Final Report

Prepared for:

Dr. Omer

Prepared by:

Solomon Wakhungu, 1IW190000

Ved Nigam, vxn190021

Samhit Nimmagadda, sxn200035

Ajay Rao, axr19019

Ryan Huggett, rjh160130

Alaan Sharif, aas190013

CS 4347.503

23 April 2023

**Table of Contents**

[**Introduction** 3](#_Toc133875209)

[**System Requirements** 4](#_Toc133875210)

[**Conceptual Design of the Database** 9](file:////Users/vednigam/Library/Mobile%20Documents/com~apple~CloudDocs/Desktop/Ved’s%20stuff/UTD/Spring2023/4347/group%20project/Final%20Report.docx#_Toc133875212)

[**Functional Dependencies and Database Normalization** 22](#_Toc133875214)

[**The Database System** 25](#_Toc133875215)

[**Additional Queries and Views** 28](#_Toc133875216)

[**User application interface** 31](#_Toc133875217)

[**Conclusions and Future Work** 32](#_Toc133875218)

References……………………………………………………………………………………………….33

[**Appendix** 35](#_Toc133875219)

# **Introduction**

This project was a group project focused on working with databases as well as doing some back-end work and connecting that with the databases to create an interactive hospital website that took a variety of information types and stored them in databases. Throughout the entirety of the project, we created different diagrams and blueprints, which consisted of constraints, requirements, relations, and SQL statements for the database before we later fully implemented the database and populated it, as well as made it cooperate with the website to perform a variety of actions which would return a result as well as make changes to the database attributes for different tables. Overall, we can categorize this project as building a hospital data management system and creating an interactive website that coordinates with the databases to store, display, and perform various other actions on specific information. This report starts with the system requirements first, then provides the conceptual design of the database, the logical database schema, the functional dependencies and normalization process for the database, an overview of the database system as a whole, additional queries and views that use the database, the user application interface design and functionalities, conclusion and ideas for future work, references, and a formal appendix.

# **System Requirements**

**Functional requirements:**

|  |
| --- |
| This is an initial set of login functional requirements; you may add more as needed.  1: The system will allow the user to log in.  2. The system will verify the username and password.  3. The system will not allow the user to log in with an invalid username or password.  4. The system will be able to remember usernames and passwords.  5. The system will allow users to create accounts.  6. The system will enable users to log out of their accounts  **Browsing Functional Requirements:**   1. The system allows receptionists to assign patients to a room. 2. The system allows receptionists to view available rooms. 3. The system allows receptionists to create patient appointments. 4. The system allows receptionists to update patient appointments. 5. The system allows receptionists to delete patient appointments. 6. The system allows receptionists to view patient appointments. 7. The system allows receptionists to check in a patient. 8. The system allows receptionists to check out a patient. 9. The system allows Billing Specialists to view billing information. 10. The system allows users to update account information. 11. The system allows doctors to prescribe medication to patients. 12. The system allows doctors to update all patient information. 13. The system allows doctors to access all patient information. 14. The system allows doctors to view lab reports. 15. The system allows ward boys to access limited patient information 16. The system allows ward boys to update limited patient information 17. The system allows nurses to access limited patient information 18. The system allows nurses to update limited patient information 19. The system allows lab assistants to view patient information. 20. The system allows lab assistants to generate/update lab reports. 21. The system allows lab assistants to view lab reports. 22. The system allows patients to view their personal information 23. The system allows patients to edit their personal information 24. The system allows patients to view their lab reports 25. The system allows patients to view their diagnosis 26. The system allows patients to view their room number   **Administrator Functional Requirements:**   1. The system allows record keepers to view all patient information. 2. The system allows record keepers to update all patient information. 3. The system allows record keepers to view room assignments. 4. The system allows record keepers to update room assignments. 5. The system allows record keepers to view metadata. 6. The system allows record keepers to update metadata. 7. The system allows record keepers to delete accounts. 8. The system allows record keepers to back up the database information. 9. The system allows record keepers to restore the database information. 10. The system allows record keepers to monitor database usage. 11. The system allows administrators to delete the database. 12. The system allows administrators to create the database. 13. The system allows administrators to view all patient information. 14. The system allows administrators to update all patient information. 15. The system allows administrators to view room assignments. 16. The system allows administrators to update room assignments. 17. The system allows administrators to view metadata. 18. The system allows administrators to update metadata. 19. The system allows administrators to delete accounts. 20. The system allows administrators to back up the database information. 21. The system allows administrators to restore the database information. 22. The system allows administrators to monitor database usage. |
|  |

**Non-functional requirements:**

|  |
| --- |
| Nonfunctional Requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability   1. The system should be available 24/7 2. The system should be flexible to accommodate schema changes 3. The system should not allow unauthorized users to access sensitive information 4. The system should be able to add patients without the database getting full or running out of ID numbers. 5. The system should be able to scale depending on the number of users. 6. The system shall shut down access to the HMS for all members except DBA/DBA office so that maintenance can be done when the system is compromised. 7. The system should store data securely 8. The system encrypts sensitive information. 9. The system should store data in a way that it should be interoperable. 10. The system can recover in case of failure 11. The system should allow for weekly backups 12. The system should have a quick response time 13. The system will refresh every 10 minutes. |

