HW 6 | Appointment Reservation System

Objectives: To gain experience with database application development, and learn how to use SQL from within Python via pymssql.

Due dates:

- Setup: due **Monday, February 26th at 11:00pm**
- Part 1: due **Wednesday, February 28th at 11:00pm**
- Part 2: due **Thursday, March 7th at 11:00pm**

- **setup video for Python**

Contents:

Introduction

A common type of application that connects to a database is a reservation system, where users schedule time slots for some centralized resource. In this assignment you will program part of an appointment scheduler for vaccinations, where the users are patients and caregivers keeping track of vaccine stock and appointments.

This application will run on the command line terminal, and connect to a database server you create with your Microsoft Azure account.

You will have two main tasks:

- Complete the design of the database schema with an E/R diagram and create table statements
- Implement the code that stores Patient information, and lets users interactively schedule their vaccine appointments. We have implemented the code that caregivers use to manage the appointment slots and inventory, which will be a useful reference for you. The implementation is broken up into two milestones, part 1 and part 2, described below.

Be Careful: This homework requires writing a non-trivial amount of code; our solution is about 600 lines, including the starter code. It will take SIGNIFICANTLY more time than your previous assignments.

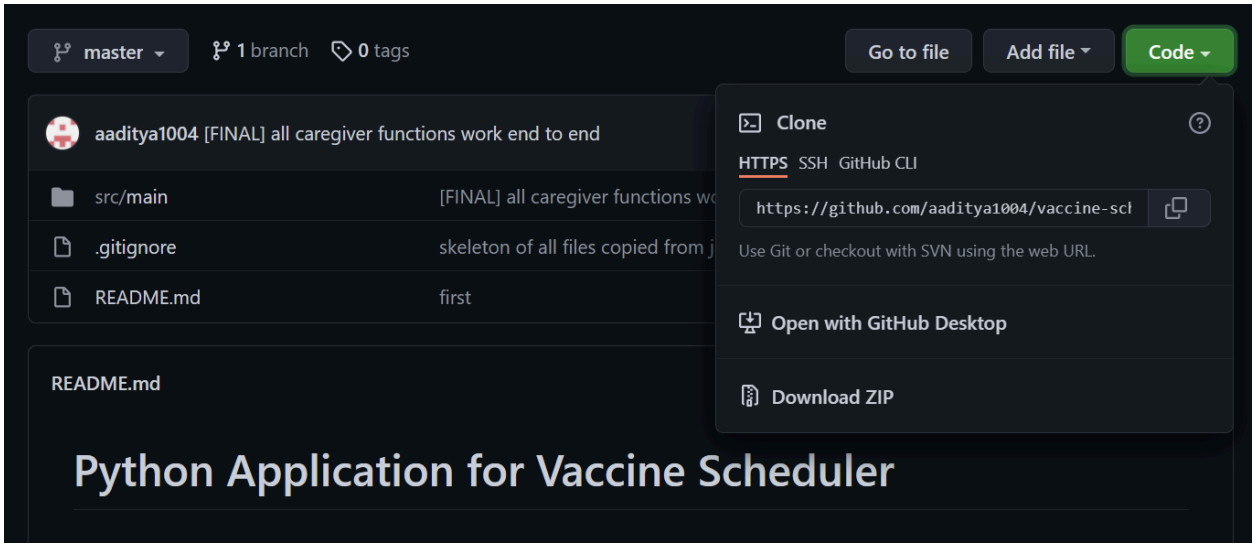
Setup

Make sure to finish this part of the assignment and upload your setup verification of [step 2.4](#) for the first 5 points!

2.1 Clone the starter code

1. Navigate to the Github repository hosting the starter code: https://github.com/aaditya1004/vaccine-scheduler-python (As you continue to work on the assignment, DO NOT manipulate the internal project structure of the assignment. This will save you some headache when it comes time to submit your code)

2. Click on the green button “Code” and select “Download ZIP” from the drop-down menu.



3. Once your download completes, decompress the ZIP file and retrieve the starter code.

2.2 Read through the starter code

We created the important folders and files you will be using to build your application:

- src.main.scheduler/

Scheduler.py:

- This is the main entry point to the command-line interface application. Once you compile and run **Scheduler.py**, you should be able to **interact with the application.**

db/:

- This is a folder holding all of the important components related to your database.
- **ConnectionManager.py**: This is a wrapper class for **connecting to the database**. Read more in 2.3.4. You should run this document to connect to the database so you can successfully interact with the application.

model/:

- This is a folder holding all the class files for your data model.
- You should implement all classes for your data model (e.g., patients, caregivers) in this folder. We have created implementations for Caregiver and Vaccines, and you need to complete the Patient class (which can heavily borrow from Caregiver). **Feel free to define more classes or change our implementation if you want!**

- src.main.resources/
  - **create.sql**: **SQL create statements for your tables**, we have included the create table code for our implementation. You should **copy, paste, and run** the code (along with all other create table statements).

