**Hospital Database Report**

Brendan Foley

Johns Hopkins University Advanced Academic Programs

Course Code: EN 605.641.81.FA22

Dr. Dar-Ning Kung

11/19/2022

Table of Contents

[1. Introduction 4](#_Toc119791043)

[1.1. Scope and Purpose of Document 4](#_Toc119791044)

[1.2. Project Objective 4](#_Toc119791045)

[2. System Requirements 4](#_Toc119791046)

[2.1. Hardware Requirements 4](#_Toc119791047)

[2.2. Software Requirements 4](#_Toc119791048)

[2.3. Functional Requirements 5](#_Toc119791049)

[2.4. Database Requirements 5](#_Toc119791050)

[3. Database Design Description 5](#_Toc119791051)

[3.1. Design Rationale 5](#_Toc119791052)

[3.2. E/R Model 6](#_Toc119791053)

[3.2.1. Entities 6](#_Toc119791054)

[3.2.2. Relationships 8](#_Toc119791055)

[3.2.3. E/R Diagram 9](#_Toc119791056)

[3.3. Relational Model 10](#_Toc119791057)

[3.3.1. Data Dictionary 10](#_Toc119791058)

[3.3.1.1. Department 10](#_Toc119791059)

[3.3.1.2. Doctor 10](#_Toc119791060)

[3.3.1.3. Specialty 11](#_Toc119791061)

[3.3.1.4. Accountant 11](#_Toc119791062)

[3.3.1.5. Visit\_has\_Doctor 11](#_Toc119791063)

[3.3.1.6. Prescription 12](#_Toc119791064)

[3.3.1.7. Medication 12](#_Toc119791065)

[3.3.1.8. Nurse 12](#_Toc119791066)

[3.3.1.9. Reference 13](#_Toc119791067)

[3.3.1.10. ReferenceType 13](#_Toc119791068)

[3.3.1.11. Patient 14](#_Toc119791069)

[3.3.1.12. Visit 14](#_Toc119791070)

[3.3.1.13. Room 15](#_Toc119791071)

[3.3.1.14. Bed 15](#_Toc119791072)

[3.3.2. Integrity Rules 15](#_Toc119791073)

[3.3.3. Operational Rules 15](#_Toc119791074)

[3.3.4. Operations 16](#_Toc119791075)

[3.4. Security 16](#_Toc119791076)

[3.5. Database Backup and Recovery 16](#_Toc119791077)

[3.6. Using Database Design or CASE Tool 17](#_Toc119791078)

[3.7. Other Possible E/R Relationships 17](#_Toc119791079)

[4. Implementation Description 18](#_Toc119791080)

[4.1. Data Dictionary 18](#_Toc119791081)

[4.2. Advanced Features 21](#_Toc119791082)

[4.2.1. Triggers 21](#_Toc119791083)

[4.2.2. Stored Procedures 22](#_Toc119791084)

[4.2.3. Functions 23](#_Toc119791085)

[4.3. Queries 24](#_Toc119791086)

[4.3.1. Find high frequency patients 24](#_Toc119791087)

[4.3.2. Find Doctors Frequently Prescribing Addictive Medications 24](#_Toc119791088)

[4.3.3. Most Used Reference 24](#_Toc119791089)

[4.3.4. 30-day Prescription Spending 25](#_Toc119791090)

[4.3.5. 30-day Bed Turnover 25](#_Toc119791091)

[4.3.6. Departments with the Most Visits in 30-days 25](#_Toc119791092)

[4.3.7. Visits with the Most Number of Doctors Involved 25](#_Toc119791093)

[4.3.8. Number of Doctors Involved vs Length of Stay 25](#_Toc119791094)

[5. CRUD Matrix 26](#_Toc119791095)

[5.1. List of Entity Types 26](#_Toc119791096)

[5.2. List of Functions 26](#_Toc119791097)

[6. Concluding Remarks 27](#_Toc119791098)

[7. Appendices 28](#_Toc119791099)

[7.1. Appendix A 28](#_Toc119791100)

[7.1.1. Database Creation 28](#_Toc119791101)

[7.1.2. Data Loading 40](#_Toc119791102)

[7.1.3. Table Viewing 42](#_Toc119791103)

[7.2. Appendix B 42](#_Toc119791104)

[8. References 44](#_Toc119791105)

# Introduction

## Scope and Purpose of Document

This document serves to detail the processes that went into making this Hospital Database, as well as several ways to implement and use it. I start off by stating base system requirements needed to operate the bare bones of the database. The design rationale that I followed for this project follows the requirements. You can see the specifications of all entities and relationships in their respective subsections of the Relational Model section. If you wish to skip the design of the database and wish to see how to implement it, then you should look at section 4, Implementation description. If you want to run an instance of this database on your own server then you can find the DDL within Appendix A, as well as some test data that I used.

Overall, this document will cover the conceptualization, design, implementation, and use of a Hospital Database.

## Project Objective

The objective of this project is to model hospital functions and entities as a relation DBMS and capture as much of the information as possible within the semester long working window. I fell quite short of a true implementation, but that is mostly due to the vastness of everything that is a hospital database.

# System Requirements

General minimum requirements to implement this solution in the way that I did.

## Hardware Requirements

I used MySQL Workbench to design and implement this project. The minimum hardware requirements listed on their website are: [2]

* CPU: Intel Core or Xeon 3GHz (or Dual Core 2GHz) or equal AMD CPU
* Cores: Single (Dual/Quad Core is recommended)
* RAM: 4 GB (6 GB recommended)
* Graphic Accelerators: nVidia or ATI with support of OpenGL 1.5 or higher
* Display Resolution: 1280×1024 is recommended, 1024×768 is minimum.

## Software Requirements

I used Windows 10 to create and run this project. MySQL also provides their minimum software requirements: [2]

The following operating systems are officially supported:

* Windows 7 (64-bit, Professional level or higher)
* Mac OS X 10.6.1+
* Ubuntu 9.10 (64bit)
* Ubuntu 8.04 (32bit/64bit)

For convenience the following builds are also available:

* Windows XP SP3, Vista
* Mac OSX (10.5 and 10.6) Intel
* Ubuntu 8.04 (i386/x64)
* Ubuntu 9.04 (i386/x64)
* Fedora 11 (i386/x64)

MySQL Workbench also has the following general requirements:

* The Microsoft .NET 3.5 Framework.
* Cairo 1.6.0 or later
* glib-2.10
* libxml-2.6
* libsigc++ 2.0
* pcre
* libzip

## Functional Requirements

This project supports standard CRUD operations. MySQL Workbench as a program is useful as a CASE program as well as an RDBMS. You can design an EED and automatically generate DDL to implement the design within MySQL Workbench, assuming you have a MySQL Server running to connect to.

## Database Requirements

For this project I used MySQL Workbench 8.0 CE and MySQL Server Version 8, however, MySQL Workbench supports any version above 5.7

# Database Design Description

## Design Rationale

I used a top-down approach when beginning my database design. This is a more effective path than choosing the bottom-up method, as it allows for breaking down and partitioning of high-level ideas, as well as potentially reducing the chance of redundancy in my model.

I have both natural and artificial primary keys present. For most “human” entities (Doctor, Nurse, etc.) we use the SSN to identify each entry as these are guaranteed to be unique. A user may notice that there is one “human” entity that does not use SSN as a primary key: Patient. This is because an individual may not have an SSN in our hospital if they are not a citizen, such as a tourist. “Employed humans” such as Nurses and Doctors are required to have an SSN for employment, so we are safe to use SSN as the primary key in this case.

I chose to model two M:N relationships between Doctor and Patient. While these are both treatment options, I felt that it was better to have two explicit relationships for two different types of care: referential and prescribed. I also chose to relate to the Patient table instead of Visit because while a Doctor may prescribe a medication during a visit, it is the person that will be taking that medication.

The Room entity uses a composite key of Floor and Number. I feel that this is the best way to track it, as it is unlikely that any of the numbering will change in an influential way. The only way that it could change would be through additions to the hospital, which would require changing some business rules for valid key values.

