B221202086 Muhammed DEMİRÖRS

B221202085 Arda ALİŞAN

DATABASE MANAGEMENT SYSTEMS PROJECT ASSIGNMENT

Scenario

A doctor examines different patients. A doctor can write prescriptions to patients. There is a nurse attached to each doctor. People are also inherited. A software system is created for a hospital. The manager wants to store and manage some information such as doctors, patients, prescriptions in the hospital. The database is expected to contain information about people, doctors, patients, nurses, drugs, prescriptions, and appointments.

Business Rules

-Each doctor has name,e-mail,telephonenumber,specialty.

-Prescription has an id.

-A drug has an id,name,drugtype(syrup,pill,injection).

-Appointment has an ID,appointmentdate,description.

-Each patient has an ame,e-mail,telephonenumber.

-Each room has an id,name.

-Each clinic has an id,name.

-Each nurse has an name,e-mail,phone number.

-In the system,each person has personid,name,e-mail,telephone number information.Also,aperson can exist in just three categories: a doctor, a nurse and a patient. One person can not be all three.
People are distinguished from each other by their id.

-A doctor can make more than one appointment.Only one doctor can be visited for an appointment.

-A prescription is created by a doctor.A doctor may not write a prescription or may write more than one prescription.

-A drug may be prescribed more than once, and a prescription may contain more than one drug.

-A patient can be at an appointment, but a patient can have more than one appointment.

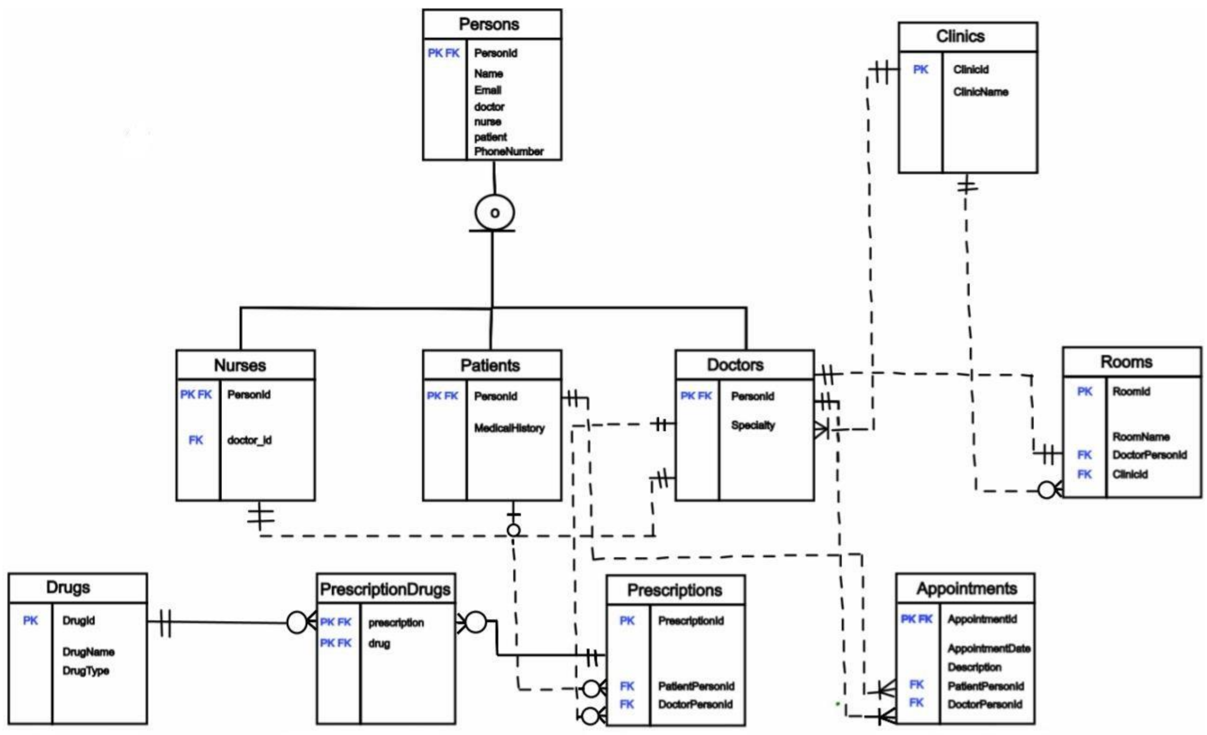
-A patient may receive no prescription at all,or may receive a prescription. A prescription may be written for a patient, or may not be written at all.

