Universitatea Tehnică Cluj-Napoca

Facultatea de Automatică și Calculatoare

Departamentul Calculatoare

Proiect

la disciplina

Introducere în Baze de Date

Lanț de Policlinici

Studenți: Angheluș Diana

Jișa Diana

Grupa: 30224

An academic: 2022 - 2023

1 Cuprins

2 Introducere

Proiectul vizează dezvoltarea unei aplicații pentru gestionarea unei rețele de policlinici prin intermediul unei baze de date. Scopul aplicației este simplificarea accesului și manipularii bazei de date prin intermediul unei interfețe grafice care poate fi utilizata de angajați. Aplicația permite crearea de programari pentru servicii medicale specifice unui medic competent și genereaza bonuri fiscale.

Exista mai multe tipuri de utilizatori, inclusiv administratori și super administratori care au acces total la date și angajați care au acces limitat la date, în funcție de tipul lor. Exista, de asemenea, trei departamente care se ocupă de resurse umane, aspecte financiare și contabile si aspecte medicale.

Obiectivele proiectului sunt eficientizarea operațiunilor policlinicii, concurenta operațiilor, securitatea datelor, evitarea evaziunii fiscale și reducerea consumului de hartie și materiale plastice.

Pentru dezvoltarea proiectului au fost folosite:

- **MySQL Workbench 6.2** - pentru crearea bazei de date, popularea initiala, dezvltarea de viewuri, proceduri si triggere si pentru crearea diagramei UML a tabelelor
- **Eclipse/Intelij** – mediu de dezvoltare Java

3 Analiza cerintelor utilizatorilor

3.1 Ipotezele policlinicii (cerinte si constrangeri)

Aplicatia gestioneaza actiunile angajatilor dintr-o unitate, aceasta este utilizeaza o baza de date care este supusa urmatoarelor cerinte:

- Exista 3 tipuri de utilizatori: angajati, administratori si super-administratori
- Un utilizator este unic identificat prin cnp-ul sau, pentru acesta mai retinem nume, prenume, adresa, număr de telefon, email, cont IBAN, numărul de contract, data angajării, funcţia deţinută în cadrul lanțului de policlinici
- Exista mai multe unitati medicale in cadrul lantului de policlinici care se identifica unic prin nume si mai contin adresa si id-ul orarului lor
- Exista mai multe tipuri de functii: inspectori de resurse umane, experti financiari, receptioneri, asistenti medicali si medici
- Pentru un **asistent medical** se va reține suplimentar tipul [1]și gradul[2].
- Pentru un **medic** se va reține suplimentar specialitatea sau specialitățile în care își desfășoară activitatea, gradul[3], codul de parafă, competențele pe care le deține pentru realizarea unor proceduri ce necesită acreditări speciale[4], titlul științific[5]. Totodată,

1
2
3
4
5

fiecare medic are negociat un procent din serviciile medicale realizate care îi revine, adiţional faţă de salariul negociat.

- Lanțul de policlinici oferă pacienților un set de **servicii medicale** specifice pentru fiecare specialitate în parte. Pentru un serviciu medical disponibil se va specifica specialitatea din care face parte, necesitatea existenței unei competențe a medicului care o efectuează, prețul asociat și durata (exprimată în minute).
- Sistemul va fi format din mai multe module care vor putea fi accesate de angajaţi în funcţie de drepturile pe care le deţin. Astfel, vor fi implementate un
  - **modul pentru gestiunea resurselor umane** ce vizează gestiunea programului de lucru şi al concediilor angajaţilor,
  - un **modul pentru operaţii financiar-contabile** care determină profitul operaţional ca diferenţă între venituri (sume încasate pentru serviciile medicale) şi cheltuieli (plăţi efectuate către angajaţi aferente salariilor) şi un
  - **modul pentru gestiunea activităților operaționale (programarea** pacienților pentru servicii medicale și înregistrarea acestora în momentul în care se prezintă în clinica medicală, emiterea bonului fiscal de către recepționeri, completarea rapoartelor medicale de către asistenții medicali și medici)

3.2 Organizarea structurata tabelar a cerintelor utilizator

Baza de date trebuie sa stocheze urmatoarele informatii:

- Unitatile medicale
- Programele unitatilor medicale
- Utilizatorii
- Orarele generice ale utilizatorilor
- Orarele specifice ale utilizatorilor
- Concediile utilizatorilor
- Asistentii medicali
- Medicii
- Specialitatile medicilor
- Competentele medicilor
- Serviciile medicale pe care le ofera un medic
- Programarile
- Rapoartele create pentru programarile indeplinite
- Istoricele pacientilor
- Cabinetele
-

Baza de date trebuie sa permita si urmatoarele operatii:

- prelucrarea informatiilor din baza de date
- Afisarea orarului lunar
- Afisarea orarului saptamanal
- Programarea de concedii
- Efectuarea unei programari
- Calcularea salariului (in functie de programari daca este medic)

- Calcularea profitului care il aduce o specializare
- Calcularea profitului pe care il aduce un medic la o unitate
- Calcularea profitului policlinicii

3.3 Determinarea si caracterizarea profilurilor de utilizator

Avem 2 tipuri de utilizatori

1) Administratori si super administratori
    - Autentificare in aplicatia
    - Introducere de noi utilizatori
    - Schimbarea datelor din baza de date
2) Angajati care sunt de 5 tipuri
    I. Inspectori de resurse umane:
        - Setare concediu pentru o persoana
        - Afisare orar lunar tratându-se concediile
        - Afisare orar saptamanal - orar specific daca e
        - Afisare concedii intr-o saptamana
        - Afisare date angajati

    II. Experti financiari
        - Afisare orar saptamanal - orar specific daca e
        - Afisare concedii intr-o saptamanal
        - Calculare profit policlinica intr-o luna = venituri(in urma serviciilor medicale)- cheltuieli(salariile cumulate + COMISIOANELE MEDICILOR)
        - Calculare salariu angajat in functie de numarul de ore din contract
        - Calculare salariu medic in functie de servicii
        - Profitul adus de medic in functie de policlinica
        - Profitul adus de o specialitate
        - Afisare orar lunar pentru cei din dep medical tratandu-se concediile

    III. Receptioneri
        - Emitere bon fiscal
        - Creare programare
        - Programare pacient (numai in data calendaristica ulterioara/curenta, pentru un medic, selectand un serviciu)

    IV. Asistenti medicali
        Completare rapoarte

    V. Medic
        - Completare raport
        - Completare investigatie
        - Parafare raport
        - Modificare servicii medicale (proprii)

4 Modelul de date si descrierea acestuia

4.1 Entitatile

Unitati_medicale ofera informatii despre fiecare unitate medicala.

Programe retine informatii despre programul fiecarei policlinici.

