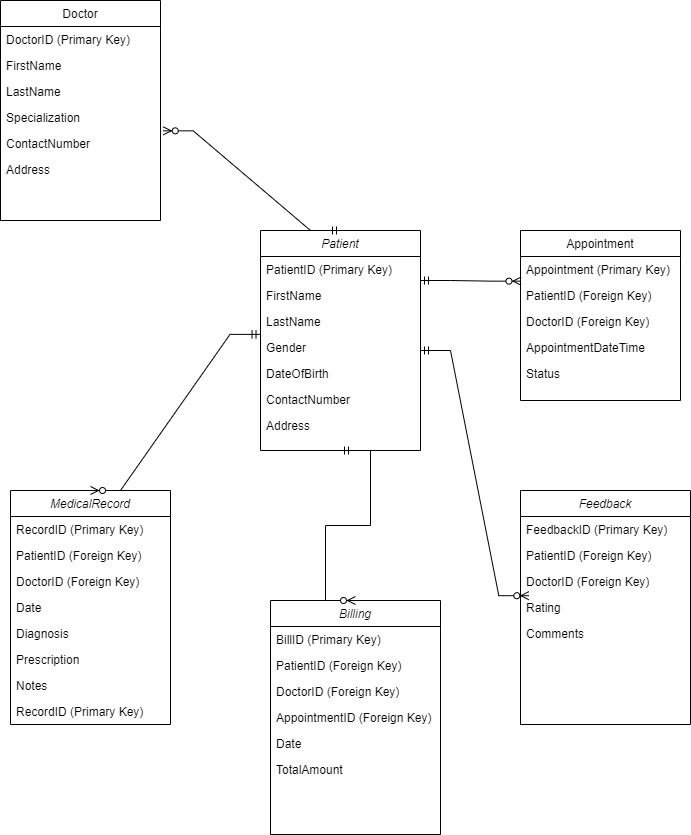
1. **Create a DB schema for Hospital Manangement System**

****

Patient Table:

PatientID (Primary Key)

FirstName

LastName

Gender

DateOfBirth

ContactNumber

Address

Doctor Table:

DoctorID (Primary Key)

FirstName

LastName

Gender

Specialization

ContactNumber

Address

Appointment Table:

AppointmentID (Primary Key)

PatientID (Foreign Key referencing Patient Table)

DoctorID (Foreign Key referencing Doctor Table)

AppointmentDateTime

Status (e.g., scheduled, canceled, completed)

MedicalRecord Table:

RecordID (Primary Key)

PatientID (Foreign Key referencing Patient Table)

DoctorID (Foreign Key referencing Doctor Table)

Date

Diagnosis

Prescription

Notes

Billing Table:

BillID (Primary Key)

PatientID (Foreign Key referencing Patient Table)

DoctorID (Foreign Key referencing Doctor Table)

AppointmentID (Foreign Key referencing Appointment Table)

Date

TotalAmount

User Table:

UserID (Primary Key)

UserName

Password

UserType (e.g., admin, doctor, receptionist)

Feedback Table:

FeedbackID (Primary Key)

PatientID (Foreign Key referencing Patient Table)

DoctorID (Foreign Key referencing Doctor Table)

Rating

Comments

1. **Define the schema along with a constraints indicating the relationship between entites.**

CREATE TABLE Patient (

PatientID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Gender CHAR(1),

DateOfBirth DATE,

ContactNumber VARCHAR(15),

Address VARCHAR(255)

);

CREATE TABLE Doctor (

DoctorID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Gender CHAR(1),

Specialization VARCHAR(100),

ContactNumber VARCHAR(15),

Address VARCHAR(255)

);

CREATE TABLE Appointment (

AppointmentID INT PRIMARY KEY,

PatientID INT,

DoctorID INT,

AppointmentDateTime DATETIME,

Status VARCHAR(20),

FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),

FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)

);

CREATE TABLE MedicalRecord (

RecordID INT PRIMARY KEY,

PatientID INT,

DoctorID INT,

Date DATE,

Diagnosis TEXT,

Prescription TEXT,

Notes TEXT,

FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),

FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)

