CS 1555/2055 – DATABASE MANAGEMENT SYSTEMS (FALL 2016)
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF PITTSBURGH

Project: PittTours

Release: Oct 31, 2016

Due: Milestone 1 - Nov 15, 2016 @ 8:00pm
Milestone 2 - Nov 29, 2016 @8:00pm
Milestone 3 - Dec 10, 2016 @8:00pm

Project Goal

The primary goal of this project is to implement a flight reservation system for *"PittTours"*, 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 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.

You will be given the descriptions and schemas of the database on which the system is based. However, you need to design a text-based interface which supports all the tasks using Java, Oracle PL/SQL and JDBC. The assignment focuses on the database component and not on the user interface. Hence, NO HTML or other graphical user interface is required for this project.

Milestone 1: The PittTours database schema and example data

The database for PittTours contains 7 tables, explained as follows. You are required to define all of the structural and semantic integrity constraints and their modes of evaluation. For both structural and semantic integrity constraints, you must state your assumptions as comments in your database creation script.

- **Airline information**
  Airline(airline_id, airline_name, airline_abbreviation, year_founded)
  PK (Airline_ID)
  Datatype
  - airline_id, varchar(5)
  - airline_name varchar(50)
  - airline_abbreviation varchar(10)
  - year_founded int

  Here are example tuples:

  ```
  001 United Airlines UAL 1931
  002 All Nippon Airways ANA 1952
  003 Delta Air Lines DAL 1924
  ```

  For Headquarter we use the name of the city that the actual headquarter of the airline is located.

- **Flight schedule information**
  Flight(flight_number, airline_id, plane_type, departure_city, arrival_city,departure_time, arrival_time, weekly_schedule)
  PK (flight_number)
  FK (plane_type) → Plane.plane_type
  FK (airline_id) → Airline.airline_id
  Datatype

- flight_number varchar(3)
- plane_type char(4)
- departure_city varchar(3)
- arrival_city varchar(3)
- departure_time varchar(4)
- arrival_time varchar(4)
- weekly_schedule varchar(7)

Here are example tuples:

```
153 001 A320 PIT  JFK  1000 1120 SMTWTFS
154 003 B737 JFK DCA 1230 1320 S-TW-FS
552 003 E145 PIT  DCA 1100 1150 SM-WT-S
```

For departure and arrival cities we use the three letter airport code. Departure and arrival times are given in "military" notation (i.e., 10pm is written as 2200). Weekly_schedule is a seven-character string with the days of the week (starting from Sunday). If instead of the corresponding letter there is a - for a day, then there is no flight that day.

- **Plane information**
  Plane(plane_type, manufacture, plane_capacity, last_service_date, year, owner_id)
  PK(plane_type)
  FK (owner_id) → Airline.airline_id
  Datatype
  - plane_type char(4)
  - manufacture varchar(10)
  - plane_capacity int
  - last_service date
  - year int
  - owner_id varchar(5)

  Here are example tuples:

```
B737 Boeing 125  09/09/2009 1996 001
A320 Airbus 155  10/01/2011 2001 001
E145 Embraer 50  06/15/2010  2008 002
```

- **Flight pricing information**
  Price(departure_city, arrival_city, airline_id, high_price, low_price)
  PK (departure_city, arrival_city)
  FK (airline_id) → Airline.airline_id
  Datatype
  - departure_city varchar(3)
  - arrival_city varchar(3)
  - high_price int
  - low_price int

  Here are example tuples:

```
PIT JFK 001 250 120
JFK PIT 003 250 120
JFK DCA 003 220 100
DCA JFK 003 210  90
PIT DCA 002 200 150
DCA PIT 001 215 150
```

Prices are in dollars. High_price corresponds to same-day travel which is traditionally over-priced, whereas low_price corresponds to travel where the departure and arrival dates are not the same day. For simplicity, we do not deal with the multiple classes of service (e.g., First, Business, Economy) or discounts that airlines typically have. In the general case, the price for flying in one direction between a pair of cities is not the same as the price for flying in the opposite direction. So to find the price of a round-trip ticket between DCA and JFK with low price, we would add $90 and $100, which gives $190. If a flight is composed of multiple legs, we only consider the end-points. For example, a round-trip flight from PIT to DCA with a connection in JFK will cost $200+$215 = $415 under a high price policy.
Note: Any direct (individual) flight of a round-trip which departures on the same day as the arrival day of a previous flight is also charged at high price. In the case of trip with a connection on the same day, you consider low price for the two legs.

