**DB Schema for Hospital Management**

Designing a comprehensive database schema for a Hospital Management System involves several entities, relationships, and database concepts. Below is a database schema that includes entities such as Patients, Doctors, Departments, Appointments, and Medical Records.

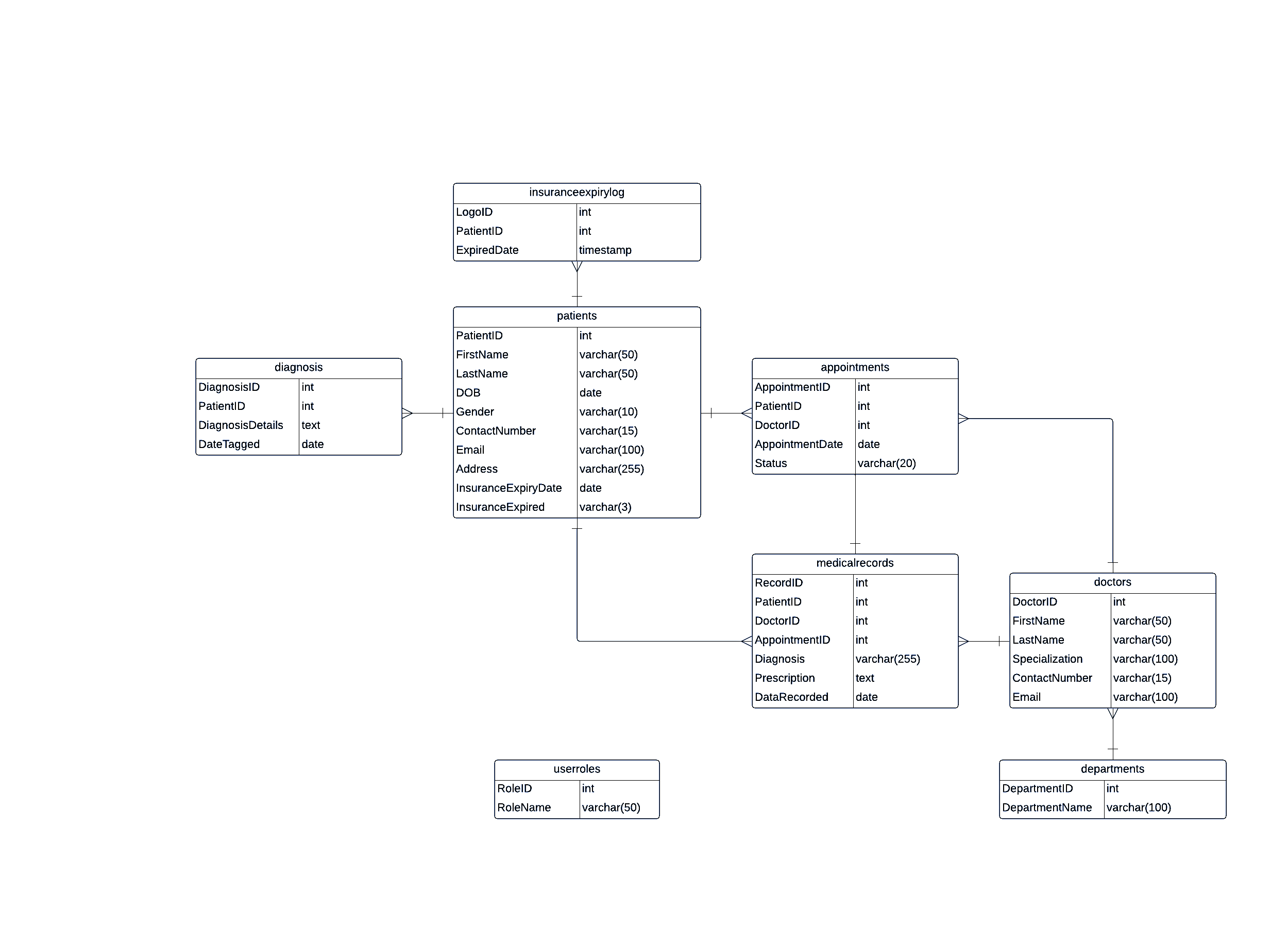


Fig. Database Schema of Hospital Management System

**Database Schema:**

1. **Patients:**

* PatientID (PK)
* FirstName
* LastName
* DOB
* Gender
* ContactNumber
* Email
* Address

