**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

### Relational schema

### Users: user\_id (Primary Key), first\_name VARCHAR(30) NOT NULL, last\_name VARCHAR(30) NOT NULL, gender CHAR(1) CHECK (gender IN ('M', 'F')) NOT NULL, date\_of\_birth DATE NOT NULL

### Contact\_Information: contact\_id (Primary Key), user\_id INT REFERENCES Users(user\_id) ON DELETE CASCADE, phone\_number VARCHAR(11) NOT NULL UNIQUE, address VARCHAR(100) NOT NULL, email VARCHAR(50) NOT NULL UNIQUE

### City: city\_id (Primary Key), name VARCHAR(30) NOT NULL UNIQUE

### District: district\_id (Primary Key), name VARCHAR(30) NOT NULL UNIQUE

### Managers: manager\_id (Primary Key)

### Hospital: hospital\_id (Primary Key), city\_id INT REFERENCES City(city\_id) ON DELETE CASCADE, district\_id INT REFERENCES District(district\_id) ON DELETE CASCADE, manager\_id INT REFERENCES

### Managers(manager\_id) ON DELETE CASCADE, name VARCHAR(30) NOT NULL, address VARCHAR(30) NOT NULL UNIQUE, phone\_number VARCHAR(30) NOT NULL UNIQUE, email VARCHAR(30) NOT NULL UNIQUE

### Patient: patient\_id INT PRIMARY KEY REFERENCES Users(user\_id) ON DELETE CASCADE, balance INT NOT NULL DEFAULT 100 CHECK(balance >= 0)

### Emergency\_Contact: em\_con\_id SERIAL PRIMARY KEY, patient\_id INT REFERENCES Patient(patient\_id) ON DELETE CASCADE, first\_name VARCHAR(30) NOT NULL, last\_name VARCHAR(30) NOT NULL, gender CHAR(1) CHECK (gender IN ('M', 'F')) NOT NULL, date\_of\_birth DATE NOT NULL CHECK (date\_of\_birth <= CURRENT\_DATE), relationship VARCHAR(30) NOT NULL, phone\_number VARCHAR(11) NOT NULL UNIQUE, address VARCHAR(100) NOT NULL, email VARCHAR(50) NOT NULL UNIQUE

### Insurance: insurance\_id SERIAL PRIMARY KEY, patient\_id INT REFERENCES Patient(patient\_id) ON DELETE CASCADE, insurance\_provider VARCHAR(30) NOT NULL, policy\_number VARCHAR(5) NOT NULL CHECK (LENGTH(policy\_number) = 5), coverage\_amount INT NOT NULL

### Departments: department\_id SERIAL PRIMARY KEY, hospital\_id INT REFERENCES Hospital(hospital\_id) ON DELETE CASCADE, manager\_id INT REFERENCES Managers(manager\_id) ON DELETE CASCADE, name VARCHAR(20) NOT NULL UNIQUE, description VARCHAR(255), phone\_number INT NOT NULL UNIQUE

### Doctors: doctor\_id INT PRIMARY KEY REFERENCES Users(user\_id) ON DELETE CASCADE, department\_id INT REFERENCES Departments(department\_id) ON DELETE CASCADE, license\_id INT NOT NULL

### Staff: staff\_id INT PRIMARY KEY REFERENCES Users(user\_id) ON DELETE CASCADE, department\_id INT REFERENCES Departments(department\_id) ON DELETE CASCADE, role VARCHAR(255) NOT NULL

### Nurses: nurse\_id INT PRIMARY KEY REFERENCES Users(user\_id) ON DELETE CASCADE

### Appointments: appointment\_id SERIAL PRIMARY KEY, patient\_id INT REFERENCES Patient(patient\_id) ON DELETE CASCADE, doctor\_id INT REFERENCES Doctors(doctor\_id) ON DELETE CASCADE, app\_date DATE NOT NULL CHECK (app\_date > CURRENT\_DATE), app\_status CHAR(2) CHECK (app\_status IN ('SC', 'CO', 'CA'))

