RAILWAY RESERVATION SYSTEM

By

Group - 20

BINDU APEKSHA ALLIPALLI

RAMYA MADHUNIKA VASTAVAI

This Report has been Submitted

in Fulfillment of the Requirements for

DASC 5300 –Foundation of Computing Course

at the University of Texas at Arlington

May 01, 2023

HONOR CODE

I pledge, on my honor, to uphold 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 sources. I will follow the highest standards of integrity and uphold the spirit of the Honor Code.

Railway Reservation System

Database:

For this Railway Reservation System, we had created the Database using the SQLITE3, which consists of 4 tables, namely Train, Train_status, Passenger and Booked. The code for these tables are given below.

CREATE: To create four tables, including Train, TrainStatus, Passenger, and Booked, complete with attributes and keys

```
#creating TRAIN table

CREATE TABLE TRAIN (

                                        "Train Number" INTEGER,

                                        "Train Name" TEXT,

                                        "Premium Fair" REAl,

                                        "General Fair" REAl,

                                        "Source Station" TEXT,

                                        "Destination Station" TEXT

                                        );
```

This SQL statement creates a new table called "TRAIN" with the following columns:

- "Train Number" with data type INTEGER
- "Train Name" with data type TEXT

- "Premium Fair" with data type REAL
- "General Fair" with data type REAL
- "Source Station" with data type TEXT
- "Destination Station" with data type TEXT

Each column is defined with its own data type, which specifies the type of data that can be stored in that column. The table can be used to store information about different trains, including their names, numbers, fares, and source and destination stations.

```
#creating Train_Status table

CREATE TABLE Train_Status (

                    TrainDate TEXT,

                    TrainName TEXT,

                    PremiumSeatsAvailable INTEGER,

                    GenSeatsAvailable INTEGER,

                    PremiumSeatsOccupied INTEGER,

                    GenSeatsOccupied INTEGER

                    );
```

This SQL statement creates a new table named `Train_Status` with six columns: `TrainDate`, `TrainName`, `PremiumSeatsAvailable`, `GenSeatsAvailable`, `PremiumSeatsOccupied`, and `GenSeatsOccupied`.

- `TrainDate` is of type `TEXT`, which is used to store the date on which the train is scheduled to run.
- `TrainName` is also of type `TEXT`, which is used to store the name of the train.
- `PremiumSeatsAvailable` is of type `INTEGER`, which is used to store the number of premium seats available on the train for the given date.
- `GenSeatsAvailable` is also of type `INTEGER`, which is used to store the number of general seats available on the train for the given date.
- `PremiumSeatsOccupied` is of type `INTEGER`, which is used to store the number of premium seats already occupied on the train for the given date.
- `GenSeatsOccupied` is also of type `INTEGER`, which is used to store the number of general seats already occupied on the train for the given date.

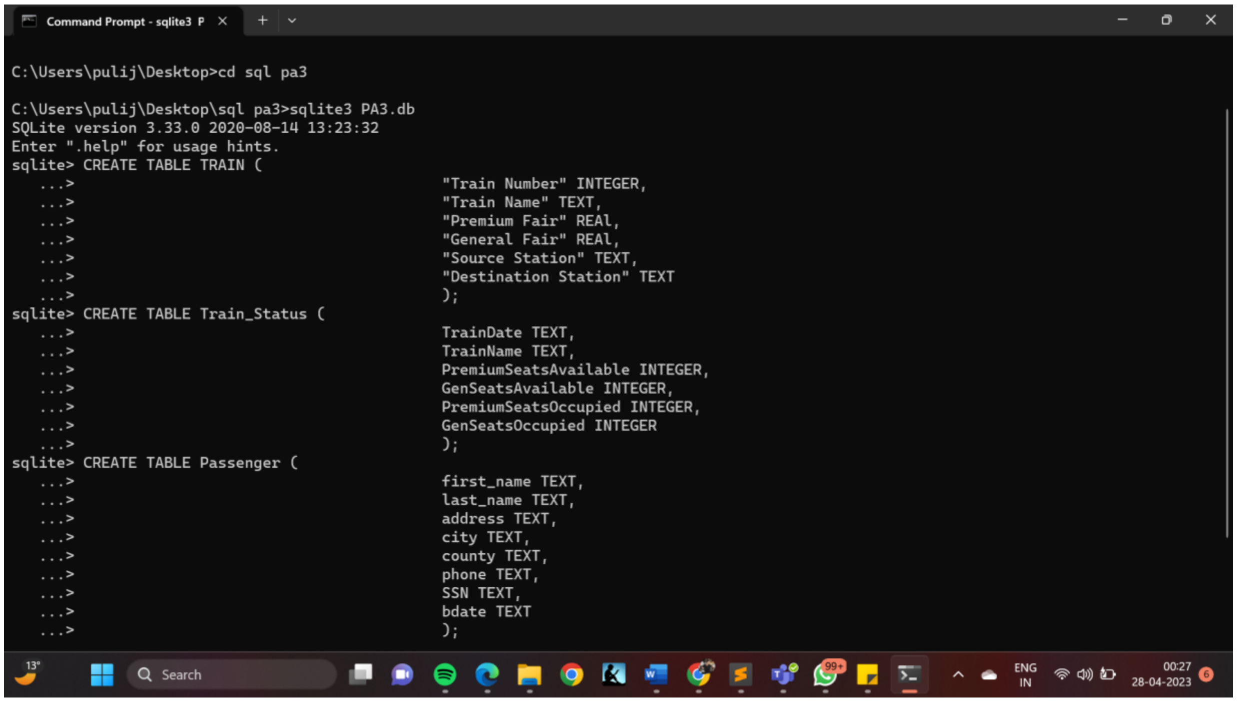
This table can be used to keep track of the availability and occupancy of seats on different trains for different dates.

```
#creating passenger table

CREATE TABLE Passenger (

                    first_name TEXT,

                    last_name TEXT,
                    address TEXT,

                    city TEXT,

                    county TEXT,

                    phone TEXT,

                    SSN TEXT,

                    bdate TEXT

                    );
```

This SQL query creates a table named "Passenger" with the following columns: "first_name", "last_name", "address", "city", "county", "phone", "SSN", and "bdate".

Each column represents a different attribute of a passenger, such as their first name, last name, address, and phone number. The "SSN" column likely represents the passenger's social security number, while the "bdate" column may represent their birth date.