Utilizator ofera informatii despre fiecare utilizator. Este legat de asistenti_medicali (fiecare asistent medical este un utilizator), medici (fiecare medic este un utilizator), orare_generice (fiecare utilizator are mai multe orare generice, unul pentru fiecare zi, iar daca este medic, poate avea la mai multe unitati, numai sa nu se suprapuna), orare_specifice(un orar specific este pe o data, deci un utilizator poate avea mai multe orare specifice), concedii (orice utilizator poate avea mai multe concedii).

Orare_generice ofera informatii depsre orarul unui utilizator intr-o zi a saptamanii.

Orare_specifice ofera informatii despre orarul spcific al unui utilizator pentru o anumita data.

Concedii ofera informatii despre concediile unui utilizator.

Asistenti_medicali ofera informatii suplimentare pentru utilizatorii care au functia de asistent medical.

Medici ofera informatii suplimentare despre utilizatorii care sunt si medici. t.

Specialitati ofera informatii despre specialitatile unui medic. Pentru a rezolva legatura one to many la medici, si pentru a asigura forma de normalizare bnfc am adaugat cheia id_specialitate cheie primara auto_increment, la fel cum am facut si la urmatoarele tabele si cateva anterioare.

Competente ofera informatii despre competentele unui medic.

Servicii_medicale ofera informatii despre serviciile medicale pe care le poate face un medic in functie de specialitatile si competentele sale. Deoarece serviciile medicale sunt legate de medic, fiecare medic poate avea alte preturi sau o durata diferita pentru un anumit serviciu.

Programari_pacient ofera informatii despre o programare, aceasta entitate fiind legata de entitatea medici (one to many, un medic poate face mai multe programari)..

Istorice ofera informatii depsre istoricul unui pacient.

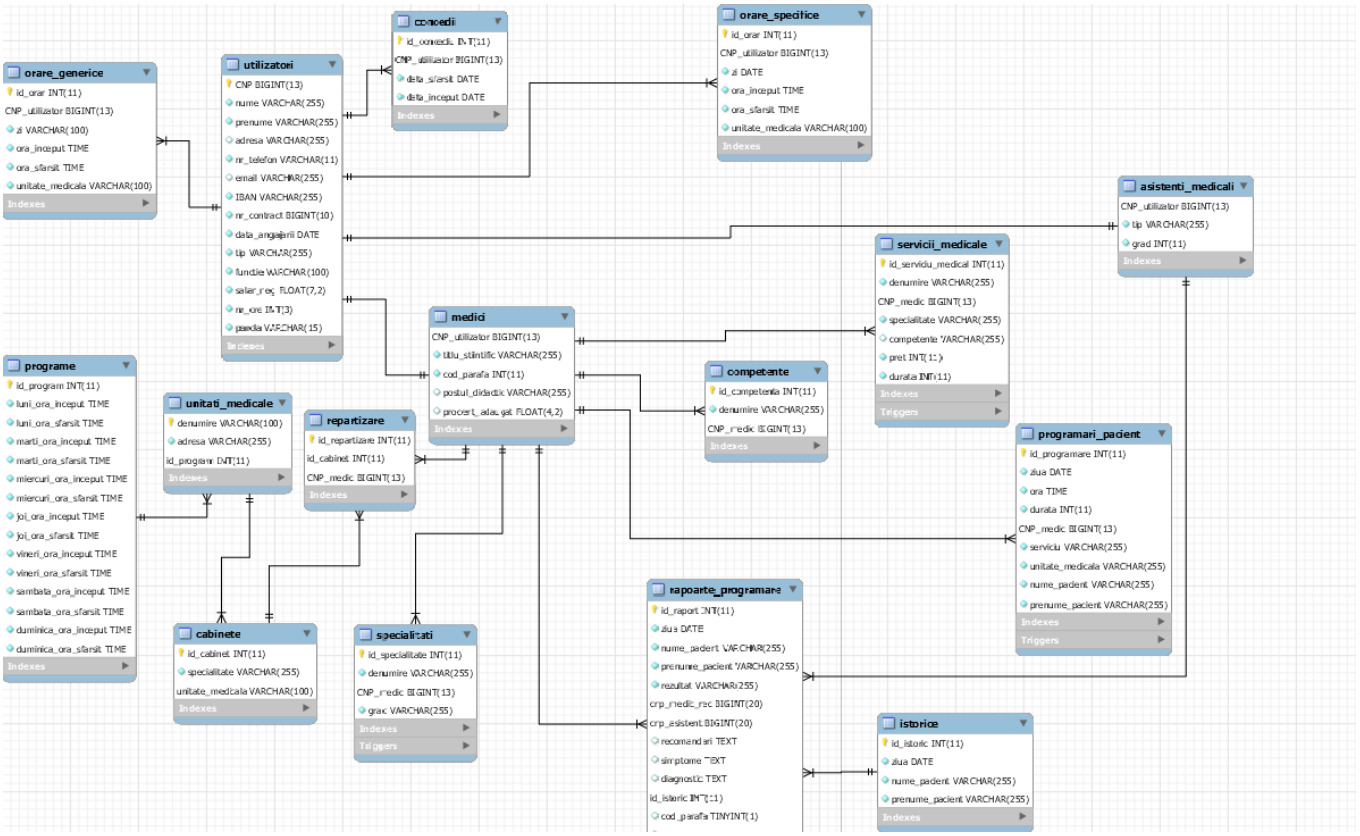
Rapoarte_programare prezinta raportul intocmit de medic si asistent medical dupa ce s-a incheiat o programare. Acesta este creat in momentul in care se face o programare, iar, dupa

ce aceatsa a fost facuta se va completa de asistent si parafa de medic. Este legata de entitatea medic, asistent si istorice.

Cabinete ofera informatii despre cabinetele de la o unitate medicala, noi considerand ca acestea ar trebui sa fie in functie de o anumita specialitate (cabinetul de cardiologie, cabinetul de neurologie etc.) Este legat de Unitati_medicale (one to many, o unitate medicala poate avea mai multe cabinete.

Repartizare este un tabel creat pentru a rezolva relaia many to many intre medici s cabinete. Cum un medic poat lucra la mai multe unitati medicala si poate avea mai multe specialitati, acesa poate fi la mai multe cabinete daca doreste, si un cabinet poate avea mai multi medici, deci rezulta o relaie many to many.

4.2 Diagrama EER/UML pentru modelul de date complet



4.3 Proceduri, funcții, triggere si view-uri

Proceduri:

1. **stabilire_concediu** (Nume VARCHAR(255), Prenume VARCHAR(255), data_inc DATE, data_sf DATE). Adauga un concediu, daca se poate, pentru un utilizator.
2. **add_new_orar_generic**(CNP_util BIGINT(13), zi VARCHAR(100), ora_inceput TIME, ora_sfarsit TIME, unitate_medicala VARCHAR(100), OUT Rezultat INT) Verifica daca se poate adauga un orar generic pentru un utilizator: daca nu se

suprapune peste un alt orar generic si daca este incadrat in programul de lucru al policlinicii respective.