-A room has only one doctor. A doctor has only one room.

-A clinic must have at least one doctor, but there can also be more than one doctor but a doctor can only be at a clinic.

-A nurse can only work with one doctor, a doctor can only work with one nurse.

-There are one or more rooms in a clinic. A room belongs to only one clinic.



Relational Schema (Textual Representation):

- Persons(PersonId:bigint, Name:varchar(30), Email:varchar(50), PhoneNumber:bigint, doctor:bool, nurse:bool, patient:bool)
- Nurses(PersonId:bigint, DoctorId:bigint)
- Patients(PersonId:bigint, MedicalHistory:varchar(100))
- Doctors(PersonId:bigint, Speciality:varchar(30))
- Clinics(ClinicId:bigint, ClinicName:varchar(30))
- Rooms(RoomId:bigint, RoomName:varchar(30), DoctorPersonId:bigint, ClinicId:bigint)
- Appointments(AppoinmentId:bigint, AppoinmentDate:timestamp,Description:VARCHAR(60), PatientPersonId:bigint, DoctorPersonId:bigint)
- Prescriptions(PrescriptionId:bigint, PatientPersonId:bigint, DoctorPersonId:bigint)
- PrescriptionDrugs(Prescription:bigint, Drug:bigint)
- Drugs(DrugId:bigint, DrugName:varchar(50), DrugType:varchar(30))