I felt that it was important to not only track Room occupancy, but to track Bed occupancy, as having as fine-grained monitoring in a hospital is a good strategy to take.

I chose to model a 1:1 nonidentifying relationship between Nurse and Room in my hospital. In many hospitals it is modeled as M:N, but I admit that this thought occurred to me a little too late to go through remodeling my queries and project to implement.

While there may be criticism about having Rooms not be permanently assigned to a Department, I believe it is safer to *not* have them assigned. This allows for any overflow incase of emergency to be put anywhere.

Lastly, some of my primary keys are composite, or non-integer primary keys. I chose to do this to create a smaller table design. However, I recently read that integer keys are preferred due to the speed up joins get from using integers rather than strings. This may be something to consider down the line, but for this current project I do not believe that that level of absolute optimization is required for high efficiency.

## E/R Model

### Entities

#### Department

The Department entity is important for several other entities defined in this section. It functions as a general bucket for many employees. It is defined by an artificial primary key idDepartment, and a unique attribute Name.

#### Room

As mentioned in the Design Rationale, Room stands alone as it can be constantly in flux of which department it belongs to, and which doctor and nurse are assigned to it.

Room is defined by a composite primary key of two attributes: floor and number. As stated in the Design Rationale, I believe this composite key is sufficient and necessary to the Room entity.

#### Doctor

The Doctor entity is crucial in any hospital database. They are responsible for patients, prescriptions, and references. The Doctor entity is defined by a natural primary key: SSN, and several attributes: firstname, lastname, salary, and specialty. There are also 2 foreign keys, idDepartment and idSpecialty. These relationships are described in the Relationship section of this document.

#### Specialty

The Specialty entity is used to enumerate specialties and extract redundancy for the Doctor entity. The specialty entity consists of a primary key idSpecialty, and a unique attribute, specialty. The specialty may be things such as Cardiovascular, neurosurgery, family medicine, etc.

#### Nurse

Similar to the Doctor entity, the Nurse entity is important to the design of a hospital database. While Doctors are not assigned to specific rooms, they are assigned to specific patients. Nurses have the converse: they are assigned to specific rooms, not specific patients. The consequence of this is that if a patient is moved from Room A to Room B, the same doctor can take care of the patient, but a different nurse will be assigned.

The Nurse entity is defined by a natural primary key, SSN, and three attributes: firstname, lastname, and salary. There are three foreign keys present: Room\_Floor, Room\_Number, and idDepartment. These foreign keys will be discussed in the Relationship section.

#### Patient

The Patient entity represents anyone that is being or has been treated in the hospital. I defined it with an artificial primary key, idPatient, since, as mentioned previously, we may have to treat people that do not have an SSN such as international tourists. I included attributes such as firstname, lastname, insurance (which I left as an attribute for simplicity, as I do not want to consider claims and policies in this project yet. I believe someone else in the class is covering this), SSN, date of birth (dob), and address (which may be international).

Since a single patient may return the hospital an unlimited number of times, we only keep one patient record for everyone that visits, regardless of the number of times visited.

#### Bed

The Bed entity is used to keep track of occupied rooms and where patients are kept. It can also be used to check for potential hazards in room sharing. For example, we would not want someone with whooping cough near an infant or child in the hospital. Bed is a very simple entity, it consists of a composite primary key idBed, which is simply the bed id in the room, and the foreign keys Room\_Floor and Room\_Number.

#### Medication

The Medication entity is used to identify what is prescribed by a doctor to a patient as part of a treatment plan. The medication entity is relatively simple and consists of the primary key idMedication, name, and company.

#### Reference

The Reference entity is used to track where doctors point patients to for third party care. When a patient becomes an outpatient, the doctor may point the patient to a specialist, such as a psychologist, GI, or chemotherapy location. The entity consists of the artificial primary key idReference, the name of the location, the address, and several foreign keys that relate the reference to the doctor and the patient.

#### ReferenceType

The ReferenceType entity is a simple reference entity used to pull out redundancy in the Reference entity. This entity consists of the primary key idReferenceType, and the attribute type to track the type of reference, such as GI, Hospital, Surgery, etc.

#### Visit

The Visit entity is used to track when patients visit and for how long. This allows the same patient to be in our records for multiple visits, without redundancy in the Patient table. The Visit entity consists of the primary key idVisit, the attributes state\_date and end\_date, and foreign keys to track the patient and the doctors that were involved in the visit, as many doctors may be required to work with a patient in a single visit. This keeps doctors accountable for their actions in every visit in case something goes wrong.

#### Prescription

The Prescription entity exists as an intersect entity in a many-to-many (M:N) relationship between Doctor and Patient. This table allows many Doctors to prescribe many Medications to many Patients. It consists of a composite primary key: date (to enable the same doctor to represcribe the same medicine to the same patient), idDoctor, idPatient, and idMedication. There are two additional attributes: price and numdoses. Price is included here instead of the Medication entity as we want to lock in the price at time of prescription. If it is included in the Medication entity we will see Update anomalies that we wish to avoid.

#### *Visit*\_*has*\_*Doctor*

This entity is part of an M:N relationship between Visit and Doctor. It allows for many Doctors to be involved in each visit a patient makes, as they might need different care on each visit. This is a simple intersect entity and does not have any attributes and is wholly defined by primary keys from the Visit and Doctor entities: idDoctor and idVisit.

#### Accountant

This entity was created to demonstrate that the Department entity can be easily expanded into other parts of the hospital outside of patientcare, such as finance, research, custodial staff, etc. The Accountant entity is quite bare. It consists of a primary key idAccountant, firstname, lastname, and salary attributes, as well as a foreign key to the Department entity.

### Relationships

It is worth noting before going into this section that if a relationship is described by the “first” half of the relation before I reach it in the “second” half, then I do not include it in the description. An example of this is with the Doctor-Visit relationship. It is detailed in the Doctor Relationships section, but not in the Visit Relationships section.

#### Department Relationships

The Department entity relates to several other entities in the same way. The Doctor, Nurse, and Accountant entities all relate to Department in a 1:M nonidentifying relationship. This is because none of the entities need the idDepartment as a part of their primary key to be uniquely identified.

#### Doctor Relationships

1. Doctor-Specialty: Nonidentifying 1:M relationship, although Specialty is specified as Not NULL. Each Doctor must have a specialty.
2. Doctor-Visit: M:N relationship through Visit\_has\_Doctor. Every visit by a patient may have 1 or more doctors. Visit\_has\_Doctor uses the idDoctor and idVisit attributes to form this relationship.
3. Doctor-Patient: M:N Relationship through Reference. Many Doctors can refer many patients to multiple places for additional treatment that the hospital might not be able to provide, or provide soon enough.
4. Doctor-Patient: M:N Relationship through Prescription. Many Doctors can Prescribe many patients medication as part of a treatment plan.

#### Nurse Relationships

1. Nurse-Room: 1:1 non-identifying relationship, but it is specified as Not Null. A Nurse needs a Room to be assigned but does not require the Room to be uniquely identified. Although Room\_Floor and Room\_Number are candidate keys. In my hospital I decided that each nurse is only allowed to be assigned to a single room, and a single room can only be assigned one nurse.

#### Room Relationships

1. Room-Bed: 1:M identifying relationship. I use an identifying relationship here because each Bed uses its Room to be uniquely identified, with the idBed being used as a counter for the bed number within the room. Each Room may have 1 or more beds, but I am only considering Rooms that are used to house patients.

#### Patient Relationships

1. Patient-Visit: 1:M non-identifying relationship. A Visit must have a patient to be created, so the idPatient field is specified as Not Null, however, all that is required to identify a Visit is the idVisit, so I used a non-identifying relationship. I used 1:M because a patient may come to the same hospital multiple times, so instead of tracking the patient visits in the patient table and having large redundancy, I use a 1:M relationship to model this.