3. **logare** (username BIGINT(13), parola VARCHAR(15), OUT Rezultat INT). Logheaza un utilizator in contul sau.
4. **completare_raport** (CNP_cadru_medical BIGINT(13), Raport INT, Campul VARCHAR(255), Textul TEXT, OUT Rezultat INT). Completeaza un raport si verifica daca se poate completa astfel (de exemplu un asistent nu va putea sa parafeze un raport.
5. **stergere_serviciu** (CNP_cadru_medical BIGINT(13), Serviciu VARCHAR(255), OUT Rezultat INT). Sterge un serviciu al unui medic.
6. **adaugare_serviciu**(CNP_cadru_medical BIGINT(13), Serviciu VARCHAR(255), Specialitate VARCHAR(255), Competenta VARCHAR(255), Pret INT, Durata INT, OUT Rezultat INT). Adauga un serviciu pentru un medic numai daca acesta are specializarea si competentele necesare.
7. **stergere_programare** (Programare INT, Nume VARCHAR(255), Prenume VARCHAR(255), OUT Rezultat INT). Sterge o programare daca pacientul nu s-a prezentat la programare.
8. **stabilire_ora_final** (ora_inceput TIME, durata INT, OUT ora_final TIME). Calculeaza ora de fnal a unei programari.
9. **creare_programare** (Nume_pacient VARCHAR(255), Prenume_pacient VARCHAR(255), Nume_medic VARCHAR(255), Prenume_medic VARCHAR(255), zi DATE, timpul TIME, serv VARCHAR(255), Unit VARCHAR(255), OUT Rezultat INT). Creaza o programare in functie de niste conditii impuse (daca doctorul ofera serviciul cerut, daca medicul nu are concediu sau are orar atunci, iar ora de final a programarii nu iese din orar).
10. **afisare_date_angajat** (Nume_angajat VARCHAR(255), Prenume_angajat VARCHAR (255), Func VARCHAR(100), OUT Rezultat INT). Afiseaza datele unui angajat.
11. **calculare_salariu** (Nume_angajat VARCHAR(255), Prenume_angajat VARCHAR(255), Luna INT, OUT Salariu BIGINT). Calculeaza salariul unui angajat, aceasta actiune facandu-se diferit in functie de functia utilizatorului, salariile medicilor se calculeaza in functie de programari, iar ale celorlalti angajati in functie de numarul de ore pe luna, daca a avut concedii.
12. **profit_medic_policlinica** (Nume_angajat VARCHAR(255), Prenume_angajat VARCHAR(255), Nume_policlinica VARCHAR(255), Luna INT, OUT Profit BIGINT). Calculeaza profitul pe care un medic il aduce unei policlinici in functie de programarile sale (suma adunata de pe salarii- suma*procentul sau).
13. **profit_specialitate_policlinica** (nume_specialitate VARCHAR(255), Nume_policlinica VARCHAR(255), Luna INT, OUT Profit BIGINT). Calculeaza profitul pe care o specializare il aduce unei policlinici.
14. **afisare_program_lunar**(Nume_angajat VARCHAR(255), Prenume_angajat VARCHAR(255), Nume_policlinica VARCHAR(255), Luna INT, An INT). afiseaza orarul lunar al unui utilizator.

15. **afisare_concedii** (Nume_angajat VARCHAR(255), Prenume_angajat VARCHAR(255)). Afiseaza concediile din saptamana curente ale unui angajat, tratandu-se concediile.
16. **afisare_orar_saptamanal** (Nume_angajat VARCHAR(255), Prenume_angajat VARCHAR(255), Nume_policlinica VARCHAR(255)). Afiseaza orarul saptamanal al unui utilizator (si orarul specific in loc de cel generic daca exista)
17. **profit_policlinica**(Nume_policlinica VARCHAR(255), Luna INT, An INT, OUT Salarii BIGINT). Calculeaza profitul unei policlinici intr-o luna ca si scaderea dintre profitul adus de programarile facute si suma salariilor tuturor angajatilor.
18. **stergere_utilizator** (Nume VARCHAR(255), Prenume VARCHAR(255), CNP_modificator BIGINT(13), OUT Rezultat INT). Sterge un utilizator.

Functii:

1. **ziua_saptamanii** (Ziua VARCHAR(255)). Primeste numele unei zile a saptamanii si returneaza a cata zi din saptamana este, duminica fiind prima zi.

View-uri:

1. **bon_fiscal** - arata bonul fiscal al unei programari in momentul in care aceasta a fost facuta.
2. **programati** – afiseaza toate programarile dintr-o zi.

Triggere:

1. **creare_istoric** – se creaza un istoric pentru un pacient dupa ce s-a introdus in tabela programari, dar numai dupa ce se verifica daca nu exista unul deja. Si de asemenea, se creaza automat un raport dupa ce se intoduce in programari
2. **stergere_raport** – dupa ce se sterge o programare se sterge si raportul ei
3. **validare_inserare_serviciu** – inainte de a se insera in serviciile unui medic se verifica daca acesta ae competentele si specializarile necesare, daca nu le are se fac toate atributele null si astfel nu se mai poate insera.
4. **repartizare_pe_cabinet** – dupa ce se atribuie o specializare unui medic, acesta va fi repartizat la toate cabinetele cu acea specializare de la unitatile medicale la care este angajat

4.4 Interogari MySql

SELECT i.nume_pacient FROM Istorice AS i WHERE nume_pacient LIKE i.nume_pacient -- vedem daca exista un istoric deja facut adica vom cauta numele si prenumele, este necesar pentru triggerul de inserare automaa in tabela istorice dupa ce s-a facut o programare

SELECT id_istoric FROM Istorice as i WHERE i.nume_pacient LIKE nume_pacient AND prenume_pacient LIKE i.prenume_pacient
-- gasim istoricul pacientului pentru a insera in rapoarte, este necesar in triggerul dupa inserarea in programari
SELECT id_specialitate FROM Specialitati AS s WHERE s.CNP_medic = new.CNP_medic AND denumire LIKE new.specialitate
-- cautam daca exista o anumita specialitate in lista de specialitati a doctorului pentru a putea vedea daca se poate adauga un serviciu

SELECT id_competenta FROM Competente as c WHERE c.CNP_medic = new.CNP_medic AND denumire LIKE new.competente
-- cautam daca exista o anumita competenta in lista de competente a doctorului pentru a putea vedea daca se poate adauga un serviciu

SELECT u.nume FROM Utilizatori AS u WHERE u.nume LIKE Nume LIMIT 1
-- vedem daca exista utilizatorul cu numele Nume, aceasta interogare se foloseste in foarte multe proceduri, similar exista una pentru prenume

SELECT COUNT(p.id_programare) FROM Programari_pacient AS p, Utilizatori AS u WHERE p.CNP_medic = u.CNP AND ziua BETWEEN data_inc AND data_sf
-- vedem cate programari are un medic intr-o perioada pentru a vedea daca poate primi concediu acea perioada