2.3 Configure your database connection

2.3.1 Installing dependencies and Anaconda

Our application relies on a few dependencies and external packages. You'll need to install those dependencies to complete this assignment.

We will be using **Python SQL Driver** to allow our Python application to connect to an Azure database. We recommend using Anaconda for completing this assignment.

Mac users, follow the instructions in the link to **install Anaconda on macOS**: https://docs.anaconda.com/anaconda/install/mac-os/

Windows users, follow the instructions in the link to **install Anaconda on Windows**: https://docs.anaconda.com/anaconda/install/windows/. You can choose to install Pycharm for Anaconda, but we recommend installing Anaconda without PyCharm as we will be using the terminal.

Check that you have Anaconda installed by running **conda -V**. If it outputs a version, you are good to go!

After installing Anaconda:

1. We first need to **create a development environment** in conda.
    a. **macOS** users: launch terminal and navigate to the source directory.
    b. **Windows** users: launch "Anaconda Prompt" and navigate to the source directory.
2. Follow the steps here to **create an environment**. Make sure you remember the name of your environment.
    a. Run: conda create -n [environment name]
3. Activate your environment following the steps here.
    a. Run: conda activate [environment name]
        i. NOTE: You will know you activated the environment if you see the (Homework_6) label instead of the (base) label
    b. To deactivate, Run: conda deactivate
4. Run "**conda install pymssql**" to install the dependencies.
5. Run "**conda env list**" and ensure that the * is next to the environment you just created

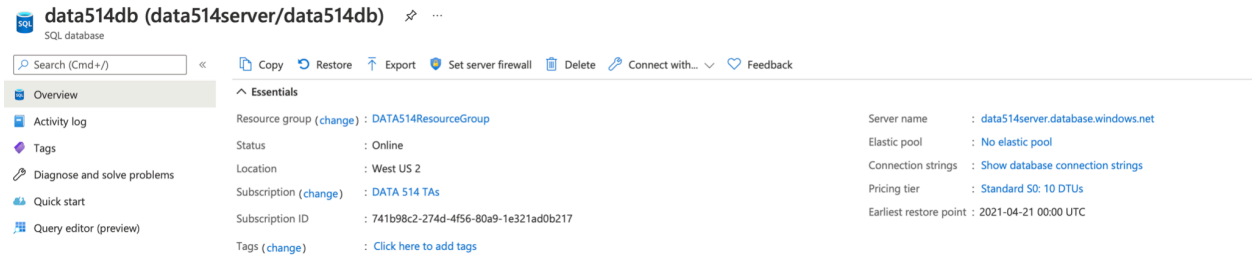
Disclaimer: You don't need to put the file you downloaded in Step 2.1 into the environment folder. You can leave it wherever you downloaded.

2.3.3 Setting up credentials

The first step is to retrieve the information to connect to your Microsoft Azure Database.

- The **server name** can be found in your Azure portal. Only use the part before ".database.windows.net"

    - In the example below, it would be **“**data514server”.
    - **<u>YOU NEED TO CHANGE THIS ACCORDING TO YOUR DATABASE!</u>**
- The **database name** can be found in your Azure portal
    - In the example below, it would be “data514db”
    - **<u>YOU NEED TO CHANGE THIS ACCORDING TO YOUR DATABASE!</u>**
- The **User ID** would be of the format **<user id>@<server name>**
    - For example, it could be **exampleUser@data514server** where “exampleUser” is the login ID that you used to **log in to the query editor on Azure** and “data514server” is the server name.
- **Password** is what you use to **log in to your query editor on the Azure portal**.
    - If your password contains special characters (!, *, etc), you will need to escape the special characters using a backslash. For example, if your password is pass!123, you would run **conda env config vars set Password=pass\!123**.



Once you’ve retrieved the credentials needed, you can set up your environment variables.

