Nicholas Castro and Allan Lee
Travel305: CSE 305 Final Project
Professor Praveen Tripathi

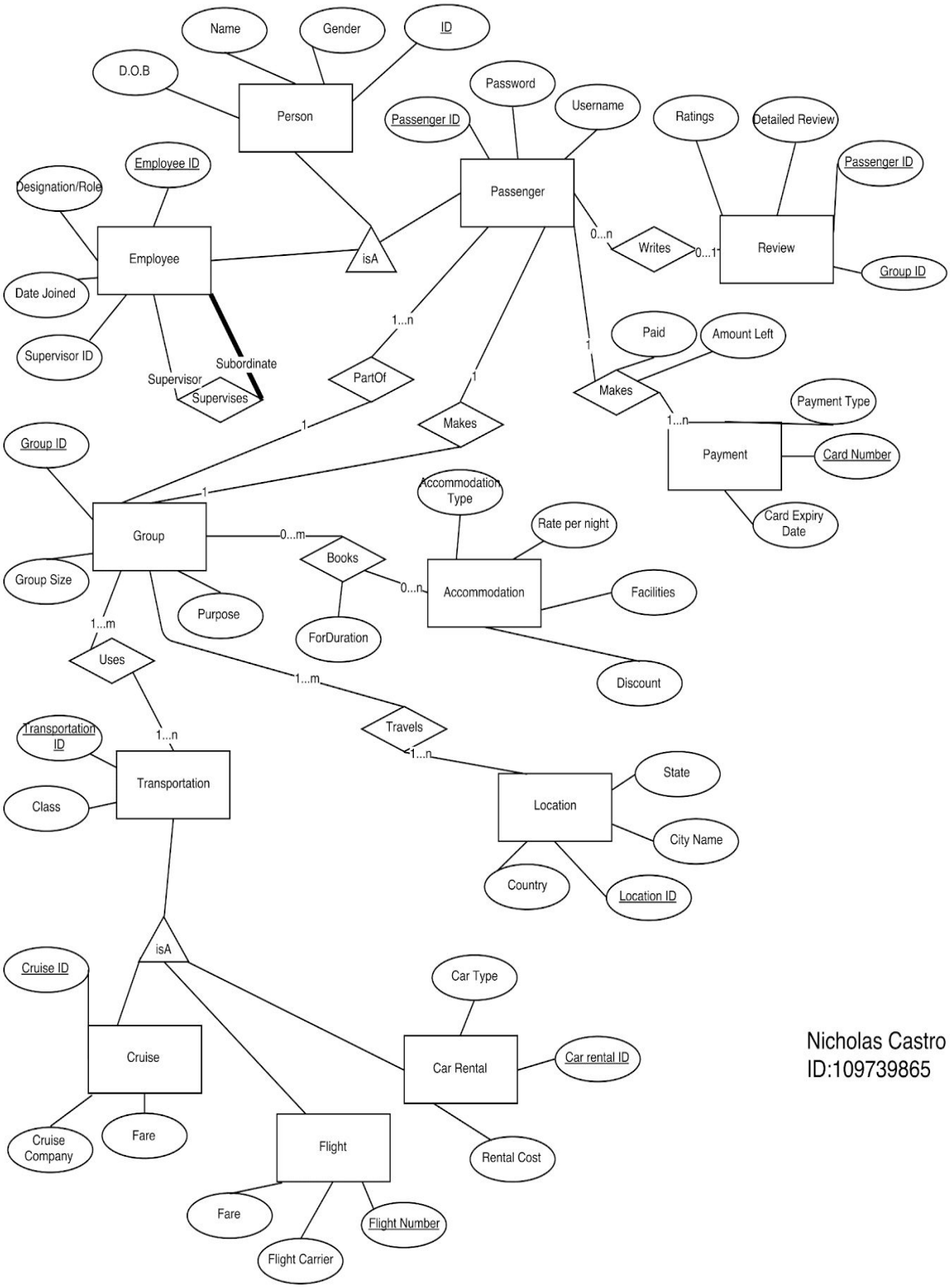
Introduction

For the final project for CSE305 we have created a travel agency website and database. We first created ER diagrams in assignment 1 to model our database transactions. Then in assignment 2 we mapped the ER diagram to SQL statements. We have made some changes to the SQL statements while implementing the transactions for our database. We used Flask, MySQL, HTML and CSS to design the backend and the frontend.

