**DASC - 5300 : FOUNDATION OF COMPUTING DATABASE PROJECT**

**NIVAS MARUDHAVANAN - (1002197280)**

**JENNITHRA SRINIVASAN - (1002206950)**

**SPRING 2024**

**DR. SOUMAYA GHARSALLAOUI**

**MAY 07, 2024**



**DEPARTMENT OF DATASCIENCE**

**HONOR CODE**

I pledge, on my honor, to upload UT Arlington’s tradition of academic integrity, a tradition that values hard work and honest effort in the pursuit of academic excellence.

I promise that I will submit only work that I personally create or that I contribute to group collaborations, and I will appropriately reference any work from other source. I will follow the highest standards of integrity and uphold the spirit of the honor code.

**DATABASE CREATION:**

def get\_sqlite\_datatype(python\_dtype):

if python\_dtype == int:

return "INTEGER"

elif python\_dtype == float:

return "REAL"

else:

return "TEXT"

def get\_column\_datatypes(file\_path):

with open(file\_path, 'r', newline='') as file:

reader = csv.reader(file)

sample\_row = next(reader)

column\_datatypes = []

for col\_val in next(reader):

if col\_val.isdigit():

column\_datatypes.append(int)

else:

try:

float(col\_val)

column\_datatypes.append(float)

except ValueError:

column\_datatypes.append(str)

sqlite\_datatypes = [get\_sqlite\_datatype(dtype) for dtype in column\_datatypes]

return sqlite\_datatypes

def create\_table\_with\_datatypes(table\_name, column\_names, column\_datatypes):

columns = ", ".join(f"\"{name}\" {dtype}" for name, dtype in zip(column\_names, column\_datatypes))

create\_statement = f"CREATE TABLE IF NOT EXISTS {table\_name} ({columns})"

conn = sqlite3.connect(db\_name)

cursor = conn.cursor()

cursor.execute(create\_statement)

conn.commit()

conn.close()

def insert\_data(table\_name, data):

conn = sqlite3.connect(db\_name)

cursor = conn.cursor()

placeholders = ", ".join("?" for \_ in data)

insert\_statement = f"INSERT INTO {table\_name} VALUES ({placeholders})"

cursor.execute(insert\_statement, data)

conn.commit()

conn.close()

def table\_creation\_insertion():

for file\_name in file\_list:

table\_name = file\_name.split('.')[0]

column\_datatypes = get\_column\_datatypes(file\_location + file\_name)

with open(file\_location + file\_name, 'r', newline='') as file:

reader = csv.reader(file)

column\_names = next(reader)

column\_names = [name.strip('\ufeff').strip() for name, dtype in zip(column\_names, column\_datatypes)]

create\_table\_with\_datatypes(table\_name, column\_names, column\_datatypes)

for row in reader:

insert\_data(table\_name, row)

def get\_sqlite\_datatype(python\_dtype):

if python\_dtype == int:

return "INTEGER"

elif python\_dtype == float:

return "REAL"

else:

return "TEXT"

def get\_column\_datatypes(file\_path):

with open(file\_path, 'r', newline='') as file:

reader = csv.reader(file)

sample\_row = next(reader)

column\_datatypes = []

for col\_val in next(reader):

if col\_val.isdigit():

column\_datatypes.append(int)

else:

try:

float(col\_val)

column\_datatypes.append(float)

except ValueError:

column\_datatypes.append(str)

sqlite\_datatypes = [get\_sqlite\_datatype(dtype) for dtype in column\_datatypes]

return sqlite\_datatypes

def create\_table\_with\_datatypes(table\_name, column\_names, column\_datatypes):

columns = ", ".join(f"\"{name}\" {dtype}" for name, dtype in zip(column\_names, column\_datatypes))

create\_statement = f"CREATE TABLE IF NOT EXISTS {table\_name} ({columns})"

conn = sqlite3.connect(db\_name)

cursor = conn.cursor()

cursor.execute(create\_statement)

conn.commit()

conn.close()

def insert\_data(table\_name, data):

conn = sqlite3.connect(db\_name)

cursor = conn.cursor()

placeholders = ", ".join("?" for \_ in data)

insert\_statement = f"INSERT INTO {table\_name} VALUES ({placeholders})"

cursor.execute(insert\_statement, data)

conn.commit()

conn.close()

def table\_creation\_insertion():

for file\_name in file\_list:

table\_name = file\_name.split('.')[0]

column\_datatypes = get\_column\_datatypes(file\_location + file\_name)

with open(file\_location + file\_name, 'r', newline='') as file:

reader = csv.reader(file)

column\_names = next(reader)

column\_names = [name.strip('\ufeff').strip() for name, dtype in zip(column\_names, column\_datatypes)]

create\_table\_with\_datatypes(table\_name, column\_names, column\_datatypes)

for row in reader:

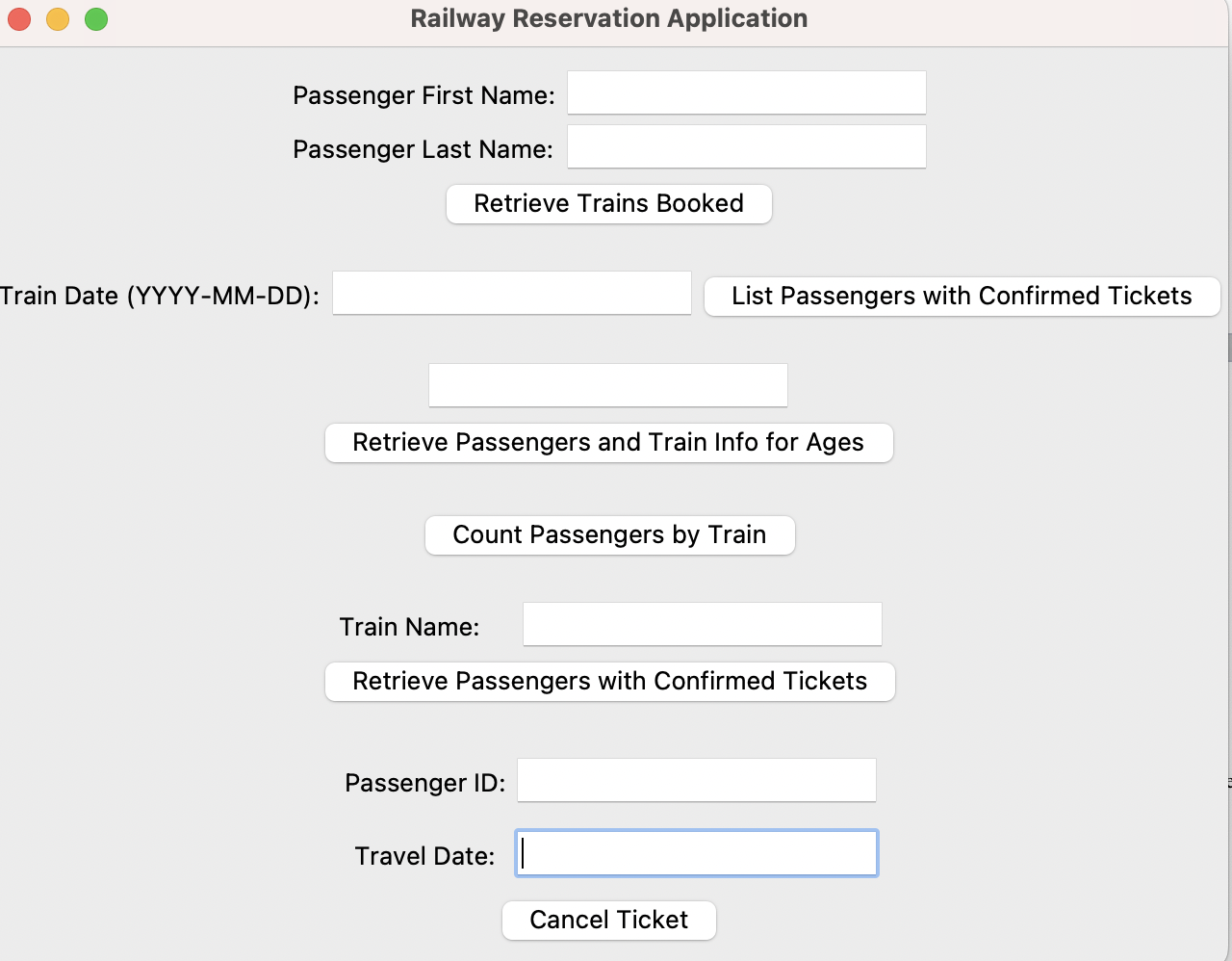
insert\_data(table\_name, row)