);

CREATE TABLE Billing (

BillID INT PRIMARY KEY,

PatientID INT,

DoctorID INT,

AppointmentID INT,

Date DATE,

TotalAmount DECIMAL(10, 2),

FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),

FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID),

FOREIGN KEY (AppointmentID) REFERENCES Appointment(AppointmentID)

);

CREATE TABLE User (

UserID INT PRIMARY KEY,

UserName VARCHAR(50),

Password VARCHAR(255),

UserType VARCHAR(20)

);

CREATE TABLE Feedback (

FeedbackID INT PRIMARY KEY,

PatientID INT,

DoctorID INT,

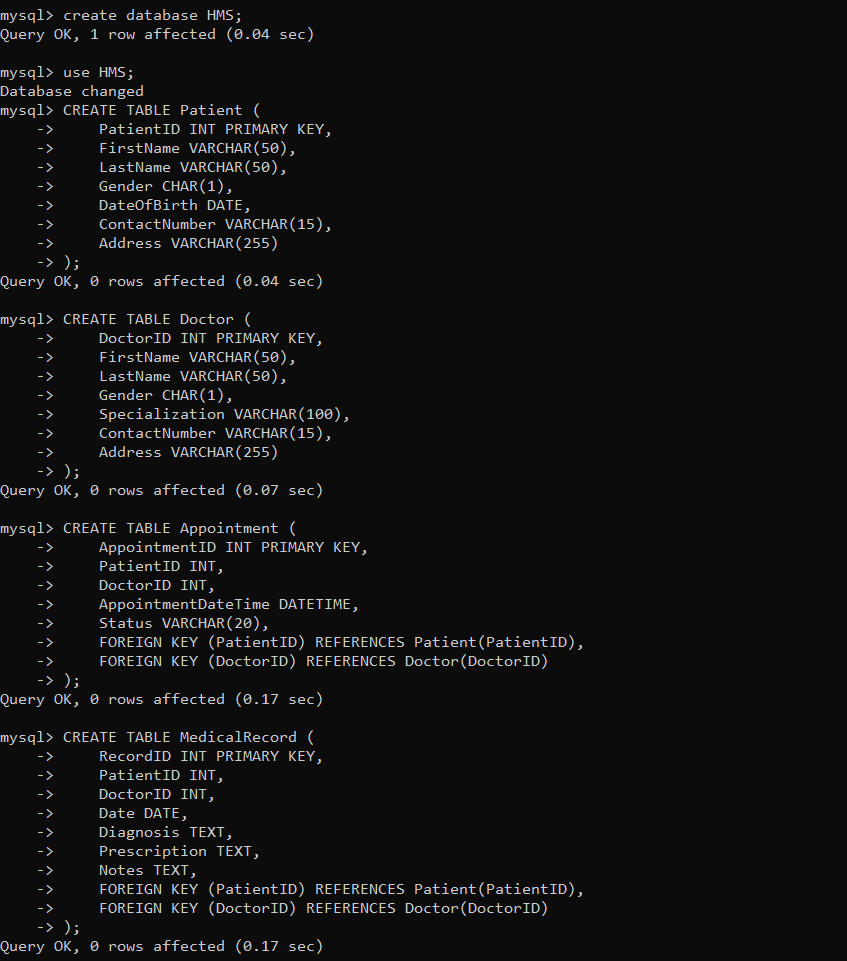
Rating INT,

Comments TEXT,

FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),

FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)

);



**Indicate the normalization form being used in schema define and why you closed to keep it that particular normal form.**

**First Normal Form (1NF):**

Each table has a primary key (e.g., PatientID, DoctorID, AppointmentID, etc.).

All columns contain atomic (indivisible) values.

**Second Normal Form (2NF):**

No partial dependencies are present, meaning that each non-prime attribute (column) is fully functionally dependent on the primary key.

For example, in the MedicalRecord table, attributes like Diagnosis, Prescription, and Notes are dependent on the entire primary key (RecordID), and not just part of it.