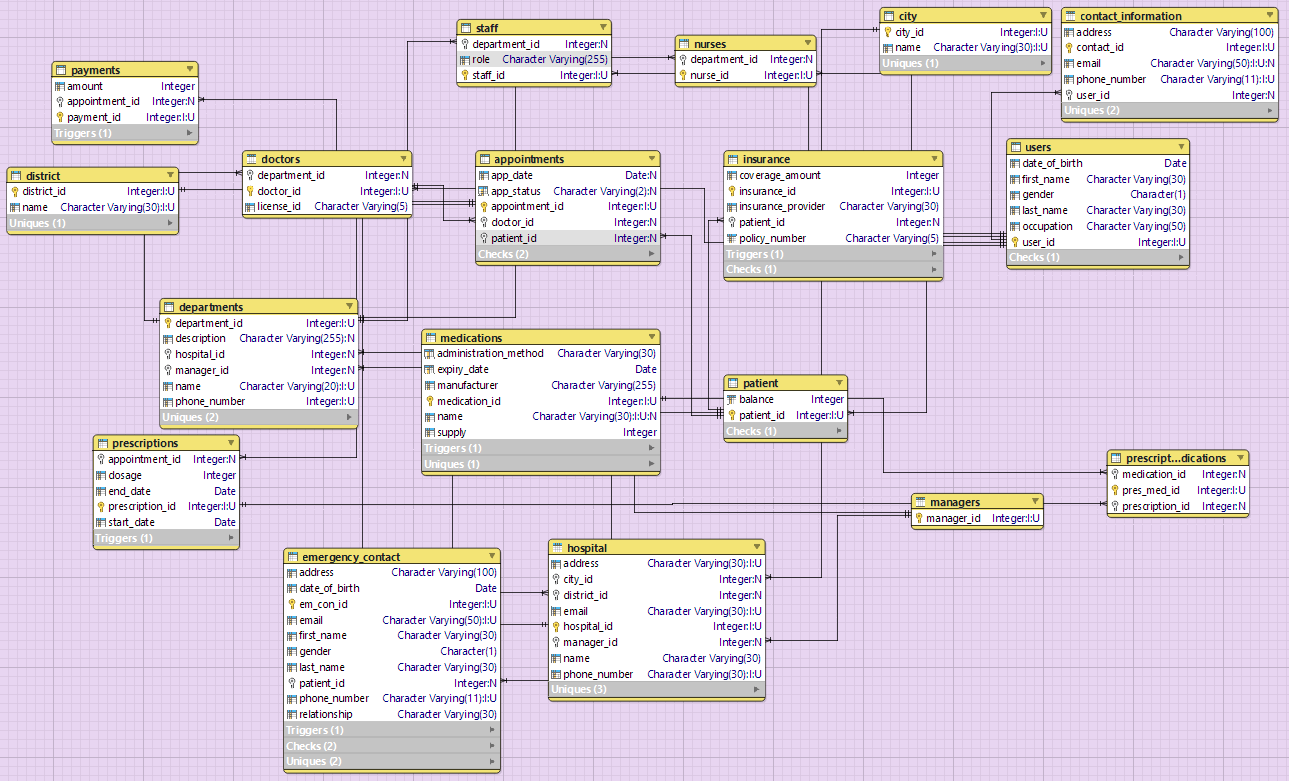
### Payments: payment\_id SERIAL PRIMARY KEY, appointment\_id INT REFERENCES Appointments(appointment\_id), amount INT NOT NULL

### Prescriptions: prescription\_id SERIAL PRIMARY KEY, appointment\_id INT REFERENCES Appointments(appointment\_id) ON DELETE CASCADE, dosage INT NOT NULL DEFAULT 5 + (FLOOR(RANDOM() \* 6))::INT, start\_date DATE NOT NULL DEFAULT CURRENT\_DATE, end\_date DATE NOT NULL DEFAULT CURRENT\_DATE + (5 + FLOOR(RANDOM() \* 4))::INT