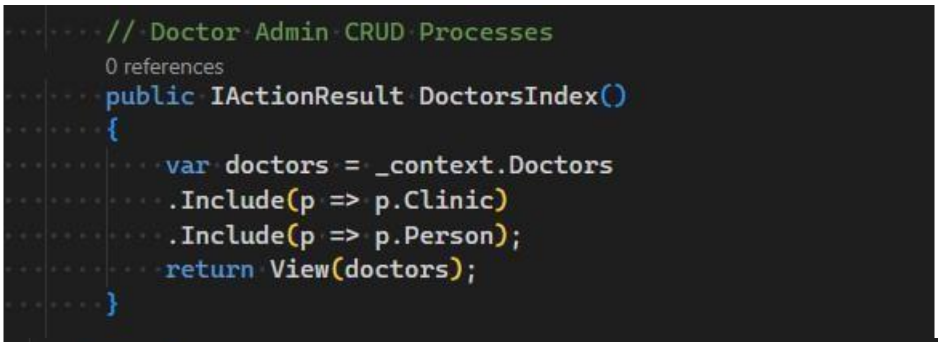
SQLstatements to create the database with the data in it

```
    -- Creation Processes


CREATE TABLE IF NOT EXISTS "__EFMigrationsHistory" (
        "MigrationId" character varying(150) NOT NULL,
        "ProductVersion" character varying(32) NOT NULL,
        CONSTRAINT "PK___EFMigrationsHistory" PRIMARY KEY ("MigrationId")
    );

CREATE TABLE "Clinics" (
        "ClinicId" bigint GENERATED BY DEFAULT AS IDENTITY,
        "ClinicName" VARCHAR(30) NOT NULL,
        CONSTRAINT "PK_Clinic" PRIMARY KEY ("ClinicId")
    );

CREATE TABLE "Drugs" (
        "DrugId" bigint GENERATED BY DEFAULT AS IDENTITY,
        "DrugName" VARCHAR(50) NOT NULL,
        "DrugType" VARCHAR(30) NOT NULL,
        CONSTRAINT "PK_Drug" PRIMARY KEY ("DrugId")
    );

CREATE TABLE "Persons" (
        "PersonId" bigint GENERATED BY DEFAULT AS IDENTITY,
        "Name" VARCHAR(30) NOT NULL,
        "Email" VARCHAR(50) NOT NULL,
        "PhoneNumber" text NOT NULL,
        doctor boolean NOT NULL,
        patient boolean NOT NULL,
        nurse boolean NOT NULL,
        CONSTRAINT "PK_Person" PRIMARY KEY ("PersonId")
    );

CREATE TABLE "Doctors" (
        "PersonId" bigint NOT NULL,
        "Specialty" VARCHAR(30) NOT NULL,
        "ClinicId" bigint NOT NULL,
        CONSTRAINT "PK_Doctor" PRIMARY KEY ("PersonId"),
        CONSTRAINT "FK_Doctor_Person_PersonId" FOREIGN KEY ("PersonId") REFERENCES
"Persons" ("PersonId"),
        CONSTRAINT "FK_Doctor_Clinic_ClinicId" FOREIGN KEY ("ClinicId") REFERENCES
"Clinics" ("ClinicId")
    );

CREATE TABLE "Patients" (
        "PersonId" bigint NOT NULL,
        "MedicalHistory" VARCHAR(100) NOT NULL,
        CONSTRAINT "PK_Patient" PRIMARY KEY ("PersonId"),
        CONSTRAINT "FK_Patient_Person_PersonId" FOREIGN KEY ("PersonId") REFERENCES
"Persons" ("PersonId")
    );

CREATE TABLE "Nurses" (
        "PersonId" bigint NOT NULL,
        "DoctorPersonId" bigint NOT NULL,
        CONSTRAINT "PK_Nurse" PRIMARY KEY ("PersonId"),
        CONSTRAINT "FK_Nurse_Person_PersonId" FOREIGN KEY ("PersonId") REFERENCES
"Persons" ("PersonId"),
        CONSTRAINT "FK_Nurse_Doctor_DoctorPersonId" FOREIGN KEY ("DoctorPersonId")
REFERENCES "Doctors" ("PersonId")
    );

CREATE TABLE "Rooms" (
        "RoomId" bigint GENERATED BY DEFAULT AS IDENTITY,
        "RoomName" VARCHAR(30) NOT NULL,
        "DoctorPersonId" bigint NOT NULL,
        "ClinicId" bigint NOT NULL,
        CONSTRAINT "PK_Room" PRIMARY KEY ("RoomId"),
        CONSTRAINT "FK_Room_Clinic_ClinicId" FOREIGN KEY ("ClinicId") REFERENCES
"Clinics" ("ClinicId"),
        CONSTRAINT "FK_Room_Doctor_DoctorPersonId" FOREIGN KEY ("DoctorPersonId")
REFERENCES "Doctors" ("PersonId")
    );

CREATE TABLE "Appointments" (
        "AppointmentId" bigint GENERATED BY DEFAULT AS IDENTITY,
        "AppointmentDate" timestamp with time zone NOT NULL,
        "Description" VARCHAR(60) NOT NULL,
        "PatientPersonId" bigint NOT NULL,
        "DoctorPersonId" bigint NOT NULL,
        CONSTRAINT "PK_Appointment" PRIMARY KEY ("AppointmentId"),
        CONSTRAINT "FK_Appointment_Doctor_DoctorPersonId" FOREIGN KEY ("DoctorPersonId")
REFERENCES "Doctors" ("PersonId"),
        CONSTRAINT "FK_Appointment_Patient_PatientPersonId" FOREIGN KEY ("PatientPersonId")
REFERENCES "Patients" ("PersonId")
    );

CREATE TABLE "Prescriptions" (
        "PrescriptionId" bigint GENERATED BY DEFAULT AS IDENTITY,
        "DoctorPersonId" bigint NOT NULL,
        "PatientPersonId" bigint NOT NULL,
        CONSTRAINT "PK_Prescription" PRIMARY KEY ("PrescriptionId"),
        CONSTRAINT "FK_Prescription_Doctor_DoctorPersonId" FOREIGN KEY ("DoctorPersonId")
REFERENCES "Doctors" ("PersonId"),
        CONSTRAINT "FK_Prescription_Patient_PatientPersonId" FOREIGN KEY ("PatientPersonId")
REFERENCES "Patients" ("PersonId")
    );

CREATE TABLE "PrescriptionDrugs" (
        prescription bigint NOT NULL,
        drug bigint NOT NULL,
        CONSTRAINT "PK_PrescriptionDrug" PRIMARY KEY (prescription, drug),
        CONSTRAINT "FK_PrescriptionDrug_Drug_drug" FOREIGN KEY (drug) REFERENCES
"Drugs" ("DrugId"),
        CONSTRAINT "FK_PrescriptionDrug_Prescription_prescription" FOREIGN KEY (prescription)
REFERENCES "Prescriptions" ("PrescriptionId")
    );


                    -- INSERTION PROCESSES

-- Clinics
INSERT INTO "Clinics" ("ClinicName") VALUES
('General Medicine Clinic'),
('Pediatrics Clinic'),
('Cardiology Center'),
('Dermatology Clinic'),
```

