Hospital Information Management System

A PROJECT REPORT

Submitted by

MOHAMMED LAMIH [RA2211028010063]

Under the Guidance of

Dr. G GEETHA

Assistant Professor, Networking & Communications

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING W/S IN CLOUD COMPUTING

DEPARTMENT OF NETWORKING & COMMUNICATION

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

RA2211028010063 Certified to be the bonafide work done by **Mohammed Lamih** of II year/IV sem B.Tech Degree Course in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2023-2024.

Date:03/05/2024

Faculty in Charge
Dr. G Geetha
Assistant Professor
Department of Networking and Communications
SRMSIT -KTR

HEAD OF THE DEPARTMENT
Dr. Annapurani. K
Professor
Department of Networking and Communications
SRMSIT -KTR

ABSTRACT

This project focuses on the design and implementation of a comprehensive Hospital Management System (HMS) database using relational database management system (RDBMS) principles. The HMS aims to streamline and optimize various administrative and clinical processes within a hospital environment.

The database schema incorporates entities such as patients, doctors, nurses, administrative staff, departments, medical records, appointments, and inventory management. Relationships between these entities are established to facilitate efficient data retrieval, storage, and manipulation.

Key features of the HMS database include patient registration, appointment scheduling, medical record management, inventory tracking, billing and invoicing, and staff management. The system is designed to support multiple users with varying access levels, ensuring data security and confidentiality.

The project involves the creation of an intuitive user interface for seamless interaction with the database, employing appropriate technologies and frameworks. Additionally, data validation and integrity constraints are implemented to maintain data accuracy and consistency.

Through this project, we aim to address the challenges faced by hospitals in managing their operations effectively, providing a scalable and customizable solution that enhances overall efficiency and patient care.

TABLE OF CONTENTS

1.Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project

INTRODUCTION

In the contemporary healthcare landscape, the efficient management of hospital resources and patient information is critical for ensuring quality care delivery and operational effectiveness. Hospital Management Systems (HMS) have emerged as indispensable tools to streamline administrative tasks, enhance clinical processes, and improve overall patient outcomes.

This project endeavors to design and implement a robust HMS database, leveraging the power of relational database management systems (RDBMS) to address the multifaceted needs of modern healthcare facilities. By developing a comprehensive database schema and accompanying user interface, the aim is to create a dynamic platform capable of managing diverse aspects of hospital operations seamlessly.

The significance of this project lies in its potential to revolutionize the way hospitals manage their day-to-day activities, from patient registration and appointment scheduling to medical record management and inventory tracking. Through careful database design and implementation, we seek to optimize

resource utilization, minimize administrative overhead, and enhance decision-making processes across all levels of hospital administration.

By integrating functionalities such as data security, user authentication, and role-based access control, the HMS database will prioritize patient privacy and confidentiality while empowering healthcare professionals with timely and accurate information. Furthermore, the project aims to foster scalability and adaptability, allowing hospitals to tailor the system to their specific requirements and accommodate future growth and technological advancements.

In essence, this project represents a proactive step towards modernizing hospital management practices, fostering greater efficiency, transparency, and patient-centric care delivery. By harnessing the power of database technology, we endeavor to create a foundation upon which hospitals can build sustainable and resilient healthcare ecosystems for the benefit of all stakeholders.

PROBLEM STATEMENT

Despite advancements in medical technology, many hospitals continue to grapple with inefficiencies in managing their operations and patient information. Manual processes, disparate systems, and paper-based records often result in delays, errors, and increased administrative burden, ultimately compromising the quality of patient care.

The primary problem addressed by this project is the need for a comprehensive Hospital Management System (HMS) database that can centralize and automate various administrative and clinical processes within a hospital environment. Key challenges include:

1. **Fragmented Information**: Hospitals often struggle with fragmented patient information scattered across disparate systems and paper-based records, leading to inefficiencies in data retrieval and decision-making.
2. **Inefficient Resource Allocation**: Manual scheduling of appointments, allocation of resources, and management of inventory can lead to underutilization or overburdening of hospital resources, impacting both operational efficiency and patient satisfaction.
3. **Data Security and Compliance**: Ensuring the security and confidentiality of patient information is paramount, yet many hospitals lack robust mechanisms for data encryption, access control, and compliance with regulatory requirements such as HIPAA.
4. **Limited Scalability and Adaptability**: Existing HMS solutions may lack the scalability and adaptability to accommodate the evolving needs of hospitals, hindering their ability to respond effectively to changes in patient volume, clinical practices, or regulatory mandates.
5. **User Experience and Accessibility**: The usability of HMS interfaces is often overlooked, resulting in poor user experience for hospital staff and

clinicians. An intuitive and user-friendly interface is essential for facilitating widespread adoption and maximizing the system's effectiveness.

Addressing these challenges requires the development of a comprehensive HMS database that integrates seamlessly with existing hospital systems, automates routine tasks, enhances data security, and prioritizes user experience. By doing so, hospitals can streamline their operations, improve clinical outcomes, and ultimately enhance the quality of patient care.

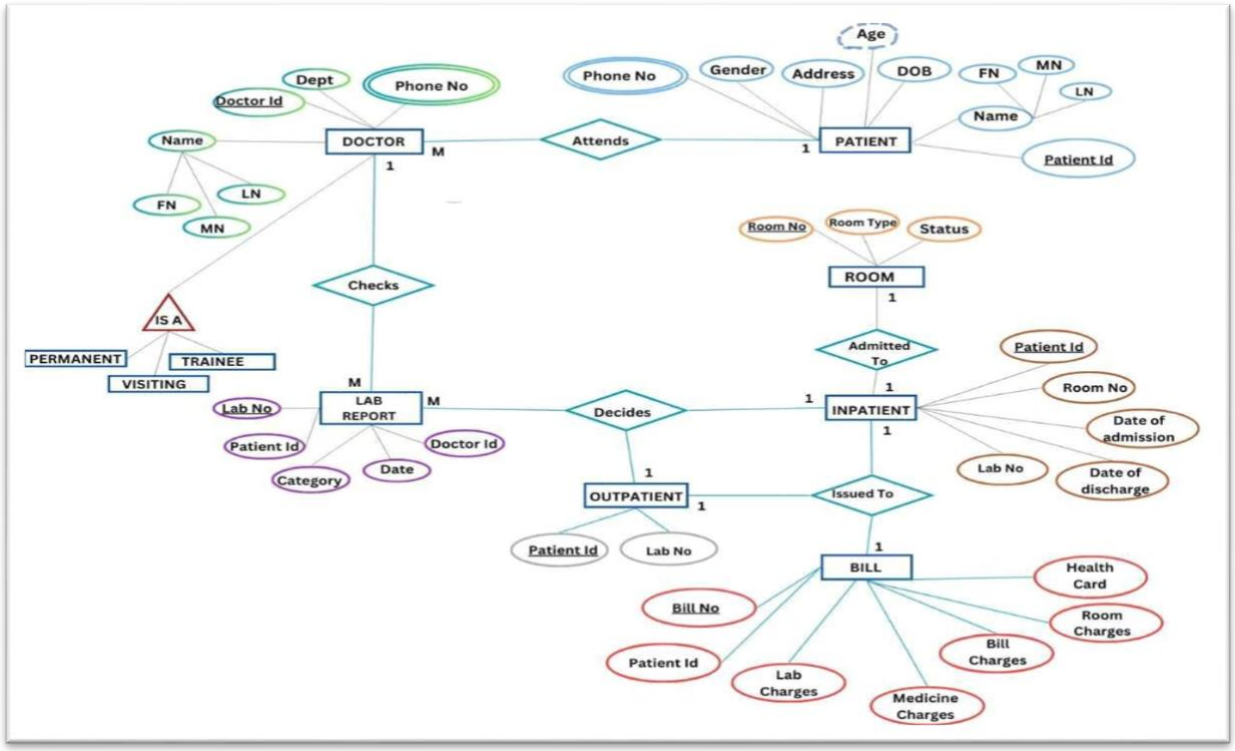
ENTITIES:

- Patient
- Doctor
- Lab Report
- Bill
- Room
- Inpatient
- Outpatient
- Nurse
- User
- Appointment
- Medical Record

RELATIONSHIPS:

1. **Patient - Doctor**:
    - Relationship: Many-to-Many
    - Description: Patients can have appointments with multiple doctors, and doctors can have multiple patients.
2. **Patient - Lab Report**:
    - Relationship: One-to-Many
    - Description: A patient can have multiple lab reports associated with their medical record.
3. **Patient - Bill**:
    - Relationship: One-to-Many
    - Description: A patient can have multiple bills associated with their medical treatment and services received.
4. **Patient - Room**:
    - Relationship: One-to-One
    - Description: During their stay, a patient is assigned to a single room.
5. **Patient - Inpatient**:
    - Relationship: One-to-One
    - Description: A patient can be admitted as an inpatient during their hospital stay.