Installation and Setup

First, download our project and into a directory.
Then create a virtual environment, and activate using the command source/venv/bin/activate
Then install flask with the command pip3 install flask.
Then install flask bootstrap with the command pip3 install flask-bootstrap.
Then install PyMySQL using command pip3 install PyMySQL.
Then install Flask forms using command pip3 install Flask-WTF.
Finally export the flask app using the command export FLASK_APP=__init__.py
Make sure that you check the config file and the __init__.py file to make sure the connection to your flask server and mysql server is set up correctly.
To initialize the database run the file initDatabase.py.
Then to populate the table run the command python3 initTables.py baseTableValues.txt.
The project is now ready to be used.

ER Diagram Stage



Mapping to SQL

```
initUserTable = "CREATE TABLE Users (ID int not null auto_increment,"\
"Email varchar(100),"\
"Name varchar(30),"\
"Password varchar(20),"\
"Gender char(1),"\
"DateOfBirth date,"\
"primary key (ID));"
```

```
CREATE TABLE Passengers (
        PassengerID INTEGER,
        PaymentID INTEGER,
        PassengerName VARCHAR(255),
        Gender VARCHAR(255),
        Age INTEGER,
        PRIMARY KEY (PassengerID)
        FOREIGN KEY(PassengerID) REFERENCES PartOf(PassengerID)
        FOREIGN KEY(PaymentID) REFERENCES Makes(PaymentID)
)
CREATE TABLE Payment (
        PaymentType VARCHAR(255),
        CardNumber VARCHAR(255),
        CardExpiryDate DATE,
        PRIMARY KEY(PaymentType, CardNumber, CardExpiryDate)
)
CREATE TABLE Review (
        PassengerID INTEGER,
        Ratings INTEGER,
        Group INTEGER,
        DetailedReview VARCHAR(1000)
        PRIMARY KEY(PassengerID),
        CHECK (Ratings > 0 AND Ratings < 6)
)
CREATE TABLE Location (
        LocationID INTEGER,
        CityName VARCHAR(255),
        Country VARCHAR(255),
        State VARCHAR(255),
        PRIMARY KEY(LocationID)
)
```

```
CREATE TABLE Transportation (
        TransportationID INTEGER,
        TransportationType VARCHAR(255),
        ClassType VARCHAR(255),
        PRIMARY KEY(TransportationID)
)
CREATE TABLE Group (
        GroupID INTEGER,
        TravelID INTEGER,
        GroupSize INTEGER,
        SourceLocation INTEGER,
        DestinationLocation INTEGER,
        ModeOfTranport VARCHAR(255),
        Purpose VARCHAR(1000),
        PRIMARY KEY (GroupID)
        FOREIGN KEY(SourceLocation) REFERENCES Location(LocationID),
        FOREIGN KEY(DestinationLocation) REFERENCES Location(LocationID),
        FOREIGN KEY(GroupID) REFERENCES Books(GroupID)
        FOREIGN KEY(TravelID) REFERENCES Travel(TravelID)
)
CREATE TABLE Accommodation (
        AccommodationType VARCHAR(255),
        Facilities VARCHAR(255),
        RatePerNight INTEGER,
        Discount INTEGER,
        PRIMARY KEY(AccommodationType, Facilities)
)
CREATE TABLE Employee (
        EmployeeID INTEGER,
        Designation/Role VARCHAR(255),
        Facilities VARCHAR(255),
        SupervisorID INTEGER,
        PRIMARY KEY (EmployeeID),
        FOREIGN KEY (SupervisorID) REFERENCES (EmployeeID)
)
CREATE TABLE CarRental (
        CarRentalConfirmationID INTEGER,
        CarType VARCHAR(255),
        Rent INTEGER,
```

```
        PRIMARY KEY(CarRentalConfirmationID)
        FOREIGN KEY(CarRentalConfirmationID) REFERENCES
        Transportation(TransportationID)
)


CREATE TABLE Flight (
        FlightNumber INTEGER,
        FlightCarrier VARCHAR(255),
        SourceLocation VARCHAR(255),
        DestinationLocation VARCHAR(255),
        Class VARCHAR(255),
        Fare INTEGER,
        PRIMARY KEY(FlightNumber)
        FOREIGN KEY(FlightNumber) REFERENCES
        Transportation(TransportationID)
)
CREATE TABLE Cruise (
        CruiseNumber INTEGER,
        SourceLocation VARCHAR(255),
        DestinationLocation VARCHAR(255),
        Fare INTEGER,
        PRIMARY KEY(CruiseNumber)
        FOREIGN KEY(CruiseNumber) REFERENCES
        Transportation(TransportationID)
)
CREATE TABLE PartOf (
        PassengerID INTEGER,
        GroupID INTEGER,
        PRIMARY KEY(PassengerID),
        FOREIGN KEY(GroupID) REFERENCES Group(GroupID)
)
CREATE TABLE Books (
        GroupID INTEGER,
        ForDuration VARCHAR(255),
        AccommodationType VARCHAR(255),
        Facilities VARCHAR(255),
        PRIMARY KEY(GroupID)
        FOREIGN KEY(AccommodationType, Facilities)
```

```
        REFERENCES Accommodation(AccommodationType, Facilities)
        CHECK (NOT EXISTS(SELECT *
                                FROM Group G
                                WHERE NOT EXISTS (SELECT *
                                                        FROM Books B
                                                        WHERE G.GroupID = B.GroupID)))
)
CREATE TABLE Writes(
        PassengerID INTEGER,
        Review VARCHAR(1000),
        PRIMARY KEY(PassengerID),
        FOREIGN KEY(PassengerID) REFERENCES Passenger(PassengerID),
        FOREIGN KEY(Review) REFERENCES Review(DetailedReview)
)
CREATE TABLE Makes (
        PassengerID INTEGER,
        PaymentID INTEGER,
        Paid FLOAT,
        AmountLeft FLOAT,
        PaymentNumber INTEGER,
        CardNumber VARCHAR(255),
        CardExpiryDate DATE,
        PRIMARY KEY (PassengerID),
        FOREIGN KEY (PaymentNumber, CardNumber, CardExpiryDate) REFERENCES
        Payment(PaymentNumber, CardNumber, CardExpiryDate)
        CHECK (NOT EXISTS(SELECT *
                                FROM Passenger P
                                WHERE NOT EXISTS (SELECT *
                                                        FROM Makes M
                                                        WHERE P.PaymentID = M.PaymentID)))
)
CREATE TABLE Travels (
        TravelID INTEGER,
        SourceLocation INTEGER,
        DestinationLocation INTEGER,
        PRIMARY KEY(TravelID)
        FOREIGN KEY(SourceLocation, DestinationLocation) REFERENCES
        Location(SourceLocation, DestintationLocation)
        CHECK (NOT EXISTS(SELECT *
```

```
                    FROM Group G
                    WHERE NOT EXISTS (SELECT *
                                      FROM Travels T
                                      WHERE G.TravelID = T.TravelID)))
)
```