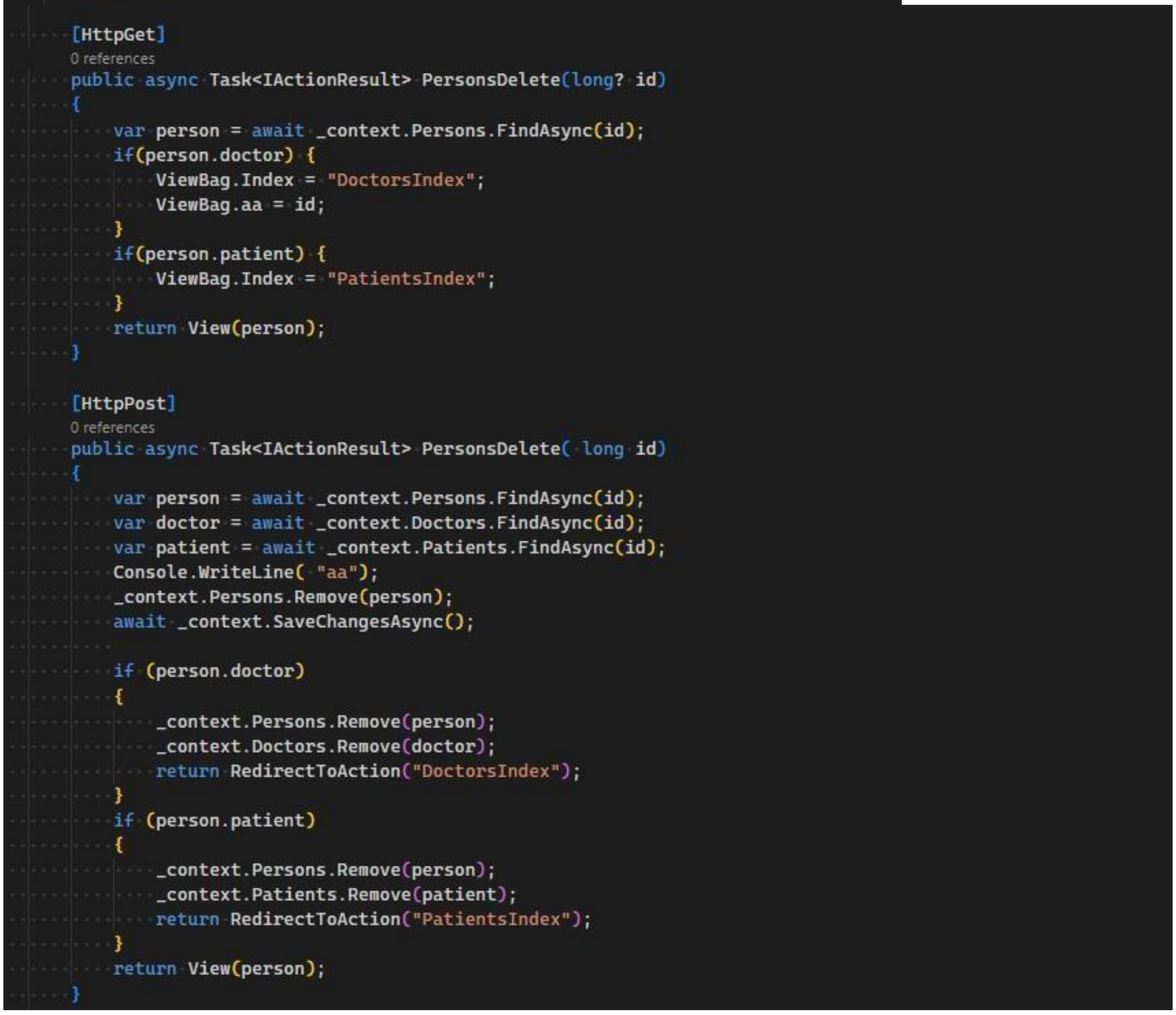
```sql
('Neurology Institute'),
('Orthopedic Hospital'),
('Oncology Department'),
('ENT Clinic'),
('Psychiatry Clinic'),
('Gastroenterology Unit');

-- Drugs
INSERT INTO "Drugs" ("DrugName", "DrugType") VALUES
('Drug 5', 'Tablet'),
('Drug 9', 'Injection'),
('Drug 10', 'Syrup'),
('Drug 8', 'Capsule'),
('Drug 5', 'Tablet'),
('Drug 1', 'Tablet'),
('Drug 3', 'Injection'),
('Drug 3', 'Syrup'),
('Drug 8', 'Capsule'),
('Drug 9', 'Injection');

-- Persons
INSERT INTO "Persons" ("Name", "Email", "PhoneNumber", doctor, patient, nurse) VALUES
('Person 1', 'person1@example.com', '1234567891', True, False, False),
('Person 2', 'person2@example.com', '1234567892', True, False, False),
('Person 3', 'person3@example.com', '1234567893', True, False, False),
('Person 4', 'person4@example.com', '1234567894', False, False, True),
('Person 5', 'person5@example.com', '1234567895', False, True, False),
('Person 6', 'person6@example.com', '1234567896', True, False, False),
('Person 7', 'person7@example.com', '1234567897', False, False, True),
('Person 8', 'person8@example.com', '1234567898', False, False, True),
('Person 9', 'person9@example.com', '1234567899', False, False, True),
('Person 10', 'person10@example.com', '12345678910', False, False, True);

-- Doctors
INSERT INTO "Doctors" ("PersonId", "Specialty", "ClinicId") VALUES
(21, 'Dermatology', 7),
(22, 'Dermatology', 7),
(23, 'Dermatology', 10),
(24, 'Pediatrics', 7),
(25, 'Dermatology', 10);

-- Patients
INSERT INTO "Patients" ("PersonId", "MedicalHistory") VALUES
(26, 'History 9'),
(27, 'History 1'),