#### Visit Relationships

1. Visit-Bed: 1:1 non-identifying relationship. This is another non-identifying relationship because the idBed foreign key is not required to uniquely identify the visit (that is handled by the artificial key: idVisit). It is also 1:1 as a patient cannot take up more than 1 bed during a visit, as I am not considering the patient moving permanently throughout the visit.

### E/R Diagram

Below is the E/R diagram of my schema.

A picture containing diagram

Description automatically generated

## Relational Model

### Data Dictionary

### Department

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idDepartment | Artificial Primary key for the Department | Int | Integer range | Primary Key | Y | Integer range |
| Name | The department name | Varchar | 45 | Unique | Y | Alphanumeric utf-8 characters |

### Doctor

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| SSN | Social Security Number | INT | 9 | Primary Key | Y | 9 base-10 digits |
| Firstname | First Name | Varchar | 45 | None | Y | Alphabetic utf-8 characters |
| Lastname | Last Name | Varchar | 45 | None | Y | Alphabetic utf-8 characters |
| Salary | Salary | Int | 9 | Check >= 0 | Y | 9 base-10 digits |
| idDepartment | Department ID | Int | Integer range | Foreign Key | Y | Numeric |
| idSpecialty | Specialty ID | Int | Integer range | Foreign Key | Y | Numeric |

### Specialty

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idSpecialty | Specialty ID | Int | Integer range | Primary Key | Y | Integer range |
| Specialty | Name of specialty | Varchar | 45 | Unique | Y | Alphabetic utf-8 characters |

### Accountant

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| SSN | Social Security Number | Int | 9 | Primary Key | Y | 9 base-10 digits |
| FirstName | First Name | Varchar | 45 | None | Y | Alphabetic utf-8 characters |
| LastName | Last Name | Varchar | 45 | None | Y | Alphabetic utf-8 characters |
| Salary | Salary | Int | 9 | Check >= 0 | Y | 9 base-10 digits |
| idDepartment | Department ID | Int | Integer range | Foreign Key | Y | Integer range |

### Visit\_has\_Doctor

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idVisit | Visit ID | Int | Integer Range | Foreign Primary Key | Y | Integer range |
| idDoctor | Doctor SSN | Int | 9 | Foreign Primary Key | Y | 9 base-10 digits |

### Prescription

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idDoctor | Doctor SSN | Int | 9 | Foreign Primary Key | Y | 9 base-10 digits |
| idPatient | Patient ID | Int | Integer Range | Foreign Primary Key | Y | Integer range |
| idMedication | Medication ID | Int | Integer Range | Foreign Primary Key | Y | Integer range |
| Price | Price of medication at time of prescription | Double | 8 | None | Y | 8 base-10 digits |
| Numdoses | Number of doses prescribed | Int | 3 | Check > 0 | Y | 3 base-10 digits |
| Date | Date prescribed | Date | Date range | Primary Key | Y | Date range |

### Medication

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idMedication | Medication ID | Int | Integer Range | Primary Key | Y | Integer range |
| Name | Medication Name | Varchar | 45 | None | Y | 45 alphanumeric utf-8 characters |
| Company | Manufacturing Company | Varchar | 45 | None | Y | 45 alphanumeric utf-8 characters |

### Nurse

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| SSN | Social Security Number | Int | 9 | Primary Key | Y | 9 base-10 digits |
| FirstName | First Name | Varchar | 45 | None | Y | 45 alphanumeric utf-8 characters |
| LastName | Last Name | Varchar | 45 | None | Y | 45 alphanumeric utf-8 characters |
| Salary | Salary | Int | 9 | Check >= 0 | Y | 9 base-10 digits in integer range |
| idDepartment | Department ID | Int | Integer Range | Foreign Key | Y | Integer range |
| Room\_Floor | Floor number | Int | 2 | Foreign Key | Y | 1-10 |
| Room\_Number | Room number | Int | 4 | Foreign Key | Y | 101-1099 |

### Reference

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idReference | Reference ID | Int | Integer Range | Primary Key | Y | Integer range |
| Name | Name of reference | Varchar | 45 | None | Y | 45 alphanumeric utf-8 characters |
| Address | Address of reference | Varchar | 45 | None | Y | 45 alphanumeric utf-8 characters |
| idDoctor | Doctor SSN | Int | 9 | Foreign Primary Key | Y | 9 base-10 digits |
| idPatient | Patient ID | Int | Integer Range | Foreign Primary Key | Y | Integer range |
| idReferenceType | Reference Type ID | Int | Integer Range | Foreign Primary Key | Y | Integer range |
| Date | Date | Date | Date range | None | Y | Date range |

### ReferenceType

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idReferenceType | Reference Type ID | Int | Integer range | Primary Key | Y | Integer range |
| Type | Type of reference | Varchar | 45 | Unique | Y | 45 Alphanumeric utf-8 characters |

### Patient

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idPatient | Patient ID | Int | Integer range | Primary Key | Y | Integer range |
| FirstName | First name | Varchar | 45 | None | Y | 45 alphanumeric utf-8 characters |
| LastName | Last name | Varchar | 45 | None | Y | 45 alphanumeric utf-8 characters |
| Insurance | Insurance Flag | Boolean | 1 | None | Y | TRUE/FALSE |
| SSN | Social Security Number | Int | 9 | None | N | 9 base-10 digits/NULL |
| Address | Home address | Varchar | 45 | None | Y | 45 alphanumeric utf-8 characters/NULL |

### Visit

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idVisit | Visit ID | Int | Integer range | Primary Key | Y | Integer Range |
| idPatient | Patient ID | Int | Integer Range | Foreign Primary Key | Y | Integer range |
| idBed | Bed ID | Int | Integer Range | Foreign Primary Key | Y | Integer range |
| Bed\_Room\_Floor | Floor ID | Int | 2 | Foreign Primary Key | Y | 1-10 |
| Bed\_Room\_Number | Room number | Int | 4 | Foreign Primary Key | Y | 101-1099 |
| Start\_date | Admittance date | Date | Date range | None | Y | Date range |
| End\_date | Release date | Date | Date range | None | N | Date range/NULL |

### Room

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| Floor | Floor number | Int | 2 | Primary Key | Y | 1-10 |
| Number | Room Number | Int | 4 | Primary Key | Y | 101-1099 |

### Bed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Description | Data Type | Size | Constraint Type | Not Null? | Valid Values |
| idBed | BedID | Int | Integer range | Primary Key | Y | Integer range |
| Room\_Floor | Floor number | Int | 2 | Foreign Primary Key | Y | 1-10 |
| Room\_Number | Room number | Int | 4 | Foreign Primary Key | Y | 101-1099 |

### Integrity Rules

In general, most of the attributes of every table are forced to be not null. If a user attempts to add an entry to a table without first specifying all mandatory fields, then the insertion will fail due to not null constraints. There are only a couple of fields that are not specified not null, but those are not mandatory fields, such as the end\_date field for a visit, as a null end\_date indicates the visit is ongoing. There are also several Checks being implemented to ensure data falls into valid ranges. To maintain referential integrity, I use Foreign Key constraints on the related fields. Below are examples of my integrity rules.

1. A doctor may give many references of the same type to the same patient on the same day, which is why we require an artificial part of the Reference primary key.
2. A Doctor may prescribe several prescriptions to the same patient on the same day (but not several prescriptions of the same medication on the same day, which is why we require an artificial part of the Prescription primary key.
3. A patient may have multiple visits on the same day.
4. A nurse can only be assigned to one room at a time.
5. A doctor can only have one specialty.
6. Visit end\_date cannot be before start\_date.

### Operational Rules

Below I will list out several operational rules that we allow or disallow.

1. Patient records cannot be deleted.
2. Patientcare-associated records cannot be deleted (doctors, prescriptions, references) until 6 years after most recent access.
3. Doctors can prescribe medication without a visit, to allow for ease of represcribing.

### Operations

Below are a list of use cases and required operations.