6. **Patient - Outpatient**:
    - Relationship: One-to-One
    - Description: A patient can be treated as an outpatient for certain medical services without being admitted.
7. **Doctor - Appointment**:
    - Relationship: One-to-Many
    - Description: A doctor can have multiple appointments scheduled with different patients.
8. **Doctor - Medical Record**:
    - Relationship: One-to-Many
    - Description: A doctor creates and maintains medical records for multiple patients they treat.
9. **Nurse - Inpatient**:
    - Relationship: One-to-Many
    - Description: A nurse can be assigned to care for multiple inpatients during their hospital stay.
10. **User - Appointment**:
    - Relationship: One-to-Many
    - Description: Users (hospital staff) can schedule appointments for multiple patients.



ER-DIAGRAM

2.Design of Relational Schemas, Creation of Database Tables for the project.

Code for creation of tables:

```
-- Table: Doctor
CREATE TABLE Doctor (
    DoctorID INT PRIMARY KEY,
    Name VARCHAR(255),
    ContactInfo VARCHAR(255),
```

```
    Specialty VARCHAR(255),
    Qualifications VARCHAR(255),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES
Department(DepartmentID)
);

-- Table: Nurse
CREATE TABLE Nurse (
    NurseID INT PRIMARY KEY,
    Name VARCHAR(255),
    ContactInfo VARCHAR(255),
    Qualifications VARCHAR(255),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES
Department(DepartmentID)
);

-- Table: Patient
CREATE TABLE Patient (
    PatientID INT PRIMARY KEY,
    Name VARCHAR(255),
    ContactInfo VARCHAR(255),
    DateOfBirth DATE,
    Gender VARCHAR(10),
    MedicalHistory TEXT
```

```
);

-- Table: Appointment
CREATE TABLE Appointment (
    AppointmentID INT PRIMARY KEY,
    PatientID INT,
    DoctorID INT,
    Date DATE,
    Time TIME,
    Reason VARCHAR(255),
    Status VARCHAR(50),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)
);

-- Table: MedicalRecord
CREATE TABLE MedicalRecord (
    RecordID INT PRIMARY KEY,
    PatientID INT,
    DoctorID INT,
    DateOfVisit DATE,
    Diagnosis TEXT,
    Medications TEXT,
    LabTestResults TEXT,
    TreatmentPlan TEXT,
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
```

```
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)
);


-- Table: Bill
CREATE TABLE Bill (
    BillingID INT PRIMARY KEY,
    PatientID INT,
    ServicesProvided TEXT,
    Charges DECIMAL(10, 2),
    PaymentStatus VARCHAR(50),
    PaymentHistory TEXT,
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)
);

-- Table: LabReport
CREATE TABLE LabReport (
    ReportID INT PRIMARY KEY,
    PatientID INT,
    TestName VARCHAR(255),
    TestResult TEXT,
    TestDate DATE,
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)
);

-- Table: Room
```

```
CREATE TABLE Room (
    RoomID INT PRIMARY KEY,
    RoomNumber VARCHAR(50),
    RoomType VARCHAR(50),
    Status VARCHAR(50)
);

-- Table: Inpatient
CREATE TABLE Inpatient (
    InpatientID INT PRIMARY KEY,
    PatientID INT,
    RoomID INT,
    AdmissionDate DATE,
    DischargeDate DATE,
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
    FOREIGN KEY (RoomID) REFERENCES Room(RoomID)
);

-- Table: Outpatient
CREATE TABLE Outpatient (
    OutpatientID INT PRIMARY KEY,
    PatientID INT,
    AppointmentID INT,
    VisitDate DATE,
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
```

```
    FOREIGN KEY (AppointmentID) REFERENCES
Appointment(AppointmentID)
);

-- Table: User
CREATE TABLE User (
    UserID INT PRIMARY KEY,
    Username VARCHAR(255),
    Password VARCHAR(255),
    Role VARCHAR(50)
);
```

