CS 4354: Concepts of Database Systems Spring 2023

Final Report

May 5, 2023

# Project Title and Group Members

Hospital Database Management System

# Problem Statement and Motivation:

A hospital needs to keep track of its patients, doctors, nurses, departments, and procedures in an organized and efficient manner. The hospital requires a database management system to manage and organize all the information and provide reli- able way to search, modify, and retrieve the data.

A relational database is suited for this problem because it involves multiple entities with complex relationships between them. For instance, a hospital has many departments, and each department has many doctors, nurses, and patients. Patients are assigned to doctors, who work in specific departments. Procedures are performed by the doctors. All these entities have complex relationships that can be captured in a relational database. On top of this, relational database allows for efficient and reliable querying and retrieval of information, which is important in a hospital setting as quick access to patient and medical information is very critical. The use of schemas and relationships between them makes it easier to ensure data integrity and prevent inconsistencies or errors in the data.

Managing a hospital’s operations and providing quality health care services to patients is a challenging task that involves handling a large amount of data. The hospital needs to manage the information about patients, doctors, nurses, and de- partments efficiently to ensure that patients receive the right care at the right time. By using the hospital management database, it reduces the chances of errors and improves patient care which is our primary motivation. Additionally, hospital can also use data analytics to generate reports and analyze trends to make informed decisions about its operations.

# Requirements details

The key entities in the database include hospitals, departments, nurses, doctors, patients, and procedures. The key relationships we need to consider are hospital- department, department-doctor, doctor-nurse, doctor-patient, and doctor- proce- dures. In the database, each hospital can have one to many departments and each department will only belong to one hospital. Each doctor can belong to only one department while each department can have one to many doctors. Each doctor also has one to many nurses assigned to them and each nurse is assigned to only one doctor. Each patient can have one to many doctors and each doctor can have one to many patients creating a many-to-many relationship. Also, each doctor performs one to many procedures and each individual procedure is performed by only one doctor. The organizational constraint that we had to be cautious of in- cluded the security and privacy of staff and patient information. Some logical constraints that we had to keep in mind were data integrity and data consistency. Overall, the database is intended to be user-friendly, reliable, and scalable to meet the growing needs of hospital organizations.

In short:

1. Managing hospital details: The database should be able to store and manage details of hospitals including their name, address, and contact details.
2. Managing department details: The database should be able to store and manage details of departments within each hospital, including their name and the hospital they belong to.
3. Managing doctor details: The database should be able to store and manage details of doctors including their name, specialization, contact details, and salary. Each doctor should be assigned to a specific department.
4. Managing patient details: The database should be able to store and man- age details of patients including their name, date of birth, sex, address, and phone number. Each patient should be assigned to a specific doctor.
5. Managing nurse details: The database should be able to store and manage details of nurses including their name, contact details, salary, and the de- partment they belong to. Each nurse should report to a specific doctor.
6. Managing procedure details: The database should be able to store and man- age details of medical procedures, including their name and the doctor who performs them.
7. Maintaining data integrity: The database should ensure data integrity by enforcing foreign key constraints between the various tables to ensure that records are correctly linked.
8. Reporting: The database should be able to generate reports based on various parameters such as the number of patients in each department or the number of procedures performed by each doctor.

# Conceptual Model

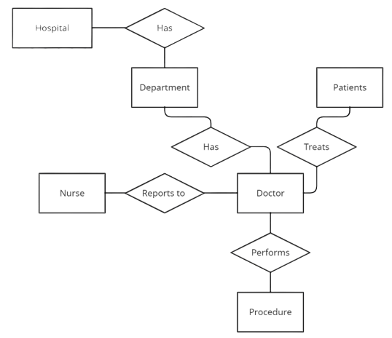


Figure 1: Conceptual Model for Hospital Database Management System

For our conceptual design, we used an ER diagram to give a visual repre- sentation of the entities and their relationships to one another. Our key entities were represented in the conceptual model by rectangles and the key relationships are represented by diamonds. The main purpose of our conceptual design is to provide a clear and organized way of representing and understanding the data re- quirements of the hospital management system.