2.3.3 Setting up environment variables

Make sure to set this in the environment you created if you’re using virtual environments! Remember, to go into the environment you created, you’ll need to activate it.

In your terminal or Anaconda Prompt, type the following:

```
conda env config vars set Server=414wi24
conda env config vars set DBName=CSE414
conda env config vars set UserID=gananya1
conda env config vars set Password=demoPass1
```

Where “{}” is replaced by the respective information you retrieved from step. Do not actually include the {}.

You will need to **reactivate your environment** after that with just the command **“conda activate [environment name]”**. **Don’t do conda deactivate before this.** After running conda activate, verify you set your credentials properly by running **conda env config vars list**. You should see the Server, DBName, UserID, and Password fields. It should look similar to this:

Server = your-server-name
DBName = your-db-name
UserID = username@your-server-name
Password = your-password

If your password field looks off, be sure that you escaped it properly, as mentioned above.

2.3.4 Working with the connection manager

In **scheduler.db.ConnectionManager.py**, we have defined a wrapper class to help you instantiate the connection to your SQL Server database. **You’ll need to run this document to connect to your database: Run “python ConnectionManager.py”.**
We recommend reading about pymssql **Connection** and **Cursor** classes for retrieving and updating information in your database.
Here’s an example of using ConnectionManager.

```
# instantiating a connection manager class and cursor
cm = ConnectionManager()
conn = cm.create_connection()F
cursor = conn.cursor()

# example 1: getting all names and available doses in the vaccine table
get_all_vaccines = "SELECT Name, Doses FROM vaccines"
try:
    cursor.execute(get_all_vaccines)
    for row in cursor:
        print(name:" + str(row[‘Name’]) + ", available_doses: " + str(row[‘Doses’]))
except pymssql.Error:
    print(“Error occurred when getting details from Vaccines”)

# example 2: getting all records where the name matches “Pfizer”
get_pfizer = "SELECT * FROM vaccine WHERE name = %s"
try:
    cursor.execute(get_pfizer, ‘fizer’)
    for row in cursor:
        print(name:" + str(row[‘Name’]) + ", available_doses: " + str(row[‘Doses’]))
except pymssql.Error:
```

```
    print("Error occurred when getting pfizer from Vaccines")
```

Helpful resources on writing pymssql:

Documentation -> https://www.pymssql.org/ref/pymssql.html

Examples -> https://www.pymssql.org/pymssql_examples.html

2.4 Verify your setup

Once you're done with everything, try to run the program and you should see the following output. You should be running the program in terminal (macOS) or Anaconda Prompt (Windows) and **in your conda environment**.

Note: Command to run the program: "python Scheduler.py" or "python3 Scheduler.py".

```
Welcome to the COVID-19 Vaccine Reservation Scheduling Application!
*** Please enter one of the following commands ***
> create_patient <username> <password>
> create_caregiver <username> <password>
> login_patient <username> <password>
> login_caregiver <username> <password>
> search_caregiver_schedule <date>
> reserve <date> <vaccine>
> upload_availability <date>
> cancel <appointment_id>
> add_doses <vaccine> <number>
> show_appointments
> logout
> quit
```

If you can see the list of options above, congratulations! You have verified your local setup.

Next, to verify that you have set up your database connection correctly, **try to create a caregiver with the command "create_caregiver <username> <password>". Make sure you have created the tables on Azure before testing this command.**

To verify you have done the setup, take a screenshot or phone picture of this screen on your computer, along with your **created caregiver on Azure (a simple SELECT showing that you have created a caregiver would work)**, and upload to gradescope for 5 points.

Deliverables for Setup

Due: **Monday, November 20th at 11:00pm**

Upload to Gradescope a screenshot or phone picture of the welcome prompt on your computer and the successful “create_caregiver <username> <password>” command, along with your created caregiver on Azure.

Requirements

Your assignment is to build a vaccine scheduling application (with a database hosted on Microsoft Azure) that can be deployed by hospitals or clinics and supports interaction with users through the terminal/command-line interface. In the real world it is unlikely that users would be using the command line terminal instead of a GUI, but all of the application logic would remain the same. For simplicity of programming, we use the command line terminal as our user interface for this assignment.