SELECT u.CNP FROM Utilizatori AS u WHERE u.nume LIKE Nume AND u.prenume LIKE Prenume
-- cautam CNP-ul utilizatorului cu numele si prenumele dat, aceasta interogare este folosita in foarte multe proceduri

SELECT id_program FROM unitati_medicale WHERE denumire like unitate_medicala
-- cautam id-ul programului unei unitati medicale, pentru a putea vedea daca orarul generic pe care vrem sa il adaugam este valid

SELECT COUNT(id_orar) FROM Orare_generice AS o WHERE CNP_util LIKE CNP_utilizator AND o.zi LIKE zi AND o.unitate_medicala LIKE unitate_medicala AND (@ora_i<o.ora_sfarsit OR @ora_s<o.ora_inceput)
-- verificam daca nu un utilizator are deja introdus un orar generic pe ziua respectiva care sa se suprapuna cu cel pe care vrem sa il introducem (pe ziua zi si intre orele @ora_i si @ora_s)

SELECT id_serviciu_medical FROM Servicii_medicale WHERE CNP_medic = CNP_cadru_medical AND denumire LIKE Serviciu
-- verificam daca exista un anumit serviciu medical pentru un medic

SELECT id_concediu FROM Concedii WHERE zi BETWEEN data_inceput AND data_sfarsit AND CNP_utilizator = @c

-- verificam daca un medic are concediu in o perioada care contine ziua ‚zi', iar daca are nu se poate face programare, cnp-ul medicului fiind retinut in @c

SELECT id_orar FROM Orare_specifice AS o WHERE @c = o.CNP_utilizator AND o.unitate_medicala LIKE Unit AND zi = o.zi AND timpul BETWEEN o.ora_inceput AND o.ora_sfarsit AND @final_prog <= o.ora_sfarsit
SELECT id_orar FROM Orare_generice AS o WHERE @c = o.CNP_utilizator AND o.unitate_medicala LIKE Unit AND o.zi LIKE 'luni' AND timpul BETWEEN o.ora_inceput AND o.ora_sfarsit AND @final_prog <= o.ora_sfarsit

-- verificam daca exista oroar specific care sa contina perioada in care se va face programare, daca nu cautam ziua in orarele generice

SELECT functie FROM Utilizatori WHERE nume LIKE Nume_angajat AND prenume LIKE Prenume_angajat
-- interogram functia unui angajat

SELECT SUM(TIME_TO_SEC(TIMEDIFF(ora_sfarsit, ora_inceput)) / 3600) FROM Orare_generice WHERE DAYOFWEEK(@data_incepere) = ziua_saptamanii(zi) AND CNP_utilizator = @c
-- cate ore ar fi lucrat un anumit angajat intr-o anumita zi

SELECT SUM(pret) FROM programari_pacient as p JOIN servicii_medicale as s WHERE AND s.CNP_medic = @c AND month(ziua) = Luna
-- calculam suma incasata de un doctor din programarile dintr-o luna

SELECT SUM(pret-pret*procent_adaugat/100) FROM programari_pacient as p JOIN servicii_medicale as s on p.serviciu like s.denumire JOIN medici as m on s.CNP_medic = m.CNP_utilizator WHERE s.specialitate like nume_specialitate AND p.unitate_medicala like Nume_policlinica AND month(p.ziua) = Luna
-- se calculeaza profitul unei unitati medicale din programarile facute pentru o specialitate intr-o luna

4.5 Cod MySql

4.5.1 Cod pentru crearea bazei de date si a tabelelor

CREATE DATABASE IF NOT EXISTS **Policlinici**;
USE Policlinici;

