Project Title: Hospital Database

Group Details

Group Name: Flaming Tigers

Group Members:

[redacted] [redacted] ([redacted])
[redacted] [redacted] ([redacted])
[redacted] [redacted] ([redacted])
[redacted] [redacted] ([redacted])
[redacted] [redacted] ([redacted])
[redacted] [redacted] ([redacted])

Project Details

Purpose

The goal of this database is to create a way of storing and keeping track of the millions of records needed to keep modern hospitals running efficiently. Due to how organized hospitals are, and how departmentalized they are; it is essential to keep track of everything that occurs so that each employee can do their job efficiently.

In order to do this, our database will store entries on patients, rooms in a hospital, departments for a hospital, hospitals, employees, positions for each employee, as well as medical records to tie into each patient and doctor. The benefits of this are to keep track of every piece of information regarding any quantity of hospitals so that they can run optimally, particularly by ensuring that any necessary information is properly recorded for when it is needed.

Potential Customers & Users

The potential customers and users for this database would involve hospitals and hospital chains so they can increase efficiency in their multi-million, sometimes billion-dollar facilities. The importance of ensuring their massive healthcare facility also requires an immense infrastructure of not only physical form but also digital form. The process of running the hospital should be as smooth and seamless as possible, and the transmission of information from doctor to doctor, patient, to nurse, and so forth is critical to the continued smooth operation of the hospital.

Data Requirements

The scope of a real hospital is massive, and it requires nearly a hundred various sequel tables to run optimally. This being a group project for Intro to Database Systems Design, the scope is being limited to the most essential topics and will contain around one dozen tables to avoid unnecessary complexity for a simulated project and the time restrictions associated with the length of the semester that this course is a part of.

To begin, the primary data requirements will be, from a general standpoint, as follows.

Patient

The most integral part of a hospital itself is its patients, as that is why the hospital exists. We will be storing information regarding assigned physicians, a primary nurse or employee, personal info, insurance information, a list of known conditions, a list of known attributes (such as medication), as well as a ledger containing all known records associated with the patient (such as checking in or out of a hospital, or getting treatment, or getting prescribed medication, etc).

Room

Due to the physical nature of a hospital, it is necessary to consider the physical space of the hospital itself. Rooms will store status and information about them such as its occupation, which nurse is assigned to the room, which patient is assigned to the room, as well as what department the room is connected to (which, the department, in turn, has a reference to the hospital it resides in). On top of this, it would also make sense to store information about the technical capabilities and accommodations of each room so it can be optimally chosen when a new, unoccupied room is necessary to house a patient (such as a ventilator or close proximity to a CT machine)

Department

Each hospital will have numerous departments that specialize in specific conditions and specific forms of treatment. This will store those types of information, in addition to various physicians and nurses that are assigned to the department. On top of this, it will also have a reference to the hospital it resides in; as we can have multiple hospitals in each chain, but each hospital can have its own department focusing on, let’s say, cancer or some other condition. A list of rooms will also be stored in this as a foreign key to quickly identify available rooms that already are applicable to the department’s focus.

Hospital

Hospitals themselves contain numerous attributes, departments, employees, and patients. This table will act as the root for all of them, and be somewhat of the top node for all branches. The hospital table will include information about employees, physical addresses, capabilities, forms of treatment, and current patients.

Employee

The employee table is used to specify specific employment information regarding various individuals, in addition to referencing their various positions. It is possible for a single employee to be employed at multiple departments, in addition to multiple hospitals; so a list of positions will be specified, directly referencing a Physician or Nurse record which will be connected to specific departments, which will in turn be connected to specific hospitals. In addition to all of this, various personal information will be stored as a result of their employment.

Physician

The table to store information about a physician or doctor itself. This will include specific criteria such as their departments, specializations, education, and more. Most information that is generic is stored in the Employee record, while the Physician record that could be stored within it focuses more on the specific attributes of the physician itself. This will have a reference to any employee it is stored within.