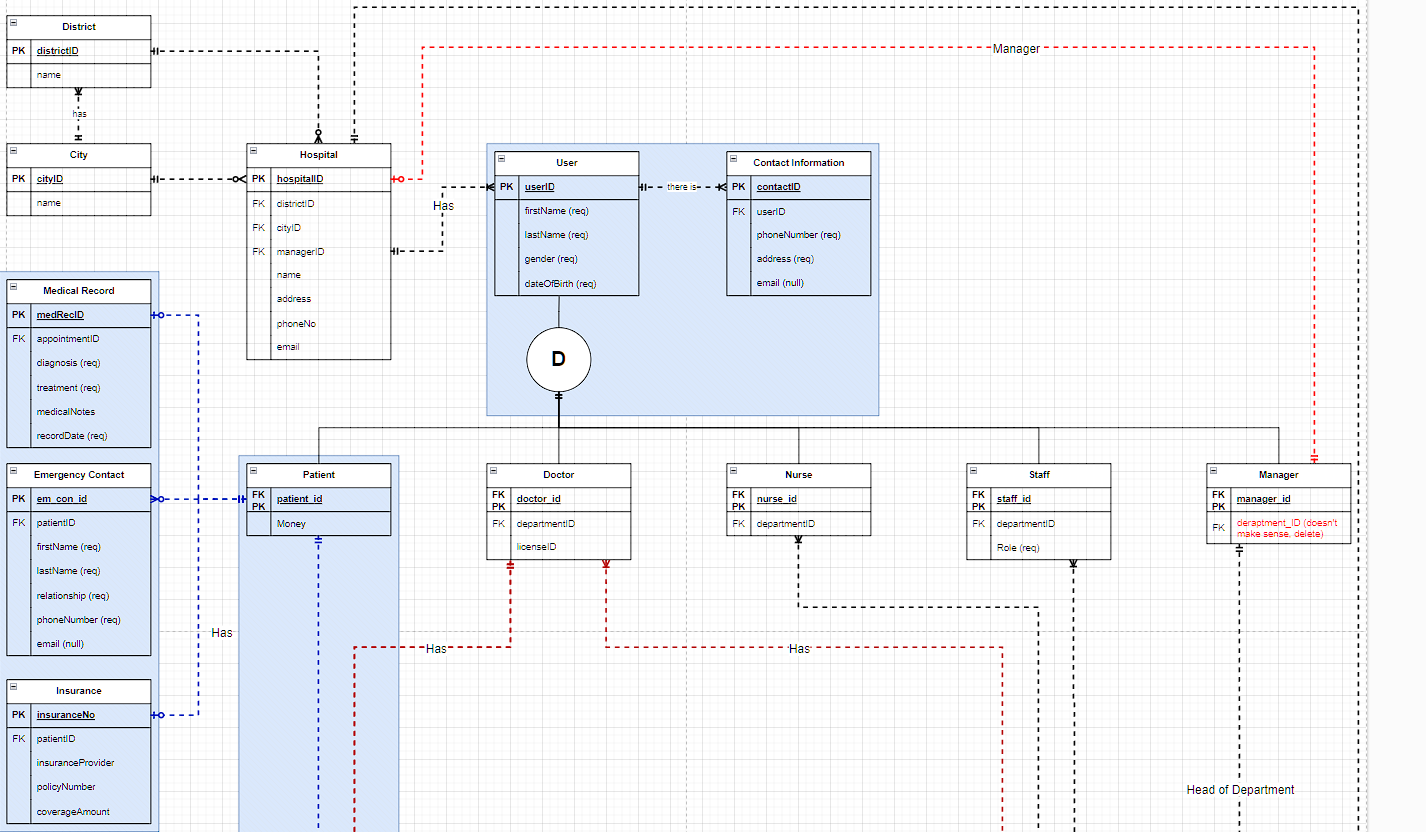
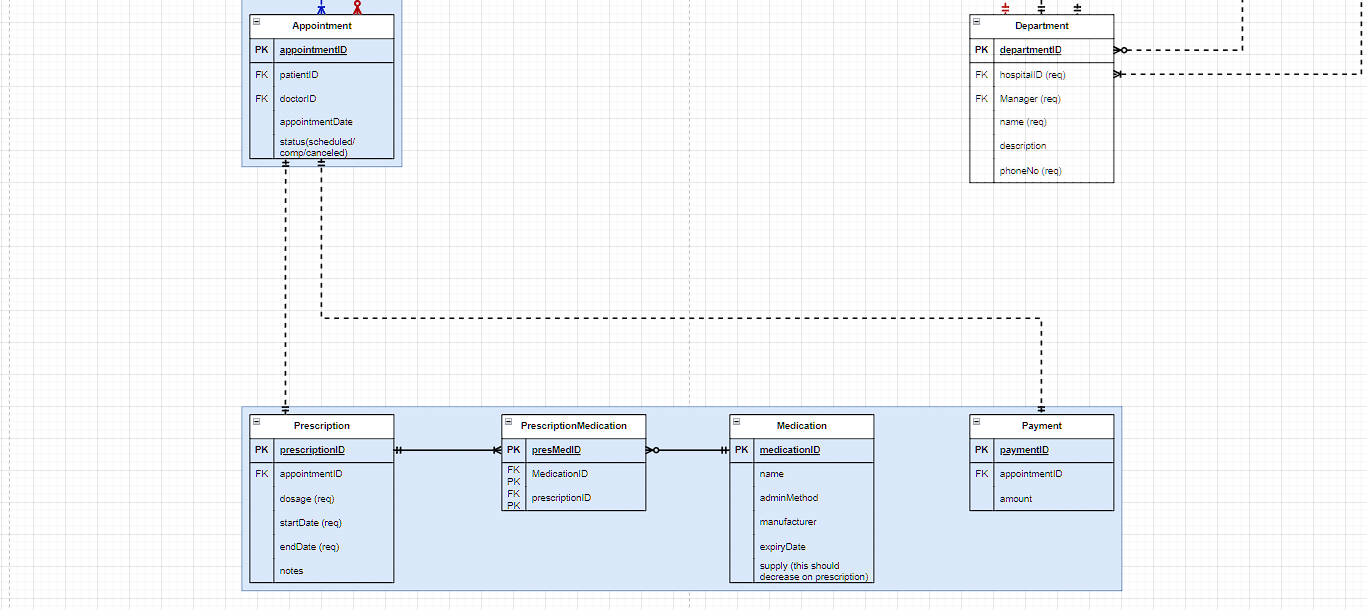
### Prescription\_Medications: pres\_med\_id SERIAL PRIMARY KEY, prescription\_id INT REFERENCES Prescriptions(prescription\_id) ON DELETE CASCADE, medication\_id INT REFERENCES Medications(medication\_id) ON DELETE CASCADE

