CS 4400 Phase III - Team Number: 31

Anushka Gupta - agupta@gatech.edu - Section A - agupta400
Shantanu Mantri - smantri7@gatech.edu - Section C - smantri7
Shubham Shah - shubham.shah@gatech.edu - Section C - sshah383
Ishan Waykul - iwaykul3@gatech.edu - Section C - iwaykul3

SQL:

1) Login: The login screen of the GUI

a)Login Check: Checks if username and password are valid for a customer in the database
//username,password is compared with user input in python code//
"SELECT username, password FROM customer"

b)Login Check Manager: Checks if username and password are valid for a manager in the database
//username,password is compared with the user input in python code//
"SELECT username, password FROM manager"

2)Register Customer: Registers a new customers and adds him/her to the database

a) checkUsernameEmail: Checks to make sure both the username and email are both valid by getting all users and their emails who are already registered
"SELECT username, email FROM customer"

b)Add Registered User: Adds the user to the customer table
//Assume username, password, and email are the values that the user entered
INSERT INTO customer (username, password, email) VALUES (%s, %s, %s)""", (username, password, email))

3) Search Train: Searches for a train schedule based on a train number

a)Train Check: Gets all train numbers and checks if train number is valid
//train_number is compared with user input in python code//

"SELECT train_number FROM train_route"

b)SearchTrain: Gets the schedule of a particular train route
//assume trainNum is the train number from the user input//
"SELECT * FROM stop WHERE train_number =" + str(trainNum) + "
ORDER BY arrivesTime ASC"

4) Make Reservation: Allows the user to make a reservation and add tickets

a)Station Info: Gets all the information about stations for the drop down menu
"SELECT * FROM station"

b)Find Trains: Finds a train route that a user can get a ticket for
//Assume departStation and arrivalStation are the two stations the user chose
"SELECT stop1.train_number, stop1.departsTime, stop2.arrivesTime, first_class_price, second_class_price FROM stop AS stop1 INNER JOIN stop as stop2 INNER JOIN train_route WHERE stop1.departsTime < stop2.arrivesTime AND stop1.name = " + "'" + departStation + "'" + "
AND stop2.name = "+ "'" + arrivalStation + "'" + "
AND stop1.train_number = stop2.train_number
AND stop1.train_number = train_route.train_number"

c)Get student: Check if the customer is a student
//Assume %s holds the string of the customer to check
SELECT isStudent FROM customer WHERE username = %s

d)Get card num: Gets the card number from the payment information
//Assume %s holds the string of the customer to check
SELECT card_number FROM payment_information WHERE username = %s

e) Add Payment info: Adds new payment information, assume %s holds the card entry, uesrname,CN, CVV, and expiration date
INSERT INTO payment_information (card_number, username, name_on_card, CVV, expiration_date) VALUES(%s,%s,%s,%s,%s)""" , (str(self.leftCardEntryVal),
str(usernameSpecial) ,
self.CNEntryVal,
str(self.CVVEntryVal),
str(self.exDateVal)))

f) Check payment info: Makes sure customer does not delete a card that is used for a future reservation
//Assume %s holds the username of the user
SELECT card_num FROM reservation NATURAL JOIN ticket WHERE ticket.departure_date >= CURDATE() AND reservation.isCancelled = 0 AND reservation.username = %s

g) Gets the card number, departure date to make sure it is valid based on the departure date
//Assume %s is the departure_date
SELECT card_num, departure_date FROM reservation NATURAL JOIN ticket WHERE departure_date > %s

h) Deletes a card
//Assume usernameSpecial is the user's username, and cardNumEntry is the number of the card to delete

DELETE FROM payment_information WHERE card_number = '{}' AND username = '{}'".format(self.cardNumEntry.get(), usernameSpecial)

i) Inserts a new reservation into the table
//Assume loginEntry is the user's username, cardVar is the card number chosen, and total cost is the cost of the reservation
INSERT INTO reservation (username, card_num, total_cost) VALUES ('" + self.loginEntry.get() + "', '" + self.cardVar.get() + "', '" + str(self.totalCost) + "')"

j) Selects the most recent RID
SELECT MAX(RID) FROM reservation

k) Inserts a new ticket into the ticket table of the database
//Assume rid is reservation id, ticket[7,6,8,10] are parts of username, departure/arrival, departure date
INSERT INTO ticket VALUES (%s, %s, %s, %s, %s, %s, %s, %s)", (str(rid), ticket[7], str(classIns), str(ticket[6]), ticket[8], str(ticket[0]), DepSt, ArrSt