Nurse

The same situation as with Physicians, except for Nurses. They will be assigned to specific patients, rooms, and departments. Nurses are usually helpers to doctors themselves, so while some information will be included about their specializations; they ultimately are very generic when stored in the database because of their wide, yet unspecific range of tasks.

Record

This will be used to store all medical records or “transactions” between a patient and some other entity (such as a hospital, doctor, insurance provider, etc). These are less specific or structured entries, and moreso a log of what occurs. Checking in to a hospital, checking out, being prescribed medication, an insurance payment being approved, undergoing a treatment, etc. These records will all be linked within the records field of each patient. On top of this, it is possible (and somewhat likely) that some records originate from hospitals outside the system; so the importance of this table being relatively unlinked to anything else beyond the patient is essential, as while it may make sense to link the Physician; it is possible that they won’t be in the system. Due to this, beyond the patient itself, everything else will be generic identifiers that are not necessarily in a linked format to any other portion of the database but will be in a format that allows them to be easily searched and estimate connections.

These are not all of the specific fields that each table will consist of, but it is the generic data requirements and an overview of them necessary to complete our Hospital Database. This will likely be adjusted as time goes on depending on the best needs of the project itself, not to mention specific identifiers and primary keys will be determined to best link the various tables of information.

Functional Requirements

The functional requirements of the database itself will need to complement that of what is stored in the database. To begin with, a generic interface is necessary to directly search the database and retrieve information from various tables.

In addition to this, additional more specialized interfaces will be created to bring out the true functionality of the database. To begin, a simple check-in and check-out interface will be useful to quickly and easily add corresponding records to the database and quickly identify a room to assign a patient to, which could just be a waiting room. On top of this, an interface should be constructed for Physicians and Nurses that quickly identify the next objective of those employees based on information known by the database. A physician will have multiple patients it is attending to, same with nurses; and the employee interface will automatically direct them to where they need to go to fulfill these objectives. Each of the components of this interface will also be separated into its own micro-interface, allowing direct access to something such as viewing the status of various rooms; but ultimately will be consolidated into the employee/patient management interface that is described above.

On top of this, an interface will be necessary to import records into the database itself. This will be useful for other hospitals to share their records with our own database so they may properly transfer patient records to go with a patient visiting different hospitals that are maintained by different databases.

Finally, a homepage for the hospital (or hospital chain) will be required for future or possible patients to research the hospital itself, or query the database to obtain contact information for their assigned doctor.