We need the following entity sets in our database schema design (hint: you should probably be defining your class files based on this!):

- **Patients: these are customers that want to receive the vaccine.**
- **Caregivers: these are employees of the health organization administering the vaccines.**
- **Vaccines: these are vaccine doses in the health organization's inventory of medical supplies that are on hand and ready to be given to the patients.**

In this assignment, you will need to:

- Complete the design of the database schema, with an E/R diagram and table statements (Part 1);
- Implement the missing functionality from the application (Part 1 & Part 2)

A few things to note:

- You should handle invalid inputs gracefully. For example, if the user types a command that doesn't exist, it is bad to immediately terminate the program. **A better design would be to give the user some feedback and allow them to re-type the command. Points will be taken off if the program terminates immediately after receiving invalid input.** While you don't have to consider all possible inputs, error handling for common errors (e.g., missing information, wrong spelling) should be considered.
- **After executing a command, you should re-route the program to display the list of commands again.** For example:
    - If a patient 'reserves' their vaccine for a date, you should update your database to reflect this information and route the patient back to the menu again.

1.3 How to handle passwords

You should never directly store any password in the database. Instead, we'll be using a technique called salting and hashing. In cryptography, salting hashes refer to adding random data to the input of a hash function to guarantee a unique output. We will store the salted password hash and the salt itself to avoid storing passwords in plain text. Use the following code snippet as a template for computing the hash given a password string:

```
import hashlib
import os
# Generate a random cryptographic salt
salt = os.urandom(16)
# Generate the hash
hash = hashlib.pbkdf2_hmac(
   'sha256',
   password.encode('utf-8'),
   salt,
   100000,
   dklen=16
)
```

Part 1

Design

You will first need to work on the **design of your database application**. Before you begin, **please carefully read the assignment specification** (including Part 2) and the **starter code**, and think about **what tables would be required to support the required operations**. Once you have an idea of how you want to design your database schema:

- Draw the ER diagram of your design and place it under **src.main.resources (design.pdf)**.
- Write the create table statements for your design, create the tables on Azure, and **save the code under src.main.resources (create.sql)**.

You will also need to implement the corresponding Python classes of your design. We have implemented Caregiver.py for you, but feel free to change any of the details. You will need the following classes, and you may implement more data models if you feel the necessity:

- Caregiver.py: data model for caregivers (implemented for you.)
- Vaccine.py: data model for vaccines (implemented for you.)
- **Patient.py: data model for patients.**
    - **You will implement this class, it can be mostly based on Caregiver.py**

Implementation

Congratulations! You're now ready to implement your design! For Part 1, you will need to implement the following functionalities. We have implemented account creation for caregivers as an example for you, please read through our implementation before you begin.

You will need to implement the following operations:

- create_patient <username> <password>
    - Print "Created user <username>" if create was successful.
        - Example: `Created user p1`
    - If the username is already taken, print "Username taken, try again".
    - For all other errors, print "Create patient failed".
    - **Your output must match exactly. Do not include the "< >" in your output.**
- login_patient <username> <password>

- Print "Logged in as <username>" if login was successful.
  - Example: `Logged in as p1`
- If a user is already logged in in the current session, you need to log out first before logging in again. In this case, print “User already logged in, try again”
- For all other errors, print "Login patient failed"
- **Your output must match exactly. Do not include the “< >” in your output.**

Deliverables for Part 1

Due: **Sunday, November 26th at 11:00pm**

You are free to define any additional files, but Part 1 should have at least the following:

- **src.main.resources**
  - **design.pdf**: the design of your database schema.
  - **create.sql**: the create statement for your tables.
- **src.main.scheduler.model**
  - **Caregiver.py**: the data model for your caregivers.
  - **Patient.py**: the data model for your users.
  - **Vaccine.py**: the data model for vaccines.
  - Any other data models you have created.
- **src.main.scheduler**
  - **Scheduler.py**: the main runner for your command-line interface.

Part 2

You will implement the rest of your application in Part 2.
For most of the operations mentioned below, **Your program will need to do some checks to ensure that the appointment can be reserved (e.g., whether the vaccine still has available doses)**. Again, you do not have to cover all of the unexpected situations, but we do require you to have a reasonable amount of checks (especially the easy ones).