### Medications: medication\_id SERIAL PRIMARY KEY, name VARCHAR(30) UNIQUE, administration\_method VARCHAR(30) NOT NULL DEFAULT 'Oral', manufacturer VARCHAR(255) NOT NULL DEFAULT 'medixCor', expiry\_date DATE NOT NULL DEFAULT CURRENT\_DATE, supply INT NOT NULL DEFAULT 1000000

Database Schema



**Crow’s Foot Diagram**

**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**

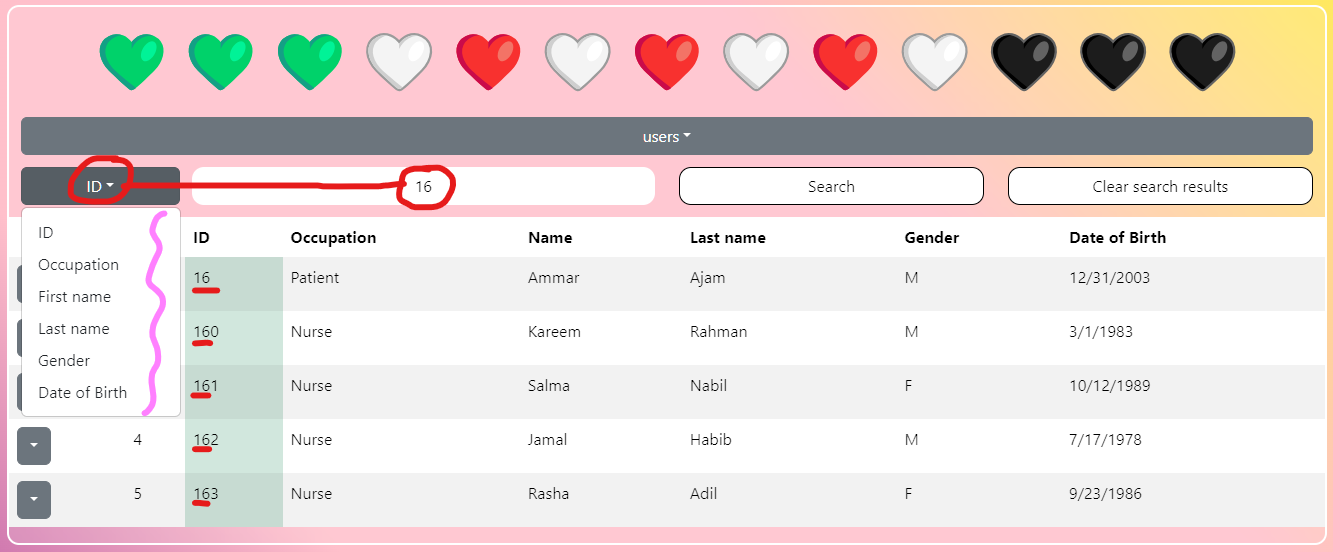
* Handles payment and prescription generation for an appointment.
* Retrieves:
  + The patient’s ID.
  + Payment amount.
  + Appointment status.
* Checks if the appointment is already completed. If it is, an exception is raised.
* Retrieves the patient’s balance and verifies if it is sufficient to cover 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.

