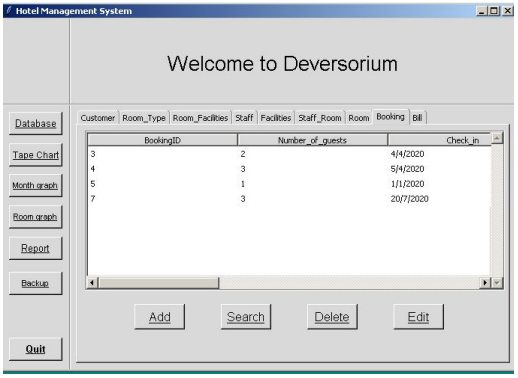
Below is a screenshot of db browser displaying the database showing that the record has been added:



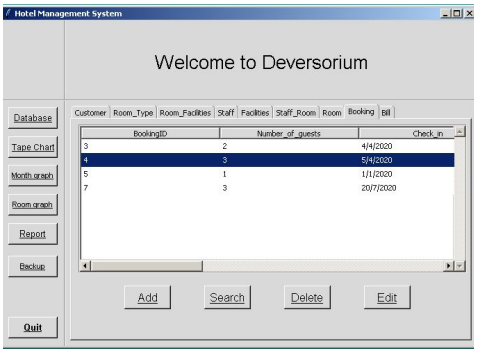
Editing records (Management)

Step 1:
The user will select the ‘Database’ button:

Step 2:
The user will then select a record from the table:

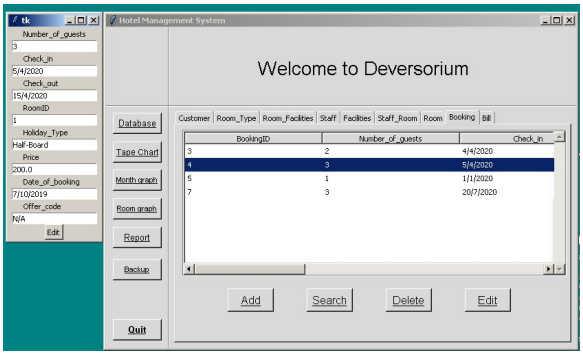


Step 4:
The user will then select the ‘Edit’ button:



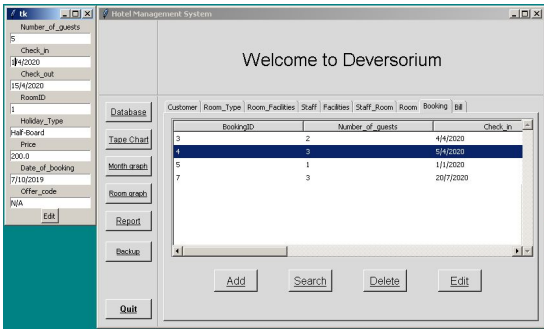
Step 5:

The entry widgets will be filled with the data from the record that was selected:



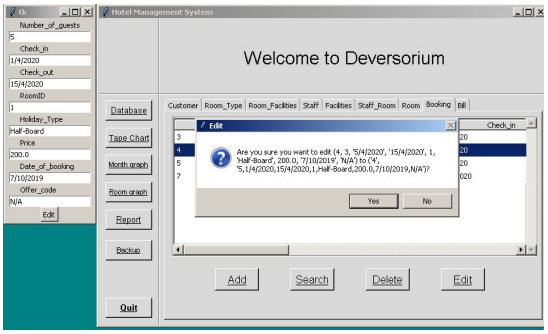
Step 6:

The user will change the record as they wish and select the ‘Edit’ button in the ‘pop-up’ window:



Step 7:

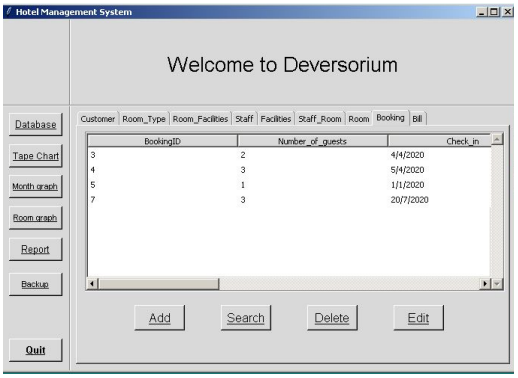
The user will proofread the new edited record and select the ‘yes’ button:



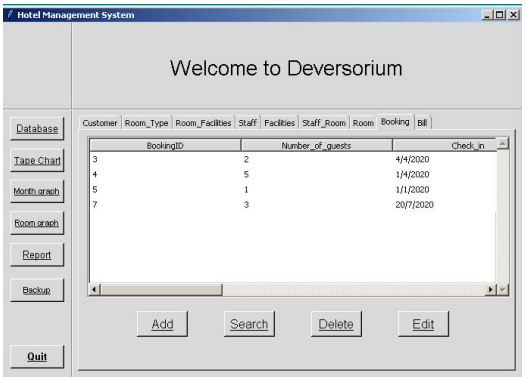
Step 8:

Below is before and after screenshots of my system:

Before:

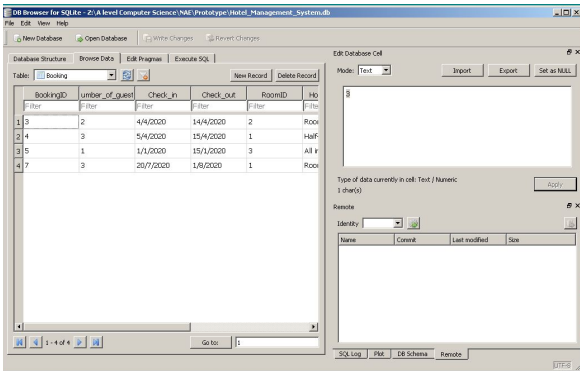


After:

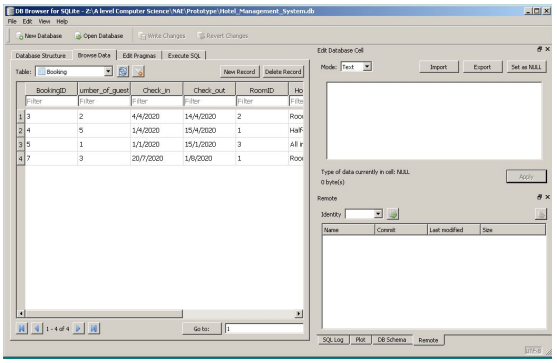


Below is before and after screenshots of the db browser displaying the database:

Before:



After:



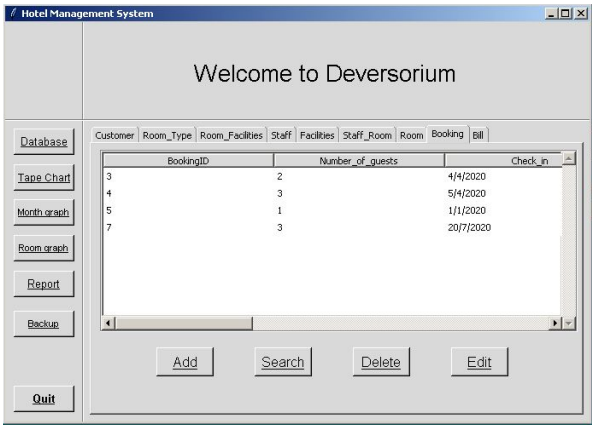
Deleting Records (Management in booking table)

Step 1:

The user will select the ‘Database’ button:

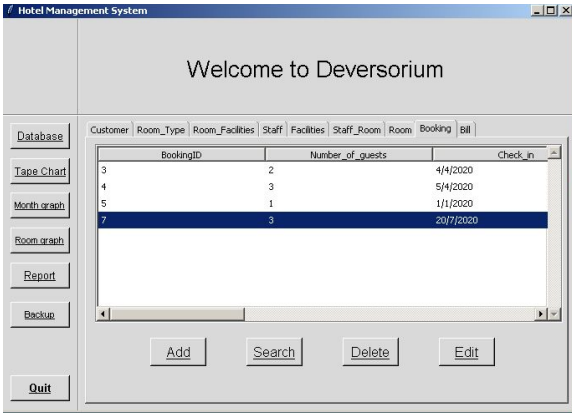
Step 2:

The user will select a record from the table:



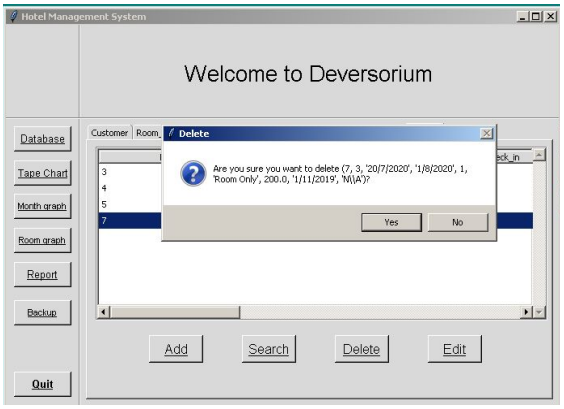
Step 3

They will select the ‘Delete’ button.



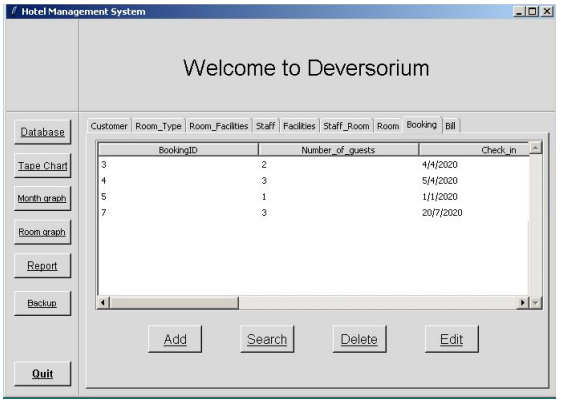
Step 4:

The user will proofread the record to be deleted and select ‘Yes’:

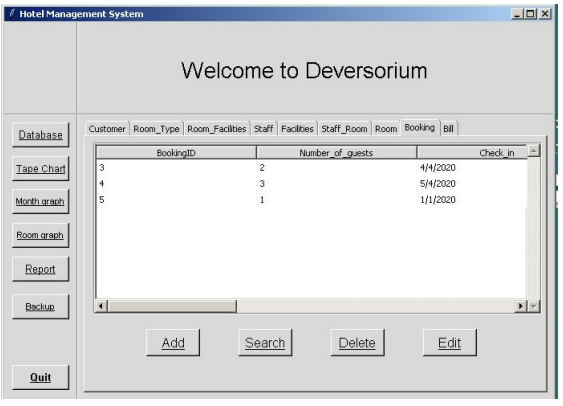


Below is the before and after screenshots of my system:

Before:

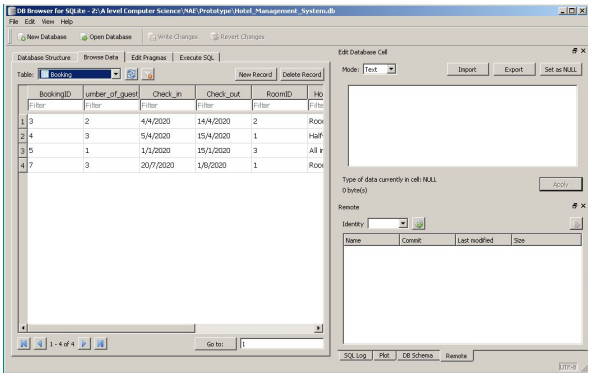


After:

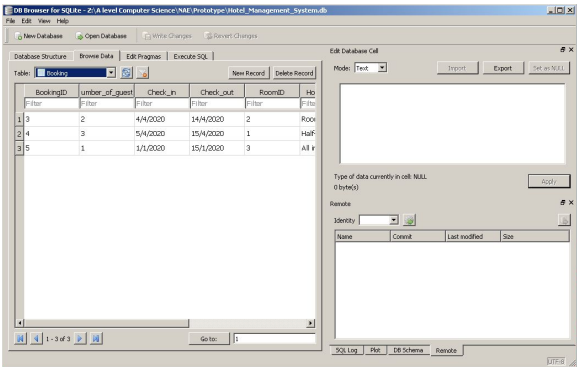


Below is the before and after screenshots of the database in db browser:

Before:



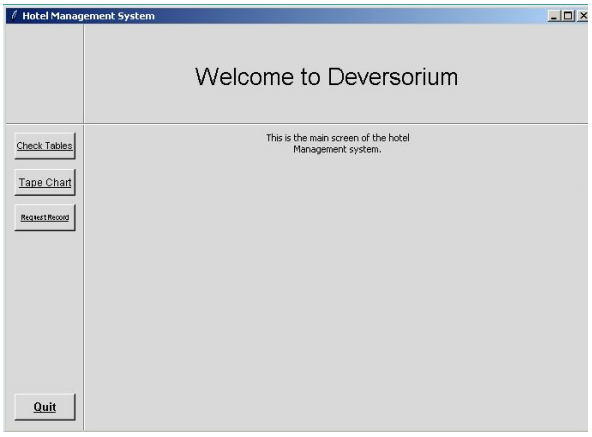
After:



Searching (Receptionist)

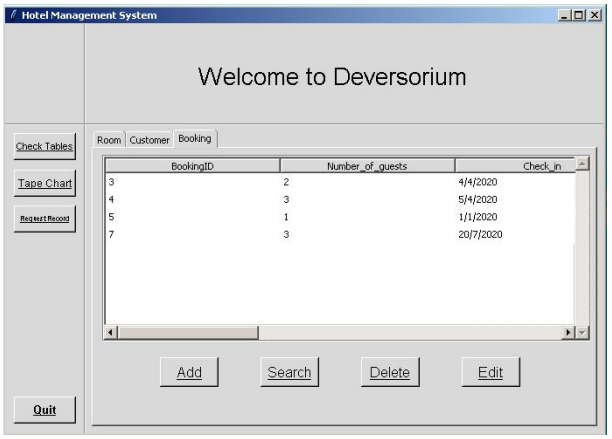
Step 1:

The user will select the ‘Check Tables’ button:

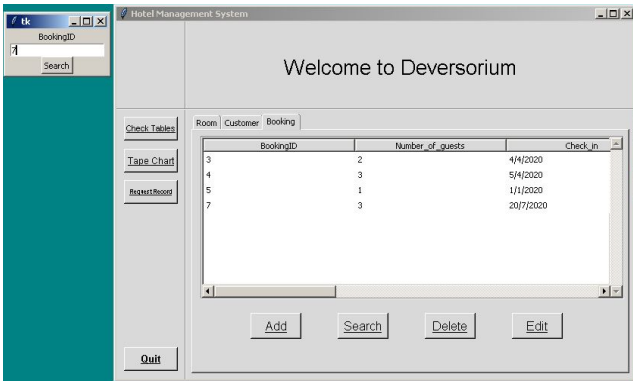


Step 2:

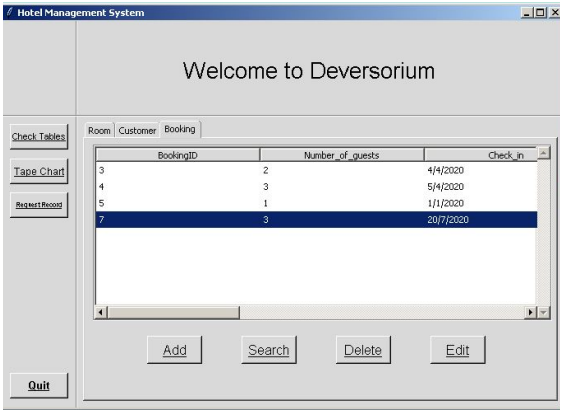
The user will then select the ‘Search’ button:



Step 3:
The user will enter the primary key of the record which they would like to find and then select the ‘search’ button:



Step 4:
The record which was searched will be highlighted in the table.



