Final Project: Report

Destiny Jenkins, Tom Nguyen & Ryan Gray
University of South Florida
Database Design
April 24, 2019

Introduction

For our final project, we have chosen to design and implement a basic airline reservation system. We utilize a terminal-based user interface written in Python and a MySQL database.

Tables

Our database is named *airlines*. Within, we have created four tables. The *Flights* table stores relevant information about each available flight that a user can purchase a ticket for. Attributes include departure date, flight number, departure city, arrival city, flight duration, etc. Our next table, *Customer*, stores required passenger data. Some attributes include first and last name, date of birth, email, and an associated personal name record, PNR. This lends itself to our next table, *Reservation*. This stores data specific to a customer's purchase of a ticket for a flight. Attributes include flight ID, date of departure, ticket number, class of service, and seat. Our final table is named *Account*, which is designed to give the user the option to create an account (thus auto-populating the *Customer* table if the user is purchasing a ticket for themselves) or to purchase a ticket without an account, in which case the user will have to fill in all of the relevant *Customer* information. These tables are created in our file sqlscript.sql. Full schemas for each table are shown below:

```
/*Make flights table*/
CREATE TABLE airline.Flights (
    Flight_ID CHAR(6) PRIMARY KEY NOT NULL,
    dt DATETIME (0),
    origin VARCHAR(255),
    destination VARCHAR(255),
    duration INTEGER,
    seats_avail INTEGER,
    seat_cap INTEGER,
    price INTEGER
);
```

```
/*Make Accounts table*/
CREATE TABLE airline.Accounts (
    username VARCHAR(50) PRIMARY KEY,
    password VARCHAR(255),
    fname VARCHAR(50),
    mname VARCHAR(50),
    lname VARCHAR(50),
    dob DATE,
    gender CHAR(1),
    address VARCHAR(255),
    city VARCHAR(255),
    state CHAR(2),
    zipcode INTEGER(5),
    email VARCHAR(255),
    phone BIGINT (11),
    paymentcard BIGINT(16),
    cvc INTEGER(3),
    expiredate DATE,
    UNIQUE KEY(paymentcard)
);
```

```sql
/*make Passengers table*/
CREATE TABLE airline.Passengers (
    Ticket_ID INTEGER(9) PRIMARY KEY,
    PNR CHAR(6),
    username VARCHAR(50),
    fname VARCHAR(50),
    mname VARCHAR(50),
    lname VARCHAR(50),
    dob DATE,
    gender CHAR(1),
    address VARCHAR(255),
    city VARCHAR(255),
    state CHAR(2),
    zipcode INTEGER(5),
    email VARCHAR(255),
    phone BIGINT(11)
);
```

```sql
/*Reservation table*/
CREATE TABLE airline.Reservation (
    PNR CHAR(6) PRIMARY KEY,
    Flight_ID CHAR(6),
    purchase_date TIMESTAMP,
    paymentcard BIGINT(16),
    cvc INTEGER(3),
    expdate DATE,
    FOREIGN KEY (Flight_ID) REFERENCES
    airline.Flights(Flight_ID)
);
```

Queries

Once the program is executed, we have the main menu give a user four distinct options. The first option is *Search Flights* which will read in user input for a departure city, arrival city and date of departure. There is error-checking to ensure that departure city and arrival city are different. This search information is then passed as a parameter to our MySQL query function, *QueryFlights()*. This query simply selects each tuple from the database that matches the function parameter values. It then displays all found tuples to the user. In this output, note that all attributes of selected rows are displayed. This is done because, to book a flight, the user will need to know the *flight_ID*, the *Flights* table's primary key.

```python
#searches all flights with same city pairs, date
def QueryFlights(data):
    cursor = mydb.cursor()
    # data tuple format -> (orig, dest, date 'yyyy-mm-dd')

    cursor.execute("""
            SELECT *
            FROM airline.flights
            WHERE origin = %s AND destination = %s
            AND DATE(dt) = %s""", data)
    result = cursor.fetchall()

    print('¥n¥n')

    if len(result) == 0:
        print('There are no flights available.
            ¥nReturning to main menu')
    else:
        for row in result:
            print(row)

    print('¥n¥n')
    cursor.close()
```