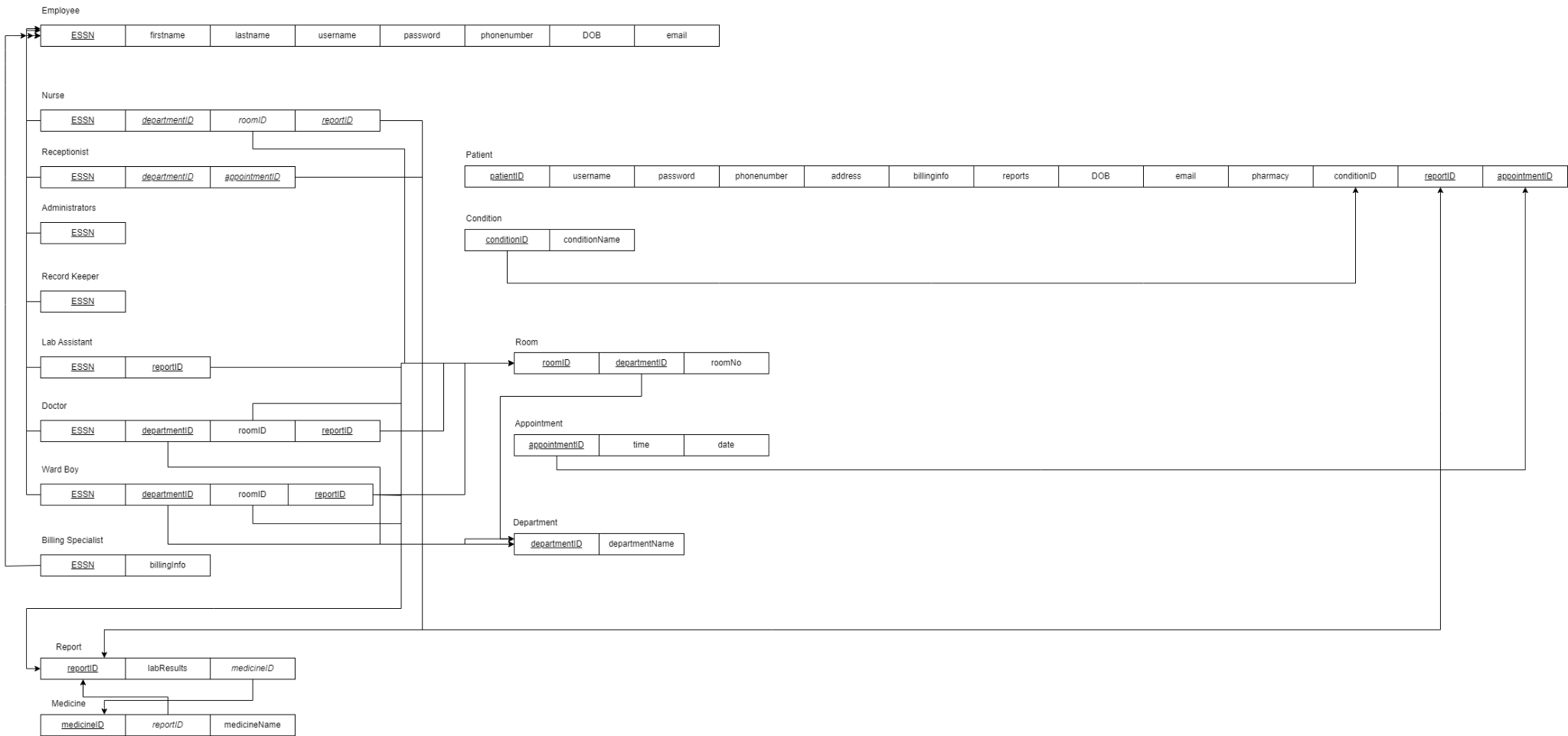
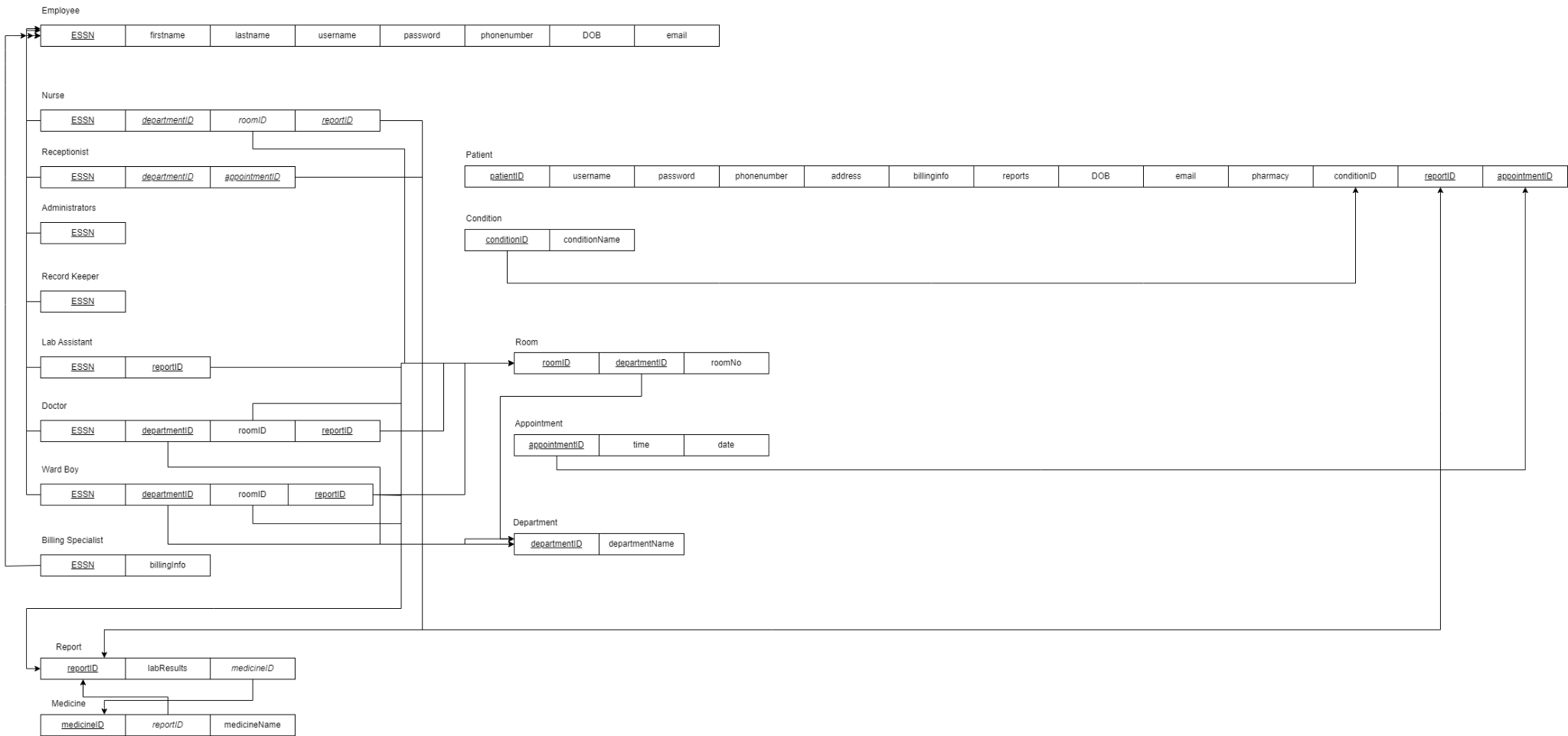