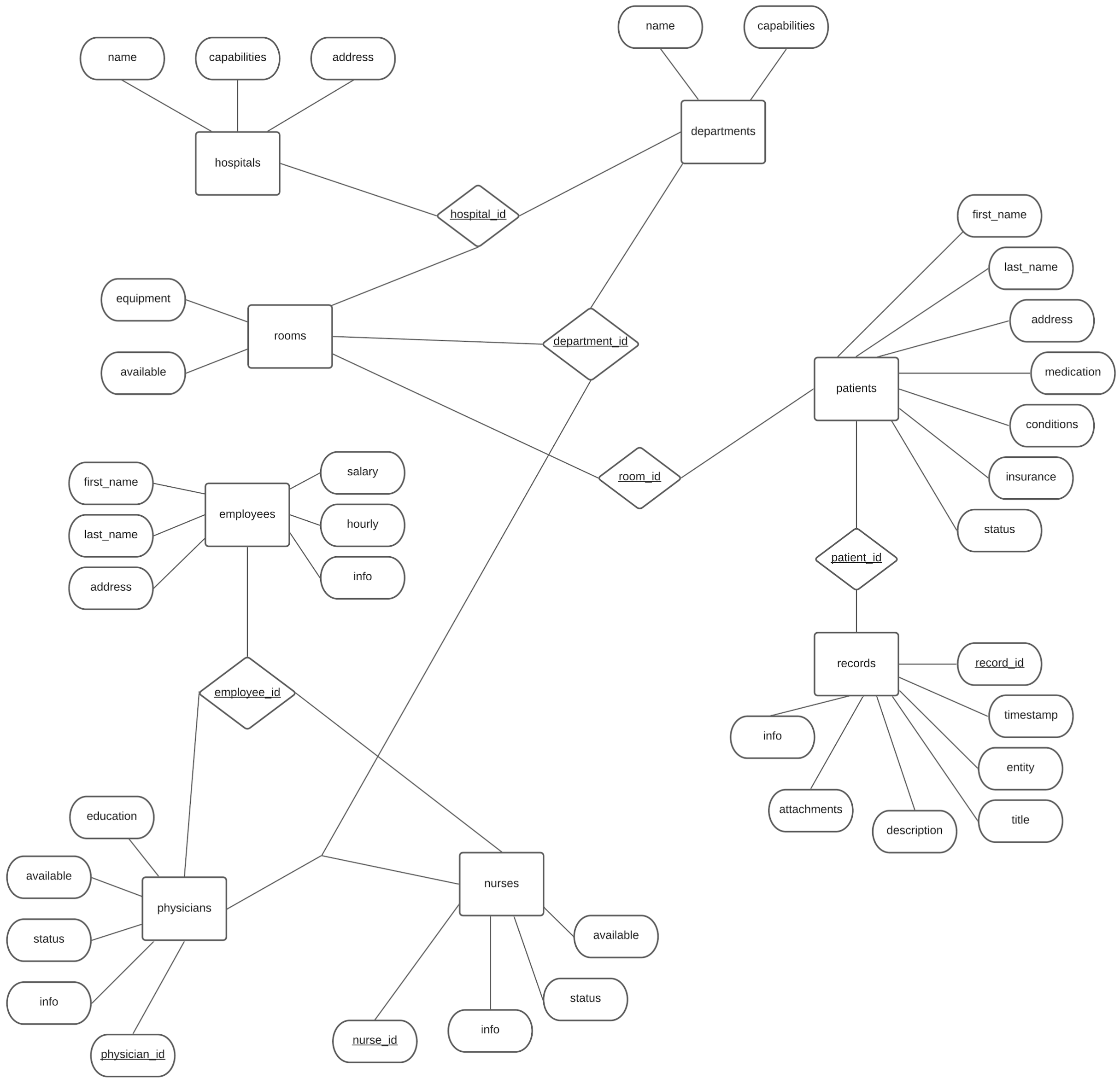
With all of this in mind, various other essential systems will need to be constructed such as employee authentication and data protection to prevent very private and confidential patient information from being revealed to those who are not authorized. Confidentiality of information is unlikely to be focused on for this project, as it is nonetheless important; our timeline for the project itself is limited, and it is not very important to Introduction to Database Systems Design.

Project Design

Conceptual Database Design

There are various relationships between entries in our Hospital Database. From connecting rooms to departments, to records being attached to patients, to recognizing which nurses and physicians belong to which departments of each hospital, and more. These relationships are identified below.

Entity-Relationship Diagram



Additional Information

The diagram above specifies the tables, which are represented by boxes, and the column names of each. The circular figures represent normal column names, while the diamonds represent shared keys that will be implemented via the use of foreign keys. Any box with underlined text represents a primary key.

Full Resolution

https://raw.githubusercontent.com/cseitz/intro-to-db-project1/main/EntityRelationshipDiagram.png

Insight into Entity Relationships

As added context, the relationships between Employees and Nurses/Physicians will be examined, in addition to the relationship between Patients and Rooms. While there are many more relationships in the database, these two are somewhat complicated and will be examined to provide insight into the other relationships, as they all function extremely similarly.

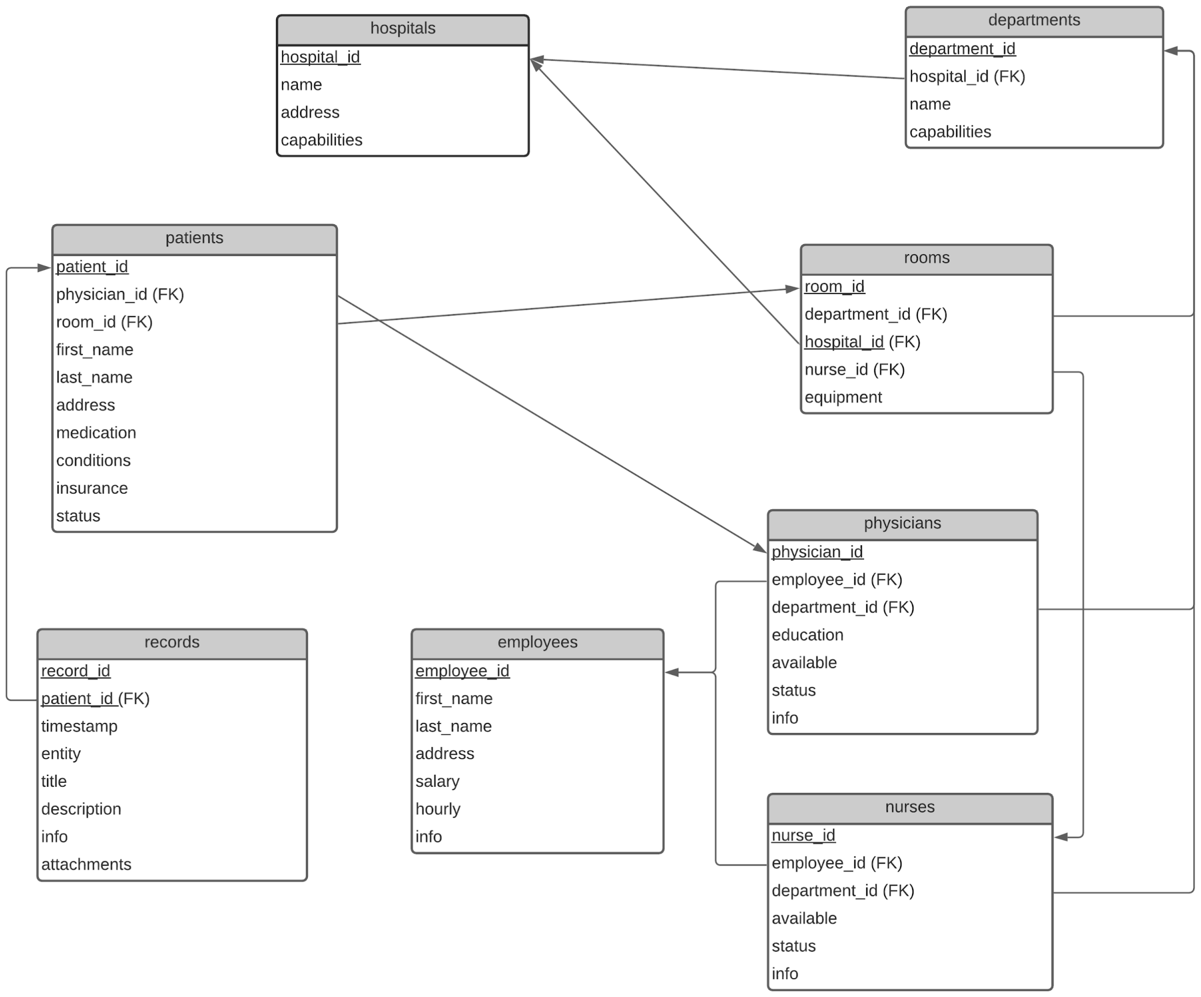
Employees and Nurses/Physicians

The relationship between Employees and Nurses/Physicians is very particular and somewhat unique. Employees use Nurses and Physicians as roles, which link them to the rest of the database. The reason why this is this case is that both roles also can be linked to specific positions, such as a department. Essentially, there is the possibility that a specific employee will be a nurse that is on-call for both Urgent Care and General Care, while a very experienced physician may actually be employed at multiple hospitals and be the head of multiple departments at multiple hospitals. Due to this, the essential information about their employment is stored in the Employees table, which contains their name, address, salary/hourly pay, and a JSON info section for miscellaneous storage of data, as this is a hospital and it needs to adapt. (This strategy has been employed in multiple places. If the table does not satisfy an unplanned need for the database, the JSON fields can be used.) In order to actually get employees from a department or hospital, one must get all nurses and physicians from departments at the hospital, and then use their employee identifier foreign key to get the employees. Yes, this is somewhat complicated; but it ensures proper relationships can be maintained, and immense flexibility of the database when storing information on a hospital's employees and their positions/roles.

Patients and Rooms

The relationship between Patients and Rooms is what links patients to a hospital if they are currently staying at a hospital. Patient records should be kept for all people who visit a hospital or simply have a hospital in the database be their primary hospital. The medical community particularly shares these records among multiple, even competing hospital chains for the sake of ensuring that all data is known about a patient in the event of an emergency. Due to this, when a patient checks in to a hospital they will be assigned a room that they will be stationed in. The reason why this is a reference to a room, and not the room referencing a patient is because some rooms may, and likely will have an occupancy greater than one. Even now during COVID-19, there are so few rooms available that extra beds are placed in rooms and multiple patients will be in the same room. Through this, a nurse can also be assigned to a specific room, and in turn, also be assigned to patients assigned to the room through the reference to the room.

