```
-- Create the 'hospital_management' database
CREATE DATABASE IF NOT EXISTS hospital_management;

-- Switch to the 'hospital_management' database
USE hospital_management;
-- Create Tables
CREATE TABLE Patients (
   PatientID INT PRIMARY KEY,
   FirstName VARCHAR(255),
   LastName VARCHAR(255),
   DateOfBirth DATE,
   Gender VARCHAR(10),
   ContactNumber VARCHAR(15),
   Address VARCHAR(255),
   InsuranceLimit DECIMAL(10,2)
);

CREATE TABLE UserRoles (
   RoleID INT PRIMARY KEY,
   RoleName VARCHAR(50)
);

CREATE TABLE Users (
   UserID INT PRIMARY KEY,
   Username VARCHAR(50),
   Password VARCHAR(255),
   RoleID INT,
   FOREIGN KEY (RoleID) REFERENCES UserRoles(RoleID)
);

CREATE TABLE Diagnosis (
   DiagnosisID INT PRIMARY KEY,
   PatientID INT,
   DiagnosisDate DATE,
   DiagnosisDetails TEXT,
   FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);

CREATE TABLE Bills (
   BillID INT PRIMARY KEY,
    PatientID INT,
    BillAmount DECIMAL(10,2),
    CheckoutDate DATE,
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);

-- Insert Values in tables

INSERT INTO Patients (PatientID, FirstName, LastName, DateOfBirth, Gender,
ContactNumber, Address, InsuranceLimit)
VALUES
(1, 'John', 'Doe', '1990-05-15', 'Male', '1234567890', '123 Main St', 5000.00),
(2, 'Jane', 'Smith', '1985-08-21', 'Female', '9876543210', '456 Oak St', 8000.00);

INSERT INTO UserRoles (RoleID, RoleName)
VALUES
(1, 'Admin'),
(2, 'Doctor'),
(3, 'Nurse');

INSERT INTO Users (UserID, Username, Password, RoleID)
VALUES
(1, 'admin_user', 'admin_password', 1),
(2, 'doctor_user', 'doctor_password', 2),
(3, 'nurse_user', 'nurse_password', 3);

INSERT INTO Diagnosis (DiagnosisID, PatientID, DiagnosisDate, DiagnosisDetails)
VALUES
(1, 1, '2022-01-10', 'Flu'),
(2, 2, '2022-02-05', 'Broken Arm');

INSERT INTO Bills (BillID, PatientID, BillAmount, CheckoutDate)
VALUES
(1, 1, 100.00, '2022-01-15'),
(2, 2, 500.00, '2022-02-20');

-- Show tables

SELECT * FROM Patients;
SELECT * FROM UserRoles;
SELECT * FROM Users;
SELECT * FROM Diagnosis;
SELECT * FROM Bills;

-- simple view for each table

Create view Patients_view as select * from Patients;
select * from Patients_view;

Create view UserRoles_view as select * from UserRoles;
select * from UserRoles_view;

Create view UserRoles_view as select * from UserRoles;
select * from UserRoles_view;

Create view Users_view as select * from Users;
select * from Users_view;

CREATE VIEW DiagnosisView AS SELECT * FROM Diagnosis;
SELECT * FROM DiagnosisView;

CREATE VIEW BillsView AS SELECT * FROM Bills;
SELECT * FROM BillsView;

-- Combines patient details with diagnosis and billing information.

CREATE VIEW PatientDetails AS
SELECT Patients.PatientID, FirstName, LastName, DateOfBirth, Gender, ContactNumber,
Address, InsuranceLimit, DiagnosisDetails, BillAmount, CheckoutDate
FROM Patients
LEFT JOIN Diagnosis ON Patients.PatientID = Diagnosis.PatientID
LEFT JOIN Bills ON Patients.PatientID = Bills.PatientID;

select * from PatientDetails;

-- An index on the PatientID column for faster retrieval of patient information.
CREATE INDEX idx_PatientID ON Patients(PatientID);

--  Write necessary queries to register new user roles and personas
```

```
select * from UserRoles;
INSERT INTO UserRoles (RoleID, RoleName) VALUES(4, 'interns');

select * from UserRoles where RoleID =4 ;

select * from Users;
INSERT INTO Users (userID,Username, Password, RoleID) VALUES (4,'Intern_user',
'Intern_Pass', 4);

SELECT * FROM Users WHERE Username = 'Intern_user';

-- Write necessary queries to add to the list of diagnosis of the patient tagged by date.