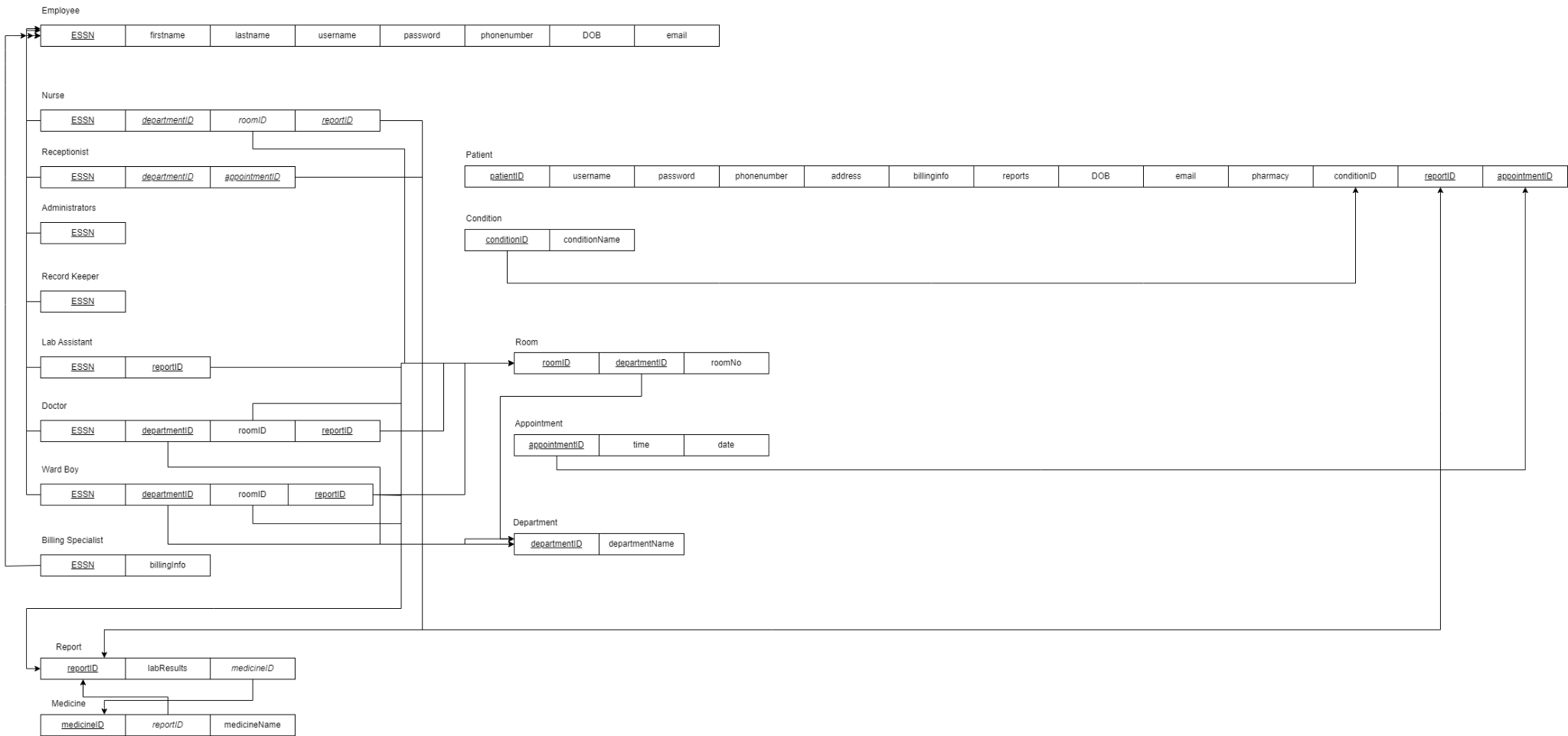
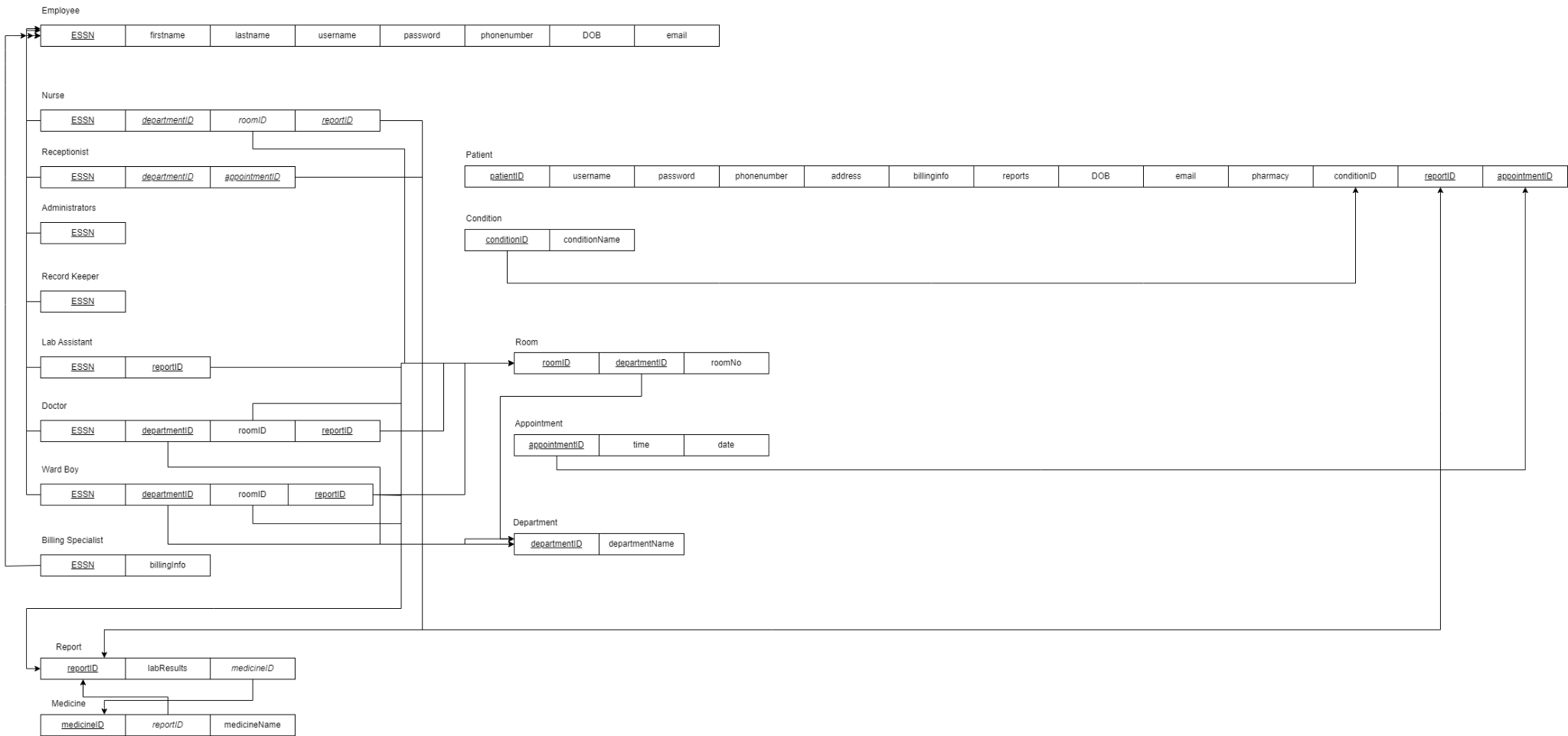
# 

