CS 1555/2055 – Database Management Systems (Fall 2020)
Dept. of Computer Science, University of Pittsburgh

Assignment #5: Triggers, Functions, Procedures, and Transactions

Release: Oct. 15, 2020 Due: 8:00 PM, Oct. 29, 2020

Goal

The goal of this assignment is to gain familiarity with Triggers, Functions, Procedures, and Transactions. In this assignment, your team will complete the first milestone of the term project by setting up the database for the project schema and inserting provided sample-data into your database.

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 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. In this first project milestone / assignment (HW5), you will be given the descriptions and schemas of the database on which the system is based.

Description:

The database for *Pitt Tours* contains 7 tables plus an auxiliary table, 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 (e.g., UNIQUE, NOT NULL, CHECK), you must state your assumptions as comments in your database creation script.

Pitt Tours Tables

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

  Here are some example tuples:

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

- **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, airline_id) → (Plane.plane_type, Plane.owner_id)
  FK (airline_id) → Airline.airline_id
  Datatype
  - flight_number integer
  - airline_id integer
  - plane_type char(4)
  - departure_city char(3)
  - arrival_city char(3)
  - departure_time varchar(4)
  - arrival_time varchar(4)
  - weekly_schedule varchar(7)

  Here are some example tuples:

  ```
  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
  ```

  For departure and arrival cities we use the three letter airport code (e.g., PIT is for Pittsburgh, DCA for Regan Washington D.C.). Departure and arrival times are given in "military" notation (i.e., 10 pm 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, manufacturer, plane_capacity, last_service_date, year, owner_id)
  PK (plane_type, owner_id)
  FK (owner_id) → Airline.airline_id
  Datatype
  - plane_type char(4)
  - manufacturer varchar(10)
  - plane_capacity integer
  - last_service date
  - year integer
  - owner_id integer

  Here are some example tuples:

  ```
  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
  ```

- **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 char(3)
  - arrival_city char(3)
  - airline_id integer
  - high_price integer
  - low_price integer

  Here are some 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 plus $215, which gives $415 under a high_price policy.

  Note: Any direct (individual) flight of a round-trip (e.g., PIT-JFK, JFK-PIT) on the same day is charged at high_price. In the case of a trip over different dates 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, frequent_miles)
  PK (cid)
  FK (frequent_miles) → Airline.airline_abbreviation
  Datatype
  - cid integer
  - 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(10)

  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 member of some airline then the frequent_miles attribute will hold the abbreviation 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 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 belongs to the customer's frequent airline (e.g., for flights that have one connection and one of the two connecting flights belongs to the frequent airline of the customer then the connecting flight that belongs to the frequent airline will receive a 10% discount).

  Here is an example tuple:

  ```
  123 Mr John Walker 2 Pitt Ave' Pittsburgh PA 4123456789 j@pitt.edu UAL
  ```

- **Reservation information**
  Reservation (reservation_number, cid, cost, credit_card_num, reservation_date, ticketed)
  PK (reservation_number)
  FK (cid) → Customer.cid
  FK (credit_card_num) → Customer.credit_card_num
  Datatype
  - reservation_number integer
  - cid integer
  - cost decimal
  - credit_card_num varchar(16)
  - reservation_date timestamp
  - ticketed boolean

- 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 integer
  - flight_number integer
  - flight_date timestamp
  - leg integer

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 `TRUE` or `FALSE`). Information in Reservation_detail table includes all the legs (starting from 1 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 to the departure (see trigger below).

Auxiliary Table

- **Our Time Information**
  OurTimestamp (c_timestamp)
  PK (c_timestamp)
  Datatype
  - c_timestamp timestamp

  You must maintain our system time (not the real system time) in this (auxiliary) table. The reason for 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 (c_timestamp). No attribute in the *Pitt Tours* tables should refer to the OurTimestamp table. OurTimestamp has only one tuple, inserted as part of initialization and is updated during time traveling.

Sample data: the provided example tuples above and in the sample-data file are intended for clarification of the description, and are not sufficient for testing the correctness of your solution. You need to add or modify the sample-data accordingly in order to test your solution.

Tasks:

1. [20 Points] Use SQL CREATE TABLE statement to create tables for the relations given in the **Description** above. You need to define primary keys foreign keys (if any) alternate keys (if any), as well as any additional semantic constraints, such default values and whether an attribute can take a NULL as a value or not.

2. [10 Points] Create a function, called `getCancellationTime`, that accepts as input a reservation number and returns the time when the reservation should be canceled (i.e., 12 hours before the flight is scheduled to depart). After the function creation, add the necessary code to execute the function `getCancellationTime`.

3. [10 Points] Create a function, called `isPlaneFull`, that checks if a *specific flight* has no available seats (i.e., the flight is fully booked). This function should return true if the plane is full and false if the plane is not full. You need to come up with all the necessary arguments (input parameters) for `isPlaneFull`. After the function creation, add the necessary code to execute the function `isPlaneFull`.

4. [15 Points] Create a procedure, called `makeReservation`, that records the details of a segment of a trip. The input to the `makeReservation` procedure is reservation_number, flight_num, departure_date and order (leg number within the trip). Note that the flight time for flight_date is calculated based on a flight_number and departure_date using the `substr(string, from [, count])` function (this could be another SQL function). To test the procedure `makeReservation`, add the necessary code to call it and form a trip (i.e., inserting an appropriate tuple in the Reservation table and manually generate the necessary information of a trip using the sample data).

5. [20 Points] 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. The function, `isPlaneFull`, should be used when checking if a flight has no available seats. 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. After the trigger definition, add the necessary code to fire the trigger `planeUpgrade`.

6. [20 Points] Write a trigger, called `cancelReservation`, that cancels (deletes) all non-ticketed reservations for a flight at the flight's cancellation time. The function, `getCancellationTime`, should be used to obtain a flight's cancellation time (i.e., 12 hours before the flight is scheduled to depart). 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. Each reservation cancellation should be a single transaction, that is, the deletion of a reservation from all effected tables should be handled as a single atomic operation. After the trigger definition, add the necessary code to fire the trigger `cancelReservation`.

What to submit [5 points]

- **<team_number>-hw5-db.sql**
  (e.g., team14-hw5-db.sql)
  In this file, please submit the answers to Task 1. In addition to providing the answers, you are expected to:
  - **include your team number/name, your names and pitt usernames at the top of the SQL script as an SQL comment**.

  The entire SQL script file should be composed of **valid SQL statements**.
- **<team_number>-hw5-query.sql**
  (e.g., team14-hw5-query.sql)
  In this file, please submit the answers to Tasks 2, 3, and 4. In addition to providing the answers, you are expected to:
  - **include your team number/name, your names and pitt usernames at the top of the SQL script as an SQL comment** and
  - identify the task number before each as an SQL comment.

  The entire SQL script file should be composed of **valid SQL statements**.
- **<team_number>-hw5-triggers.sql**
  (e.g., team14-hw5-triggers.sql)
  In this file, please submit the answers to Tasks 5 and 6. In addition to providing the answers, you are expected to:

  - **include your team number/name, your names and pitt usernames at the top of the SQL script as an SQL comment**, and
  - identify the task number before each as an SQL comment.
- **<team_number>-hw5-insert.sql**
  (e.g., team14-hw5-insert.sql)
  In this file, please submit the insert statement you used for populating the data in your database. In addition to providing the answers, you are expected to:
  - **include your team number/name, your names and pitt usernames at the top of the SQL script as an SQL comment**.
- **<team_number>-hw5-output.txt>**
  (e.g., team14-hw5-output.txt)
  In this file, please submit the output of your SQL scripts (i.e., the above files. To do that in PSQL you can use the command "\o" as seen in the recitation video.
  In addition to providing the answers, you are expected to:
  - **include your team number/name, your names and pitt usernames at the top of the text file**.

How to submit

The project including this assignment (HW5) will be collected also by the TAs via your team's private GitHub repository that is shared only with the TAs (GitHub username: ralseghayer, GitHub username: nixonb91).

- **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. In your GitHub repository you are required to have the **four files specified in What to Submit**.
- Submit your files (i.e., four SQL script files and a text output file) that contain your solution through the Web-base 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. It is your responsibility to make sure the assignment was properly submitted.**
- Submit your files by the due date **(8:00pm Oct. 29, 2020). There is no late submission.**

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.