Relational Schema Diagram



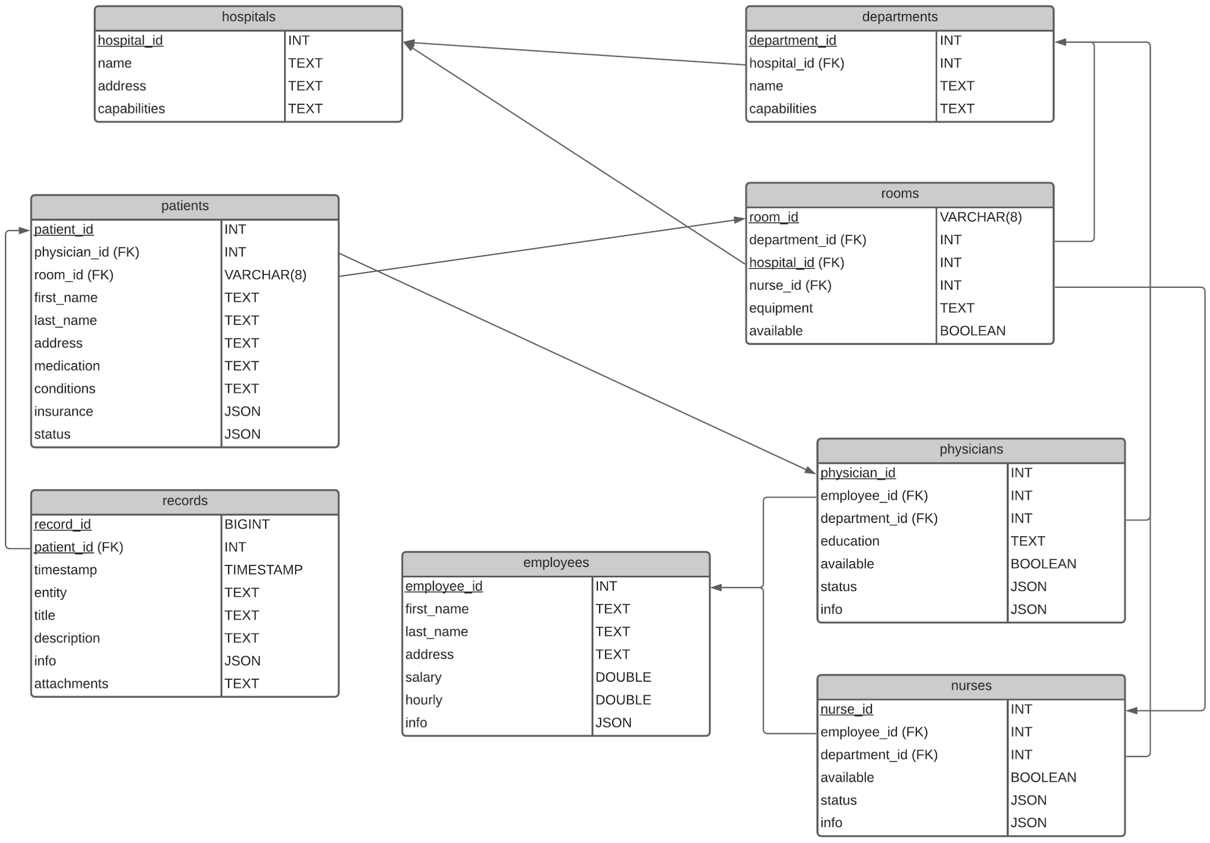
Additional Information

The diagram above converts the Entity-Relationship diagram into a more table-oriented Relational Schema Diagram, and foreign keys are identified to connect non-redundant relationships.

Full Resolution

https://raw.githubusercontent.com/cseitz/intro-to-db-project1/main/RelationalSchemaDiagram.png

Schema Diagram



Additional Information

The diagram above finalizes the conversion of the Entity-Relationship to the Schema Diagram which can be used to structure the Database. Typing is included for all columns in all tables. Some JSON attributes have been added to provide additional flexibility during production for data storage that may not be as critical (thus, not included in the database planning), yet still necessary; in addition to flexible attributes such as where a nurse is located, or if a physician is currently performing surgery. These things are very different, can require many different pieces of information, and simply cannot be fulfilled with a structured database without going to a level of complexity that is beyond the scope of this Introduction to Database Systems and Design Project. Thus, a JSON attribute is extremely versatile for this.

Additionally, I'll briefly explain a few abnormal datatypes.

- Records have a BIGINT auto-incrementing identifier because they record nearly every action that is performed on a patient, such as being prescribed medication, checking in and out of a hospital, etc. Due to this, it is extremely likely that the database would run out of identifiers if the datatype was just a normal integer.
- Rooms have a VarChar identifier, with a length of 8. The reasoning behind this is because rooms will be specifically set up for a hospital, based on their available floor space. There is the possibility that a sub-building name needs to be prefixed, or rooms are separated into A, B, C, etc. It is essential that room numbers reflect what is located next to their door frame, so using a Variable Character String makes sense.

Full Resolution

https://raw.githubusercontent.com/cseitz/intro-to-db-project1/main/SchemaDiagram.png

Project Implementation

Introduction

The project was implemented by using Node.js, Express, NGINX, Vue, and MySQL. Node.js and Express form the backend of the web application and are used to execute queries to the database, while Vue is used as a frontend framework for developing the website. NGINX is used as the routing software for the server itself.

Features

Sadly, due to the time constraints of the project, numerous features were not able to be implemented in time. Considering how a hospital revolves around its patients, that was the main focus of what was implemented. Nearly all features involving patients have been implemented, while a few other areas of the website may be lacking. With that in mind, the project provides enough functionality to make a somewhat usable hospital database that is future-proof and forward-thinking.

Usage of MySQL

SQL was used in various ways to make the website and database functions. To begin, there is a clear usage of SELECT for the search boxes on each webpage. From selecting hospitals to finding patients by name, to finding records from a specific patient; SELECT was used. On top of this, JOINs were used multiple times, such as when fetching nurses and physicians. Due to nurses and physicians being roles, they did not have essential information for humans to identify who the person is, rather just an identifier for their employee record. Due to this, any time when fetching nurses or physicians, a join is made with their matching employee record so things like their name can be returned.