1. Adding a patient – insertion of patient
2. Adding a doctor/nurse/accountant – insertion of department and doctor/nurse/accountant
3. Finding patient’s nurse – retrieve and joins of a visit, bed, room, and nurse
4. Finding patient prescriptions – retrieve and join of Patient and Prescription
5. Changing a nurse’s room – update Nurse

## Security

In the current instance of the hospital database, there are a few security features implemented, but others would be implemented as part of the grand application implementation. The easiest implementation of security is using proper user provisioning and password protection. User access would also have to be protected at several layers above the database layer, such as in the application layer, as well as through business regulation and physical access control (id cards, two-factor authentication, etc.). These strategies, while absolutely crucial in any business, are not implementable in a single database. SQL injection would have to be protected mostly upstream of the database as well. Using packages and libraries in application layer languages would aid in adding another layer of protection to bad actors attempting to disrupt or obtain data from our database. Maintaining an audit trail of which users modified which records and when is also important to identify misuse, whether malicious or not.

If we look beyond the application and into network layer communication, we can find additional security measures that can be taken. Using solely private, password protected networks and communicating within a VPN will bolster defense. Additionally, only using encrypted means of transporting data will enforce further security. Lastly, storing encrypted or hashed data will obscure the sensitive information even further.

Since this is a hospital database, there are strict regulations in how to go about operating and storing data. Policies such as HIPAA require certain practices to be followed when it comes to medical information. Abiding by these rules is required for any hospital for function in the U.S.

## Database Backup and Recovery

Using MySQL with InnoDB gives us a robust backup and recovery system right out of the box. There are four main categories of failure:

1. Operating system crash
2. Power failure
3. File system crash
4. Hardware problem

In the event of an operating system crash or power failure, InnoDB will maintain a log of recently committed and not committed changes to the db. Upon restart, InnoDB will push the committed changes through to the database and roll back the uncommitted ones to maintain integrity and consistency.

File system and hardware issues are more dangerous. In these cases, reformatting the MySQL data disk may be necessary if correcting the problem is not possible. You can also restore from a backup, assuming a backup has been made. Creating self-made backups or setting an automated backup schedule is crucial in recovery.

MySQL offers two modes of backup creation: MySQL Enterprise Backup, and mysqldump. MySQL Enterprise Backup performs an online physical backup, while mysqldump performs an online logical backup. It is important to create a schedule and pick a low load time, as mysqldump enforces a Read lock at the beginning of the dump.

The MySQL developers caution against creating multiple full backups, as it creates highly redundant backups in low-update areas of the database. Instead, they advise using a beginning (or infrequent) full backup, then creating incremental backups until the next scheduled full backup time. This will optimize for storage space and backup time.

Another layer of robust fault tolerance is using a Redundant Array of Independent Disks (RAID). By using various levels of RAID, one can develop a highly reliable and crash-resistant system.

## Using Database Design or CASE Tool

In the very beginning of the project, I was using LucidChart for quick sketches of what some preliminary tables and relations may look like. However, I quickly jumped to MySQL Workbench for creating my EER diagram and exporting of the finished diagram into SQL code, which I then ran in MySQL workbench to generate the final database.

I found MySQL workbench to be quite easy to use. There were a few caveats that took me some time to figure out, such as exporting EER models into code and how to make a new table without leaving the EER Diagram tab. However, outside of those minor issues (that admittedly are more my fault than the tools), I found it fun and easy to use. It is a great deal better than LucidChart since it is specifically designed to be “SQL-friendly” and easily show and update attribute constraints. It is also impressive how good the community version of the tool is.

## Other Possible E/R Relationships

It is amusing to look back at the early semester designs and see how much I have tinkered with over the length of the course. Creating the Visit entity was a more recent change, along with creating more reference tables. It took me longer than I care to admit to stop using so many artificial primary keys and begin using composite or natural keys such as SSN. Even up to the last day of the project, I was changing around a few attributes and relationships. I bring all of this up to show how much I didn’t know I didn’t know and that I am sure there are plenty of additions or changes that can be made to this database in the coming weeks. One major change that I can see happening is my implementation of Rooms and treatment options.

One thing that I debated on changing up until the minute of submission was how to handle References and Prescriptions. There are two alternate ways that I can think of implementing them. One is to switch the relationship from an M:N between Doctor and Patient to an M:N between Doctor and Visit, since the treatment is provided during the visit. However, I think this is less natural than what I currently have (M:N Doctor-Patient). Another implementation could be having an abstracted “treatment” entity that relates to various types of treatment such as procedures, observations, prescriptions, references, etc. and have the treatment type enumerated. This is more of an object-oriented/EERD approach, and I did not pursue that line of thinking in this implementation.

# Implementation Description

## Data Dictionary

Below is the data dictionary generated from the code from the cited GitHub [1].

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| table\_nmS | obj\_typ | ord | is\_key | column\_nm | data\_typ | nullable |
| accountant | TBL | 1 | PK | SSN | int(10) | NOT NULL |
| accountant | TBL | 2 |  | FirstName | varchar(45) | NOT NULL |
| accountant | TBL | 3 |  | LastName | varchar(45) | NOT NULL |
| accountant | TBL | 4 |  | Salary | int(10) | NOT NULL |
| accountant | TBL | 5 | FK | idDepartment | int(10) | NOT NULL |
| bed | TBL | 1 | PK | idBed | int(10) | NOT NULL |
| bed | TBL | 2 | FK,PK | Room\_Floor | int(10) | NOT NULL |
| bed | TBL | 3 | FK,PK | Room\_Number | int(10) | NOT NULL |
| department | TBL | 1 |  | Dname | varchar(15) | NOT NULL |
| department | TBL | 1 | PK | idDepartment | int(10) | NOT NULL |
| department | TBL | 1 |  | Dname | varchar(15) | NOT NULL |
| department | TBL | 2 |  | Dnumber | int(10) | NOT NULL |
| department | TBL | 2 |  | Dnumber | int(10) | NOT NULL |
| department | TBL | 2 | UK | Name | varchar(45) | NOT NULL |
| department | TBL | 3 |  | mgrssn | char(9) | NULL |
| department | TBL | 3 |  | Mgrssn | char(9) | NULL |
| department | TBL | 4 |  | mgrstartdate | date(3) | NULL |
| department | TBL | 4 |  | Mgrstartdate | date(3) | NULL |
| doctor | TBL | 1 | PK | SSN | int(10) | NOT NULL |
| doctor | TBL | 2 |  | FirstName | varchar(45) | NOT NULL |
| doctor | TBL | 3 |  | LastName | varchar(45) | NOT NULL |
| doctor | TBL | 4 |  | Salary | int(10) | NOT NULL |
| doctor | TBL | 5 | FK | idDepartment | int(10) | NOT NULL |
| doctor | TBL | 6 | FK | idSpecialty | int(10) | NOT NULL |
| medication | TBL | 1 | PK | idMedication | int(10) | NOT NULL |
| medication | TBL | 2 |  | name | varchar(45) | NOT NULL |
| medication | TBL | 3 |  | company | varchar(45) | NOT NULL |
| nurse | TBL | 1 | PK | SSN | int(10) | NOT NULL |
| nurse | TBL | 2 |  | FirstName | varchar(45) | NOT NULL |
| nurse | TBL | 3 |  | LastName | varchar(45) | NOT NULL |
| nurse | TBL | 4 |  | Salary | int(10) | NOT NULL |
| nurse | TBL | 5 | FK | idDepartment | int(10) | NOT NULL |
| nurse | TBL | 6 | FK | Room\_Floor | int(10) | NOT NULL |
| nurse | TBL | 7 | FK | Room\_Number | int(10) | NOT NULL |
| patient | TBL | 1 | PK | idPatient | int(10) | NOT NULL |
| patient | TBL | 2 |  | FirstName | varchar(45) | NOT NULL |
| patient | TBL | 3 |  | LastName | varchar(45) | NOT NULL |
| patient | TBL | 4 |  | Insurance | tinyint(3) | NOT NULL |
| patient | TBL | 5 |  | SSN | int(10) | NULL |
| patient | TBL | 6 |  | address | varchar(45) | NOT NULL |
| prescription | TBL | 1 | FK,PK | idDoctor | int(10) | NOT NULL |
| prescription | TBL | 2 | FK,PK | idPatient | int(10) | NOT NULL |
| prescription | TBL | 3 | FK,PK | idMedication | int(10) | NOT NULL |
| prescription | TBL | 4 |  | price | decimal(8,0) | NOT NULL |
| prescription | TBL | 5 |  | numdoses | int(10) | NOT NULL |
| prescription | TBL | 6 | PK | date | date(3) | NOT NULL |
| reference | TBL | 1 | PK | idReference | int(10) | NOT NULL |
| reference | TBL | 2 |  | Name | varchar(45) | NOT NULL |
| reference | TBL | 3 |  | Address | varchar(45) | NOT NULL |
| reference | TBL | 4 | FK,PK | idDoctor | int(10) | NOT NULL |
| reference | TBL | 5 | FK,PK | idPatient | int(10) | NOT NULL |
| reference | TBL | 6 | FK,PK | idReferenceType | int(10) | NOT NULL |
| reference | TBL | 7 |  | date | date(3) | NOT NULL |
| referencetype | TBL | 1 | PK | idReferenceType | int(10) | NOT NULL |
| referencetype | TBL | 2 |  | type | varchar(45) | NOT NULL |
| room | TBL | 1 | PK | Floor | int(10) | NOT NULL |
| room | TBL | 2 | PK | Number | int(10) | NOT NULL |
| specialty | TBL | 1 | PK | idSpecialty | int(10) | NOT NULL |
| specialty | TBL | 2 | UK | specialty | varchar(45) | NOT NULL |
| visit | TBL | 1 | PK | idVisit | int(10) | NOT NULL |
| visit | TBL | 2 |  | start\_date | date(3) | NOT NULL |
| visit | TBL | 3 |  | end\_date | date(3) | NULL |
| visit | TBL | 4 | FK | idBed | int(10) | NOT NULL |
| visit | TBL | 5 | FK | Bed\_Room\_Floor | int(10) | NOT NULL |
| visit | TBL | 6 | FK | Bed\_Room\_Number | int(10) | NOT NULL |
| visit | TBL | 7 | FK | idPatient | int(10) | NOT NULL |
| visit\_has\_doctor | TBL | 1 | FK,PK | idVisit | int(10) | NOT NULL |
| visit\_has\_doctor | TBL | 2 | FK,PK | idDoctor | int(10) | NOT NULL |