1. **Doctors:**

* DoctorID (PK)
* FirstName
* LastName
* Specialization
* ContactNumber
* Email

1. **Departments:**

* DepartmentID (PK)
* DepartmentName

1. **Appointments:**

* AppointmentID (PK)
* PatientID (FK referencing Patients)
* DoctorID (FK referencing Doctors)
* AppointmentDate
* Status (e.g., scheduled, canceled)

1. **MedicalRecords:**

* RecordID (PK)
* PatientID (FK referencing Patients)
* DoctorID (FK referencing Doctors)
* AppointmentID (FK referencing Appointments)
* Diagnosis
* Prescription
* DateRecorded

1. **Diagnosis:**

* DiagnosisID(PK)
* PatientID(FK referencing Patients)
* DiagnosisDetails
* DateTagged

1. **InsuranceExpiryLog:**

* LogID(PK)
* PatientID(FK referencing Patients)
* ExpiredDate

1. **UserRoles:**

* RoleID(PK)
* RoleName

**Relationships:**

1. **Patients and Appointments:**

* Relationship: One-to-Many
* Type: One Patient can have Many Appointments, but each Appointment is associated with only one Patient.

1. **Doctors and Appointments:**

* Relationship: One-to-Many
* Type: One Doctor can have Many Appointments, but each Appointment is associated with only one Doctor.

1. **Patients and MedicalRecords:**

* Relationship: One-to-Many
* Type: One Patient can have Many Medical Records, but each Medical Record is associated with only one Patient.

1. **Doctors and MedicalRecords:**

* Relationship: One-to-Many
* Type: One Doctor can have Many Medical Records, but each Medical Record is associated with only one Doctor.

1. **Appointments and MedicalRecords:**

* Relationship: One-to-One
* Type: Each Appointment is associated with one and only one Medical Record, and vice versa.

1. **Patients and Diagnosis:**

* Relationship: One-to-Many
* Type: One Patient can have Many Diagnoses, but each Diagnosis is associated with only one Patient.

1. **Patients and InsuranceExpiryLog:**

* Relationship: One-to-Many
* Type: One Patient can have Many Insurance Expiry Logs, but each Insurance Expiry Log is associated with only one Patient.

**Views:**

1. **UpcomingAppointmentsView:**

* Provides a view of upcoming appointments, filtering out past appointments.

1. **PatientHistoryView:**

* Combines information from Patients, Appointments, and MedicalRecords to display a patient's medical history.

1. **OptimizeView:**

* Optimizes Repeated Read Operations Using Views/Materialized Views.

**Indexing:**

* Index on PatientID in the Patients table for efficient retrieval of patient-related information.
* Index on DoctorID in the Doctors table for efficient retrieval of doctor-related information.
* Index on AppointmentDate in the Appointments table for efficient sorting and filtering of appointments.

**Stored Procedures:**

1. **CreateAppointmentProcedure:**

* Accepts inputs like PatientID, DoctorID, and AppointmentDate, then creates a new appointment and updates the appointment status.

1. **GeneratePatientBillProcedure:**

* Accepts PatientID and calculates the total bill based on appointments and medical records, then generates a bill.

**Triggers:**

1. **AppointmentStatusTrigger:**

* Automatically updates the status of an appointment when the AppointmentDate is in the past.

1. **InsuranceExpiryTrigger:**

* Trigger to indicate when a patient's medical insurance limit has expired.

**Normalization:**

The schema is designed to be in at least the Third Normal Form (3NF). This means that data redundancy is minimized, and relationships between entities are appropriately established. For example:

* Patient information is stored in the Patients table.
* Doctor information is stored in the Doctors table.
* Department information is stored in the Departments table.

The use of foreign keys establishes relationships between the tables, and the structure avoids unnecessary duplication of data, adhering to the principles of normalization. This helps in maintaining data integrity and reducing update anomalies.

**Github Link –**

https://github.com/ManasiNarkhede/SolutionAnalyst\_InternSS

The above GitHub link provides access to the code for above mentioned Database Schema.