# Logical Model

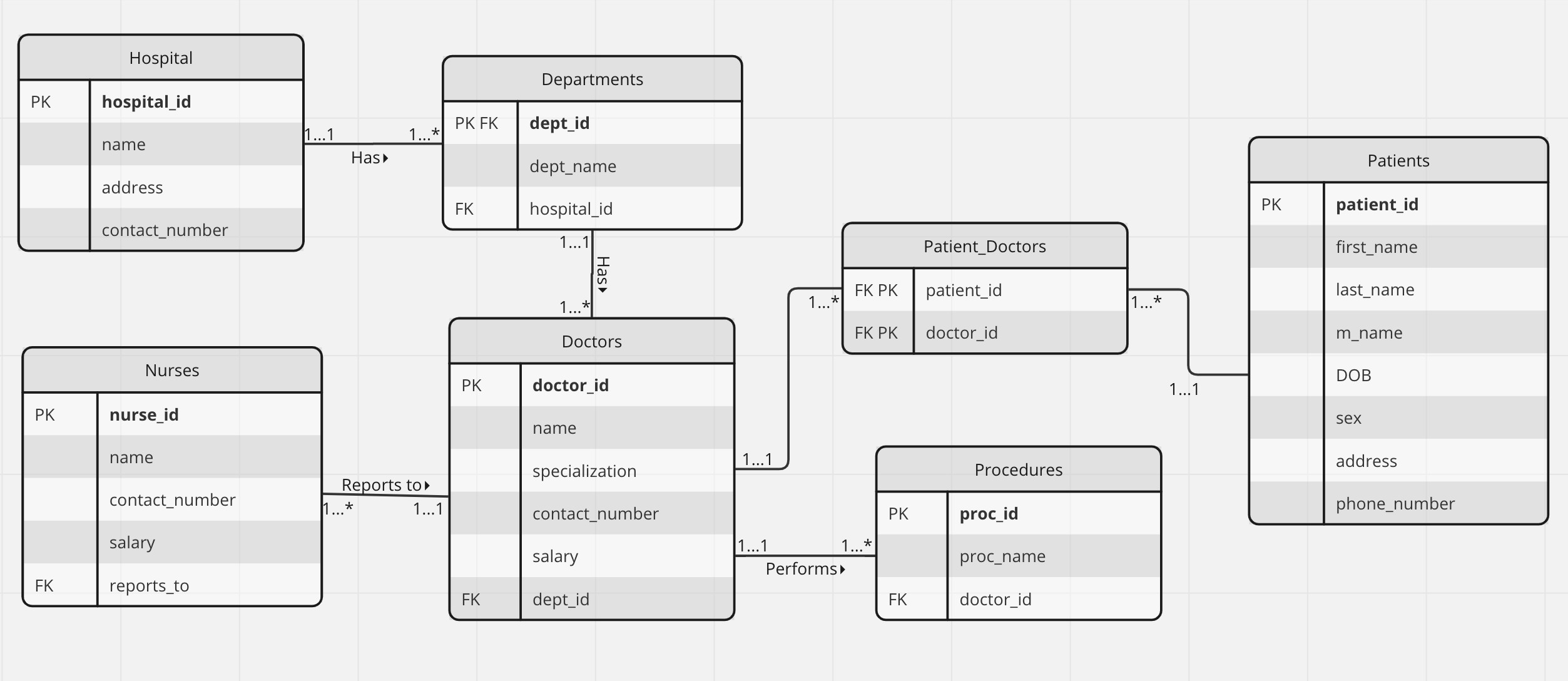


Figure 2: Logical Model for Hospital Database Management System

In the logical model for the hospital management database system, we have used primary keys for all tables. Each table has a unique primary key that does not change. For example, we used hospital id as the primary key for the Hospital table, dept id for the Departments table, nurse id for the Nurses table, doctor id for the Doctors table, patient id for the Patients table, and proc id as the primary key for the Procedures table.

In the logical model, all relationships between the tables are established using foreign keys. For instance, hospital id is used as a foreign key in the Departments table to reference back to the Hospital table. Similarly, dept id in the Doctors ta- ble is used to reference the Departments table, doctor id in the Procedures table is used to reference the Doctors table, and reports to is used in the Nurses table the reference the primary key in the Doctors table. Additionally, we have created a

joining table for the many-to-many relationship between the Patients and Doctors tables. This joining table uses the primary keys from both tables as foreign keys to establish the relationship. This design choice was made to avoid creating a multi-valued attribute in either of the tables which could lead to data redundancy and inconsistencies.