While there was not enough time to add creation, updating, or deleting to anything except patients and patient records, the functionality exists. One can go and register a new patient from the patient search screen, which uses an INSERT query. The same can also be done with records by importing records directly on a patient's profile. As with insertion, both can be updated and deleted just the same.

In addition to this, a program was also constructed to generate sample data for our hospital. You can find it in the repository under "data-generator", and it is how we generated thousands of unique and properly linked records to be inserted into the database. We can generate thousands of realistic entries for all of our tables in a matter of milliseconds.

If this project was to continue being developed in the future, there remains much more room to make improvements and additions to the database. For example, this project is lacking authentication, and focused on database operations. Giving a password field to employees then allowing them to log in would be essential to keeping the database secure. In addition to this, more in-depth relationships can be created that would, for example, automatically update the "Conditions" field on a patient when a record is added stating they were diagnosed with a condition or disorder. There are thousands of operations needed to keep a real hospital, let alone a chain of hospitals running optimally, and this project only scratched the surface of what real hospital databases consist of.

Deployment

This project was deployed to Chris Seitz's personal server, hosted by DigitalOcean. MySQL Server 5.7.32 is also on this server, which is how the database is hosted. PHPMyAdmin is also in use as a direct interface to the database itself.

Because NGINX is used as the primary routing software for the server, which has some conflicts with the native meshing between Apache 2, PHP, and MySQL. Due to this, NGINX uses reverse proxies to forward to Apache 2, which is hosted on a non-standard, but firewall-protected port. This effectively makes the LEMP stack usable without switching the server to run off of Apache 2, as the LEMP stack is still running behind the scenes; just not on ports 80 and 443.

A live version of the project is available at https://db-project.cseitz.dev

Source Code

The source code for this project can be found at https://github.com/cseitz/intro-to-db-project1

Export of the MySQL database has been included within the source code within the directory named “export”. Please also refer to README.md within the root of the repository for instructions on running it on your own.

https://github.com/cseitz/intro-to-db-project1/blob/master/export.sql

Conclusions

This project was one of the biggest projects many of us have worked on before for our classes. While we didn’t have enough time for everything we wanted to do, this project was fun to work on and a good test of what we learned in the class.

Problems Faced

The biggest problem that we faced was time. During Task 1, we came up with the idea of making a hospital database. While this was an extraordinary idea for a database that presented various applications that we could implement… it was also really complicated and in-depth. We rarely had issues with the implementation of features, rather than having enough time to implement those features.

Additionally, due to the intentional scaling down of the scope for this project in Task 1, we also accidentally left some holes in the database design itself. For example, rooms cannot get who is in them because while a patient can be added to a room, the key on the patient itself only involves the room number; not the hospital id, which must be combined with the room number to uniquely identify a room. Nevertheless, this would rarely be a problem in practice as one needs to identify the room a patient is in will likely be on the hospital grounds, so whatever room number is returned will obviously be in relation to the specific hospital that someone knows the patient is located at. On top of this, if properly using patient records, one would see “Checked into Kent General Hospital” as a recent record which would be added during check-in. Sadly, we did not get a singular interface done in time for check-in, but it can be simulated by treating patients and creating records, both of which have been implemented.

Closing Remarks

Ultimately this project was huge in scope, and we got a considerable amount of it done. While we were not able to do everything we had planned, we learned a lot from doing this, not only by implementing features of the database but also from making mistakes in the process.

The source code for this project can be found at:

https://github.com/cseitz/intro-to-db-project1

Export of the 'hospitals' SQL database can be found at:

https://github.com/cseitz/intro-to-db-project1/blob/master/export.sql

A live site of the project can be found at:

https://db-project.cseitz.dev

In the event that live site is non-operational, email [redacted]

(I have many other things running on the server)