(28, 'History 2'),
(29, 'History 9'),
(30, 'History 7');

-- Nurses
INSERT INTO "Nurses" ("PersonId", "DoctorPersonId") VALUES
(6, 5),
(7, 1),
(8, 4),
(9, 1),
(10, 2);

-- Rooms
INSERT INTO "Rooms" ("RoomName", "DoctorPersonId", "ClinicId") VALUES
('Room 3', 5, 7),
('Room 4', 4, 5),
('Room 4', 5, 10),
('Room 8', 3, 7),
('Room 3', 2, 3),
('Room 2', 3, 4),
('Room 3', 5, 3),
('Room 4', 2, 8),
('Room 4', 1, 4),
('Room 3', 3, 7);

-- Appointments
INSERT INTO "Appointments" ("AppointmentDate", "Description", "PatientPersonId",
"DoctorPersonId") VALUES
(NOW() + INTERVAL '1 days', 'Description 4', 7, 3),
(NOW() + INTERVAL '2 days', 'Description 6', 10, 3),
(NOW() + INTERVAL '3 days', 'Description 2', 7, 3),
(NOW() + INTERVAL '4 days', 'Description 10', 10, 5),
(NOW() + INTERVAL '5 days', 'Description 7', 7, 4),
(NOW() + INTERVAL '6 days', 'Description 9', 9, 1),
(NOW() + INTERVAL '7 days', 'Description 6', 8, 3),
(NOW() + INTERVAL '8 days', 'Description 6', 10, 1),
(NOW() + INTERVAL '9 days', 'Description 2', 9, 3),
(NOW() + INTERVAL '10 days', 'Description 7', 7, 5);

-- Prescriptions
INSERT INTO "Prescriptions" ("DoctorPersonId", "PatientPersonId") VALUES
(1, 6),
(3, 9),
(4, 9),
(5, 9),
(1, 7),
(1, 9),
(3, 6),
(4, 6),
(3, 7),
(3, 10);

-- PrescriptionDrugs
INSERT INTO "PrescriptionDrugs" (prescription, drug) VALUES
(15, 14),
(12, 17),
(10, 15),
(19, 16),
(16, 14),
(17, 13),
(11, 17),
(12, 18),
(12, 19),
(15, 15);



// UPDATE Process



-- TRIGGERS

-- TRIGER 1
```

```
        -- Triger TABLE

CREATE TABLE "AppointmentLog" (
   "LogId" SERIAL PRIMARY KEY,
   "AppointmentId" bigint NOT NULL,