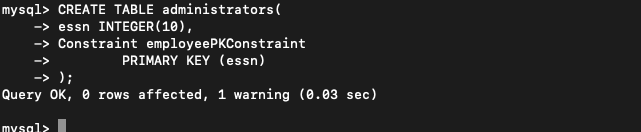
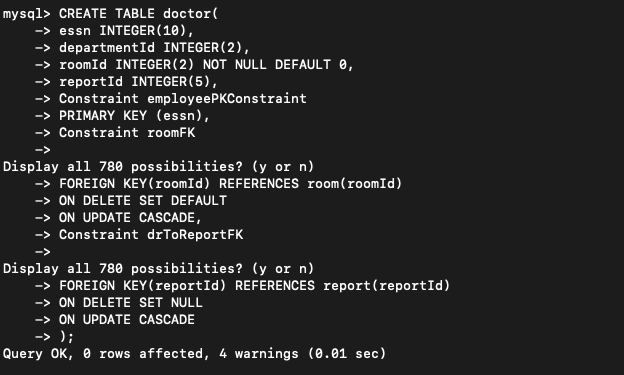
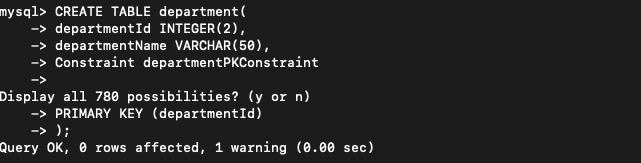
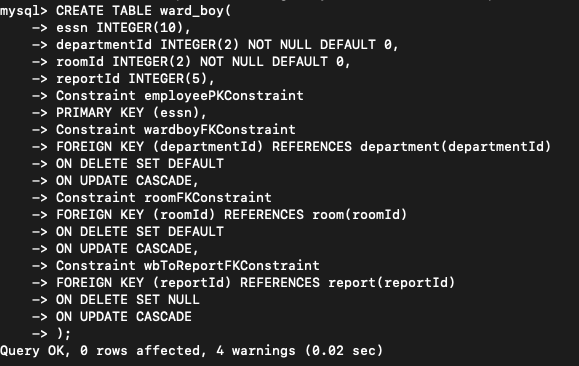
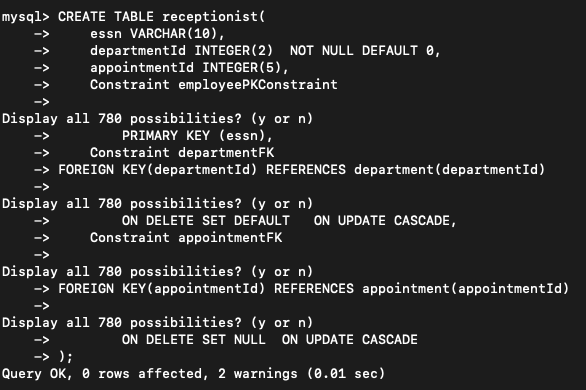
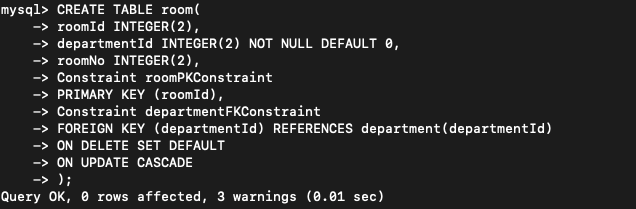
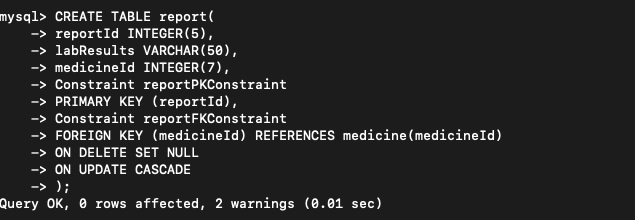
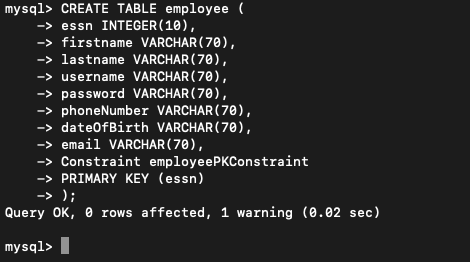
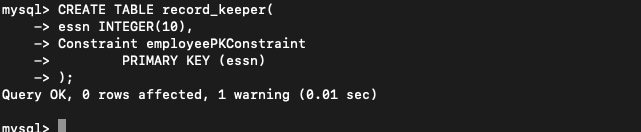
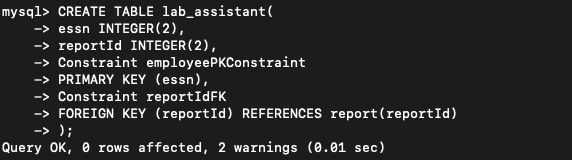
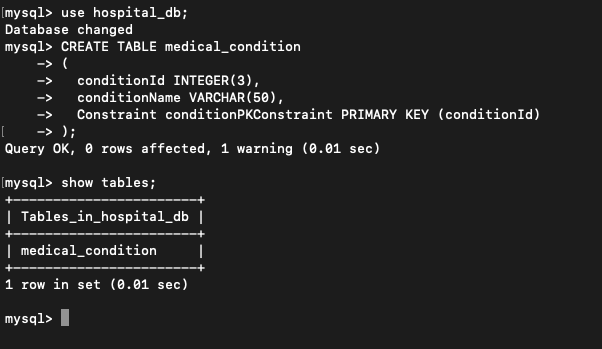
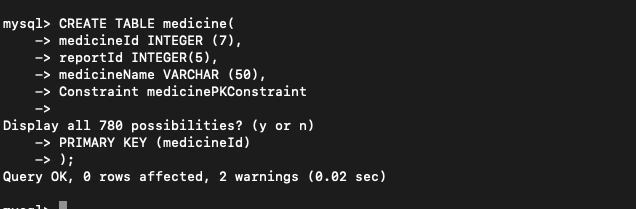
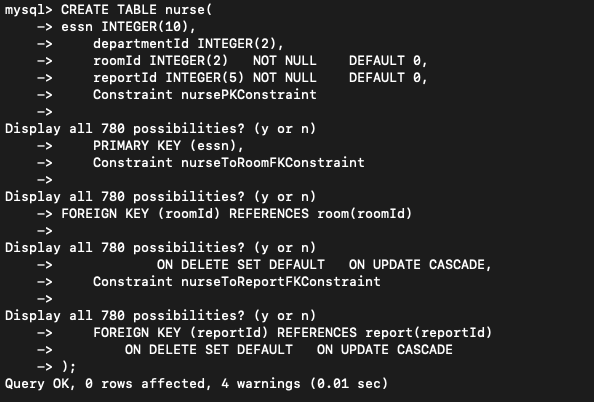
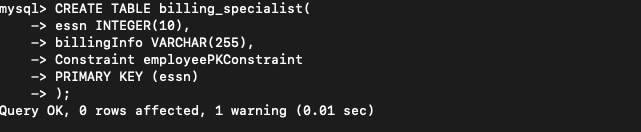
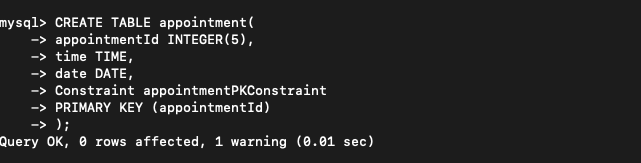
# **ER Diagram**

