USE CASE STUDY REPORT

Group No.: Group 05

Student Names: Khushboo Galrani and Thao

Executive Summary:

This study focuses on the design and deployment of a comprehensive relational database system using SQL for efficient hospital management. The system covers the entire patient journey within a hospital setting, aiming to optimize workflow, staff coordination, and patient care.

Patients initiate their interaction with the system by providing personal information during registration, including name, phone number, date of birth, social security number, career details, and medical history. The system then monitors vital signs such as heart rate, blood pressure, respiratory rate, and temperature. Upon completion of the monitoring phase, patients consult with doctors to discuss their conditions, and based on this information, doctors prescribe necessary diagnostic tests. Test results, along with experiment details and technician information, are recorded and sent directly to the treating doctor.

The diagnostic phase is followed by accurate diagnoses and the creation of personalized treatment plans accessible to patients through a portal. The system facilitates decision-making on hospitalization or outpatient treatment, with scheduled follow-up appointments to assess patient conditions, track recovery progress, and address complications.

The system includes a pharmacy department for medication distribution, accompanied by patient education on usage, dosages, and potential side effects. A patient care center tailors treatment plans to individual preferences and budget constraints. Administrative functions are integral, with the administration department managing daily hospital operations, including budgeting, staffing, and regulatory compliance. The finance department handles billing, insurance claims, and revenue management. The patient outcomes department closely monitors patient feedback and outcomes to facilitate continuous improvement.

Database implementation involved a conceptual model using EER and UML, leading to a relational model with primary and foreign keys. Migration to RDBMS platforms and exploration of Neo4j provided insights into database execution and storage mechanisms. SQL queries enabled data retrieval, and integration with Python or R facilitated the generation of visualizations, revealing valuable insights into correlations between medical conditions and blood types or trends in doctors' salaries.

In summary, this study successfully implemented various data management techniques, creating a robust hospital management system that optimizes workflow, enhances patient care, and supports data-driven decision-making within the healthcare setting.

I. Introduction

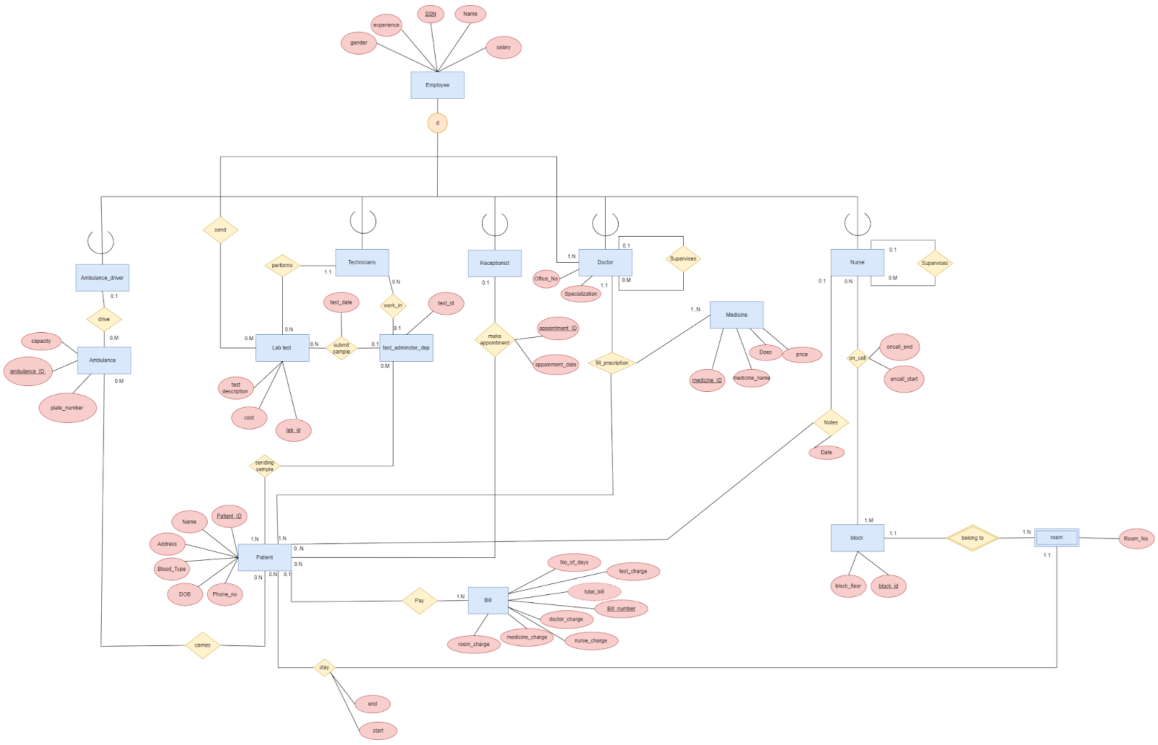
The healthcare system is the primary concern in numerous nations, and most governments place a high emphasis on leveraging well-being, and health for their citizens.For instance, the American government has spent trillions of dollars in the healthcare system, bolstering the hospital facility and training staffs annually to elevate the health for residence. Furthermore, the operation in the hospital should be refined and expanded each year, leaning toward to the most swifter and efficient patient care .

