**OBJECTIVE:** How to use SQLAlchemy to access a back-end database in PostgreSQL that is structured according to your phase 1 design.

**INTRODUCTION:** Writing the necessary code in Python to enter and update all the tables from your phase 1 design is going to take too long, and I want to concentrate on a few basic features in SQLAlchemy, rather than use the same features over and over again since there will be a good deal of overlapping code from one class to the next.

We are going to take a hybrid approach. We will populate some of the tables from your phase 1 design with data before running your application, and we will only read data from those tables. We will insert, update, and delete from the rest of your tables.

It is very important that you write this application **without any assumptions regarding the data already in the database**. Pretend as though each time this application is run a) someone else could be running it as well, and b) you have no idea what data is in those tables that I’ll have you load up ahead of time. That will force you to perform due diligence when you accept input from the user and/or update the data in the database.

The Python application that you are going to write should provide a **very simple console** interface. Read and write everything from/to the console. There is no need to use a snazzy GUI development environment. The objective is to concentrate on the database access in this assignment.

**PROCEDURE:**

1. Take the ERD from phase 1 and execute the CREATE TABLE statements that it produces.
2. Write insert statements to populate:
   1. Employees, Buildings, rooms, doors, hooks, and any junction tables between any two of those.
   2. Do not go overboard with the initial data. Somewhere near a half-dozen or so rows in each table will be fine.
   3. Execute those outside of your application to insert the necessary “seed” data.
3. Write your Python application to update the rest of the tables. You will need a menu option to:
   1. Create a new Key.
   2. Request access to a given room by a given employee.
   3. Capture the **issue** of a key to an employee
   4. Capture losing a key
   5. Report out all the rooms that an employee can enter, given the keys that he/she already has.
   6. Delete a key.
   7. Delete an employee.
   8. Add a new door that can be opened by an existing hook.
   9. Update an access request to move it to a new employee.
   10. Report out all the employees who can get into a room.
4. Demonstrate your application
   1. As much as possible, have everyone in the team involved in the demonstration.
   2. Create a video recording of your demonstration.
   3. **Please provide me audio**. Just watching a cursor flying around the screen and text flying by is not going to help me much. Instead, have someone narrating the demonstration while it is going on.
   4. You can post it to YouTube if you like, just make sure that I can get to it and that you provide me with the appropriate link.

**WHAT TO TURN IN:**

* Your UML model from phase 1 for reference. If you had to change the model while developing your application, that’s fine, just give me the latest version.
* Your Python project in a single .zip file.
* Console output (you can do this in a single text file) from a sample run of your application.
* Screenshot of the contents of your key issue table at the end to illustrate that you were able to perform inserts.