# **Conceptual Design of the Database**

**List of business rules and integrity constraints:**

|  |
| --- |
| **Integrity Constraints:**   1. Employee    1. The attribute ESSN is the primary key to Employee 2. Nurse:    1. The attribute ESSN is the foreign key to relation Employee    2. The attribute departmentID is the foreign key relation to Department    3. The attribute reportID is the foreign key relation to Report    4. The attribute roomID is the foreign key relation to Room 3. Receptionist    1. The attribute ESSN is the foreign key to relation Employee    2. The attribute departmentID is the foreign key relation to Department    3. The attribute appointmentID is the foreign key relation to Appointment 4. Administrator    1. The attribute ESSN is the foreign key to relation Employee 5. Record Keeper    1. The attribute ESSN is the foreign key to relation Employee    2. The attribute reportID is the foreign key relation to Report 6. Lab Assistant    1. The attribute ESSN is the foreign key to relation Employee    2. The attribute reportID is the foreign key relation to Report 7. Doctor    1. The attribute ESSN is the foreign key to relation Employee    2. The attribute departmentID is the foreign key relation to Department    3. The attribute roomID is the foreign key relation to Room    4. The attribute reportID is the foreign key relation to Report 8. Ward boy    1. The attribute ESSN is the foreign key to relation Employee    2. The attribute departmentID is the foreign key relation to Department    3. The attribute roomID is the foreign key relation to Room    4. The attribute reportID is the foreign key relation to Report 9. Patient    1. The patientID is the primary key to Patient    2. The roomID is the foreign key to relation Room    3. The attribute reportID is the foreign key relation to Report    4. The attribute conditionID is the foreign key relation to Condition    5. The attribute appointmentID is the foreign key relation to Appointment 10. Billing Specialist 11. The attribute ESSN is the foreign key to the relation Employee 12. Condition     1. The attribute conditionID is the primary key to Condition 13. Appointment     1. The attribute appointmentID is the primary key to relation Appointment 14. Room     1. The roomID is the primary key to Room     2. The departmentID is the foreign key to the relation Department 15. Department     1. The departmentID is the primary key to Department 16. Report     1. The reportID is a primary key for Report     2. The medicineID is the foreign key to Medicine 17. Medicine     1. The medicineID is a primary key for Medicine     2. The reportID is the foreign key to the relation Report   **Business Rules:**   1. Each room has only one patient 2. A doctor can only work in one department 3. A nurse can only work in one department 4. A billing specialist can only work in one department 5. A receptionist can only work in one department 6. An administrator can only work in one department 7. A record keeper can only work in one department 8. A ward boy can only work in one department 9. Patients can be treated by multiple departments 10. Patients can have multiple appointments 11. Patients can have multiple reports 12. Each employee can have only one role 13. Each employee can only work in one hospital 14. Each department can have multiple employees 15. Each hospital can have multiple employees 16. Each hospital can have multiple rooms 17. Each hospital can have multiple patients |
|  |

**SQL Statements Used to Construct Database**

****

# 

**Logical Database Schema**

# **Functional Dependencies and Database Normalization**

There are many functional dependencies for each relation in the section presented above (Logical Database Schema).