The data types for each column are not specified in the query, so they will default to the default data types for SQLite (which is likely TEXT for most columns).


```
C:\Users\pulij\Desktop>cd sql pa3

C:\Users\pulij\Desktop\sql pa3>sqlite3 PA3.db
SQLite version 3.33.0 2020-08-14 13:23:32
Enter ".help" for usage hints.
sqlite> CREATE TABLE TRAIN (
   ...>                                         "Train Number" INTEGER,
   ...>                                         "Train Name" TEXT,
   ...>                                         "Premium Fair" REAL,
   ...>                                         "General Fair" REAL,
   ...>                                         "Source Station" TEXT,
   ...>                                         "Destination Station" TEXT
   ...>                                         );
sqlite> CREATE TABLE Train_Status (
   ...>                                         TrainDate TEXT,
   ...>                                         TrainName TEXT,
   ...>                                         PremiumSeatsAvailable INTEGER,
   ...>                                         GenSeatsAvailable INTEGER,
   ...>                                         PremiumSeatsOccupied INTEGER,
   ...>                                         GenSeatsOccupied INTEGER
   ...>                                         );
sqlite> CREATE TABLE Passenger (
   ...>                                         first_name TEXT,
   ...>                                         last_name TEXT,
   ...>                                         address TEXT,
   ...>                                         city TEXT,
   ...>                                         county TEXT,
   ...>                                         phone TEXT,
   ...>                                         SSN TEXT,
   ...>                                         bdate TEXT
   ...>                                         );
```

#creating Booked table

```
CREATE TABLE Booked (

                                             Passanger_ssn TEXT,

                                             Train_Number INTEGER,

                                             Ticket_Type TEXT,

                                             Status TEXT

                                             );
```

This SQL command creates a new table named "Booked" with the following columns:

- "Passenger_ssn": a TEXT field that stores the social security number of the passenger who made the booking.
- "Train_Number": an INTEGER field that stores the unique identification number of the train.
- "Ticket_Type": a TEXT field that specifies the type of the ticket booked, such as "Premium" or "General".
- "Status": a TEXT field that indicates the current status of the booking, such as "Confirmed" or "Cancelled".

The CREATE TABLE command is used to define the structure of a new table in a database, including its column names and data types.

We have 4 datasets which are booked.csv, Train.csv, passenger.csv and Trainstatus.csv which have to be imported to the database ‘PA3.db’ using “.import booked.csv Booked” and others respectively.

SELECT: To print every table in the database.

#printing tables:

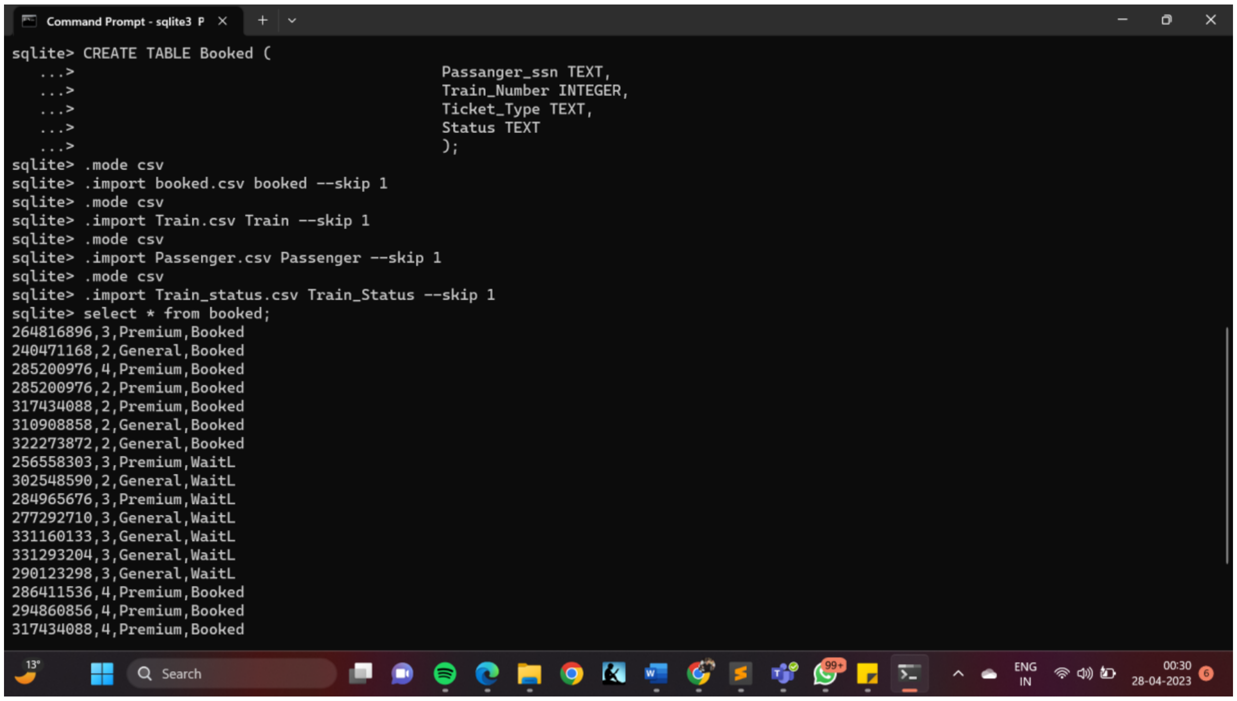
Select * from booked;

Select * from Passenger;

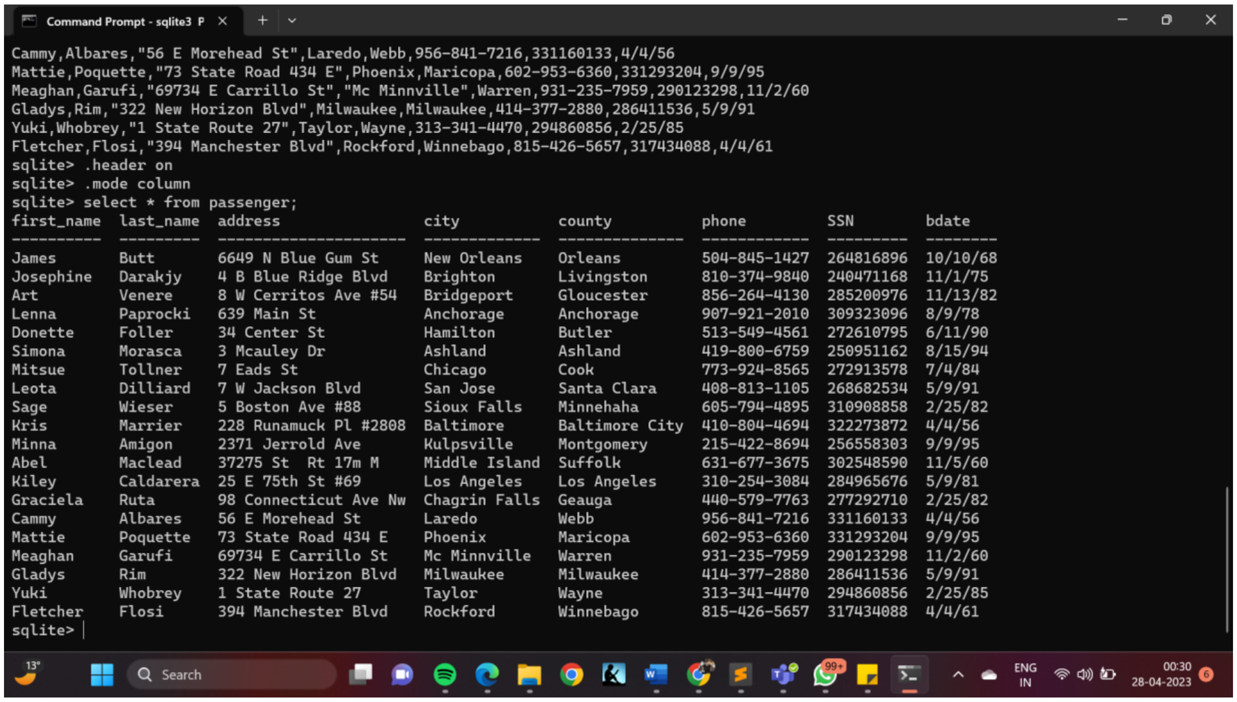
Select * from Train;