## Advanced Features

### Triggers

My triggers focus on preventing deletions of patient records as follows.

1. Preventing Patient record deletion:

delimiter $$

CREATE TRIGGER Prevent\_early\_deletion

BEFORE DELETE ON Patient

FOR EACH ROW

BEGIN

signal sqlstate '45000' set message\_text = 'Records can not be deleted';

END

$$

delimiter ;

1. Preventing Prescription Record deletion

delimiter $$

CREATE TRIGGER Prevent\_early\_Prescription\_deletion

BEFORE DELETE ON Prescription

FOR EACH ROW

BEGIN

signal sqlstate '45000' set message\_text = 'Prescription records can not be deleted';

END

$$

delimiter ;

1. Preventing Reference record deletion

delimiter $$

CREATE TRIGGER Prevent\_early\_Reference\_deletion

BEFORE DELETE ON Reference

FOR EACH ROW

BEGIN

signal sqlstate '45000' set message\_text = 'Reference records can not be deleted';

END

$$

delimiter ;

1. Preventing Visit record deletion

delimiter $$

CREATE TRIGGER Prevent\_early\_Visit\_deletion

BEFORE DELETE ON Reference

FOR EACH ROW

BEGIN

signal sqlstate '45000' set message\_text = 'Visit records can not be deleted';

END

$$

delimiter ;

### Stored Procedures

1. The first stored procedure I made was to get all of the foreign (SSN is null) patients we have treated.

CREATE DEFINER=`root`@`localhost` PROCEDURE `select\_foreign\_patients`()

BEGIN

SELECT \* FROM Patients WHERE isnull(SSN);

END

1. The next stored procedure I made was to get all visits within a user-defined number of days. Here I used an IF statement to raise an error if the value ways less than 1 day (because that would not be in the past) or greater than 100 years (as we would not have records and it seems malicious).

CREATE DEFINER=`root`@`localhost` PROCEDURE `select\_visits\_within\_x\_days`(IN num\_days INT)

BEGIN

IF (num\_days < 1 OR num\_days > 36525) THEN

signal sqlstate '45000' set message\_text = 'Visit records can not be deleted';

END IF;

SELECT \* FROM visit WHERE start\_date BETWEEN CURDATE() - INTERVAL num\_days DAY AND CURDATE();

END

1. The third stored procedure I made was to get all of a user’s prescriptions. This would aid in identifying users that use prescriptions too quickly, which could indicate misuse.

CREATE DEFINER=`root`@`localhost` PROCEDURE `get\_patient\_prescriptions`(IN patient INT)

BEGIN

SELECT \* FROM Patient p JOIN Prescription pres ON p.idPatient = pres.idPatient WHERE p.idPatient = patient;

END

1. The fourth stored procedure I made was one so that we can find optimal spaces to add patients as the enter the hospital.

CREATE DEFINER=`root`@`localhost` PROCEDURE `find\_all\_open\_beds`()

BEGIN

SELECT \* FROM Bed WHERE (idBed, Room\_Floor, Room\_Number) NOT IN (SELECT idBed, Bed\_Room\_Floor, Bed\_Room\_Number FROM Visit WHERE isnull(end\_date));

END

### Functions

1. For my first function, I wanted to calculate the date that a prescription would run out.

CREATE DEFINER=`root`@`localhost` FUNCTION `calculate\_prescription\_end`(num\_doses INT, start\_date DATE) RETURNS date

DETERMINISTIC

BEGIN

RETURN DATE\_ADD(start\_date, INTERVAL num\_doses day);

END

1. My second function returns the lifetime expenses on prescriptions for a specified patient.

CREATE DEFINER=`root`@`localhost` FUNCTION `lifetime\_prescription\_expense`(patient INT) RETURNS int

DETERMINISTIC

BEGIN

SET @res := (SELECT SUM(price \* numdoses) FROM Prescription WHERE idPatient = patient);

RETURN @res;

END

1. My third function is used to the number of open beds so that we know the current utilization of the hospital

CREATE DEFINER=`root`@`localhost` FUNCTION `find\_num\_open\_beds`() RETURNS int

DETERMINISTIC

BEGIN

SET @open\_beds := (SELECT COUNT(\*) FROM Bed WHERE (idBed, Room\_Floor, Room\_Number) NOT IN (SELECT idBed, Bed\_Room\_Floor, Bed\_Room\_Number FROM Visit WHERE isnull(end\_date)));

RETURN @open\_beds;

END

## Queries

Below are relatively simple, albeit mission critical, queries that would be used frequently in hospital performance reports.

## Find high frequency patients

Finding out who is frequently visiting may be an indication of something undiagnosed, or an inefficiency in treatment.

SELECT idPatient, COUNT(\*) AS Num\_Visits FROM visit GROUP BY idPatient Order By Num\_Visits DESC;

## Find Doctors Frequently Prescribing Addictive Medications

Although I did not include a “danger” flag for medication, this query finds doctors that frequently prescribe addictive medications such as oxycontin and valium.

SELECT p.idDoctor, COUNT(\*) AS Dangerous\_Prescriptions FROM Prescription p JOIN Medication m ON p.idMedication = m.idMedication WHERE name = "Valium" OR name = "oxycontin" GROUP BY p.idDoctor

## Most Used Reference

In order to assess how often we refer patients to outside help, and to assess which organizations are dealing with a large number of our patients (potentially indicating a high level of care), I wanted to count the number of patients being referred to each location.

SELECT Name, COUNT(\*) as Num\_Referenecs FROM reference GROUP BY Name ORDER BY Num\_Referenecs DESC

## 30-day Prescription Spending

For this query, I wanted to get a total sum of prescription spending over a rolling 30-day period. I was then curious about what the medication breakdown of the spending was, so I queried further.