*Note that some tables have been adjusted in order to get functionality to work in the middle of design.

Implementation of Transactions

The project folder is divided into several sections and the main items to be concerned about is forms.py, routes.py, and templates folder. The forms.py file are the FLASK forms that we create that enables us to add in buttons and fields for users to interact with the system and the database. The routes.py file is where we define all of the methods and transactions that causes web pages to be rendered or redirected as well as make changes or retrieve information from the database. The templates folder holds HTML files and inside the HTML folder is Jinja2 code that enables the FLASK forms to work as well as to help us dynamically load in content. Here are some examples of the transactions being programmed:

Figure 1.1: Displaying location, accommodation, and transportation information

```
@app.route('/')
@app.route('/index')
def index():
    cursor.execute("SELECT * FROM Location;")
    locations = cursor.fetchmany(4)
    cursor.execute("SELECT * FROM Cruise;")
    cruises = cursor.fetchmany(4)
    cursor.execute("SELECT * FROM Accommodation;")
    accommodations = cursor.fetchmany(4)
    cursor.execute("SELECT * FROM Flight;")
    flights = cursor.fetchmany(4)
    cursor.execute("SELECT * FROM CarRental;")
    carRentals=cursor.fetchmany(4)
    if flask_login.current_user.is_authenticated:
        user_name = getName()
        return render_template('index.html', base_template = "base_loggedin.html", name = user_name, locations=locations, cruises=cruises, accommodations=accommodations,
    return render_template('index.html', base_template = "base.html", locations=locations, cruises=cruises, accommodations=accommodations, flights=flights,carRentals=carR
```