Select * from Train_status;

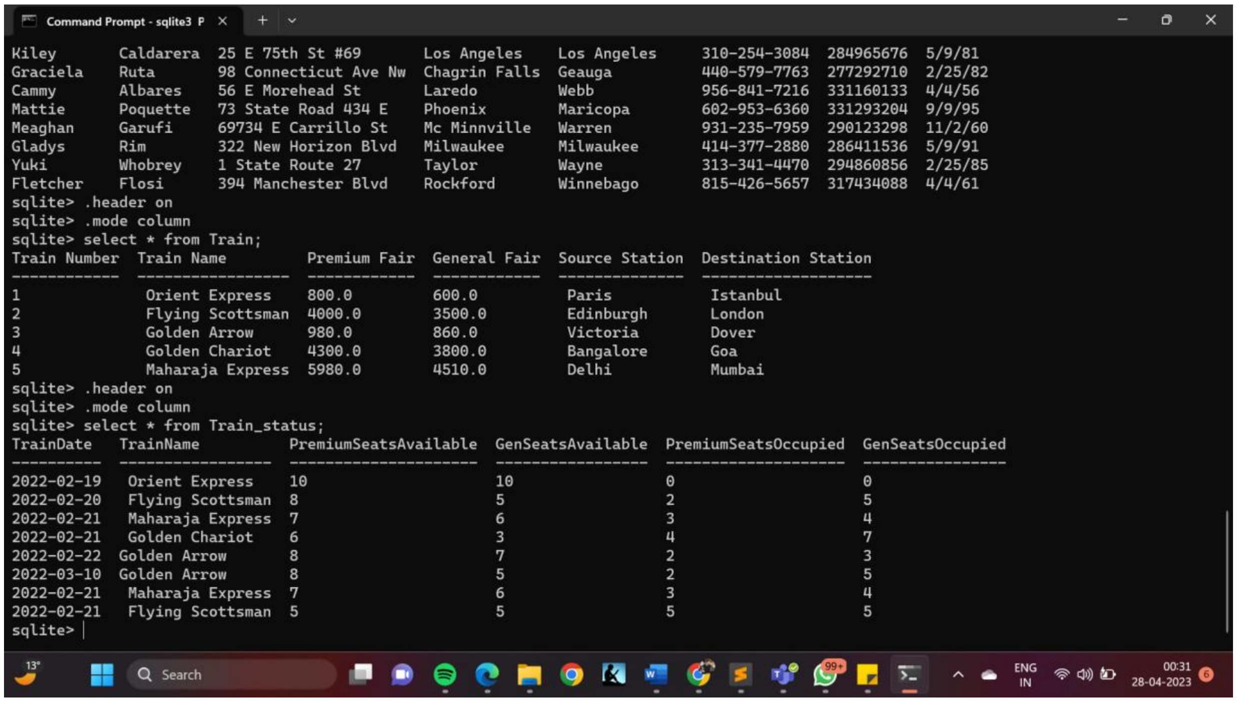
SELECT * will retrieve all columns and rows from the respective tables. If the tables have a large number of rows, this could result in a significant amount of data being returned. It's generally good practice to limit the columns and rows returned to only the ones needed for your specific query. We used .header on and .mode column in order to print the column names and print the values column-wise.



```
sqlite> CREATE TABLE Booked (
   ...>                                         Passanger_ssn TEXT,
   ...>                                         Train_Number INTEGER,
   ...>                                         Ticket_Type TEXT,
   ...>                                         Status TEXT
   ...>                                         );
sqlite> .mode csv
sqlite> .import booked.csv booked --skip 1
sqlite> .mode csv
sqlite> .import Train.csv Train --skip 1
sqlite> .mode csv
sqlite> .import Passenger.csv Passenger --skip 1
sqlite> .mode csv
sqlite> .import Train_status.csv Train_Status --skip 1
sqlite> select * from booked;
264816896,3,Premium,Booked
240471168,2,General,Booked
285200976,4,Premium,Booked
285200976,2,Premium,Booked
317434088,2,Premium,Booked
310908858,2,General,Booked
322273872,2,General,Booked
256558303,3,Premium,WaitL
302548590,2,General,WaitL
284965676,3,Premium,WaitL
277292710,3,General,WaitL
331160133,3,General,WaitL
331293204,3,General,WaitL
290123298,3,General,WaitL
286411536,4,Premium,Booked
294860856,4,Premium,Booked
317434088,4,Premium,Booked
```



```
Cammy,Albares,"56 E Morehead St",Laredo,Webb,956-841-7216,331160133,4/4/56
Mattie,Poquette,"73 State Road 434 E",Phoenix,Maricopa,602-953-6360,331293204,9/9/95
Meaghan,Garufi,"69734 E Carrillo St","Mc Minnville",Warren,931-235-7959,290123298,11/2/60
Gladys,Rim,"322 New Horizon Blvd",Milwaukee,Milwaukee,414-377-2880,286411536,5/9/91
Yuki,Whobrey,"1 State Route 27",Taylor,Wayne,313-341-4470,294860856,2/25/85
Fletcher,Flosi,"394 Manchester Blvd",Rockford,Winnebago,815-426-5657,317434088,4/4/61
sqlite> .header on
sqlite> .mode column
sqlite> select * from passenger;
first_name  last_name  address                city           county           phone         SSN        bdate
----------  ---------  ---------------------  -------------  ---------------  ------------  ---------  --------
James       Butt       6649 N Blue Gum St     New Orleans    Orleans          504-845-1427  264816896  10/10/68
Josephine   Darakjy    4 B Blue Ridge Blvd    Brighton       Livingston       810-374-9840  240471168  11/1/75
Art         Venere     8 W Cerritos Ave #54   Bridgeport     Gloucester       856-264-4130  285200976  11/13/82
Lenna       Paprocki   639 Main St            Anchorage      Anchorage        907-921-2010  309323096  8/9/78
Donette     Foller     34 Center St           Hamilton       Butler           513-549-4561  272610795  6/11/90
Simona      Morasca    3 Mcauley Dr           Ashland        Ashland          419-800-6759  250951162  8/15/94
Mitsue      Tollner    7 Eads St              Chicago        Cook             773-924-8565  272913578  7/4/84
Leota       Dilliard   7 W Jackson Blvd       San Jose       Santa Clara      408-813-1105  268682534  5/9/91
Sage        Wieser     5 Boston Ave #88       Sioux Falls    Minnehaha        605-794-4895  310908858  2/25/82
Kris        Marrier    228 Runamuck Pl #2808  Baltimore      Baltimore City   410-804-4694  322273872  4/4/56
Minna       Amigon     2371 Jerrold Ave       Kulpsville     Montgomery       215-422-8694  256558303  9/9/95
Abel        Maclead    37275 St  Rt 17m M     Middle Island  Suffolk          631-677-3675  302548590  11/5/60
Kiley       Caldarera  25 E 75th St #69       Los Angeles    Los Angeles      310-254-3084  284965676  5/9/81
Graciela    Ruta       98 Connecticut Ave Nw  Chagrin Falls  Geauga           440-579-7763  277292710  2/25/82
Cammy       Albares    56 E Morehead St       Laredo         Webb             956-841-7216  331160133  4/4/56
Mattie      Poquette   73 State Road 434 E    Phoenix        Maricopa         602-953-6360  331293204  9/9/95
Meaghan     Garufi     69734 E Carrillo St    Mc Minnville   Warren           931-235-7959  290123298  11/2/60
Gladys      Rim        322 New Horizon Blvd   Milwaukee      Milwaukee        414-377-2880  286411536  5/9/91
Yuki        Whobrey    1 State Route 27       Taylor         Wayne            313-341-4470  294860856  2/25/85
Fletcher    Flosi      394 Manchester Blvd    Rockford       Winnebago        815-426-5657  317434088  4/4/61
sqlite> |
```



```
Kiley       Caldarera  25 E 75th St #69       Los Angeles    Los Angeles      310-254-3084  284965676  5/9/81
Graciela    Ruta       98 Connecticut Ave Nw  Chagrin Falls  Geauga           440-579-7763  277292710  2/25/82
Cammy       Albares    56 E Morehead St       Laredo         Webb             956-841-7216  331160133  4/4/56
Mattie      Poquette   73 State Road 434 E    Phoenix        Maricopa         602-953-6360  331293204  9/9/95
Meaghan     Garufi     69734 E Carrillo St    Mc Minnville   Warren           931-235-7959  290123298  11/2/60
Gladys      Rim        322 New Horizon Blvd   Milwaukee      Milwaukee        414-377-2880  286411536  5/9/91
Yuki        Whobrey    1 State Route 27       Taylor         Wayne            313-341-4470  294860856  2/25/85
Fletcher    Flosi      394 Manchester Blvd    Rockford       Winnebago        815-426-5657  317434088  4/4/61
sqlite> .header on
sqlite> .mode column
sqlite> select * from Train;
Train Number  Train Name         Premium Fair  General Fair  Source Station  Destination Station
------------  -----------------  ------------  ------------  --------------  -------------------
1              Orient Express    800.0         600.0          Paris           Istanbul
2              Flying Scottsman  4000.0        3500.0         Edinburgh       London
3              Golden Arrow      980.0         860.0          Victoria        Dover
4              Golden Chariot    4300.0        3800.0         Bangalore       Goa
5              Maharaja Express  5980.0        4510.0         Delhi           Mumbai
sqlite> .header on
sqlite> .mode column
sqlite> select * from Train_status;
TrainDate   TrainName          PremiumSeatsAvailable  GenSeatsAvailable  PremiumSeatsOccupied  GenSeatsOccupied
----------  -----------------  ---------------------  -----------------  --------------------  ----------------
2022-02-19   Orient Express    10                     10                 0                     0
2022-02-20   Flying Scottsman  8                      5                  2                     5
2022-02-21   Maharaja Express  7                      6                  3                     4
2022-02-21   Golden Chariot    6                      3                  4                     7
2022-02-22  Golden Arrow       8                      7                  2                     3
2022-03-10  Golden Arrow       8                      5                  2                     5
2022-02-21   Maharaja Express  7                      6                  3                     4
2022-02-21   Flying Scottsman  5                      5                  5                     5
sqlite> |
```