Let's start with the patient table. The primary key for this table is patientId, and there are several foreign key constraints. The functional dependencies for this table are as follows:

patientId → roomID, username, password, phoneNumber, address, pharmacy, reportID, billingInfo, appointmentId, dateOfBirth, email, conditionId

This functional dependency indicates that the patientId uniquely determines the values of all the other attributes in the table. For example, if we know the patientId, we can determine the roomID, username, password, phoneNumber, address, pharmacy, reportID, billingInfo, appointmentId, dateOfBirth, email, and conditionId for that patient.

Moving on to the employee table, which includes information on all the hospital's employees. The primary key for this table is essn. The functional dependencies for this table are as follows:

essn → firstname, lastname, username, password, phoneNumber, dateOfBirth, email

This functional dependency indicates that the essn uniquely determines the values of the firstname, lastname, username, password, phoneNumber, dateOfBirth, and email attributes in the table.

Next, let's look at the nurse table, which includes information on all the nurses in the hospital. The primary key for this table is essn. The functional dependencies for this table are as follows:

essn → departmentID, roomID, reportID

This functional dependency indicates that the essn uniquely determines the values of the departmentID, roomID, and reportID attributes in the table.

Moving on to the receptionist table, which includes information on all the receptionists in the hospital. The primary key for this table is essn. The functional dependencies for this table are as follows:

essn → departmentID, appointmentId

This functional dependency indicates that the essn uniquely determines the values of the departmentID and appointmentId attributes in the table.

Moving on to the doctor table, which includes information on all the doctors in the hospital. The primary key for this table is essn. The functional dependencies for this table are as follows:

essn → departmentID, roomID, reportID

The room table has the following functional dependencies:

roomID -> departmentID, roomNo

Here, the roomID uniquely identifies a room, and the departmentID and roomNo are dependent on it. This means that a room can only belong to one department, and each room has a unique room number.

The appointment table has no functional dependencies, as all of its attributes are unique and do not depend on any other attribute in the table.

The Condition table has no functional dependencies, as the conditionId and conditionName are both unique and do not depend on each other.

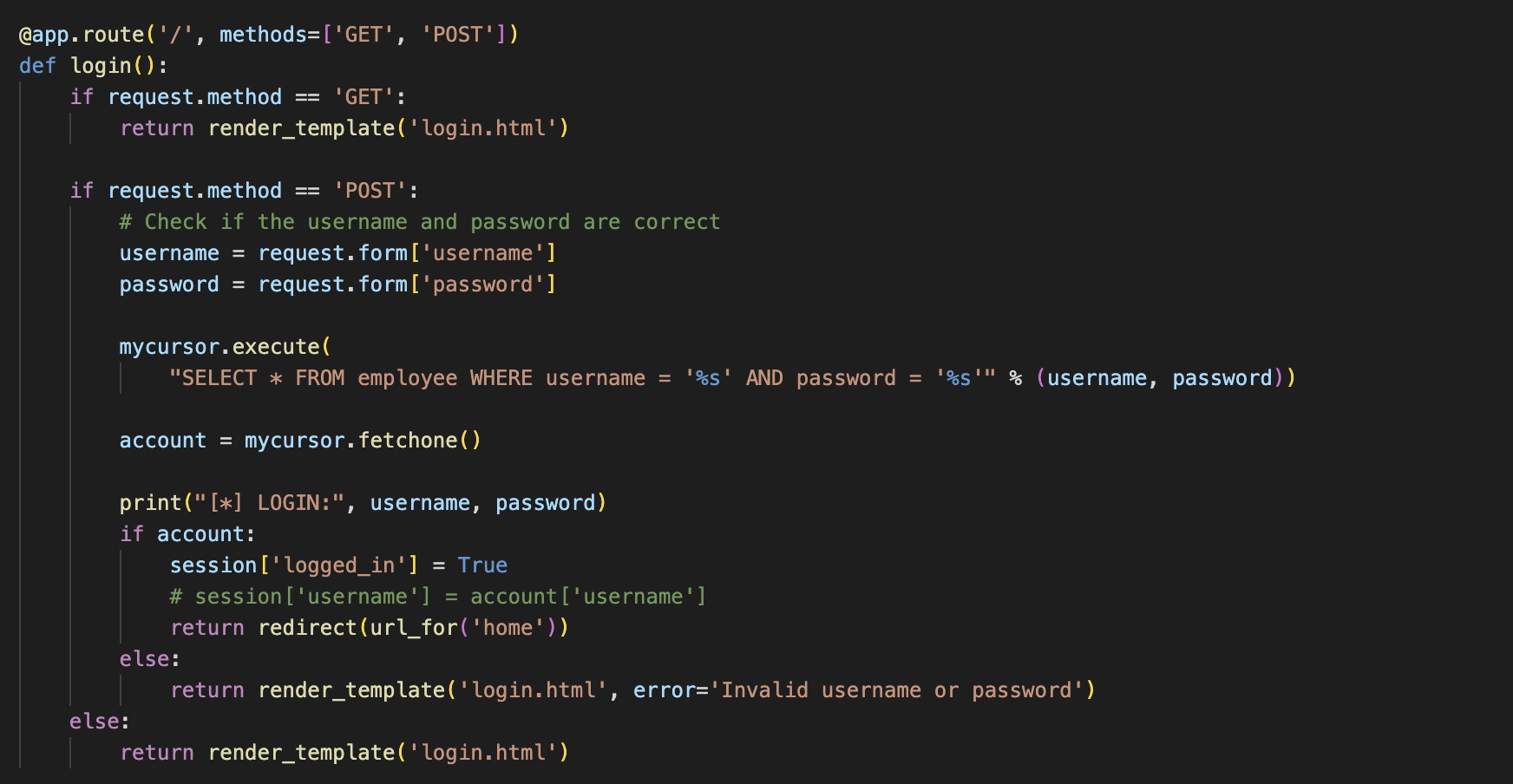
Finally, the medicine table has the following functional dependencies:

medicineId -> reportID, medicineName

The tables are in 3NF since there are no transitive dependencies. The primary keys and foreign keys have also been assigned to maintain referential integrity.he medicine uniquely identifies a medicine, and the reportID and medicineName are dependent on it. This means that each medicine is associated with a unique report, and has a unique name.