Figure 1.2: Signup and Login

```
@app.route('/signup',methods=["POST","GET"])
def signup():
    form = SignupForm()
    if form.validate_on_submit():
        name = form.name.data
        dob=form.dob.data
        email=form.email.data
        password=form.password.data
        gender=form.gender.data
        sql="INSERT INTO Users VALUES (NULL,"+"\'"+email+"\',"+"\'"+name+"\',"+"\'"+password+"\',"+'\''+gender+'\','+'\''+dob+'\')'
        cursor.execute(sql)
        connection.commit()
        return render_template('signupcomplete.html')
    return render_template('signup.html', title="Signup", form=form)
```

As you can see here, we retrieve the form's data to place into our SQL statement. The cursor executes the SQL statement which is a string in Python and we execute "connection.commit()" to commit the insertion into our database

```python
@app.route('/login',methods=['GET','POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        email=form.email.data
        password=form.password.data
        cursor.execute("SELECT * FROM Users WHERE Email = \'" + email +  "\';")
        data = cursor.fetchall()
        if(len(data) == 0):
            flash("Invalid email or password")
            return redirect(url_for('login'))
        if(data[0]["Password"] != password):
            flash("Invalid email or password")
            return redirect(url_for('login'))
        if(data[0]["Email"] == email and data[0]["Password"] == password):
            user = User(data[0]["Email"])
            flask_login.login_user(user)
            return redirect(url_for('index'))
    return render_template('login.html', title="Login", form=form, loggedin = False)
```

We retrieve the email and password from the form data and get the user associated with the email. We place the data retrieved from the SQL statement into a variable called data which is an array of dictionaries in Python. By indexing it, we can retrieve the information associated with the execution and check for correctness that will enable the user to login.

Figure 1.3: Making/Joining a Group

```python
@app.route('/creategroup', methods=['GET','POST'])
def createGroup():
    form = CreateGroup()
    username = getName()
    if form.validate_on_submit():
        email = form.email.data
        name = form.name.data
        travelID = form.travelID.data
        cursor.execute("SELECT * FROM Users WHERE Email = \'" + email + "\' AND Name = \'" + name + "\';")
        data = cursor.fetchall()
        if len(data) == 0:
            flash("Account does not exist")
            redirect(url_for("createGroup"))
        else:
            cursor.execute("SELECT * FROM `Group` WHERE TravelID = " + str(travelID) + ";")
            data = cursor.fetchall()
            if len(data) != 0:
                flash("Travel ID already exists. Enter another value")
                redirect(url_for("createGroup"))
            else:
                sql = "INSERT INTO `Group` VALUES (NULL, " + str(travelID) + ", 1, NULL, NULL, NULL, NULL, NULL, NULL, NULL);"
                cursor.execute(sql)
                connection.commit()
                sql = "SELECT * FROM Users WHERE Email = \'" + email + "\';"
                cursor.execute(sql)
                data = cursor.fetchall()
                PassengerID = data[0]["ID"]
                sql = "SELECT * FROM `Group` WHERE travelID = " + str(travelID) + ";"
                cursor.execute(sql)
                data = cursor.fetchall()
                groupID = data[0]["GroupID"]
                sql = "INSERT INTO PartOf VALUES (" + str(PassengerID) + ", " + str(groupID) + ");"
                cursor.execute(sql)
                connection.commit()
                flash("Group has been created under ID: " + str(travelID))
                redirect(url_for("createGroup"))
    return render_template('creategroup.html', name = username, form = form)
```