Code for prototype

Main Script:

Main Script.py - Z:\A level Computer Science\NAE\Prototype\Main Script.py (3.7.0)

File Edit Format Run Options Window Help

```
import sys

import tkinter as tk

from tkinter import messagebox

import tkinter.ttk as ttk

from read_only_class import *

def clear_frame_or_window(window): # window is either a frame or tk.Tk().
    """This clears the window."""
    for widget in window.winfo_children(): # itearates through a list of all the tk widgets in the window.
        widget.destroy()

def retrn_table_names(db_name): # Needs to be a string. db_name needs to be the directory of the database including the file extension '.db'. This will only work if it is a db.
    """This will return all the table names of a database as a list."""
    lst_of_tables=[]
    with sqlite3.connect(db_name) as db:
        cursor = db.cursor()
        cursor.execute("select name from sqlite_master")
        lst_of_tables_in_tuple=cursor.fetchall()
        for table_name in lst_of_tables_in_tuple: # iterates through the tuple to only get the names of the tables and nothing else.
            lst_of_tables.append(table_name[0])
        return lst_of_tables # returns the names of the tables in a list.
```

```
class MainMenu:
    def __init__(self, top=None):
        '''This class poipulates the template of all the other classes.'''

        _bgcolor = '#d9d9d9'  # X11 color: 'gray85'
        _fgcolor = '#000000'  # X11 color: 'black'
        _compcolor = '#d9d9d9' # X11 color: 'gray85'
        _analcolor = '#d9d9d9' # X11 color: 'gray85'
        _ana2color = '#ececec' # Closest X11 color: 'gray92'
        font14 = "-family Arial -size 10 -weight bold -slant roman "  \
            "-underline 1 -overstrike 0"
        self.style = ttk.Style()
        if sys.platform == "win32":
            self.style.theme_use('winnative')
        self.style.configure('.',background=_bgcolor)
        self.style.configure('.',foreground=_fgcolor)
        self.style.configure('.',font="TkDefaultFont")
        self.style.map('.',background=
            [('selected', _compcolor), ('active',_ana2color)])

        top.geometry("684x450+283+165")
        top.title("Hotel Management System")
        top.configure(background="#d9d9d9")
        top.configure(highlightbackground="#d9d9d9")
        top.configure(highlightcolor="black")

        self.TSeparator1 = ttk.Separator(top)
        self.TSeparator1.place(relx=0.132, rely=-0.044, relheight=1.711)
        self.TSeparator1.configure(orient="vertical")

        self.TSeparator2 = ttk.Separator(top)
        self.TSeparator2.place(relx=-0.058, rely=0.244, relwidth=2.047)

        self.ImageTFrame1 = ttk.Frame(top)
        self.ImageTFrame1.place(relx=-0.029, rely=-0.044, relheight=0.278
                , relwidth=0.154)
        self.ImageTFrame1.configure(borderwidth="2")

        self.menubar = tk.Menu(top,font="TkMenuFont",bg=_bgcolor,fg=_fgcolor)
        top.configure(menu = self.menubar)
```

```
        self.Quit_Button = tk.Button(top)
        self.Quit_Button.place(relx=0.015, rely=0.911, height=31, width=71)
        self.Quit_Button.configure(activebackground="#ececec")
        self.Quit_Button.configure(activeforeground="#000000")
        self.Quit_Button.configure(background="#d9d9d9")
        self.Quit_Button.configure(disabledforeground="#a3a3a3")
        self.Quit_Button.configure(font=font14)
        self.Quit_Button.configure(foreground="#000000")
        self.Quit_Button.configure(highlightbackground="#d9d9d9")
        self.Quit_Button.configure(highlightcolor="black")
        self.Quit_Button.configure(pady="0")
        self.Quit_Button.configure(text='''Quit''')
        self.Quit_Button.configure(command=lambda:quit_command(self))

        self.Welcome_Label = tk.Label(top)
        self.Welcome_Label.place(relx=0.146, rely=0.022, height=100, width=550)
        self.Welcome_Label.configure(activebackground="#f9f9f9")
        self.Welcome_Label.configure(activeforeground="black")
        self.Welcome_Label.configure(background="#d9d9d9")
        self.Welcome_Label.configure(disabledforeground="#a3a3a3")
        self.Welcome_Label.configure(font="-family {Arial} -size 20")
        self.Welcome_Label.configure(foreground="#000000")
        self.Welcome_Label.configure(highlightbackground="#d9d9d9")
        self.Welcome_Label.configure(highlightcolor="black")
        self.Welcome_Label.configure(text='''Welcome to Deversorium''')
        self.Welcome_Label.configure(justify='center')

        self.Main_Frame = tk.Frame(top)
        self.Main_Frame.place(relx=0.142, rely=0.254, relheight=0.734
                , relwidth=0.850)
        self.Main_Frame.configure(borderwidth="2")
        self.Main_Frame.configure(background="#d9d9d9")

        def quit_command(self):
            """This will display a quit message box and if
                the user clicks yes then it will destroy the window."""
            if messagebox.askyesno("Quit","Are you sure you want to QUIT?"):
                top.destroy()

    def get_table(self,db_name,table):
        """This method will display a table with read only permissions."""
        clear_frame_or_window(self.Main_Frame)
        read_only = ReadOnlyPermission(self.Main_Frame,table)
        read_only.create_treeview(db_name,table)

    def welcome_message(self,message):
        """This will display a message in the main frame."""
        label = tk.Label(self.Main_Frame,text=message,background="#d9d9d9")
        label.pack()
```

```
class ManagementMainScreen(MainMenu):
    def __init__(self,top=None):
        """This class populates the management main screen."""

        super().__init__(top) # calling parent class.

        font11 = "-family Arial -size 8 -weight normal -slant roman "  \
            "-underline 1 -overstrike 0"
        font12 = "-family Arial -size 9 -weight normal -slant roman "  \
            "-underline 1 -overstrike 0"
        font13 = "-family Arial -size 10 -weight normal -slant roman "  \
            "-underline 1 -overstrike 0"

        ## private attribute. (destroy in all subclasses)
        self._Month_Graph_Button = tk.Button(top)
        self._Month_Graph_Button.place(relx=0.015, rely=0.444, height=31, width=71)
        self._Month_Graph_Button.configure(activebackground="#ececec")
        self._Month_Graph_Button.configure(activeforeground="#000000")
        self._Month_Graph_Button.configure(background="#d9d9d9")
        self._Month_Graph_Button.configure(disabledforeground="#a3a3a3")
        self._Month_Graph_Button.configure(font=font11)
        self._Month_Graph_Button.configure(foreground="#000000")
        self._Month_Graph_Button.configure(highlightbackground="#d9d9d9")
        self._Month_Graph_Button.configure(highlightcolor="#000000")
        self._Month_Graph_Button.configure(pady="0")
        self._Month_Graph_Button.configure(text='''Month graph''')

        ## private attribute. (destroy in all subclasses)
        self._Room_Graph_Button = tk.Button(top)
        self._Room_Graph_Button.place(relx=0.015, rely=0.533, height=31, width=71)
        self._Room_Graph_Button.configure(activebackground="#ececec")
        self._Room_Graph_Button.configure(activeforeground="#000000")
        self._Room_Graph_Button.configure(background="#d9d9d9")
        self._Room_Graph_Button.configure(disabledforeground="#a3a3a3")
        self._Room_Graph_Button.configure(font=font11)
        self._Room_Graph_Button.configure(foreground="#000000")
        self._Room_Graph_Button.configure(highlightbackground="#d9d9d9")
        self._Room_Graph_Button.configure(highlightcolor="black")
        self._Room_Graph_Button.configure(pady="0")
        self._Room_Graph_Button.configure(text='''Room graph''')


        self.Tape_Chart_Button = tk.Button(top)
        self.Tape_Chart_Button.place(relx=0.015, rely=0.356, height=31, width=71)
        self.Tape_Chart_Button.configure(activebackground="#ececec")
        self.Tape_Chart_Button.configure(activeforeground="#000000")
        self.Tape_Chart_Button.configure(background="#d9d9d9")
        self.Tape_Chart_Button.configure(disabledforeground="#a3a3a3")
        self.Tape_Chart_Button.configure(font=font12)
        self.Tape_Chart_Button.configure(foreground="#000000")
        self.Tape_Chart_Button.configure(highlightbackground="#d9d9d9")
        self.Tape_Chart_Button.configure(highlightcolor="black")
        self.Tape_Chart_Button.configure(pady="0")
        self.Tape_Chart_Button.configure(text='''Tape Chart''')

        ## private attribute. (destroy in all subclasses)
        self._Database_Button = tk.Button(top)
        self._Database_Button.place(relx=0.015, rely=0.267, height=31, width=71)
        self._Database_Button.configure(activebackground="#ececec")
        self._Database_Button.configure(activeforeground="#000000")
        self._Database_Button.configure(background="#d9d9d9")
        self._Database_Button.configure(disabledforeground="#a3a3a3")
        self._Database_Button.configure(font=font13)
        self._Database_Button.configure(foreground="#000000")
        self._Database_Button.configure(highlightbackground="#d9d9d9")
        self._Database_Button.configure(highlightcolor="black")
        self._Database_Button.configure(pady="0")
        self._Database_Button.configure(text='''Database''')
        self._Database_Button.configure(command=lambda:self.get_database("Hotel_Management_System.db",retrn_table_names("Hotel_Management_System.db")))

        ## private attribute. (destroy in all subclasses)
        self._Report_Button = tk.Button(top)
        self._Report_Button.place(relx=0.015, rely=0.622, height=31, width=71)
        self._Report_Button.configure(activebackground="#ececec")
        self._Report_Button.configure(activeforeground="#000000")
        self._Report_Button.configure(background="#d9d9d9")
        self._Report_Button.configure(disabledforeground="#a3a3a3")
        self._Report_Button.configure(font=font13)
        self._Report_Button.configure(foreground="#000000")
        self._Report_Button.configure(highlightbackground="#d9d9d9")
        self._Report_Button.configure(highlightcolor="black")
        self._Report_Button.configure(pady="0")