For Part 2, you will need to implement the following operations:

- search_caregiver_schedule <date>
    - Both patients and caregivers can perform this operation.
    - Output the username for the caregivers that are available for the date ordered alphabetically by the username of the caregiver.
    - Then, output the vaccine name and number of available doses for that vaccine separated by a space.
        - For example if we had 3 available caregivers (c1, c2, c3) and 3 available vaccines (covid, flu, hpv), the output of "**search_caregiver_schedule 12-03-2023**" should be:

          ```
          c1
          c2
          c3
          covid 2
          flu 5
          hpv 3
          ```

            - **Do not include any other formatting (punctuation, titles, etc). The output should look exactly as above.**
    - If no user is logged in, print "Please login first"
    - For all other errors, print "Please try again"
- reserve <date> <vaccine>
    - Patients perform this operation to reserve an appointment.
    - Caregivers can only see a maximum of one patient per day, meaning that if the reservation went through, the caregiver is no longer available for that date.

    - If there are available caregivers, choose the caregiver by alphabetical order and print “Appointment ID {appointment_id}, Caregiver username {username}”.
        - For example, the output of “**reserve 12-03-2023 Covid**” should be:
          ```
          Appointment ID 1, Caregiver username c1
          ```
    - If no caregiver is available, print “No caregiver is available” and return.
    - If not enough vaccine doses are available, print "Not enough available doses" and return.
    - If no user is logged in, print “Please login first” and return.
    - If the current user logged in is not a patient, print “Please login as a patient” and return.
    - For all other errors, print "Please try again".
- show_appointments
    - Output the scheduled appointments for the current user (both patients and caregivers).
    - For caregivers, you should print the appointment ID, vaccine name, date, and patient name. Order by the appointment ID. Separate each attribute with a space.
        - For example, if caregiver c1 is logged in, the output of “**show_appointments**” should look like:
          ```
          1 covid 12-03-2023 p1
          3 flu 12-05-2023 p3
          ```
    - For patients, you should print the appointment ID, vaccine name, date, and caregiver name. Order by the appointment ID. Separate each attribute with a space.
        - For example, if patient p1 is logged in, the output of “**show_appointments**” should look like:
          ```
          1 covid 12-03-2023 c1
          2 hpv 12-04-2023 c2
          ```
    - If no user is logged in, print “Please login first”
    - For all other errors, print "Please try again"
- Logout
    - If not logged in, you should print “Please login first”. Otherwise, print “Successfully logged out”.
    - For all other errors, print "Please try again".

Deliverables for Part 2

Due: **Friday, December 1st at 11:00pm**

When you're finished, please turn in the entire repository by compressing the project folder into a zip file, then uploading it on Gradescope.

NOTE: Gradescope has a known error where some files will not upload "Server responded with 0 code." If this happens you may use the github upload option.

Grading

Your grade for this homework will be worth 100 points, divided as:

- Setup (5 points)
    - Finish setup through step 2.4 and upload your verification to Gradescope

- Part 1 (45 points)
    - Design (15 points)

        Your database design including the files design.pdf and create.sql

    - Implementation (30 points)

        Your working implementation of the Part 1 functions:

        create_patient, login_patient

- Part 2 (55 points)
    - Implementation (55 points)

        The remainder of the functions for Patient:
        search_caregiver_schedule, reserve, show_appointments, logout

Additionally, you may receive up to 10 points of extra credit for implementing one of the options below

Optional Extra credit

You can do either one of the following extra tasks by the final due date for 10 extra credit points.

1. Add guidelines for strong passwords. In general, it is advisable that all passwords used to access any system should be strong. Add the following check to only allow strong passwords:
    a. At least 8 characters.
    b. A mixture of both uppercase and lowercase letters.
    c. A mixture of letters and numbers.
    d. Inclusion of at least one special character, from “!”, “@”, “#”, “?”.
2. Both caregivers and patients should be able to cancel an existing appointment. Implement the cancel operation for both caregivers and patients. Hint: both the patient’s schedule and the caregiver’s schedule should reflect the change when an appointment is canceled.

```
> cancel <appointment_id>
```

Common Questions

Q: My Azure subscription has been canceled?! What should I do?
A: Ask a course staff or post on Ed for more credits.

Q: My IDE is not recognizing my imports, what should I do?
A: You will likely need to set up source root for your IDE. Check the documentation for your specific IDE on how to set up source roots.

Q: I’m getting an error: “AttributeError: 'NoneType' object has no attribute 'cursor'”.
A: This is indicating that there is something wrong with your database connection setup. Verify that you have followed our instructions and read the error messages carefully.

Q: I’m setting a DB error when trying to create a caregiver for my setup.
A: Make sure you have created your tables on Azure.