**Q.1 Create a DB Schema for Hospital Management System.**

Tables:

|  |
| --- |
| Patients |
| patient\_id (INT, Primary Key) |
| patient\_name (VARCHAR) |
| date\_of\_birth (DATE) |
| gender (CHAR) |
| address (VARCHAR) |
| phone\_number (VARCHAR) |
| email (VARCHAR) |

|  |
| --- |
| Appointments |
| appointment\_id (INT, Primary Key) |
| patient\_id (INT, Foreign Key referencing Patients table) |
| doctor\_id (INT,Foreign Key referencing Doctors table) |
| appointment\_date (DATE) |
| appointment\_time (VARCHAR) |
| status (e.g., pending, confirmed, canceled)(VARCHAR) |
| remarks (VAECHAR) |



|  |
| --- |
| MedicalRecords |
| record\_id (INT, Primary Key) |
| patient\_id (INT, Foreign Key referencing Patients table) |
| doctor\_id (INT, Foreign Key referencing Doctors table) |
| appointment\_id (INT, Foreign Key referencing Appointments table) |
| diagnosis |
| Treatment |
| prescription |
| date (DATE) |



|  |
| --- |
| Staff |
| staff\_name (VARCHAR) |
| department\_id (INT, Foreign Key referencing Departments table) |
| position (VARCHAR) |
| phone\_number (VARCHAR) |
| email (VARCHAR) |



|  |
| --- |
| Doctors |
| doctor\_id (INT, Primary Key) |
| doctor\_name (VARCHAR) |
| specialization (VARCHAR) |
| phone\_number (VARCHAR) |
| email (VARCHAR) |



|  |
| --- |
| Departments |
| department\_id (INT, Primary Key) |
| department\_name (VARCHAR) |
| description (VHARCHAR) |
| head\_doctor\_id (INT, Foreign Key referencing Doctors table) |



Relationships:

* Each patient can have multiple medical records and appointments.
* Each doctor can have multiple appointments and medical records.
* Each appointment is associated with one patient and one doctor.
* Each department can have multiple staff members.
* Each department is headed by one doctor.

This schema provides a foundation for managing patients, doctors, appointments, departments, staff, and medical records within a hospital management system. Depending on specific requirements

**Q.2 Define the schema along with the constraints indicating the relationships between the entities.**

CREATE TABLE Patients (

patient\_id INT PRIMARY KEY,

patient\_name VARCHAR(100),

date\_of\_birth DATE,

gender CHAR(1),

address VARCHAR(255),

phone\_number VARCHAR(20),

email VARCHAR(100)

);

CREATE TABLE Doctors (

doctor\_id INT PRIMARY KEY,

doctor\_name VARCHAR(100),

specialization VARCHAR(100),

phone\_number VARCHAR(20),

email VARCHAR(100)

);

CREATE TABLE Appointments (

appointment\_id INT PRIMARY KEY,

patient\_id INT,

doctor\_id INT,

appointment\_date DATE,

appointment\_time TIME,

status VARCHAR(20),

remarks TEXT,

FOREIGN KEY (patient\_id) REFERENCES Patients(patient\_id),

FOREIGN KEY (doctor\_id) REFERENCES Doctors(doctor\_id)

);

CREATE TABLE Departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(100),

description TEXT,

head\_doctor\_id INT,

FOREIGN KEY (head\_doctor\_id) REFERENCES Doctors(doctor\_id)

);

CREATE TABLE Staff (

staff\_id INT PRIMARY KEY,

staff\_name VARCHAR(100),

department\_id INT,

position VARCHAR(100),

phone\_number VARCHAR(20),

email VARCHAR(100),

FOREIGN KEY (department\_id) REFERENCES Departments(department\_id)

);

CREATE TABLE MedicalRecords (

record\_id INT PRIMARY KEY,

patient\_id INT,

doctor\_id INT,

appointment\_id INT,

diagnosis TEXT,

treatment TEXT,

prescription TEXT,

date DATE,

FOREIGN KEY (patient\_id) REFERENCES Patients(patient\_id),

FOREIGN KEY (doctor\_id) REFERENCES Doctors(doctor\_id),

FOREIGN KEY (appointment\_id) REFERENCES Appointments(appointment\_id));

**Q.3 Be sure to make use of the database concepts like Views, Relationships, Indexing, Stored Procedure and triggers.**

**Indexing: Indexes are created on commonly queried columns to improve query performance.**

CREATE INDEX idx\_patient\_id ON Appointments(patient\_id);