DML commands (INSERT & Result)

```
-- Patient
INSERT INTO Patient (PatientID, Name, ContactInfo, DateOfBirth, Gender,
MedicalHistory)
VALUES
   (1, 'John Smith', 'john@example.com', TO_DATE('1990-05-15', 'YYYY-
MM-DD'), 'Male', 'None'),
   (2, 'Jane Doe', 'jane@example.com', TO_DATE('1985-09-20', 'YYYY-MM-
DD'), 'Female', 'Hypertension'),
   (3, 'Michael Johnson', 'michael@example.com', TO_DATE('1978-12-10',
```

```
'YYYY-MM-DD'), 'Male', 'Diabetes');
-- Doctor
INSERT INTO Doctor (DoctorID, Name, ContactInfo, Specialty,
Qualifications, DepartmentID)
VALUES
    (1, 'Dr. John Smith', 'john@example.com', 'Cardiology', 'MD, Cardiologist',
1),
    (2, 'Dr. Emily Brown', 'emily@example.com', 'Pediatrics', 'MD, Pediatrician',
2),
    (3, 'Dr. David Lee', 'david@example.com', 'Orthopedics', 'MD, Orthopedic
Surgeon', 3);
-- Lab Report

INSERT INTO LabReport (ReportID, PatientID, TestName, TestResult,
TestDate)
VALUES
    (1, 1, 'Blood Test', 'Normal', TO_DATE('2024-05-01', 'YYYY-MM-DD')),
    (2, 2, 'X-Ray', 'No fractures detected', TO_DATE('2024-04-28', 'YYYY-MM-
DD')),
    (3, 3, 'Urinalysis', 'Abnormal', TO_DATE('2024-05-02', 'YYYY-MM-DD'));
-- Bill
INSERT INTO Bill (BillingID, PatientID, ServicesProvided, Charges,
```

```
PaymentStatus, PaymentHistory)
VALUES
    (1, 1, 'Consultation, Blood Test', 150.00, 'Unpaid', 'None'),
    (2, 2, 'Consultation, X-Ray', 200.00, 'Paid', '2024-04-30: Paid in full'),
    (3, 3, 'Consultation, Urinalysis', 120.00, 'Partially Paid', '2024-05-05: Partial payment received');
-- Room
INSERT INTO Room (RoomID, RoomNumber, RoomType, Status)
VALUES
    (1, '101', 'Single', 'Occupied'),
    (2, '202', 'Double', 'Available'),
    (3, '303', 'Single', 'Occupied');
-- Inpatient
INSERT INTO Inpatient (InpatientID, PatientID, RoomID, AdmissionDate, DischargeDate)
VALUES
    (1, 1, 1, TO_DATE('2024-05-01', 'YYYY-MM-DD'), TO_DATE('2024-05-05', 'YYYY-MM-DD')),
    (2, 2, 2, TO_DATE('2024-04-28', 'YYYY-MM-DD'), TO_DATE('2024-05-02', 'YYYY-MM-DD')),
    (3, 3, 3, TO_DATE('2024-05-02', 'YYYY-MM-DD'), TO_DATE('2024-05-07', 'YYYY-MM-DD'));
```

```
-- Outpatient
INSERT INTO Outpatient (OutpatientID, PatientID, AppointmentID,
VisitDate)
VALUES
    (1, 1, 1, TO_DATE('2024-05-10', 'YYYY-MM-DD')),
    (2, 2, 2, TO_DATE('2024-05-08', 'YYYY-MM-DD')),
    (3, 3, 3, TO_DATE('2024-05-12', 'YYYY-MM-DD'));
-- Nurse
INSERT INTO Nurse (NurseID, Name, ContactInfo, Qualifications,
DepartmentID)
VALUES
    (1, 'Sarah Johnson', 'sarah@example.com', 'RN, BSN', 1),
    (2, 'Michael Brown', 'michael@example.com', 'RN, BSN', 2),
    (3, 'Emily Smith', 'emily@example.com', 'RN, BSN', 3);
-- User
INSERT INTO User (UserID, Username, Password, Role)
VALUES
    (1, 'admin', 'admin123', 'Administrator'),
    (2, 'nurse1', 'nursepass1', 'Nurse'),
    (3, 'doctor1', 'doctorpass1', 'Doctor');
-- Appointment
INSERT INTO Appointment (AppointmentID, PatientID, DoctorID, Date,
```

```
Time, Reason, Status)
VALUES
    (1, 1, 1, TO_DATE('2024-05-10', 'YYYY-MM-DD'),
TO_TIMESTAMP('08:00:00', 'HH24:MI:SS'), 'Routine check-up', 'Scheduled'),
    (2, 2, 2, TO_DATE('2024-05-12', 'YYYY-MM-DD'),
TO_TIMESTAMP('10:00:00', 'HH24:MI:SS'), 'Pediatric consultation',
'Scheduled'),
    (3, 3, 3, TO_DATE('2024-05-15', 'YYYY-MM-DD'),
TO_TIMESTAMP('09:30:00', 'HH24:MI:SS'), 'Orthopedic follow-up',
'Scheduled');
-- MedicalRecord
INSERT INTO MedicalRecord (RecordID, PatientID, DoctorID, DateOfVisit,
Diagnosis, Medications, LabTestResults, TreatmentPlan)
VALUES
    (1, 1, 1, TO_DATE('2024-05-10', 'YYYY-MM-DD'), 'Hypertension',
'Medication A, Medication B', 'Normal', 'Follow-up in 2 weeks.'),
    (2, 2, 2, TO_DATE('2024-05-12', 'YYYY-MM-DD'), 'Upper respiratory
infection', 'Antibiotic C', 'No abnormalities', 'Complete bed rest for 3 days.'),
    (3, 3, 3, TO_DATE('2024-05-15', 'YYYY-MM-DD'), 'Fractured ankle', 'Pain
medication, Anti-inflammatory D', 'Fracture confirmed, X-ray attached',
'Orthopedic consultation scheduled.')
```

```
SQL> set linesize 160;
SQL> set wrap off;
SQL> select * from doctor;
rows will be truncated

rows will be truncated

rows will be truncated


  DOCTORID NAME
---------- ------------------------------------------------------------
----
         1 Dr. John Smith
         2 Dr. Emily Brown
         3 Dr. David Lee
```

```
SQL> select * from user1;
rows will be truncated

rows will be truncated


    USERID USERNAME
---------- ------------------------------------------------------------
----
         1 admin
         2 nurse1
         3 doctor1
```

```
SQL> select * from labrecord;
select * from labrecord
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> select * from LabReport;
rows will be truncated

rows will be truncated


  REPORTID  PATIENTID TESTNAME
---------- ---------- ------------------------------------------------------------
----
         1          1 Blood Test
         2          2 X-Ray
         3          3 Urinalysis

SQL> select * from room;

    ROOMID ROOMNUMBER                                          ROOMTYPE                                           STATUS
---------- ------------------------------------------------- -------------------------------------------------- --------------------
----
         1 101                                                Single                                             Occupied
         2 202                                                Double                                             Available
         3 303                                                Single                                             Occupied

SQL> select * from inpatient;

INPATIENTID  PATIENTID     ROOMID ADMISSION DISCHARGE
----------- ---------- ---------- --------- ---------
          1          1          1 01-MAY-24 05-MAY-24
          2          2          2 28-APR-24 02-MAY-24
          3          3          3 02-MAY-24 07-MAY-24
```

```
SQL> select * from medicalrecord;
rows will be truncated

  RECORDID  PATIENTID   DOCTORID DATEOFVIS DIAGNOSIS                                        MEDICATIONS                                        LABTESTRESU
LTS
---------- ---------- ---------- --------- -------------------------------------------- -------------------------------------------------- ----------
----
         1          1          1 10-MAY-24 Hypertension                                 Medication A, Medication B                         Normal
         2          2          2 12-MAY-24 Upper respiratory infection                  Antibiotic C                                       No abnormal
itie
         3          3          3 15-MAY-24 Fractured ankle                              Pain medication, Anti-inflammatory D               Fracture co
nfir

SQL> select * from bill;

 BILLINGID  PATIENTID SERVICESPROVIDED                                CHARGES PAYMENTSTATUS                                      PAYMENTHISTORY
---------- ---------- ---------------------------------------------- ---------- -------------------------------------------------- --------------------
----
         1          1 Consultation, Blood Test                              150 Unpaid                                             None
         2          2 Consultation, X-Ray                                   200 Paid                                               2024-04-30: Paid in f
ull
         3          3 Consultation, Urinalysis                              120 Partially Paid                                     2024-05-05: Partial p
ayme
```

```
SQL> select * from nurse;
rows will be truncated

rows will be truncated


   NURSEID NAME
---------- --------------------------------------------------------------------------------
----
         1 Sarah Johnson
         2 Michael Brown
         3 Emily Smith

SQL> select * from appointment;
rows will be truncated


APPOINTMENTID  PATIENTID   DOCTORID REASON
------------- ---------- ---------- --------------------------------------------------
----
            1          1          1 Routine check-up
            2          2          2 Pediatric consultation
            3          3          3 Orthopedic follow-up
```

```
SQL> select * from patient;
rows will be truncated

rows will be truncated

rows will be truncated

rows will be truncated


 PATIENTID NAME
---------- --------------------------------------------------------------------------------
----
         1 John Smith
         2 Jane Doe
         3 Michael Johnson
```

Alter

ALTER TABLE Doctor

ADD (Email VARCHAR2(255));

Schema:

Patients: (PatientID (Primary Key), Name, Gender, DateOfBirth, Address, Phone, Email)

Doctor: (DoctorID (Primary Key), Name, Gender, DateOfBirth, Address, Phone, Email, Specialty, Qualifications)

LabReport: (ReportID (Primary Key), PatientID (Foreign Key), TestName, TestResult, TestDate)

Bill: (BillID (Primary Key), PatientID (Foreign Key), ServicesProvided, Charges, PaymentStatus, PaymentHistory)

Room: (RoomID (Primary Key), RoomNumber, RoomType, Status)

Inpatient: (InpatientID (Primary Key), PatientID (Foreign Key), RoomID (Foreign Key), AdmissionDate, DischargeDate)

Outpatient: (OutpatientID (Primary Key), PatientID (Foreign Key), AppointmentID (Foreign Key), VisitDate)

Nurse: (NurseID (Primary Key), Name, Gender, DateOfBirth, Address, Phone, Email, Qualifications)

User: (UserID (Primary Key), Username, Password, Role)

Appointment: (AppointmentID (Primary Key), PatientID (Foreign Key), DoctorID (Foreign Key), Date, Time, Reason, Status)

MedicalRecord: (RecordID (Primary Key), PatientID (Foreign Key), DoctorID (Foreign Key), DateOfVisit, Diagnosis, Medications, LabTestResults, TreatmentPlan)

Patient

Column Name	Data Type	Constraints
Patient Id	INT	PRIMARY KEY
Name	VARCHAR(100)	NOT NULL
DOB	DATE	NOT NULL
Age	INT	NOT NULL
Address	VARCHAR(250)	NOT NULL
Gender	VARCHAR(15)	NOT NULL
Phone No	INT	NOT NULL

Doctor

Column Name	Data Type	Constraints
Doctor Id	INT	PRIMARY KEY
Name	VARCHAR(100)	NOT NULL
Dept	VARCHAR(25)	NOT NULL
Phone No	INT	NOT NULL

Lab Report

Column Name	Data Type	Constraints
Lab No	INT	PRIMARY KEY
Patient Id	INT	NOT NULL
Category	VARCHAR(50)	NOT NULL
Date	DATE	NOT NULL
Doctor Id	INT	NOT NULL

Inpatient

Column Name	Data Type	Constraints
Patient Id	INT	PRIMARY KEY
Room No	INT	NOT NULL
Date of Admission	DATE	NOT NULL
Date of Discharge	DATE	NOT NULL
Lab No	INT	

Outpatient

Column Name	Data Type	Constraints
Patient Id	INT	PRIMARY KEY
Lab No	INT	

Room

Column Name	Data Type	Constraints
Room No	INT	PRIMARY KEY
Room Type	VARCHAR(50)	NOT NULL
Status	VARCHAR(25)	NOT NULL

Bill

Column Name	Data Type	Constraints
Bill No	INT	PRIMARY KEY
Patient Id	INT	NOT NULL
Lab Charges	FLOAT	
Medicine Charges	FLOAT	
Bill Charges	FLOAT	NOT NULL
Room Charges	FLOAT	
Health Card	VARCHAR(250)	

3.Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors.