****

**Source codes**

https://github.com/DarkAndLighto/DatabaseManagementSystems

CREATE TABLE users (

user\_id SERIAL PRIMARY KEY,

first\_name VARCHAR(30) NOT NULL,

last\_name VARCHAR(30) NOT NULL,

gender CHAR(1) CHECK (gender IN ('M', 'F')) NOT NULL,

date\_of\_birth DATE NOT NULL

);

CREATE TABLE contact\_information (

contact\_id SERIAL PRIMARY KEY,

user\_id INT REFERENCES users(user\_id) ON DELETE CASCADE,

phone\_number VARCHAR(11) NOT NULL UNIQUE,

address VARCHAR(100) NOT NULL,

email VARCHAR(50) NOT NULL UNIQUE

);

CREATE TABLE hospital (

hospital\_id SERIAL PRIMARY KEY,

city\_id INT REFERENCES city(city\_id) ON DELETE CASCADE NOT NULL,

district\_id INT REFERENCES district(district\_id) ON DELETE CASCADE NOT NULL,

manager\_id INT REFERENCES managers(manager\_id) ON DELETE CASCADE NOT NULL,

name VARCHAR(30) NOT NULL,

address VARCHAR(30) NOT NULL UNIQUE,

phone\_number VARCHAR(30) NOT NULL UNIQUE,

email VARCHAR(30) NOT NULL UNIQUE

);

CREATE TABLE city (

city\_id SERIAL PRIMARY KEY,

name VARCHAR(30) NOT NULL UNIQUE

);

CREATE TABLE district (

district\_id SERIAL PRIMARY KEY,

name VARCHAR(30) NOT NULL UNIQUE

);

CREATE TABLE patient(

patient\_id INT PRIMARY KEY REFERENCES users(user\_id) ON DELETE CASCADE,

balance INT NOT NULL DEFAULT 100 CHECK(balance >= 0)

);

ALTER TABLE patient

ADD COLUMN balance INT NOT NULL DEFAULT 100 CHECK(balance >= 0);

DROP COLUMN balance;

CREATE TABLE emergency\_contact(

em\_con\_id SERIAL PRIMARY KEY,

patient\_id INT REFERENCES patient(patient\_id) ON DELETE CASCADE,

first\_name VARCHAR(30) NOT NULL,

last\_name VARCHAR(30) NOT NULL,

gender CHAR(1) CHECK (gender IN ('M', 'F')) NOT NULL,

date\_of\_birth DATE NOT NULL CHECK (date\_of\_birth <= CURRENT\_DATE),

relationship VARCHAR(30) NOT NULL,

phone\_number VARCHAR(11) NOT NULL UNIQUE,

address VARCHAR(100) NOT NULL,

email VARCHAR(50) NOT NULL UNIQUE

);

CREATE TABLE insurance(

insurance\_id SERIAL PRIMARY KEY,

patient\_id INT REFERENCES patient(patient\_id) ON DELETE CASCADE,

insurance\_provider VARCHAR(30) NOT NULL,

policy\_number VARCHAR(5) NOT NULL CHECK (length(policy\_number) = 5),

coverage\_amount INT NOT NULL

);

CREATE TABLE departments(

department\_id SERIAL PRIMARY KEY,

hospital\_id INT REFERENCES hospital(hospital\_id) ON DELETE CASCADE,

manager\_id INT REFERENCES managers(manager\_id) ON DELETE CASCADE,

name VARCHAR(20) NOT NULL UNIQUE,

description VARCHAR(255),

phone\_number INT NOT NULL UNIQUE

);

CREATE TABLE managers(

manager\_id SERIAL PRIMARY KEY

);