We use the first selection to check that the account associated with the creation of the group exists.
If it passes that test then we check to see if the entered travelID is currently valid (nobody else is using it) in the next SQL selection. If it passes that test, then we can insert a group into the Group table and insert into the PartOf table the user that just joined their group.

```
# User joins a group with a valid group ID.
def joinGroupFunction(ID, travelID):
    sql = "SELECT GroupID FROM `Group` WHERE TravelID = " + str(travelID) + ";"
    cursor.execute(sql)
    data = cursor.fetchall()
    if len(data) == 0:
        flash("That is not a valid travel ID")
        return False
    else:
        sql = "INSERT INTO PartOf VALUES (" + str(ID) + ", " + str(data[0]["GroupID"]) + ");"
        cursor.execute(sql)
        connection.commit()
        sql = "SELECT GroupSize FROM `Group` WHERE GroupID = " + str(data[0]["GroupID"]) + ";"
        cursor.execute(sql)
        group_data = cursor.fetchall()
        group_size = group_data[0]["GroupSize"]
        group_size = int(group_size) + 1
        sql = "UPDATE `Group` SET GroupSize = " + str(group_size) + " WHERE GroupID = " + str(data[0]["GroupID"]) + ";"
        cursor.execute(sql)
        connection.commit()
        return True

# User is deleted from the specific group in the group table.
def leaveGroupFunction(ID):
    sql = "SELECT PassengerID FROM PartOf WHERE PassengerID = " + str(ID) + ";"
    cursor.execute(sql)
    data = cursor.fetchall()
    if len(data) == 0:
        flash("You are not in a group!")
        return False
    else:
        sql = "SELECT GroupID FROM PartOf WHERE PassengerID = " + str(ID) + ";"
        cursor.execute(sql)
        group_id = cursor.fetchall()
        sql = "SELECT GroupSize FROM `Group` WHERE GroupID = " + str(group_id[0]["GroupID"]) + ";"
        cursor.execute(sql)
        group_data = cursor.fetchall()
        group_size = group_data[0]["GroupSize"]
        group_size = int(group_size) - 1
        sql = "UPDATE `Group` SET GroupSize = " + str(group_size) + " WHERE GroupID = " + str(group_id[0]["GroupID"]) + ";"
        cursor.execute(sql)
        connection.commit()
        sql = "DELETE FROM PartOf WHERE PassengerID = " + str(ID) + ";"
        cursor.execute(sql)
        connection.commit()
        return True
```

Some helper functions that we use…

```python
@app.route('/joingroup', methods=['GET','POST'])
def joinGroup():
    form = JoinGroup()
    username = getName()
    email = flask_login.current_user.get_id()
    sql = "SELECT ID FROM Users WHERE Email = \'" + email + "\';"
    cursor.execute(sql)
    data = cursor.fetchall()
    ID = data[0]["ID"]
    sql = "SELECT * FROM PartOf WHERE PassengerID = " + str(ID) + ";"
    cursor.execute(sql)
    data = cursor.fetchall()
    sql = "SELECT * FROM `Group`;"
    cursor.execute(sql)
    groupData = cursor.fetchall()
    if len(data) == 0:
        travelID = None
        group_status = False
    else:
        groupID = data[0]["GroupID"]
        sql = "SELECT * FROM `Group` WHERE GroupID = " + str(groupID) + ";"
        cursor.execute(sql)
        data = cursor.fetchall()
        travelID = data[0]["TravelID"]
        group_status = True
    if form.is_submitted():
        if 'submit' in request.form:
            desired_travel_id = form.travelID.data
            if travelID != None:
                flash("You are already in a group! Leave your group first.")
                redirect_option = False
            elif desired_travel_id == None:
                flash("That is not a valid travel ID")
                redirect_option = False
            else:
                redirect_option = joinGroupFunction(ID, desired_travel_id)
        else:
            redirect_option = leaveGroupFunction(ID)
        if redirect_option:
            return redirect(url_for('index'))
    return render_template('joingroup.html', name = username, form = form, groups =
```

The first two selections in this function helps us to check if the current user is in a group. It will update the HTML page accordingly if they are. The third selection is to help us display all Groups currently in the database. The if-else statement is where we check if the user is in a group using the information from the first two selections. The following if statement checks to see if the form is submitted. Inside is an if-else statement that checks if the button you pressed is the “submit’ or “leave group” button. If you pressed submit, it will do some checks to see if you are already in a group or if you tried to join a group with an invalid travel ID. If it passes the checks then it will run the joinGroupFunction method. If you selected the other option, it will run the leaveGroupFunction method.