        self._Report_Button.configure(text='''Report''')


        ## private attribute. (destroy in all subclasses)
        self._Backup_Button = tk.Button(top)
        self._Backup_Button.place(relx=0.015, rely=0.722, height=31, width=71)
        self._Backup_Button.configure(activebackground="#ececec")
        self._Backup_Button.configure(activeforeground="#000000")
        self._Backup_Button.configure(background="#d9d9d9")
        self._Backup_Button.configure(disabledforeground="#a3a3a3")
        self._Backup_Button.configure(font=font11)
        self._Backup_Button.configure(foreground="#000000")
        self._Backup_Button.configure(highlightbackground="#d9d9d9")
        self._Backup_Button.configure(highlightcolor="#000000")
        self._Backup_Button.configure(pady="0")
        self._Backup_Button.configure(text='''Backup''')

    def get_database(self,db_name,tables):
        """This will create a notebook with each tab being a table.
           Each table has been given read and write permission."""
        clear_frame_or_window(self.Main_Frame)
        nb = ttk.Notebook(self.Main_Frame)
        ## Below is a for loop that will create a tab for each element in tables.
        for table in tables:
            frame=tk.Frame(nb,background="#d9d9d9")
            read_write = ReadAndWritePermission(frame,nb)
            read_write.create_treeview(db_name,table)
            nb.add(frame,text=table)
        nb.pack(expand=1,fill="both")


class CustomerMainScreen(MainMenu):
    def __init__(self, top=None):
        """This populates the customer's main screen"""

        super().__init__(top) # calling parent class.

        font14 = "-family Arial -size 10 -weight bold -slant roman "  \
                 "-underline 1 -overstrike 0"
        font15 = "-family Arial -size 7 -weight normal -slant roman "  \
                "-underline 1 -overstrike 0"
        font16 = "-family Arial -size 6 -weight normal -slant roman "  \
                 "-underline 1 -overstrike 0"

        ## private attribute. (destroy in all subclasses)
        self._Check_Booking_Button = tk.Button(top)
        self._Check_Booking_Button.place(relx=0.015, rely=0.356, height=31, width=71)
        self._Check_Booking_Button.configure(activebackground="#ececec")
        self._Check_Booking_Button.configure(activeforeground="#000000")
        self._Check_Booking_Button.configure(background="#d9d9d9")
        self._Check_Booking_Button.configure(disabledforeground="#a3a3a3")
        self._Check_Booking_Button.configure(font=font15)
        self._Check_Booking_Button.configure(foreground="#000000")
        self._Check_Booking_Button.configure(highlightbackground="#d9d9d9")
        self._Check_Booking_Button.configure(highlightcolor="black")
        self._Check_Booking_Button.configure(pady="0")
        self._Check_Booking_Button.configure(text='''Check Booking''')

        ## private attribute. (destroy in all subclasses)
        self._Book_Button = tk.Button(top)
        self._Book_Button.place(relx=0.015, rely=0.267, height=31, width=71)
        self._Book_Button.configure(activebackground="#ececec")
        self._Book_Button.configure(activeforeground="#000000")
        self._Book_Button.configure(background="#d9d9d9")
        self._Book_Button.configure(disabledforeground="#a3a3a3")
        self._Book_Button.configure(font="-family {Arial} -size 9 -underline 1")
        self._Book_Button.configure(foreground="#000000")
        self._Book_Button.configure(highlightbackground="#d9d9d9")
        self._Book_Button.configure(highlightcolor="black")
        self._Book_Button.configure(pady="0")
        self._Book_Button.configure(text='''Book''')
        self._Book_Button.configure(command=lambda:self.book("Hotel_Management_System.db","Room"))
```

```
        self.Request_Record_Button = tk.Button(top)
        self.Request_Record_Button.place(relx=0.015, rely=0.444, height=31, width=71)
        self.Request_Record_Button.configure(activebackground="#ececec")
        self.Request_Record_Button.configure(activeforeground="#000000")
        self.Request_Record_Button.configure(background="#d9d9d9")
        self.Request_Record_Button.configure(disabledforeground="#a3a3a3")
        self.Request_Record_Button.configure(font=font16)
        self.Request_Record_Button.configure(foreground="#000000")
        self.Request_Record_Button.configure(highlightbackground="#d9d9d9")
        self.Request_Record_Button.configure(highlightcolor="black")
        self.Request_Record_Button.configure(pady="0")
        self.Request_Record_Button.configure(text='''Request Record''')

    def book(self,db_name,table):
        """This will only display the room table with a button
           that allows the user to book a holiday."""
        clear_frame_or_window(self.Main_Frame)
        book_hol = BookHoliday(self.Main_Frame,table)
        book_hol.create_treeview(db_name,table)
```

```
class CleanerMainScreen(CustomerMainScreen):
    def __init__(self, top=None):
        """This populates the cleaner's main screen."""

        super().__init__(top) # calling parent class.

        ## destroying all private
        self._Check_Booking_Button.destroy()
        self._Book_Button.destroy()

        self.Room_Table_Button = tk.Button(top)
        self.Room_Table_Button.place(relx=0.015, rely=0.267, height=31, width=71)
        self.Room_Table_Button.configure(activebackground="#ececec")
        self.Room_Table_Button.configure(activeforeground="#000000")
        self.Room_Table_Button.configure(background="#d9d9d9")
        self.Room_Table_Button.configure(disabledforeground="#a3a3a3")
        self.Room_Table_Button.configure(font="-family {Arial} -size 9 -underline 1")
        self.Room_Table_Button.configure(foreground="#000000")
        self.Room_Table_Button.configure(highlightbackground="#d9d9d9")
        self.Room_Table_Button.configure(highlightcolor="black")
        self.Room_Table_Button.configure(pady="0")
        self.Room_Table_Button.configure(text='''Room Table''')
        self.Room_Table_Button.configure(command=lambda:self.get_table("Hotel_Management_System.db","Room"))
```

```
class AccountantsMainScreen(CustomerMainScreen):
    def __init__(self, top=None):
        """This populates the accountant's main screen."""

        super().__init__(top) # calling parent class.

        ## destroying all private
        self._Check_Booking_Button.destroy()
        self._Book_Button.destroy()

        font13 = "-family Arial -size 10 -weight normal -slant roman "  \
                 "-underline 1 -overstrike 0"
        font14 = "-family Arial -size 10 -weight bold -slant roman "  \
                 "-underline 1 -overstrike 0"
        font15 = "-family Arial -size 7 -weight normal -slant roman "  \
                 "-underline 1 -overstrike 0"

        self.Bill_Table_Button = tk.Button(top)
        self.Bill_Table_Button.place(relx=0.015, rely=0.267, height=31, width=71)
        self.Bill_Table_Button.configure(activebackground="#ececec")
        self.Bill_Table_Button.configure(activeforeground="#000000")
        self.Bill_Table_Button.configure(background="#d9d9d9")
        self.Bill_Table_Button.configure(disabledforeground="#a3a3a3")
        self.Bill_Table_Button.configure(font="-family {Arial} -size 9 -underline 1")
        self.Bill_Table_Button.configure(foreground="#000000")
        self.Bill_Table_Button.configure(highlightbackground="#d9d9d9")
        self.Bill_Table_Button.configure(highlightcolor="black")
        self.Bill_Table_Button.configure(pady="0")
        self.Bill_Table_Button.configure(text='''Bill Table''')
        self.Bill_Table_Button.configure(command=lambda:self.get_table("Hotel_Management_System.db","Bill"))

        self.Salary_Button = tk.Button(top)
        self.Salary_Button.place(relx=0.015, rely=0.444, height=31, width=71)
        self.Salary_Button.configure(activebackground="#ececec")
        self.Salary_Button.configure(activeforeground="#000000")
        self.Salary_Button.configure(background="#d9d9d9")
        self.Salary_Button.configure(disabledforeground="#a3a3a3")
        self.Salary_Button.configure(font=font13)
        self.Salary_Button.configure(foreground="#000000")
        self.Salary_Button.configure(highlightbackground="#d9d9d9")
        self.Salary_Button.configure(highlightcolor="black")
        self.Salary_Button.configure(pady="0")
        self.Salary_Button.configure(text='''Salary''')
```

```
class ReceptionistMainScreen(CustomerMainScreen,ManagementMainScreen):
    def __init__(self, top=None):
        """This populates Receptionist's main screen"""

        super().__init__(top) # calling parent class.

        ## destroying all private
        self._Check_Booking_Button.destroy()
        self._Book_Button.destroy()
        self._Month_Graph_Button.destroy()
        self._Room_Graph_Button.destroy()
        self._Database_Button.destroy()
        self._Report_Button.destroy()
        self._Backup_Button.destroy()

        font11 = "-family Arial -size 8 -weight normal -slant roman "  \
                 "-underline 1 -overstrike 0"

        self.Check_Tables_Button = tk.Button(top)
        self.Check_Tables_Button.place(relx=0.015, rely=0.267, height=31, width=71)
        self.Check_Tables_Button.configure(activebackground="#ececec")
        self.Check_Tables_Button.configure(activeforeground="#000000")
        self.Check_Tables_Button.configure(background="#d9d9d9")
        self.Check_Tables_Button.configure(disabledforeground="#a3a3a3")
        self.Check_Tables_Button.configure(font=font11)
        self.Check_Tables_Button.configure(foreground="#000000")
        self.Check_Tables_Button.configure(highlightbackground="#d9d9d9")
        self.Check_Tables_Button.configure(highlightcolor="black")
        self.Check_Tables_Button.configure(pady="0")
        self.Check_Tables_Button.configure(text='''Check Tables''')
        self.Check_Tables_Button.configure(command=lambda:self.get_database("Hotel_Management_System.db",["Room","Customer","Booking"]))


if __name__ == '__main__':
    root = tk.Tk()
    cust = ManagementMainScreen(root)
    message = "This is the main screen of the hotel\nManagement system."
    cust.welcome_message(message)
    root.mainloop()
```