Our next MySQL query function is *isFlightAvail()*. This function serves the simple purpose of verifying if there is sufficient space available on any particular flight. It takes a single argument, the *flight_ID*. First it locates the tuple by matching *flight_ID*, then checks if the *seats_avail* attribute is greater than 0. The query selects the tuple if both conditions are met. The function then returns true if the query returns a tuple, otherwise returns false.

```python
def isFlightAvail(flight_ID):

    cursor = mydb.cursor()

    cursor.execute("""
            SELECT *
            FROM airline.flights
            WHERE Flight_ID = %s
            AND seats_avail > 0""", (flight_ID,))
    results = cursor.fetchall()

    print('¥n¥n')

    if len(results) == 0:
        print('no matching flight')
        cursor.close()
        return False
    else:
        #print(results)
        cursor.close()
        return True
```

Our next query function is called *showPrice*. This function takes a single parameter, the *flight_ID*. It simply selects the tuple that matches the function parameter, *flight_ID* and returns the price.

```
def showPrice(flight_ID):

    cursor = mydb.cursor()

    cursor.execute("""
            SELECT price
            FROM airline.flights
            WHERE Flight_ID = %s""", (flight_ID,))
    results = cursor.fetchall()

    return results[0][0]
```

Our next query function is called *getRowFromReservation()*. Given a single parameter, *pnr* (personal name record), the query searches for and returns the tuple that contains the same *pnr* attribute value.

```
def getRowFromReservation(pnr):
    cursor = mydb.cursor()

    cursor.execute("""
        SELECT *
        FROM airline.Reservation
        WHERE PNR = %s""", (pnr,))
    results = cursor.fetchall()

    cursor.close()
    return results #list of tuples
```

The new query function is called *getRowFromPassengers()*. It takes a single parameter called *ticketID*. The query selects all tuples that contain a matching value on the *ticketID* attribute. The associated query within this function is very similar to the function and query above, *getRowFromReservation()*. The difference is that the primary key of the *Reservations* table is *pnr* and the primary key of the *Passengers* table is *ticketID*, so these two functions serve to extrapolate distinct rows based on the user having either their *pnr* or their *ticketID*.

```
def getRowFromPassengers(ticketID):
    cursor = mydb.cursor()

    cursor.execute("""
        SELECT *
        FROM airline.Passengers
        WHERE Ticket_ID = %s""", (ticketID,))
    results = cursor.fetchall()

    cursor.close()
    return results #list of tuples
```

In the next query function, called *addNewPassenger()*, we are creating/inserting a new tuple into the *Passengers* table. This is done by passing a multi-value data array as the sole parameter into the function call. Then the query runs the *insert* command with all the attribute values that were provided by the user.

```
def addNewPassenger(data):
    cursor = mydb.cursor()

    cursor.execute("""
        INSERT INTO airline.Passengers
        (Ticket_ID, PNR, username, fname,
        lname, mname, dob, gender, address,
        city, state, zipcode, email, phone)
        VALUES (%s, %s, %s, %s, %s, %s, %s,
        %s, %s, %s, %s, %s, %s, %s)""",
        (data))

    mydb.commit()
    cursor.close()
```

The next function is responsible for creating and inserting a new reservation into the *Reservations* table. In a very similar fashion as the above function *addNewPassenger()*, *addNewReservation()* takes a multi-value array as a single argument and writes each of these values into the corresponding attributes of a *Reservations* tuple.

```
def addNewReservation(data):
    cursor = mydb.cursor()

    cursor.execute("""
        INSERT INTO airline.Reservation
        (PNR, Flight_ID, purchase_date,
        paymentcard, cvc, expdate)
        VALUES (%s, %s, %s, %s, %s, %s)""",
        (data))

    mydb.commit()
    cursor.close()
```