SELECT * FROM Diagnosis;
INSERT INTO Diagnosis (DiagnosisID, PatientID, DiagnosisDate, DiagnosisDetails)
VALUES (3, 1, '2022-02-10', 'Allergy');

select * from Diagnosis where PatientID = 1 and DiagnosisID=3 ;

-- Write necessary queries to fetch required details of a particular patient.

SELECT * FROM Patients WHERE PatientID = 1 ;
select * from PatientDetails where PatientID = 1;

-- Write necessary queries to prepare a bill for the patient at the end of checkout.

select * from Bills where PatientID =1;

-- Create a view to store the total bill amount for each patient
CREATE VIEW TotalBillsView AS
SELECT PatientID, SUM(BillAmount) AS TotalBill
FROM Bills
GROUP BY PatientID;

select * from Bills;

-- Update the Patients table with the total bill amount
UPDATE Patients
SET InsuranceLimit = InsuranceLimit - IFNULL((SELECT TotalBill FROM TotalBillsView
WHERE TotalBillsView.PatientID = Patients.PatientID), 0)
WHERE PatientID = 1;

SET SQL_SAFE_UPDATES = 0;

UPDATE Patients
SET InsuranceLimit = InsuranceLimit - IFNULL((SELECT TotalBill FROM TotalBillsView
WHERE TotalBillsView.PatientID = Patients.PatientID), 0);

-- Check the updated Patients table
SELECT * FROM Patients;

select * from TotalBillsView where PatientID=1;
```

```
-- Write necessary queries to fetch and show data from various related tables (Joins)

SELECT Patients.PatientID, FirstName, LastName, DiagnosisDetails, BillAmount,
CheckoutDate
FROM Patients
LEFT JOIN Diagnosis ON Patients.PatientID = Diagnosis.PatientID
LEFT JOIN Bills ON Patients.PatientID = Bills.PatientID;

-- Optimize repeated read operations using views/materialized views.

SELECT * FROM Patients_view;
SELECT * FROM UserRoles_view;
SELECT * FROM Users_view;
SELECT * FROM DiagnosisView;
SELECT * FROM BillsView;
SELECT * FROM PatientDetails;
select * from TotalBillsView;

-- Optimize read operations using indexing wherever required. (Create index on at least 1 table)
```

```
SHOW INDEX FROM Patients;

-- Try optimizing bill generation using stored procedures.

DELIMITER //

CREATE PROCEDURE GenerateBill(IN patient_id INT, IN bill_amount DECIMAL(10,2))
BEGIN
   INSERT INTO Bills (PatientID, BillAmount, CheckoutDate)
   VALUES (patient_id, bill_amount, CURDATE());
END //

DELIMITER ;

SHOW CREATE PROCEDURE GenerateBill;

select * from Bills;

-- Update the remaining insurance limit for the patient
      UPDATE Patients
      SET InsuranceLimit = 4800
      WHERE PatientID = PatientID;

         select * from Patients;


-- Add necessary triggers to indicate when a patient's medical insurance limit has expired.
-- Create Trigger to Check Insurance Limit Expiry
DELIMITER //
CREATE TRIGGER CheckInsuranceExpiry AFTER INSERT ON Bills
FOR EACH ROW
BEGIN
   DECLARE remaining_limit DECIMAL(10, 2);

   -- Calculate the remaining insurance limit
   SET remaining_limit = (SELECT InsuranceLimit - NEW.BillAmount FROM Patients
WHERE PatientID = NEW.PatientID);

   -- Check if the remaining limit is below a threshold (e.g., $0.00)
   IF remaining_limit < 0 THEN
```

```
        -- Add actions or notifications for insurance limit expiry
        -- For example, update a flag in the Patients table or notify relevant personnel
        UPDATE Patients SET InsuranceExpired = 1 WHERE PatientID = NEW.PatientID;

        -- Add additional actions or notifications as needed
    END IF;
END //
DELIMITER ;
```