**PROJECT HOMEWORK**

**DATABASE MANAGEMENT SYSTEMS**



**Creator:**

Ammar Ajam G231210554

Faculty of Computer and Information Sciences

Computer Engineering

Sakarya University

October-December 2024

**Summary**

This project involves designing a database schema for a hospital management system. The schema includes tables to manage users (patients, doctors, nurses, staff), their contact information, hospitals, cities, districts, departments, and appointments. It also tracks emergency contacts, insurance details, payments, prescriptions, and medications.

Key features:

* **Users**: Includes patients, doctors, and staff, linked to contact and hospital data.
* **Hospital and Departments**: Hospitals are associated with cities, districts, managers, and departments.
* **Appointments and Payments**: Manages patient-doctor appointments with statuses and linked payments.
* **Prescriptions and Medications**: Tracks prescribed medications with dosages, dates, and supplies.

Integrity is maintained with foreign key constraints and ON DELETE CASCADE for relational consistency. Constraints ensure valid data entries (e.g., dates, unique fields, and checks for proper values). The project implements default values, randomizations for prescriptions, and cascading deletes for efficient data management.

**Business Rules**

**Business Rules (Using only 1:M, 1:1, etc.):**

**users**

* 1:1 with contact\_information (via user\_id)
* 1:M with patient (user can be a patient or not?)
* 1:M with doctors/nurses/staff/managers (user can be one of these roles)

**contact\_information**

* 1:1 with users (one contact info per user)

**managers**

* 1:1 with hospital
* 1:M with departments

**hospital**

* 1:1 with city
* 1:1 with district
* 1:1 with managers
* 1:M with departments

**city**

* 1:M with district
* 0:M with hospital

**district**

* 1:1 with city
* 0:M with hospital

**patient**

* 1:1 with users
* 1:M with appointments
* 1:M with payments
* 0:1 with insurance
* 0:M with emergency\_contact

**doctors**

* 1:1 with users
* 1:M with appointments (a doctor can have many appointments)
* 1:1 with departments

**nurses**

* 1:1 with users
* 1:1 with departments

**staff**

* 1:1 with users
* 1:1 with departments

**departments**

* 1:1 with managers
* 1:M with doctors
* 1:M with nurses
* 1:M with staff
* 1:M with rooms

**appointments**

* 1:1 with patient
* 1:1 with doctor
* 1:1 with payments
* 1:1 with prescriptions
* 0:1 with room

**payments**

* 1:1 with appointments

**prescriptions**

* 1:1 with appointments
* 1:M with prescription\_medications

**medications**

* 0:M with prescription\_medications

**insurance**

* 1:1 with patient

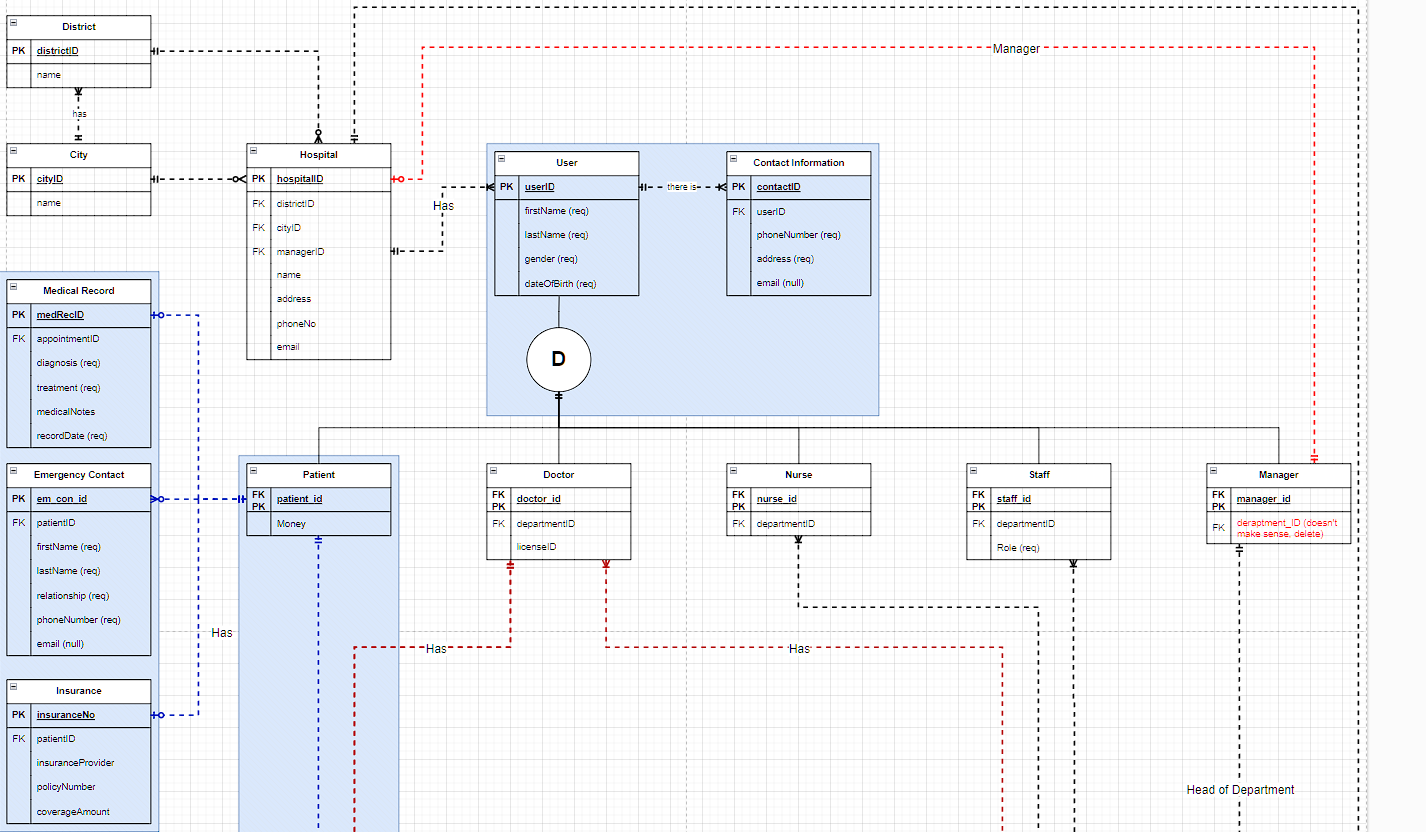
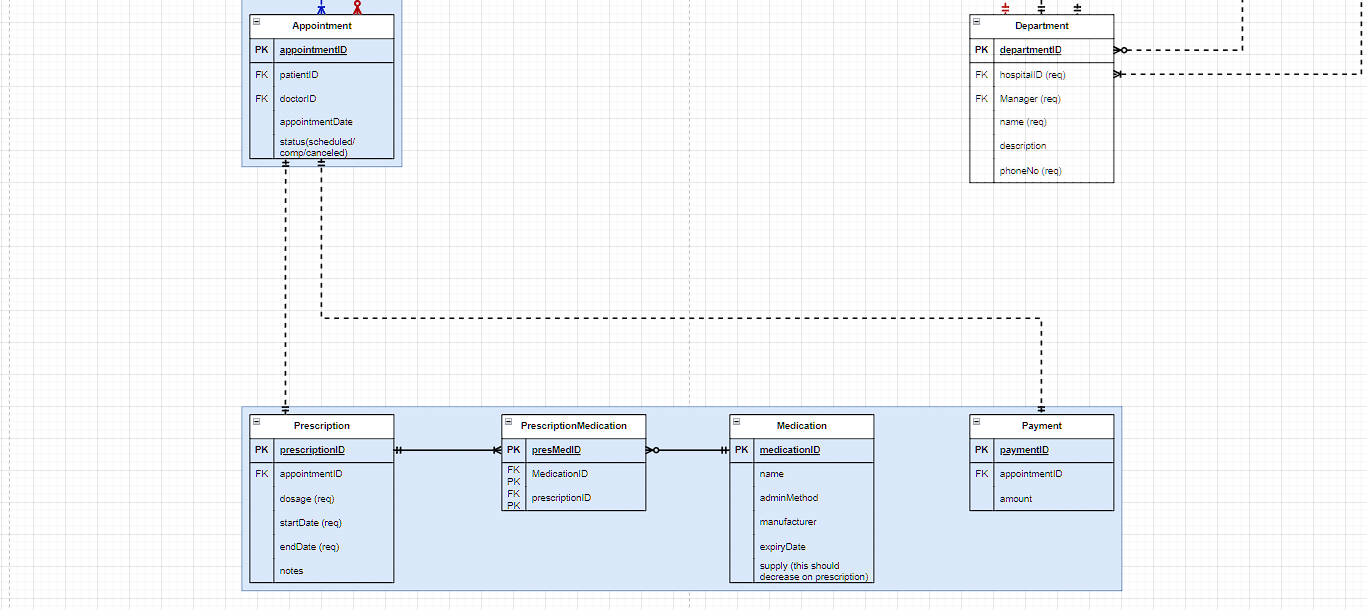
**emergency\_contact**

* 1:1 with patient

**prescription\_medications** (junction table)

* M:M between prescriptions and medications

Crow’s Foot Diyagram

**Triggers**

**1. Trigger: trigger\_set\_payment\_amount**

* Ensures the correct payment amount is assigned to the payments table before inserting a record. The amount depends on the department of the doctor associated with the appointment.
* Fetches the department name from the departments table, based on the appointment's doctor.
* Sets payment amounts based on predefined rules:
  + 50 for "Gastroenterology."
  + 75 for "Phlebotomy."
  + 25 for "Dermatology."
* If the department name is unrecognized, it raises an exception.

**2. Trigger: set\_random\_values**

* Automatically assigns random values to the fields of a new medications record before insertion.
* Assigns:
  + A random administration\_method (e.g., "Oral," "Injection").
  + A random manufacturer (e.g., "PharmaCorp").
  + A random name (e.g., "Healzyme").
  + A random expiry date between 1 and 2 years from the current date.
* Ensures diverse, realistic medication data in the database.

**3. Trigger: after\_prescription\_insert**

* Executes after a new prescription is inserted into the prescriptions table.
* Automatically:
  + Selects a random medication from the medications table.
  + Reduces the medication's supply by the dosage specified in the prescription.
  + Inserts a relationship between the prescription and the medication into the prescription\_medications table.
* Maintains medication supply and automates linking prescriptions to medications.

**4. Trigger: after\_insert\_insurance**

* Executes after a new insurance record is inserted into the insurance table.
* Automatically updates the patient’s balance by adding the insurance coverage amount to their existing balance.
* Ensures real-time updates to patient financial data.

**5. Trigger: enforce\_max\_emergency\_contacts\_trigger**

* Prevents a patient from having more than 5 emergency contacts.
* Checks the number of emergency contacts associated with the patient before inserting a new record into the emergency\_contact table.
* If the count is 5 or more, it raises an exception.

**Procedures**

**1. Procedure: add\_new\_user**

* Adds a new patient to the system by creating records in the users, contact\_information, and patient tables.
* Inserts basic details (e.g., first\_name, last\_name, gender, date\_of\_birth) into the users table.
* Retrieves the newly created user\_id and uses it to insert contact information (phone\_number, address, email).
* Inserts the user\_id into the patient table to link the user as a patient.

