CS 1555/2055 – DATABASE MANAGEMENT SYSTEMS (FALL 2020)
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF PITTSBURGH

Project: Pitt Tours

Release: Oct 14, 2020

Due: Milestone 1 / HW 5 - Oct 28, 2020 @ 8:00pm
Milestone 2 - Nov 24, 2020 @8:00pm

Project Goal

The goal of your term project is to implement a flight reservation system for *Pitt Tours*, the imaginary multi-billion travel agency company of the University of Pittsburgh. The core of such a system is a database system and a set of triggers, procedures/functions and ACID transactions. The secondary goal is to learn how to work as a member of a team, which designs and develops a relatively large, real database application. The first project milestone, which was the assignment HW5, you were given the descriptions and schemas of the database on which the system is based and implemented a number of core triggers, functions and stored procedures. In this part of the project you will develop the "full" flight reservation system using Java, PostgreSQL and JDBC.

Triggers

In HW5, you developed two triggers: `planeUpgrade`, and `cancelReservation`. In this phase you are asked to develop three additional triggers. You should feel free to develop more triggers to make your application more robust and efficient.

- You should create a trigger, called `planeDowngrade`, that changes the plane (type) of a flight to an immediately smaller capacity plane (type) owned by the airline, if it exists, when a reservation is deleted on that flight, making the larger capacity plane unnecessary. You can reuse the code from the sample solution.
- You should create a trigger, called `adjustTicket`, that adjusts the cost of a reservation when the price of one of its legs changes before the ticket is issued.
- You should write a trigger, called `frequentFlyer`, that adjusts the frequent_miles program of a customer when the customer buys a ticket. The customer is automatically enrolled in the airline's frequent_miles program with which the customer has traveled most legs. In the case of a tie, the airline that has collected the most fare from the customer is selected. A second tie is resolved in favor of the frequent_miles program of the airline whose sale fires the trigger, if this airline is part of the tie; otherwise the second tie is resolved as follows. The airline currently in the customer record is selected if it is part of the tie, else an airline among the tie is randomly selected.

Milestone 2: Bringing it all together

For this milestone, you need to design two easy-to-use interfaces for two different user groups: *administrators* and *customers*. A simple text menu with different options is sufficient enough for this project. You may design nice graphic interfaces, but it is not required and it carries *NO bonus* points. A good design may be a two level menu with the lower level one providing different options for different users, depending on whether the user is an administrator or a customer.

Each menu contains a set of tasks as described below. In implementing these tasks, you can use the sample solution (database schema and sample SQL functions, stored procedures and triggers) of HW5

as a starting point. You may need to modify them in developing a robust JAVA app. You can create temporary tables and (materialized) views if needed, but you should not change the table schemas without permission. You are also expected to follow the best practices of program decomposition, using (stored) functions, (stored) procedures, and views as appropriate.

Finally, you are also expected to create a JAVA driver program to demonstrate the correctness of your Pitt Tours back-end by calling all of the tasks below. (Hint: You may want to develop the driver program as you develop the functions as a way to test them.)

Administrator Interface

1: Erase the database
2: Load airline information
3: Load schedule information
4: Load pricing information
5: Load plane information
6: Generate passenger manifest for specific flight on given day
7: Update the current timestamp

Note: You should process the INSERT statements of each table as a single transaction. For instance, all INSERT statements for inserting the sample data of Airline information should be processed in a single transaction while all INSERT statements for Planes should occur in a separate transaction from Airline information.

Task #1: Erase the database
Ask the user to verify deletion of all the data.
Simply delete all the tuples of all the tables in the database.

Task #2: Load airline information
Ask the user to supply the filename where the airline information is stored.
Load the information from the specified file into the appropriate table(s).

Here is an example of such an input file:

```
1 United Airlines    UAL 1931
2 All Nippon Airways ANA 1952
3 Delta Air Lines    DAL 1924
```

Task #3: Load schedule information
Ask the user to supply the filename where the schedule information is stored.
Load the information from the specified file into the appropriate table(s).
Here is an example of such an input file:

```
153 1 A320 PIT  JFK  1000 1120 SMTWTFS
154 3 B737 JFK  DCA  1230 1320 S-TW-FS
552 3 E145 PIT  DCA  1100 1150 SM-WT-S
```

Task #4: Load pricing information
Ask the user to choose between L (Load pricing information) and C (change the price of an existing flight).

If the user chooses C, then ask the user to supply the departure_city, arrival_city, high_price and low_price. Your program needs to update the prices for the flight specified by the departure_city and arrival_city that the user enters.

If the user chooses L, then ask the user to supply the filename where the pricing information is stored. Load the information from the specified file into the appropriate table(s).