Designing GUI:

Creating the Railway Reservation System GUI main window:

```
import tkinter as tk

import sqlite3

# create a connection to the database

conn = sqlite3.connect('PA3.db')

# create a cursor to execute SQL queries

cur = conn.cursor()

#Print tables

cur.execute("SELECT *FROM Train")

rows=cur.fetchall()

print("Train")

for i in rows:

        print(i)

cur.execute("SELECT *FROM Train_Status")

rows=cur.fetchall()

print("TrainStatus")

for i in rows
```

```
  print(i)



cur.execute("SELECT *FROM Passenger")

rows=cur.fetchall()

print("Passenger")

for i in rows:

        print(i)



cur.execute("SELECT *FROM Booked")

rows=cur.fetchall()

print("Booked")

for i in rows:

        print(i)
```

Query 1:

User input the passenger’s last name and first name and retrieve all trains they are booked on.

GUI python code:

```
def queryfun1():

    # Get the passenger's first and last name

    fname = ent1.get()

    lname = ent2.get()



    # Execute the SQL query

    cur.execute('''select t.'Train Number',t.'Train Name' from Booked as b

                join Passenger as p on b.Passanger_SSN=SSN

                join Train as t on b.Train_Number=t.'Train Number'

                where p.First_Name=? and p.Last_Name=?

                 ''', (fname, lname))



    # Display the results in the listbox

    result = cur.fetchall()

    #textbox.delete('1.0', tk.END)
    for row in result:
        textbox.insert(tk.END, f"Train Number: {row[0]}, Train Name: {row[1]}\n")
```

SQL Query:

```
select t.'Train Number',t.'Train Name' from Booked as b
                    join Passenger as p on b.Passanger_SSN=SSN
                    join Train as t on b.Train_Number=t.'Train Number'
                    where p.First_Name=? and p.Last_Name=?
```

Explanation:

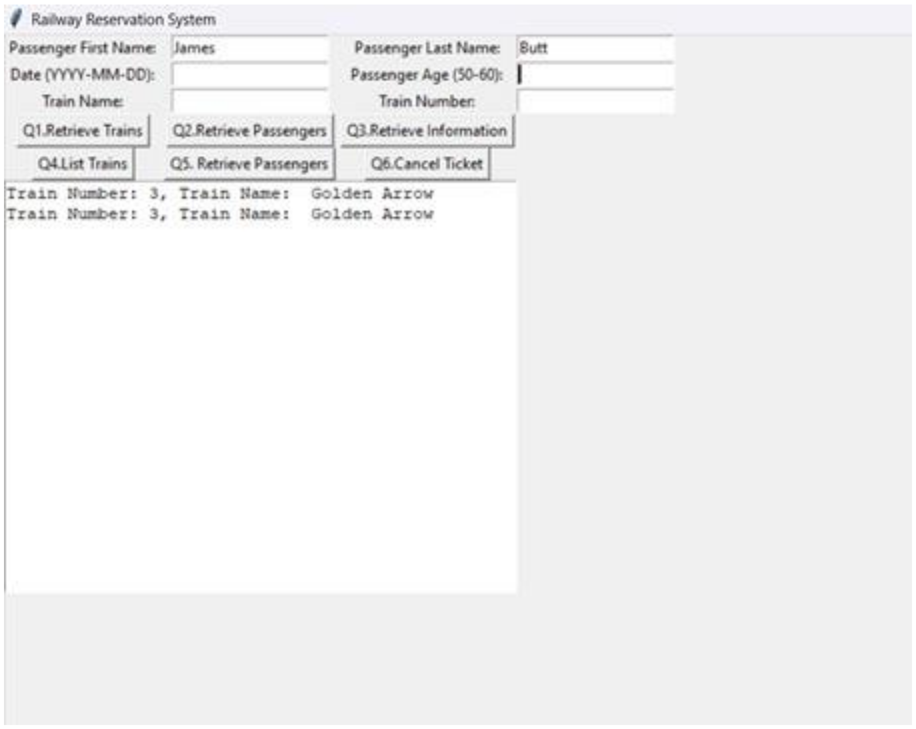
This SQL query looks up the Train Number and Train Name for a particular passenger by first and last name from the Train table.

In order to match the bookings with the appropriate passengers, the query first performs a JOIN operation between the Booked and Passenger records using the Passanger_SSN and SSN columns, respectively.

The train information related to each booking is then retrieved using a second JOIN with the Train database using the Train_Number column from the Booked table and the 'Train Number' column from the Train table.

The WHERE clause also filters the results based on the passenger's first and last name. When the query is run, the? placeholders are probably meant to be substituted with actual values.

Overall, using a passenger's first and last name, you can use this query to learn which trains they have reserved.



```
Train Number: 3, Train Name:  Golden Arrow
Train Number: 3, Train Name:  Golden Arrow
```

Query 2:

User input the Date and list of passengers travelling on entered day with confirmed tickets displays on UI.

GUI Python code:

```
def queryfun2():

    # Get the date

    date = ent3.get()



    # Execute the SQL query

    cur.execute('''SELECT p.First_Name, p.Last_Name FROM Passenger as p, Train_Status
                    join Booked as b on b.Passanger_SSN=p.SSN

                    WHERE Status='Booked' AND 'Train Number' IN (SELECT 'Train Number'

                    FROM Train WHERE TrainDate=?)

                    ''', (date,))


    # Display the results in the listbox

    result = cur.fetchall()

    textbox.delete('1.0', tk.END)

    for row in result:

        textbox.insert(tk.END, f"{row[0]} {row[1]}\n")
```

SQL QUERY:

```
SELECT p.First_Name, p.Last_Name FROM Passenger as p, Train_Status

                    join Booked as b on b.Passanger_SSN=p.SSN

                    WHERE Status='Booked' AND 'Train Number' IN (SELECT 'Train Number'

                    FROM Train WHERE TrainDate=?)
```

Explanation:

This SQL query will return the first and last names of anyone who has reserved a seat on a train for a particular date using a parameter.

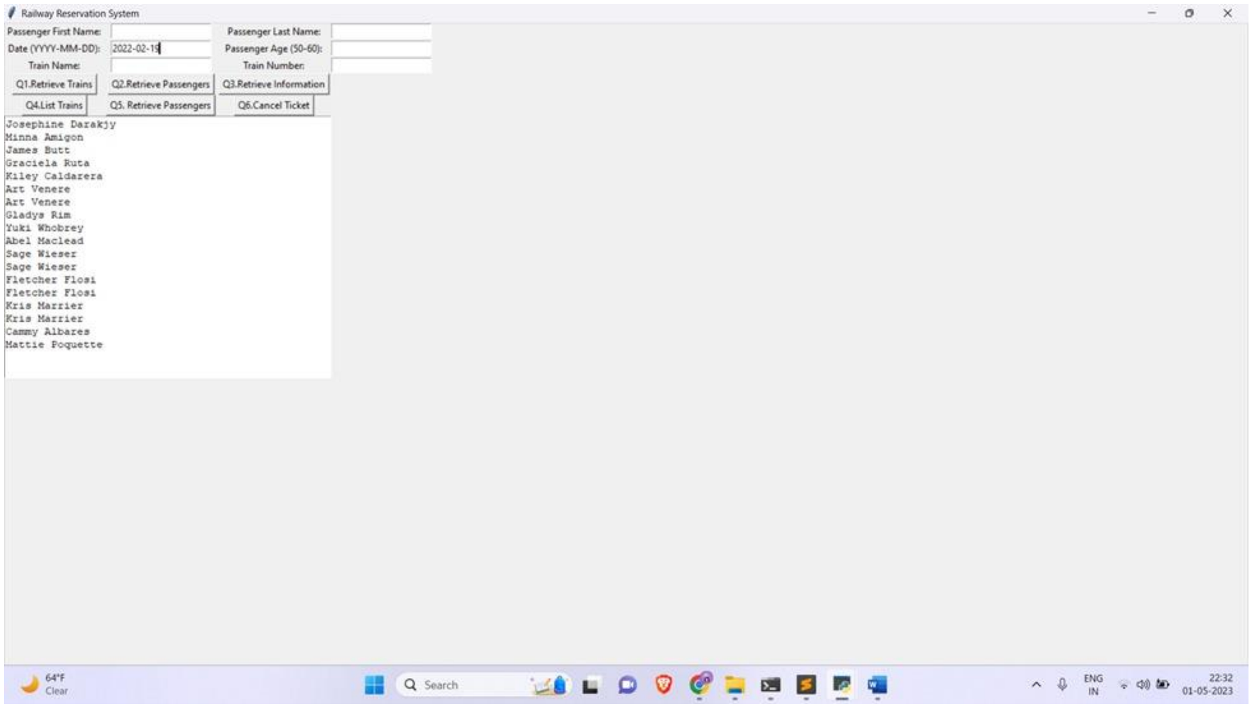
The query begins with a cross join between the Train_Status and Passenger tables. Every row in the Passenger table gets linked with every record in the Train_Status table because the join does not specify any criteria for matching rows.