CREATE TABLE doctors(

doctor\_id INT PRIMARY KEY REFERENCES users(user\_id) ON DELETE CASCADE,

department\_id INT REFERENCES departments(department\_id) ON DELETE CASCADE,

license\_id INT NOT NULL

);

CREATE TABLE staff(

staff\_id INT PRIMARY KEY REFERENCES users(user\_id) ON DELETE CASCADE,

department\_id INT REFERENCES departments(department\_id) ON DELETE CASCADE,

role VARCHAR(255) NOT NULL

);

CREATE TABLE nurses(

nurse\_id INT PRIMARY KEY REFERENCES users(user\_id) ON DELETE CASCADE

);

CREATE TABLE appointments(

appointment\_id SERIAL PRIMARY KEY,

patient\_id INT REFERENCES patient(patient\_id) ON DELETE CASCADE,

doctor\_id INT REFERENCES doctors(doctor\_id) ON DELETE CASCADE,

app\_date data NOT NULL CHECK (app\_date > CURRENT\_DATE),

app\_status char(2) CHECK (app\_status IN ('SC', 'CO', 'CA'))

);

select \* from appointments;

ALTER TABLE appointments

ADD CONSTRAINT app\_date CHECK (app\_date > CURRENT\_DATE);

CREATE TABLE payments(

payment\_id SERIAL PRIMARY KEY,

appointment\_id INT REFERENCES appointments(appointment\_id),

amount INT NOT NULL

);

DROP TABLE prescriptions;

CREATE TABLE prescriptions (

prescription\_id SERIAL PRIMARY KEY,

appointment\_id INT REFERENCES appointments(appointment\_id) ON DELETE CASCADE,

dosage INT NOT NULL DEFAULT 5 + (FLOOR(RANDOM() \* 6))::INT,

start\_date DATE NOT NULL DEFAULT CURRENT\_DATE,

end\_date DATE NOT NULL DEFAULT CURRENT\_DATE + (5 + FLOOR(RANDOM() \* 4))::INT

);

DROP TABLE prescription\_medications;

CREATE TABLE prescription\_medications(

pres\_med\_id SERIAL PRIMARY KEY,

prescription\_id INT REFERENCES prescriptions(prescription\_id) ON DELETE CASCADE,

medication\_id INT REFERENCES medications(medication\_id) ON DELETE CASCADE

);

DROP TABLE medications;

CREATE TABLE medications(

medication\_id SERIAL PRIMARY KEY,

name VARCHAR(30) UNIQUE,

administration\_method VARCHAR(30) NOT NULL DEFAULT 'Oral',

manufacturer VARCHAR(255) NOT NULL DEFAULT 'medixCor',

expiry\_date DATE NOT NULL DEFAULT CURRENT\_DATE,

supply int NOT NULL DEFAULT 1000000

);

CREATE OR REPLACE FUNCTION set\_payment\_amount()

RETURNS TRIGGER AS $$

DECLARE

department\_name VARCHAR(20);

BEGIN

-- Retrieve the department name based on the appointment's doctor

SELECT d.name

INTO department\_name

FROM appointments a

JOIN doctors doc ON a.doctor\_id = doc.doctor\_id

JOIN departments d ON doc.department\_id = d.department\_id

WHERE a.appointment\_id = NEW.appointment\_id;

-- Set the payment amount based on the department

CASE department\_name

WHEN 'Gastroenterology' THEN NEW.amount := 50;

WHEN 'Phlebotomy' THEN NEW.amount := 75;

WHEN 'Dermatology' THEN NEW.amount := 25;

ELSE RAISE EXCEPTION 'Unknown department: %', department\_name;

END CASE;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_set\_payment\_amount

BEFORE INSERT ON payments

FOR EACH ROW

EXECUTE FUNCTION set\_payment\_amount();

CREATE OR REPLACE PROCEDURE generate\_prescription(

IN new\_appointment\_id INT

)

LANGUAGE plpgsql

AS $$

BEGIN

-- Insert a new prescription entry for the provided appointment ID

INSERT INTO prescriptions (appointment\_id)

VALUES (new\_appointment\_id);

-- Return success message for logging/debugging

RAISE NOTICE 'Prescription generated for appointment ID %.', new\_appointment\_id;

END;

$$;

drop FUNCTION assign\_random\_values\_to\_medications;

CREATE OR REPLACE FUNCTION assign\_random\_values\_to\_medications()