Here is an example of such an input file:

```
PIT JFK 1 250 120
JFK PIT 3 250 120
JFK DCA 3 220 100
DCA JFK 3 210  90
PIT DCA 2 200 150
DCA PIT 1 200 150
```

Task #5: Load plane information
Ask the user to supply the filename where the plane information is stored.
Load the information from the specified file into the appropriate table(s).

Here is an example of such an input file:

```
B737 Boeing 125 09/09/2018 2006 1
A320 Airbus 155 10/01/2020 2011 1
E145 Embraer 50 06/15/2019 2018 2
B737 Boeing 125 17/08/2018 2007 2
```

Task #6: Generate passenger manifest for specific flight on given day
Ask the user to supply the flight number and the date.
Search the database to locate those passengers who bought tickets for the given flight and the given date. Print the passenger list (salutation, first name, last name).

Task #7: Update the current timestamp
Ask the user to supply a date and time to be set as the current timestamp (c_timestamp) in OurTimestamp table.

Customer Interface

1: Add customer
2: Show customer info, given customer name
3: Find price for flights between two cities
4: Find all routes between two cities
5: Find all routes between two cities of a given airline
6: Find all routes with available seats between two cities on a given date
7: Add reservation
8: Delete reservation
9: Show reservation info, given reservation number
10: Buy ticket from existing reservation
11: Find the top-k customers for each airline
12: Find the top-k traveled customers for each airline
13: Rank the airlines based on customer satisfaction

Task #1: Add customer

Ask the user to supply all the necessary fields for the new customer: *salutation* (Mr/Mrs/Ms), *first name, last name, address (street, city, state), phone number, email address, credit card number, credit card expiration date, frequent_miles*. Your program must print the appropriate prompts so that the user supplies the information one field at a time.

Produce an error message if a customer with the same first and last name already exists. Assign a unique ID (i.e., cid) for the new user. Insert all the supplied information (including cid) into the database. Display the unique ID as a confirmation of successfully adding the new customer in the database

Task #2: Show customer info, given customer name

Ask the user to supply the customer name.

Query the database and print all the information stored for the customer (do not display the information on reservations).

Task #3: Find price for flights between two cities

Ask the user to supply the two cities (city A and city B).

Print the high and low prices for a one-way ticket from city A to city B, the prices for a one-way ticket from city B to city A, and the prices for a round-trip ticket between city A and city B. A connecting city is visited only once in a trip (i.e., no loops).

Task #4: Find all routes between two cities

Ask the user to supply the departure city and the arrival city.

Find all possible one-way routes between the given city combination by checking their weekly schedules. Print a list of flight number, departure city, departure time,and arrival time for all routes.

Direct routes are trivial. In addition to direct routes, we also allow routes with *only one connection* (i.e. two flights in the route). However, for a connection between two flights to be valid, both flights must be operating the same day at least once a week (when looking at their weekly schedules) and, also, the arrival time of the first flight must be at least one hour before the departure time of the second flight.
Hint: For simplicity you may split this into two queries: one that finds and prints the direct routes, and one that finds and prints the routes with one connection.

Task #5: Find all routes between two cities of a given airline

Ask the user to supply the departure city, the arrival city and the *name of the airline*.

Query the schedule database and find all possible one-way routes between the given city combination. Print a list of airline id, flight number, departure city, departure time, and arrival time for all routes.

Direct routes are trivial. In addition to direct routes, we also allow routes with *only one connection* (i.e. two flights in the route). However, for a connection between two flights to

be valid, both flights must be operating the same day at least once a week (when looking at their weekly schedules) and, also, the arrival time of the first flight must be at least one hour before the departure time of the second flight.
Hint: For simplicity you may split this into two queries: one that finds and prints the direct routes, and one that finds and prints the routes with one connection.

Task #6: Find all routes with available seats between two cities on given date
Ask the user to supply the departure city, the arrival city, and the date. Same with the previous task, print a list of flight number, departure city, departure time, and arrival time for all available routes.

Note that this is a difficult query. You need to build upon the previous task. You need to be careful for the case where we have a non-direct, one-connection route and one of the two flights has available seats, while the other one does not.

Task #7: Add reservation
Ask the user to supply the information for all the flights that are part of his/her reservation. For example, for each "leg" of the reservation you should be asking for the flight number and the departure date. There can be a minimum of one leg (one-way ticket, direct route) and a maximum of four legs (round-trip ticket, with one connection each way). A simple way to do this is to ask the user to supply the flight number first, and then the date for each leg, and if they put a flight number of 0 assume that this is the end of the input.

After getting all the information from the user, your program must verify that there are still available seats in the given flights. If there are seats available on all flights, generate a unique reservation number and print this back to the user, along with a confirmation message. Otherwise, print an error message.