Read_only_class:

read_only_class.py - 2:\A level Computer Science\NAE\Prototype\read_only_class.py (3.7.0)

File Edit Format Run Options Window Help

```
import sys
import tkinter as tk
import tkinter.ttk as ttk
from tkinter import messagebox

import sqlite3

from library import *
```

```
def make_booking(db_name,ents,booking_fields,room,primary_key):
    """This subroutine will get all the data entered through the entry widgets and will call the subroutine which adds the booking
       to the table. This also will 'set-up' the message of the email."""

    """db_name needs to be a string, ents needs to be an array of instances of entry widgets, booking_fields needs to be an array of the fields in the booking table,
       room needs to be the record which was seleceted and primary_key is the primary_key of the booking table."""

    tview=None # This allows 'add_item' to be called without an error.
    data = get_entry_widgets_automatic(ents) # gets the data entered in through the entry widgets.

    ## converts the following to a tring so it can be used in the 'add_item' subroutine.
    add_data = create_string(data)
    booking_fields=create_string(booking_fields)

    add_item(tview,add_data,booking_fields,db_name,"Booking",primary_key) # calls 'add_item'

    room_number=room[1] # gets the room number of the room that was selected.
    sender = "js3155237@gmail.com"
    reciever = "lukea496@gmail.com"
    subject = "Booking Confirmation"
    password_of_sender = "Python123!"

    ## This is the message of the email written in html.
    body = ("""
    <html>
        <body>
            <p>Hello, This email is just to confirm the detials of your booking</p>
            <p>Number of guests = %s</p>
            <p>Check in date = %s</p>
            <p>Check out date = %s</p>
            <p>Room Number = %s</p>
            <p>Holiday Type = %s</p>
            <p>Price = %s</p>
            <p>Date of boooking = %s</p>
            <p>Offer code = %s</p>
            <p>Thank you for booking your holiday with us. We can't wait to see you.</p>
            <p>If all of the above fields are empty please try booking again</p>
            <p>Yours Sincerly,</p>
            <p>Hotel</p>
        </body>
    </html>
    """ % (data[0],data[1],data[2],str(room_number),data[4],data[5],data[6],data[7]))# formats the string to enter the data of the booking.

    email(sender,reciever,body,subject,password_of_sender) # calls 'email' which sends the email with the relevant information.
```

```
def insert_record_button(tview,ents,fields,db_name,table,primary_key_name,record,root,option):
    """This gets the data drom the entry widegts when a button is pressed."""
    data = get_entry_widgets_automatic(ents)
    data = create_string(data)
    fields=create_string(fields)
    if option==1: # When option is 1 that means the data needs to be added to the database.
        add_item(tview,data,fields,db_name,table,primary_key_name)
    if option==2: # When option is 2 a record needs to be edited into the database.
        edit_item(tview,data,table,db_name,primary_key_name,fields,record)
    root.destroy() # Destroys the 'pop-up' window.

def search_button_pressed(db_name,table,field_name_to_search,root,ent,tview):
    """Gets the data in the entry widgets and destroys the 'pop-up' window"""
    search_item=str(ent.get())
    search(db_name,table,field_name_to_search,search_item,tview)
    root.destroy()
```

```
class ReadOnlyPermission:
    def __init__(self, top=None,table=None,nb=None):
        '''This will populate a window with a search button and allows for the creation of a treeview.'''

        font9 = "-family Arial -size 12 -weight normal -slant roman "  \
            "-underline 1 -overstrike 0"

        self.Treeview_Frame = tk.Frame(top)
        self.Treeview_Frame.place(relx=0.017, rely=0.022, relheight=0.678, relwidth=0.975)
        self.Treeview_Frame.configure(relief='groove')
        self.Treeview_Frame.configure(borderwidth="2")
        self.Treeview_Frame.configure(relief="groove")
        self.Treeview_Frame.configure(background="#d9d9d9")

        self.Search_Button = tk.Button(top)
        self.Search_Button.place(relx=0.333, rely=0.756, height=34, width=67)
        self.Search_Button.configure(activebackground="#ececec")
        self.Search_Button.configure(activeforeground="#000000")
        self.Search_Button.configure(background="#d9d9d9")
        self.Search_Button.configure(disabledforeground="#a3a3a3")
        self.Search_Button.configure(font=font9)
        self.Search_Button.configure(foreground="#000000")
        self.Search_Button.configure(highlightbackground="#d9d9d9")
        self.Search_Button.configure(highlightcolor="black")
        self.Search_Button.configure(pady="0")
        self.Search_Button.configure(text='''Search''')
        self.Search_Button.configure(command=lambda: self.create_search_entry("Hotel_Management_System.db",nb,table))
```

```
    def create_treeview(self,db_name,table): # db_name needs to be a string. table needs to be a string.
        """This will create a treeview for a particular table in a database and populate it."""
        column_names = retrn_field_names(table,db_name) # gets the field names from the table.

        ## creating the tview.
        tview=ttk.Treeview(self.Treeview_Frame)
        columns = []
        for i in range(len(column_names)):
            slot = "SLOT_{0}".format(i)
            columns.append(slot) # This is used to give the columns headings.
            tview["columns"]=(columns)
            tview["show"]="headings" #This hides column 0.
            tview.column(slot)

        for i in range(len(columns)):
            tview.heading("SLOT_{0}".format(i),text=column_names[i]) # Will name each column by their respective field name.

        ## creating both the horizontal scroll bar and vertical scroll bar.
        vsb=ttk.Scrollbar(self.Treeview_Frame, orient="vertical", command=tview.yview)
        vsb.pack(side="right",fill="y")

        hsb=ttk.Scrollbar(self.Treeview_Frame, orient="horizontal", command=tview.xview)
        hsb.pack(side="bottom",fill="x")

        tview.configure(yscrollcommand=vsb.set)
        tview.configure(xscrollcommand=hsb.set)

        refresh(tview,db_name,table) # Will populate the treeview.
        tview.pack()

    def create_search_entry(self,db_name,nb,table): # db_name needs to be a string. nb can be an instance of a notebook or None. table needs to be a string or None only if nb has been passed
        """This creates a new window with an entry widget and a search button."""
        root=tk.Tk()
        if nb != None:
            table = nb.tab(nb.index("current"),"text") # This will return the table name ny getting the name of the tab. from 'https://stackoverflow.com/questions/14000944/finding-the-current
        fields = retrn_field_names(table,db_name)
        field_to_search = fields[0] # This is the field which will be used to search through. Change the number in the '[]' to select a different field.
        tview=get_tview(self.Treeview_Frame)
        field_to_search_label=tk.Label(root,text=field_to_search)
        field_to_search_label.pack()
        ent = tk.Entry(root)
        ent.pack(padx=5) # leaves a gap of '5' both left and right of the entry widget.
        search_button_new_window=tk.Button(root,text="Search",command=lambda: search_button_pressed(db_name,table,field_to_search,root,ent,tview))
        search_button_new_window.pack()
```

```
class BookHoliday(ReadOnlyPermission):
    def __init__(self, top=None,table=None):
        '''This will populate a window with everything that the 'ReadOnlyPermission' class has and and a book button.'''

        font9 = "-family Arial -size 12 -weight normal -slant roman "  \
            "-underline 1 -overstrike 0"

        super().__init__(top,None,table)

        self.Book_Hol_Button = tk.Button(top)
        self.Book_Hol_Button.place(relx=0.533, rely=0.756, height=34, width=67)
        self.Book_Hol_Button.configure(activebackground="#ececec")
        self.Book_Hol_Button.configure(activeforeground="#000000")
        self.Book_Hol_Button.configure(background="#d9d9d9")
        self.Book_Hol_Button.configure(disabledforeground="#a3a3a3")
        self.Book_Hol_Button.configure(font=font9)
        self.Book_Hol_Button.configure(foreground="#000000")
        self.Book_Hol_Button.configure(highlightbackground="#d9d9d9")
        self.Book_Hol_Button.configure(highlightcolor="black")
        self.Book_Hol_Button.configure(pady="0")
        self.Book_Hol_Button.configure(text='''Book''')
        self.Book_Hol_Button.configure(command=lambda: self.book_pressed("Hotel_Management_System.db"))

    def book_pressed(self,db_name): # db_name needs to be a string.
        """This will populate a new window with the entry widgets for the booking table."""
        root=tk.Tk()
        tview=get_tview(self.Treeview_Frame)
        room=get_recordl(tview)
        booking_fields = retrn_field_names("Booking",db_name)
        primary_key = booking_fields[0]
        booking_fields = booking_fields[1:] # deletes the first element in the list (the primary key).
        ents=create_entry(root, booking_fields) # creates the entry widgets.
        book_button=tk.Button(root,text="Book",command=lambda:make_booking(db_name,ents,booking_fields,room,primary_key))
        book_button.pack()
        try:
            insert_into_entry([ents[3]],[room[0]]) # this will insert the primary key of the room table into the respective entry widget.
        except TypeError: # if a room is not selected this error will occur.
            root.destroy()
            tk.messagebox.showwarning("Error","You have not selected a room.")
```

```
class ReadAndWritePermission(ReadOnlyPermission):
    def __init__(self, top=None,nb=None):
        '''This will populate the window with a delete,edit and add button.'''

        font9 = "-family Arial -size 12 -weight normal -slant roman "  \
            "-underline 1 -overstrike 0"

        super().__init__(top,None,nb)

        self.Edit_Button = tk.Button(top)
        self.Edit_Button.place(relx=0.733, rely=0.756, height=34, width=67)
        self.Edit_Button.configure(activebackground="#ececec")
        self.Edit_Button.configure(activeforeground="#000000")
        self.Edit_Button.configure(background="#d9d9d9")
        self.Edit_Button.configure(disabledforeground="#a3a3a3")
        self.Edit_Button.configure(font=font9)
        self.Edit_Button.configure(foreground="#000000")
        self.Edit_Button.configure(highlightbackground="#d9d9d9")
        self.Edit_Button.configure(highlightcolor="black")
        self.Edit_Button.configure(pady="0")
        self.Edit_Button.configure(text='''Edit''')
        self.Edit_Button.configure(command= lambda: self.create_entry_window("Hotel_Management_System.db",nb,2))

        self.Delete_Button = tk.Button(top)
        self.Delete_Button.place(relx=0.533, rely=0.756, height=34, width=67)
        self.Delete_Button.configure(activebackground="#ececec")
        self.Delete_Button.configure(activeforeground="#000000")
        self.Delete_Button.configure(background="#d9d9d9")
        self.Delete_Button.configure(disabledforeground="#a3a3a3")
        self.Delete_Button.configure(font=font9)
        self.Delete_Button.configure(foreground="#000000")
        self.Delete_Button.configure(highlightbackground="#d9d9d9")
        self.Delete_Button.configure(highlightcolor="black")
        self.Delete_Button.configure(pady="0")
        self.Delete_Button.configure(text='''Delete''')
        self.Delete_Button.configure(command= lambda: self.create_entry_window("Hotel_Management_System.db",nb,3))
```

```
        self.Add_Button = tk.Button(top)
        self.Add_Button.place(relx=0.133, rely=0.756, height=34, width=67)
        self.Add_Button.configure(activebackground="#ececec")
        self.Add_Button.configure(activeforeground="#000000")
        self.Add_Button.configure(background="#d9d9d9")
        self.Add_Button.configure(disabledforeground="#a3a3a3")
        self.Add_Button.configure(font=font9)
        self.Add_Button.configure(foreground="#000000")
        self.Add_Button.configure(highlightbackground="#d9d9d9")
        self.Add_Button.configure(highlightcolor="black")
        self.Add_Button.configure(pady="0")
        self.Add_Button.configure(text='''Add''')
        self.Add_Button.configure(command= lambda: self.create_entry_window("Hotel_Management_System.db",nb,1))

    def create_entry_window(self,db_name,nb,option): # db_name needs to be a string. nb needs to be an instance of a Notebook. option needs to be either a 1,2 or 3.
        """This will populate a new window with an entry widget for each field and either an add or delete button."""
        record=None

        table_name = nb.tab(nb.index("current"),"text") # Gets the name of the table by getting the title of the tab.
        fields = return_field_names(table_name,db_name)
        primary_key_name = fields[0]
        fields = fields[1:]

        tview=get_tview(self.Treeview_Frame)

        if option==1:
            root = tk.Tk()
            entry_widgets = create_entry(root,fields)
            add_button_new_window=tk.Button(root,text="Add",command=lambda:insert_record_button(tview,entry_widgets,fields,db_name,table_name,primary_key_name,record,root,1))
            add_button_new_window.pack()

        if option==2 and tview!=None: # an error will occur if tview is None and it is passed to 'edit_item' through 'inser_record_button' and therefore it cannot be None.
            root = tk.Tk()
            entry_widgets = create_entry(root,fields)
            record=get_record1(tview) # contians the primary key.
            entry_record = record[1:] # does not contian the primary key.

            insert_into_entry(entry_widgets,entry_record)
            edit_button_new_window=tk.Button(root,text="Edit",command=lambda:insert_record_button(tview,entry_widgets,fields,db_name,table_name,primary_key_name,record,root,2))
            edit_button_new_window.pack()

        if option==3:
            delete_item(tview,fields,db_name,table_name,primary_key_name)
```