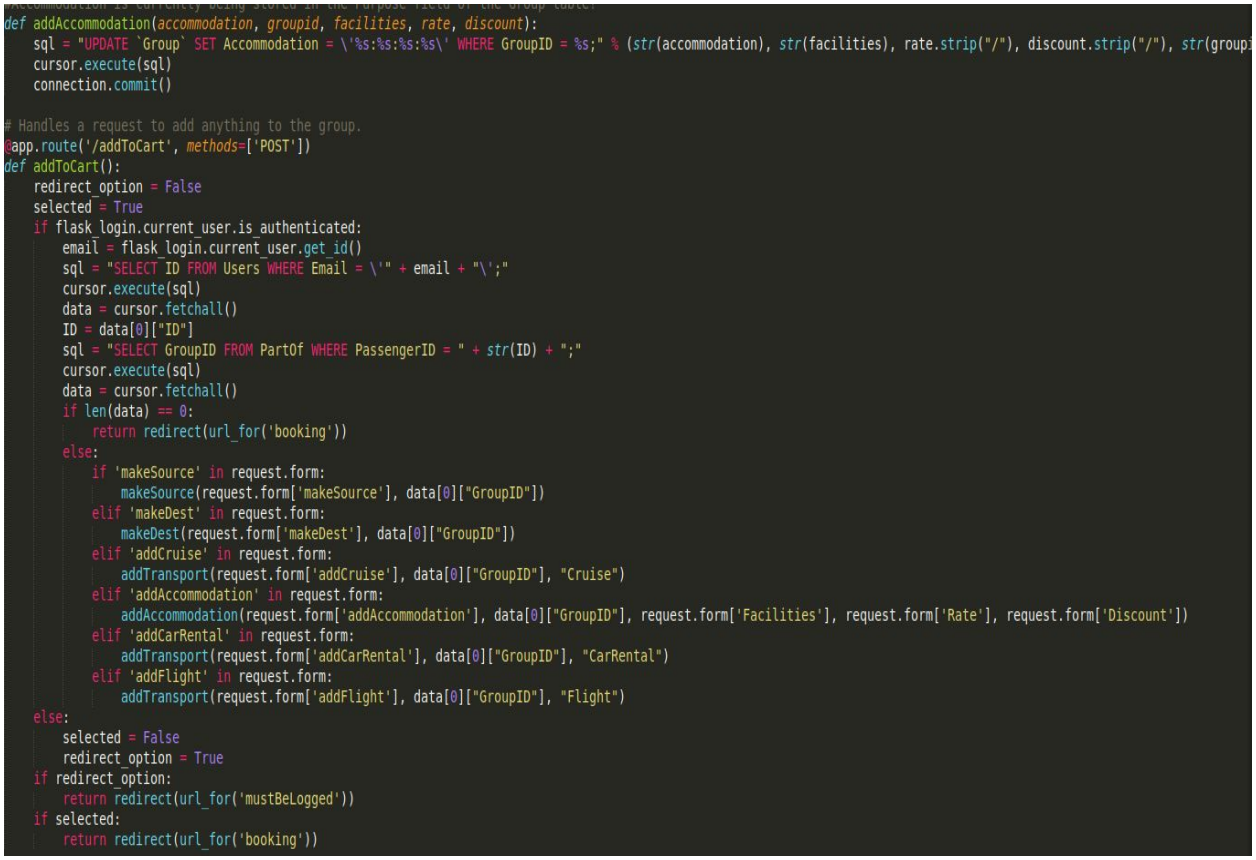
Figure 1.4: Booking a Trip

```
def makeSource(ID, groupID):
    sql = "UPDATE `Group` SET SourceLocation = " + str(ID) + " WHERE GroupID = " + str(groupID) + ";"
    cursor.execute(sql)
    connection.commit()
    sql = "SELECT CityName FROM Location WHERE LocationID = " + str(ID) + ";"
    cursor.execute(sql)
    cityName = cursor.fetchall()
    cityName = cityName[0]["CityName"]
    sql = "UPDATE `Group` SET SrcName = \'" + cityName + "\' WHERE GroupID = " + str(groupID) + ";"
    cursor.execute(sql)
    connection.commit()

# User picks a destination location for their groups trip and the group table is updated.
def makeDest(ID, groupID):
    sql = "UPDATE `Group` SET DestinationLocation = " + str(ID) + " WHERE GroupID = " + str(groupID) + ";"
    cursor.execute(sql)
    connection.commit()
    sql = "SELECT CityName FROM Location WHERE LocationID = " + str(ID) + ";"
    cursor.execute(sql)
    cityName = cursor.fetchall()
    cityName = cityName[0]["CityName"]
    sql = "UPDATE `Group` SET DestName = \'" + cityName + "\' WHERE GroupID = " + str(groupID) + ";"
    cursor.execute(sql)
    connection.commit()

# User can add any form of transportation to their trip.
def addTransport(ID, groupID, travelType):
    tid =str(ID)
    print("THE ID IS: " + tid, file=sys.stdout)
    print(travelType, file=sys.stdout)
    sql = "SELECT TransportationType FROM Transportation WHERE TransportationID = " + str(ID) + ";"
    cursor.execute(sql)
    transport = cursor.fetchall()
    transport = transport[0]["TransportationType"]
    sql = "UPDATE `Group` SET ModeOfTransport = \'" + transport + "\' WHERE GroupID = " + str(groupID) + ";"
    cursor.execute(sql)
    cost = ""
    if travelType=="Flight":
        sql = "SELECT Fare FROM Flight WHERE FlightNumber = " + str(ID) + ";"
        cost="Fare"
    elif travelType=="Cruise":
        sql = "SELECT Fare FROM Cruise WHERE CruiseNumber = " + str(ID) + ";"
        cost ="Fare"
    elif travelType=="CarRental":
        sql = "SELECT Rent FROM CarRental WHERE CarRentalConfirmationID = " + str(ID) + ";"
        cost="Rent"
    cursor.execute(sql)
    transportCost = cursor.fetchone()[cost]
    print(transportCost, file=sys.stdout)
    sql = "UPDATE `Group` SET TransportCost = \'" + str(transportCost) + "\' WHERE GroupID = " + str(groupID) + ";"
    cursor.execute(sql)
    connection.commit()
```