This Python script is designed to facilitate the transformation of CSV data into SQLite database tables. It starts by defining functions to determine the appropriate SQLite data types for the columns based on the types of data present in the CSV files. The get\_column\_datatypes() function reads a sample of data from the CSV file to infer the data types of each column, such as integers, floats, or strings. These inferred data types are then converted to SQLite-compatible types using the get\_sqlite\_datatype() function.

Once the data types are determined, the script proceeds to create SQLite tables with the correct column names and data types using the create\_table\_with\_datatypes() function. For each CSV file provided, the script iterates through the rows, inserting the data into the corresponding SQLite table with the insert\_data() function.

Overall, this script automates the process of creating SQLite database tables from CSV files, ensuring data integrity and compatibility between the two formats. It's a handy tool for managing and analyzing data stored in CSV files within an SQLite database environment.

**UI INITIALIZATION:**

****

**# Create main window**

root = tk.Tk()

root.title("Railway Reservation Application")

This code snippet creates the primary window for a Railway Reservation Application using Tkinter, a Python GUI toolkit. The tk.Tk() method call generates the main window instance, and root.title() changes the window's title to "Railway Reservation Application". This window will be the primary interface for users dealing with the reservation system.

**# Query 1:** Retrieve trains booked by passenger

frame\_query\_1 = tk.Frame(root)

frame\_query\_1.pack(pady=10)

label\_first\_name = tk.Label(frame\_query\_1, text="Passenger First Name:")

label\_first\_name.grid(row=0, column=0)

entry\_first\_name = tk.Entry(frame\_query\_1)

entry\_first\_name.grid(row=0, column=1)

label\_last\_name = tk.Label(frame\_query\_1, text="Passenger Last Name:")

label\_last\_name.grid(row=1, column=0)

entry\_last\_name = tk.Entry(frame\_query\_1)

entry\_last\_name.grid(row=1, column=1)

button\_query\_1 = tk.Button(frame\_query\_1, text="Retrieve Trains Booked", command=query\_1)

button\_query\_1.grid(row=2, column=0, columnspan=2)

This code sample creates a part within the main window interface of a Railway Reservation Application, allowing users to obtain trains reserved by a passenger. It has input sections for the passenger's first and last names, as well as a button labeled "Retrieve Trains Booked". When you click the button, it calls a function (query\_1) that queries the database and displays the results.

**# Query 2:** List passengers with confirmed tickets on a specific date

frame\_query\_2 = tk.Frame(root)

frame\_query\_2.pack(pady=10)

label\_train\_date = tk.Label(frame\_query\_2, text="Train Date (YYYY-MM-DD):")

label\_train\_date.grid(row=0, column=0)

entry\_train\_date = tk.Entry(frame\_query\_2)

entry\_train\_date.grid(row=0, column=1)

button\_query\_2 = tk.Button(frame\_query\_2, text="List Passengers with Confirmed Tickets", command=query\_2)

button\_query\_2.grid(row=0, column=2)

This portion of code adds a user interface component within the Railway Reservation Application's main window for querying passengers who have confirmed tickets for a certain date. It creates a frame (frame\_query\_2) with an entry field (entry\_train\_date) for entering the required date in the YYYY-MM-DD format. A label (label\_train\_date) specifies the input format. A button labeled "List Passengers with Confirmed Tickets" is also included, which when selected executes the query\_2 function, which is expected to handle the retrieval of passenger information depending on the specified date.

**# Query 3:** Retrieve passengers and train information for ages 50 to 60

frame\_query\_3 = tk.Frame(root)

frame\_query\_3.pack(pady=10)

button\_query\_3 = tk.Button(frame\_query\_3, text="Retrieve Passengers and Train Info for Ages 50-60", command=query\_3)

button\_query\_3.pack()

This code snippet adds a part to the Railway Reservation Application's main window interface for performing Query 3, which retrieves information about passengers aged 50 to 60, as well as their train data. It generates a frame (frame\_query\_3) to organize the UI elements, which includes a button labeled "Retrieve Passengers and Train Info for Ages 50-60". Clicking this button invokes the query\_3 function, which is expected to retrieve and show relevant passenger and train information for the chosen age range.