# 

# **The Database System**

Our application uses the Python Flask architecture to implement a web app that is connected to a local SQL database. The backend of the application is written in Python3 and the frontend of the application is written in Html. Flask implements a connection between the backend and the front end. After having all code on your local computer, along with the MySQL database having been downloaded, the developer will have to run the main.py file. This is the file that contains “routes” for the Flask app and tells the app which functions to call. The following is the Flask route that initiates the app:

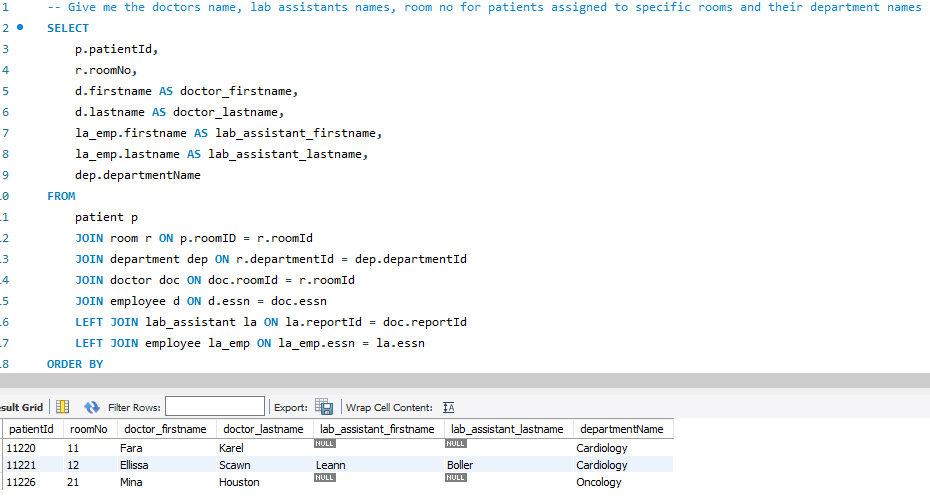
The app.route is set to ‘/’ which means that this is the first page the app should start with. Our application implemented a simple sign-in feature by running validation through out SQL database. The following is our connector.py file; this contains the connection credentials to the local database dump found on our GitHub:

After the log in route, the user will be presented with the possible operations that we implemented. These functions include changing report status, displaying patient information, viewing all doctors, assigning nurses to rooms, creating appointments, viewing all nurses, and any other function we develop. Our prototype app is developed for possible functions doctors and receptionists may perform, so our routes are specific to doctor and receptionist. The following are the routes for doctor and receptionist:

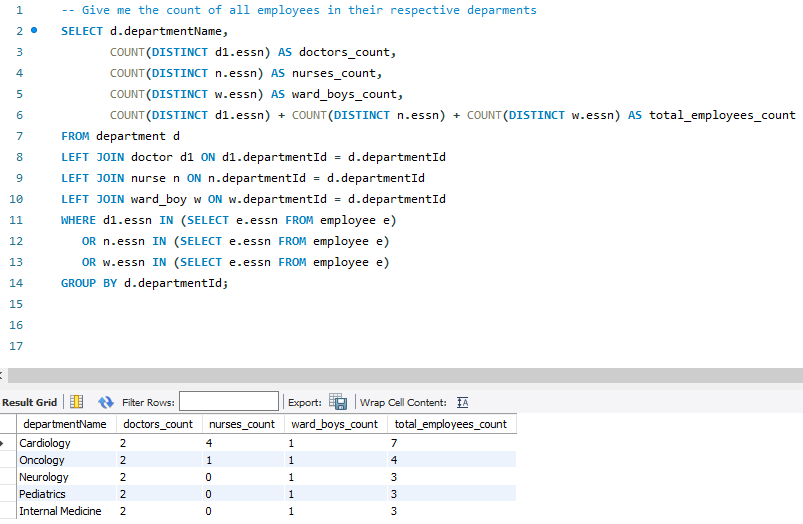


Each route renders an HTML template. Each route also contains switch/if statements for the option that is clicked. We have separate doctor.py and receptionist.py files that contain the backend for what each function does. For example, when option 1 is called in the doctor route, this corresponds to option 1 in the dictionary for the doctor.py file. In our case, option 1 in the doctor.py file is to change the report status; so, the route will render the template for the change report status function along with establishing a connection between the Html frontend and the Python backend. For the functions that require input from the user, the Html files render input forms accordingly.

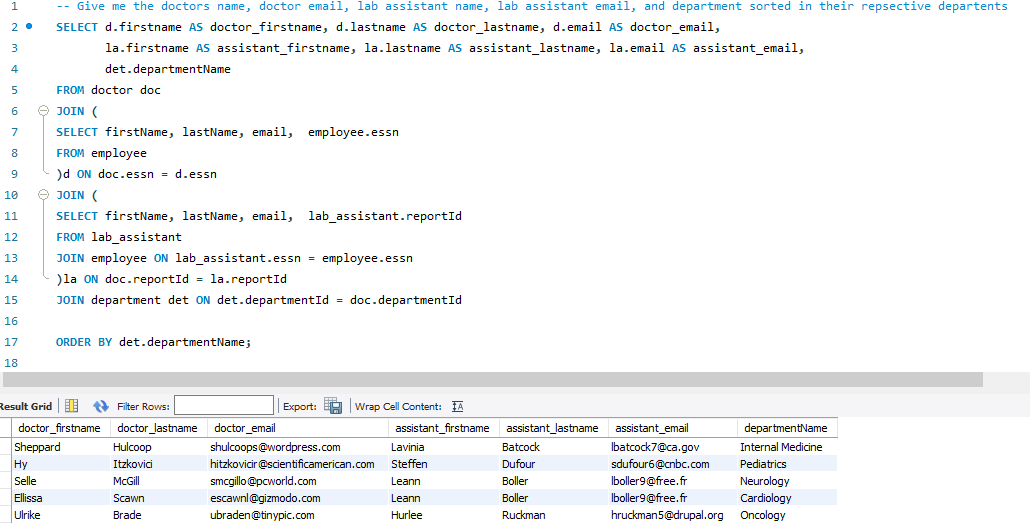
# **Additional Queries and Views**

In the complex query below it is selecting the doctors first and last name, their assistants first and last name, the room number of the patients and the department name associated with the room number.