5) Update Reservation: Allows the user to update a reservation by changing the date

a) Find Reservation: Locates the specific user's tickets
//Assume resIDEntry is the Reservation ID the user inputted
//Assume loginEntry is the user's username, so we can check if that particular reservation belongs to the user
"SELECT * FROM reservation NATURAL JOIN ticket NATURAL JOIN train_route WHERE reservation.rid= resIDEntry
AND reservation.username= loginEntry

b)Get stops: Gets the stops of the train to update

//Assume record[0] is the train number
//Assume record[10] is the departure stop name
//Assume record[11] is the arrival stop name
"SELECT * FROM stop WHERE stop.train_number= record[0] AND
(stop.name= + record[10] + OR stop.name= " + record[11]
ORDER BY `stop`.`departsTime` ASC"

c)On Update: Updates the reservation by placing the new date value in the database
//Assume changedDate is the newDate in format YYYY-MM-DD
//Assume resIDEntry is the Reservation ID the user entered
//Assume recordList[i][0] contains the train number that is on a particular ticket to update
"UPDATE `cs4400_Team_31`.`ticket` SET `departure_date` =
changedDate.strftime('%Y-%m-%d')
WHERE `ticket`.`rid` = resIDEntry
AND `ticket`.`train_number` =" +"'" + self.recordList[i][0]

d)Total Cost: Gets the total cost, and then changes it to fulfill the update reservation cost
//Assume resIDEntry is the reservaton ID the user inputted
//Assume recordList[i][0] is the train number of the particular ticket to update
SELECT total_cost FROM reservation NATURAL JOIN ticket WHERE
reservation.rid = str(self.resIDEntry.get())
AND ticket.train_number = recordList[i][0]

e)Update Cost: Updates the total cost in Reservation table after editing it as per the rules
//Assume priceEntry is the new Price of the reservation
//Assume resIDEntry is the user's reservation ID for the reservation to-be-updated

"UPDATE `cs4400_Team_31`.`reservation` SET `total_cost` = priceEntry WHERE `reservation`.`RID`= self.resIDEntry

6) Cancel Reservation

//Checks validity of RID self.loginEntry.get() is the username the user used to login

a) SELECT RID, isCancelled FROM reservation WHERE username = self.loginEntry.get()

//Gets all the ticket information where self.cancelResIDInput.get() is the RID that user wants to cancel

b) SELECT * FROM ticket NATURAL JOIN train_route NATURAL JOIN reservation WHERE RID = self.cancelResIDInput.get()

//Gets the departure time and arrival time for the specified stations and trains where tNum is the train number, deptFrom is the depature station and arriveAt is the arrival station

c) SELECT * FROM stop WHERE stop.train_number = tNum AND (stop.name= deptFrom OR stop.name= arriveAt) ORDER BY `stop`.`departsTime` ASC

//Updates the reservation specified by the RID so that canceled reservation has an updated cost and isCancelled attribute is set to cancelled(1)

d) UPDATE reservation SET isCancelled = 1, total_cost = updatedCost WHERE RID = self.cancelResIDInput.get()

7) Give Review

//Checks to make sure that the customer is reviewing a valid train number

a) SELECT train_number FROM train_route

//Inserts review into the database

b) INSERT INTO review (train_number, username, comment, rating)
VALUES (%s, %s, %s, %s), (trainNum, username, comment, rating)

//Gets all the reviews for a particular train
c) SELECT * FROM review WHERE train_number = trainNumber

8) School Info

//Adds student information to a users account

a) UPDATE customer SET isStudent = 1 WHERE username = username

9) Manager Revenue Report

//Shows the revenue generated from the last 3 months

a) SELECT EXTRACT(MONTH FROM X.departure_date),SUM(total_cost)
FROM (SELECT * FROM ticket NATURAL JOIN reservation) AS X
WHERE EXTRACT(MONTH FROM X.departure_date) = 2 OR
EXTRACT(MONTH FROM X.departure_date) = 3 OR EXTRACT(MONTH
FROM X.departure_date) = 4 GROUP BY (EXTRACT(MONTH FROM
X.departure_date))

10) Manager Popular Route Report

//Shows the most popular route travelled in the last 3 months

a) SELECT * FROM (SELECT EXTRACT(MONTH FROM departure_date) AS Month,train_number,COUNT(train_number) AS popularity FROM ticket LEFT JOIN reservation ON ticket.rid = reservation.rid WHERE isCancelled =0 GROUP BY `Month`,train_number ORDER BY `popularity` DESC) AS X WHERE MONTH = 3 ORDER BY `X`.`popularity` DESC LIMIT 3

b) SELECT * FROM (SELECT EXTRACT(MONTH FROM departure_date) AS Month,train_number,COUNT(train_number) AS popularity FROM ticket LEFT JOIN reservation ON ticket.rid = reservation.rid WHERE isCancelled =0 GROUP BY `Month`,train_number ORDER BY `popularity` DESC) AS X WHERE MONTH = 4 ORDER BY `X`.`popularity` DESC LIMIT 3

c) SELECT * FROM (SELECT EXTRACT(MONTH FROM departure_date) AS Month,train_number,COUNT(train_number) AS popularity FROM ticket LEFT JOIN reservation ON ticket.rid = reservation.rid WHERE isCancelled =0 GROUP BY `Month`,train_number ORDER BY `popularity` DESC) AS X WHERE MONTH = 5 ORDER BY `X`.`popularity` DESC LIMIT 3