- **Customer information**
  Customer(cid, salutation, first_name, last_name, credit_card_num, street, credit_card_expire, city, state, phone, email)
  PK (cid)
  Datatype
  - cid varchar(9)
  - salutation varchar(3)
  - first_name varchar(30)
  - last_name varchar(30)
  - credit_card_num varchar(16)
  - credit_card_expire date
  - street varchar(30)
  - city varchar(30)
  - state varchar(2)
  - phone varchar(10)
  - email varchar(30)
  - frequent_miles varchar(5)

  PittTours must keep information on all its customers. Cid is the unique customer identification number. A salutation can be one of three values Mr, Mrs and Ms. If the customer is a frequent_mile number of some airline then the frequent_miles attribute will hold the ID of that airline. As a frequent_mile member, the customer will get 10% discount on the advertised price when booking with their frequent airline. For the simplicity, we assume each customer can have at most one frequent_airline. When booking flights involves connections, this discount can still apply to the part of flight that are belong to the customer's frequent airline (e.g., for fights that have one connection and one of the two connecting flight was belong to the frequent airline of the customer then this connecting flight that belongs to the frequent airline will receive a 10% discount).

- **Reservation information**
  Reservation(reservation_number, cid, cost, credit_card_num, reservation_date, ticketed)
  PK (reservation_number)
  FK (cid) → Customer.cid
  Datatype
  - reservation_number varchar(5)
  - cid varchar(9)
  - cost int
  - credit_card_num varchar(16)
  - reservation_date date
  - ticketed varchar(1)

  Reservation_detail(reservation_number, flight_number, flight_date, leg)
  PK (reservation_number, leg)
  FK (reservation_number) → Reservation.reservation_number
  FK (flight_number) → Flight.flight_number
  Datatype
  - reservation_number varchar(5)
  - flight_number varchar(3)
  - flight_date date
  - leg int

  After verifying that a particular route has available seats, a reservation can be made for a particular trip for a customer. Reservation information for a particular reservation is maintained in two tables. Information in Reservation table includes the total cost of the trip, the date on which day the reservation was placed and a flag (ticketed) indicating whether a ticket has been issued (the value is either 'Y' or 'N'). Information in Reservation_detail table includes all the legs (starting from 0 and increased by 1 for consecutive legs) of the flight along with the dates they are performed. Note that each reservation covers an entire trip, i.e., includes the return portion for round-trip reservations. For simplicity, we do not allow a customer to cancel or modify an existing reservation, although unpaid/non-ticketed reservations are cancelled by the system automatically 12 hours prior the departure (see trigger below).

- **Our time information**
  Date(c_date)
  PK (c_date)
  Datatype
  - c_date: date

  You must maintain our system time (not the real system time) in this table. The reason of making our system time different from the real system time is to make it easy to generate scenarios (time traveling) to debug and test your project. All timing should be based on our system time.

Triggers

- 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 create a trigger called `planeUpgrade`, that changes the plane (type) of a flight to an immediately higher capacity plane (type), if it exists, when a new reservation is made on that flight and there are no available seats (i.e., the flight is fully booked). A change of plane will fail only if the currently assigned plane for the flight is the one with the biggest capacity. For simplicity, we assume that there are always available planes for a switch to succeed.
- You should write a trigger, called `cancelReservation`, that cancels (deletes) all non-ticketed reservations for a flight, 12 hours prior the flight (i.e., 12 hours before the flight is scheduled to depart) and if the number of ticketed passengers fits in a smaller capacity plane, then the plane for that flight should be switched to the smaller-capacity plane.

Note: Once you have created a schema and integrity constraints for storing all of this information, you should generate sample data to insert into your tables. Generate the data to represent at least 200 users, 300 reservations, 10 airlines, 30 planes and 100 flights.