In short:

1. Hospital Table:The primary key for the Hospital table is the hospital id which is a surrogate key. This choice was made because a hospital may have multiple physical locations and it is easier to manage the data using a unique identifier than relying on natural keys such as hospital names which might be subject to change.
2. Departments Table:The primary key for the Departments table is dept id which is also a surrogate key.A department can have a unique name but there may be different departments with the same name in different hospitals. Hence, a surrogate key is useful for uniquely identifying a department.
3. Doctors Table:The primary key for the Doctors table is doctor id. This is also a surrogate key. Although a doctor may have a unique name and specialization, it is possible that multiple doctors with the same name and specialization may exist in a hospital. Hence, surrogate key ensures better uniqueness. The foreign key for dept id ensures that each doctor is associ- ated with only one department.
4. Patients Table:The primary key for the Patients table is patient id. This is also a surrogate key. A patient may have a unique name and date of birth, but multiple patients with the same name and date of birth can exist. Hence, a surrogate key is useful for uniquely identifying a patient.
5. Nurses Table: The primary key for the Nurses table is nurse id. This is also a surrogate key for the same reasons as above. The foreign key for reports to ensures that each nurse reports to only one doctor.
6. Procedures Table: The primary key for the Procedures table is proc id. This is also a surrogate key for the same reasons as above. The foreign key doctor id ensures that each procedure is associated with only one doctor.

Hence, using a surrogate key rather than a natural key (such as the hospital name) allows for better performance in database operations and ensures data integrity in cases where the natural key could be subject to change.

The design is normalized up the third normal form (3NF), ensuring that there is no redundant data in any of the tables. We have also checked for fan traps and chasm traps and mitigated them by breaking down the tables into separate entities and relationships.

Overall, this logical model provides an efficient and effective way to man- age the hospital’s operations and information. With unique primary keys, properly established relationships using foreign keys, and normalization up to the 3NF and no denormalization required, the database system can help hospitals manage patient, doctor, nurse, and procedure information.