Constraints

Constraints are rules or conditions applied to a database table to ensure the accuracy, integrity, and consistency of the data. They define the limitations and requirements that data must adhere to when it is inserted, updated, or deleted from the database. Here are some common types of constraints in DBMS:

Primary Key Constraint: This constraint ensures that each row in a table is uniquely identifiable. It prevents duplicate or null values in the column(s) designated as the primary key.

Foreign Key Constraint: A foreign key constraint establishes a relationship between two tables. It ensures that values in a specific column (foreign key) in one table correspond to values in another table's primary key column. This maintains referential integrity between related tables.

Unique Constraint: Similar to a primary key, a unique constraint ensures that each value in a specified column or group of columns is unique. However, unlike a primary key, it allows null values (except in columns marked as NOT NULL).

Check Constraint: Check constraints enforce specific conditions on data values in a column. For instance, a check constraint can ensure that a numeric value falls within a certain range or that a text value meets a particular format.

Not Null Constraint: This constraint ensures that a column does not accept null values. It requires every row to have a value in that column.

```
ALTER TABLE MedicalRecord
ADD CONSTRAINT check_date_of_visit CHECK (DateOfVisit <=
SYSDATE);
```

Sets

Database management systems (DBMS), "sets" typically refer to the mathematical concept of sets applied to data retrieval and manipulation. Here

are a few key aspects of sets in DBMS:

Set Theory: Sets in DBMS are based on set theory from mathematics. A set is a collection of distinct objects, known as elements, that share a common property. In DBMS, sets are often used to represent data and perform operations like union, intersection, difference, etc., just like in mathematics.

Set Operations: DBMS uses set operations to manipulate data sets. The primary set operations include:

- Union: Combines all unique elements from two or more sets.
- Intersection: Retrieves elements common to two or more sets.
- Difference: Retrieves elements present in one set but not in another.
- Cartesian Product: Combines every element from one set with every element from another set, resulting in a new set of pairs.
- Join: A relational database operation that combines rows from two or more tables based on a related column between them.
- SQL and Set Operations: Structured Query Language (SQL) supports set operations, allowing users to perform set-based queries on data stored in relational databases. For example:
- UNION: Combines the results of two or more SELECT statements, removing duplicates.
- INTERSECT: Retrieves rows common to two SELECT statements.
- EXCEPT or MINUS: Retrieves rows from the first SELECT statement

that are not present in the second SELECT statement.

- Benefits of Set Operations: Using set operations in DBMS offers several benefits, including:
- Simplifying complex queries by breaking them down into smaller, more manageable parts.
- Improving query performance, as set operations are often optimized by the database engine.
- Facilitating data manipulation and analysis tasks, such as data aggregation, filtering, and comparison.

```
-- Retrieve the names of patients and doctors from their respective tables.
SELECT Name, 'Patient' AS Role
FROM Patients
UNION
SELECT Name, 'Doctor' AS Role
FROM Doctor;
```

```
SQL> SELECT Name, 'Patient' AS Role
  2  FROM Patient
  3  UNION
  4  SELECT Name, 'Doctor' AS Role
  5  FROM Doctor;
rows will be truncated


NAME
--------------------------------------------------------------------------------
----
Dr. David Lee
Dr. Emily Brown
Dr. John Smith
Jane Doe
John Smith
Michael Johnson

6 rows selected.
```

Joins

Joins are operations in relational databases that combine rows from two or more tables based on a related column between them. This related column is typically a primary key in one table that matches a foreign key in another table, establishing a relationship between the tables. Joins allow users to retrieve related data from multiple tables in a single result set, enabling more comprehensive data analysis and reporting. Types of joins include:

a) Inner Join: Retrieves rows from both tables where there is a match based on the join condition.

b) Left (Outer) Join: Retrieves all rows from the left table and matching rows from the right table. If no match is found, NULL values are

included for the right table columns.

c) Right (Outer) Join: Similar to the left join but retrieves all rows from the right table and matching rows from the left table.
d) Full (Outer) Join: Retrieves all rows from both tables, including rows without matches. If no match is found, NULL values are included for the unmatched columns.
e) Cross Join: Produces the Cartesian product of two tables, combining every row from the first table with every row from the second table.

Joins are fundamental for relational databases as they allow for data normalization (breaking data into separate tables for efficiency and consistency) while still enabling efficient data retrieval by reassembling related data as needed.

```
-- Retrieve the names of patients along with their corresponding doctors
SELECT p.Name AS PatientName, d.Name AS DoctorName
FROM Patients p
INNER JOIN MedicalRecord mr ON p.PatientID = mr.PatientID
INNER JOIN Doctor d ON mr.DoctorID = d.DoctorID;
```

```
SQL> SELECT p.Name AS PatientName, d.Name AS DoctorName
  2  FROM Patient p
  3  INNER JOIN MedicalRecord mr ON p.PatientID = mr.PatientID
  4  INNER JOIN Doctor d ON mr.DoctorID = d.DoctorID;

PATIENTNAME
--------------------------------------------------------------------------------
DOCTORNAME
--------------------------------------------------------------------------------
John Smith
Dr. John Smith

Jane Doe
Dr. Emily Brown

Michael Johnson
Dr. David Lee
```

- We use the INNER JOIN keyword to combine the Patients and MedicalRecord tables based on the PatientID column, and then combine the result with the Doctor table based on the DoctorID column.
- The ON keyword specifies the condition for joining the tables. In this case, we join Patients and MedicalRecord on their PatientID columns, and then join the result with the Doctor table on the DoctorID columns.
- We alias the tables as p for Patients, mr for MedicalRecord, and d for Doctor to make the query more readable.
- We select the Name column from both the Patients and Doctor tables, and rename them as PatientName and DoctorName, respectively, to avoid ambiguity.

The result of this query will include the names of patients along with the names of their corresponding doctors.

Views

Views in relational databases are virtual tables that are derived from one or more tables (or other views) based on a predefined SQL query. Unlike physical tables, views do not store data themselves; instead, they represent a logical layer that presents data in a particular format or subset based on the underlying tables. Types of views include:

- Simple View: Based on a single table or a straightforward SQL query.
- Complex View: Involves multiple tables or includes more complex SQL logic.
- Materialized View: A view that stores the result set of the query physically, updating it periodically or based on certain conditions. It improves query performance but requires more storage and maintenance.