This SQL statement retrieves information from multiple tables: patient, room, department, doctor, employee, and lab\_assistant. It joins these tables using various key relationships and selects specific columns from each table. The SELECT clause lists the columns to be returned in the result set, including patientId and roomNo from patient and room tables, respectively. The first and last names of the doctor and lab assistant are retrieved from the employee table using the essn foreign key relationships between the doctor, lab\_assistant and employee tables. The departmentName column is retrieved from the department table. The FROM clause specifies the tables to be queried and joins them using the appropriate foreign keys: patient joins with room, room joins with department, department joins with doctor, and doctor joins with employee. Additionally, the lab\_assistant and lab\_assistant tables are joined to the employee table using LEFT JOIN, allowing for cases where there is no lab assistant assigned to a doctor. The result set is sorted in ascending order by patientId using the ORDER BY clause.

For the 2nd sql statement we are requesting the count of doctors, ward boys, and nurses along with total employee headcount in each department. This SQL statement selects the department name, the count of distinct doctors, nurses, and ward boys in each department, and the total count of employees in each department. It then performs a left join on the department table with the doctor, nurse, and ward boy tables using the department id as the join key. The WHERE clause restricts the results to only those departments that have at least one employee who exists in the employee table. Finally, the results are grouped by department id and ordered by department id.

Lastly, This statement selects the first name, last name, and email of doctors, lab assistants, and the department name they are associated with. It uses a nested query to retrieve the first name, last name, and email of employees who are lab assistants, and joins it with the doctor table on their shared report ID. The employee table is also joined on the doctor table to retrieve the first name, last name, and email of doctors. The department table is joined to the doctor table to retrieve the department name. The final result set is ordered by department name.



# **User application interface**

For the development process of our user interface, we used Flask. It connects a Html frontend to a Python backend to render a web application. Our Python backend calls a local MySQL server which contains all relevant data to the hospital database we designed. The user will first be prompted to log in with their username and password, which if incorrect would inform them that there was either an incorrect username or password and to try again. Once they log in, they will be prompted to the home page where they can select from a variety of actions to do which are split into a doctor and receptionist entity, each which has their own respective functions. For the doctor entity, they can select to change the report status and view the patient info. For the receptionist entity, the user can select to view all doctors, assign a nurse to a room, create an appointment, and view all nurses. Based on what the user selects, the appropriate view is pulled up or the needed changes are made to the required database tables which will satisfy the actions requested by the user. The functions for the doctor and receptionist entity are implemented in SQL since we created tables for each entity and each table contained whatever attributes that were correlated with either doctor or receptionist in this case and actions were performed on the attributes and tables based on what function the user selected.

# **Graphical user interface, application, website Description automatically generated**

# **Conclusions and Future Work**

The objective of this project was to design and implement a hospital management database using a database management system, such as MySQL. To accomplish this, we first analyzed the assignment requirements and categorized the different departments and individuals in each department. We then identified the foreign keys required to establish the necessary relationships between tables, enabling us to generate statements and views based on the data. Our initial design phase was well-structured and effectively linked the various elements of the system together.

However, during the coding phase, we encountered several challenges while working with Flask, Python, and HTML to develop both the front end and back end of the system. Given that most of us had limited prior experience with these softwares, we had to rely on online resources, including Flask's documentation, to establish connectors with MySQL and implement the necessary features. We also faced initial difficulties with integrating the backend with HTML and had to learn on the go. Nonetheless, we were able to overcome these challenges with hard work and collaboration.

As for future work, we could explore expanding the system's functionalities to include additional features, such as automated billing and scheduling systems. We could also consider integrating the database with machine learning algorithms to improve patient care and resource allocation. Additionally, we could focus on enhancing the user interface to make it more intuitive and user-friendly, as well as improving data accuracy and security. Finally, we could explore implementing the system in a cloud-based environment to enable remote access and increase scalability.

**References**

“How to Connect Python with SQL Database?” *GeeksforGeeks*, GeeksforGeeks, 22 Aug. 2022, https://www.geeksforgeeks.org/how-to-connect-python-with-sql-database/#.

“Random Data Generator and API Mocking Tool: JSON / CSV / SQL / Excel.” *Mockaroo*, https://mockaroo.com/.

“SQL Syntax Checker, SQL Syntax Validator.” *SQL Syntax Checker And SQL Syntax Validator*, https://www.coderstool.com/sql-syntax-checker.

*SQL Injection*, https://www.w3schools.com/sql/sql\_injection.asp.

“HTML Value Attribute.” *HTML Button Value Attribute*, https://www.w3schools.com/tags/att\_button\_value.asp.

Nishant, Vinayak. “Flask MySQL - Setting up a Flask and Mysql Database Connection.” *AskPython*, 6 Aug. 2022, https://www.askpython.com/python-modules/flask/flask-mysql-database.

Gaurav, I am working as Software Development Engineer with Intelligrape Software. I am currently working on Groovy and Grails. Working here is a great fun. “Creating a New Database from a Mysql Dump File.” *TO THE NEW BLOG*, https://www.tothenew.com/blog/creating-a-new-database-from-a-mysql-dump-file/.

“#73 Python Database Connection | MySQL.” *YouTube*, YouTube, 4 Nov. 2018, https://www.youtube.com/watch?t=432&v=vR5utJvN4JY&feature=youtu.be&ab\_channel=Telusko. Accessed 1 May 2023.

# **Appendix**

Look at the uploaded zipped file