# Physical Model

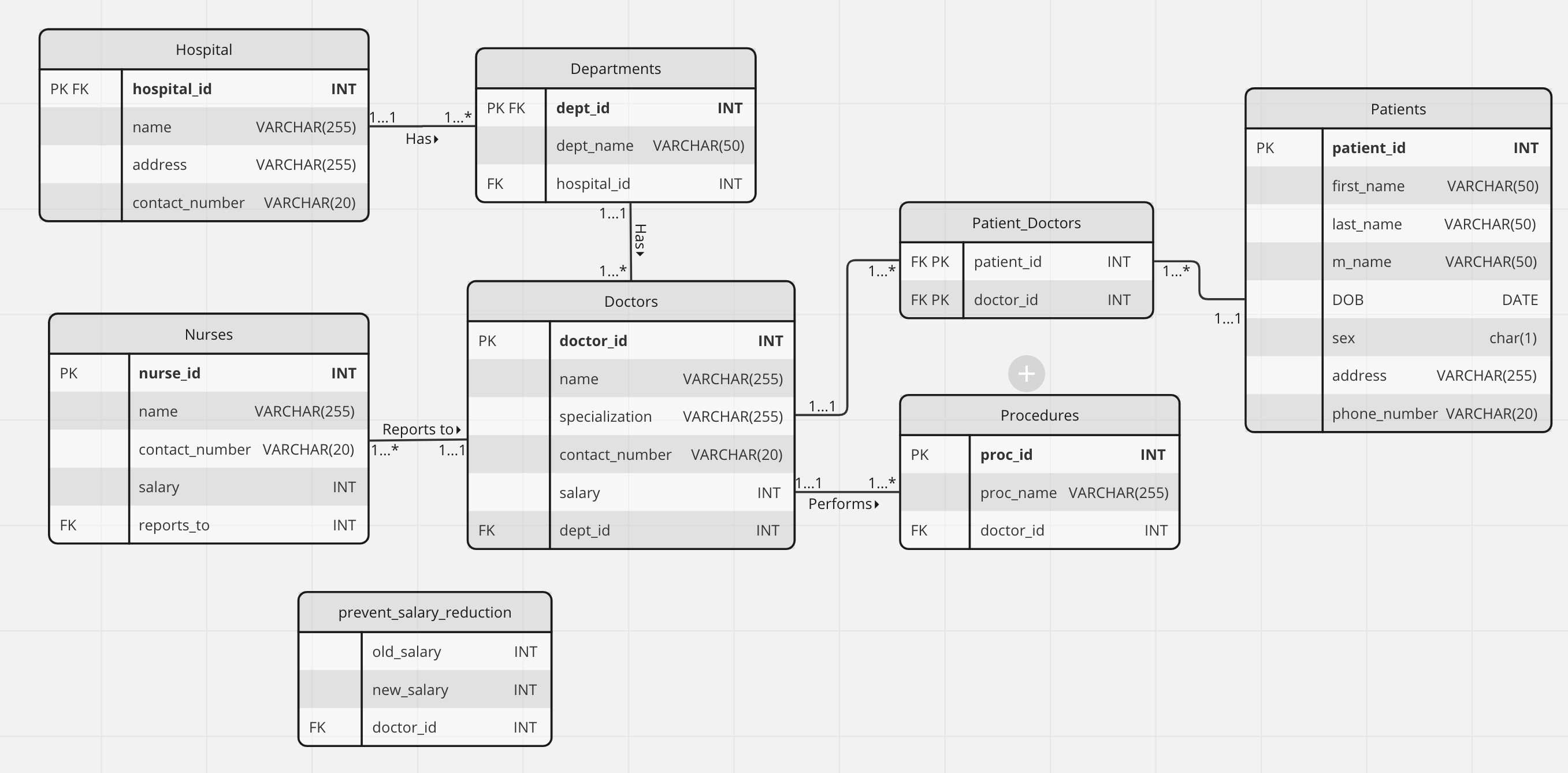


Figure 3: Physical Model for Hospital Database Management System

## Engine, Character set and Collation

1. We have used MySQL version 8.0.29 which uses **InnoDB** as the default storage engine. It provides ACID (atomicity, consistency, isolation, and durability) transaction support.
2. To ensure compatibility across platforms and support all Unicode charac- ters, we have selected UTF-8-MB4 as the character set.
3. We have selected ”UTF-8-MB4 0900 AI CI” collation sequence. It spec- ifies how characters are sorted and compared. In this case, it uses the ”accent-insensitive” and ”case-insensitive” comparison options, which means that accents and case differences are ignored when comparing strings.

We have used appropriate data types for each field attribute, such as INTEGER for primary keys, VARCHAR for name, address, and contact number fields, DATE for DOB, and CHAR for the sex attribute in the Patients table.

The trigger **prevent salary reduction** is necessary to ensure that the salary of a doctor is not reduced below the current value. This trigger will be executed be- fore an update operation is performed on the doctors table and will check whether the new salary is less than the current salary. If the new salary is less, then the trigger will raise a custom error message that will be displayed to the user instead of the default error message generated by the database management system and prevent the update operation from being executed.

The event **nurse salary increase** is necessary to automatically increase the salary of all nurses in the database by 5% every 6 months if their salary is less than 80000. This event is scheduled to run every 6 months and will update the salaries of all eligible nurses in the nurses table.

Many triggers degrade database performance. However, short and simple triggers will perform better which is why we have used only one and short trigger. Both the trigger and event are necessary to enforce business rules and automate certain tasks, respectively. The trigger ensures that the salary of a doctor is never reduced below the current value, which is an important business rule to maintain fairness and retain talent. The event automates the process of increasing the salaries of nurses, which is a repetitive task that can be prone to errors if done manually.

Creating an index involves additional storage space and maintenance overhead. It may not be necessary to create an index on a column because our table is small. When we extend our tables in future, we will be implementing indexes over fre- quent queries.

Views **Family Medicine Patients** and **Orthopedics Patients** were created to re- trieve the patient information for all patients who are being treated by doctors in the ”Family Medicine” and ”Orthopedics” department respectively. These views provide a simplified way to retrieve information from the database, as the com- plex JOIN operations are already performed in the view definition. These views enhances security as it can be used to restrict access to certain data. Additionally, if there are any changes in the underlying tables, the views can be easily updated without affecting the applications that use them.