Milestone 2: Interface and Functions

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:

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

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:

```
001 United Airlines UAL Chicago 1931
002 All Nippon Airways ANA Tokyo 1952
003 Delta Air Lines DAL Georgia 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:

```
153 001 A320 PIT  JFK  1000 1120 SMTWTFS
154 003 B737 JFK DCA 1230 1320 S-TW-FS
552 003 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 001 250 120
JFK PIT 003 250 120
JFK DCA 003 220 100
DCA JFK 003 210  90
PIT DCA 002 200 150
DCA PIT 001 200 150
```

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  09/09/2009 1996 001
A320 Airbus 155  10/01/2011 2001 001
E145 Embraer 50  06/15/2010  2008 002
```

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 given flight for the given date. Print the passenger list (salutation, first name, last name).

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 given day

7: For a given airline, find all routes with available seats between two cities on given day
8: Add reservation
9: Show reservation info, given reservation number
10: Buy ticket from existing reservation

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*. 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 last and first name already exists.
Assign a unique PittRewards number (i.e., Cid) for the new user.
Insert all the supplied information and the PittRewards number into the database.
Display the PittRewards number 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), including the PittRewards number i.e. the cid.

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.

Task #4: Find all routes between two cities

Ask the user to supply the departure city and the arrival city. Query the schedule database and find all possible one-way routes between the given city combination. 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 might be the most difficult query of the project. 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: For a given airline, find all routes with available seats between two cities on given date
Ask the user to supply the departure city, the arrival city, the date and the name of the airline. Same with the previous task, print a list of airline id, flight number, departure, city, departure time, and arrival time for all available routes.

Note that this might be the most difficult query of the project. 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 #8: 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 said 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 #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 case of an 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.

Milestone 3: Bringing it all together

The primary task for this phase is to create a Java driver program to demonstrate the correctness of your PittTours backend by calling all of the above functions. It may prove quite handy to write this driver as you develop the functions as a way to test them. Your driver should also include a benchmark feature that stress the stability of your system buy calling each function automatically for multiple time with reasonably large amount of data and display corresponding results each time a function is being called.

Now this may not seem like a lot for a third milestone (especially since it is stated that you may want to do this work alongside milestone 2). The reasoning for this is to allow you to focus on incorporating feedback from the TA regarding milestones 1 and 2 into your project as part of milestone 3. This will be the primary focus of milestone 3.

How to proceed

Before implementing the required SQL queries within your application program, you should test them using sqlplus (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 Oracle (DBMS), providing appropriate success or failure feedback to the user. Also, your application must check the input data from the user. 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 Oracle (e.g., isolation level, locking modes) 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 either Unixs (Pitt Unix) or Windows environments (Lab PC or your laptop), but we will only test your code in Unix. So even though your code works in Windows environment, you should make sure that it also works on Unixs and does not uses any specialized libraries before you submit your project. **It is your responsibility to submit code that works**.

Project submission

Milestone 1 of the project is due **8:00pm, Nov 15, 2016**. The first milestone should contain only the SQL part of the project. Specifically, the scripts to create the tables and the triggers along with the definition of at least one stored procedure and one function. If you wish to receive more feedback, feel free to include any or all of your SQL queries in your repository.

Note that after the First Phase submission, you should continue working on your project without waiting for our feedback. Furthermore, you should feel free to correct and enhance your SQL part with new views, functions, procedures etc.

Milestone 2 of the project is due **8:00 pm, Nov 29, 2016**. The second milestone should contain, in addition to the SQL part, the Java code.

Milestone 3 of the project is due **8:00 pm, Dec 10, 2016**. The third milestone should contain, in addition to the second milestone, the driver and benchmark.

NO late submission is allowed.

How to submit your project

When you submit your milestones you need to **email us your commit ID** together with the full web link to your repository each time you submit a milestone. For the project, each group can **only submit once per milestone per group**.

Group Rules

- You are asked to form groups of two. You need to notify the instructor and the TA by emailing group information to cs1555-staff@cs.pitt.edu (Remember that you need to send the email from your pitt email address, otherwise the email will not go through). The email should include the names of the team members and should be CC-ed to both team members (otherwise the team will not be accepted).
- The deadline to declare teams is **Nov. 2, 2016** as already announced. If no email was sent before this time, you will be assigned a team member by the instructor. Each group will be assigned a unique number which will be sent by email upon receiving the notification email.
- It is expected that both 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. Program that fail to comply or run or connect to the database server earn zero and no partial points.

Enjoy your class project!