CREATE TABLE IF NOT EXISTS **Unitati_medicale**
(denumire VARCHAR(100) PRIMARY KEY NOT NULL,

```
adresa VARCHAR(255) NOT NULL,
id_program INT NOT NULL);

create table if not exists Programe
(id_program INT AUTO_INCREMENT PRIMARY KEY,
luni_ora_inceput TIME NOT NULL,
luni_ora_sfarsit TIME NOT NULL,
marti_ora_inceput TIME NOT NULL,
marti_ora_sfarsit TIME NOT NULL,
miercuri_ora_inceput TIME NOT NULL,
miercuri_ora_sfarsit TIME NOT NULL,
joi_ora_inceput TIME NOT NULL,
joi_ora_sfarsit TIME NOT NULL,
vineri_ora_inceput TIME NOT NULL,
vineri_ora_sfarsit TIME NOT NULL,
sambata_ora_inceput TIME NOT NULL,
sambata_ora_sfarsit TIME NOT NULL,
duminica_ora_inceput TIME NOT NULL,
duminica_ora_sfarsit TIME NOT NULL);

ALTER TABLE Unitati_medicale
ADD FOREIGN KEY (id_program) REFERENCES Programe(id_program);

CREATE TABLE IF NOT EXISTS Utilizatori
(CNP BIGINT(13) PRIMARY KEY NOT NULL,
nume VARCHAR(255) NOT NULL,
prenume VARCHAR(255) NOT NULL,
adresa VARCHAR(255),
nr_telefon VARCHAR(11) NOT NULL,
email VARCHAR(255),
IBAN VARCHAR(255) NOT NULL,
nr_contract BIGINT(10) NOT NULL,
data_angajarii DATE NOT NULL,
tip VARCHAR(255) NOT NULL,
functie VARCHAR(100) NOT NULL,
salar_neg FLOAT(7,2) NOT NULL,
nr_ore INT(3) NOT NULL,
parola VARCHAR(15) NOT NULL);

CREATE TABLE IF NOT EXISTS Orare_generice
(id_orar INT PRIMARY KEY AUTO_INCREMENT,
CNP_utilizator BIGINT(13) NOT NULL,
zi VARCHAR(100) NOT NULL,
ora_inceput TIME NOT NULL,
ora_sfarsit TIME NOT NULL,
unitate_medicala VARCHAR(100) NOT NULL,
FOREIGN KEY(CNP_utilizator) REFERENCES Utilizatori(CNP));

CREATE TABLE IF NOT EXISTS Orare_specifice
(id_orar INT PRIMARY KEY AUTO_INCREMENT,
CNP_utilizator BIGINT(13) NOT NULL,
zi DATE NOT NULL,
ora_inceput TIME NOT NULL,
ora_sfarsit TIME NOT NULL,
unitate_medicala VARCHAR(100) NOT NULL,
```

```
FOREIGN KEY(CNP_utilizator) REFERENCES Utilizatori(CNP));

CREATE TABLE IF NOT EXISTS Concedii
(id_concediu INT PRIMARY KEY AUTO_INCREMENT,
CNP_utilizator BIGINT(13) NOT NULL,
data_sfarsit DATE NOT NULL,
data_inceput DATE NOT NULL,
FOREIGN KEY(CNP_utilizator) REFERENCES Utilizatori(CNP));

CREATE TABLE IF NOT EXISTS Asistenti_medicali
(CNP_utilizator BIGINT(13) PRIMARY KEY NOT NULL,
tip VARCHAR(255) NOT NULL,
grad INT NOT NULL,
FOREIGN KEY (CNP_utilizator) REFERENCES Utilizatori(CNP));

CREATE TABLE IF NOT EXISTS Medici
(CNP_utilizator BIGINT(13) PRIMARY KEY NOT NULL,
titlu_stiintific VARCHAR(255) NOT NULL,
cod_parafa INT NOT NULL,
postul_didactic VARCHAR(255),
procent_adaugat FLOAT(4,2),
FOREIGN KEY (CNP_utilizator) REFERENCES Utilizatori(CNP));

CREATE TABLE IF NOT EXISTS Specialitati
(id_specialitate INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
denumire VARCHAR(255) NOT NULL,
CNP_medic BIGINT(13) NOT NULL,
grad VARCHAR(255) NOT NULL,
FOREIGN KEY (CNP_medic) REFERENCES Medici(CNP_utilizator));

CREATE TABLE IF NOT EXISTS Competente
(id_competenta INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
denumire VARCHAR(255) NOT NULL,
CNP_medic BIGINT(13) NOT NULL,
FOREIGN KEY (CNP_medic) REFERENCES Medici(CNP_utilizator));

CREATE TABLE IF NOT EXISTS Servicii_medicale
(id_serviciu_medical INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
denumire VARCHAR(255) NOT NULL,
CNP_medic BIGINT(13) NOT NULL,
specialitate VARCHAR(255) NOT NULL,
competente VARCHAR(255),
pret INT NOT NULL,
durata INT NOT NULL,
FOREIGN KEY (CNP_medic) REFERENCES Medici(CNP_utilizator));

CREATE TABLE IF NOT EXISTS Programari_pacient
(id_programare INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
ziua DATE NOT NULL,
ora TIME NOT NULL,
durata INT NOT NULL,
CNP_medic BIGINT(13) NOT NULL,
serviciu VARCHAR(255) NOT NULL,
unitate_medicala VARCHAR(255) NOT NULL,
nume_pacient varchar(255) NOT NULL,
```

```
prenume_pacient varchar(255) NOT NULL,
FOREIGN KEY (cnp_medic) REFERENCES Medici(CNP_utilizator));

CREATE TABLE IF NOT EXISTS Istorice
(id_istoric INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
ziua DATE NOT NULL,
nume_pacient varchar(255) NOT NULL,
prenume_pacient varchar(255) NOT NULL);

CREATE TABLE IF NOT EXISTS Rapoarte_programare
(id_raport INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
ziua DATE NOT NULL,
nume_pacient varchar(255) NOT NULL,
prenume_pacient varchar(255) NOT NULL,
rezultat VARCHAR(255) NOT NULL,
cnp_medic_rec BIGINT,
cnp_asistent BIGINT,
recomandari TEXT,
simptome TEXT,
diagnostic TEXT,
id_istoric INT NOT NULL,
cod_parafa BOOLEAN,
investigatie TEXT,
FOREIGN KEY (id_istoric) REFERENCES Istorice(id_istoric),
FOREIGN KEY (cnp_medic_rec) REFERENCES Medici(CNP_utilizator),
FOREIGN KEY (cnp_asistent) REFERENCES Asistenti_medicali(CNP_utilizator));

CREATE TABLE IF NOT EXISTS Cabinete
(id_cabinet INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
specialitate VARCHAR(255) NOT NULL,
unitate_medicala VARCHAR(100) NOT NULL,
FOREIGN KEY (unitate_medicala) REFERENCES Unitati_medicale(denumire));

CREATE TABLE IF NOT EXISTS Repartizare
(id_repartizare INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
id_cabinet INT,
CNP_medic BIGINT(13),
FOREIGN KEY (id_cabinet) REFERENCES Cabinete(id_cabinet),
FOREIGN KEY (CNP_medic) REFERENCES Medici(CNP_utilizator));
```

4.5.2 Cod proceduri

```
DROP PROCEDURE IF EXISTS completare_raport;
DELIMITER \\
CREATE PROCEDURE completare_raport (CNP_cadru_medical BIGINT(13), Raport INT, Campul
VARCHAR(255), Textul TEXT, OUT Rezultat INT)
BEGIN
        SET Rezultat := 0;
        SET @parafa := (SELECT cod_parafa
                                        FROM Rapoarte_programare
            WHERE Raport = id_raport);
```

```
        IF (@parafa IS FALSE) THEN
                SET @subsemnatul := (SELECT functie
                                                        FROM Utilizatori
                                                        WHERE CNP = CNP_cadru_medical);
                IF (@subsemnatul LIKE 'medic') THEN
                        CASE
      WHEN Campul LIKE 'cod_parafa' THEN
                                UPDATE Rapoarte_programare
        SET cod_parafa = TRUE
        WHERE id_raport = Raport;
                        WHEN Campul LIKE 'investigatie' THEN
                                UPDATE Rapoarte_programare
        SET investigatie = Textul
        WHERE id_raport = Raport;
                        END CASE;
      SET Rezultat := 1;
    ELSE
                        IF (@subsemnatul LIKE 'asistent medical') THEN
                                SET @asis := (SELECT CNP_asistent
                                                        FROM Rapoarte_programare
                WHERE id_raport = Raport);
                                IF (@asis IS NULL OR @asis = CNP_cadru_medical) THEN
                                        CASE
                                        WHEN Campul LIKE 'rezultat' THEN
                                                UPDATE Rapoarte_programare
                                                SET rezultat = TRUE
                                                WHERE id_raport = Raport;
                                        WHEN Campul LIKE 'recomandari' THEN
                                                UPDATE Rapoarte_programare
                                                SET recomandari = Textul
                                                WHERE id_raport = Raport;
                                        WHEN Campul LIKE 'simptome' THEN
                                                UPDATE Rapoarte_programare
                                                SET simptome = Textul
                                                WHERE id_raport = Raport;
                                        WHEN Campul LIKE 'diagnostic' THEN
                                                UPDATE Rapoarte_programare
                                                SET diagnostic = Textul
                                                WHERE id_raport = Raport;
                                        END CASE;

            IF (@asis IS NULL) THEN
                                                UPDATE Rapoarte_programare
              SET CNP_asistent = CNP_cadru_medical
              WHERE id_raport = Raport;
                                        END IF;

            SET Rezultat := 1;
                                END IF;
        END IF;
      END IF;
    END IF;
END; \\
DELIMITER ;
```