Total:

SELECT SUM(price \* numdoses) AS Total\_Spending FROM prescription WHERE date BETWEEN CURDATE() - INTERVAL 30 DAY AND CURDATE()

Medication-wise breakdown:

SELECT m.name, SUM(price \* numdoses) AS Total\_Spending FROM prescription p JOIN Medication m ON p.idMedication = m.idMedication WHERE date BETWEEN CURDATE() - INTERVAL 30 DAY AND CURDATE() GROUP BY name

## 30-day Bed Turnover

I wanted to see which beds were occupied for the most amount of time on a rolling, 30-day basis. The Occupied\_Time column is expressed as a proportion of occupied time over 30 days (num\_days occupied / 30)

SELECT idBed, Bed\_Room\_Floor, Bed\_Room\_Number, SUM(DATEDIFF(IF(isnull(end\_date), CURDATE(), end\_date), start\_date) + 1)/30 AS Occupied\_Time FROM visit WHERE start\_date BETWEEN CURDATE() - INTERVAL 30 DAY AND CURDATE() GROUP BY idBed, Bed\_Room\_Floor, Bed\_Room\_Number;

## Departments with the Most Visits in 30-days

This is another common reporting statistic. In order to see how much traffic each department gets, and possibly decide where additional funding is needed, we look at how many people visit each department in a 30-day period.

SELECT dept.name, COUNT(\*) AS Num\_Visits FROM visit\_has\_doctor vhd JOIN doctor d ON vhd.idDoctor = d.SSN JOIN department dept ON d.idDepartment = dept.idDepartment JOIN visit v ON v.idVisit = vhd.idVisit WHERE start\_date BETWEEN CURDATE() - INTERVAL 30 DAY AND CURDATE()GROUP BY dept.name

## Visits with the Most Number of Doctors Involved

Here I was curious about how we could report potentially highly complex treatment plans arising from a hard-to-fight condition. Knowing which types of visits will require additional personnel will streamline patientcare.

SELECT v.idVisit, COUNT(\*) AS Num\_Doctors FROM visit v JOIN visit\_has\_doctor vhd ON v.idVisit = vhd.idVisit GROUP BY v.idVisit

## Number of Doctors Involved vs Length of Stay

This query stemmed from the pervious one. I was curious how the number of doctors would impact visit length. For example, does the number of doctors linearly or exponentially increase visit time?

SELECT v.idVisit, v.idPatient, COUNT(\*)/SUM(DATEDIFF(IF(isnull(end\_date), CURDATE(), end\_date), start\_date) + 1) AS Doctors\_v\_length FROM visit v JOIN visit\_has\_doctor vhd ON v.idVisit = vhd.idVisit GROUP BY v.idVisit, v.idPatient

# CRUD Matrix

The CRUD Matrix is shown below. Definitions for entities and functions are shown in the following subsections. I had to use the E and F labels for the matrix in order to maintain legibility of the CRUD Matrix. You will notice that there are not a lot of DELETE or UPDATE operations occurring. I attempted to design the database to be somewhat immutable, as once a treatment is given it should not be deleted or updated since we want to be able to audit potential issues in treatment should something adverse occur or if we need to know previous medications.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F/E | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | E10 | E11 | E12 | E13 | E14 |
| F1 |  |  |  |  |  |  | C |  |  |  |  |  |  |  |
| F2 | R | C | R |  |  |  |  |  |  |  |  |  |  |  |
| F3 |  |  |  | C | R |  |  |  |  |  |  |  |  |  |
| F4 |  |  |  |  | R | R | R | C |  |  |  |  |  |  |
| F5 |  | R |  |  |  |  | R |  |  | R | C |  |  |  |
| F6 |  | R |  |  |  |  | R |  |  |  |  | R | C |  |
| F7\* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## List of Entity Types

E1: Department

E2: Doctor

E3: Specialty

E4: Nurse

E5: Room

E6: Bed

E7: Patient

E8: Visit

E9: Visit\_has\_Doctor

E10: Medication

E11: Prescription

E12: ReferenceType

E13: Reference

E14: Accountant

## List of Functions

F1: Add Patient

F2: Add Doctor

F3: Add Nurse

F4: Add Visit

F5: Prescribe Medication

F6: Provide Reference

F7: Delete Patient\*

\* - Not Allowed, I just want to clarify within a CRUD Matrix

# Concluding Remarks

I ended up enjoying this project more than I thought I would at the outset. Tinkering away at the design was quite satisfying, and seeing it come together over time was nice. The biggest lesson I learned was to sit with the E/R Diagram for awhile before starting anything else. I thought that I had finished it for sure several times, and yet I kept going back to change a piece here and there, which meant that I had to change parts of this report as I update the original schema. Having a well thought out schema from the start would have minimized the redundant work that I had to do. Another lesson that I learned was to use as few artificial keys as possible. One of the things that took awhile to update in this document after changing the schema was my primary key usage. For example, I started with an artificial key for the Room entity, but then I realized that it was inefficient and created more storage load, so I deleted the key and used Floor and Number as a composite primary key. That switching from artificial primary key happened with a decent amount of my tables. Another large lesson I learned from actually doing the project was using foreign keys and nonidentifying relationships in a wiser way. At first all of my relationships were identifying and I wondered why so many keys were being brought into other relations. Eventually I realized that I had not used any nonidentifying relationships where I should be using them. That update was another long one to propagate through this project. The last major lesson that I had solidified by this assignment was the construction of SQL statements. My original approach was much different from the approach I used to construct the statements in this assignment. I now start with a single table select and work my way out from there as opposed to trying to get all of the complex pieces working in one go.

I think that the biggest weakness of this report is my Advanced Features section. I did not really feel a need to have so many advanced features in my database, so I was really reaching to get the 3 of each category laid out in the example report. My triggers are basically the same thing; however I could not think of any additional uses that I could have with triggers for my database. I also used stored procedures more as functions, with not OUT parameters or return variables, besides the set results.

I think that a strength of my report is my querying, as I believe that the queries I created would truly be helpful in a real world setting for analyzing important information. I enjoyed making the 30-day aggregations.

If I had more time I would like to greatly expand several parts of this database that feel shallow. Specifically, the Room entity could be expanded into different types of rooms and procedures that take place in those rooms. I could also get more fine-grained timing of procedures and bed availability by using DATETIME instead of DATE. A big part of the Visit entity that I feel is missing is a diagnosis. This would be interesting to track for doctor involvement and length of stay, as well as frequency of returns. Lastly, there are so many attributes that could be added to the entities in order to flesh them out further, however these would be more informational than critical to the structure of the database, so I only included important example attributes and critical ones. As I admitted in my Design Rationale section, I would like to change how I model the Nurse-Room relationship and change it from 1:1 to an M:N relationship to more accurately capture how a hospital would function. A Hospital database is so complex that a single person working on it part time could not possibly capture all of the vast amounts of information that need to be tracked.

Finally, this project inspired me to work on a database for my lab. I am hoping to take what I learned from MySQL Workbench and the material from the class and improve out inventory tracking with SQL instead of using spreadsheets. I am excited to implement some advanced features in that project for alerting after time has passed or a stock is seen low!

Thank you for teaching this class, Professor. Have a happy Thanksgiving and Winter!