The function shown below is for *EnterInfo()* function. This allows the user to input all relevant information needed to purchase an fare.

```
def EnterInfo():
    print("*****************************************")
    print("            Passenger INFO               ")
    print("*****************************************")
    firstname = input("Enter your first name: ")
    lastname = input("Enter your last name: ")
    middlename = input("Enter your middle name or none: ")
    dob = datetime.datetime.strptime(input("Enter your dob: mm/dd/yyyy: "),
    '%m/%d/%Y').date()
    gender = input("What is your gender(M/F/N)?: ")
    address = input("Enter your address (EX 1234 Beverly Hills Suite 129): ")
    city = input("City?: ")
    state = input("2-letter State (Ex. FL for Florida): ")
    zipcode = int(input("Zip code?: "))
    email = input("Email (Ex. mysql@jmail.com): ")
    phonenum = int(input("Phone: (Ex. 1239999999): "))

    if(middlename == 'none'):
        middlename = None

    return (firstname,
            lastname,
            middlename,
            dob,
            gender,
            address,
            city,
            state,
            zipcode,
            email,
            phonenum)
```

The *EnterPaymentInfo()* function does exactly what you think. It prompts the user to enter their credit card information and then uses this information to create a new reservation.

```
def EnterPaymentInfo():
    print("*****************************************")
    print("               Payment INFO              ")
    print("*****************************************")
    cc = int(input("Credit/debit card number: "))
    cvc = int(input("3-digit CVC (located on back of card): "))
    expiredate = datetime.datetime.strptime(input
    ("Expiration date (MM/YYYY): "), '%m/%Y').date()

    return (cc, cvc, expiredate)
```

Program Execution

Our final project is a collection of 3 files. *sqlscript.sql* contains the table creations and *sqlQueries.py* is a collection of functions written to execute specific MySQL commands on our database. Our database and MySQL communicate through the use of a Python connector for MySQL. Finally, we have *FinalProject.py*, a source file that contains *main()* and starts program execution.

As mentioned, program execution begins in *FinalProject.py*. We start by providing a terminal-based display menu containing selection options for the user, as shown below:

```
*****************************************
*Welcome to the Flight Reservation Kiosk!*
*****************************************
*Please make a selection:                *
*(1) Search Flights                      *
*(2) Book a Flight                       *
*(3) Check Current or Past Resrvations   *
*(4) Exit Program / Logout               *
*****************************************
```

If the user selects option 1, then the *SearchFlightsPage()* function is called, which subsequently calls *QueryFlights()* after getting the desired search flight information from the user. The program will output all available flights that meet the user's search criteria. If no flights are found, we print an appropriate "no flights found" response. If the user finds a flight they wish to purchase, it is important that they record or make note of the value in the first attribute, *flight_ID*. This information is required in order to book a ticket later on. The results of a sample flight search are shown in the screenshot below:

Once the user is ready to book a flight, they can select option 2 from the main menu. This will call the *BookFlight()* function, which will ask for the *flight_ID* from the user. A query is run to check seat availability and if there are no seats available on that specific flight, the program returns to the main menu. Otherwise, another

query is run to select the price of a ticket for that flight and is displayed to the user. If the user selects *y/Y* to purchase the ticket, they are prompted to enter both personal information and purchase method information via two function calls, *EnterInfo()* and *EnterPaymentInfo()*. Three screenshots are shown below. In sequential order, they show:

1. the program outputting ticket price upon choosing option 2
2. The interface for the user inputting their personal information
3. Collecting payment information

```
2
Book Flight chosen
Enter a flight ID: GX3522



Price of Flight:189
Would you like to buy a ticket? (Y/N): Y
```

```
****************************************
            Passenger INFO
****************************************
Enter your first name: Doug
Enter your last name: Theman
Enter your middle name or none: S
Enter your dob: mm/dd/yyyy: 03/26/1994
What is your gender(M/F/N)?: M
Enter your address (EX 1234 Beverly Hills Suite 129): 3234 Overtherandbow St Apt 153
City?: Tampa
2-letter State (Ex. FL for Florida): FL
Zip code?: 33619
Email (Ex. mysql@jmail.com): ThemanDoug@gmail.com
Phone: (Ex. 1239999999): 8134510967
```

```
****************************************
                Payment INFO
****************************************
Credit/debit card number: 4794947265842039
3-digit CVC (located on back of card): 666
Expiration date (MM/YYYY): 6/2024



Payment processed
Ticket bought
```