The hospital has many specialized hospital departments, various teams, and an enormous system to robust the process productively. Once individuals approach hospitals, the patient will interact with the healthcare information department, the interdisciplinary department, patient history and monitoring, diagnosis and treatment, patient outcomes, administration, and then finance management. Last but not least, hospitals also offer various healthcare services ranging from emergency care to elective care, long-term care, and rehabilitation.

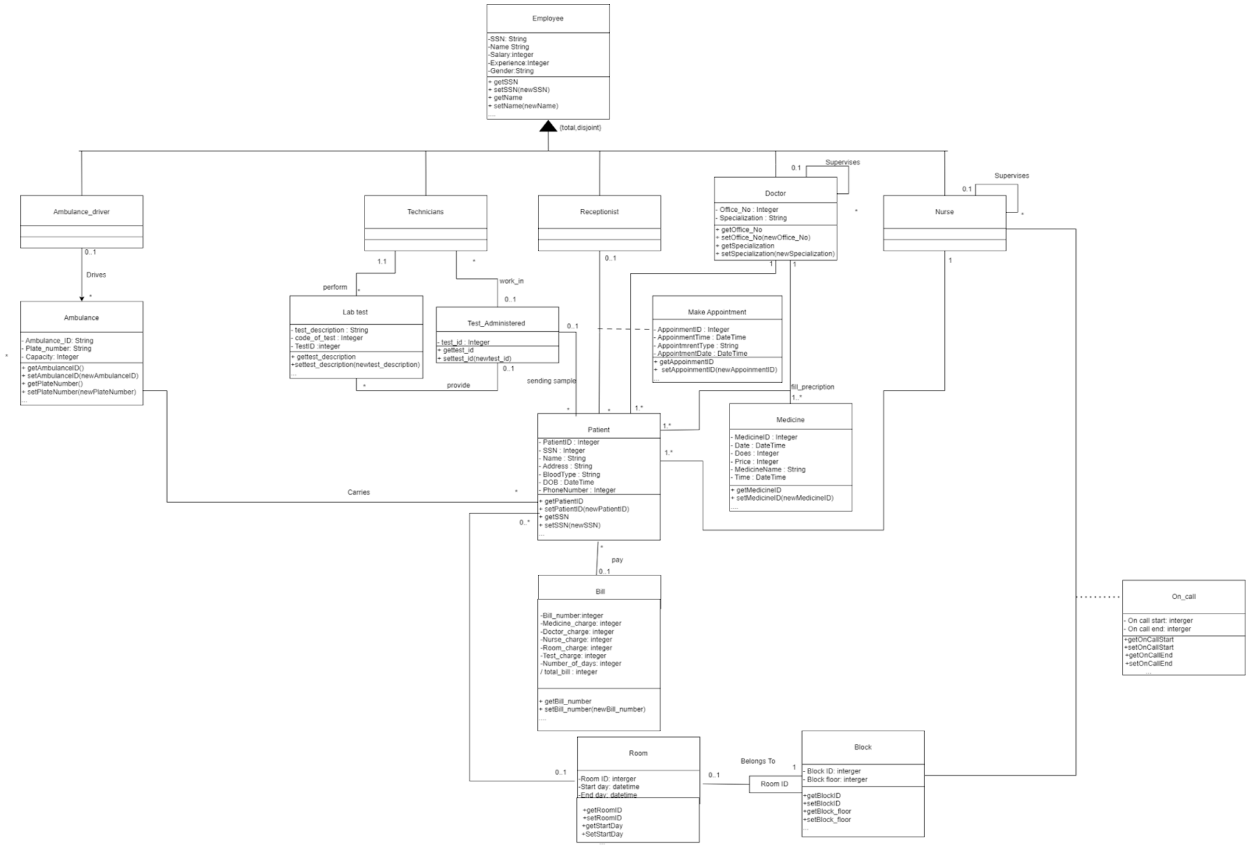
To manage the sophisticated system and support the operation effectively, the cutting-edge technology is the core factor that can be insufficient.That is the reason why our project aims to create the hospital management model from the perspective of the medical manager, showing how to enhance and execute the treatment process quickly and productively.