   "LogMessage" text NOT NULL,
   "LogTime" timestamp NOT NULL DEFAULT NOW()
);

CREATE OR REPLACE FUNCTION log_appointment()
RETURNS TRIGGER AS $$
BEGIN
   INSERT INTO "AppointmentLog" ("AppointmentId", "LogMessage")
   VALUES (NEW."AppointmentId", 'New appointment added.');
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_appointment_insert
BEFORE INSERT ON "Appointments"
FOR EACH ROW
EXECUTE FUNCTION log_appointment();


INSERT INTO "Persons" ("Name", "Email", "PhoneNumber", doctor, patient, nurse) VALUES
('Person 11', 'person12@example.com', '1234567891', False, True, False);

INSERT INTO "Persons" ("Name", "Email", "PhoneNumber", doctor, patient, nurse) VALUES
('Person 12', 'person13@example.com', '1234567891', True, False, False);


-- Patient
INSERT INTO "Patients" ("PersonId", "MedicalHistory") VALUES
(13, 'History 11');

-- Doctor
INSERT INTO "Doctors" ("PersonId", "Specialty", "ClinicId") VALUES
(14, 'Dermatology', 7);

-- Add a new appointment
INSERT INTO "Appointments" ("AppointmentDate", "Description", "PatientPersonId",
"DoctorPersonId")
VALUES (NOW() + INTERVAL '2 days', 'Routine check-up', 13, 14);

-- Log tablosundaki kaydı kontrol et
SELECT * FROM "AppointmentLog";




-- TRIGER 2

CREATE OR REPLACE FUNCTION cascade_delete_patient_appointments()
RETURNS TRIGGER AS $$
BEGIN
   DELETE FROM "Appointments" WHERE "PatientPersonId" = OLD."PersonId";
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

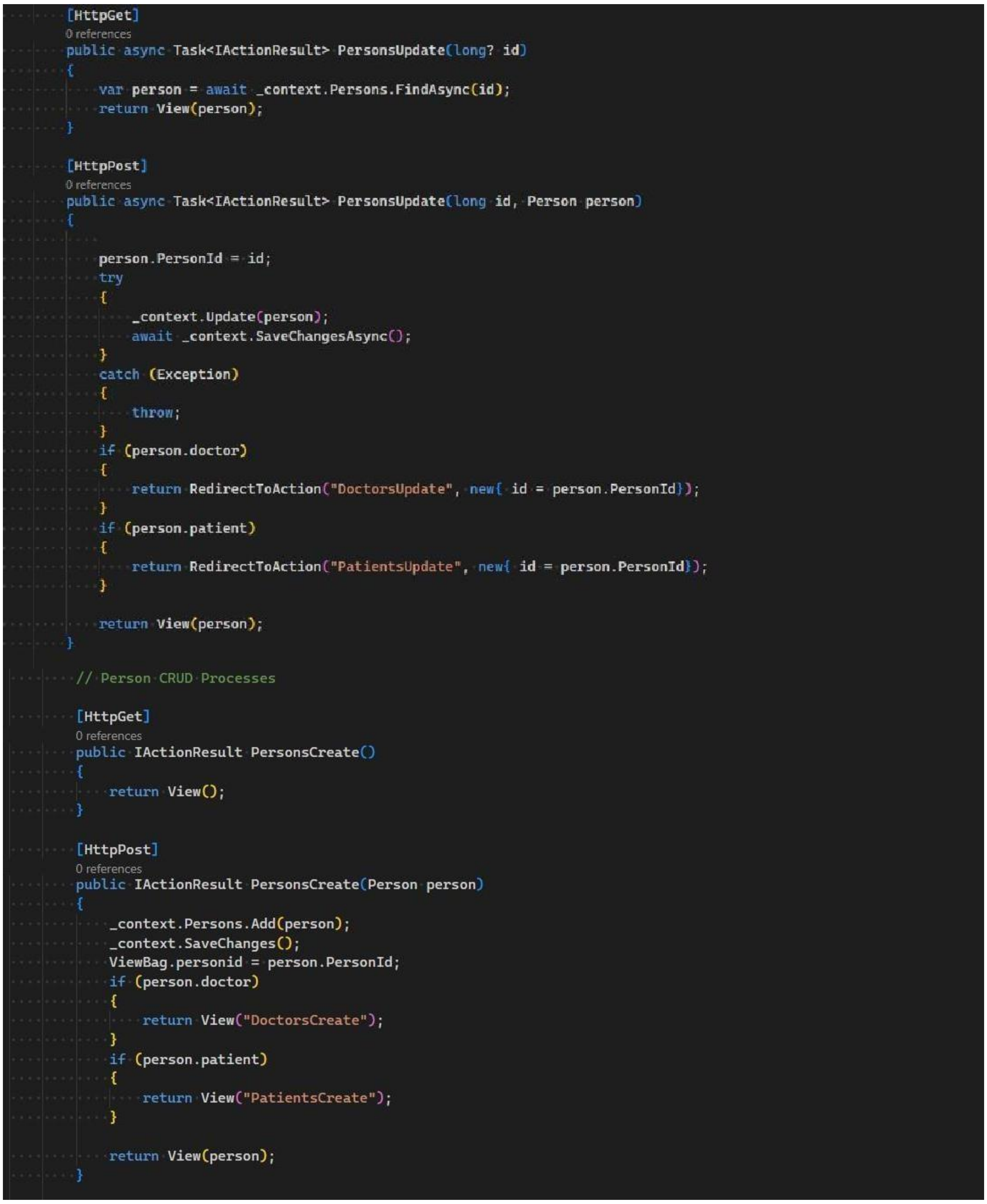
```
CREATE TRIGGER after_patient_delete
BEFORE DELETE ON "Patients"
FOR EACH ROW
EXECUTE FUNCTION cascade_delete_patient_appointments();