RETURNS TRIGGER AS $$

DECLARE

admin\_methods TEXT[] := ARRAY['Oral', 'Injection', 'Topical', 'Inhalation'];

manufacturers TEXT[] := ARRAY['PharmaCorp', 'MediSupply', 'HealthPlus', 'BioLife'];

medication\_names TEXT[] := ARRAY[

'Panacea', 'Medica', 'CuraTab', 'Healzyme', 'TheraMax',

'ReliefX', 'VitaPlus', 'NeuroAid', 'ImmunoCare', 'CardioFit'

];

year\_offset INT;

BEGIN

-- Assign random values for each field

NEW.administration\_method := admin\_methods[ceil(random() \* array\_length(admin\_methods, 1))::int];

NEW.manufacturer := manufacturers[ceil(random() \* array\_length(manufacturers, 1))::int];

NEW.name := medication\_names[ceil(random() \* array\_length(medication\_names, 1))::int];

-- Calculate random expiry date

year\_offset := (1 + ceil(random() \* 1)::int); -- Randomly choose 1 or 2 years

NEW.expiry\_date := CURRENT\_DATE + (interval '1 year' \* year\_offset);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER set\_random\_values

BEFORE INSERT ON medications

FOR EACH ROW

EXECUTE FUNCTION assign\_random\_values\_to\_medications();

CREATE OR REPLACE FUNCTION after\_insert\_prescription()

RETURNS TRIGGER AS $$

DECLARE

random\_medication\_id INT;

dosage INT := NEW.dosage;

BEGIN

--select a random medication

SELECT medication\_id INTO random\_medication\_id

FROM medications

ORDER BY random()

LIMIT 1;

---Update the supply of the selected medication

UPDATE medications

SET supply = supply - dosage

WHERE medication\_id = random\_medication\_id;

-- Insert the relation into prescription\_medications

INSERT INTO prescription\_medications(prescription\_id, medication\_id)

VALUES (NEW.prescription\_id, random\_medication\_id);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER after\_prescription\_insert

AFTER INSERT ON prescriptions

FOR EACH ROW

EXECUTE FUNCTION after\_insert\_prescription();

CREATE OR REPLACE FUNCTION after\_insert\_insurance()

RETURNS TRIGGER AS $$

BEGIN

-- Update the patient's balance

UPDATE patient

SET balance = balance + NEW.coverage\_amount

WHERE patient.patient\_id = NEW.patient\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER after\_insert\_insurance

AFTER INSERT ON insurance

FOR EACH ROW

EXECUTE FUNCTION after\_insert\_insurance();

CREATE OR REPLACE FUNCTION enforce\_max\_emergency\_contacts()

RETURNS TRIGGER AS $$

BEGIN

-- Check the current number of emergency contacts for the patient

IF (SELECT COUNT(\*) FROM emergency\_contact WHERE patient\_id = NEW.patient\_id) >= 5 THEN

RAISE EXCEPTION 'A patient can have a maximum of 5 emergency contacts.';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce\_max\_emergency\_contacts\_trigger

BEFORE INSERT ON emergency\_contact

FOR EACH ROW

EXECUTE FUNCTION enforce\_max\_emergency\_contacts();

/\* This should be add\_new\_patient\*\*   \*/

CREATE OR REPLACE PROCEDURE add\_new\_user(

    IN first\_name TEXT,

    IN last\_name TEXT,

    IN gender CHAR,

    IN date\_of\_birth DATE,

    IN phone\_number INT,

    IN address TEXT,

    IN email TEXT

)

LANGUAGE plpgsql

AS $$

DECLARE

    new\_user\_id INT;

BEGIN

    INSERT INTO users(occupation, first\_name, last\_name, gender, date\_of\_birth)

    VALUES ('Patient', first\_name, last\_name, gender, date\_of\_birth)

    RETURNING user\_id INTO new\_user\_id;

    INSERT INTO contact\_information(user\_id, phone\_number, address, email)

    VALUES (new\_user\_id, phone\_number, address, email);

    INSERT INTO patient(patient\_id)

    VALUES (new\_user\_id);

END $$;

drop procedure book\_new\_appointment;

CREATE OR REPLACE PROCEDURE book\_new\_appointment(

    IN patient\_id INT,

    IN doctor\_id INT,

    IN app\_date DATE,

    IN app\_status CHAR(2)

)