Helper functions which are self explanatory. We simply update the appropriate sections of the Group table. makeSource updates the source location, makeDest updates the destination location, and addTransport checks to see what mode of transport we are using and updates the ModeOfTransport attribute to the one we picked.



```
#Accommodation is currently being stored in the Purpose field of the Group table.
def addAccommodation(accommodation, groupid, facilities, rate, discount):
    sql = "UPDATE `Group` SET Accommodation = \'%s:%s:%s:%s\' WHERE GroupID = %s;" % (str(accommodation), str(facilities), rate.strip("/"), discount.strip("/"), str(groupi
    cursor.execute(sql)
    connection.commit()

# Handles a request to add anything to the group.
@app.route('/addToCart', methods=['POST'])
def addToCart():
    redirect_option = False
    selected = True
    if flask_login.current_user.is_authenticated:
        email = flask_login.current_user.get_id()
        sql = "SELECT ID FROM Users WHERE Email = \'" + email + "\';"
        cursor.execute(sql)
        data = cursor.fetchall()
        ID = data[0]["ID"]
        sql = "SELECT GroupID FROM PartOf WHERE PassengerID = " + str(ID) + ";"
        cursor.execute(sql)
        data = cursor.fetchall()
        if len(data) == 0:
            return redirect(url_for('booking'))
        else:
            if 'makeSource' in request.form:
                makeSource(request.form['makeSource'], data[0]["GroupID"])
            elif 'makeDest' in request.form:
                makeDest(request.form['makeDest'], data[0]["GroupID"])
            elif 'addCruise' in request.form:
                addTransport(request.form['addCruise'], data[0]["GroupID"], "Cruise")
            elif 'addAccommodation' in request.form:
                addAccommodation(request.form['addAccommodation'], data[0]["GroupID"], request.form['Facilities'], request.form['Rate'], request.form['Discount'])
            elif 'addCarRental' in request.form:
                addTransport(request.form['addCarRental'], data[0]["GroupID"], "CarRental")
            elif 'addFlight' in request.form:
                addTransport(request.form['addFlight'], data[0]["GroupID"], "Flight")
    else:
        selected = False
        redirect_option = True
    if redirect_option:
        return redirect(url_for('mustBeLogged'))
    if selected:
        return redirect(url_for('booking'))
```

addAccommodation updates the accommodation associated with the group. addToCart is the intermediate between the helper functions and the buttons on the HTML page. It checks to see what kind of button the user clicked on and chooses the appropriate helper function to use and update the Group table.