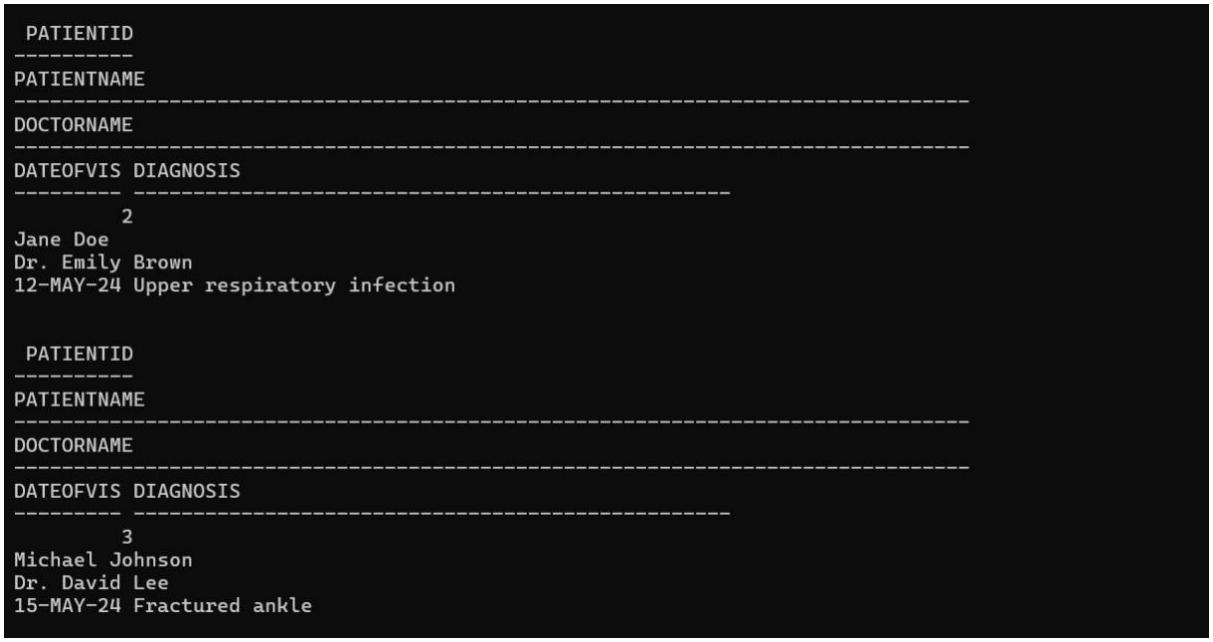
Views provide several benefits, including:

- Simplifying complex queries by abstracting the underlying structure and presenting data in a more understandable format.
- Enhancing security by limiting access to specific columns or rows, masking sensitive data, or enforcing business rules through the view definition.
- Improving performance by precomputing and storing frequently used query results in materialized views, reducing the need for repetitive computations.

-- Create a view that shows the details of patients along with their corresponding doctors and medical records

```
CREATE VIEW PatientDoctorView AS
SELECT p.PatientID, p.Name AS PatientName, d.Name AS DoctorName, m.DateOfVisit, m.Diagnosis
FROM Patients p
INNER JOIN MedicalRecord m ON p.PatientID = m.PatientID
INNER JOIN Doctor d ON m.DoctorID = d.DoctorID;
```

```
SQL> CREATE VIEW PatientDoctorView AS
  2  SELECT p.PatientID, p.Name AS PatientName, d.Name AS DoctorName, m.DateOfVisit, m.Diagnosis
  3  FROM Patient p
  4  INNER JOIN MedicalRecord m ON p.PatientID = m.PatientID
  5  INNER JOIN Doctor d ON m.DoctorID = d.DoctorID;

View created.

SQL> select * from PatientDoctorView;

 PATIENTID
----------
PATIENTNAME
--------------------------------------------------------------------------------
DOCTORNAME
--------------------------------------------------------------------------------
DATEOFVIS DIAGNOSIS
--------- --------------------------------------------------
         1
John Smith
Dr. John Smith
10-MAY-24 Hypertension
```



- We create a view named PatientDoctorView.
- The view selects the PatientID, Name from Patients table aliased as PatientName, Name from Doctor table aliased as DoctorName, DateOfVisit, and Diagnosis from MedicalRecord table.
- We use INNER JOIN to join Patients with MedicalRecord based on PatientID, and then join the result with Doctor based on DoctorID.
- The view will allow us to query patient details along with their corresponding doctor's name, date of visit, and diagnosis without having to write complex joins every time.

Triggers:

Databases refers to a set of actions that are automatically performed or executed in response to certain events or conditions. These events could be data

modifications (such as insertion, update, or deletion) or schema changes within the database. Triggers are used to enforce business rules, maintain data integrity, and automate tasks within the database management system (DBMS).

Here are some key points about triggers:

Types of Triggers:

- Row-Level Triggers: These triggers are executed once for each row affected by the triggering event. For example, an INSERT trigger might execute once for each row being inserted into a table.
- Statement-Level Triggers: These triggers are executed once for each SQL statement, regardless of how many rows are affected by that statement. They are commonly used for schema-level events like CREATE, ALTER, or DROP operations.

Trigger Events:

- INSERT: Triggered when new data is inserted into a table.
- UPDATE: Triggered when existing data is updated in a table.
- DELETE: Triggered when data is deleted from a table.
- CREATE, ALTER, DROP: Triggered when schema-level changes occur, such as creating a new table, altering the structure of an existing table, or dropping a table.
- Trigger Actions:
- BEFORE Trigger: Executes before the triggering event (e.g., before an

INSERT, UPDATE, or DELETE operation). It can be used to modify incoming data or perform validations before the actual operation.

- AFTER Trigger: Executes after the triggering event has occurred. It can be used to perform actions such as logging changes, updating related data, or enforcing referential integrity.

Uses of Triggers:

- Enforcing Data Integrity: Triggers can enforce complex business rules or constraints that cannot be easily expressed using standard constraints (e.g., ensuring certain conditions are met before allowing an update).
- Auditing and Logging: Triggers can be used to log changes made to data, track user activity, or maintain an audit trail for compliance purposes.
- Automating Tasks: Triggers can automate tasks such as sending notifications, updating denormalized data, or invoking external procedures based on specific events.
- Implementing Business Logic: Triggers can encapsulate business logic within the database, reducing the need for such logic to be implemented in application code.

-- Create a trigger that automatically updates the LastUpdated column in the Patients table whenever a new record is inserted or an existing record is updated

```
CREATE OR REPLACE TRIGGER UpdateLastUpdated
BEFORE INSERT OR UPDATE ON Patients
```

```
FOR EACH ROW
BEGIN
    :NEW.LastUpdated := SYSDATE;
END;
```

Cursor:

A cursor is a database object used to retrieve, navigate, and manipulate data row by row, particularly within procedural programming constructs like stored procedures, functions, or triggers. Cursors provide a way to work with result sets that are returned by a SQL query, allowing developers to process data in a more controlled and iterative manner.

4.Analyzing the pitfalls, identifying the dependencies, and applying normalizations

A functional dependency (FD) is a fundamental concept in database management systems (DBMS) that describes the relationship between attributes (columns) in a relation (table). It specifies how the values of certain attributes uniquely determine the values of other attributes within the same relation. In other words, if certain attributes are functionally dependent on another attribute or set of attributes, knowing the values of the determining attributes allows you to predict the values of the dependent attributes.

1. **Patient Table:**
    - Functional Dependencies: PatientID -> {Name, Gender, DateOfBirth, Address, Phone, Email}
    - Candidate Key: PatientID
    - Super Key: PatientID
2. **Doctor Table:**
    - Functional Dependencies: DoctorID -> {Name, Gender, DateOfBirth, Address, Phone, Email, Specialty, Qualifications}
    - Candidate Key: DoctorID
    - Super Key: DoctorID
3. **Lab Report Table:**
    - Functional Dependencies: ReportID -> {PatientID, TestName, TestResult, TestDate}
    - Candidate Key: ReportID
    - Super Key: ReportID
4. **Bill Table:**
    - Functional Dependencies: BillID -> {PatientID, ServicesProvided, Charges, PaymentStatus, PaymentHistory}
    - Candidate Key: BillID
    - Super Key: BillID

5. **Room Table:**
    - Functional Dependencies: RoomID -> {RoomNumber, RoomType, Status}
    - Candidate Key: RoomID
    - Super Key: RoomID
6. **Inpatient Table:**
    - Functional Dependencies: InpatientID -> {PatientID, RoomID, AdmissionDate, DischargeDate}
    - Candidate Key: InpatientID
    - Super Key: InpatientID
7. **Outpatient Table:**
    - Functional Dependencies: OutpatientID -> {PatientID, AppointmentID, VisitDate}
    - Candidate Key: OutpatientID
    - Super Key: OutpatientID
8. **Nurse Table:**
    - Functional Dependencies: NurseID -> {Name, Gender, DateOfBirth, Address, Phone, Email, Qualifications}
    - Candidate Key: NurseID
    - Super Key: NurseID
9. **User Table:**
    - Functional Dependencies: UserID -> {Username, Password, Role}

