Instructions: Upload your submission as a .zip or tar.gz file.

Files to submit for group (Assignment called Homework 7 on ICON)
- hw7.zip or hw7.tar.gz
- Inside the archive should be all files required to run and test your database. You do not need to include the MySQL jar or library file.

Files to submit per individual (Assignment called HW6-7 Review)
- Online you will write very briefly:
  - your contribution to HW 6 and 7
  - your partner’s contribution to HW 6 and 7

Goals for this assignment
- Implement a database from a model
- Write valid transactions from a specification
- Test a transactional workload
- Communicate your application and design considerations

Description

In HW6, you designed a database for either a hotel management system or e-banking system. In HW7, you will implement the database, as well as queries and transactions that the business requires. You will implement the application in either Python or Java and use MySQL as the backing database.

In a complete system, you would implement a presentation layer, such as a web interface, but to keep the focus on the database, you will limit interaction to be programmatic.

As with HW6, for HW7 submit only one project per team.

Tip: read the whole document and start early. This HW does not have any starter code and has several specific requirements.
Getting started

1. Make sure you can connect successfully to the MySQL database with either Python or Java. To do so, follow the directions under the MySQL section of http://homepage.cs.uiowa.edu/~bdmyers/cs4400_fa16/resources/
2. Download the sample applications code from https://github.com/bmyerz/mysql_apps/archive/master.zip
   (alternatively, you may use git to clone https://github.com/bmyerz/mysql_apps.git)
3. Use the README.md in that folder to try out the application

What you must do

You will implement two database clients, a manager client and a customer client, for your application. Each client will support a fixed set of queries and transactions.

Commands to support for both scenarios

When you run your application, it must first require a login. There are two options for logging in: manager or a specific customer. For the customer, you must provide the customer id (or whatever unique identifier, i.e. primary key, your database design has for a customer).

login('manager')

login('customer', unique_id)

You do not need to worry about authentication in this homework. And, authorization will be implemented by construction because of the limited types of queries each user can run.

Commands to support specific to the scenario

Implement the queries and transactions for your scenario, as described in HW6. It is up to you to decide on the appropriate arguments for each query/transaction based on your database design. But for example, in the hotel, check_in might take one argument, which is the reservation number. In the e-bank, new_account might take a customer identifier and an account type and initial balance.

Hotel management

Required queries to support for management user:

a. rooms_occupied: view the currently occupied rooms
b. **housekeeping**: list house-keeping assignments

customer user:
   a. **rooms available**: view the room types and costs that are still available
   b. **cost at checkout**: calculate the total cost for the guest at checkout time
   c. **my reservations**: list future reservations for the guest

Required transactions to support for

management user:
   a. **check in**: check-in a guest (must be day of reservation or fail)
   b. **check out**: check-out a guest (if the checkout is early then calculate the refund to the guest)
   c. **mark serviced**: record that a room has been serviced for the day; no room should be serviced twice in the same day

customer user:
   a. **reserve**: make a reservation (denied if no more rooms of requested type)
   b. **cancel**: cancel a reservation (denied if same day)

E-banking

Implement the queries and transactions for your scenario, as described in HW6.

Required queries to support for

management user:
   a. **summary**: calculate the total deposits and withdrawals from the whole bank in a given time range

customer user:
   b. **accounts**: view all customer’s accounts
   c. **balance**: view balance of a customer account
   d. **transactions**: view transactions from latest month for a customer
   e. **balance all**: view the total balance of all of a customer’s accounts

Required transactions to support for

management user:
   a. **new account**: add an account for a customer
   b. **process fees**: withdraw fees from every customer account and put them into the bank’s master account. Fees are assessed according to transactions in the last month
c. **process_interest**: deposit interest into every customer account taken from the bank’s master account. Interest rate is determined by the account type and the interest rate for that type of account.

d. **update_fee**: update current fees for each kind of transaction

customer user:

e. **transfer**: transfer money between two accounts owned by the same customer (denied if one account has too little money)

**Test programs and example interactions**

HW7 only requires you to build a programming interface. However, you must provide a number of test programs that create a client and issue sequences of method calls to it to simulate interaction. Write enough programs to cover the interesting situations.

The test programs should be drawn from the situations you listed in HW6, but you can also add additional ones.

**Implementation**

a. Implement each of your two clients as a class (e.g. ManagerClient and CustomerClient), where the methods are the different queries and transactions.

b. See the example application for how to issue queries and transactions for Python or Java.

c. It is highly recommended that you simply set your transaction isolation level to ‘serializable’. There is an example of this setting in the example application.

d. Your transactions must behave in a way that is consistent with a real application. If a transaction fails because MySQL indicates deadlock, then you should retry the transaction. If a transaction fails because a resource is depleted (e.g., rooms, money) then you should return an error to the user.

e. Feel free to add additional queries or transactions as appropriate, but you must include all of the required ones.

f. You do need not create an interface beyond the programmatic one (i.e., classes and methods). *Every method must* print a useful message to the user.

g. Your code must be well documented. Specify description/parameters/return for each “public” method. Comment code as needed to make it clear what it is doing.
h. Although not required, it is highly recommended that you use version control to work with your team.

**Grading**

Since this assignment is relatively open-ended, we will be grading by appointment.

Each group will sign up for a 20-minute slot.

Be prepared to
- Demonstrate on your computer 2-3 important scenarios. They must all involve two users running at the same time.
- Discuss important design considerations that came up. For example, how your design changed if at all, what was difficult/surprising.
- Answer questions about how any of your queries or transactions work

20 minutes is **very short**!
- Bring a copy of your ER/relational models on paper
- Every group member must be there and present roughly equal amounts.
- You must rehearse/run your scenarios before the appointment and make sure they work.
- You must have a list of design considerations that you are ready to present.