# Appendices

## Appendix A

### Database Creation

-- MySQL Script generated by MySQL Workbench

-- Sat Nov 19 19:44:34 2022

-- Model: New Model Version: 1.0

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema new\_schema1

-- -----------------------------------------------------

DROP SCHEMA IF EXISTS `new\_schema1` ;

-- -----------------------------------------------------

-- Schema new\_schema1

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `new\_schema1` ;

USE `new\_schema1` ;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Patient`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Patient` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Patient` (

`idPatient` INT NOT NULL AUTO\_INCREMENT,

`FirstName` VARCHAR(45) NOT NULL,

`LastName` VARCHAR(45) NOT NULL,

`Insurance` TINYINT NOT NULL,

`SSN` INT(9) NULL,

`address` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idPatient`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Department`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Department` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Department` (

`idDepartment` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idDepartment`))

ENGINE = InnoDB;

CREATE UNIQUE INDEX `Name\_UNIQUE` ON `new\_schema1`.`Department` (`Name` ASC) VISIBLE;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Specialty`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Specialty` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Specialty` (

`idSpecialty` INT NOT NULL AUTO\_INCREMENT,

`specialty` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idSpecialty`))

ENGINE = InnoDB;

CREATE UNIQUE INDEX `specialty\_UNIQUE` ON `new\_schema1`.`Specialty` (`specialty` ASC) VISIBLE;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Doctor`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Doctor` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Doctor` (

`SSN` INT(9) NOT NULL,

`FirstName` VARCHAR(45) NOT NULL,

`LastName` VARCHAR(45) NOT NULL,

`Salary` INT(9) NOT NULL,

`idDepartment` INT NOT NULL,

`idSpecialty` INT NOT NULL,

PRIMARY KEY (`SSN`),

CONSTRAINT `fk\_Doctor\_Department1`

FOREIGN KEY (`idDepartment`)

REFERENCES `new\_schema1`.`Department` (`idDepartment`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Doctor\_Specialty1`

FOREIGN KEY (`idSpecialty`)

REFERENCES `new\_schema1`.`Specialty` (`idSpecialty`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Doctor\_Department1\_idx` ON `new\_schema1`.`Doctor` (`idDepartment` ASC) VISIBLE;

CREATE INDEX `fk\_Doctor\_Specialty1\_idx` ON `new\_schema1`.`Doctor` (`idSpecialty` ASC) VISIBLE;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Room`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Room` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Room` (

`Floor` INT(2) NOT NULL,

`Number` INT(4) NOT NULL,

PRIMARY KEY (`Floor`, `Number`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Nurse`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Nurse` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Nurse` (

`SSN` INT(9) NOT NULL,

`FirstName` VARCHAR(45) NOT NULL,

`LastName` VARCHAR(45) NOT NULL,

`Salary` INT(9) NOT NULL,

`idDepartment` INT NOT NULL,

`Room\_Floor` INT(2) NOT NULL,

`Room\_Number` INT(4) NOT NULL,

PRIMARY KEY (`SSN`),

CONSTRAINT `fk\_Nurse\_Department1`

FOREIGN KEY (`idDepartment`)

REFERENCES `new\_schema1`.`Department` (`idDepartment`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Nurse\_Room1`

FOREIGN KEY (`Room\_Floor` , `Room\_Number`)

REFERENCES `new\_schema1`.`Room` (`Floor` , `Number`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Nurse\_Department1\_idx` ON `new\_schema1`.`Nurse` (`idDepartment` ASC) VISIBLE;

CREATE INDEX `fk\_Nurse\_Room1\_idx` ON `new\_schema1`.`Nurse` (`Room\_Floor` ASC, `Room\_Number` ASC) VISIBLE;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Accountant`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Accountant` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Accountant` (

`SSN` INT(9) NOT NULL,

`FirstName` VARCHAR(45) NOT NULL,

`LastName` VARCHAR(45) NOT NULL,

`Salary` INT(9) NOT NULL,

`idDepartment` INT NOT NULL,

PRIMARY KEY (`SSN`),

CONSTRAINT `fk\_Accountant\_Department1`

FOREIGN KEY (`idDepartment`)

REFERENCES `new\_schema1`.`Department` (`idDepartment`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Accountant\_Department1\_idx` ON `new\_schema1`.`Accountant` (`idDepartment` ASC) VISIBLE;

-- -----------------------------------------------------

-- Table `new\_schema1`.`ReferenceType`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`ReferenceType` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`ReferenceType` (

`idReferenceType` INT NOT NULL AUTO\_INCREMENT,

`type` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idReferenceType`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Reference`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Reference` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Reference` (

`idReference` INT NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(45) NOT NULL,

`Address` VARCHAR(45) NOT NULL,

`idDoctor` INT NOT NULL,

`idPatient` INT NOT NULL,

`idReferenceType` INT NOT NULL,

`date` DATE NOT NULL,

PRIMARY KEY (`idReference`, `idDoctor`, `idPatient`, `idReferenceType`),

CONSTRAINT `fk\_Reference\_Doctor1`

FOREIGN KEY (`idDoctor`)

REFERENCES `new\_schema1`.`Doctor` (`SSN`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Reference\_Patient1`

FOREIGN KEY (`idPatient`)

REFERENCES `new\_schema1`.`Patient` (`idPatient`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Reference\_ReferenceType1`

FOREIGN KEY (`idReferenceType`)

REFERENCES `new\_schema1`.`ReferenceType` (`idReferenceType`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Reference\_Doctor1\_idx` ON `new\_schema1`.`Reference` (`idDoctor` ASC) VISIBLE;

CREATE INDEX `fk\_Reference\_Patient1\_idx` ON `new\_schema1`.`Reference` (`idPatient` ASC) VISIBLE;

CREATE INDEX `fk\_Reference\_ReferenceType1\_idx` ON `new\_schema1`.`Reference` (`idReferenceType` ASC) VISIBLE;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Medication`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Medication` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Medication` (

`idMedication` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) NOT NULL,

`company` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idMedication`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Prescription`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Prescription` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Prescription` (

`idDoctor` INT(9) NOT NULL,

`idPatient` INT NOT NULL,

`idMedication` INT NOT NULL,

`price` DECIMAL(8) NOT NULL,

`numdoses` INT NOT NULL,

`date` DATE NOT NULL,

PRIMARY KEY (`idDoctor`, `idPatient`, `idMedication`, `date`),

CONSTRAINT `fk\_Medication\_Doctor1`

FOREIGN KEY (`idDoctor`)

REFERENCES `new\_schema1`.`Doctor` (`SSN`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Medication\_Patient1`

FOREIGN KEY (`idPatient`)

REFERENCES `new\_schema1`.`Patient` (`idPatient`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Prescribes\_Medication1`

FOREIGN KEY (`idMedication`)

REFERENCES `new\_schema1`.`Medication` (`idMedication`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Medication\_Doctor1\_idx` ON `new\_schema1`.`Prescription` (`idDoctor` ASC) VISIBLE;

CREATE INDEX `fk\_Medication\_Patient1\_idx` ON `new\_schema1`.`Prescription` (`idPatient` ASC) VISIBLE;

CREATE INDEX `fk\_Prescribes\_Medication1\_idx` ON `new\_schema1`.`Prescription` (`idMedication` ASC) VISIBLE;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Bed`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Bed` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Bed` (

`idBed` INT NOT NULL,

`Room\_Floor` INT(2) NOT NULL,

`Room\_Number` INT(4) NOT NULL,

PRIMARY KEY (`idBed`, `Room\_Floor`, `Room\_Number`),

CONSTRAINT `fk\_Bed\_Room1`

FOREIGN KEY (`Room\_Floor` , `Room\_Number`)

REFERENCES `new\_schema1`.`Room` (`Floor` , `Number`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Bed\_Room1\_idx` ON `new\_schema1`.`Bed` (`Room\_Floor` ASC, `Room\_Number` ASC) VISIBLE;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Visit`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Visit` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Visit` (

`idVisit` INT NOT NULL AUTO\_INCREMENT,

`start\_date` DATE NOT NULL,

`end\_date` DATE NULL,

`idBed` INT NOT NULL,

`Bed\_Room\_Floor` INT(2) NOT NULL,

`Bed\_Room\_Number` INT(4) NOT NULL,

`idPatient` INT NOT NULL,

PRIMARY KEY (`idVisit`),

CONSTRAINT `fk\_Visit\_Bed1`

FOREIGN KEY (`idBed` , `Bed\_Room\_Floor` , `Bed\_Room\_Number`)

REFERENCES `new\_schema1`.`Bed` (`idBed` , `Room\_Floor` , `Room\_Number`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Visit\_Patient1`

FOREIGN KEY (`idPatient`)