- Candidate Key: UserID
- Super Key: UserID

10. **Appointment Table:**
    - Functional Dependencies: AppointmentID -> {PatientID, DoctorID, Date, Time, Reason, Status}
    - Candidate Key: AppointmentID
    - Super Key: AppointmentID
11. **Medical Record Table:**
    - Functional Dependencies: RecordID -> {PatientID, DoctorID, DateOfVisit, Diagnosis, Medications, LabTestResults, TreatmentPlan}
    - Candidate Key: RecordID
    - Super Key: RecordID

NORMALISATION & TYPES OF NORMALISATION

Normalization refers to the process of organizing data in a database efficiently. The goal is to reduce data redundancy and dependency, ensuring that the database is structured in a way that minimizes inconsistencies and anomalies. Normalization is the process of organizing data in a systematic and efficient manner. The goal of normalization is to minimize data redundancy and

dependency, ensuring that the database structure is optimized for storage, retrieval, and maintenance. Normalization involves breaking down a large table into smaller, related tables and establishing relationships between them. This helps in reducing data duplication, which can lead to inconsistencies and anomalies. By following specific rules and guidelines, normalization helps in creating a more structured and reliable database design.

Types of Normalization:

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF)
4. Boyce-Codd Normal Form (BCNF)
5. Fourth Normal Form (4NF)

Normalization of MedicalRecord table :-

Here's the structure of the MedicalRecord table:

MedicalRecord:

- RecordID (Primary Key)
- PatientID (Foreign Key)
- DoctorID (Foreign Key)
- DateOfVisit

- Diagnosis
- Medications
- LabTestResults
- TreatmentPlan

1NF (First Normal Form):

To convert the MedicalRecord table to 1NF, we need to ensure that each column contains atomic values, and there are no repeating groups or arrays.

MedicalRecord_1NF:

- RecordID (Primary Key)
- PatientID (Foreign Key)
- DoctorID (Foreign Key)
- DateOfVisit
- Diagnosis
- Medication_1
- Medication_2
- ...
- LabTestResult_1
- LabTestResult_2
- ...
- TreatmentPlan

2NF (Second Normal Form):

In 2NF, we need to ensure that the table is in 1NF and that all non-key attributes are fully functionally dependent on the entire primary key.

MedicalRecord_2NF:

- RecordID (Primary Key)
- DateOfVisit
- Diagnosis
- TreatmentPlan
- Medication (Composite Key: RecordID, MedicationNumber)
- LabTestResult (Composite Key: RecordID, LabTestNumber)

3NF (Third Normal Form):

In 3NF, we need to ensure that there are no transitive dependencies.

MedicalRecord_3NF:

- RecordID (Primary Key)
- DateOfVisit
- Diagnosis
- TreatmentPlan
- DoctorID (Foreign Key)
- PatientID (Foreign Key)

Medication:

- RecordID (Foreign Key)

- MedicationNumber (Primary Key)

- Medication

LabTestResult:

- RecordID (Foreign Key)

- LabTestNumber (Primary Key)

- LabTestResult

BCNF (Boyce-Codd Normal Form):

In BCNF, we need to ensure that every determinant is a candidate key.

MedicalRecord_BCNF:

- RecordID (Primary Key)

- DateOfVisit

- Diagnosis

- TreatmentPlan

Medication:

- RecordID (Foreign Key)

- MedicationNumber (Primary Key)

- Medication

LabTestResult:

- RecordID (Foreign Key)
- LabTestNumber (Primary Key)
- LabTestResult

Doctor:

- DoctorID (Primary Key)
- Other Doctor attributes

Patient:

- PatientID (Primary Key)
- Other Patient attributes

4NF (Fourth Normal Form):

In 4NF, we ensure that the table does not have any multi-valued dependencies.

No changes needed for MedicalRecord_4NF as it already meets 4NF requirements.

5NF (Fifth Normal Form):

In 5NF, we ensure that the table does not have any join dependencies.

No changes needed for MedicalRecord_5NF as it already meets 5NF

requirements.

This normalization process ensures that the MedicalRecord table is free from anomalies and is structured in such a way that minimizes redundancy and dependency issues.

5.Implementation of concurrency control and recovery mechanisms

Concurrency control is a pivotal aspect of database management, vital for upholding data integrity and consistency in multi-user environments. It encompasses a range of mechanisms and techniques aimed at managing simultaneous access to data by multiple transactions, thereby averting scenarios where transactions may inadvertently disrupt each other, leading to data corruption or inconsistencies.

One widely adopted method in concurrency control is locking. Here, transactions acquire locks on data items to forestall concurrent modifications by other transactions. Locks can be of various types such as read locks and write locks, each regulating the level of access permitted to a data item. Locking

mechanisms play a pivotal role in maintaining data integrity by ensuring that transactions interact with data in a controlled and synchronized manner.
Another prevalent approach is timestamping, wherein transactions are assigned unique timestamps based on their initiation times. These timestamps are then leveraged to order transactions, with precedence typically accorded to transactions with earlier timestamps in cases of conflicts. Timestamping facilitates the execution of transactions in a serializable order, mitigating conflicts and upholding consistency. Contrasting with locking and timestamping, optimistic concurrency control operates on the assumption that conflicts are infrequent. Transactions proceed without acquiring locks, with conflict checks deferred until the transaction's completion. In case of a conflict, the transaction is rolled back and re-executed. While less restrictive than locking, this approach necessitates meticulous handling of conflicts to avert unnecessary rollbacks.
Concurrency control is an intricate domain within database management, with the suitability of various techniques contingent upon the specific application requirements. Its overarching objective is to enable transactions to execute concurrently without compromising data integrity, consistency, or system performance.

The realm of database management encompasses four pivotal aspects:
Atomicity: This principle ensures that transactions adhere to an "all or nothing"

paradigm. Transactions are either completed in full, ensuring the success of all operations within the transaction, or none of them are executed, thus maintaining data integrity.

Consistency: This aspect guarantees that the database remains in a consistent state before and after the execution of a transaction. Transactions must not compromise the overall consistency of the database, ensuring that all constraints and rules are upheld.

Isolation: Isolation mandates that concurrent transactions do not interfere with each other, preserving data integrity and circumventing anomalies. Each transaction must operate independently of others, ensuring that the results of concurrent transactions remain predictable and accurate.

Durability: Durability guarantees that once a transaction is committed, its effects are permanently saved and persist even in the face of system failures or crashes. Data changes made by committed transactions must endure, ensuring long-term data integrity and reliability.

Recovery mechanisms are pivotal components of database management systems (DBMS), crucial for ensuring data durability and system reliability, particularly in the event of failures. These mechanisms play a critical role in restoring the database to a consistent state post-failure while minimizing disruptions to

ongoing transactions.

Undo Recovery (Rollback): This mechanism involves reversing the effects of transactions that were incomplete at the time of failure. It ensures that changes made by these transactions are not permanently applied to the database, maintaining data integrity.

Redo Recovery: In contrast, redo recovery entails reapplying the effects of transactions that were completed but not yet permanently stored in the database at the time of failure. It prevents the loss of changes made by these transactions, thereby preserving data consistency.

DBMSs employ techniques such as checkpointing and logging to enhance data consistency and durability. Checkpointing involves periodically saving the current database state to disk along with information about committed transactions not yet written to disk. This facilitates recovery to a consistent state post-failure by restoring the last checkpoint and replaying the logs.

Concurrency control in databases is all about making sure that when multiple people are trying to change the same data at the same time, nothing goes wrong. Imagine you and a friend both want to update your contact information on a shared online platform. Concurrency control ensures that your updates don't accidentally mess up each other's changes.

One way to do this is by using locks. It's like when you lock a door to make sure no one else can enter while you're inside. In databases, locks prevent others from

changing data while you're already working on it. There are different types of locks, like read locks (which let you view data without changing it) and write locks (which allow you to make changes).