- Aceasta procedura se ocupa cu actualizarea directa a raportului cu datele introduse fie de medic, fie de asistent, in functie de niste conditii.
  Aceasta procedura se asigura ca raportul selectat nu este parafat, lucru care permite modificarea acestuia, iar contrariul permite doar vizualizarea sa. Apoi, stabileste ce functie doreste sa faca aceasta modificare, deoarece medicul are acces doar la modificarea unui atribut, pe cand, asistentul are acces la restul. Totodata, nu orice asistent poate face modificari, ci doar primul care a efectuat primul update pe atributele dedicate functiei sale; odata cu acest prim update, procedura va retine CNP-ul asistentului care a efectuat modificarea.

```
DROP PROCEDURE IF EXISTS stabilire_ora_final;
DELIMITER //
CREATE PROCEDURE stabilire_ora_final (ora_inceput TIME, durata INT, OUT ora_final TIME)
BEGIN
	-- aceasta este procedura care calculeaza ora de final a programarii
	SET @dur := (SELECT SEC_TO_TIME(durata*60));
	SET @dur := ADDTIME(ora_inceput, @dur);
	SET ora_final := @dur;
END; //
DELIMITER ;
```

- Cum nu exista nici o procedura de maipulare a datelor de timp care sa adauge un numar de minute la o ora am creat-o noi, am transformat numarul de minute in numar de secunde si ne-am folosit de functia care transforma secundele intr-o data de tip time, iar apoi am facut o adunare normal pe date time.

```
DROP PROCEDURE IF EXISTS calculare_salariu;
DELIMITER \\
CREATE PROCEDURE calculare_salariu (Nume_angajat VARCHAR(255), Prenume_angajat
VARCHAR(255), Luna INT, OUT Salariu BIGINT)
BEGIN
	SET @func := NULL;
  SET @func := (SELECT functie
                        FROM Utilizatori
     WHERE nume LIKE Nume_angajat AND prenume LIKE Prenume_angajat);
	SET @c := NULL;
  SET @c := (SELECT CNP
                        FROM Utilizatori
                        WHERE nume LIKE Nume_angajat AND prenume LIKE
Prenume_angajat);
	IF (@func IS NOT NULL AND @func <> 'medic') THEN -- aici este salariul calculat pentru
angajatii care nu sunt medici
		SET @data_incepere := (SELECT MIN(data_inceput) -- calculam perioada in care
exista concedii
                                        FROM Concedii
		 WHERE CNP_utilizator = @c AND MONTH(data_inceput) = Luna);
		SET @data_sf := (SELECT MAX(data_sfarsit)
                                        FROM Concedii
		 WHERE CNP_utilizator = @c AND MONTH(data_inceput) = Luna);
		SET @nr_ore_lipsa := 0; -- vom calcula numarul de ore lipsa, deoarece concediul nu
este platit asa ca le vom scade din numarul dde ore din contract
```

```
                WHILE @data_incepere <= @data_sf DO
                        SET @ok := (SELECT id_concediu
                                                FROM Concedii
                WHERE @data_incepere BETWEEN data_inceput AND data_sfarsit
                AND CNP_utilizator = @c);
                        IF (@ok IS NOT NULL) THEN -- daca exista concediu in ziua respectiva
atunci cautam cate ore ar fi lucrat in ziua respectiva dupa ziua saptamanii
                                SET @nr := (SELECT
SUM(TIME_TO_SEC(TIMEDIFF(ora_sfarsit, ora_inceput)) / 3600)
                                                        FROM Orare_generice
                WHERE DAYOFWEEK(@data_incepere) = ziua_saptamanii(zi)
                AND CNP_utilizator = @c);
                                SET @nr_ore_lipsa := @nr_ore_lipsa + @nr;
                        END IF;
        SET @data_incepere := DATE_ADD(@data_incepere, INTERVAL 1 DAY);
                END WHILE;
    SET @ore := (SELECT nr_ore
                                        FROM Utilizatori
                                        WHERE CNP = @c);
                SET @sal := (SELECT salar_neg
                                        FROM Utilizatori
            WHERE CNP = @c);
    SET Salariu := (@ore - @nr_ore_lipsa) * @sal;
        ELSE
                IF (@func IS NOT NULL OR @func LIKE 'medic') THEN -- daca este medic numai
insumam preturile consultatiilor sale
                        set @s := 0;
                        set @s := (select SUM(pret) from programari_pacient as p join
servicii_medicale as s where p.serviciu like s.denumire
                                        and s.CNP_medic = @c and month(ziua) = Luna);
                END IF;
                SET Salariu := @s;
  END IF;
END; \\
DELIMITER ;
```

- Aceasta procedura se imparte in doua: partea care afla salariul unui angajat care nu e medic, si partea care afla salariul unui medic (deoarece am ales sa facem bonusul, salariul medicului se calculeaza in functie de numarul de programari).
  Cum s-a specificat ca salariul este dat pentru un numar de ore, care se afla in datele utilizatorului trebuie sa vedem daca a avut concediu si a lipsit la o parte din acele ore. Pentru a calcula numarul de ore pe care le-a lipsit calculam prima oara intervalul in care a avut zile de concediu. Apoi parcurgem intervalul si verificam daca exista un concediu care sa aiba data cu care parcurgem cuprinsa in intervalul de concediu, apoi vedem ce zi a saptamanii este si vedem cate ore ar fi lucrat in ziua respective si le contorizam intr-o variabila. Apoi scadem din numarul total de ore aceasta perioada calculate si inmultim cu salariul negociat pe ora.
  Pentru a calcula salariul unui medic ii parcurgem toate programarile din luna respective si le adunam. Aceasta operatie o facem cu ajutorul unui join pe tabela de programari si de servicii care ne trebuie pentru a afla pretul unei programari care nu este retinut in programare ci in serviciul corespunzator.

```
DROP PROCEDURE IF EXISTS profit_policlinica;
DELIMITER \\
CREATE PROCEDURE profit_policlinica(Nume_policlinica VARCHAR(255), Luna INT, An INT,
OUT Salarii BIGINT)
BEGIN
        DECLARE utilizatorul VARCHAR(100);
                DECLARE done BOOLEAN DEFAULT FALSE;
                DECLARE cursorul CURSOR FOR (SELECT CNP
                                                        FROM Utilizatori);
                DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN cursorul;
    BEGIN
    SET @salarii:=0;
                parcurgere_utilizator: LOOP
                        FETCH cursorul INTO utilizatorul;
                        IF done THEN
                                LEAVE parcurgere_utilizator;
                        END IF;
                        SET @polic := (SELECT COUNT(id_orar) from orare_generice where
CNP_utilizator = utilizatorul and unitate_medicala like Nume_policlinica);
        IF(@polic <> 0) THEN
                                SET @nume := (select nume from utilizatori where CNP =
utilizatorul);
          SET @prenume := (select prenume from utilizatori where CNP = utilizatorul);
          SET @aux:=null;
          call calculare_salariu(@nume,@prenume, Luna, @aux);
          SET @salarii:=@salarii-@aux;
          SET @e_medic:=(select functie from utilizatori where CNP = utilizatorul);
          IF (@e_medic like 'medic') THEN
                                        SET @aux1:=null;
            call profit_medic_policlinica(@nume, @prenume,Nume_policlinica,Luna,@aux1);
            SET @salarii:=@salarii+@aux1;
          END IF;
        END IF;
                END LOOP;
    SET Salarii := @salarii;
    END;
        CLOSE cursorul;
END; \\
DELIMITER ;
```

- Pentru a afla profitul policlinicii trebuie sa aflam profitul total din toate programarile pe luna respectiva si totalul salariilor angajatilor de la policlinica.
  Profitul total il putem calcula insumand profitul pe care l-a adus fiecare medic, iar totalul salariilor parcurgand toti utilizatorii si verificand daca sunt angajati la policlinica respectiva.

