MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DATABASE SYSTEMS LAB (CSE 2241) MINI PROJECT REPORT ON

HOSPITAL MANAGEMENT SYSTEM

SUBMITTED TO

Department of Computer Science & Engineering

Under the guidance of Dr. Anup Bhat B

by

Name	Registration Number	Roll Number	Semester
Arnav Devdatt Naik	230962336	55	IV
Om Dhondge	230962282	47	IV

Table of Contents

1. Abstract

The Hospital Management System (HMS) is designed to simplify and enhance the way hospitals handle their day-to-day operations. This system caters to various stakeholders, including doctors, nurses, administrative staff, and patients, ensuring that everyone has access to the information they need. Key features include patient registration, appointment scheduling, staff and doctor management, room assignments, and billing. With a robust database management system (DBMS) at its core, the HMS ensures secure and efficient data storage, enabling quick retrieval and reliable operations. By adhering to relational database principles, the system ensures the integrity of medical records while boosting the overall efficiency of hospital workflows. This integrated approach not only streamlines hospital processes but also improves patient care, staff productivity, and administrative oversight.

2. Problem Statement and Requirement Specification

2.1 Problem Statement

Hospitals often face challenges in managing large volumes of patient data, appointments, staff schedules, billing, and medical records using traditional manual or semi-automated methods. These approaches can lead to inefficiencies, data inconsistencies, delays in patient care, and increased administrative workload. There is a need for a centralized, secure, and efficient Hospital Management System (HMS) that can automate and streamline these processes, ensuring accurate data management, improved workflow, and better service delivery for all stakeholders within the hospital environment.

2.2 Requirement Specification

REQ 1: The system will maintain comprehensive records for patients, doctors, and staff.

REQ 2: Users will be able to search for patient records using criteria like name, ID, or contact information.

REQ 3: Upon patient discharge, the system will generate a detailed invoice covering treatment costs and room charges.

REQ 4: Updates to patient records, such as changes to ongoing treatments or contact details, will be seamlessly managed.

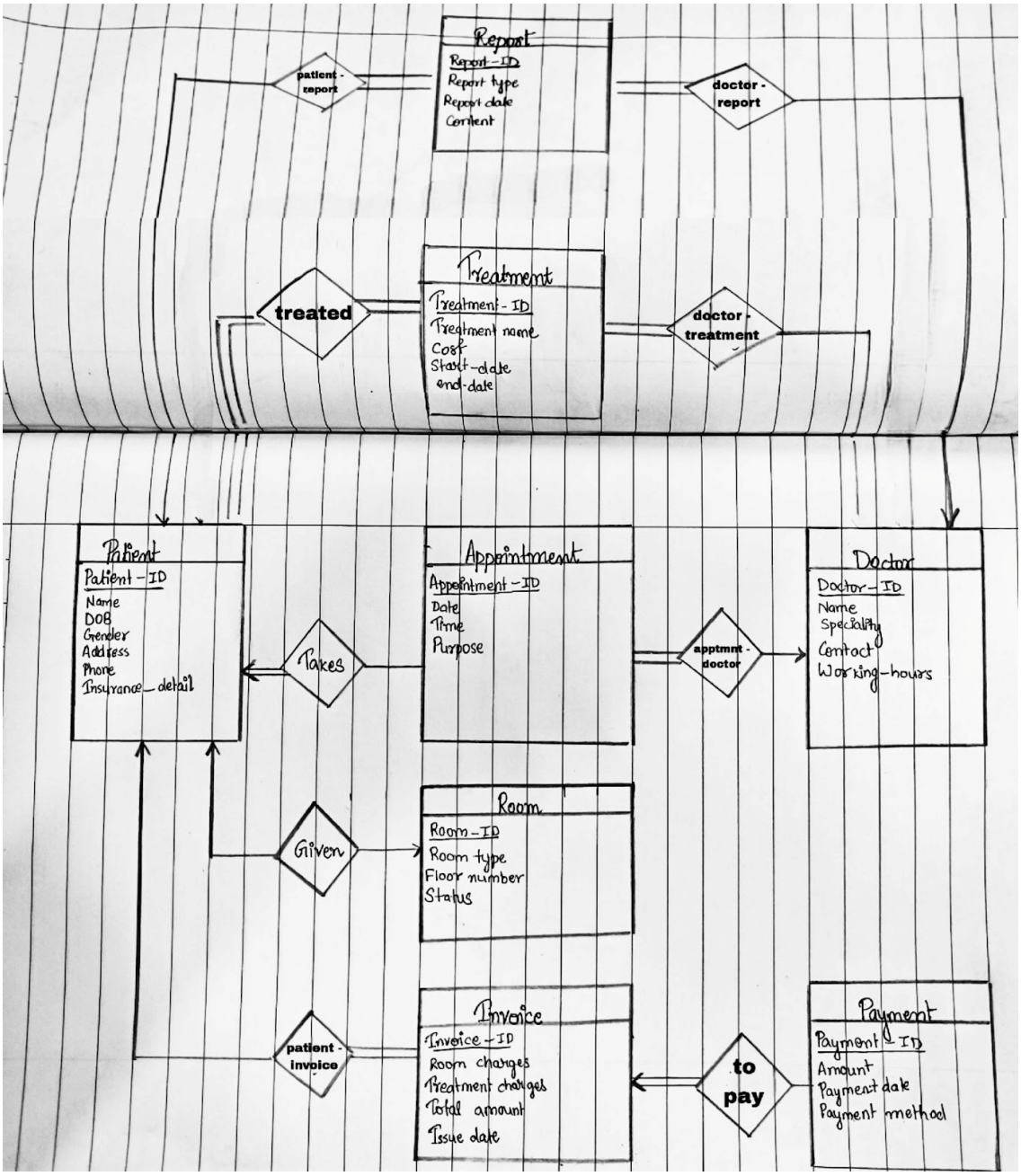
REQ 5: Room allocations will be automatically updated based on patient admissions and discharges.

REQ 6: Appointment scheduling will include conflict detection to avoid overlaps in doctors' schedules.

REQ 7: The system will generate reports, such as daily admission and discharge summaries, revenue analysis, and inventory updates.

3. Project Design

3.1. ER Diagram



3.2. Relational Tables

1) Patient (Patient_ID (PK), Name, DOB, Gender, Address, Phone, Insurance_Detail)

2) Doctor (Doctor_ID (PK), Name, Specialty, Contact, Working_Hours)

3) Appointment (Appointment_ID (PK), Patient_ID (FK), Doctor_ID (FK), Date, Time, Purpose)

4) Treatment (Treatment_ID (PK), Doctor_ID (FK), Treatment_Name, Cost, Start_Date, End_Date)

5) Report (Report_ID (PK), patient_id (fk), doctor_id (fk) Report_Type, Report_Date, diagnosis)

6) Room (Room_ID (PK), Patient_ID (FK), Room_Type, Floor_Number, Status)

7) Invoice (Invoice_ID (PK), Patient_ID (FK), Room_Charges, Treatment_Charges, Total_Amount, Issue_Date)

8) Payment (Payment_ID (PK), Invoice_ID (FK), Amount, Payment_Date, Payment_Method)

9) Patient_Treatment (Patient_ID (PK), Treatment_ID (PK))

3.3. Sample Data

Patient Table

Patient_ID	Name	DOB	Gender	Address	Phone	Insurance_Detail
1	John Doe	1980-05-14	Male	123 Elm St	555-1001	ABC Insurance
2	Jane Smith	1992-07-21	Female	456 Oak St	555-1002	XYZ Insurance
3	Michael Johnson	1985-12-05	Male	789 Pine St	555-1003	LMN Insurance
4	Emily Davis	1998-03-30	Female	321 Maple St	555-1004	PQR Insurance
5	Daniel Brown	1975-09-18	Male	654 Birch St	555-1005	ABC Insurance
6	Sarah Wilson	1990-11-23	Female	987 Cedar St	555-1006	XYZ Insurance
7	David Martinez	1982-07-07	Male	741 Spruce St	555-1007	LMN Insurance
8	Laura Thompson	1995-02-14	Female	852 Walnut St	555-1008	PQR Insurance
9	James Anderson	1978-08-22	Male	963 Chestnut St	555-1009	ABC Insurance
10	Olivia White	1987-06-17	Female	159 Redwood St	555-1010	XYZ Insurance

Doctor Table

Doctor_ID	Name	Specialty	Contact	Working_Hours
1	Dr. Alice Brown	Cardiology	555-2001	9 AM - 5 PM
2	Dr. Bob White	Neurology	555-2002	10 AM - 6 PM
3	Dr. Charlie Green	Orthopedics	555-2003	8 AM - 4 PM
4	Dr. Diana Black	Pediatrics	555-2004	11 AM - 7 PM
5	Dr. Evan Grey	Dermatology	555-2005	9 AM - 3 PM

Appointment Table

Appointment_ID	Patient_ID	Doctor_ID	Date	Time	Purpose
1	1	1	2025-03-27	10:00:00	Regular Checkup
2	2	2	2025-03-28	11:30:00	Consultation
3	3	3	2025-03-29	14:00:00	Fracture Checkup
4	4	4	2025-03-30	16:30:00	Child Vaccination
5	5	5	2025-03-31	09:45:00	Skin Allergy Consultation

Treatment Table

Treatment_ID	Doctor_ID	Treatment_Name	Cost	Start_Date	End_Date
1	1	Heart Surgery	5000.00	2025-02-01	2025-02-10
2	2	Brain MRI	1500.00	2025-03-05	NULL
3	3	Knee Replacement	10000.00	2025-01-15	2025-01-30

4	4	Flu Treatment	300.00	2025-03-10	2025-03-12
5	5	Acne Therapy	500.00	2025-02-20	2025-03-05

Report Table

Report_ID	Patient_ID	Doctor_ID	Report_Type	Report_Date	Diagnosis
1	1	1	ECG	2025-03-01	Normal
2	2	2	MRI Scan	2025-03-06	Mild anomaly detected
3	3	3	X-Ray	2025-02-15	Fracture healing well
4	4	4	Blood Test	2025-03-11	All levels normal
5	5	5	Skin Biopsy	2025-03-07	No malignancy found

Room Table

Room_ID	Patient_ID	Room_Type	Floor_Number	Status
101	1	Single	2	Occupied
102	2	Deluxe	3	Available
103	3	ICU	1	Occupied
104	4	General	4	Available
105	5	Semi-Private	2	Occupied

3.4. Normalization

1. Patient Relation

Attributes: Patient_ID (PK), Name, DOB, Gender, Address, Phone, Insurance_Detail

Functional Dependencies:

- Patient_ID → Name, DOB, Gender, Address, Phone, Insurance_Detail

Candidate Keys: Patient_ID

BCNF Analysis: The only determinant is Patient_ID, which is the primary key and thus a superkey. Therefore, the Patient relation is in BCNF.

2. Doctor Relation

Attributes: Doctor_ID (PK), Name, Specialty, Contact, Working_Hours

Functional Dependencies:

- Doctor_ID → Name, Specialty, Contact, Working_Hours

Candidate Keys: Doctor_ID

BCNF Analysis: The only determinant is Doctor_ID, which is the primary key and thus a superkey. Therefore, the Doctor relation is in BCNF.

3. Appointment Relation

Attributes: Appointment_ID (PK), Patient_ID (FK), Doctor_ID (FK), Date, Time, Purpose

Functional Dependencies:

- Appointment_ID → Patient_ID, Doctor_ID, Date, Time, Purpose
- {Patient_ID, Doctor_ID, Date, Time} → Appointment_ID, Purpose

Candidate Keys: Appointment_ID, {Patient_ID, Doctor_ID, Date, Time}

BCNF Analysis: Both determinants (Appointment_ID and {Patient_ID, Doctor_ID, Date, Time}) are candidate keys, so the Appointment relation is in BCNF.

4. Treatment Relation

Attributes: Treatment_ID (PK), Doctor_ID (FK), Treatment_Name, Cost, Start_Date, End_Date

Functional Dependencies:

- Treatment_ID → Doctor_ID, Treatment_Name, Cost, Start_Date, End_Date

Candidate Keys: Treatment_ID

BCNF Analysis: The only determinant is Treatment_ID, which is the primary key and thus a superkey. Therefore, the Treatment relation is in BCNF.

5. Report Relation

Attributes: Report_ID (PK), Patient_ID (FK), Doctor_ID (FK), Report_Type, Report_Date, Diagnosis

Functional Dependencies:

- Report_ID → Patient_ID, Doctor_ID, Report_Type, Report_Date, Diagnosis

Candidate Keys: Report_ID

BCNF Analysis: The only determinant is Report_ID, which is the primary key and thus a superkey. Therefore, the Report relation is in BCNF.

6. Room Relation

Attributes: Room_ID (PK), Patient_ID (FK), Room_Type, Floor_Number, Status

Functional Dependencies:

- Room_ID → Patient_ID, Room_Type, Floor_Number, Status

Candidate Keys: Room_ID

BCNF Analysis: The only determinant is Room_ID, which is the primary key and thus a superkey. Therefore, the Room relation is in BCNF.

7. Invoice Relation

Attributes: Invoice_ID (PK), Patient_ID (FK), Room_Charges, Treatment_Charges, Total_Amount, Issue_Date

Functional Dependencies:

- Invoice_ID → Patient_ID, Room_Charges, Treatment_Charges, Total_Amount, Issue_Date

Candidate Keys: Invoice_ID

BCNF Analysis: The only determinant is Invoice_ID, which is the primary key and thus a superkey. Therefore, the Invoice relation is in BCNF.

8. Payment Relation

Attributes: Payment_ID (PK), Invoice_ID (FK), Amount, Payment_Date, Payment_Method

Functional Dependencies:

- Payment_ID → Invoice_ID, Amount, Payment_Date, Payment_Method

Candidate Keys: Payment_ID

BCNF Analysis: The only determinant is Payment_ID, which is the primary key and thus a superkey. Therefore, the Payment relation is in BCNF.

9. Patient_Treatment Relation

Attributes: Patient_ID (PK), Treatment_ID (PK)

Functional Dependencies:

- {Patient_ID, Treatment_ID} → (no additional attributes)

Candidate Keys: {Patient_ID, Treatment_ID}

BCNF Analysis: This is a junction table with a composite primary key and no additional attributes. The only determinant is the primary key itself, so the Patient_Treatment relation is in BCNF.

4. DDL Commands

```
CREATE TABLE Patient (
    Patient_ID INT PRIMARY KEY,
    Name VARCHAR2(255) NOT NULL,
    DOB DATE NOT NULL,
    Gender VARCHAR2(10) CHECK (Gender IN ('Male', 'Female', 'Other')),
    Address CLOB,
    Phone VARCHAR2(15) UNIQUE,
    Insurance_Detail CLOB
```

```
);

CREATE TABLE Doctor (
    Doctor_ID INT PRIMARY KEY,
    Name VARCHAR2(255) NOT NULL,
    Specialty VARCHAR2(255) NOT NULL,
    Contact VARCHAR2(15) UNIQUE,
    Working_Hours VARCHAR2(50)
);

CREATE TABLE Appointment (
    Appointment_ID INT PRIMARY KEY,
    Patient_ID INT,
    Doctor_ID INT,
    Appointment_Date DATE NOT NULL,
    Appointment_Time TIMESTAMP NOT NULL,
    Purpose CLOB,
    FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID) ON DELETE CASCADE,
    FOREIGN KEY (Doctor_ID) REFERENCES Doctor(Doctor_ID) ON DELETE SET NULL
);

CREATE TABLE Treatment (
    Treatment_ID INT PRIMARY KEY,
    Doctor_ID INT,
    Treatment_Name VARCHAR2(255) NOT NULL,
    Cost NUMBER(10,2) NOT NULL,
    Start_Date DATE,
    End_Date DATE,
    FOREIGN KEY (Doctor_ID) REFERENCES Doctor(Doctor_ID) ON DELETE SET NULL
);

CREATE TABLE Report (
```

```
    Report_ID INT PRIMARY KEY,
    Patient_ID INT,
    Doctor_ID INT,
    Report_Type VARCHAR2(255) NOT NULL,
    Report_Date DATE NOT NULL,
    Diagnosis CLOB,
    FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID) ON DELETE CASCADE,
    FOREIGN KEY (Doctor_ID) REFERENCES Doctor(Doctor_ID) ON DELETE SET NULL
);

CREATE TABLE Room (
    Room_ID INT PRIMARY KEY,
    Patient_ID INT,
    Room_Type VARCHAR2(50),
    Floor_Number INT,
    Status VARCHAR2(20) CHECK (Status IN ('Occupied', 'Available')),
    FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID) ON DELETE SET NULL
);

CREATE TABLE Invoice (
    Invoice_ID INT PRIMARY KEY,
    Patient_ID INT,
    Room_Charges NUMBER(10,2) NOT NULL,
    Treatment_Charges NUMBER(10,2) NOT NULL,
    Total_Amount NUMBER(10,2) NOT NULL,
    Issue_Date DATE NOT NULL,
    FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID) ON DELETE CASCADE
);

CREATE TABLE Payment (
    Payment_ID INT PRIMARY KEY,
    Invoice_ID INT,
```

```
    Amount NUMBER(10,2) NOT NULL,
    Payment_Date DATE NOT NULL,
    Payment_Method VARCHAR2(50),
    FOREIGN KEY (Invoice_ID) REFERENCES Invoice(Invoice_ID) ON DELETE CASCADE
);

CREATE TABLE Patient_Treatment (
    Patient_ID INT,
    Treatment_ID INT,
    PRIMARY KEY (Patient_ID, Treatment_ID),
    FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID) ON DELETE CASCADE,
    FOREIGN KEY (Treatment_ID) REFERENCES Treatment(Treatment_ID) ON DELETE CASCADE
);
```

5. SQL Queries

1. Get all patient details:

```
SELECT * FROM Patient;
```

2. Get all doctors and their specialties:

```
SELECT Doctor_ID, Name, Specialty FROM Doctor;
```

3. Get all appointments for a specific doctor

```
SELECT a.Appointment_ID, p.Name AS Patient_Name, a.Date, a.Time, a.Purpose
FROM Appointment a
JOIN Patient p ON a.Patient_ID = p.Patient_ID
WHERE a.Doctor_ID = 1;
```

4. Check available rooms:

```
SELECT * FROM Room WHERE Status = 'Available';
```

5. Get total bill for each patient:

```
SELECT i.Invoice_ID, p.Name AS Patient_Name, i.Room_Charges, i.Treatment_Charges, i.Total_Amount
FROM Invoice i
JOIN Patient p ON i.Patient_ID = p.Patient_ID;
```

6. Treatments received by a patient:

```
SELECT t.Treatment_Name, t.Cost, t.Start_Date, t.End_Date
FROM Patient_Treatment pt
JOIN Treatment t ON pt.Treatment_ID = t.Treatment_ID
WHERE pt.Patient_ID = 3;
```

```
-- Replace 3 with the desired patient ID.
```

7. Patients currently admitted (rooms occupied):

```
SELECT r.Room_ID, r.Room_Type, r.Floor_Number, p.Name
FROM Room r
JOIN Patient p ON r.Patient_ID = p.Patient_ID
WHERE r.Status = 'Occupied';
```

8. Unpaid or partially paid invoices (optional enhancement):

```
SELECT i.Invoice_ID, p.Name, i.Total_Amount, SUM(pay.Amount) AS Paid
FROM Invoice i
JOIN Patient p ON i.Patient_ID = p.Patient_ID
LEFT JOIN Payment pay ON i.Invoice_ID = pay.Invoice_ID
GROUP BY i.Invoice_ID, p.Name, i.Total_Amount
HAVING SUM(pay.Amount) < i.Total_Amount;
```

9. Generate daily admission and discharge summary:

```
-- Admissions (based on appointments)
SELECT a.Date AS Admission_Date, COUNT(*) AS Total_Admissions
FROM Appointment a
GROUP BY a.Date
ORDER BY a.Date DESC;
```

```
-- Discharges (based on invoice issue date)
SELECT i.Issue_Date AS Discharge_Date, COUNT(*) AS Total_Discharged
FROM Invoice i
GROUP BY i.Issue_Date
ORDER BY i.Issue_Date DESC;
```

10. Revenue generated each day:

```
SELECT Issue_Date, SUM(Total_Amount) AS Daily_Revenue
FROM Invoice
GROUP BY Issue_Date
ORDER BY Issue_Date DESC;
```

11. Search for patient by name or contact:

```
SELECT * FROM Patient
WHERE LOWER(Name) LIKE '%john%' OR Phone LIKE '%1001%';
```

12. List doctors with no appointments today:

```
SELECT d.Doctor_ID, d.Name
FROM Doctor d
WHERE NOT EXISTS (
  SELECT 1 FROM Appointment a
```

```
  WHERE a.Doctor_ID = d.Doctor_ID AND a.Date = SYSDATE
);
```

13. Most frequent treatments:

```
SELECT t.Treatment_Name, COUNT(*) AS Frequency
FROM Patient_Treatment pt
JOIN Treatment t ON pt.Treatment_ID = t.Treatment_ID
GROUP BY t.Treatment_Name
ORDER BY Frequency DESC;
```

14. Update contact number of a patient:

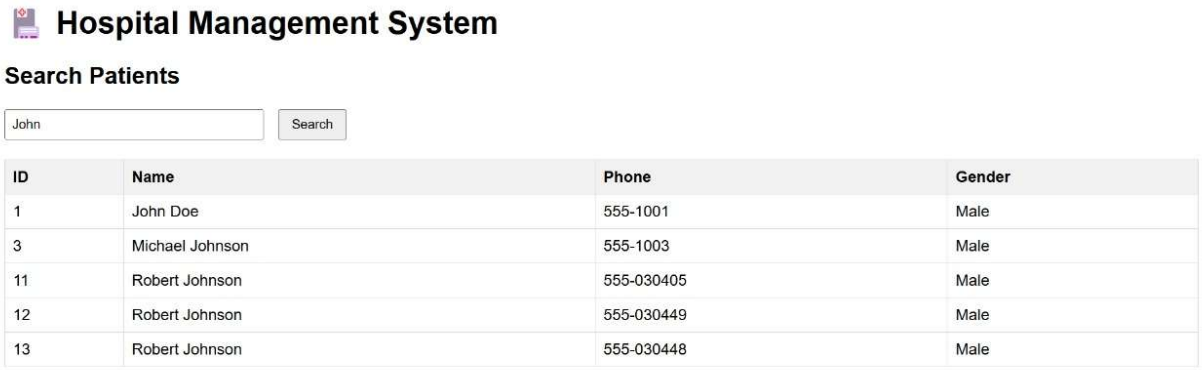
```
UPDATE Patient
SET Phone = '555-9999'
WHERE Patient_ID = 4;
```

15. Delete old appointments (e.g., before Jan 2024);

```
DELETE FROM Appointment
WHERE Date < TO_DATE('2024-01-01', 'YYYY-MM-DD');
```

6. UI Design

6.1. UI Design Screenshots



6.2. Database Connectivity Code

Python Code for DB connectivity :

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
import cx_Oracle

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
```

```
    allow_headers=["*"],
)

# Replace with your actual DB credentials and TNS
conn = cx_Oracle.connect("system", "omthegreat", "localhost/FREE")
cursor = conn.cursor()

@app.get("/patients")
def search_patients(search: str = ""):
    query = """
        SELECT Patient_ID, Name, Phone, Gender FROM Patient
        WHERE LOWER(Name) LIKE :search OR Phone LIKE :search OR TO_CHAR(Patient_ID) = :id
    """
    search_term = f"%{search.lower()}%"
    cursor.execute(query, {'search': search_term, 'id': search})
    rows = cursor.fetchall()
    return [{"id": r[0], "name": r[1], "phone": r[2], "gender": r[3]} for r in rows]
```

7. PL/SQL Functions and Triggers

```
DROP TABLE Room_Status_Log;

CREATE TABLE Room_Status_Log (
    Log_ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
```

```
    Room_ID NUMBER,
    Old_Status VARCHAR2(20),
    New_Status VARCHAR2(20),
    Change_Date DATE
);
```

-- REQ 2: Improved patient search with better name matching and error handling

```
CREATE OR REPLACE FUNCTION Search_Patient(
    p_id IN NUMBER DEFAULT NULL,
    p_name IN VARCHAR2 DEFAULT NULL
) RETURN SYS_REFCURSOR IS
  result SYS_REFCURSOR;
BEGIN
  IF p_id IS NULL AND p_name IS NULL THEN
    RAISE_APPLICATION_ERROR(-20001, 'At least one search parameter (ID or name) must be provided');
  END IF;

  OPEN result FOR
  SELECT Patient_ID, Name, DOB, Gender, Address, Phone, Insurance_Detail
  FROM Patient
  WHERE (p_id IS NULL OR Patient_ID = p_id)
    AND (p_name IS NULL OR REGEXP_LIKE(LOWER(Name), LOWER(p_name), 'i'));

  RETURN result;
EXCEPTION
  WHEN OTHERS THEN
```

```
    IF result%ISOPEN THEN
      CLOSE result;
    END IF;
    RAISE;
END;
/
```

-- REQ 3: Enhanced invoice generation with actual invoice creation

```
CREATE OR REPLACE PROCEDURE Generate_Invoice(p_patient_id IN NUMBER) IS
  v_total_treatment_cost NUMBER := 0;
  v_room_charge NUMBER := 0;
  v_invoice_id NUMBER;
  v_patient_exists NUMBER;
BEGIN
  -- Check if patient exists
  SELECT COUNT(*) INTO v_patient_exists FROM Patient WHERE Patient_ID = p_patient_id;
  IF v_patient_exists = 0 THEN
    RAISE_APPLICATION_ERROR(-20002, 'Patient not found');
  END IF;

  -- Calculate treatment costs from Patient_Treatment junction table (more accurate)
  SELECT NVL(SUM(t.Cost), 0) INTO v_total_treatment_cost
  FROM Treatment t
  JOIN Patient_Treatment pt ON t.Treatment_ID = pt.Treatment_ID
  WHERE pt.Patient_ID = p_patient_id;

  -- Calculate room charges with proper duration calculation
```

```
SELECT CASE r.Room_Type
      WHEN 'Single' THEN 1000
      WHEN 'Deluxe' THEN 2000
      WHEN 'ICU' THEN 3000
      WHEN 'General' THEN 500
      WHEN 'Semi-Private' THEN 800
      ELSE 0 END *
      (SELECT NVL(MAX(TRUNC(SYSDATE) - TRUNC(a.Appointment_Date)), 1)
       FROM Appointment a
       WHERE a.Patient_ID = p_patient_id)
      INTO v_room_charge
FROM Room r
WHERE r.Patient_ID = p_patient_id AND r.Status = 'Occupied';

-- Generate invoice sequence
SELECT NVL(MAX(Invoice_ID), 0) + 1 INTO v_invoice_id FROM Invoice;

-- Insert into Invoice table
INSERT INTO Invoice (Invoice_ID, Patient_ID, Room_Charges, Treatment_Charges, Total_Amount, Issue_Date)
VALUES (v_invoice_id, p_patient_id, v_room_charge, v_total_treatment_cost,
       (v_room_charge + v_total_treatment_cost), SYSDATE);

COMMIT;

DBMS_OUTPUT.PUT_LINE('Invoice generated successfully for Patient ID ' || p_patient_id);
DBMS_OUTPUT.PUT_LINE('Invoice ID: ' || v_invoice_id);
DBMS_OUTPUT.PUT_LINE('Treatment Charges: Rs. ' || v_total_treatment_cost);
DBMS_OUTPUT.PUT_LINE('Room Charges: Rs. ' || v_room_charge);
```

```
  DBMS_OUTPUT.PUT_LINE('Total Amount: Rs. ' || (v_total_treatment_cost + v_room_charge));
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Error generating invoice: ' || SQLERRM);
    RAISE;
END;
/
```

-- REQ 4 : Enhanced patient update with validation

```
CREATE OR REPLACE PROCEDURE Update_Patient_Info(
    p_id IN NUMBER,
    new_contact IN VARCHAR2 DEFAULT NULL,
    new_insurance IN CLOB DEFAULT NULL
) IS
  v_update_count NUMBER := 0;
BEGIN
  IF new_contact IS NULL AND new_insurance IS NULL THEN
    RAISE_APPLICATION_ERROR(-20003, 'At least one update value must be provided');
  END IF;

  IF new_contact IS NOT NULL THEN
    -- Validate phone format (simple validation)
    IF NOT REGEXP_LIKE(new_contact, '^[0-9]{10,15}$') THEN
      RAISE_APPLICATION_ERROR(-20004, 'Invalid phone number format');
    END IF;
```

```
    UPDATE Patient SET Phone = new_contact WHERE Patient_ID = p_id;
    v_update_count := v_update_count + SQL%ROWCOUNT;
  END IF;

  IF new_insurance IS NOT NULL THEN
    UPDATE Patient SET Insurance_Detail = new_insurance WHERE Patient_ID = p_id;
    v_update_count := v_update_count + SQL%ROWCOUNT;
  END IF;

  IF v_update_count = 0 THEN
    RAISE_APPLICATION_ERROR(-20005, 'Patient ID not found');
  END IF;

  COMMIT;
  DBMS_OUTPUT.PUT_LINE('Patient information updated successfully');
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    RAISE;
END;
/
```

-- REQ 5: Improved room status trigger with logging

```
CREATE OR REPLACE TRIGGER trg_update_room_status
AFTER UPDATE OF Status ON Room
FOR EACH ROW
DECLARE
```

```
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  IF :NEW.Status = 'Available' AND :OLD.Status != 'Available' THEN
    UPDATE Room SET Patient_ID = NULL WHERE Room_ID = :NEW.Room_ID;

    -- Log the room status change
    INSERT INTO Room_Status_Log (Room_ID, Old_Status, New_Status, Change_Date)
    VALUES (:NEW.Room_ID, :OLD.Status, :NEW.Status, SYSDATE);

    COMMIT;
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    -- Log error but don't prevent original update
    DBMS_OUTPUT.PUT_LINE('Error in room status trigger: ' || SQLERRM);
END;
/
```

-- REQ 6: Enhanced appointment conflict check with time window

```
CREATE OR REPLACE FUNCTION Check_Appointment_Conflict(
    p_doctor_id NUMBER,
    p_date DATE,
    p_time TIMESTAMP,
    p_duration_min NUMBER DEFAULT 30
) RETURN VARCHAR2 IS
  v_conflict_count NUMBER;
```

```
BEGIN
  IF p_doctor_id IS NULL OR p_date IS NULL OR p_time IS NULL THEN
    RETURN 'Invalid parameters';
  END IF;

  SELECT COUNT(*) INTO v_conflict_count
  FROM Appointment
  WHERE Doctor_ID = p_doctor_id
    AND Appointment_Date = p_date
    AND (
      -- Existing appointment starts during new appointment
      (Appointment_Time >= p_time AND
       Appointment_Time < p_time + (p_duration_min/1440))
      OR
      -- New appointment starts during existing appointment
      (p_time >= Appointment_Time AND
       p_time < Appointment_Time + (30/1440)) -- Assuming standard 30-min appointments
    );