Another method is timestamping. It's like giving each transaction a unique timestamp based on when it started. Transactions are then ordered by these timestamps, and conflicts (where two transactions try to change the same thing) are resolved by giving priority to the earlier transaction.

Optimistic concurrency control is more relaxed. It assumes conflicts are rare, so transactions don't lock data while working. Instead, conflicts are only checked at the end of a transaction. If there's a conflict, the transaction is undone and tried again. The goal of all these methods is to keep data safe and consistent, even when many people are accessing or changing it simultaneously.

ACID Properties:

- Atomicity: This means that when you do something in a database (like updating your profile), either everything you wanted to do gets done, or none of it happens. It's like ordering a combo meal—you get everything or nothing.
- Consistency: This ensures that the database doesn't end up in a weird state after a transaction. It's like making sure your bank balance is always accurate after you deposit or withdraw money.
- Isolation: This keeps transactions separate from each other, so one person's

changes don't mess with another's. It's like having individual rooms for different tasks, so they don't interfere with each other.

- Durability: Once something is saved in a database, it stays there even if something goes wrong (like a power outage). It's like writing something down on paper—it's there even if your computer crashes.

About recovery mechanisms:

These are like safety nets for databases. They help fix things if something goes wrong, like if the power suddenly goes out while you're using an app.

- Undo Recovery (Rollback): This undoes changes that didn't finish because of a problem, like if you were halfway through updating your profile but had to stop.
- Redo Recovery: This re-applies changes that were completed but not yet saved, like if you finished updating your profile but the system crashed before saving.

There are also tools like checkpointing (which saves the current state of things at regular intervals) and logging (which keeps a record of all changes), helping databases recover and stay reliable even when things get messy.

6.Code for the project

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/animations.css">
    <link rel="stylesheet" href="css/main.css">
    <link rel="stylesheet" href="css/index.css">
    <title>eDoc</title>
    <style>
        table{
            animation: transitionIn-Y-bottom 0.5s;
        }
    </style>

</head>
<body>

    <div class="full-height">
        <center>
        <table border="0">
            <tr>
                <td width="80%">
                    <font class="edoc-logo"> </font>
                    <font class="edoc-logo-sub">| DBMS PROJECT</font>
                </td>
                <td width="10%">
                  <a href="login.php"  class="non-style-link"><p class="nav-item">LOGIN</p></a>
                </td>
                <td  width="10%">
                    <a href="signup.php" class="non-style-link"><p class="nav-item" style="padding-right: 10px;">REGISTER</p></a>
                </td>
            </tr>

            <tr>
                <td  colspan="3">
```

```
            <tr>
                <td  colspan="3">
                    <p class="heading-text">Book your Appointment.</p>

                </td>
            </tr>
            <tr>
                <td  colspan="3">
                    <p class="sub-text2">Transform the way your hospital operates and cares for patients with our easy-to-use management
                </td>
            </tr>
            <tr>

                <td colspan="3">
                    <center>
                    <a href="login.php" >
                        <input type="button" value="Make Appointment" class="login-btn btn-primary btn" style="padding-left: 25px;padding
                    </a>
                </center>
                </td>

            </tr>
            <tr>
                <td colspan="3">

                </td>
            </tr>
        </table>
        <p class="sub-text2 footer-hashen">A Web Solution by Lamih.</p>
    </center>

    </div>
</body>
</html>
```

Index Page

```php
<?php

    $database= new mysqli("localhost","root","","lamihdb");
    if ($database->connect_error){
        die("Connection failed:  ".$database->connect_error);
    }

?>
```

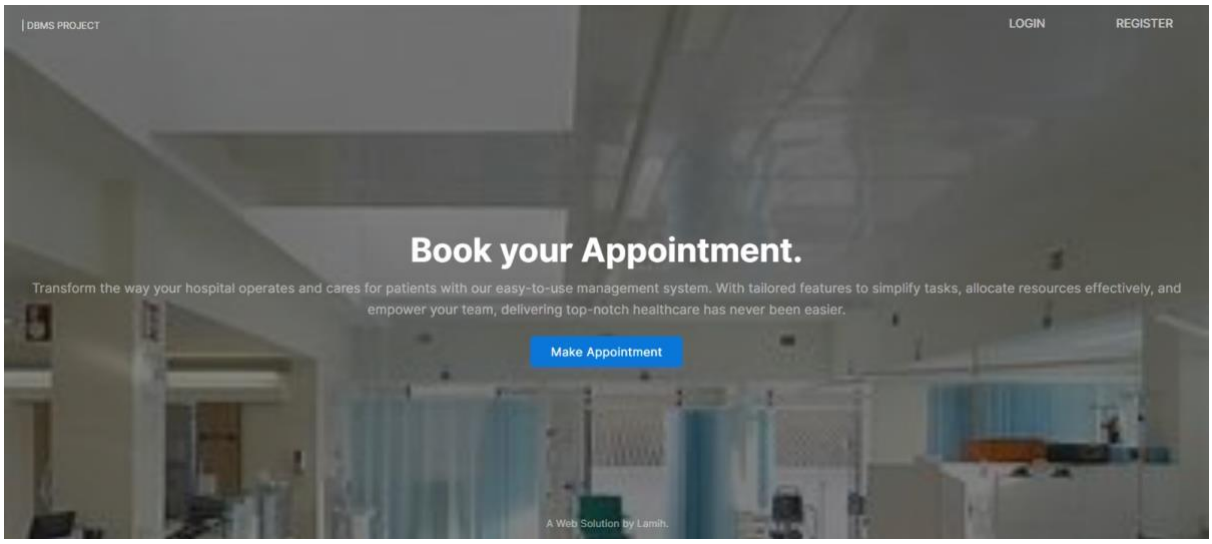
Connection

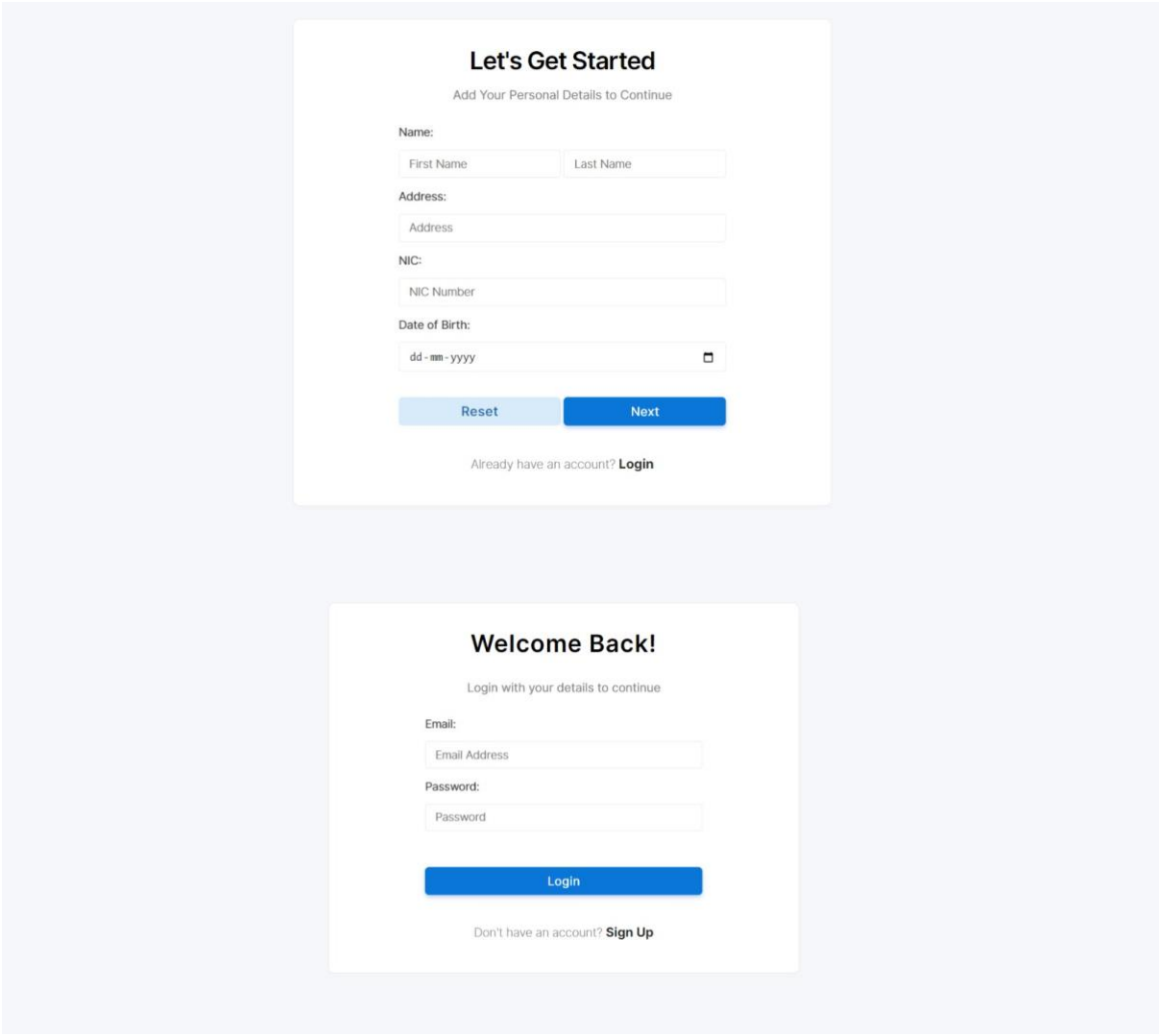
```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="../css/animations.css">
    <link rel="stylesheet" href="../css/main.css">
    <link rel="stylesheet" href="../css/admin.css">