**Third Normal Form (3NF):**

No transitive dependencies are present. In other words, non-prime attributes are not dependent on other non-prime attributes.

For instance, in the Patient table, Address is dependent on PatientID and not on any other non-prime attributes.

The schema is designed to be in 3NF to minimize data redundancy and dependency, which helps in maintaining data integrity and consistency. By organizing the data into separate tables with clear relationships and avoiding transitive dependencies, the schema supports efficient data retrieval and updates.

**3.Make the use of database concept like views, indexing, relationships, stored procedure and triggers.**

**Views**

CREATE VIEW PatientAppointments AS

SELECT

P.PatientID,

P.FirstName,

P.LastName,

A.AppointmentID,

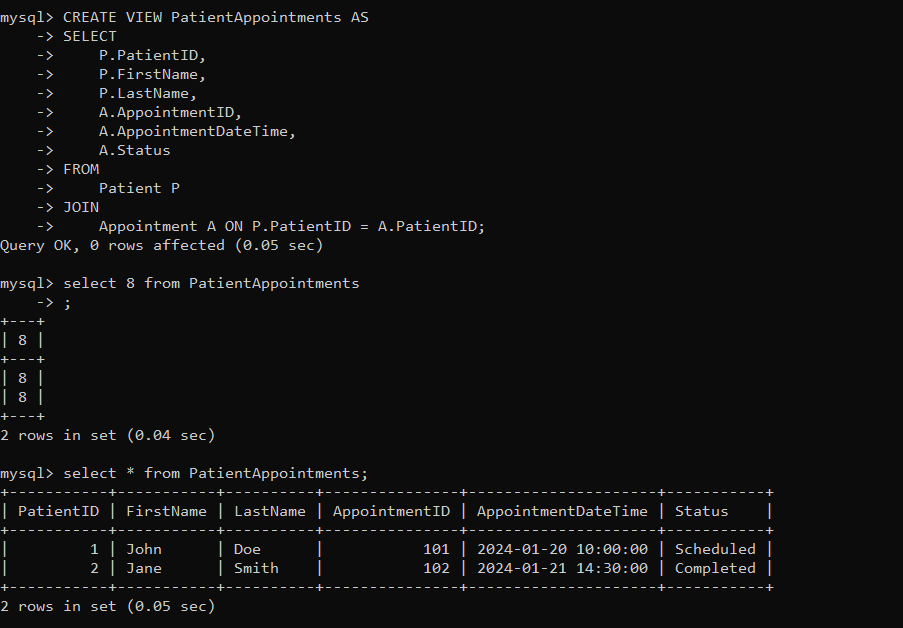
A.AppointmentDateTime,

A.Status

FROM

Patient P

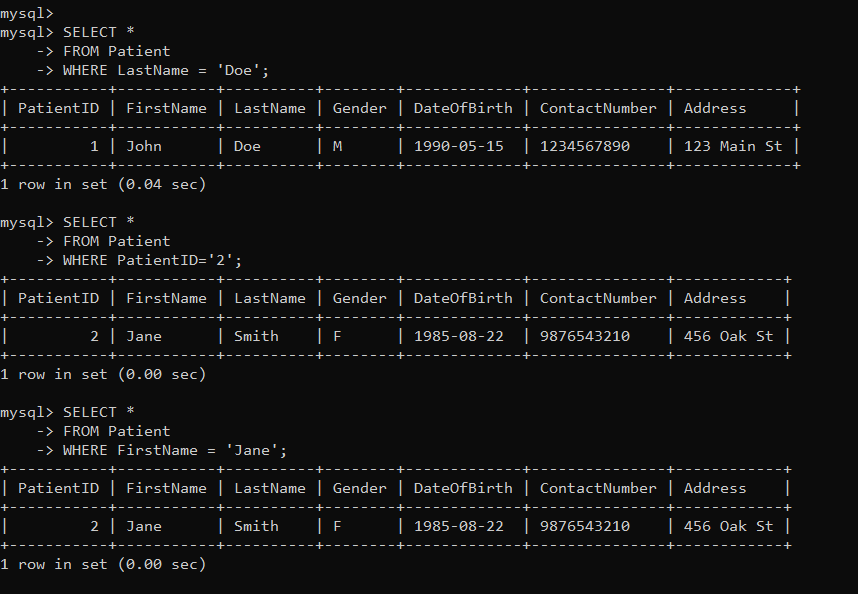
JOIN

Appointment A ON P.PatientID = A.PatientID; 

**Indexes:**

CREATE INDEX idx\_PatientLastName ON Patient(LastName);

CREATE INDEX idx\_DoctorSpecialization ON Doctor(Specialization);



**Relationships:**

ALTER TABLE Appointment

ADD CONSTRAINT FK\_Appointment\_Patient

FOREIGN KEY (PatientID) REFERENCES Patient(PatientID);

ALTER TABLE Appointment

ADD CONSTRAINT FK\_Appointment\_Doctor

FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID);

**Stored Procedures:**

CREATE PROCEDURE GetPatientAppointments(IN patientID INT)

BEGIN

SELECT

P.PatientID,

P.FirstName,

P.LastName,

A.AppointmentID,

A.AppointmentDateTime,

A.Status

FROM

Patient P

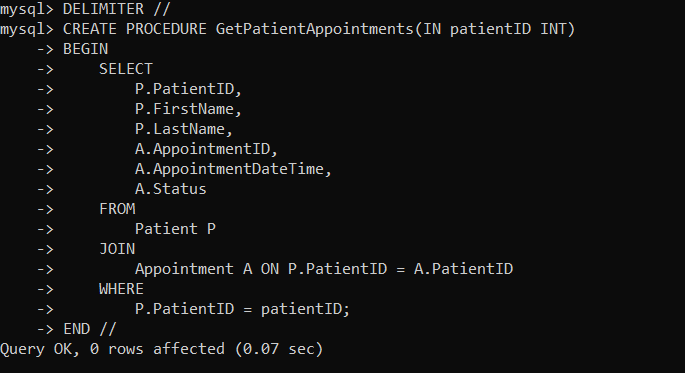
JOIN

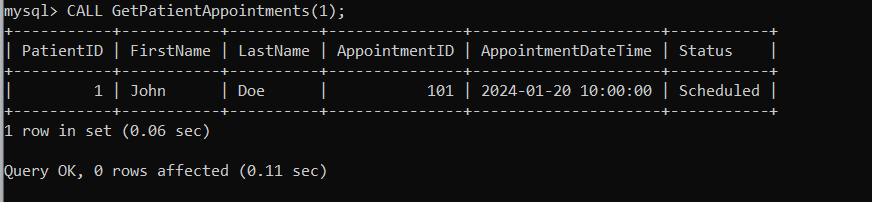
Appointment A ON P.PatientID = A.PatientID

WHERE

P.PatientID = patientID;

END





**Triggers:**

CREATE TRIGGER BeforeAppointmentInsert

BEFORE INSERT ON Appointment

FOR EACH ROW

BEGIN

IF NEW.AppointmentDateTime < NOW() THEN

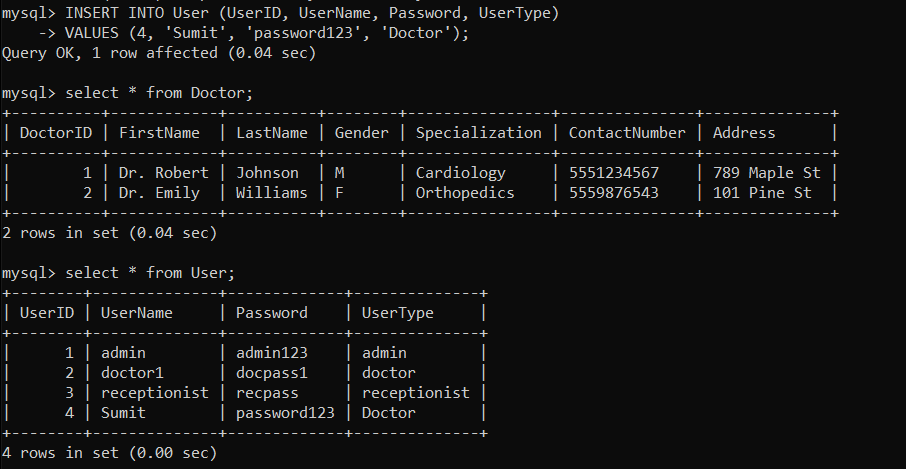
SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot schedule appointments in the past';

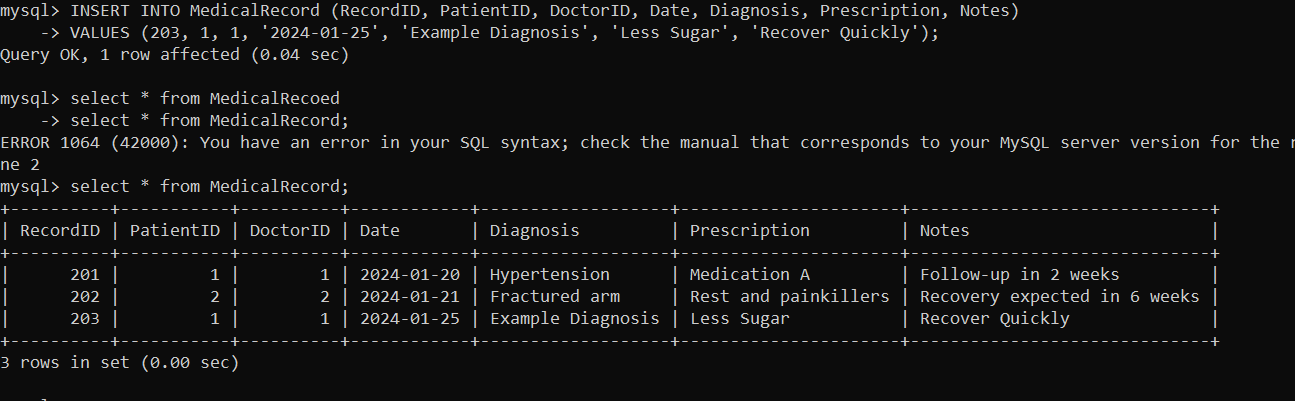
END IF;

END;

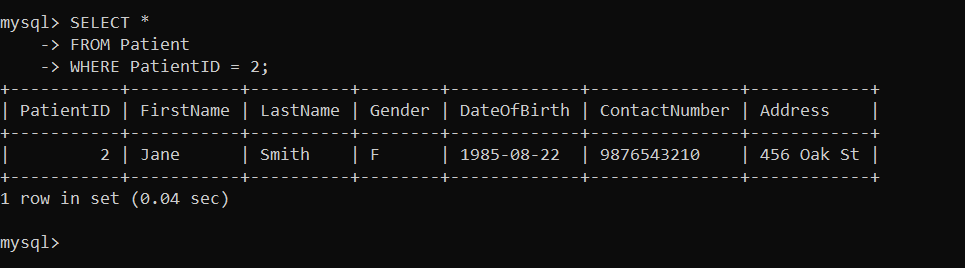
**Write a necessary query to register new users roles and permissions.**