**# Query 4:** Count passengers by train

frame\_query\_4 = tk.Frame(root)

frame\_query\_4.pack(pady=10)

button\_query\_4 = tk.Button(frame\_query\_4, text="Count Passengers by Train", command=query\_4)

button\_query\_4.pack()

This code segment adds a part to the Railway Reservation Application's main window interface for performing Query 4, which seeks to count the number of passengers per train. It creates a frame (frame\_query\_4) to organize the interface components and has a button named "Count Passengers by Train". Clicking this button invokes the query\_4 function, which is meant to calculate and display the passenger count for each train.

**# Query 5:** Retrieve passengers with confirmed tickets in a train

frame\_query\_5 = tk.Frame(root)

frame\_query\_5.pack(pady=10)

label\_train\_name = tk.Label(frame\_query\_5, text="Train Name:")

label\_train\_name.grid(row=0, column=0)

entry\_train\_name = tk.Entry(frame\_query\_5)

entry\_train\_name.grid(row=0, column=1)

button\_query\_5 = tk.Button(frame\_query\_5, text="Retrieve Passengers with Confirmed Tickets", command=query\_5)

button\_query\_5.grid(row=1, column=0, columnspan=2)

This code creates a part in the main window interface of the Railway Reservation Application for Query 5, which involves obtaining people who have confirmed tickets for a given train. It generates a frame (frame\_query\_5) to organize the interface elements, which include an entry field (entry\_train\_name) for entering the train name and a button labeled "Retrieve Passengers with Confirmed Tickets" to invoke the query\_5 function, which retrieves and displays the relevant passenger information.

**# Query 6**: Cancel ticket

frame\_query\_6 = tk.Frame(root)

frame\_query\_6.pack(pady=10)

passenger\_id\_label = tk.Label(frame\_query\_6, text="Passenger ID:")

passenger\_id\_label.grid(row=0, column=0, pady=5)

passenger\_id\_entry = tk.Entry(frame\_query\_6)

passenger\_id\_entry.grid(row=0, column=1, pady=5)

travel\_date\_label = tk.Label(frame\_query\_6, text="Travel Date:")

travel\_date\_label.grid(row=1, column=0, pady=5)

travel\_date\_entry = tk.Entry(frame\_query\_6)

travel\_date\_entry.grid(row=1, column=1, pady=5)

cancel\_button = tk.Button(frame\_query\_6, text="Cancel Ticket", command=cancel\_ticket)

cancel\_button.grid(row=2, column=0, columnspan=2)

root.mainloop()

This code adds a user interface component to the main window of the Railway Reservation Application for Query 6, which includes canceling a ticket. It constructs a frame (frame\_query\_6) to organize the interface elements, which include entry fields (passenger\_id\_entry and travel\_date\_entry) for entering the passenger ID and travel date, respectively. To ensure clarity, labels (passenger\_id\_label and travel\_date\_label) are provided. When the "Cancel Ticket" button is pressed, it executes the cancel\_ticket function, allowing users to cancel the ticket for the selected passenger and travel date. Finally, the root.mainloop() function is invoked to initiate the event loop and allow user interaction with the application's graphical interface.

**QUERY 1:**

def query\_1():

passenger\_first\_name = entry\_first\_name.get()

passenger\_last\_name = entry\_last\_name.get()

query = "SELECT Train.\"Train Name\" FROM Train JOIN Booked ON Passenger.SSN = Booked.Passanger\_ssn JOIN Passenger ON Booked.Train\_Number = Train.\"Train Number\" WHERE Passenger.first\_name=? AND Passenger.last\_name=?"

result = execute\_query(query, (passenger\_first\_name, passenger\_last\_name))

messagebox.showinfo("Trains Booked", "The user is booked on the following trains: \n" + "\n".join(str(row[0]) for row in result))