CREATE INDEX idx\_doctor\_id ON Appointments(doctor\_id);

CREATE INDEX idx\_appointment\_date ON Appointments(appointment\_date);

CREATE INDEX idx\_department\_id ON Staff(department\_id);

CREATE INDEX idx\_record\_id ON MedicalRecords(patient\_id, doctor\_id, appointment\_id);

**Views: Views are created to provide simplified access to frequently needed**

data.CREATE VIEW PatientAppointments AS

SELECT Patients.patient\_id, Patients.patient\_name, Appointments.appointment\_id, Appointments.appointment\_date, Appointments.appointment\_time

FROM Patients

INNER JOIN Appointments ON Patients.patient\_id = Appointments.patient\_id;

CREATE VIEW DoctorAppointments AS

SELECT Doctors.doctor\_id, Doctors.doctor\_name, Appointments.appointment\_id, Appointments.appointment\_date, Appointments.appointment\_time

FROM Doctors

INNER JOIN Appointments ON Doctors.doctor\_id = Appointments.doctor\_id;

**Stored Procedure: A procedure ScheduleAppointment is created to encapsulate the logic of scheduling appointments.**

CREATE PROCEDURE ScheduleAppointment(

IN p\_patient\_id INT,

IN p\_doctor\_id INT,

IN p\_appointment\_date DATE,

IN p\_appointment\_time TIME,

IN p\_remarks TEXT

)

BEGIN

INSERT INTO Appointments (patient\_id, doctor\_id, appointment\_date, appointment\_time, status, remarks)

VALUES (p\_patient\_id, p\_doctor\_id, p\_appointment\_date, p\_appointment\_time, 'pending', p\_remarks);

END //

**Trigger: A trigger UpdateAppointmentStatus is created to log changes in appointment status to a history table whenever an appointment status is updated.**

CREATE TRIGGER UpdateAppointmentStatus

AFTER UPDATE ON Appointments

FOR EACH ROW

BEGIN

IF OLD.status <> NEW.status THEN

INSERT INTO AppointmentStatusHistory (appointment\_id, old\_status, new\_status, updated\_at)

VALUES (OLD.appointment\_id, OLD.status, NEW.status, NOW());

    END IF;

END;

**Q.4 Indicate the Normalization form being used in the schema defined and why you chose to keep it that particular normal form.**

* First Normal Form (1NF): Each table has a primary key, and each column contains atomic values. There are no repeating groups or arrays. For example, each column in the Patients, Doctors, Appointments, Departments, Staff, and MedicalRecords tables contains atomic values.
* Second Normal Form (2NF): The schema is in 1NF, and all non-key attributes are fully dependent on the primary key. There are no partial dependencies. For example, in the MedicalRecords table, diagnosis, treatment, and prescription are dependent only on the record\_id, which is the primary key.
* Third Normal Form (3NF): The schema is in 2NF, and there are no transitive dependencies. In other words, there are no attributes that depend on non-key attributes. For example, in the Departments table, the head\_doctor\_id is dependent on the department\_id, which is the primary key. There are no attributes that depend on other non-key attributes.

**Q.5 Once your schema is well defined, choose any Relational Database system (MySQL, MariaDB, etc) and practically implement the schema so that you are able to perform at least the following operations.**

HMS should be capable to recognize already registered patients and user roles.

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

* Register a new Doctor:

INSERT INTO Doctors (doctor\_id,doctor\_name, specialization, phone\_number, email)

VALUES (202,'Dr.Divya Kumar’, 'Cardiology', '91-8973425789', 'divya.kumar@example.com');

* Register a new Patient:

INSERT INTO Patients (patient\_id,patient\_name, date\_of\_birth, gender, address, phone\_number, email)

VALUES (1,'Ashish Jadhav’, '1990-05-15', 'M', '123 Main st, pune, India', '91-9876543210', 'ashish@example.com');

* Register a new Staff:

INSERT INTO Staff (staff\_id,staff\_name, department\_id, position, phone\_number, email)

VALUES (34,'Priti Gaikwad’, 1, 'Nurse', '91-9551234567', priti@example.com');

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

INSERT INTO MedicalRecords (patient\_id, doctor\_id, appointment\_id, diagnosis, date)

VALUES (2, 103, 457, 'New diagnosis description', '2024-02-16');

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

SELECT \*

FROM Patients

WHERE patient\_id = 3;

**to fetch only the patient's name, date of birth, and email:**

SELECT patient\_name, date\_of\_birth

FROM Patients

WHERE patient\_id = <patient\_id>;