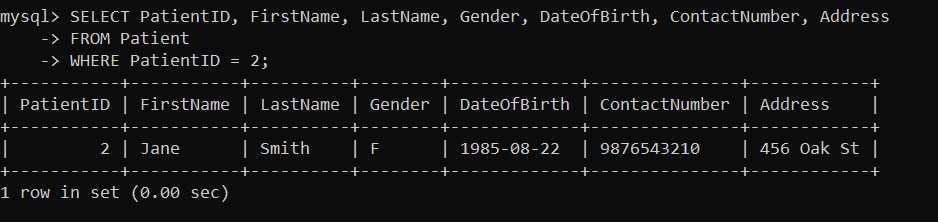


**Write necessary queries to add to the list of diagnosis of the patient tagged by date**

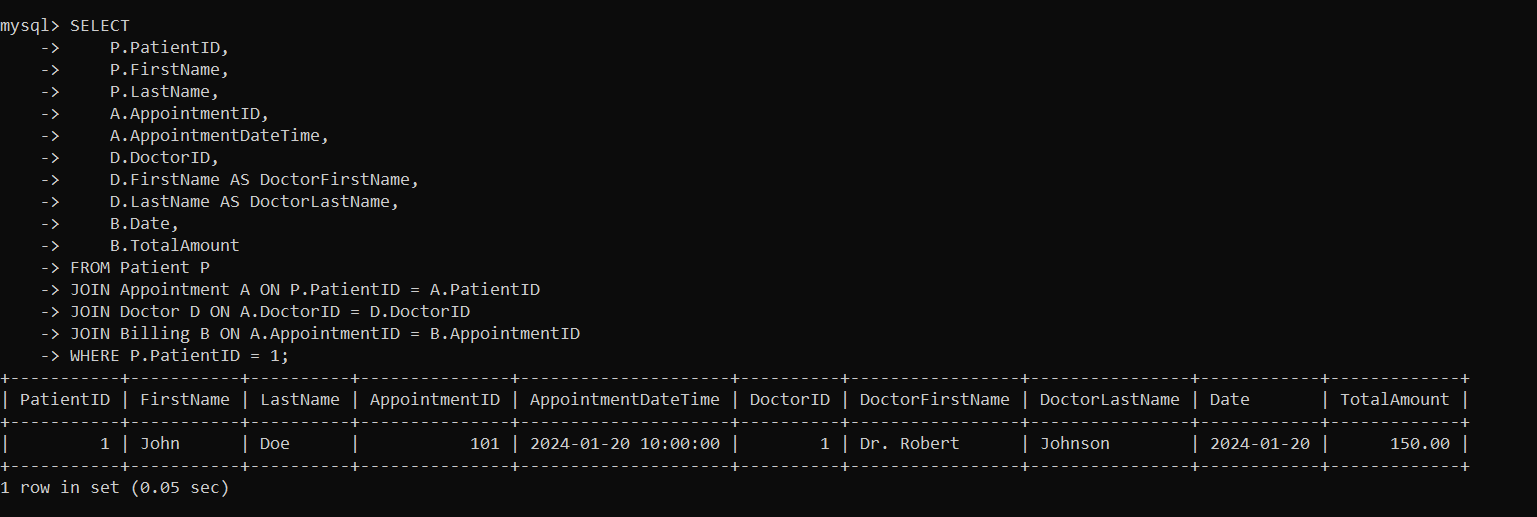


**Write necessary queris to fetch required details of a particular patient.**

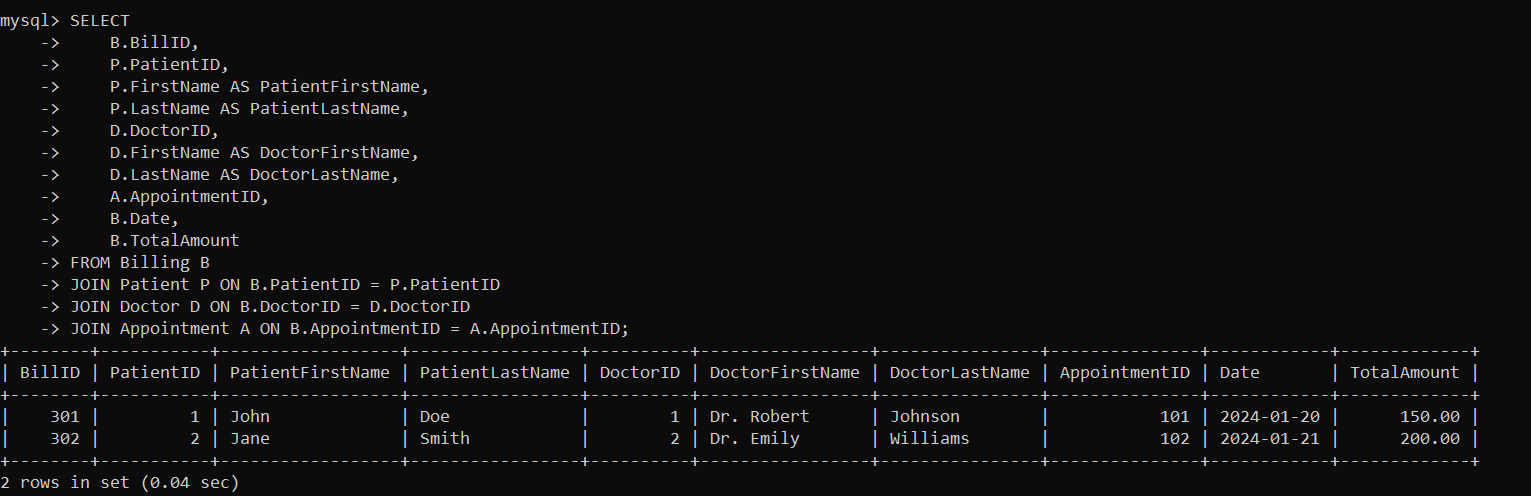




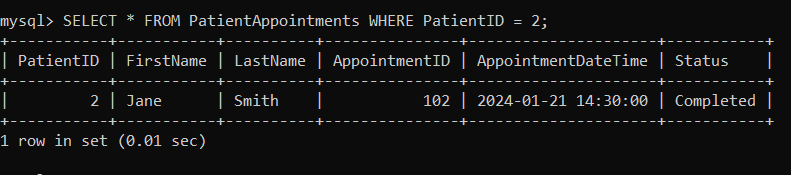