Hence, the choice to implement these triggers, events, and views was made to en- hance the system’s efficiency, usability, and security. We believe that these design choices will ensure that the system functions effectively and provides a seamless user experience.

# Images from Demo

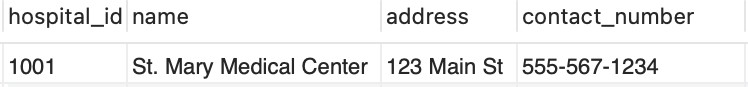


Figure 4: Hospital Table

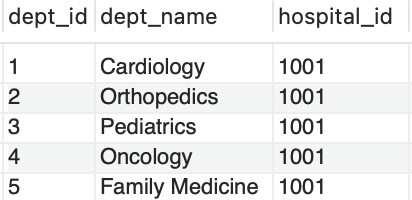


Figure 5: Departments Table

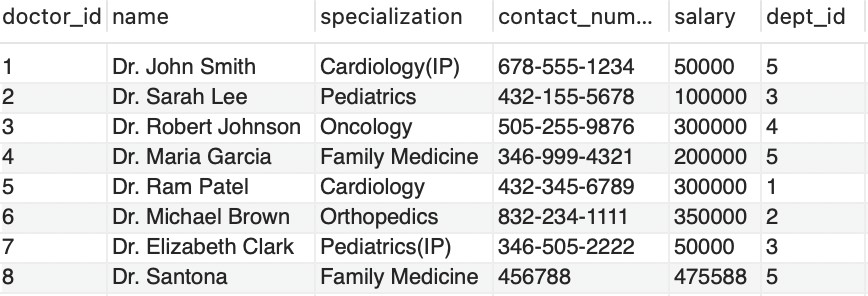


Figure 6: Doctors Table

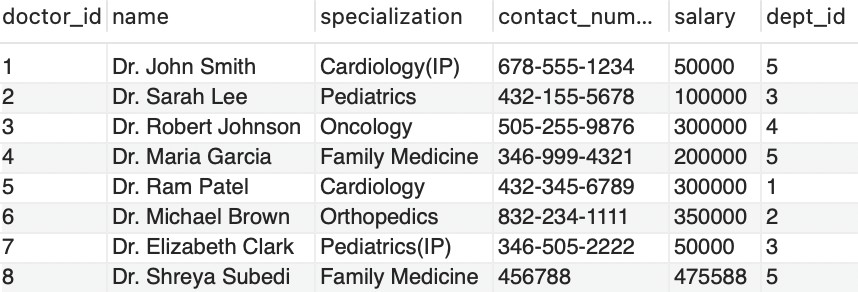


Figure 7: Doctors updated Table



Figure 8: Patients Table

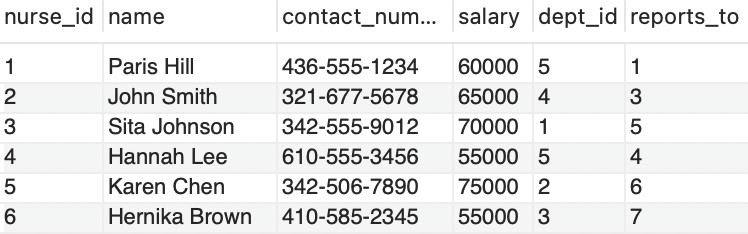


Figure 9: Nurses Table

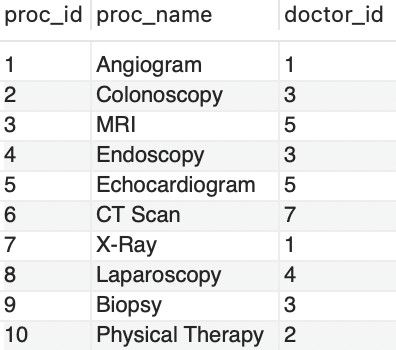


Figure 10: Procedures Table

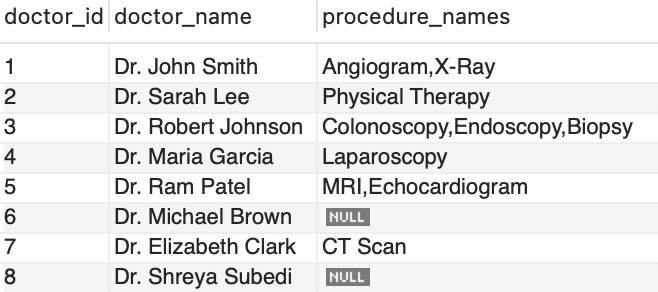


Figure 11: All the doctors who have performed the procedures along with the ones who have not.

SELECT d.doctor id, d.name AS doctor name, GROUP CONCAT(p.proc name) AS procedure names

