ASSIGNMENT - 2

DATABASE MANAGEMENT SYSTEM

SIDDARTH.P

192324264

Healthcare Management System Database Design

This database system is designed to manage patients, doctors, appointments, and prescriptions. Below is a detailed overview:

---

1. Database Tables Design

Table: Patients

Stores patient information for identification and medical history.

Column	Data Type	Description
patient_id	INT (PK)	Unique identifier for each patient
first_name	VARCHAR(50)	Patient's first name
last_name	VARCHAR(50)	Patient's last name
date_of_birth	DATE	Date of birth
phone_number	VARCHAR(15)	Contact number
email	VARCHAR(100)	Email address
address	VARCHAR(255)	Residential address

Table: Doctors

Stores doctor information and specialties.

Column	Data Type	Description
doctor_id	INT (PK)	Unique identifier for each doctor
first_name	VARCHAR(50)	Doctor's first name
last_name	VARCHAR(50)	Doctor's last name
specialization	VARCHAR(100)	Doctor's area of specialization
phone_number	VARCHAR(15)	Contact number
email	VARCHAR(100)	Email address

Table: Appointments

Tracks appointments between patients and doctors.

Column	Data Type	Description
appointment_id	INT (PK)	Unique identifier for each appointment
patient_id	INT (FK)	References Patients(patient_id)
doctor_id	INT (FK)	References Doctors(doctor_id)
appointment_date	DATETIME	Date and time of the appointment
status	VARCHAR(20)	Status of the appointment (e.g., Scheduled, Completed, Canceled)

Table: Prescriptions

Stores prescriptions issued by doctors to patients.

Column	Data Type	Description
prescription_id	INT (PK)	Unique identifier for each prescription
appointment_id	INT (FK)	References Appointments(appointment_id)
medicine_name	VARCHAR(100)	Name of the prescribed medicine
dosage	VARCHAR(50)	Dosage instructions
notes	TEXT	Additional notes

2. Stored Procedures

a. Book Appointment

Creates a new appointment for a patient with a doctor.

```
CREATE PROCEDURE BookAppointment(
  IN patientId INT,
  IN doctorId INT,
  IN appointmentDate DATETIME
)
BEGIN
  INSERT INTO Appointments (patient_id, doctor_id, appointment_date, status)
  VALUES (patientId, doctorId, appointmentDate, 'Scheduled');
END;
```

b. Update Appointment

Updates the details of an existing appointment.

```
CREATE PROCEDURE UpdateAppointment(
  IN appointmentId INT,
  IN newDate DATETIME,
  IN newStatus VARCHAR(20)
)
BEGIN
  UPDATE Appointments
  SET appointment_date = newDate,
    status = newStatus
  WHERE appointment_id = appointmentId;
END;
```

c. Cancel Appointment

Cancels an existing appointment and updates its status.

```
CREATE PROCEDURE CancelAppointment(IN appointmentId INT)
BEGIN
  UPDATE Appointments
  SET status = 'Canceled'
  WHERE appointment_id = appointmentId;
END;
```

3. Triggers

a. Update Doctor Availability

Automatically updates the availability status of a doctor when a new appointment is booked.

```
CREATE TRIGGER AfterAppointmentInsert
AFTER INSERT ON Appointments
FOR EACH ROW
BEGIN
  UPDATE Doctors
  SET phone_number = CONCAT('Busy - ', phone_number)
  WHERE doctor_id = NEW.doctor_id;
END;
```

b. Update Patient History

Automatically records a completed appointment in the patient's history.

```
CREATE TRIGGER AfterAppointmentUpdate
AFTER UPDATE ON Appointments
FOR EACH ROW
```

```
BEGIN
  IF NEW.status = 'Completed' THEN
    INSERT INTO Prescriptions (appointment_id, medicine_name, dosage, notes)
    VALUES (NEW.appointment_id, 'Follow-up Needed', 'N/A', 'Completed checkup');
  END IF;
END;
```

4. SQL Queries

a. Analyze Doctor Availability

Lists the availability of doctors based on their scheduled appointments.

```
SELECT d.doctor_id,
    d.first_name,
    d.last_name,
    COUNT(a.appointment_id) AS scheduled_appointments
FROM Doctors d
LEFT JOIN Appointments a ON d.doctor_id = a.doctor_id AND a.status = 'Scheduled'
GROUP BY d.doctor_id, d.first_name, d.last_name;
```

b. Patient Visits Report

Counts the total number of visits by each patient.

```
SELECT p.patient_id,
    p.first_name,
    p.last_name,
    COUNT(a.appointment_id) AS total_visits
FROM Patients p
LEFT JOIN Appointments a ON p.patient_id = a.patient_id
WHERE a.status = 'Completed'
```

```
GROUP BY p.patient_id, p.first_name, p.last_name

ORDER BY total_visits DESC;
```

Conclusion:

This database structure, combined with stored procedures and triggers, ensures a streamlined approach to managing appointments, maintaining doctor availability, and analyzing patient interactions effectively.