II. Conceptual Data Modeling

1. EER



2. UML



III. Mapping Conceptual Model to Relational Model

Primary Key - Bold **Foreign Key - Italicized**

Employee(**SSN**, Name, Salary, Experience, Gender)

Nurse(***Nurse_ID***, superviser)

Nurse_ID: foreign key refers to SSN in relation Employee, NOT NULL

Doctor(***Doctor_ID***,Office_No, Specilization, superviser)

Doctor_ID: foreign key refers to SSN in relation Employee, NOT NULL

Receptionist(***Receptionist_ID***)

Receptionist_ID: foreign key refers to SSN in relation Employee, NOT NULL

Test_administer_dep(**Test_id**)

Technicians(***Technicians_ID****, Test_ID*)

Technicians_ID: foreign key refers to SSN_employee in relation Employee, NOT NULL

Test_ID: foreign keys refers to *Test_ID* in the relation test_administer_dep, NULL allowed

Ambulance_driver(***Driver_ID***)

driver_ID : foreign key refers to SSN in relation Employee, NOT NULL

Ambulance(**Ambulance_ID**, plate_number, capacity, *Driver_ID)*

Driver_ID: foreign key refers to driver_ID in relation in Ambulance_driver, NULL allowed

Block(**Block_ID**, Block_Floor)

Room(**Room_No**, ***Block_ID***)

Block_ID: foreign key refers to Block_ID in relation in Block, NOT NULL

Patient(**Patient_ID**, Name, Address, Blood_Type, DOB, Phone_no, Date, *Nurse_ID*, *Doctor_ID)*

Nurse_ID: foreign key refers to Nurse_ID in relation in Nurse, NOT NULL

Doctor_ID: foreign key refers to Doctor_ID in relation in Nurse, NOT NULL

Patient_Room(***Patient_ID, Room_No, Block_ID,*** start, end)

Room_No: foreign key refers to Room_No in relation in Room, NOT NULL

Block_ID: foreign key refers to Block_ID in relation in Block, NOT NULL

Patient_ID: foreign key refers to Patient_ID in relation in Patient, NOT NULL

Appointment(**Appointment_ID,** Appointment_Date, *Patient_ID*, *Receptionist_ID)*

Patient_ID: foreign key refers to Patient_ID in relation in Patient, NOT NULL

Receptionist_ID: foreign key refers to Receptionist_ID in relation in Receptionist, NOT NULL

Carries(***Patient_ID,Ambulance_ID***)

Patient_ID: foreign key refers to Patient_ID in relation in Patient, NOT NULL

Receptionist_ID: foreign key refers to Receptionist_ID in relation in Receptionist, NOT NULL

On_call(***Nurse_ID, Block_ID***, **date,** oncall_end, oncall_start)

Nurse_ID: foreign key refers to Nurse_ID in relation in Patient, NULL allowed

Block_ID: foreign key refers to Block_ID in relation in Block, NULL allowed

Bill(**Bill_number**, medicine_charge, doctor_charge, nurse_charge, test_charge, room charge, No_of_days ,Total_bill, *Patient_ID*)

Patient_ID : foregin key refers to patient ID in relation Patient, NOT NULL

Fill_precription(***Medicince_ID, Patient_ID, Doctor_ID***)

Medicine_ID: foreign key refers to medicine_ID in relation medicine, NOT NULL

Patient_ID : foregin key refers to patient_ID in relation Patient, NOT NULL

Doctor_ID: foreign key refers to Doctor_ID in relation in Nurse, NOT NULL

Medicine(**Medicine_ID**, medicine_name, price, does)

Lab_test(**Lab_ID**, test description, cost_of_test, *test_id*, *Technician_id*)

test_id: foreign key refers to test_id in the relation test_administer_dep

Technician_ID: foreign key refers to Technician_ID in relation in Technician, NOT NULL

Sending_sample(***Test_ID***, ***Patient_ID***)

Test_id is the foreign key refers to test_id in the relation test administered: NULL allowed

Patient_id is the foreign key refers to patient id in the relation patient: NULL allowed

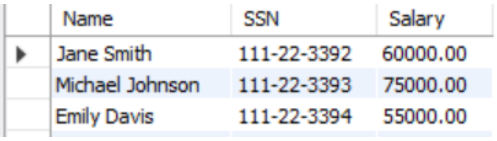
IV. Implementation of Relation Model via MySQL and NoSQL

MySQL Implementation :

The database was created in MySQL and the following queries were performed :

Query 1: Retrieve employees with experience greater than 5 years

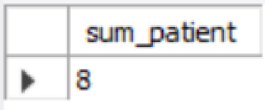
```
SELECT Name, SSN, Salary
FROM Employee
WHERE Experience > 5;
```



Name	SSN	Salary
Jane Smith	111-22-3392	60000.00
Michael Johnson	111-22-3393	75000.00
Emily Davis	111-22-3394	55000.00

Query 2: To count the number of patients who paid more than 5000$

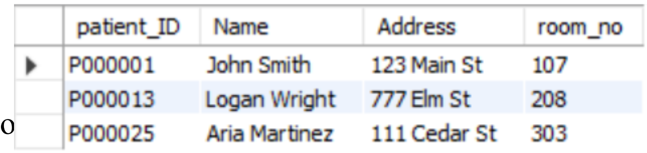
```
SELECT COUNT(Patient) AS sum_patient
FROM (
   SELECT a.Patient_ID AS Patient
   FROM Patient a
   LEFT JOIN Bill b ON a.Patient_ID = b.Patient_ID
   GROUP BY a.Patient_ID
   HAVING SUM(b.Total_bill) > 5000.00 ) AS a;
```



sum_patient
8

Query 3: Retrieve patients with patient_ID, address, and room number who stayed in the hospital in January 2023

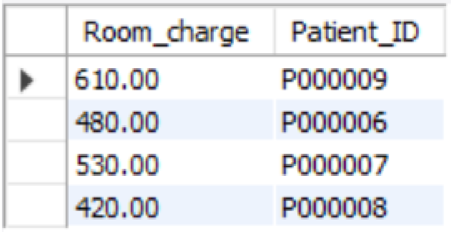
```
SELECT a.Patient_ID, a.Name, a.Address,
   (SELECT b.Room_No FROM Patient_Room b
WHERE a.Patient_ID = b.Patient_ID) AS room_no
FROM Patient a
WHERE MONTH(a.Date) = 1 AND YEAR(a.Date) = 2023;
```



patient_ID	Name	Address	room_no
P000001	John Smith	123 Main St	107
P000013	Logan Wright	777 Elm St	208
P000025	Aria Martinez	111 Cedar St	303

Query 4: Determine how much patients who stay in Block B in 2023 spend

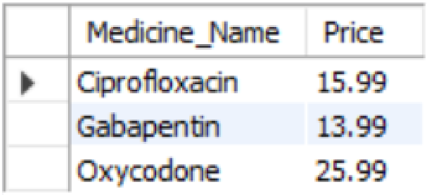
```
SELECT b.Room_charge, r.Patient_ID
FROM Patient_Room r
INNER JOIN Bill b ON b.Patient_ID = r.Patient_ID
WHERE r.Block_ID IN (
   SELECT Block_ID FROM Patient_Room
   WHERE Block_ID = 'B' AND
   YEAR(Start) = 2023);
```



Room_charge	Patient_ID
610.00	P000009
480.00	P000006
530.00	P000007
420.00	P000008