The bookings made by each passenger are then retrieved using an inner join with the Booked database using the Passanger_SSN column from Booked and the SSN column from the Passenger table.

The WHERE clause then restricts the list of bookings to those with the status "Booked" and for trains scheduled on the given date. In order to do this, bookings are matched with the appropriate trains using the TrainNumber column from the Train database and the 'Train Number' column from the Booked table. A subquery is then used to extract the 'Train Number' values for the trains that are scheduled on the given day.

Finally, the SELECT clause only returns the passengers' first and last names.

In general, you can use this query to learn which people have reserved a train for a particular date.



Passenger First Name:
Passenger Last Name:
Date (YYYY-MM-DD): 2022-02-19
Passenger Age (50-60):
Train Name:
Train Number:

```
Josephine Darakjy
Minna Amigon
James Butt
Graciela Ruta
Kiley Caldarera
Art Venere
Art Venere
Gladys Rim
Yuki Whobrey
Abel Maclead
Sage Wieser
Sage Wieser
Fletcher Flosi
Fletcher Flosi
Kris Marrier
Kris Marrier
Cammy Albares
Mattie Poquette
```

Query 3:

GUI Python code:

```
def ViewPassengerAge():

    age_entry = age.get()

    age_entry = int(age_entry)

    ViewPassengerAge_conn = sqlite3.connect('PA3.db')

    ViewPassengerAge_cur = ViewPassengerAge_conn.cursor()

    ViewPassengerAge_cur.execute("""SELECT Train.'Train Number', Train.'Train Name',
Train.'Source Station', Train.'Destination Station', Passenger.first_name, Passenger.last_name,
Passenger.address, Booked.Ticket_Type, Booked.Status FROM Booked INNER JOIN Passenger
ON Booked.Passanger_ssn = Passenger.SSN INNER JOIN Train ON Booked.Train_Number = Train.'Train Number' WHERE strftime('%Y', "now") - CAST(substr(Passenger.bdate,7,4) as INTEGER) BETWEEN ? AND ? ORDER BY Train.'Train Number'""", (age_entry,age_entry,))

    records = ViewPassengerAge_cur.fetchall()

    print_records = '\n'.join([f"{record[0]} {record[1]} {record[2]} {record[3]} {record[4]} {record[5]} {record[6]} {record[7]} {record[8]}" for record in records])

    ViewPassengerAge_conn_label = Label(root, text = print_records)

    ViewPassengerAge_conn_label.grid(row = 9, column = 0, columnspan = 2)

    ViewPassengerAge_conn.commit()

    ViewPassengerAge_conn.close()

ViewPassengerAge_button = Button(root, text = 'View Passenger between Age 50 and 60', command = ViewPassengerAge)

ViewPassengerAge_button.grid(row = 15, column = 2, pady = 10, padx = 10)
```

SQL QUERY:

```
SELECT Train.'Train Number', Train.'Train Name', Train.'Source Station', Train.'Destination Station', Passenger.first_name, Passenger.last_name, Passenger.address, Booked.Ticket_Type, Booked.Status FROM Booked INNER JOIN Passenger ON Booked.Passanger_ssn = Passenger.SSN INNER JOIN Train ON Booked.Train_Number = Train.'Train Number' WHERE strftime('%Y', "now") - CAST(substr(Passenger.bdate,7,4) as INTEGER) BETWEEN ? AND ? ORDER BY Train.'Train Number
```

Explanation:

The above is a SQL query that retrieves data from three tables - 'Booked', 'Passenger', and 'Train'. The query joins these tables using their respective keys and filters the results based on a condition.

The condition that is used in this query is based on the age of the passengers. Specifically, the query selects the passengers whose year of birth is between a certain range of years. The range is calculated based on the current year (obtained using the strftime function), minus the year of birth of the passenger (extracted from the 'bdate' field of the 'Passenger' table).

The selected data includes the train number, train name, source station, destination station, passenger's first and last names, address, ticket type, and booking status. The results are sorted in ascending order by the train number.

Overall, it seems like a well-written query that effectively retrieves the required data from the three tables, using appropriate join conditions and filters.

Query 4:

List all the train name along with count of passengers it is carrying.

GUI Python code:

```
def queryfun4():

    # Execute the SQL query

    cur.execute('''SELECT Train.'Train Name', COUNT(Booked.Passanger_ssn) AS
Passenger_Count FROM Train JOIN Booked ON Train.'Train Number' =
Booked.Train_Number GROUP BY Train.'Train Name';''')



    # Display the results in the listbox

    result = cur.fetchall()

    textbox.delete('1.0', tk.END)

    for row in result:

        textbox.insert(tk.END, f"Train Name: {row[0]}, Number of Passengers: {row[1]}\n")
```

SQL QUERY:

```
SELECT Train.'Train Name', COUNT(Booked.Passanger_ssn) AS Passenger_Count

FROM Train JOIN Booked ON Train.'Train Number' = Booked.Train_Number

GROUP BY Train.'Train Name';
```

<u>Explanation:</u>

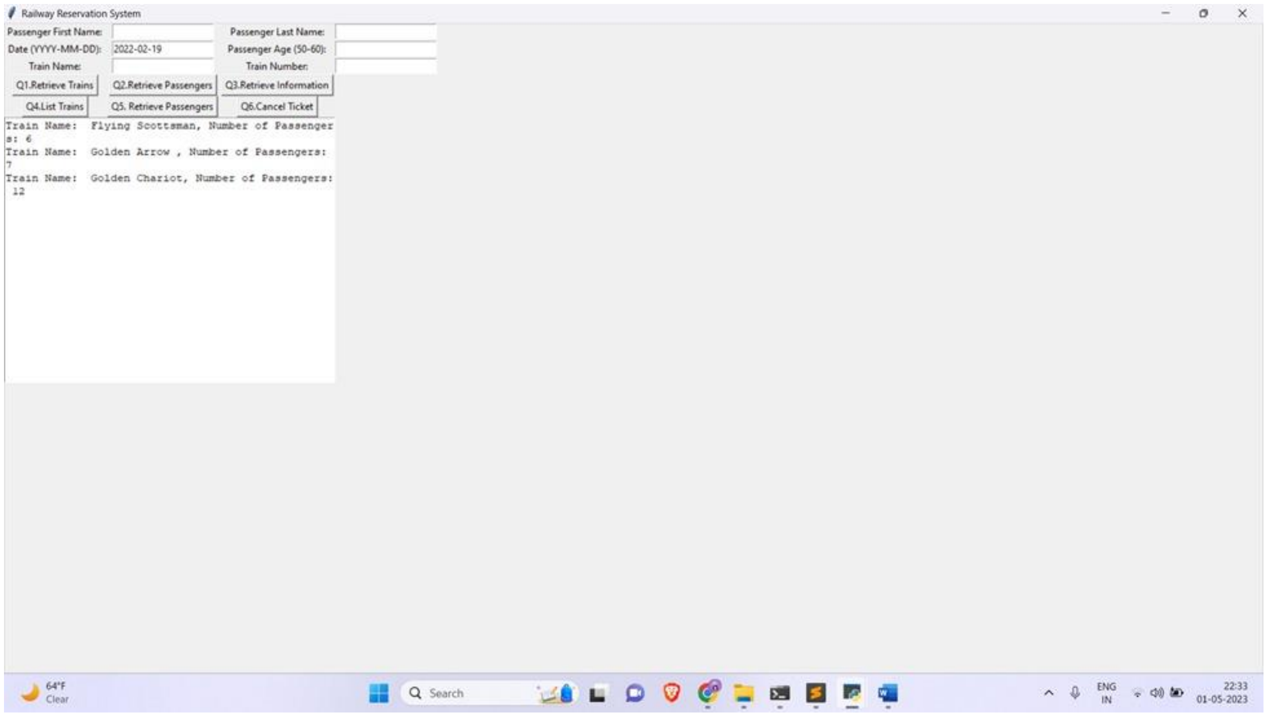
This SQL query returns the names of each train as well as the number of people who have purchased tickets for that train.