Figure 1.5: Displaying cost of trip

```
@app.route('/booking', methods=['GET','POST'])
def booking():
    form = Book()
    user_name = getName()
    email = flask_login.current_user.get_id()
    sql = "SELECT ID FROM Users WHERE Email = \'" + email + "\';"
    cursor.execute(sql)
    data = cursor.fetchall()
    ID = data[0]["ID"]
    sql = "SELECT GroupID FROM PartOf WHERE PassengerID = " + str(ID) + ";"
    cursor.execute(sql)
    data = cursor.fetchall()
    print(data,file=sys.stdout)
    if form.is_submitted():
        sql = "SELECT * FROM `Group` WHERE GroupID = \'%s\';" % (str(ID))
        cursor.execute(sql)
        items = cursor.fetchall()
        f = False
        for key, val in items[0].items():
            if val == None:
                f = True
                break
        if f or form.duration.data == None:
            flash("You have not selected enough information to book this trip!")
        else:
            v = booking_parse_accommodation(items[0]["Accommodation"])
            sql = "SELECT TransportCost FROM `Group` WHERE GroupID = \'%s\';" % (str(ID))
            cursor.execute(sql)
            transport_cost = cursor.fetchone()["TransportCost"]
            print(transport_cost,file=sys.stdout)
            accommodation_cost = v[2]
            discount = v[3]
            total_cost = (int(form.duration.data) * int(accommodation_cost)) - int(discount) + int(transport_cost)
            total_accommodation_cost = (int(form.duration.data)*int(accommodation_cost))
            print(total_cost, file=sys.stdout)
            sql = "INSERT INTO Books VALUES (%s,%s,\'%s\',\'%s\',%s);" % (str(items[0]["GroupID"]), str(form.duration.data), v[0], v[1], str(total_cost))
            cursor.execute(sql)
            connection.commit()
            return render_template("bookedtrips.html",base_template="base_loggedin.html",accommodation_cost=accommodation_cost,
                discount=discount, transport_cost=transport_cost,total_cost=total_cost,total_accommodation_cost=total_accommodation_cost, duration=str(for
    if len(data) == 0:
        groupInfo = ()
        acc = None
        val = True
    else:
        groupID = data[0]["GroupID"]
        sql = "SELECT * FROM `Group` WHERE GroupID = " + str(groupID) + ";"
        cursor.execute(sql)
        groupInfo = cursor.fetchall()
        acc = booking_parse_accommodation(groupInfo[0]['Accommodation'])
        val = False
    return render_template('booking.html', base_template = "base_loggedin.html", name = user_name, group = groupInfo, val = val, acc = acc, form = form)
```

The booking function checks to see if all of the fields necessary for a group trip, like accommodation, mode of transport, source and destination location and trip duration are filled out. If they are all filled out we book the trip. We calculate the accommodation cost, the discount on the accommodation, the duration of the trip and the cost for the mode of transport. We then send all of that information along with the total cost to the bookedtrip html page. This then displays a receipt for a user.

Transactions Supported

1. Users can signup for an account and login to their account.
2. Accommodation options are displayed with their locations, their price and their facilities.
3. Travel options are displayed with their source and destination locations and their fare.
4. A user can create and join groups.

5. A user can add locations, accommodations, and a mode of travel to their group trip.
6. Once a group decides on all of the options for their trip, they can book the trip for an amount of days and they get an itemized receipt.

Group Member Contribution

Allan Lee: Database Initialization and Scripting, Backend using flask, powerpoint and documentation.
Nicholas Castro: Backend using flask, Frontend using bootstrap, HTML and CSS, and documentation.