**Write necessary queries to prepare bill for patient at the end of checkout.**



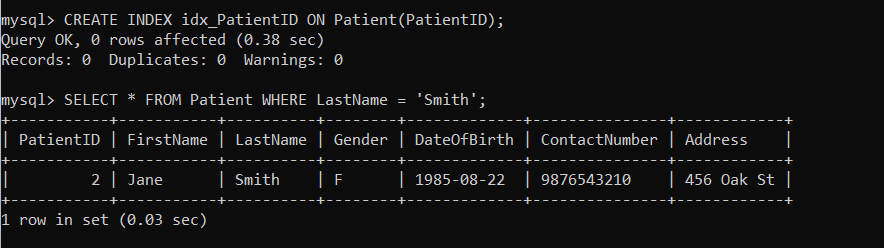
**Write necessary queries to fetch and show data from various related table(joins)**



**Optimized repeated read operations using views / materalized views.**



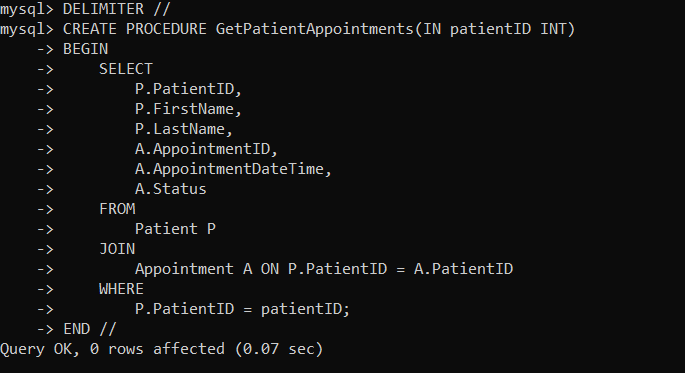
**Optimized read operations using indexing whenever required.**

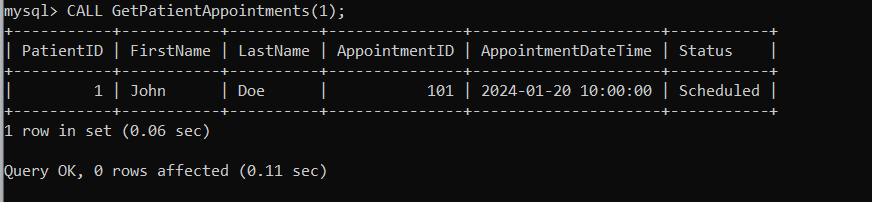


**Optimize repeated read operations using views/materialized views.**

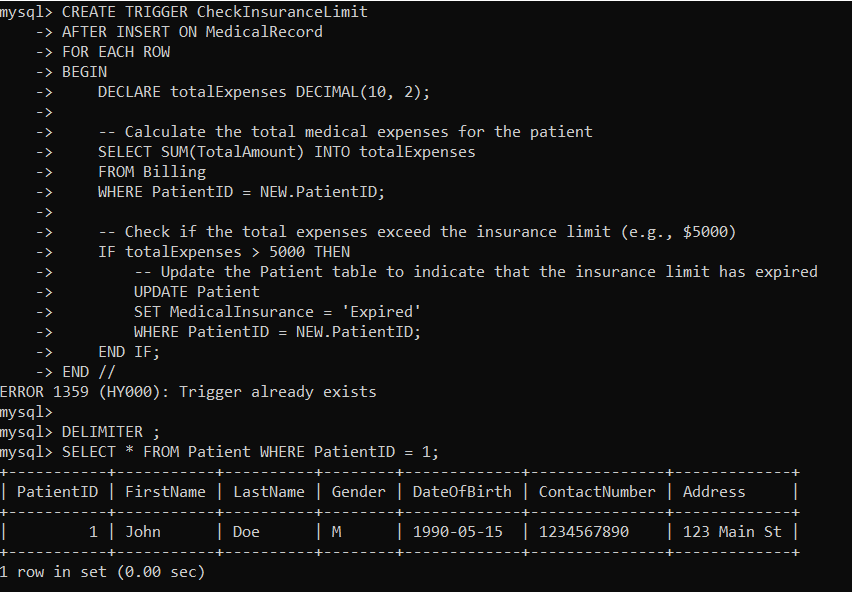


**Try optimizing bill generation using stored prodecures.**





**Add necessary triggers to indicate when patients medical insurance limit has expired**



**Question 2**

**What is Design Pattern**

**Design Pattern are the well proved solution of commonly occurring problem in software design. Suppose the developer is building the software every developer has its own approach. The already defined approach which is best for software is best for software is known as design pattern.**

**Types are**

**Creational Pattern are used to help for creating the Object.**

Factory Method

Abstract Method

Single Tone

Prototype

Builder

Object Pool

**Structural Design Pattern is used to structured the object and classes.**

Adapter Pattern

Bridge pattern

Composite

Proxy

Flyweight

**Behavioral Design Pattern is used if we have two objects than how the object will interact to each other.**

Iterator Pattern

Command Pattern

Observer Pattern

State Pattern

Template Pattern

Mediator Pattern

**What is rendering Pattern?**

**In web development, a rendering pattern refers to the way in which the HTML, CSS, and JavaScript code is all processed and rendered in a web application or website. Different rendering patterns are used to achieve different performance and user experience goals.**

**Client-Side Rendering (CSR):**

Use Case: Interactive Web Applications with Real-Time Data Updates

You are developing a single-page application (SPA) where user interactions and real-time updates are crucial, such as social media feeds, messaging apps, or collaborative tools.

Why CSR is Appropriate:

CSR allows for a highly interactive user experience as the client's browser can dynamically update content without refreshing the entire page.

Real-time data fetching and updates are well-suited for CSR, providing a responsive and dynamic user interface.

**Server-Side Rendering (SSR):**

Use Case: Content-Rich Websites with SEO Focus

You are working on an e-commerce platform or a content-heavy website where SEO is a top priority, and the content changes frequently.

Why SSR is Appropriate:

SSR enables the server to generate HTML content dynamically, providing a fast initial loading experience for users and ensuring better SEO by delivering pre-rendered HTML to search engine crawlers.

Dynamic content, personalized user experiences, and frequent updates can be efficiently handled on the server side.

**Static Site Generation (SSG):**

Use Case: Content-Centric Websites with Infrequent Updates

You are building a blog, documentation website, or a portfolio where the content is relatively static, and updates occur at predictable intervals.

Why SSG is Appropriate:

SSG generates HTML during the build process, resulting in fast loading times as users receive pre-rendered static HTML files.

SEO benefits are substantial, as search engines can easily index content present in the initial HTML response.