Our database is on the account: rh3129

URL of web application: http://34.23.193.218:8111

Description: As described in Part 1 of the project, we implemented a Hospital Recommendation System. The premise of the project is that during COVID-19, resource shortages made it uncertain whether a given hospital could treat a person for their condition. Our project allows users (and hospital admin) to check whether a patient can be treated at a given hospital based on hospital resource information stored in the database and patient requirements. Based on these factors, the main functionality of the website is that it will output up to three recommended hospitals that can treat the patient, listed in order of distance from the patient's address (whether that be at a current hospital or their home). Distance is calculated using the geocoder API from Google.

Also, in addition to the description in Part 1, we added extra functionality to the hospital admin view of the website. Once logged in with an admin username and password, the admin can delete/add employees to their hospital, update resource inventory for their hospital, and also search for the number of a given resource at all hospitals in the database. Patients do not require a login, and their ssn/date/time is used as a primary key to store their hospital search. If they receive a recommendation to go to a different hospital, that recommendation is also stored using ssn/date/time as a foreign key in the 'refers' table, in case it is necessary to later query again for that patient visit.

Two Interesting Web Pages: One interesting web page is the "*Hospital Recommendation*" page, which is used for finding recommended hospitals for patients to go to. This page can be reached either by a patient at home or by an admin of a hospital. In the case of access by a patient, they simply enter in all the necessary information (ssn, address, condition, etc.) and they are shown the three nearest hospitals that have the resources and doctors to treat them. We implemented this functionality by first querying our database, looking for hospitals that have the resources required as well as the specific doctor specialization needed to treat their ailment. We then use the geocoder API to calculate the distance from the patient to each of the hospitals that are able to treat the patient and output the three closest hospitals. For the admin view, the functionality is largely the same however if the hospital they are currently at is able to treat the patient, instead of showing the three nearest recommendations, our application will output a line saying that the current hospital can treat the patient. If there are no hospitals that can treat the patient, either due to lack of resources or doctors, the application will output a line saying that at the moment there are no available hospitals to treat their condition.

Another interesting web page is the "*Delete Employee*" page because of the complex query logic necessary to decide which employees can be deleted by the currently logged in admin. The employee_id of the currently logged in admin is stored in a global variable, curr_id. The logged in admin is associated with a single hospital, so they should be able to delete any admin or doctor also associated with **their** hospital. However, the ‘doctor’ and ‘admin’ entities inherit from ‘employee’ with overlap and no coverage. This means that some admin are also doctors and vice versa. This also means that employees that are not admin nor doctors do not have a specific hospital, so the logged in admin can delete those generic employees as well. The page allows the logged in admin to delete simply by inputting the employee_id of the desired employee, so this complex query is used to figure out the valid list of ids that could be deleted by the currently logged in admin:

```
(SELECT employee_id from employee
EXCEPT (SELECT A.employee_id FROM admin_manages A FULL OUTER
JOIN doctor D ON A.employee_id=D.employee_id))
UNION
(SELECT A.employee_id FROM admin_manages A FULL OUTER JOIN
works_at W ON A.employee_id = W.employee_id WHERE (A.h_address =
:h_addr OR W.h_address = :h_addr)), {'h_addr': h_addr[0]})
```

If the admin attempts to delete an employee whose employee_id is not in the valid list or does not exist, a line is printed out to indicate this error.

AI Usage Description: We did not use AI in the creation of our web application.