**OBJECTIVE:** Learn how to use MongoDB to serialize data that is structured according to your phase 1 design.

**INTRODUCTION:** Writing the necessary code in Python to enter and update **all** the classes from your phase 1 design is going to take too long, and I want to concentrate on a few basic features in MongoDB, 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 collections from your phase 1 design with data before running your application, and we will only read data from those collections. We will insert, update, and delete from the rest of your collections.

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 collections 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.

**Ideally**, if you did a good job of modular programming with minimal coupling between the logic of your application and the back-end database, the transition from SQLAlchemy and PostgreSQL to MongoDB should be relatively smooth. The same testing that you used on Phase 2 of the project should work just as well here, and much of your code from Phase 2 should port over nicely. Ideally.

In general, you need to validate the user input, either by using a try/catch block or checking the user input before submitting it to MongoDB. For instance, when you prompt the user for the employee to delete, you need to make sure that the employee exists before deleting them. You also want to make sure that any documents that depend upon that employee are also deleted, so that you do not leave any “orphans” in your collections.

Most of you succeeded in building a simple menu system for phase II of this project, but just to be clear, I want you to set up a loop:

* Show the user the menu
* Accept the user’s menu choice
* Prompt the user for any information needed to perform that menu choice
* Report out whether the operation was a success
* Prompt the user again. One of the options is to exit the program.

**PROCEDURE:**

1. Using your UML from phase 1, decide how you are going to implement that in MongoDB. I am personally not aware of an analog to ERD in MongoDB, you will have to get creative about how you capture the decisions that you make regarding your implementation strategy. Remember that you have several Mongo tools at your disposal:
   1. Uniqueness constraints
   2. References (to simulate relationships)
   3. MongoDB schemas on your collections.
2. Write insert statements to populate:
   1. Employees, Buildings, rooms, doors, hooks, and any junction collections between any two of those.
   2. Do not go overboard with the initial data. Somewhere near a half-dozen or so documents in each collection 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 collections. You will need a menu option to:
   1. Create a new Key.
      1. Present the user with a list of the available hooks
      2. Prompt them for which hook they will use to make the key
      3. Generate the key number. I prefer that you use a serial for that
   2. Request access to a given room by a given employee.
      1. Present the user with a list of the Employees by name and prompt for which one
      2. Present the user with a list of the buildings and rooms and prompt for which one – note: you **can** do that as one prompt if you really want to.
   3. Capture the **issue** of a key to an employee
      1. This could be part of giving access if you structured your data that way
      2. Prompt them for the Access
      3. Then **you** either find the existing key that meets that need, or you make a new one on a hook of your choice that opens at least one of the doors to that room.
   4. Capture losing a key
      1. Prompt them for the key request that was lost.
      2. Capture the date and time of the loss. You **can** default to current date and time.
   5. Report out all the rooms that an employee can enter, given the keys that he/she already has.
      1. Prompt for the employee
      2. List the rooms that they have access to
         1. Order by building, then room
         2. Remove duplicates
   6. Delete a key.
      1. Check for any references to that key.
         1. Either delete the references first
         2. Or put into a try/catch block and let the user know that the key is in use and you cannot delete it.
      2. Only delete the key if it will **not** cause an exception to show on the screen.
   7. Delete an employee.
      1. Same basic cautions as deleting a key
   8. Add a new door that can be opened by an existing hook.
      1. Prompt them for the hook
      2. Prompt them for the building
      3. Prompt them for the room
      4. Provide a menu of the available door names and prompt for which door they want
   9. Update an access request to move it to a new employee.
      1. Prompt for the old employee
      2. Prompt for which access (by room) of theirs that you’re to move
      3. Prompt for the new employee
   10. Report out all the employees who can get into a room.
       1. Prompt for the room
       2. List the employees by name
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.
  + This should include any code that you ran to create your collections and establish schemas for them.
  + Also any code that you used to build uniqueness constraints.
* 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 collection at the end to illustrate that you were able to perform inserts.