The query begins with an inner join between the Train and Booked databases, matching reservations with the appropriate trains using the 'Train Number' column from the Train table and the Train_Number column from the Booked table.

The query then counts how many times each passenger SSN appears in the Booked table for each train using the COUNT() method. 'Passenger_Count' is the alias for this number.

Using the GROUP BY clause, the query then organizes the outcomes according to the 'Train Name' column from the Train table. As a result, the results are organized by train, and the query is able to determine the passenger count for each train separately.

This query gives you an overview of the number of passengers for each train in the Booked table.



Date (YYYY-MM-DD): 2022-02-19

```
Train Name:  Flying Scottsman, Number of Passenger
s: 6
Train Name:  Golden Arrow , Number of Passengers:
7
Train Name:  Golden Chariot, Number of Passengers:
 12
```

Query 5:

Enter a train name and retrieve all the passengers with confirmed status travelling in that train.

GUI Python code:

```
def queryfun5():

    # Get the train name

    tname = ent5.get()

    # Execute the SQL query

    cur.execute('''SELECT Passenger.first_name, Passenger.last_name, Booked.Ticket_Type
        FROM Booked, passenger, train where Booked.Train_Number = Train.'Train Number' and
Booked.Passanger_ssn = Passenger.SSN and Train.'Train Name' = ? AND Booked.Status =
'Booked';''', (tname,))


    # Display the results in the listbox
    result = cur.fetchall()
    textbox.delete('1.0', tk.END)
    for row in result:
        textbox.insert(tk.END, f"{row[0]} {row[1]}\n")
```

SQL Query:

```
SELECT Passenger.first_name, Passenger.last_name, Booked.Ticket_Type
        FROM Booked, passenger, train where Booked.Train_Number = Train.'Train Number' and
Booked.Passanger_ssn = Passenger.SSN and Train.'Train Name' = ? AND Booked.Status =
'Booked';
```

Explanation:

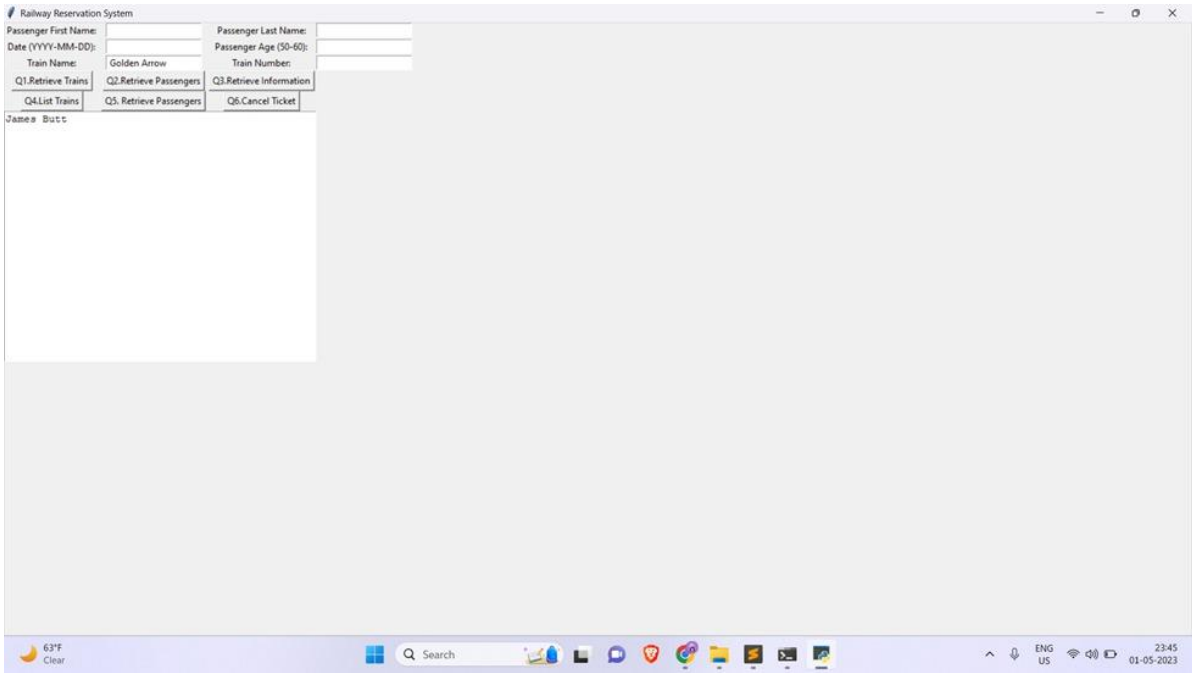
This SQL query returns the first name, last name, and kind of ticket for passengers whose booking status is "Booked" and who have purchased tickets for a certain train that is identified by name.

The Booked, Passenger, and Train tables are cross-joined at the beginning of the query. Every row in the Booked table gets linked with every row in the Passenger and Train tables because the join does not define any criteria for matching rows.

Then, using the "Train Name" column from the Train table to match the train name specified in the parameter, the WHERE clause filters the results to only include bookings for the particular train name and bookings with a status of "Booked."

The SELECT clause also returns the passenger's ticket type and first and last names.

In general, this query enables you to receive information about travelers who have purchased tickets for a particular train and whose status is "Booked."