  IF v_conflict_count > 0 THEN
    RETURN 'Conflict Detected: ' || v_conflict_count || ' overlapping appointment(s)';
  ELSE
    RETURN 'No Conflict';
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    RETURN 'Error checking appointment: ' || SQLERRM;
END;
```

/

-- REQ 7: Enhanced daily report with counts and formatting

```
CREATE OR REPLACE PROCEDURE Daily_Report(
    p_date IN DATE DEFAULT TRUNC(SYSDATE),
    p_include_details IN BOOLEAN DEFAULT TRUE
) IS
  v_admission_count NUMBER := 0;
  v_discharge_count NUMBER := 0;
  v_appt_rec Appointment%ROWTYPE;
  v_room_rec Room%ROWTYPE;
BEGIN
  -- Get admission count (appointments)
  SELECT COUNT(*) INTO v_admission_count
  FROM Appointment
  WHERE TRUNC(Appointment_Date) = TRUNC(p_date);

  -- Get discharge count (rooms made available)
  SELECT COUNT(*) INTO v_discharge_count
  FROM Room_Status_Log
  WHERE New_Status = 'Available' AND TRUNC(Change_Date) = TRUNC(p_date);

  DBMS_OUTPUT.PUT_LINE('=== DAILY HOSPITAL REPORT FOR ' || TO_CHAR(p_date, 'DD-MON-
YYYY') || ' ===');
  DBMS_OUTPUT.PUT_LINE('Total Admissions: ' || v_admission_count);
  DBMS_OUTPUT.PUT_LINE('Total Discharges: ' || v_discharge_count);
```

```
IF p_include_details THEN
  DBMS_OUTPUT.PUT_LINE(CHR(10) || '--- Admission Details ---');
  FOR appt_rec IN (
    SELECT a.Appointment_ID, p.Name AS Patient_Name, d.Name AS Doctor_Name,
           TO_CHAR(a.Appointment_Time, 'HH24:MI') AS Time
    FROM Appointment a
    JOIN Patient p ON a.Patient_ID = p.Patient_ID
    JOIN Doctor d ON a.Doctor_ID = d.Doctor_ID
    WHERE TRUNC(a.Appointment_Date) = TRUNC(p_date)
    ORDER BY a.Appointment_Time
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Time: ' || appt_rec.Time || ' | Patient: ' || appt_rec.Patient_Name ||
                         ' | Doctor: ' || appt_rec.Doctor_Name);
  END LOOP;

  DBMS_OUTPUT.PUT_LINE(CHR(10) || '--- Discharge Details ---');
  FOR room_rec IN (
    SELECT r.Room_ID, r.Room_Type, p.Name AS Patient_Name
    FROM Room_Status_Log l
    JOIN Room r ON l.Room_ID = r.Room_ID
    LEFT JOIN Patient p ON r.Patient_ID = p.Patient_ID
    WHERE l.New_Status = 'Available' AND TRUNC(l.Change_Date) = TRUNC(p_date)
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Room: ' || room_rec.Room_Type || ' ' || room_rec.Room_ID ||
                         ' | Patient: ' || NVL(room_rec.Patient_Name, 'N/A'));
  END LOOP;
END IF;
```

```
  DBMS_OUTPUT.PUT_LINE('=== END OF REPORT ===');
  DBMS_OUTPUT.PUT_LINE ('REQ 8 FULFLLED');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error generating report: ' || SQLERRM);
END;
/



-- Additional utility function to calculate patient bill
CREATE OR REPLACE FUNCTION Calculate_Patient_Balance(
    p_patient_id IN NUMBER
) RETURN NUMBER IS
  v_total_invoiced NUMBER := 0;
  v_total_paid NUMBER := 0;
BEGIN
  -- Get total invoiced amount
  SELECT NVL(SUM(Total_Amount), 0) INTO v_total_invoiced
  FROM Invoice
  WHERE Patient_ID = p_patient_id;

  -- Get total payments made
  SELECT NVL(SUM(p.Amount), 0) INTO v_total_paid
  FROM Payment p
  JOIN Invoice i ON p.Invoice_ID = i.Invoice_ID
  WHERE i.Patient_ID = p_patient_id;

  RETURN (v_total_invoiced - v_total_paid);
```

```
EXCEPTION
  WHEN OTHERS THEN
    RETURN NULL;
END;
/
```