Library

```
library.py - 2:\A level Computer Science\NAE\Prototype\library.py (3.7.0)
File  Edit  Format  Run  Options  Window  Help
import tkinter as tk
import tkinter.ttk as ttk
from tkinter import messagebox

import smtplib # Used to send email.
from email.mime.multipart import MIMEMultipart # Used to create the message of the email.
from email.mime.text import MIMEText # Used to create the message of the email.

import sqlite3


def get_record(tview): # tview needs to be an instance of a treeview.
    """This will return the data in a selected record.
       This works if MORE than one record can be selected. (use for delation of multiple records)."""
    focused=tview.selection() # this returns a list of the index values of the string.
    records=[]
    if focused != "": # This prevents an error if nothing has been selected.
        for i in focused:
            index_value=str(i)[1:]# This disreagrds the first character of a string.
            index_value=int(index_value,16)# This changes the hex number into decimal.
            for child in tview.get_children():
                x=str(child)[1:]
                x=int(x,16)
                if x==index_value:
                    record = tview.item(child)["values"] # gets the data stored in the specific row in the tveiw.
            records.append(record)
        return records # returns a 2d array.


def delete_item(tview,fields,db_name,table,primary_key_name):
    """This will delete a selected record."""
    ## Changes these to string values as they need to be strings.
    primary_key_name=str(primary_key_name)
    fields=str(fields)
    table=str(table)
    db_name=str(db_name)

    records=get_record(tview) # can delete multiple records.
    for record in records: # iterates through the records that need to be deleted.
        primary_key=record[0] # gets the primary_key of the record that needs to be deleted.

        with sqlite3.connect(db_name) as db:
            cursor = db.cursor()
            cursor.execute("select * from {1} where {2}={0}".format(str(primary_key),table,primary_key_name)) # gets the record to be deleted to show the user.
            record=cursor.fetchone()
            if messagebox.askyesno("Delete","Are you sure you want to delete {0}?".format(record)): # Asks the user if they are sure.
                sql="delete from {1} where {2}={0}".format(str(primary_key),table,primary_key_name)
                cursor.execute(sql)
        refresh(tview,db_name,table) # will re-populate the tview to show changes.


def edit_item(tview,data,table,db_name,primary_key_name,fields,record):
    """This will edit a selected item."""
    fields=create_lst(fields)
    sql_code_for_fields=""

    ## The for loop below will generate a '=?' for every field in the table to insert into the sql command.
    for i in range(len(fields)):
        sql_code_without_coma=fields[i]+"=?"
        sql_code_for_fields=sql_code_for_fields+sql_code_without_coma
    ## The try and excepy block below is to check when the loop is one before the last iteration. This will stop a ',' character being added to the end of the string.
        try:
            rogue_value=fields[i+1]
            sql_code_for_fields=sql_code_for_fields+","
        except: # if any error occurs it will go to the next iteration.
            next

    primary_key=record[0] # The reason it is here is because it needs to have the original index value (i.e before the record is editied).
    with sqlite3.connect(db_name) as db:
        cursor=db.cursor()
        cursor.execute("select * from {1} where {2}={0}".format(str(primary_key),table,primary_key_name)) # Gets the record before it is edited.
        record=cursor.fetchone()
        edited_record=str(primary_key),data
        if messagebox.askyesno("Edit","Are you sure you want to edit {0} to {1}?".format(record,edited_record)): # Asks the user if they are sure.
            sql="update {2} set {1} where {3}={0}".format(str(primary_key),sql_code_for_fields,table,primary_key_name) # Edits the record.
            data=create_lst(data) # Turns data (which is a string) into a list.
            cursor.execute(sql,data)
            db.commit()
    refresh(tview,db_name,table) # Will re-populate the tview to show changes.


## This works if ONLY one record can be selected. (use for editing a single record)
def get_record1(tview): # This will return the data in a selected record.
    """This subroutine will return 1 record which is selected within the tview."""
    focused=tview.focus()
    if focused != "": #T his prevents an error if nothing has been selected.
        index_value=str(focused)[1:]#T his disreagrds the first character of a string.
        index_value=int(index_value,16)#T his changes the hex number into decimal.
        for child in tview.get_children():
            x=str(child)[1:]
            x=int(x,16)
            if x==index_value:
                return tview.item(child)["values"]
    else:
        None


def insert_into_entry(ents,data): # ents and data must have the same number of elements.
    """Will insert the data into the entry widgets."""
    for i in range(len(ents)):
        ents[i].insert(0,data[i])

def get_tview(root):
    """This will return the instance of a treeview within a window."""
    for widget in root.winfo_children():
        temp_widget=str(widget)
        if "treeview" in temp_widget:
            tview=widget
            break
        else:
            tview=None
    return tview

def query_maker(fields):
    """This will create a '?' for every element within field."""
    fields=create_lst(fields)
    sql_code_for_fields=""
    for i in range(len(fields)):
        sql_code_without_coma="?"
        sql_code_for_fields=sql_code_for_fields+sql_code_without_coma
        if len(fields)-1 != i:
            sql_code_for_fields=sql_code_for_fields+","
    return sql_code_for_fields


def create_lst(data):
    """This will create a list when given a string. If data is "a,b,c,d" then it will return ["a","b","c","d"]."""
    lst_of_data=[]
    element_of_lst_of_data=""

    for i in data: #This for loop is so more than 1 field can be inserted in 1 entry box.
        if i==",":
            lst_of_data.append(element_of_lst_of_data)
            element_of_lst_of_data=""
        else:
            element_of_lst_of_data=element_of_lst_of_data+i
    lst_of_data.append(element_of_lst_of_data)
    return lst_of_data
```

```
def add_item(tview,data,fields,db_name,table,primary_key_name):
    """This will add an item to the end of the table.
       If the database is NOT linked to a tview the 'tview will be None'."""
    primary_key_name=str(primary_key_name)
    fields=str(fields)
    table=str(table)
    db_name=str(db_name)
    data=create_lst(data)

    with sqlite3.connect(db_name) as db:
        cursor=db.cursor()
        question_marks = query_maker(fields)
        sql="insert into {0} ({1}) values ({2})".format(table,fields,question_marks)
        cursor.execute(sql,data)
        db.commit()

    if tview != None: # If it is adding to a database that is not linked to a treeview, tview will be equal to none.
        # What is happening here is that the record has already been added. If the user clicks 'no' in the message box this will delete the added record.
        with sqlite3.connect(db_name) as db:
            cursor=db.cursor()