Passenger First Name:
Passenger Last Name:
Date (YYYY-MM-DD):
Passenger Age (50-60):
Train Name: Golden Arrow
Train Number:

```
James Butt
```

QUERY 6:

GUI Python code:

```
def cancelTicket():

    cancelTicket_conn = sqlite3.connect('PA3.db')

    cancelTicket_cur = cancelTicket_conn.cursor()
    cancelTicket_cur.execute("SELECT * FROM Booked,passenger WHERE Passanger_ssn = ? AND Train_Number = ? AND Status = 'Booked'",(ssn.get(), TrainNumber.get(), ))

    cancelTicket_cur.execute("DELETE FROM Booked,passenger WHERE Passanger_ssn = ? AND Train_Number = ? ",(ssn.get(),TrainNumber.get(),))

    if ticket_type == 'Premium':

        cancelTicket_cur.execute("UPDATE Booked SET Status = 'Booked' WHERE Train_Number = ? AND Ticket_Type = 'Premium' AND Status = 'WaitL' AND Passanger_ssn IN (SELECT Passanger_ssn FROM Booked WHERE Train_Number = ? AND Ticket_Type = 'Premium' AND Status = 'WaitL' ORDER BY Train_Number LIMIT 1)",(TrainNumber.get(),TrainNumber.get(),))

    else:

        cancelTicket_cur.execute("UPDATE Booked SET Status = 'Booked' WHERE Train_Number = ? AND Ticket_Type = 'General' AND Status = 'WaitL' AND Passanger_ssn IN (SELECT Passanger_ssn FROM Booked WHERE Train_Number = ? AND Ticket_Type = 'General' AND Status = 'WaitL' ORDER BY Train_Number LIMIT 1)", (TrainNumber.get(),TrainNumber.get(),))

    cursor.execute(query)

    conn.commit()



    records = cancelTicket_cur.fetchall()
    print_records = '\n'.join([f"{record[0]} {record[1]} {record[2]} {record[3]}" for record in
records])

    cancelTicket_conn_label = Label(root, text = print_records)

    cancelTicket_conn_label.grid(row = 17, column = 0, columnspan = 2)

    cancelTicket_conn.commit()

    cancelTicket_conn.close()



cancelTicket_button = Button(root, text = 'cancel Ticket', command = cancelTicket)

cancelTicket_button.grid(row = 16, column = 2, pady = 10, padx = 10)
```

SQL QUERY:

```
SELECT * FROM Booked,passenger WHERE Passanger_ssn = ? AND Train_Number = ? AND
Status = 'Booked'",(ssn.get(), TrainNumber.get(), ))

    cancelTicket_cur.execute("DELETE FROM Booked,passenger WHERE Passanger_ssn = ?
AND Train_Number = ? ",(ssn.get(),TrainNumber.get(),))

    if ticket_type == 'Premium':

        cancelTicket_cur.execute("UPDATE Booked SET Status = 'Booked' WHERE
Train_Number = ? AND Ticket_Type = 'Premium' AND Status = 'WaitL' AND Passanger_ssn
IN (SELECT Passanger_ssn FROM Booked WHERE Train_Number = ? AND Ticket_Type =
'Premium' AND Status = 'WaitL' ORDER BY Train_Number LIMIT
1)",(TrainNumber.get(),TrainNumber.get(),))
```

```
    else:

        cancelTicket_cur.execute("UPDATE Booked SET Status = 'Booked' WHERE

Train_Number = ? AND Ticket_Type = 'General' AND Status = 'WaitL' AND Passanger_ssn IN

(SELECT Passanger_ssn FROM Booked WHERE Train_Number = ? AND Ticket_Type =

'General' AND Status = 'WaitL' ORDER BY Train_Number LIMIT 1
```

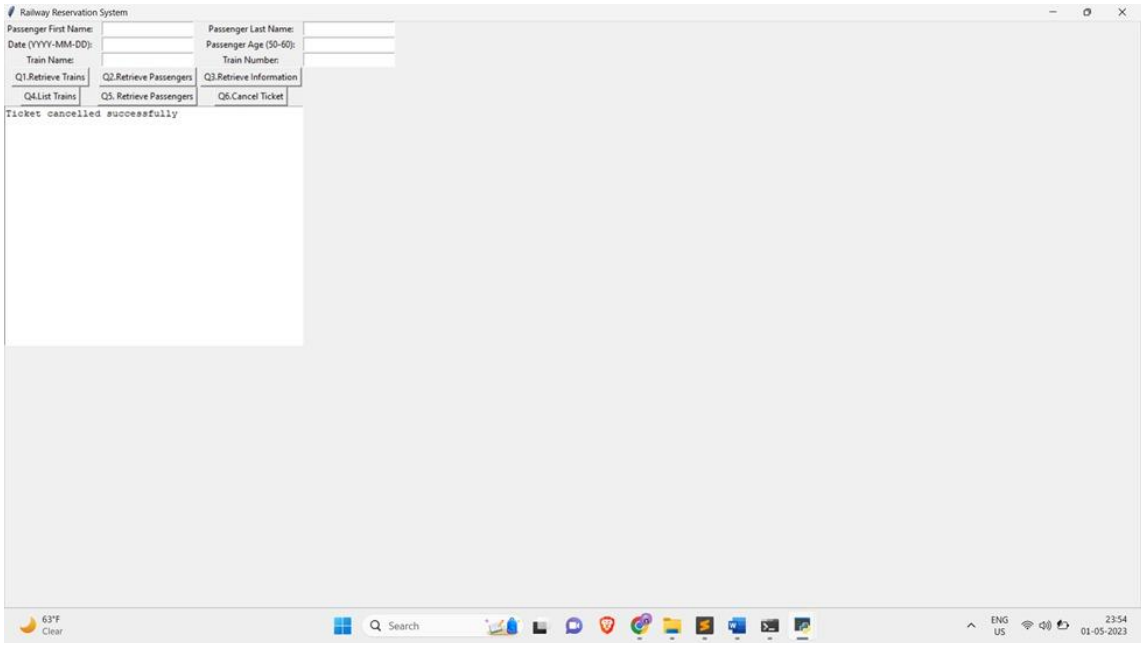
Explanation:

The first SQL statement selects all rows from the 'Booked' and 'passenger' tables where the passenger's SSN and train number match the values provided in the input parameters, and the booking status is 'Booked'. This is used to verify that a booking exists for the given passenger and train, and that it can be canceled.

The second SQL statement deletes the row from the 'Booked' table where the passenger's SSN and train number match the input parameters. This effectively cancels the booking.

The third SQL statement updates the 'Booked' table to change the status of a waitlisted passenger to 'Booked' if a premium or general ticket becomes available for the same train. The specific SQL statement that is executed depends on the type of ticket that was booked (premium or general).

Overall, the SQL statements in this script seem appropriate for canceling a ticket and updating the booking status of waitlisted passengers, provided that the input parameters are validated and sanitized to prevent SQL injection attacks.



```
Ticket cancelled successfully
```

Team Contribution:

Database Development	Apeksha and Ramya
GUI Development	Apeksha and Ramya
Query Development	Apeksha and Ramya
Testing	Apeksha
Documentation	Ramya