**2. Procedure: book\_new\_appointment**

* Books an appointment for a patient with a specific doctor on a specified date.
* Inserts a record into the appointments table with the provided patient\_id, doctor\_id, app\_date, and app\_status.
* Retrieves the appointment\_id of the new appointment.
* Inserts a corresponding record into the payments table. The payment amount is assigned by the trigger\_set\_payment\_amount.

**3. Procedure: add\_emergency\_contact**

* Adds an emergency contact for a specific patient.
* Inserts a record into the emergency\_contact table with the contact’s details (e.g., first\_name, last\_name, gender, date\_of\_birth, relationship, phone\_number, address, email).
* Relies on the enforce\_max\_emergency\_contacts\_trigger to ensure no patient exceeds 5 emergency contacts.

**4. Procedure: process\_appointment**

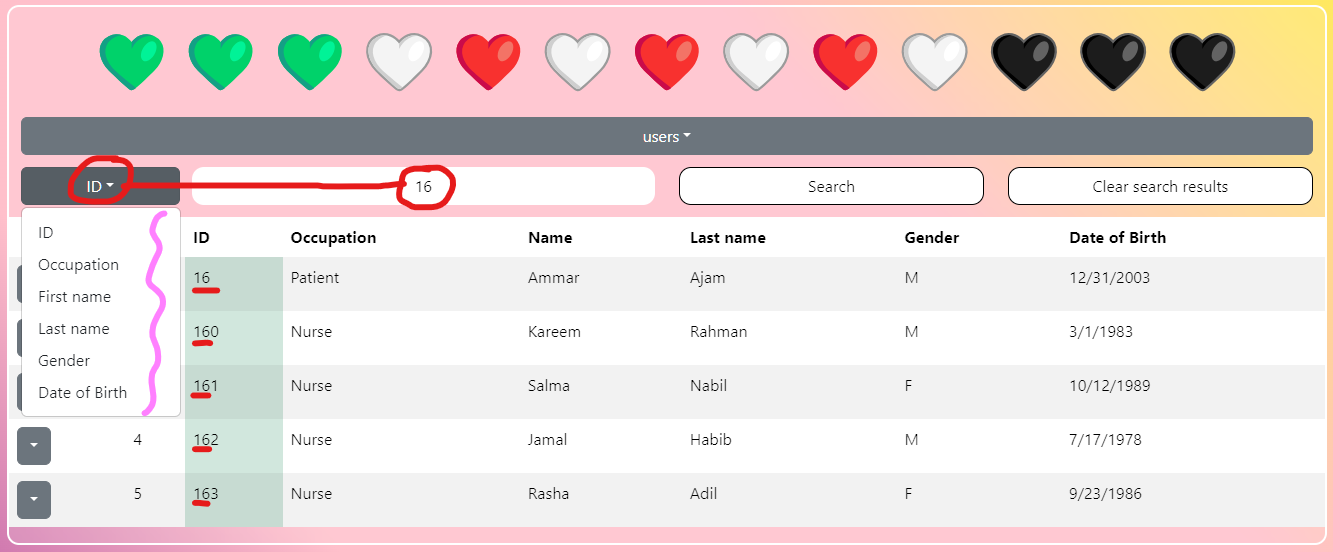
* Handles payment and prescription generation for an appointment.
* Retrieves:
  + The patient’s ID.
  + Payment amount.
  + Appointment status.
* Checks if the appointment is already completed. If it is, an exception is raised.
* Retrieves the patient’s balance and verifies if it is sufficient to cover the payment:
  + If sufficient:
    - Deducts the payment from the patient’s balance.
    - Marks the appointment status as completed (CO).
    - Calls the generate\_prescription procedure to create a prescription.
  + If insufficient:
    - Raises an exception indicating the patient’s balance is insufficient.

**CRUD Operations**

You can **CREATE/UPDATE/DELETE** on some tables (within what makes sense in a hospital system) like adding a new patient, appointment, emergency contact, etc.



You can **READ** from ALL tables, based on its own columns.

****