-- Let's try to delete the "Patients"
DELETE FROM "Persons" WHERE "PersonId" = 3;
DELETE FROM "Patients" WHERE "PersonId" = 3;



-- TRIGER 3


        -- Triger TABLE

CREATE TABLE "DoctorNameLog" (
  "LogId" SERIAL PRIMARY KEY,
  "DoctorPersonId" bigint NOT NULL,
  "OldName" VARCHAR(50) NOT NULL,
  "NewName" VARCHAR(50) NOT NULL,
  "ChangeTime" timestamp NOT NULL DEFAULT NOW()
);


CREATE OR REPLACE FUNCTION log_doctor_name_changes()
RETURNS TRIGGER AS $$
BEGIN
  -- If the name has changed, save the old and new names to the log table.
  IF NEW."Name" IS DISTINCT FROM OLD."Name" THEN
    INSERT INTO "DoctorNameLog" ("DoctorPersonId", "OldName", "NewName", "ChangeTime")
    VALUES (OLD."PersonId", OLD."Name", NEW."Name", NOW());
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER doctor_name_update_trigger
BEFORE UPDATE ON "Persons"
FOR EACH ROW
WHEN (OLD."doctor" = TRUE AND NEW."doctor" = TRUE)
EXECUTE FUNCTION log_doctor_name_changes();

-- Let's update
UPDATE "Persons" SET "Name" = 'Memati' WHERE "PersonId" = 3;

-- Let's check the log table
SELECT * FROM "DoctorNameLog";




-- TRIGER 4

        -- Triger TABLE

CREATE TABLE "ClinicReport" (
  "ReportId" SERIAL PRIMARY KEY,
  "ClinicId" bigint NOT NULL,
    "ClinicName" VARCHAR(50) NOT NULL,
    "LastUpdated" timestamp NOT NULL DEFAULT NOW()
);



CREATE OR REPLACE FUNCTION update_clinic_reports()
RETURNS TRIGGER AS $$
BEGIN
    -- Update report table if clinic name or information has changed
    IF NEW."ClinicName" IS DISTINCT FROM OLD."ClinicName" THEN
        UPDATE "ClinicReport"
        SET "ClinicName" = NEW."ClinicName",
            "LastUpdated" = NOW()
        WHERE "ClinicId" = NEW."ClinicId";

    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER clinic_update_trigger
BEFORE UPDATE ON "Clinics"
FOR EACH ROW
EXECUTE FUNCTION update_clinic_reports();


-- Let's update the clinic name
UPDATE "Clinics" SET "ClinicName" = 'City Health Center' WHERE "ClinicId" = 6;

-- Let's check the clinical report table
SELECT * FROM "ClinicReport";



-- FUNCTIONS

-- Function 1

CREATE OR REPLACE FUNCTION get_patients_by_doctor(doctor_id BIGINT)
RETURNS TABLE (PersonId BIGINT, PersonName TEXT, MedicalHistory TEXT) AS $$
BEGIN
    RETURN QUERY
    SELECT p."PersonId", p."Name", pa."MedicalHistory"
    FROM "Persons" p
    JOIN "Patients" pa ON p."PersonId" = pa."PersonId"
    JOIN "Appointments" a ON a."PatientPersonId" = pa."PersonId"
    WHERE a."DoctorPersonId" = doctor_id;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_patients_by_doctor(3); -- 3 numaralı doktora ait hastalar