            # The code below is to get the primary key of the item to be added. The reason for this is so the record can be displayed in the message box.
            cursor.execute("select * from {0}".format(table))
            table_contents=cursor.fetchall()
            maximum=0
            for record in table_contents:
                if record[0]>maximum:
                    maximum=record[0]
            primary_key=maximum # The maximum value is the primary key of the record to be added.
            cursor.execute("select * from {2} where {1}={0}".format(str(primary_key),primary_key_name,table))
            record=cursor.fetchone()
            if not messagebox.askyesno("Add","Are you sure you want to add {0}?".format(record)):
                cursor.execute("delete from {2} where {1}={0}".format(str(primary_key),primary_key_name,table))
                db.commit()


            refresh(tview,db_name,table) # An error will occur here if there is no tview linked. This is the reason for the 'if tview != None'.
```

```
def refresh(tview,db_name,table):
    """This will first delete all the records in the treeview and then add all the records in the table to the treeview."""
    table=str(table)
    db_name=str(db_name)
    tview.delete(*tview.get_children())
    with sqlite3.connect(db_name) as db:
        cursor=db.cursor()
        cursor.execute("select * from {0}".format(table))
        records=cursor.fetchall()

    for record in records:
        tview.insert("","end",values=record)

def create_string(data):
    """Takes a list (all elements need to be a string) as input and returns a string.
       e.g. input=["a","b","c"] and output="a,b,c"""
    lst_as_string = ""
    for i in range(len(data)):
        data[i] = str(data[i])
        if i != len(data):
            lst_as_string = lst_as_string+","+data[i]
    lst_as_string = lst_as_string[1:]
    return lst_as_string

def get_entry_widgets_automatic(ents):
    """This will return a list of data that is contained in each of the entry widgets in the list of entry widgets."""
    data=[]
    for ent in ents:
        single_ent=ent.get()
        data.append(single_ent)
    return data
```

```
def create_entry(root, fields):
    """This will create an entry widget for every element in fields.
       It will also create a label with the text of the elements in fields."""
    entries = []
    for field in fields:
        row = tk.Frame(root) # all the entry widgets will be in this frame
        row.pack(padx=5)
        lab = tk.Label(row, width=15, text=field, anchor='w')
        ent = tk.Entry(row)
        lab.pack()
        ent.pack()
        entries.append(ent)
    return entries

def retrn_field_names(table,db_name):
    """Returns a list of the field names in a table."""
    lst_of_field_names=[]
    with sqlite3.connect(db_name) as db:
        cursor = db.cursor()
        cursor.execute("PRAGMA TABLE_INFO ({0})".format(table)) # This gets the field names, the data types and other stuff
        lst_of_fields_and_other_stuff=cursor.fetchall()
        lst_of_fields_and_other_stuff
        for i in lst_of_fields_and_other_stuff:
            field_name=i[1] #The reason for this is because the field names are stored in element 1 in ecah tuple in the list.
            lst_of_field_names.append(field_name)
        return lst_of_field_names
```

```
def search(db_name,table,field_name_to_search,search_item,tview):
    """Searches a table for a value.
       The record will be highlighted in the tview."""
    with sqlite3.connect(db_name) as db:
        cursor=db.cursor()
        cursor.execute("select * from {1} where {2}={0}".format(search_item,table,field_name_to_search))#This is for the dialogue box.
        record=cursor.fetchone() # gets searched record.
        cursor.execute("select * from {0}".format(table))
        records=cursor.fetchall() # gets all records form table.
        db.commit()

    if record==None:
        messagebox.showwarning("Search","This record does not exist")
    else:
        for i in records: # allows the position of the record in the tview to be found.
            if i == record:
                position_in_tview = records.index(record)

        ## The 3 lines below will highlight a searched ID and move the scrollbar to that record.
        child_id = tview.get_children()[position_in_tview] # item is the index of the search item in the records list. 'child_id' stores the postion of the searched item in the tview. The res
        tview.selection_set(child_id) # this will highlight the row.
        tview.see(child_id) # this will move the tview to that row.

def email(sender,receiver,body,subject,password_of_sender):
    """Sends an email to the receiver."""
    server = smtplib.SMTP('smtp.gmail.com', 587) # This is the server being used to send the emails. To find the name of the server simple google 'gmail smtp server' for gmail and 'outlook sm

    server.starttls() # This encrpyts the email.

    server.login("js3155237@gmail.com", password_of_sender) # Logs in to the server. The first parameter is the username and the second is the password.

    ## creating the message.
    msg = MIMEMultipart()
    msg["From"] = sender
    msg["To"] = receiver
    msg["Subject"] = subject

    msg.attach(MIMEText(body,"html")) # this will link the body variable to the IMEMultiplart() object. The 'html' is used as the body will be in html. This can be changed to 'plain' if just

    message = msg.as_string()

    server.sendmail(sender, receiver, message) # The first parameter is the sender, the second is the receiver and the thirs is the message itself.

    server.quit() # Ends the session.
```

Evaluation of prototype

One strength of my prototype relates to the objective 1.2.1.1 as I am successfully able to send an email to a customer. This is beneficial for the user (in this case the customer) as they will be able to check the details of their booking change them if necessary through the system. Although I cannot send the email to a specific customer as I have decided not to implement a login screen for the prototype and therefore I am unable to do this due to the fact that I need to know which customer made the booking. The objectives 1.2.2 and 1.2.3 have been successfully achieved as the booking table has been able to be outputted as a table to each of the respective users. The objective 1.4 has also been achieved as users can delete multiple records in a table by simply clicking on them. The user is also able to select just one record in the table if this is what is needed. A strength my system has is that when updating records the entry widgets already contain the data of the record to be updated and therefore the user only needs to change these, this relates to the objective 1.3. This assists the user as they do not need to re-type the data they do not wish to change. To select a record to update the user only needs to click on the record within the table.

This image could be the logo for the hotel. A problem that my prototype has is when adding to a table, if there are many fields within the table, the pop up window will be expand to the length of the window. This could be improved by grouping the entry widgets in groups of five and place these groups side by side. A weakness my system has is that when the whole database is shown in the management screen the linking tables are also displayed.