4.5.3 Cod triggere

```
use policlinici;

DROP TRIGGER IF EXISTS creare_istoric;
```

```
DELIMITER //
CREATE TRIGGER creare_istoric AFTER INSERT ON Programari_pacient
FOR EACH ROW BEGIN
        SET @nume := NULL;
   SET @prenume := NULL;
   SET @nume := (SELECT i.nume_pacient -- vedem daca exista un istoric deja facut adica vom
cauta numele si prenumele
   FROM Istorice AS i
   WHERE nume_pacient LIKE i.nume_pacient);

   SET @prenume := (SELECT i.prenume_pacient
   FROM Istorice AS i
   WHERE prenume_pacient LIKE i.prenume_pacient);

   IF (@nume IS NULL OR @prenume IS NULL) THEN   -- daca nu s-au gasit, atunci introducem in
tsorice
                INSERT INTO Istorice (nume_pacient, prenume_pacient, ziua) VALUES
(new.nume_pacient, new.prenume_pacient, new.ziua);
   END IF;

   SET @istoric := (SELECT id_istoric
   FROM Istorice as i
   WHERE i.nume_pacient LIKE nume_pacient AND prenume_pacient LIKE i.prenume_pacient);
   -- raportul se va crea automat cu crearea unei programari
   INSERT INTO Rapoarte_programare (nume_pacient, prenume_pacient, ziua, rezultat, id_istoric,
cnp_medic_rec) VALUES (new.nume_pacient, new.prenume_pacient, new.ziua, 'Pozitiv', @istoric,
new.CNP_medic);
END; //
DELIMITER ;
```

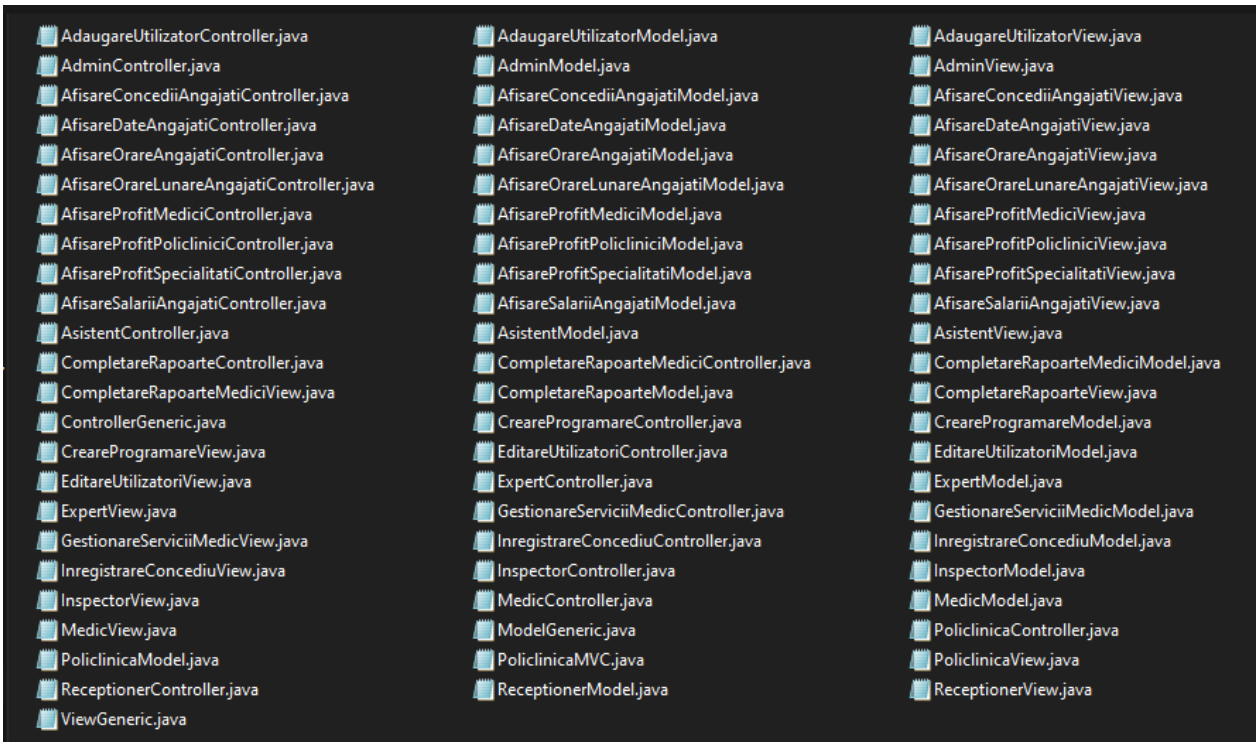
DROP TRIGGER IF EXISTS stergere_raport;

```
DELIMITER //
CREATE TRIGGER stergere_raport AFTER DELETE ON Programari_pacient -- daca se sterge o
programare trebuie sa se stearga automat si raportul ei
FOR EACH ROW BEGIN
        DELETE FROM Rapoarte_programare
   WHERE nume_pacient LIKE old.nume_pacient AND prenume_pacient LIKE
old.prenume_pacient;
END; //
DELIMITER ;

DROP TRIGGER IF EXISTS repartizare_pe_cabinet;
DELIMITER //
CREATE TRIGGER repartizare_pe_cabinet AFTER INSERT ON Specialitati
        FOR EACH ROW BEGIN
                DECLARE unitatea VARCHAR(100);
                DECLARE done BOOLEAN DEFAULT FALSE;
                DECLARE cursorul CURSOR FOR (SELECT unitate_medicala
                                                        FROM Orare_generice
                                                        WHERE new.CNP_medic
LIKE CNP_utilizator);
                DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE; --
parcurgem toate unitatile medicale la care este doctorul angajat si
      -- creeam reaprtizarea catre cabinetul existent cu specializarea introdusa
                OPEN cursorul;
```

```
                parcurgere_unitati: LOOP
                        FETCH cursorul INTO unitatea;
                        IF done THEN
                                LEAVE parcurgere_unitati;
                        END IF;
        SET @id_cab := NULL;
                        SET @id_cab := (SELECT id_cabinet -- verificam daca exista cabinetul
                                                FROM Cabinete
                                                WHERE unitatea LIKE unitate_medicala
                                                AND NEW.denumire LIKE specialitate);
                        IF (@id_cab IS NOT NULL) THEN -- iar daca exista intorduem
                                SET @ok := NULL;
          SET @ok := (SELECT id_repartizare -- doar daca nu a fost introdusa deja repartizarea
                                                        FROM Repartizare
                  WHERE id_cabinet = @id_cab AND CNP_medic = new.CNP_medic);
                                IF (@ok IS NULL) THEN
                                        INSERT INTO Repartizare (id_cabinet, CNP_medic)
    VALUES (@id_cab, new.CNP_medic);
                                END IF;
                        END IF;
                END LOOP;
        CLOSE cursorul;
    END; //
    DELIMITER ;

    DROP TRIGGER IF EXISTS validare_inserare_serviciu;
    DELIMITER \\
    CREATE TRIGGER validare_inserare_serviciu BEFORE INSERT ON Servicii_medicale
    FOR EACH ROW BEGIN
        SET @ok := NULL;
        SET @ok := (SELECT id_specialitate -- verificam daca doctorul are specializarea medicala
    potrivita
                                FROM Specialitati AS s
          WHERE s.CNP_medic = new.CNP_medic
          AND denumire LIKE new.specialitate);
       IF (@ok IS NOT NULL) THEN
                SET @ok2 := (SELECT id_competenta -- si daca are competentele necesare
                                        FROM Competente as c
            WHERE c.CNP_medic = new.CNP_medic
            AND denumire LIKE new.competente);
                IF (@ok2 IS NULL) THEN
                        SET NEW.denumire = NULL;
                END IF;
        ELSE
                SET NEW.denumire = NULL;
        END IF;
    END; \\
DELIMITER ;
```