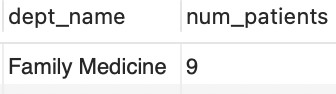
FROM Doctors d

LEFT JOIN Procedures p ON d.doctor id = p.doctor id GROUP BY d.doctor id;

Figure 12: All the nurses who earn more than the doctors they report.

SELECT T1.nurse id, T1.name FROM Nurses T1, Doctors T2 WHERE

T1.reports to = T2.doctor id AND T1.salary ¿ T2.salary;

Figure 13: Retrieves the department with the highest number of patients SELECT d.dept name, COUNT(pd.patient id) AS num patients

FROM Departments d

JOIN Doctors doc ON d.dept id = doc.dept id

JOIN Patients Doctors pd ON doc.doctor id = pd.doctor id GROUP BY d.dept name

ORDER BY num patients DESC LIMIT 1;

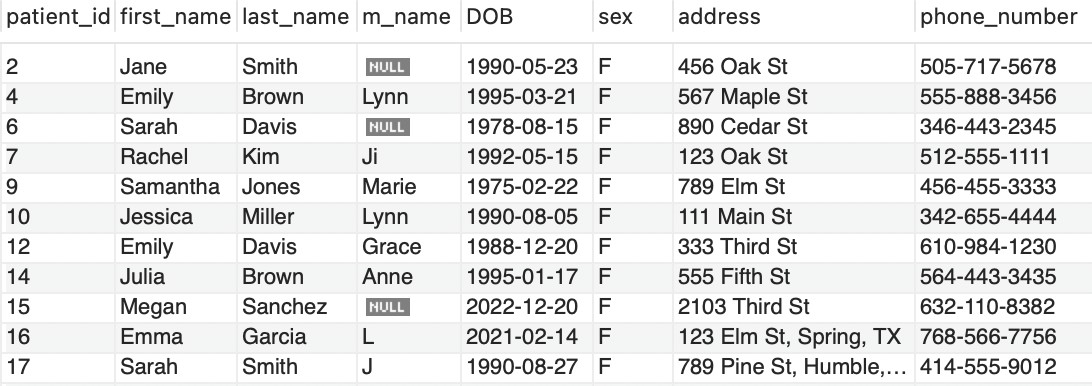


Figure 14: Retrieves all the female patients SELECT \* FROM Patients WHERE sex LIKE ’F%’;

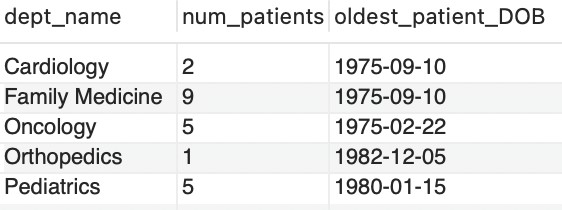


Figure 15: The number of patients associated with the specific department, and the date of birth of the oldest patient associated with that department.

SELECT d.dept name, COUNT(DISTINCT pd.patient id) AS num patients, MIN(p.DOB) AS oldest patient DOB

FROM Departments d

LEFT JOIN Doctors doc ON d.dept id = doc.dept id

LEFT JOIN Patients Doctors pd ON doc.doctor id = pd.doctor id LEFT JOIN Patients p ON pd.patient id = p.patient id

GROUP BY d.dept name;

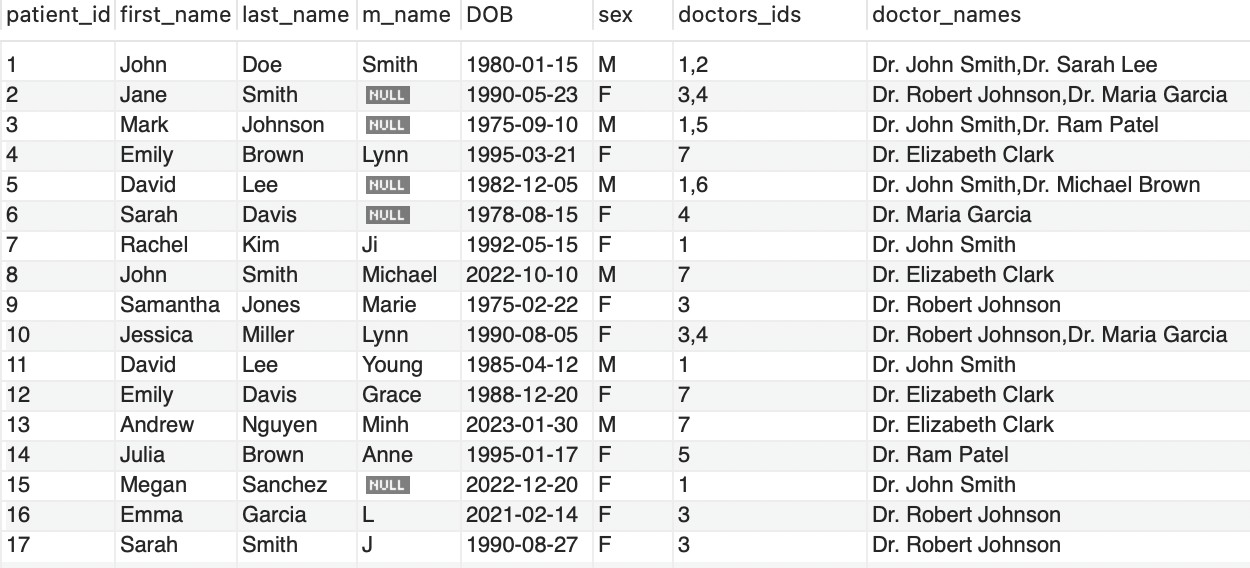


Figure 16: A list of patients name, id and their treating doctor’s name and ids separated by comma, where multiple doctors may treat multiple patients and vice versa.

SELECT p.patient id, p.first name, p.last name, p.m name, p.DOB,p.sex, GROUP CONCAT(d.doctor id SEPARATOR ’,’) AS doctors ids, GROUP CONCAT(d.name) AS doctor names

FROM Patients p

JOIN Patients Doctors pd ON p.patient id = pd.patient id JOIN Doctors d ON pd.doctor id = d.doctor id

GROUP BY p.patient id;

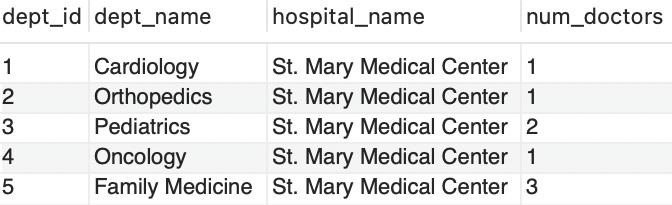


Figure 17: Retrieve the list of all departments along with the hospital they belong to and the number of doctors in each department:

SELECT Departments.dept id, Departments.dept name, Hospital.name AS hos- pital name, COUNT(Doctors.doctor id) AS num doctors

FROM Departments

INNER JOIN Hospital ON Departments.hospital id = Hospital.hospital id LEFT JOIN Doctors ON Departments.dept id = Doctors.dept id

GROUP BY Departments.dept id;