**A screenshot of a computer

Description automatically generated**

**QUERY 2:**

def query\_2():

train\_date = entry\_train\_date.get()

query = "SELECT DISTINCT Passenger.first\_name, Passenger.last\_name\

FROM Passenger JOIN Booked ON Passenger.SSN = Booked.Passanger\_ssn \

JOIN Train ON Booked.Train\_Number = Train.\"Train Number\"\

JOIN Train\_Status ON Train.\"Train Name\" = Train\_Status.TrainName\

WHERE Train\_Status.TrainDate = ? \

AND Booked.Staus = 'Booked'"

print(query)

result = execute\_query(query, (train\_date,))

print(result)

messagebox.showinfo("Passengers with Confirmed Tickets", "Passengers with confirmed tickets on the entered date: \n" + "\n".join([f"{row[0]} {row[1]}" for row in result]))

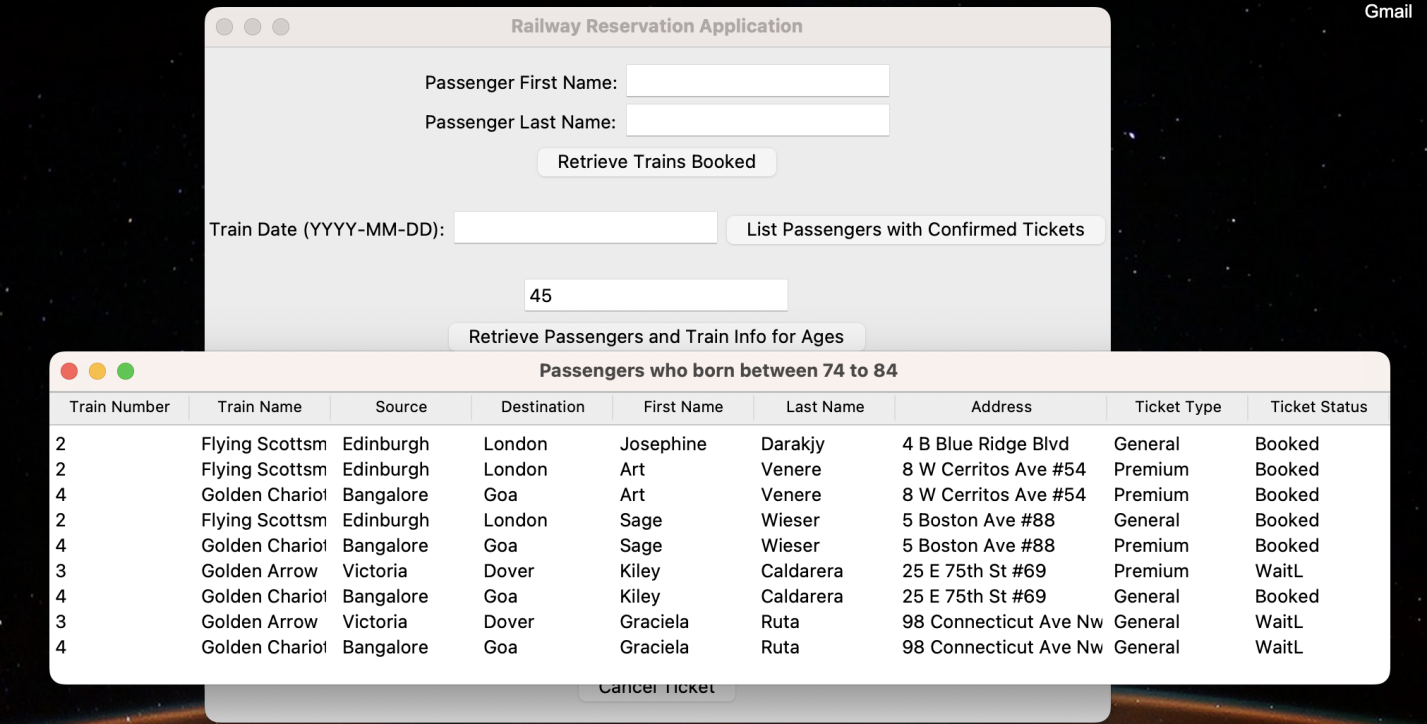
**A screenshot of a computer

Description automatically generated**

**QUERY 3:**

def query\_3():