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

SELECT

P.patient\_name,

A.appointment\_date,

D.doctor\_name,

SUM(TREATMENT\_COST) AS total\_cost

FROM Appointments A

JOIN Patients P ON A.patient\_id = P.patient\_id

JOIN Doctors D ON A.doctor\_id = D.doctor\_id

JOIN (

SELECT appointment\_id,

SUM(treatment\_cost) AS TREATMENT\_COST

FROM MedicalRecords

GROUP BY appointment\_id

) MR ON A.appointment\_id = MR.appointment\_id

WHERE A.patient\_id = <patient\_id>

AND A.status = 'completed'

GROUP BY P.patient\_name, A.appointment\_date, D.doctor\_name;

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

**Fetch Patient appointment with doctor details:**

SELECT

Patients.patient\_name,

Appointments.appointment\_date,

Appointments.appointment\_time,

Doctors.doctor\_name,

Doctors.specialization

FROM Appointments

JOIN Patients ON Appointments.patient\_id = Patients.patient\_id

JOIN Doctors ON Appointments.doctor\_id = Doctors.doctor\_id;

**Fetch staff details with their departments:**

SELECT

Staff.staff\_name,

Staff.position,

Departments.department\_name

FROM Staff

JOIN Departments ON Staff.department\_id = Departments.department\_id;

**Fetch medical records with patient and doctor information:**

SELECT

Patients.patient\_name,

Doctors.doctor\_name,

MedicalRecords.diagnosis,

MedicalRecords.treatment,

MedicalRecords.prescription,

MedicalRecords.date

FROM MedicalRecords

JOIN Patients ON MedicalRecords.patient\_id = Patients.patient\_id

JOIN Doctors ON MedicalRecords.doctor\_id = Doctors.doctor\_id;

**Fetchappontments along with patient,doctor and department details:**

SELECT

Patients.patient\_name,

Doctors.doctor\_name,

Departments.department\_name,

Appointments.appointment\_date,

Appointments.appointment\_time

FROM Appointments

JOIN Patients ON Appointments.patient\_id = Patients.patient\_id

JOIN Doctors ON Appointments.doctor\_id = Doctors.doctor\_id

JOIN Departments ON Doctors.department\_id = Departments.department\_id;

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

**Patient Appointments View:**

CREATE VIEW PatientAppointments AS

SELECT

Patients.patient\_id,

Patients.patient\_name,

Appointments.appointment\_id,

Appointments.appointment\_date,

Appointments.appointment\_time,

Doctors.doctor\_name

FROM Appointments

JOIN Patients ON Appointments.patient\_id = Patients.patient\_id

JOIN Doctors ON Appointments.doctor\_id = Doctors.doctor\_id;

**Doctor Appointments View:**

CREATE VIEW DoctorAppointments AS

SELECT

Doctors.doctor\_id,

Doctors.doctor\_name,

Appointments.appointment\_id,

Appointments.appointment\_date,

Appointments.appointment\_time,

Patients.patient\_name

FROM Appointments

JOIN Doctors ON Appointments.doctor\_id = Doctors.doctor\_id

JOIN Patients ON Appointments.patient\_id = Patients.patient\_id;

**Materialized Views:**

CREATE MATERIALIZED VIEW DailyAppointmentSummary AS

SELECT

appointment\_date,

COUNT(appointment\_id) AS total\_appointments

FROM Appointments

GROUP BY appointment\_date;

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

CREATE INDEX idx\_appointment\_date ON Appointments(appointment\_date);

**- Try optimizing bill generation using stored procedures.**

CREATE PROCEDURE GenerateBill(

IN p\_patient\_id INT

)

BEGIN

DECLARE total\_cost DECIMAL(10, 2);

SELECT SUM(treatment\_cost) INTO total\_cost

FROM MedicalRecords

WHERE patient\_id = p\_patient\_id;

SELECT CONCAT('Total bill for patient ', (SELECT patient\_name FROM Patients WHERE patient\_id = p\_patient\_id), ': $', total\_cost) AS bill;

END //

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

CREATE PROCEDURE GenerateBill(

IN p\_patient\_id INT

)

BEGIN

DECLARE total\_cost DECIMAL(10, 2);

SELECT SUM(treatment\_cost) INTO total\_cost

FROM MedicalRecords

WHERE patient\_id = p\_patient\_id;

SELECT CONCAT('Total bill for patient ', (SELECT patient\_name FROM Patients WHERE patient\_id = p\_patient\_id), ': $', total\_cost) AS bill;

END //