Once this information is provided to the user, another function is called, *buyTicket()*. The information just collected from the user is passed into this function along with the *flight_ID*. A unique *pnr* and *ticket_ID* are randomly generated, then a new tuple is added into both the *Passengers* table as well as the *Reservation* table.

If a user wants to select current or past reservations, we have incorporated this functionality into our main menu as well. Using *option 3,* we are able to retrieve existing reservations from our *Reservations* table by accessing existing accounts in our *Accounts* table. If the user already has an account established, then the program prompts for the email, followed by the password. If the pair matches perfectly, then a query is called to retrieve all existing reservations on that account. Otherwise, if the user does not have an account and wishes to continue as a guest, they can still retrieve existing reservations by typing in their PNR and name. Both cases are shown below:

```
****************************************
*Welcome to the Flight Reservation Kiosk!*
****************************************
*Please make a selection:               *
*(1) Search Flights                     *
*(2) Book a Flight                      *
*(3) Check Current or Past Resrvations  *
*(4) Exit Program / Logout              *
****************************************
Selection: 3
Email: email@email.com
Password: 123abc
PNR: HG7FS9 Flight ID123456 Purchase Date: 2019-04-23
****************************************
```

Program Functions

Below is a list of all of the functions in the program and a brief summary of what each function does:

DisplayMenu(): Displays a menu with four options of how the user can proceed with the program (Search flights, Book a Flight, Check Current or Past Reservations, Exit Program / Logout)

MakeSelection(): This function selects which option from the menu in DisplayMenu() is used based on the user's input

SearchFlights(): Takes in flight information from the user and queries the Flights table to find flights

BookFlight(): Takes in a Flight ID from the user, then either returns flight info matching that flight id, or a message saying that no flight with that id number was found. If the flight is found, the function asks the user if they would like to purchase a ticket for that flight

EnterInfo(): Allows user to enter personal information and saves it to the database

EnterPaymentInfo(): Allows user to enter payment information and saves it to the database

checkReservation(): Calls checkReservationDisplay()

checkReservationDisplay(): Displays selection menu that allows user to choose whether they want to login, create an account, or continue as a guest

selectionReservation(): If the user's selection is 3, the function calls findReservation(). If the user's selection is not 3, the function returns to the display menu

findReservation(): Finds user's reservation based on user input of pnr, first name, and last name

buyTicket(): produces a ticket for the user with a pnr, purchase date, and ticket id, then adds a new row to the passenger and reservation tables

generateNewTicketID(): generates ticket ID for buyTicket() function

generateNewPNR(): generates PNR for buyTicket() function

MyFlights(): Calls UserLogin() function, then, if login is successful, calls DisplayFlighta() function. If UserLogin() is not successful, returns to display menu

UserLogin(): This function allows the user to login with an email address and password.

DisplayFlights(): Displays flights that match user's card number

quitP(): exits program

Conclusion

During this project's development, we learned about the development process, as well as expanded our knowledge of databases. Initially, we had planned to create a web-based program using HTML, CSS, and PHP. While we were able to create webpages with HTML and CSS, we struggled with the PHP element of the project because none of us had ever worked with PHP before. The learning process was taking much longer than we originally anticipated, so we abandoned the web-based program idea and decided to make a terminal-based

program instead. As a result of working on this project we were able to apply concepts that we learned in class, such as database design and queries, to a real-life project. We were also able to learn new skills such as creating a database using MySQL Workbench, incorporating sql in Python code using the Python connector for MySQL, and creating and running a local server. These skills could be very useful in future endeavours.