-- Function 2

CREATE OR REPLACE FUNCTION get_rooms_by_clinic(clinic_id BIGINT)
RETURNS TABLE (RoomName TEXT, DoctorName TEXT) AS $$
BEGIN
    RETURN QUERY
    SELECT r."RoomName", p."Name" AS DoctorName
    FROM "Rooms" r
    JOIN "Doctors" d ON r."DoctorPersonId" = d."PersonId"
    JOIN "Persons" p ON d."PersonId" = p."PersonId"
    WHERE r."ClinicId" = clinic_id;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_rooms_by_clinic(7); -- 7 numaralı kliniğin odaları


-- Function 3

CREATE OR REPLACE FUNCTION get_drugs_by_patient(patient_id BIGINT)
RETURNS TABLE (DrugName TEXT, PrescriptionId BIGINT) AS $$
BEGIN
    RETURN QUERY
    SELECT d."DrugName", pd."prescription" AS PrescriptionId
    FROM "PrescriptionDrugs" pd
    JOIN "Prescriptions" p ON pd."prescription" = p."PrescriptionId"
    JOIN "Drugs" d ON pd."drug" = d."DrugId"
    WHERE p."PatientPersonId" = patient_id;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_drugs_by_patient(6); -- 6 numaralı hastanın ilaçları


-- Function 4

CREATE OR REPLACE FUNCTION get_appointment_count_by_doctor(doctor_id BIGINT)
RETURNS INT AS $$
DECLARE
    appointment_count INT;
BEGIN
    SELECT COUNT(*)
    INTO appointment_count
    FROM "Appointments"
    WHERE "DoctorPersonId" = doctor_id;

    RETURN appointment_count;
END;
$$ LANGUAGE plpgsql;

SELECT get_appointment_count_by_doctor(3); -- 3 numaralı doktorun toplam randevuları
```



```
// Doctor Admin CRUD Processes
0 references
public IActionResult DoctorsIndex()
{
    var doctors = _context.Doctors
    .Include(p => p.Clinic)
    .Include(p => p.Person);
    return View(doctors);
}
```



```
[HttpGet]
0 references
public async Task<IActionResult> PersonsDelete(long? id)
{
    var person = await _context.Persons.FindAsync(id);
    if(person.doctor) {
        ViewBag.Index = "DoctorsIndex";
        ViewBag.aa = id;
    }
    if(person.patient) {
        ViewBag.Index = "PatientsIndex";
    }
    return View(person);
}

[HttpPost]
0 references
public async Task<IActionResult> PersonsDelete( long id)
{
    var person = await _context.Persons.FindAsync(id);
    var doctor = await _context.Doctors.FindAsync(id);
    var patient = await _context.Patients.FindAsync(id);
    Console.WriteLine( "aa");
    _context.Persons.Remove(person);
    await _context.SaveChangesAsync();

    if (person.doctor)
    {
        _context.Persons.Remove(person);
        _context.Doctors.Remove(doctor);
        return RedirectToAction("DoctorsIndex");
    }
    if (person.patient)
    {
        _context.Persons.Remove(person);
        _context.Patients.Remove(patient);
        return RedirectToAction("PatientsIndex");
    }
    return View(person);
}
```



```
        [HttpGet]
        0 references
        public async Task<IActionResult> PersonsUpdate(long? id)
        {
            var person = await _context.Persons.FindAsync(id);
            return View(person);
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> PersonsUpdate(long id, Person person)
        {

            person.PersonId = id;
            try
            {
                _context.Update(person);
                await _context.SaveChangesAsync();
            }
            catch (Exception)
            {
                throw;
            }
            if (person.doctor)
            {
                return RedirectToAction("DoctorsUpdate", new{ id = person.PersonId});
            }
            if (person.patient)
            {
                return RedirectToAction("PatientsUpdate", new{ id = person.PersonId});
            }

            return View(person);
        }

         // Person CRUD Processes

         [HttpGet]
         0 references
         public IActionResult PersonsCreate()
         {
             return View();
         }

         [HttpPost]
         0 references
         public IActionResult PersonsCreate(Person person)
         {
             _context.Persons.Add(person);
             _context.SaveChanges();
             ViewBag.personid = person.PersonId;
             if (person.doctor)
             {
                 return View("DoctorsCreate");
             }
             if (person.patient)
             {
                 return View("PatientsCreate");
             }

             return View(person);
         }
```