    <title>Dashboard</title>
    <style>
        .dashbord-tables{
            animation: transitionIn-Y-over 0.5s;
        }
        .filter-container{
            animation: transitionIn-Y-bottom  0.5s;
        }
        .sub-table,.anime{
            animation: transitionIn-Y-bottom 0.5s;
        }
    </style>


</head>
<body>
    <?php

    //learn from w3schools.com

    session_start();

    if(isset($_SESSION["user"])){
        if(($_SESSION["user"])=="" or $_SESSION['usertype']!='p'){
            header("location: ../login.php");
        }else{
            $useremail=$_SESSION["user"];
```

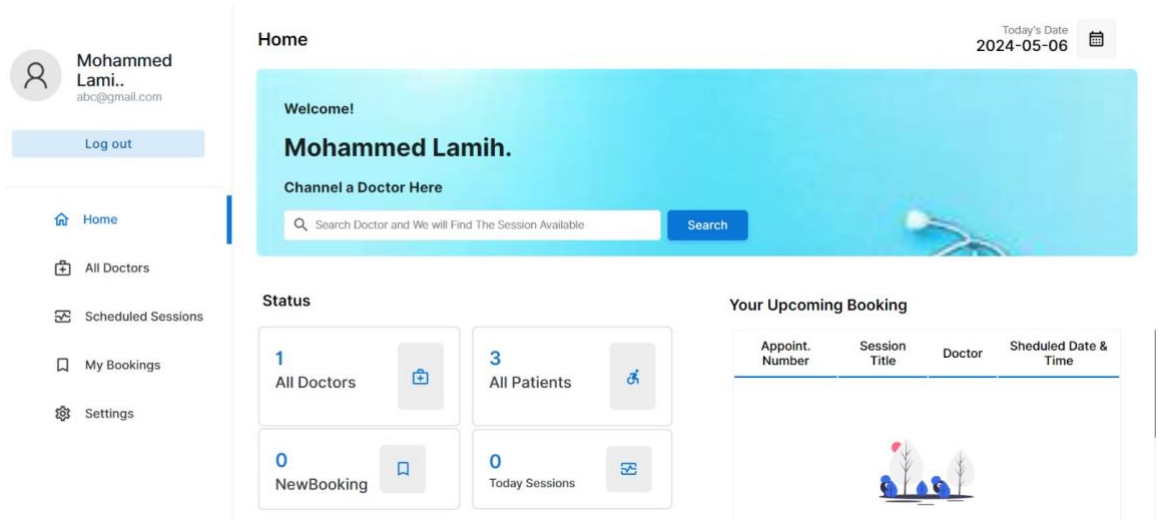
```php
    }

}else{
    header("location: ../login.php");
}


//import database
include("../connection.php");

$sqlmain= "select * from patient where pemail=?";
$stmt = $database->prepare($sqlmain);
$stmt->bind_param("s",$useremail);
$stmt->execute();
$userrow = $stmt->get_result();
$userfetch=$userrow->fetch_assoc();

$userid= $userfetch["pid"];
$username=$userfetch["pname"];


//echo $userid;
//echo $username;

?>
<div class="container">
    <div class="menu">
        <table class="menu-container" border="0">
            <tr>
                <td style="padding:10px" colspan="2">
                    <table border="0" class="profile-container">
                        <tr>
                            <td width="30%" style="padding-left:20px" >
                                <img src="../img/user.png" alt="" width="100%" style="border-radius:50%">
                            </td>
                            <td style="padding:0px;margin:0px;">
```

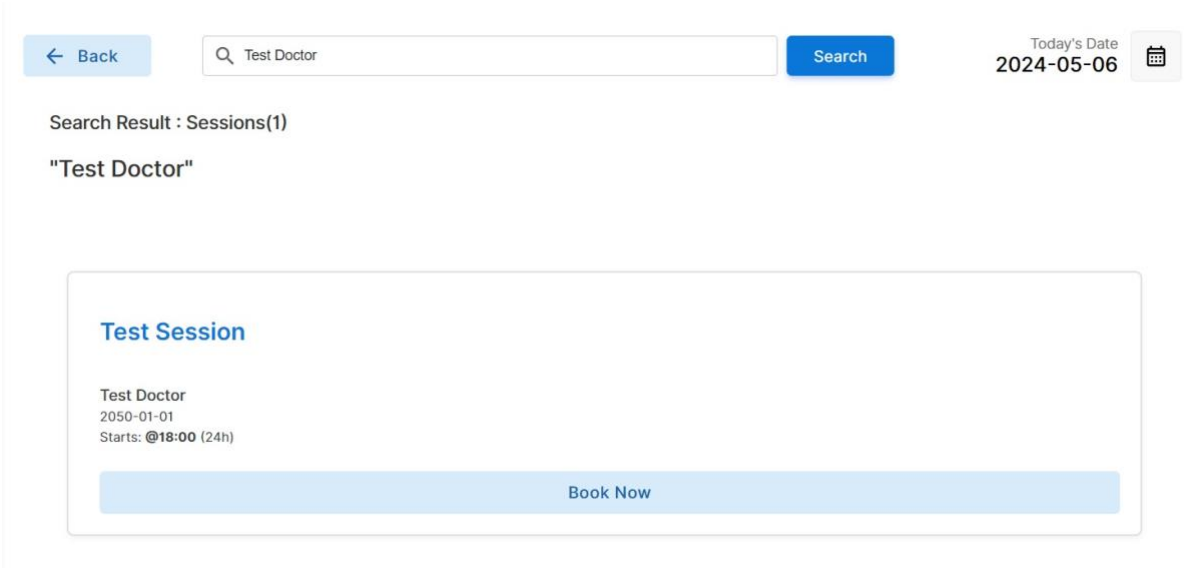
Patient

6. Results & Discussion (Screenshots):




Let's Get Started
Add Your Personal Details to Continue
Name:
First Name
Last Name
Address:
Address
NIC:
NIC Number
Date of Birth:
dd - mm - yyyy
Reset
Next
Already have an account? Login
Welcome Back!
Login with your details to continue
Email:
Email Address
Password:
Password
Login
Don't have an account? Sign Up


Mohammed Lami..
abc@gmail.com
Log out
Home
All Doctors
Scheduled Sessions
My Bookings
Settings
Home
Today's Date
2024-05-06
Welcome!
Mohammed Lamih.
Channel a Doctor Here
Search Doctor and We will Find The Session Available
Search
Status
1
All Doctors
3
All Patients
0
NewBooking
0
Today Sessions
Your Upcoming Booking
Appoint. Number
Session Title
Doctor
Sheduled Date & Time



8. Online Course Certificate: