CSA0593 – DATABASE MANAGEMENT SYSTEM

PRAKALYA. PV

192311418

ASSIGNMENT- 5

**"Develop a database for a hospital outpatient clinic with patient appointments, doctors, treatments, and prescriptions."  
  
-Model tables for patients, appointments, doctors, treatment records, and prescription history.**

**-Write stored procedures for scheduling appointments, recording treatments, and issuing prescriptions.**

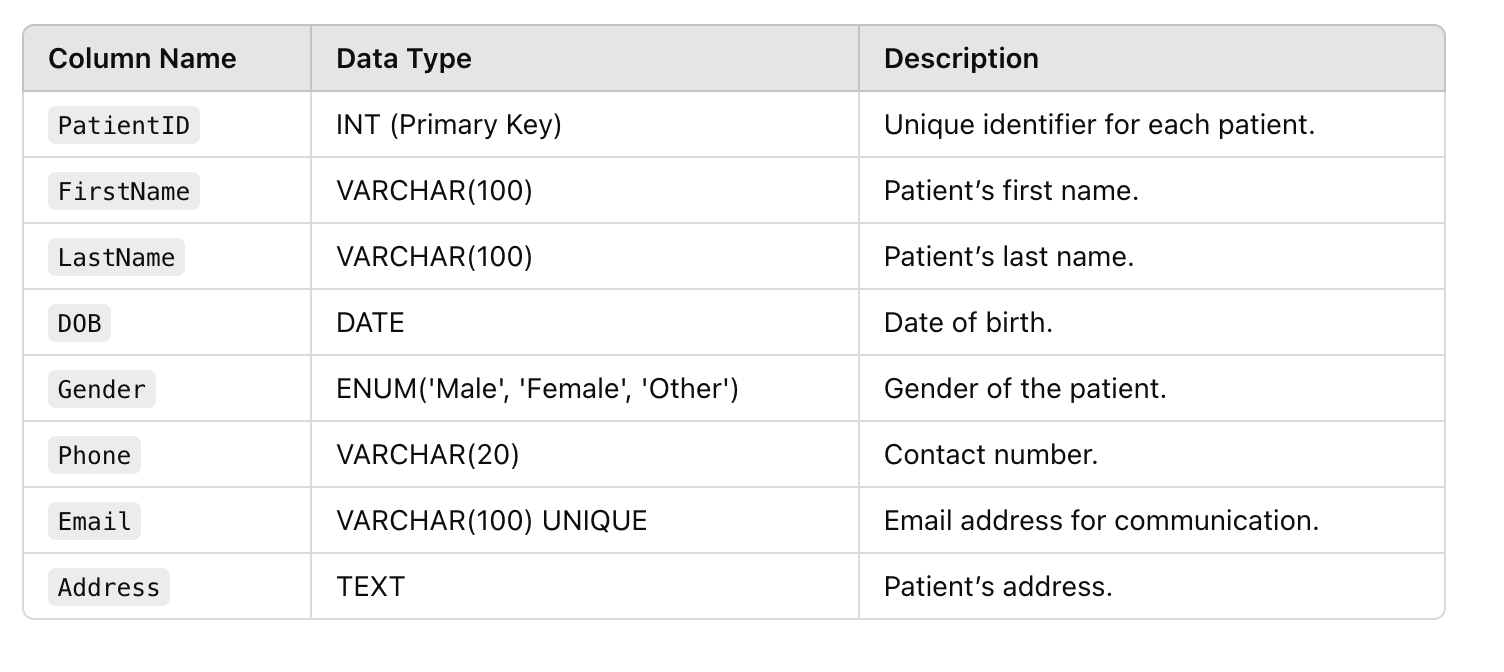
**-Implement triggers to update doctor schedules and patient prescription records.**

**-Write SQL queries to analyse appointment no-shows, common treatments, and prescription frequency by doctor.**

To develop a database for a hospital outpatient clinic, we need to design tables that manage patient appointments, doctors, treatments, and prescriptions. Here’s how we can structure the database:

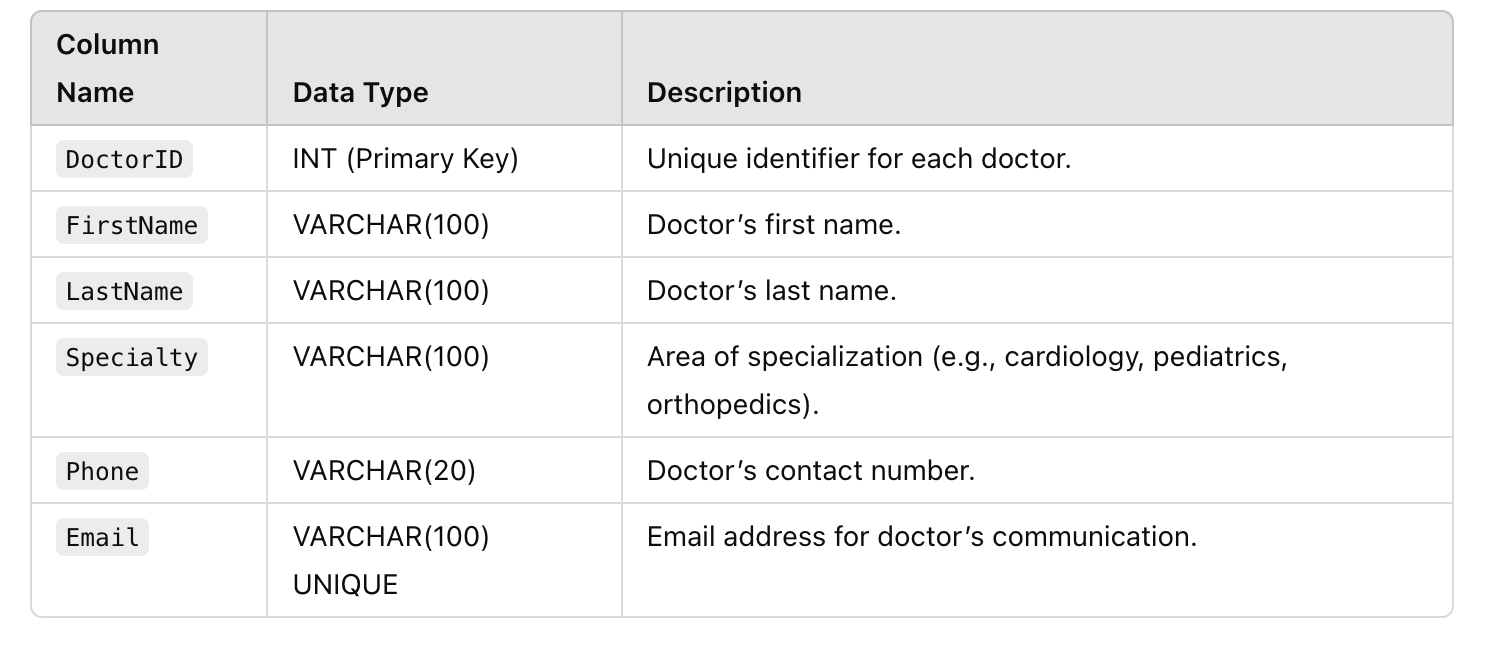
**1. Table: Patients**

This table stores information about the patients.

****

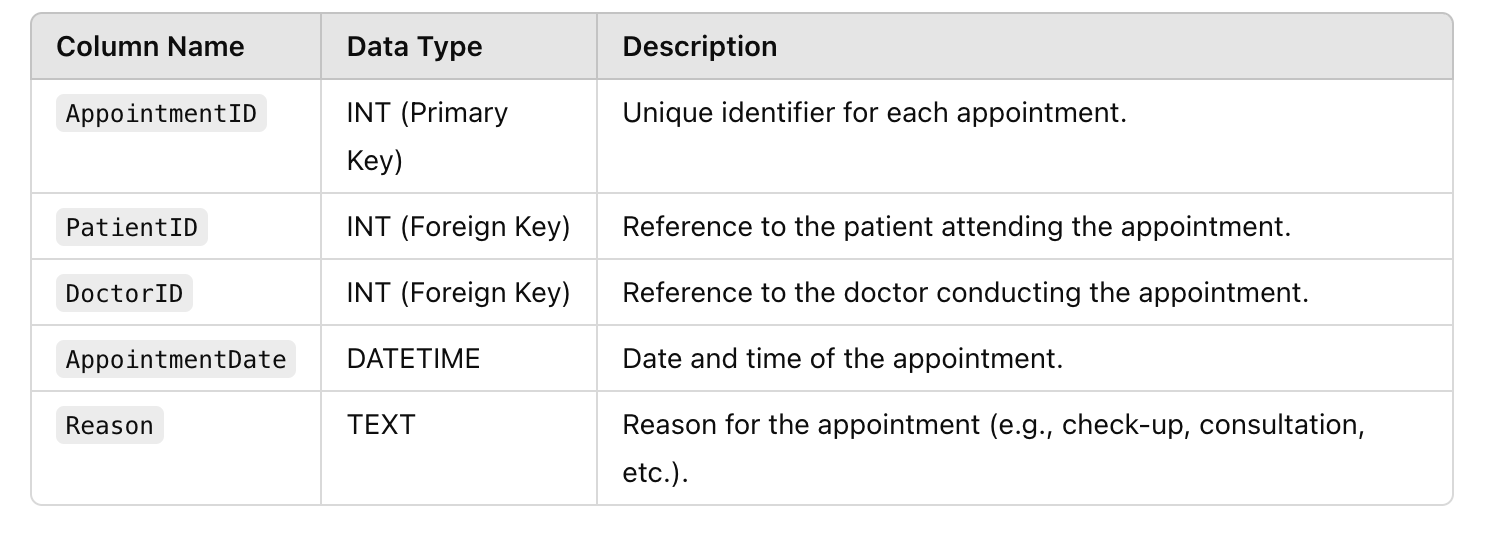
### ****2. Table: Doctors****

This table stores information about the doctors working at the clinic.

****

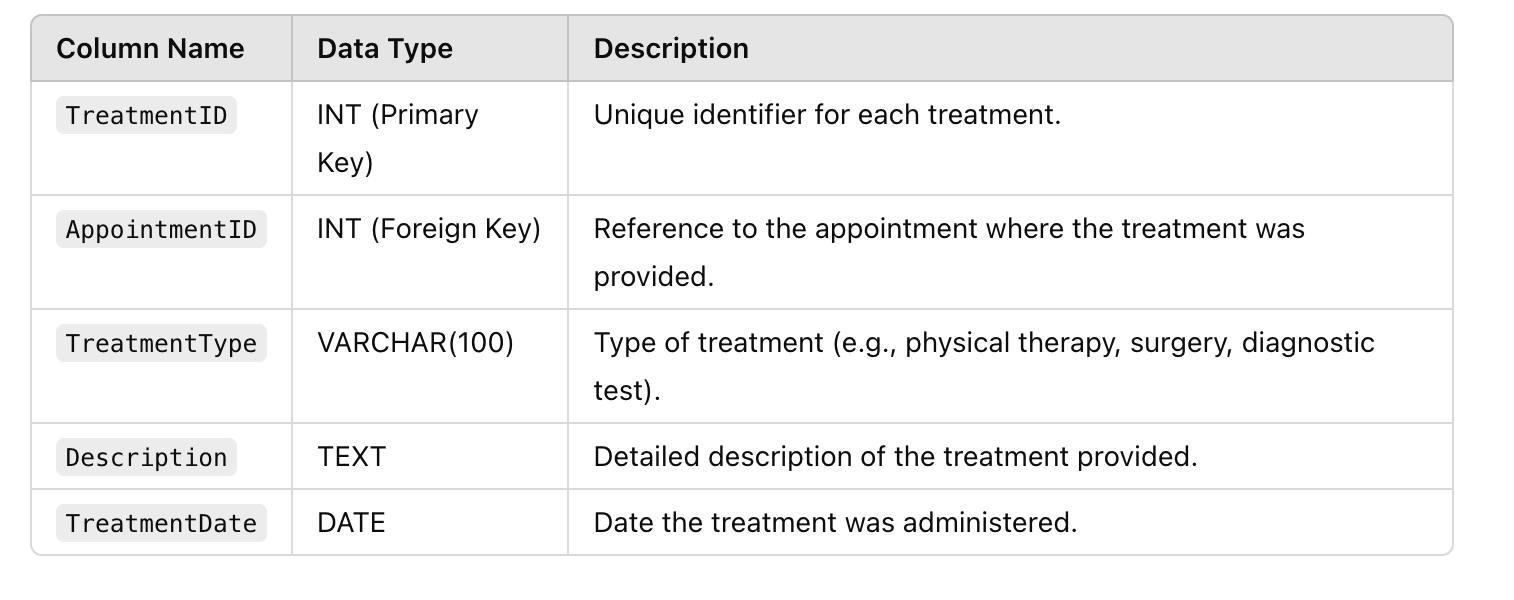
### ****3. Table: Appointments****

This table records patient appointments with doctors.

****

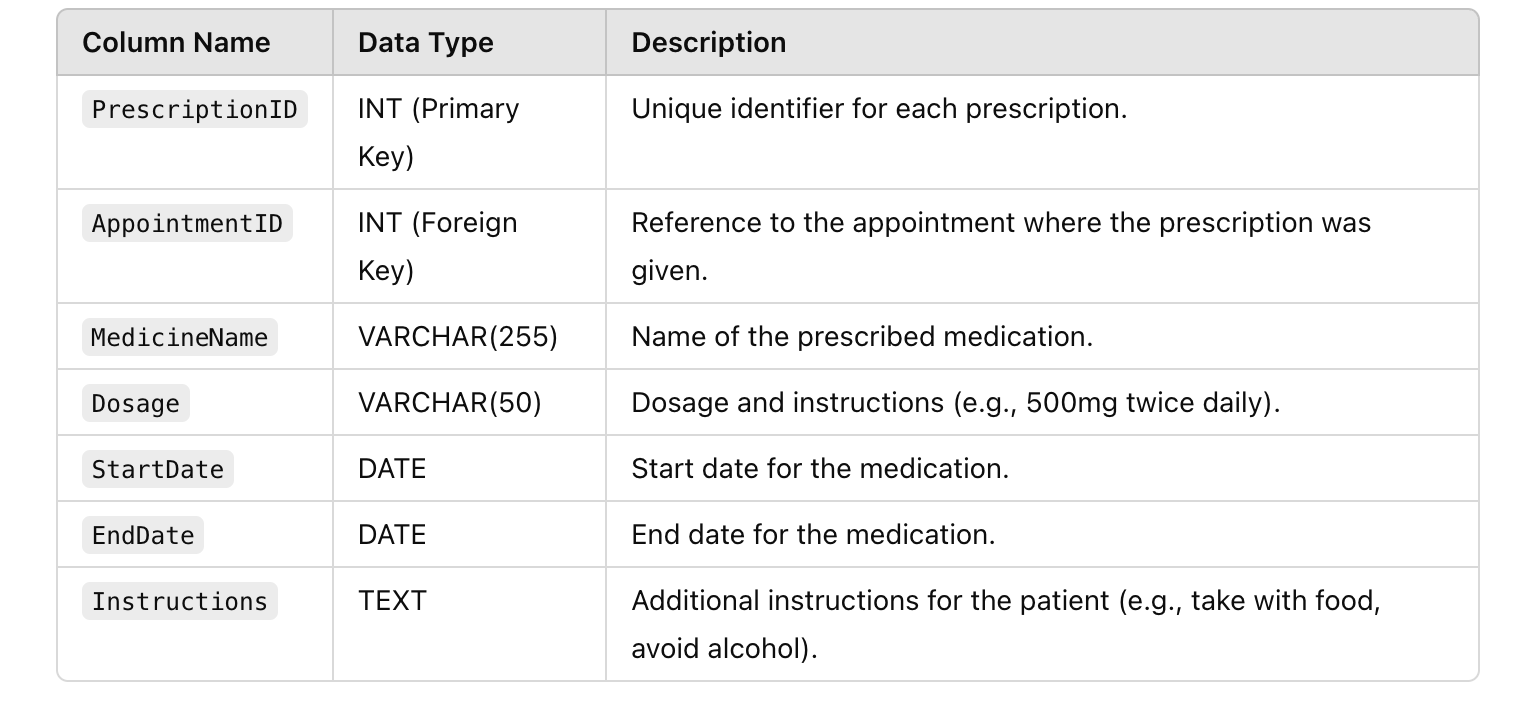
### ****4. Table: Treatments****

This table stores information about the treatments provided during appointments.

****

### ****5. Table: Prescriptions****

This table stores information about medications prescribed to patients.

****

### ****Relationships****

1. **Patients ↔ Appointments**:
   * A patient can have multiple appointments, but each appointment is linked to a single patient.
2. **Doctors ↔ Appointments**:
   * A doctor can have multiple appointments with different patients.
3. **Appointments ↔ Treatments**:
   * An appointment can have multiple treatments associated with it.
4. **Appointments ↔ Prescriptions**:
   * Each appointment can have one or more prescriptions, as multiple medications might be prescribed during a single visit.

**-Write stored procedures for scheduling appointments, recording treatments, and issuing prescriptions:**

**1. Stored Procedure for Scheduling an Appointment**

This procedure schedules an appointment for a patient with a doctor.

DELIMITER $$

CREATE PROCEDURE ScheduleAppointment(

IN p\_PatientID INT,

IN p\_DoctorID INT,

IN p\_AppointmentDate DATETIME,

IN p\_Reason TEXT

)

BEGIN

INSERT INTO Appointments (PatientID, DoctorID, AppointmentDate, Reason)

VALUES (p\_PatientID, p\_DoctorID, p\_AppointmentDate, p\_Reason);

created appointment

SELECT LAST\_INSERT\_ID() AS NewAppointmentID;

END $$

DELIMITER ;

**Usage:**

CALL ScheduleAppointment(1, 3, '2024-12-10 10:00:00', 'Routine check-up');

### ****2. Stored Procedure for Recording Treatments****

This procedure records the treatments administered during a specific appointment.

DELIMITER $$

CREATE PROCEDURE RecordTreatment(

IN p\_AppointmentID INT,

IN p\_TreatmentType VARCHAR(100),

IN p\_Description TEXT,

IN p\_TreatmentDate DATE

)

BEGIN

INSERT INTO TreatmentRecords (AppointmentID, TreatmentType, Description, TreatmentDate)

VALUES (p\_AppointmentID, p\_TreatmentType, p\_Description, p\_TreatmentDate);

record

SELECT LAST\_INSERT\_ID() AS NewTreatmentID;

END $$

DELIMITER ;

**Usage:**

CALL RecordTreatment(101, 'Physical Therapy', 'Therapeutic exercise for knee rehabilitation', '2024-12-10');

### ****3. Stored Procedure for Issuing a Prescription****

This procedure issues a prescription for a patient based on a specific appointment.

DELIMITER $$

CREATE PROCEDURE IssuePrescription(

IN p\_AppointmentID INT,

IN p\_MedicineName VARCHAR(255),

IN p\_Dosage VARCHAR(50),

IN p\_StartDate DATE,

IN p\_EndDate DATE,

IN p\_Instructions TEXT

)

BEGIN

INSERT INTO PrescriptionHistory (AppointmentID, MedicineName, Dosage, StartDate, EndDate, Instructions)

VALUES (p\_AppointmentID, p\_MedicineName, p\_Dosage, p\_StartDate, p\_EndDate, p\_Instructions);

prescription record

SELECT LAST\_INSERT\_ID() AS NewPrescriptionID;

END $$

DELIMITER ;

**Usage:**

CALL IssuePrescription(101, 'Ibuprofen', '500mg twice daily', '2024-12-10', '2024-12-20', 'Take with food');

**-Write SQL queries to analyse appointment no-shows, common treatments, and prescription frequency by doctor:**

**1. SQL Query to Analyze Appointment No-Shows**

An appointment no-show occurs when a patient does not show up for the scheduled appointment. You can identify no-shows by checking if the appointment date has passed and the appointment is not marked as completed or attended.

Let's assume we have an AppointmentStatus column in the Appointments table that indicates whether the patient attended the appointment (e.g., values like 'Completed', 'No Show', 'Cancelled').

**SQL Query to Find No-Shows**

SELECT

p.PatientID,

p.FirstName,

p.LastName,

a.AppointmentID,

a.AppointmentDate,

a.Reason,

a.AppointmentStatus

FROM

Appointments a

JOIN

Patients p ON a.PatientID = p.PatientID

WHERE

a.AppointmentDate < CURRENT\_TIMESTAMP -- Past appointments

AND a.AppointmentStatus = 'No Show' -- Status indicating no show

ORDER BY

a.AppointmentDate DESC;

#### **Explanation:**

* This query retrieves all patients who missed their past appointments (no-shows).
* It includes patient details (name), the appointment ID, the date and reason for the appointment, and the status of the appointment.

### ****2. SQL Query to Find Common Treatments****

To find the most common treatments administered, you can query the TreatmentRecords table. This query groups the treatments by type and counts how often each treatment has been administered.

#### **SQL Query to Find Common Treatments**

SELECT

TreatmentType,

COUNT(\*) AS TreatmentCount

FROM

TreatmentRecords

GROUP BY

TreatmentType

ORDER BY

TreatmentCount DESC;

#### **Explanation:**

* This query counts the number of times each treatment type has been recorded in the TreatmentRecords table.
* It groups the results by TreatmentType and orders them in descending order based on the treatment count.

### ****3. SQL Query to Find Prescription Frequency by Doctor****

This query analyzes the frequency of prescriptions issued by each doctor. It links the PrescriptionHistory table with the Appointments table to associate prescriptions with specific doctors.

#### **SQL Query to Find Prescription Frequency by Doctor**

SELECT

d.DoctorID,

d.FirstName AS DoctorFirstName,

d.LastName AS DoctorLastName,

COUNT(\*) AS PrescriptionCount

FROM

PrescriptionHistory ph

JOIN

Appointments a ON ph.AppointmentID = a.AppointmentID

JOIN

Doctors d ON a.DoctorID = d.DoctorID

GROUP BY

d.DoctorID, d.FirstName, d.LastName

ORDER BY

PrescriptionCount DESC;

#### **Explanation:**

* This query retrieves the frequency of prescriptions issued by each doctor.
* It joins the PrescriptionHistory and Appointments tables to get prescription data linked to doctors and aggregates it by doctor.
* It counts the number of prescriptions issued by each doctor and sorts the results in descending order of frequency.

**-Implement triggers to update doctor schedules and patient prescription records:**

**1. Trigger to Update Doctor's Schedule After an Appointment is Scheduled**

This trigger updates the doctor's schedule when a new appointment is added. It checks the availability of the doctor and updates the schedule to reflect the appointment time.

**Trigger to Update Doctor's Schedule on New Appointment**

Assuming there is a DoctorSchedules table that tracks a doctor's available hours:

DELIMITER $$

CREATE TRIGGER UpdateDoctorScheduleOnAppointment

AFTER INSERT ON Appointments

FOR EACH ROW

BEGIN

DECLARE available\_count INT;

SELECT COUNT(\*) INTO available\_count

FROM DoctorSchedules

WHERE DoctorID = NEW.DoctorID

AND ScheduleDate = DATE(NEW.AppointmentDate)

AND StartTime <= TIME(NEW.AppointmentDate)

AND EndTime >= TIME(NEW.AppointmentDate)

IF available\_count > 0 THEN

UPDATE DoctorSchedules

SET Available = 0 -- Mark as unavailable during the appointment time

WHERE DoctorID = NEW.DoctorID

AND ScheduleDate = DATE(NEW.AppointmentDate)

AND StartTime <= TIME(NEW.AppointmentDate)

AND EndTime >= TIME(NEW.AppointmentDate);

END IF;

END $$

DELIMITER ;

#### **Explanation:**

* After a new appointment is inserted into the Appointments table, the trigger checks the DoctorSchedules table to see if the doctor is available at the scheduled time.
* If the doctor is available, the trigger updates the schedule to mark that time slot as unavailable for that doctor.

#### **Assumptions:**

* DoctorSchedules table tracks the doctor's working hours and availability with DoctorID, ScheduleDate, StartTime, EndTime, and an Available flag (0 for unavailable, 1 for available).

### ****2. Trigger to Update Patient's Prescription Record****

This trigger updates the prescription records whenever a new prescription is issued for a patient. It can be used to track the prescription issuance in real-time and log changes.

#### **Trigger to Update Prescription Record on New Prescription**

DELIMITER $$

CREATE TRIGGER UpdatePrescriptionRecord

AFTER INSERT ON PrescriptionHistory

FOR EACH ROW

BEGIN

INSERT INTO PatientPrescriptionHistory (PatientID, MedicineName, Dosage, StartDate, EndDate, Instructions, PrescriptionDate)

VALUES (NEW.PatientID, NEW.MedicineName, NEW.Dosage, NEW.StartDate, NEW.EndDate, NEW.Instructions, CURRENT\_TIMESTAMP);

UPDATE Patients

SET TreatmentStatus = 'Under Treatment'

WHERE PatientID = NEW.PatientID;

END $$

DELIMITER ;

#### **Explanation:**

* After a new prescription is issued (when a new row is inserted into the PrescriptionHistory table), the trigger inserts a corresponding record into the PatientPrescriptionHistory table for logging purposes (this is optional but useful for audit trails).
* Additionally, the trigger can update the patient's treatment status to "Under Treatment" or another status as necessary.

#### **Assumptions:**

* PatientPrescriptionHistory table tracks the history of prescriptions issued to patients, with columns like PatientID, MedicineName, Dosage, StartDate, EndDate, Instructions, and PrescriptionDate.
* The Patients table has a TreatmentStatus column that is updated to reflect the patient's ongoing treatment.

**CONCLUSION:**

In short, the hospital outpatient clinic database design efficiently manages patient appointments, treatments, and prescriptions. Stored procedures automate key processes like scheduling and prescribing, while triggers ensure real-time updates to doctor schedules and prescription records. SQL queries provide valuable insights into clinic operations, helping optimize resources and improve patient care. This system enhances operational efficiency, data accuracy, and decision-making within the clinic.