5 Detalii de implementare

5.1 Structura claselor in Java



5.2 Manual de utilizare/instalare

Incepeti prin a va asigura ca aveti un server MySQL care are toate tabelele, procedurile, trigger-urile si datele populate asa cum sunt descrise. Dupa ce verificati ca totul functioneaza corect, deschideti fisierul "PoliclinicaMVC.java" si modificati rândul 35, astfel incat sa setati conexiunea cu baza de date prin introducerea numelui de utilizator si a parolei corecte ale serverului pe care il detineti.

Dupa ce totul este pregatit, puteti sa deschideti aplicatia prin aplicarea comenzii "Run & Build" din fisierul "PoliclinicaMVC.java". Aceasta va deschide pagina de logare pentru conturile utilizatorilor care au fost introdusi pana in acel moment in baza de date. Pe aceasta pagina, aveti posibilitatea de a accesa fereastra corespunzatoare functiei pe care o detine utilizatorul care se logheaza (numele de utilizator este CNP-ul acestuia, iar parola este cea aleasa la crearea contului) prin apasarea butonului "Login" sau, daca sunteti administrator sau superadministrator, puteti sa va logati pe pagina speciala acestora prin apasarea butonului "Login admin", de unde, in functie de ierarhia tipurilor de utilizatori, puteti adauga sau sterge acestia (superadministrator -> administrator -> angajat). Paginile dedicate fiecarei functii contin butoane care duc utilizatorul la functionalitatile specifice acestuia, dar, totodata, fiecare dintre acestea contin interogari personale legate de concedii, salariu si programul de lucru. Tot pe aceste pagini, puteti gasi butonul de delogare, care va duce utilizatorul la prima pagina

unde a avut loc logarea; orice buton apasat de pe aceste pagini va deschide o alta (fara a inchide meniul principal personal), care, dupa terminarea operatiilor executate pe ea, trebuie inchisa prin apasarea butonului "Exit".

6 Bonusuri

1. Pentru *bonsul de repartizare a medicilor pe cabinete*, am pornit cu ideea ca fiecare unitate medicala are un anumit numar de cabinete predefinite (incarcate odata cu unitatea) fiecare fiind definit de o specializare, iar acest fiind unic pe unitate, dar multiplu, posibil, multiplu pe lantul de policlinici; atunci cand un medic isi defineste o specializare(intrdoucere in baza de date), cu ajutorul unui trigger (presupunand ca deja orarul sau de lucru este definit – acesta poate lucra la diferite untiati) se va cauta prin toate unitatile la care acesta lucreaza si se va repartiza la fiecare cabinet, din aceste unitati, care este definit de noua specializare. Acest bonus se foloseste de 2 tabele adiacente(Repartizari si Cabinete), datele medicului sitriggerul respectiv.

2. Pentru *bonusul de calculare a salariului unui medic* ne-am folosit de serviciile medicale personalizate care retin cnp-ul medicului care le efectueaza si pretul serviciului. Ii parcurgem medicului toate programarile din luna respectiva si le adunam, acesta suma reprezentand salariul. Aceasta operatie o facem cu ajutorul unui join intre tabela de programari si cea de servicii (joinul fiind pe cnp-ul medicului), tabela de servicii ne trebuie pentru a afla pretul unei programari, pret care nu este retinut in programare ci in serviciul corespunzator. Acest bonus se foloseste de procedura de baza (calculare salariu angajat) cu modificarile si strategia explicate mai sus.

3. Pentru *bonusul de gestionarea a serviciilor unui medic*, am lasat acest lucru interactiv si prezent in interfata, astfel serviciile medicale pentru un medic pot fi modificate in pagina specifica gestionarii servicilor medicale. In aceasta pagina se pot adauga/sterge servicii medicale pentru un anumit medic, servicii care au un pret si o durata. Totodata, serviciile noi adaugate unui medic trebuie sa corespunda cu specialitatea si competentele sale. Daca un anumit medic nu intruneste aceste conditii, noul serviciu nu va fi adaugat, iar in interfata grafica va aparea o eroare. Acest bonus se foloseste de 2 proceduri dedicate (stergerea si adaugarea unui serviciu), la nivelul carora se executa verificarile.

7 Concluzii. Limitari si dezvoltari ulterioare

Aplicatia ofera suport deplin pentru angajati pentru a-si opera toate interactiunile cu o unitate medicala acestea fiind personalizate pentru fiecare functie. Este o aplicatie practica si usor de folosit, aducand multe atuuri unui lant de policlinici modern. Astfel, cu ajutorul acestei aplicatii se poate renunta la mijloace traditionale de retinere a datelor, ducumente fizice si dosare, toate datele fiind stocate in aceasta baza de date, care ofera si o viteza mai mare la cautare si introducerea de date.
Interfata creata face aplicatia sa fie user-friendly, orice persoana cu un minim de cunostinte tehnice o poate folosi, aceasta facilitand munca pentru toti angajatii.

Dezvoltari ulterioare:

In viitor, ar putea fi implementate mai multe masuri de securitate a datelor, precum criptarea parolelor sau limitarea accesului la anumite date doar la posesorul acestora sau la persoanele autorizate. De asemenea, ar putea fi implementat un istoric al analizelor efectuate pentru fiecare pacient. In plus, ar putea fi posibil sa se poata prelua datele unui pacient direct de pe cardul lor de sanatate prin intermediul unui cititor de carduri