Task #8: Delete reservation
Ask the user to supply the reservation number to be deleted.

When a reservation and all the associated flight information is removed, the plane of each flight should be switched to the smaller-capacity plane, if the number of ticketed passengers fits in a smaller capacity plane.

Task #9: Show reservation info, given reservation number
Ask the user to supply the reservation number.

Query the database to get all the flights for the given reservation and print this to the user. Print an error message in the case of a non-existent reservation number.

Task #10: Buy ticket from existing reservation
Ask the user to supply the reservation number.

Mark the fact that the reservation was converted into a purchased ticket.

Task #11: Find the top-k customers for each airline
Ask the user to supply k the number of top-k customers they desire to display.

The system should display, for each airline, the top-k customers who have paid the most amount of money.

Task #12: Find the top-k traveled customers for each airline
Ask the user to supply k the number of top-k customers they desire to display.

The system should display, for each airline, the top-k customers with the highest number of legs with that airline.

Task #13: Rank the airlines based on customer satisfaction
The system should rank the airline companies based on the number of the customers who bought tickets.

How to proceed

Before implementing the required new/modified SQL queries within your application program, you should test them using pgsql (after you populate the database with test data). Only after you are confident that you have the correct query should you start implementing it in JDBC.

The tasks have been ordered so that you continue building on previous tasks as you move along. Therefore, it is advisable to implement them in order, starting from administrative task #1.

You *should use* views, stored functions and procedures when implementing your tasks. Recall that stored functions/procedures as well as triggers can be used to make your code more efficient besides enforcing integrity constraints. You should feel free to use additional triggers as you feel necessary.

All errors should be captured "gracefully" by your application. That is, for all tasks, you are expected to check for and properly react to any errors reported by PostgreSQL (DBMS), providing appropriate success or failure feedback to the user. Also, your application must check the input data from the user for validity and to avoid any vulnerabilities (e.g., SQL injection). You need to create your own test data starting with the example data above.

Attention must be paid in defining transactions appropriately. Specifically, you need to design the **SQL transactions** appropriately and when necessary, use the concurrency control mechanisms supported by PostgreSQL (i.e., locking modes for serializability) to make sure that inconsistent state will not occur.

There is one situation that you should definitely handle:

- If two customers attempt to make a reservation on the same flight from two different terminals when the flight has only one seat available, one of the reservations should fail.

Finally, you can develop your project on your local machine, but we will only test your code on class3. So even though your code works locally, you should make sure that it also works on the class3 server. **It is your responsibility to submit code that works**.

What to Submit

The second and final milestone should contain, in addition to the SQL part, the Java code, and the driver.

You are required to turn in three files (use team01 to name your files if you are in group 1):

- team01.tar or team01.zip:
  A packed or compressed file that contains all program source files. You could also hand in team01.java if that's the only source file you have.
- team01.sql:
  A PostgreSQL script file that contains the SQL code (e.g., create the database, define the trigger, stored functions and procedures).
- team01-report.doc:
  A user's manual for the system, the system's limitations (e.g., missing or non-implemented functionality) and the possibilities for improvements.

How to submit your project

- Email your Git commit ID and the full web link to your Git repository to cs1555-staff with all team members CC'd. For this milestone, each group can **only submit once**. In addition to submitting the Git commit ID via email to **cs1555-staff with all team members CC'd**, you must **add the TAs as collaborators** (Github usernames: nixonb91, ralseghayer) to the Github repository. Make sure the GitHub repository is **private**. In your GitHub repository you are required to have the **three files specified in What to Submit**.
- Upload the three submission files of your project through the Web-based submission interface you have used for previous Assignments.Note: **Only one team member** should submit the files through the Web-base submission interface since this is a group assignment.
- Submit your files by the due date **(8:00pm, Nov. 24, 2020). There is no late submission.**
- **It is your responsibility to make sure the project was properly submitted.**

Group Rules

- It is expected that all members of a group will be involved in all aspects of the project development. Division of labor to data engineering (db component) and software engineering (JAVA component) is not acceptable since each member of the group will be evaluated on both components.
- The work in this assignment is to be done *independently* by each group. Discussions with other groups on the assignment should be limited to understanding the statement of the problem. Cheating in any way, including giving your work to someone else will result in an F for the course and a report to the appropriate University authority.

Grading

The project will be graded on correctness, robustness (error-checking, i.e., it produces user-friendly and meaningful error messages) and readability. You will not be graded on efficient code with respect to speed although bad programming will certainly lead to incorrect programs. Programs that fail to compile or run or connect to the database server earn zero and no partial points.

Enjoy your class project!