age\_range = int(entry\_age.get())  
 start = age\_range - (age\_range % 10)  
 end = start+10  
 current\_year = datetime.now().year  
 start\_year = current\_year - start  
 end\_year = current\_year - end  
 l ast\_two\_start = str(start\_year)[-2:]  
 last\_two\_end = str(end\_year)[-2:]  
 query = f"""  
 SELECT   
 t."Train Number",   
 t."Train Name",   
 t."Source Station",   
 t."Destination Station",   
 p.first\_name,   
 p.last\_name,   
 p.address,   
 b.Ticket\_Type,   
 b.Staus  
 FROM   
 Passenger p   
 JOIN Booked b ON p.SSN = b.Passanger\_ssn   
 JOIN Train t ON b.Train\_Number = t."Train Number"  
 WHERE   
 CAST(substr(p.bdate, -2) AS INTEGER) BETWEEN {last\_two\_end} AND {last\_two\_start}  
 """  
 print(query)  
 result = execute\_query(query)

****

**QUERY 4:**

def query\_4():

query = "SELECT Train.\"Train Name\" , COUNT(\*) FROM Booked JOIN Train ON Booked.Train\_Number = Train.\"Train Number\" GROUP BY Train.\"Train Name\""

result = execute\_query(query)

messagebox.showinfo("Passenger Count by Train", "Passenger count by Train Name: \n" + "\n".join([f"{row[0]}: {row[1]} passengers" for row in result])

**A screenshot of a computer

Description automatically generated**

**QUERY 5:**

def query\_5():

train\_name = entry\_train\_name.get()

query = "SELECT Passenger.first\_name, Passenger.last\_name FROM Booked JOIN Passenger ON Booked.Passanger\_ssn = Passenger.SSN WHERE Booked.Train\_Number=(SELECT \"Train Number\" FROM Train WHERE \"Train Name\"=?) AND Booked.Staus='Booked'"

result = execute\_query(query, (" "+train\_name,))

messagebox.showinfo("Passengers with Confirmed Tickets in a Train", "Passengers with confirmed tickets in the entered train: \n" + "\n".join([f"{row[0]} {row[1]}" for row in result])

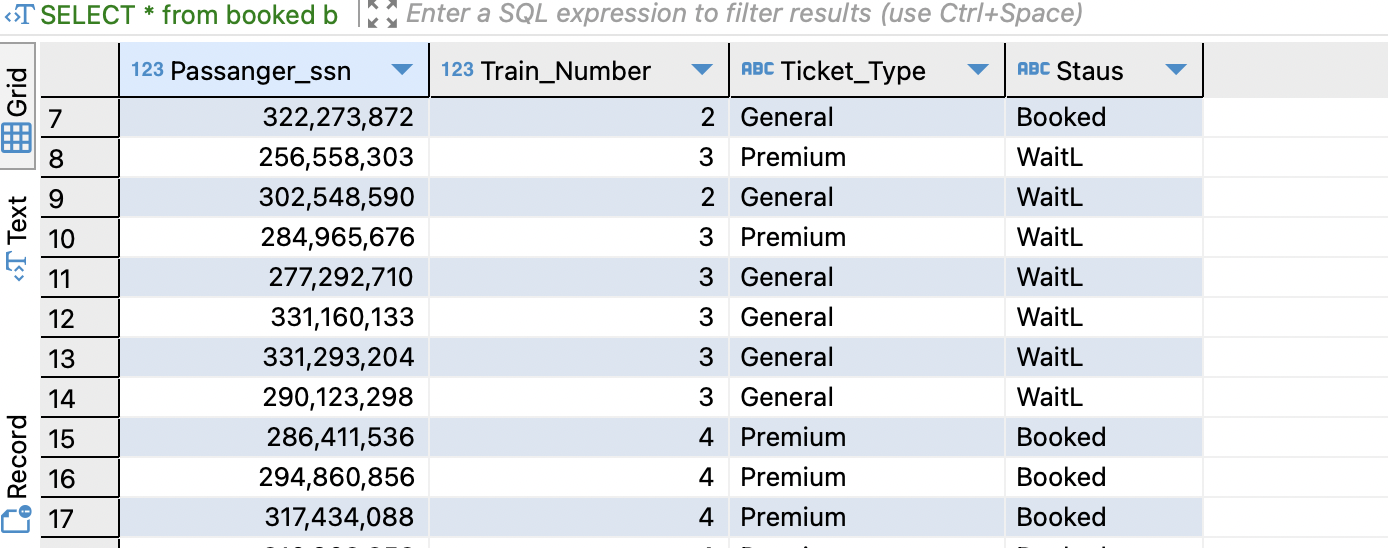
**A screenshot of a computer

Description automatically generated**

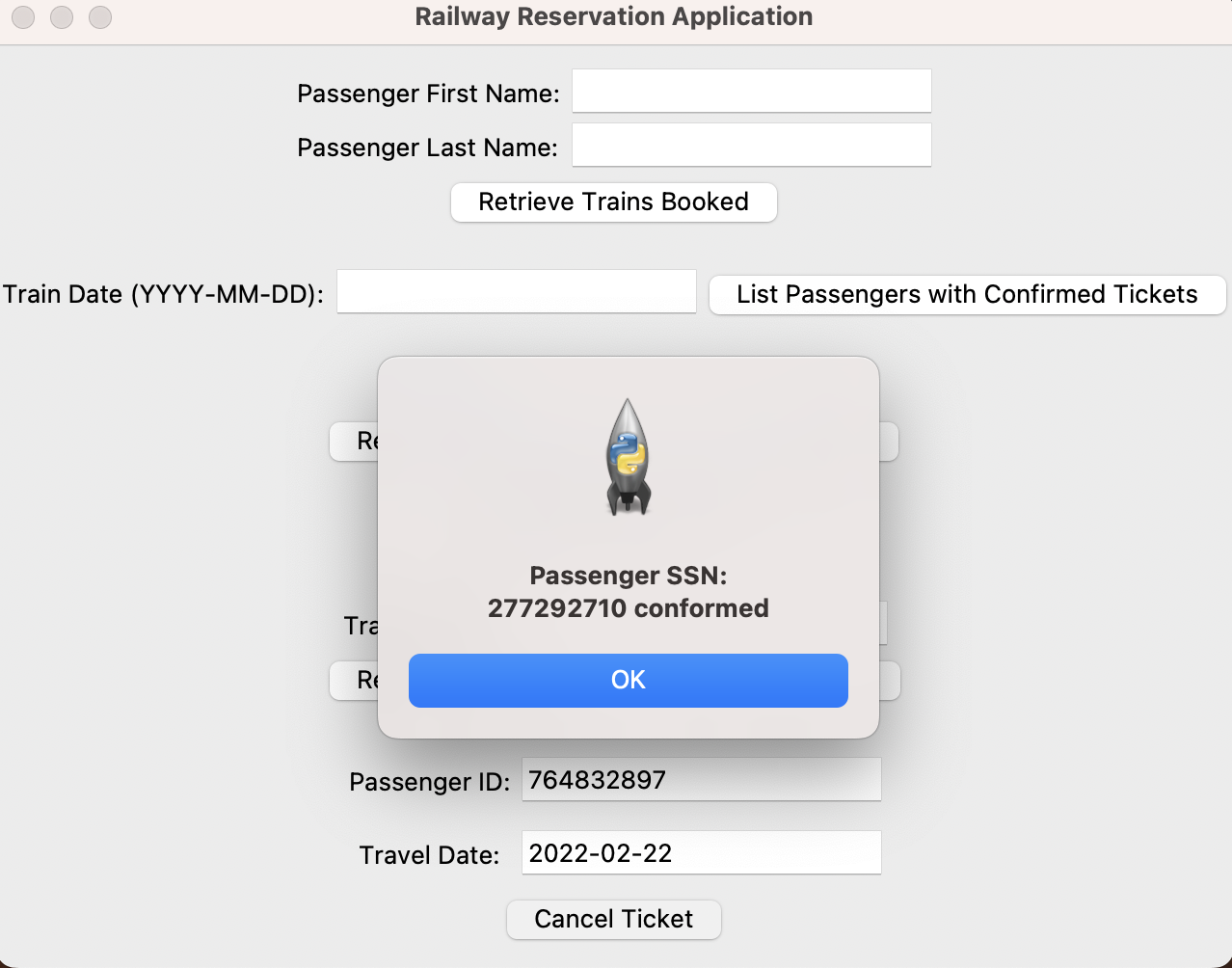
**QUERY 6:**

def cancel\_ticket():  
 passenger\_id = passenger\_id\_entry.get()  
 travel\_date = travel\_date\_entry.get()  
 try:  
 output\_show = f"""SELECT Passanger\_ssn   
 FROM booked   
 WHERE Ticket\_Type = (  
 SELECT Ticket\_Type   
 FROM booked   
 WHERE Passanger\_ssn = {passenger\_id})   
 AND Train\_Number = (  
 SELECT Train\_Number   
 FROM booked   
 WHERE Passanger\_ssn = {passenger\_id})   
 AND Staus = 'WaitL'   
 LIMIT 1 """  
 result\_show = execute\_query(output\_show)  
 query = f"""  
 UPDATE Booked  
 SET Staus = 'Booked'  
 WHERE Passanger\_ssn = (  
 SELECT Passanger\_ssn  
 FROM Booked  
 WHERE Ticket\_Type = (  
 SELECT Ticket\_Type  
 FROM Booked  
 WHERE Passanger\_ssn = {passenger\_id}  
 )  
 AND Train\_Number = (  
 SELECT Train\_Number  
 FROM Booked  
 WHERE Passanger\_ssn = {passenger\_id}  
 )  
 AND Staus = 'WaitL'  
 AND EXISTS (  
 SELECT 1  
 FROM Train\_Status ts  
 JOIN Train t ON TRIM(t."Train Name") = TRIM(ts.TrainName)  
 WHERE ts.TrainDate = '{travel\_date}'  