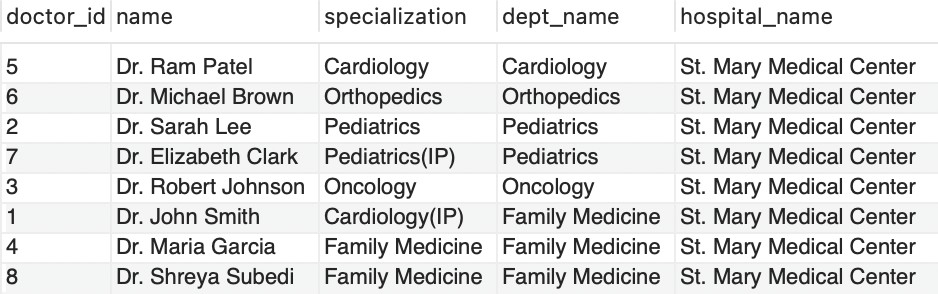


Figure 18: Retrieve the list of all doctors along with their specialization, the de- partment they belong to, and the hospital they work in

SELECT Doctors.doctor id, Doctors.name, Doctors.specialization, Departments.dept name, Hospital.name AS hospital name FROM Doctors INNER JOIN Departments ON

Doctors.dept id = Departments.dept id INNER JOIN Hospital ON Departments.hospital id

= Hospital.hospital id;



Figure 19: Prevents the salary reduction for doctors.

DELIMITER //

CREATE TRIGGER prevent salary reduction BEFORE UPDATE ON doctors

FOR EACH ROW BEGIN

IF NEW.salary ¡ OLD.salary THEN SIGNAL SQLSTATE ’45000’

SET MESSAGE TEXT = ”Salary cannot be reduced!”; END IF;

END // DELIMITER ;

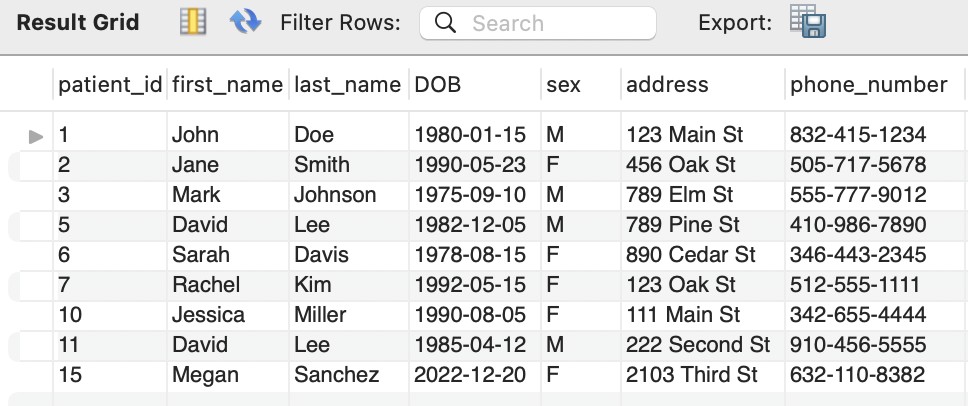
UPDATE doctors set salary = 4000 where doctor id = 1;

Figure 20: Retrieves all the Family Medicine patients CREATE VIEW Family Medicine Patients AS

SELECT p.patient id, p.first name, p.last name, p.DOB, p.sex, p.address, p.phone number FROM Patients p JOIN Patients Doctors pd ON p.patient id = pd.patient id JOIN

Doctors d ON pd.doctor id = d.doctor id JOIN Departments dept ON d.dept id = dept.dept id WHERE dept.dept name = ’Family Medicine’ ORDER BY p.patient id;

# Discussion

## Strengths:

* + 1. The hospital database efficiently stores and manages various data types, such as patients, doctors, nurses, procedures, and departments, ensur- ing well-organized and easily accessible data with clear relationships between different types.
    2. Data integrity is enforced through primary and foreign key connec- tions, and all tables are in 3rd normalized form and no denormalization required, reducing redundancy and improving data integrity.
    3. Overall, the database has been designed with scalability, performance, security, and data accuracy in mind, making it a solid foundation for future expansion.

## Weaknesses:

* + 1. Security may become an issue when keeping information private.
    2. Due to security issues, there could be potential violation of HIPPA laws of patients.

## Ommissions

* + 1. More detailed data like ssn of patients were removed.
    2. Insurance information was removed.
    3. Additional staff roles

## Potential Extensions

* + 1. A table to track patients appointments which would allow for schedul- ing patient’s visit.
    2. Deletion of the patients on the system not seen over a year.
    3. Billings table to track patient’s payment.