REFERENCES `new\_schema1`.`Patient` (`idPatient`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Visit\_Bed1\_idx` ON `new\_schema1`.`Visit` (`idBed` ASC, `Bed\_Room\_Floor` ASC, `Bed\_Room\_Number` ASC) VISIBLE;

CREATE INDEX `fk\_Visit\_Patient1\_idx` ON `new\_schema1`.`Visit` (`idPatient` ASC) VISIBLE;

-- -----------------------------------------------------

-- Table `new\_schema1`.`Visit\_has\_Doctor`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `new\_schema1`.`Visit\_has\_Doctor` ;

CREATE TABLE IF NOT EXISTS `new\_schema1`.`Visit\_has\_Doctor` (

`idVisit` INT NOT NULL,

`idDoctor` INT(9) NOT NULL,

PRIMARY KEY (`idVisit`, `idDoctor`),

CONSTRAINT `fk\_Visit\_has\_Doctor\_Visit1`

FOREIGN KEY (`idVisit`)

REFERENCES `new\_schema1`.`Visit` (`idVisit`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Visit\_has\_Doctor\_Doctor1`

FOREIGN KEY (`idDoctor`)

REFERENCES `new\_schema1`.`Doctor` (`SSN`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Visit\_has\_Doctor\_Doctor1\_idx` ON `new\_schema1`.`Visit\_has\_Doctor` (`idDoctor` ASC) VISIBLE;

CREATE INDEX `fk\_Visit\_has\_Doctor\_Visit1\_idx` ON `new\_schema1`.`Visit\_has\_Doctor` (`idVisit` ASC) VISIBLE;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

### Data Loading

This is an admittedly minimal set and is by no means exhaustive, but it was sufficient for my testing so far. It is possible that I missed a combination of attribute extremes or relationships.

-- Department Insert --

INSERT INTO department (Name)

VALUES ("Finance"), ("Surgery"), ("Oncology"), ("GI");

-- Specialty Insert --

INSERT INTO specialty (specialty)

VALUES ("Neurosurgery"), ("Cardiothoracic Surgery"), ("Pancreatic Cancer"), ("Microbiome");

-- Accountant Insert --

INSERT INTO accountant (SSN, FirstName, LastName, Salary, idDepartment)

VALUES (000000000, "Accountant", "Man", "1", 1);

-- Doctor Insert --

INSERT INTO doctor (SSN, FirstName, LastName, Salary, idDepartment, idSpecialty)

VALUES (000000001, "John", "Smith", 100000, 2, 1), (000000002, "Emma", "Watson", 100001, 2, 1), (000000003, "Edward", "Stark", 99999, 2, 2), (000000004, "Alex", "Dean", 80000, 3, 3), (000000005, "Melanie", "Mills", 130000, 4, 4);

INSERT INTO medication (name, company)

VALUES ("Valium", "BigPharma"), ("Oxycontin", "BigPharma"), ("Tylenol", "SmallPharma"), ("TLC", "Mom's Love");

INSERT INTO patient (FirstName, LastName, Insurance, SSN, Address)

VALUES ("Brendan", "Foley", 1, 000000006, "JHH East Baltimore"), ("Everett", "Vacek", 1, 000000007, "Somewhere"), ("Alex", "Nguyen", 0, 000000008, "There"), ("Lewis", "Hamilton", 1, NULL, "London, UK"), ("Nicholas", "Latifi", 0, NULL, "Not F1");

INSERT INTO room (Floor, Number)

VALUES (1, 101), (2, 202), (10, 1099);

INSERT INTO referencetype (type)

VALUES ("GI"), ("X-ray"), ("MRI");

INSERT INTO bed (idBed, Room\_Floor, Room\_Number)

VALUES (1, 1, 101), (2, 1, 101), (1, 2, 202), (2, 2, 202), (3, 2, 202), (1, 10, 1099);

INSERT INTO nurse (SSN, FirstName, LastName, Salary, idDepartment, Room\_Floor, Room\_Number)

VALUES (000000009, "Jenny", "From The Block", 70000, 1, 1, 101), (000000010, "Honey", "Badger", 80000, 2, 2, 202), (000000011, "Max", "Verstappen", 100000, 3, 10, 1099);

INSERT INTO reference (Name, Address, idDoctor, idPatient, idReferenceType, date)

VALUES ("Johns Hopkins Radiology", "10 Baltimore St.", 000000001, 1, 1, '2022-11-19');

INSERT INTO prescription (idDoctor, idPatient, idMedication, price, numdoses, date)

VALUES (000000002, 1, 1, 10.00, 100, '2021-11-18'), (000000002, 1, 1, 10.00, 100, '2022-11-18'), (000000001, 2, 2, 1.00, 5, '2022-11-1');

INSERT INTO visit (idPatient, start\_date, end\_date, idBed, Bed\_Room\_Floor, Bed\_Room\_Number)

VALUES (000000006, '2022-11-19', NULL, 1, 1, 101), (000000007, '2021-11-10', '2021-11-20', 1, 1, 101), (000000008, '2022-10-31', '2022-10-31', 2, 2, 202), (000000008, '2022-10-31', '2022-10-31', 2, 2, 202);

INSERT INTO visit\_has\_doctor (idVisit, idDoctor)

VALUES (1, 1), (2, 2), (3, 3), (4, 4)

### Table Viewing

SELECT \* FROM accountant;

Select \* FROM bed;

Select \* FROM department;

Select \* FROM doctor;

Select \* FROM medication;

Select \* FROM nurse;

Select \* FROM prescription;

Select \* FROM reference;

Select \* FROM referencetype;

Select \* FROM room;

Select \* FROM specialty;

Select \* FROM visit;

Select \* FROM visit\_has\_doctor;

## Appendix B

Index to the Data Dictionary.

|  |  |
| --- | --- |
| Column Name | Table Name |
| SSN | accountant |
| FirstName | accountant |
| LastName | accountant |
| Salary | accountant |
| idDepartment | accountant |
| idBed | bed |
| Room\_Floor | bed |
| Room\_Number | bed |
| Dname | department |
| idDepartment | department |
| Dname | department |
| Dnumber | department |
| Dnumber | department |
| Name | department |
| mgrssn | department |
| Mgrssn | department |
| mgrstartdate | department |
| Mgrstartdate | department |
| SSN | doctor |
| FirstName | doctor |
| LastName | doctor |
| Salary | doctor |
| idDepartment | doctor |
| idSpecialty | doctor |
| idMedication | medication |
| name | medication |
| company | medication |
| SSN | nurse |
| FirstName | nurse |
| LastName | nurse |
| Salary | nurse |
| idDepartment | nurse |
| Room\_Floor | nurse |
| Room\_Number | nurse |
| idPatient | patient |
| FirstName | patient |
| LastName | patient |
| Insurance | patient |
| SSN | patient |
| address | patient |
| idDoctor | prescription |
| idPatient | prescription |
| idMedication | prescription |
| price | prescription |
| numdoses | prescription |
| date | prescription |
| idReference | reference |
| Name | reference |
| Address | reference |
| idDoctor | reference |
| idPatient | reference |
| idReferenceType | reference |
| date | reference |
| idReferenceType | referencetype |
| type | referencetype |
| Floor | room |
| Number | room |
| idSpecialty | specialty |
| specialty | specialty |
| idVisit | visit |
| start\_date | visit |
| end\_date | visit |
| idBed | visit |
| Bed\_Room\_Floor | visit |
| Bed\_Room\_Number | visit |
| idPatient | visit |
| idVisit | visit\_has\_doctor |
| idDoctor | visit\_has\_doctor |

# References

1. DataResearchLabs. “sql\_scripts” (2021) *GitHub.* MySQL,Retrieved from https://raw.githubusercontent.com/DataResearchLabs/sql\_scripts/main/mysql/data\_dictionary/data\_dict\_dump.sql
2. *Installing and Launching MySQL Workbench*. MySQL :: Mysql workbench :: 3.1 hardware requirements. (n.d.). Retrieved November 19, 2022, from http://download.nust.na/pub6/mysql/doc/workbench/en/wb-requirements-hardware.html