 AND t."Train Number" = (SELECT Train\_Number FROM Booked WHERE Passanger\_ssn = '{passenger\_id}')  
 AND (ts.PremiumSeatsAvailable > 0 OR ts.GenSeatsAvailable > 0)  
 )  
 LIMIT 1  
 );  
 """  
 query\_delete = f"""DELETE FROM Booked WHERE Passanger\_ssn = {passenger\_id}"""  
 print(query)  
 execute\_query(query)  
 execute\_query(query\_delete)  
 print(result\_show)  
 if result\_show:  
 messagebox.showinfo("Passenger SSN gets", f"Passenger SSN: {result\_show[0][0]} Ticket gets confirmed and {passenger\_id} Ticket gets Cancelled")  
 else:  
 messagebox.showinfo("No Result", "No result found.")  
  
 except sqlite3.Error as e:  
 messagebox.showerror("Database Error", f"An error occurred: {str(e)}")

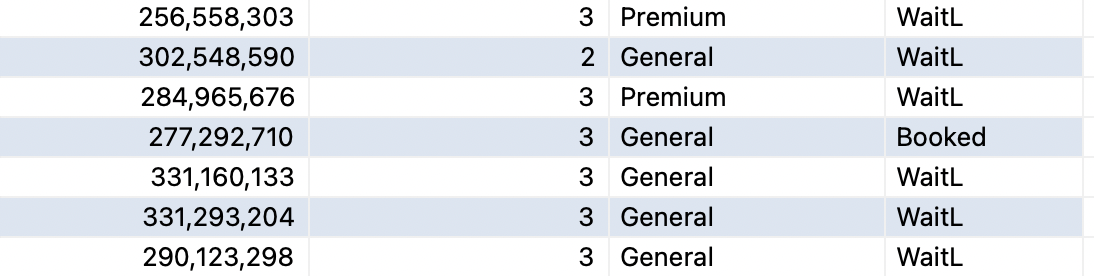
**TABLE BEFORE UPDATING:**



**UI UPDATE:**



**AFTER UPDATE:**



**TEAM CONTRIBUTION:**

**IMPORTING THE DATA TO SQLITE3**: NIVAS MARUDHAVANAN

**CONNECTING GUI TO SQL**: NIVAS MARUDHAVANAN, JENNITHRA SRINIVASAN

**BUILDING GUI**: NIVAS MARUDHAVANAN, JENNITHRA SRINIVASAN

**DEVELOPING REQUIRED QUERIES**: NIVAS MARUDHAVANAN, JENNITHRA SRINIVASAN

**DOCUMENTATION**: JENNITHRA SRINIVASAN