LANGUAGE plpgsql

AS $$

DECLARE

    new\_appointment\_id INT;

BEGIN

    -- Insert into appointments and retrieve the ID

    INSERT INTO appointments(patient\_id, doctor\_id, app\_date, app\_status)

    VALUES (patient\_id, doctor\_id, app\_date, app\_status)

    RETURNING appointment\_id INTO new\_appointment\_id;

    -- Insert into payments (amount will be set by the trigger)

    INSERT INTO payments(appointment\_id)

    VALUES (new\_appointment\_id);

END;

$$;

CREATE OR REPLACE PROCEDURE add\_emergency\_contact(

    IN patient\_id INT,

    IN first\_name TEXT,

    IN last\_name TEXT,

    IN gender CHAR,

    IN date\_of\_birth DATE,

    IN relationship TEXT,

    IN phone\_number TEXT,

    IN address TEXT,

    IN email TEXT

)

LANGUAGE plpgsql

AS $$

BEGIN

    INSERT INTO emergency\_contact (patient\_id, first\_name, last\_name, gender, date\_of\_birth, relationship, phone\_number, address, email)

    VALUES (patient\_id, first\_name, last\_name, gender, date\_of\_birth, relationship, phone\_number, address, email);

END;

$$;

///process appointment

DROP PROCEDURE process\_appointment;

CREATE OR REPLACE PROCEDURE process\_appointment(

    IN process\_appointment\_id INT

)

LANGUAGE plpgsql

AS $$

DECLARE

    process\_patient\_id INT;

    process\_amount INT;

    process\_patient\_balance INT;

    process\_appointment\_status CHAR(2);

BEGIN

    -- Retrieve details about the appointment and payment

    SELECT a.patient\_id, p.amount, a.app\_status

    INTO process\_patient\_id, process\_amount, process\_appointment\_status

    FROM appointments a

    JOIN payments p ON a.appointment\_id = p.appointment\_id

    WHERE a.appointment\_id = process\_appointment\_id;

    -- Check if the appointment is already completed

    IF process\_appointment\_status = 'CO' THEN

        RAISE EXCEPTION 'Appointment is already completed.';

    END IF;

    -- Get the current balance of the patient

    SELECT pt.balance INTO process\_patient\_balance

    FROM patient pt

    WHERE pt.patient\_id = process\_patient\_id;

    -- Check if the patient has enough balance

    IF process\_patient\_balance >= process\_amount THEN

        -- Deduct the payment amount from the patient's balance

        UPDATE patient

        SET balance = balance - process\_amount

        WHERE patient\_id = process\_patient\_id;

        -- Set the appointment status to 'CO' (Completed)

        UPDATE appointments

        SET app\_status = 'CO'

        WHERE appointment\_id = process\_appointment\_id;

        -- Call the new procedure to generate a prescription

        CALL generate\_prescription(process\_appointment\_id);

        -- Return success message

        RAISE NOTICE 'Appointment completed successfully, and prescription generated.';

    ELSE

        -- Return insufficient balance message

        RAISE EXCEPTION 'Insufficient balance to complete the appointment.';

    END IF;

END;

$$;

///add doctor

DO $$

DECLARE

    new\_user\_id INT;

BEGIN

    INSERT INTO users (first\_name, last\_name, gender, date\_of\_birth, occupation)

    VALUES ('Rojda', 'bulbul', 'F', '11/10/2002', 'Doctor')

    RETURNING user\_id INTO new\_user\_id;

    INSERT INTO contact\_information (user\_id, phone\_number, address, email)

    VALUES (new\_user\_id, 523522432, 'Sanliurfa', 'ezzrojim@gmail.com');

    INSERT INTO doctors (doctor\_id, department\_id, license\_id)

    VALUES (new\_user\_id, 3, 34271);

END $$;

///add manager

DO $$

DECLARE

    new\_user\_id INT;

BEGIN

    INSERT INTO users (first\_name, last\_name, gender, date\_of\_birth, occupation)

    VALUES ('Hadi', 'Kazziha', 'M', '8/13/1993', 'Manager')

    RETURNING user\_id INTO new\_user\_id;

    INSERT INTO contact\_information (user\_id, phone\_number, address, email)

    VALUES (new\_user\_id, 9631812401, 'Damascus', 'h@ka');

    INSERT INTO managers (manager\_id)

    VALUES (new\_user\_id);

END $$;

select \* from managers;