Query 5: Retrieve the 3 most expensive medicines

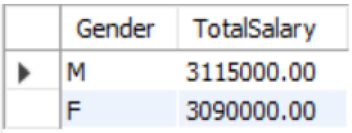
```
SELECT a.Medicine_Name, a.Price
FROM Medicine a
WHERE 3 > (
   SELECT COUNT(*)
   FROM Medicine b
   WHERE a.Price < b.Price );
```



Medicine_Name	Price
Ciprofloxacin	15.99
Gabapentin	13.99
Oxycodone	25.99

Query 6: Compare gender and salary

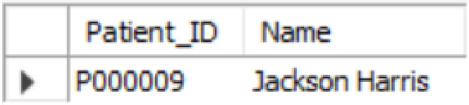
```
SELECT Gender, SUM(Salary) AS TotalSalary
FROM Employee
GROUP BY Gender;
```



Gender	TotalSalary
M	3115000.00
F	3090000.00

Query 7: Retrieve patients whose total bill is maximum

```
SELECT B.Patient_ID, P.Name
FROM Bill B INNER JOIN Patient P
ON B.Patient_ID = P.Patient_ID
WHERE B.Total_bill >= ALL (SELECT B.Total_bill FROM Bill B);
```

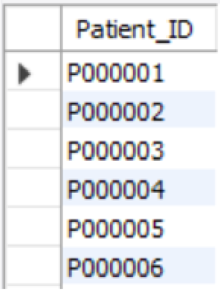


Patient_ID	Name
P000009	Jackson Harris

Query 8: Retrieve the unique list of patients who either have a prescription or have undergone a lab test

```
SELECT DISTINCT Patient_ID
```

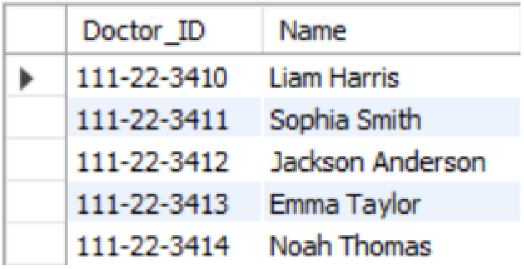
```
FROM (
   SELECT Patient_ID FROM Fill_prescription
   UNION
   SELECT Patient_ID
   FROM sending_sample s INNER JOIN lab_test l
   ON s.test_id = l.test_id ) AS A
ORDER BY 1;
```



Patient_ID
P000001
P000002
P000003
P000004
P000005
P000006

Query 9: Retrieve the list of doctors who do not have patients in Block C in 2023

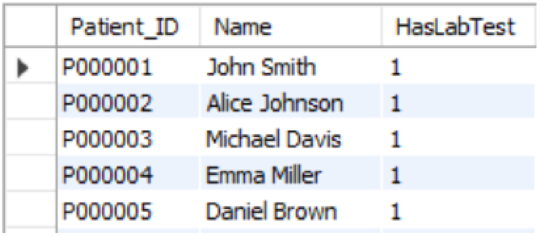
```
SELECT DISTINCT d.Doctor_ID, e.Name
FROM Employee e INNER JOIN doctor d
ON e.SSN = d.Doctor_ID
LEFT JOIN Patient p
ON d.Doctor_ID = p.Doctor_ID
WHERE p.Patient_ID IS NULL OR
NOT EXISTS (
   SELECT 1
   FROM Patient_Room r
   WHERE r.Patient_ID = p.Patient_ID
   AND r.Block_ID = 'C'
   AND YEAR(r.Start) = 2023);
```



Doctor_ID	Name
111-22-3410	Liam Harris
111-22-3411	Sophia Smith
111-22-3412	Jackson Anderson
111-22-3413	Emma Taylor
111-22-3414	Noah Thomas

Query 10: List patients along with a flag indicating whether they have undergone any lab test

```
SELECT Patient_ID, Name,
   EXISTS (
      SELECT 1
      FROM sending_sample s
      INNER JOIN
      lab_test lt
      ON s.test_id = lt.test_id
      WHERE Patient.Patient_ID = s.Patient_ID
   ) AS HasLabTest
FROM Patient;
```



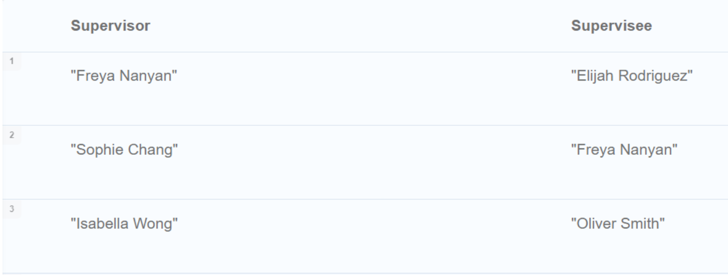
Patient_ID	Name	HasLabTest
P000001	John Smith	1
P000002	Alice Johnson	1
P000003	Michael Davis	1
P000004	Emma Miller	1
P000005	Daniel Brown	1

NoSQL Implementation :

Three tables(Doctors, Patients, Disorder) and three relations(Cause_pain, is_checking, supervise) have been created in Neo4j playground. The following Cypher queries were performed:

Query 1: Retrieve the supervisor of each doctor

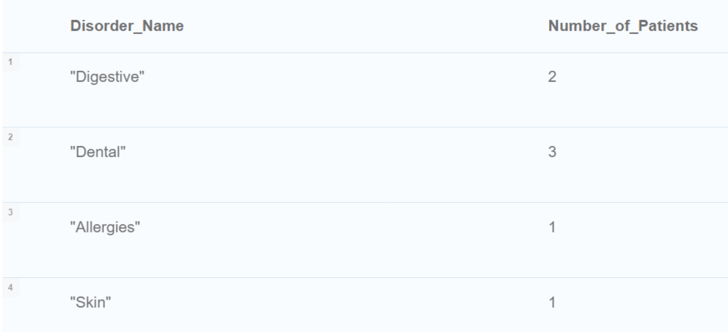
```
MATCH (supervisor:Doctor)-[:SUPERVISE]->(supervisee:Doctor)
RETURN supervisor.doctor_name AS Supervisor, supervisee.doctor_name AS Supervisee
```



	Supervisor	Supervisee
1	"Freya Nanyan"	"Elijah Rodriguez"
2	"Sophie Chang"	"Freya Nanyan"
3	"Isabella Wong"	"Oliver Smith"

Query 2: Query to count the number of patients for each disorder

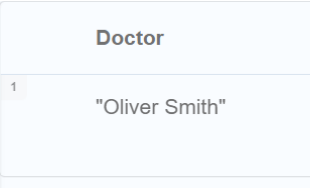
```
MATCH (disorder:DISORDERS)<-[:CAUSE_PAIN]-(patient:Patient)
RETURN disorder.name AS Disorder_Name, COUNT(patient) AS Number_of_Patients;
```



	Disorder_Name	Number_of_Patients
1	"Digestive"	2
2	"Dental"	3
3	"Allergies"	1
4	"Skin"	1

Query 3: To find a doctor who is checking the patient for the longest time

```
MATCH (doctor:Doctor)-[a:IS_CHECKING]->(patient:Patient)
With doctor, a.minutes AS checkingTime
ORDER BY checkingTime DESC
LIMIT 1
RETURN doctor.doctor_name AS Doctor
```



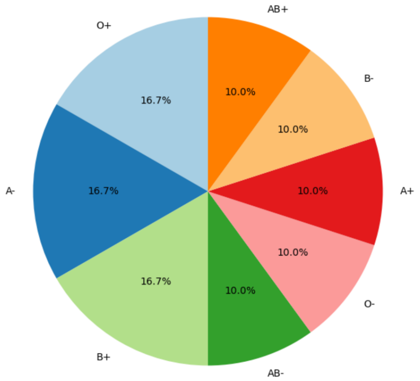
	Doctor
1	"Oliver Smith"

V. Database Access via R or Python

This Python script employs the mysql.connector library to establish a connection with a MySQL database, configuring connection details through the db_config dictionary. The connect_to_mysql() function is responsible for connection establishment, printing a confirmation message upon success. Utilizing the execute_query() function, the script executes various SQL queries to retrieve data from the MySQL database, with pandas utilized to convert the query results into a DataFrame. The obtained data is then leveraged to generate visualizations using the matplotlib.pyplot library.

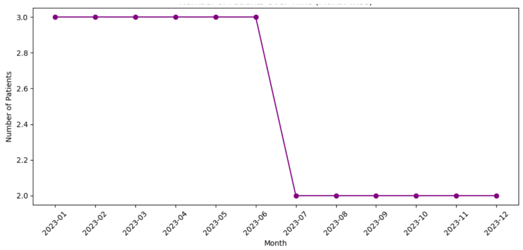
Graph 1: Blood Type Distribution

Pie chart showing the percentage distribution of blood types among patients.



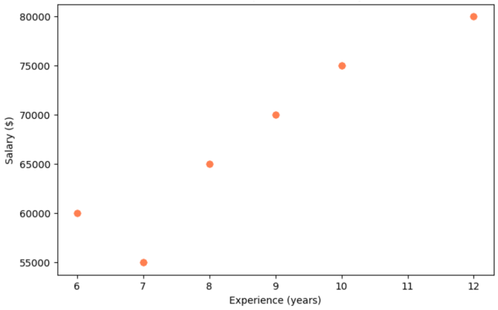
Graph 3: Patients Over Time

Line plot depicting the monthly count of patients, revealing trends over time.



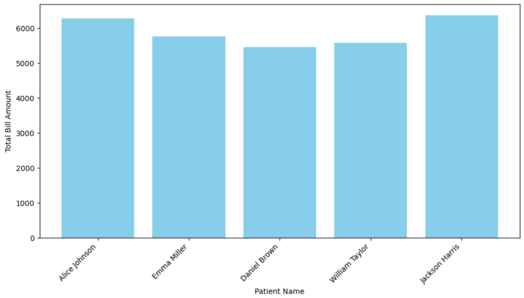
Graph 2: Doctor's Experience vs. Salary

Scatter plot correlating doctors' experience (in years) with their respective salaries.



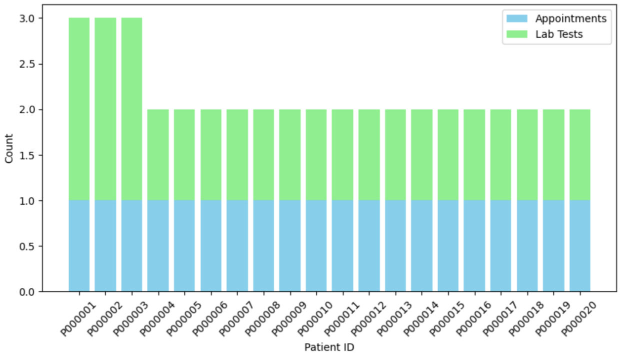
Graph 4: Patients Exceeding Avg Bill

Bar chart highlighting patients with total billing amounts surpassing the average.



Graph 5: Appointments and Lab Tests per Patient

Stacked bar chart illustrating the count of appointments and lab tests for each patient.



VII. Summary and recommendation

In summary, the constructed database system serves to securely store patient history, manage staff information, and streamline hospital operations. The patient check-up process involves medical tests in the laboratory, with comprehensive records stored in the system, covering staff and patient information through conceptual models like EER, UML, and a relational model. This approach allows for future updates, optimizing hospital operations and providing valuable insights into common diseases.

The system's significant advantage lies in enforcing integrity rules, ensuring consistency and accuracy in database storage. Efforts to adhere to rules and constraints enable scalability through advanced techniques like clustering or sharding. Despite challenges such as time constraints and resource shortages, the project addresses key hospital functions, though some roles were involuntarily omitted. While the absence of real data for import isn't a severe issue, future updates could enhance the system's accuracy and reliability for potential analyses.

To enhance the system comprehensively, it is recommended to address the omission of key roles, including pharmacy, insurance personnel, and accountants in the database system. These roles play critical functions in hospital operations, such as medication distribution, managing insurance claims, and financial aspects, contributing to a more holistic representation. Additionally, efforts should be directed towards incorporating overlooked functions supporting main attributes, ensuring a thorough hospital management model that accurately reflects the complex healthcare environment.

While the absence of real data import is not a critical issue, considering efforts to include actual patient and staff information can significantly improve the system's accuracy for potential analyses. Continuous optimization measures are crucial to adapting the system to evolving healthcare requirements, ensuring its relevance and effectiveness over time. Establishing user feedback mechanisms is equally important, providing a valuable avenue to capture insights from hospital staff interactions. This feedback loop facilitates ongoing improvements, aligning the system with practical needs and enhancing user satisfaction in the dynamic healthcare landscape.