Universitatea Tehnică Cluj-Napoca

Facultatea de Automatică și Calculatoare

Departamentul Calculatoare



**Proiect de Semestru**

la disciplina

Introducere in Baze de Date

**Lant de Policlinici**

**Studenti:** Pucani Liviu Catalin

Tudor Mihai Nicusor

Varvara Raluca Ana Maria

**Grupa:** 30222

An academic: 2020 - 2021

Cuprins

[2 Introducere 3](#_Toc61546675)

[3 Analiza cerintelor utilizatorilor 3](#_Toc61546676)

[3.1 Ipotezele policlinicii (cerinte si constrangeri) 3](#_Toc61546677)

[3.2 Organizarea structurata tabelar a cerintelor utilizator 4](#_Toc61546678)

[3.3 Determinarea si caracterizarea profilurilor de utilizator 5](#_Toc61546679)

[4 Modelul de date si descrierea acestuia 6](#_Toc61546680)

[4.1 Entitati si atributele lor 6](#_Toc61546681)

[4.2 Diagrama EER/UML pentru modelul de date complet 9](#_Toc61546682)

[4.3 Proceduri, triggere si view-uri 9](#_Toc61546683)

[4.4 Normalizarea datelor 11](#_Toc61546684)

[4.5 Interogari MySql 11](#_Toc61546685)

[4.6 Cod MySql 13](#_Toc61546686)

[4.6.1 Cod pentru crearea bazei de date si a tabelelor 13](#_Toc61546687)

[4.6.2 Cod proceduri 16](#_Toc61546688)

[4.6.3 Cod triggere 20](#_Toc61546689)

[5 Detalii de implementare 22](#_Toc61546690)

[5.1 Structura claselor in Java 22](#_Toc61546691)

[5.2 Manual de utilizare/instalare 23](#_Toc61546692)

[5.3 Elemente de securitate ale aplicatiei 24](#_Toc61546693)

[6 Bonusuri 24](#_Toc61546694)

[7 Concluzii. Limitari si dezvoltari ulterioare 25](#_Toc61546695)

# Introducere

Proiectul presupune dezvoltarea unei aplicatii care lucreaza cu baze de date pentru gestiunea unui lant de policlinici. Scopul aplicatiei este simplificarea operatiilor cu baza de date prin oferirea unei interfete grafice pe care angajatii sa o poata utiliza. Aplicatia ofera sprijin atat pentru interogarea bazei de date cat si pentru manipularea acesteia. Cu ajutorul aplicatiei se pot crea programari pentru un serviciu medical numai la un medic care are competentele necesare, iar apoi aplicatia genereaza un bon fiscal.

Exista mai multe tipuri de utilizatori, administratori si superadministratori care pot sa prelucreze datele oricui, si angajati care sunt de 5 tipuri fiecare putand sa acceseze sau manipuleze alte date. Exista 3 departamente: resurse umane, financiar contabil si medical, fiecare departament fiind raspunzator de cate un alt aspect al functionarii policlinicii.

Pentru crearea aceste baze de date am avut ma multe **scopuri**:

* Eficientizarea operatiilor care se pot face intr-o policlinica. Accesul la date este unul mult **mai rapid** decat varianta clasica de a cauta dosare si a scrie pe hartie rapoarte si programari;
* **Concurenta operatiior**; se pot completa mai mlte rapoarte in acelasi timp de pe mai multe computere;
* Este o aplicatie mult mai sigura, nu are toata lumea accesul la date, prin folosirea unor dosare fizice oricine ar avea acces la ele, in schimb aplicatia ofera accesul la date numai daca persoana are functia specifica**, nu se mai poate umbla fara permisie si nu se pot fura date**;
* Este o aplicatie usor de folosit pe care chiar si o persoana care are cunostinte minime informatice o poate utiliza usor si sigur;
* Oferirea automata a unui bon duce la imposibilitatea evaziunii fiscale;
* Este o aplicatie **ecofriendly** deoarece reduce la minim consumul de hartie si alte materiale plastice pentru partea administrativa a unei policlinici

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

# Analiza cerintelor utilizatorilor

## 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]](#footnote-1)și gradul[[2]](#footnote-2).
* Pentru un **medic** se va reține suplimentar specialitatea sau specialitățile în care își desfășoară activitatea, gradul[[3]](#footnote-3), codul de parafă, competențele pe care le deține pentru realizarea unor proceduri ce necesită acreditări speciale[[4]](#footnote-4), titlul științific[[5]](#footnote-5). Totodată, fiecare medic are negociat un procent din serviciile medicale realizate care îi revine, adiţional față 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) și 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)

## Organizarea structurata tabelar a cerintelor utilizator

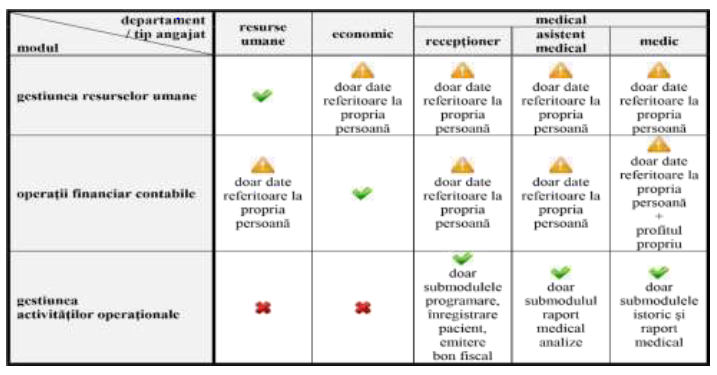
Baza de date trebuie sa stocheze urmatoarele informatii:

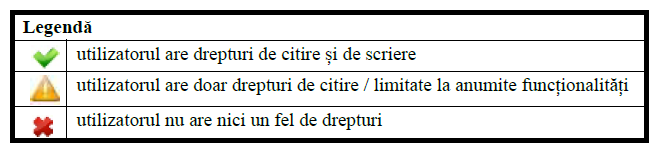
* Unitatile medicale
* Programele unitatilor medicale
* Utilizatorii
* Orarele generice ale utilizatorilor
* Orarele specifice ale utilizatorilor
* Concediile utilizatorilor
* Asistentii medicali
* Medicii
* Specialitatile medicilor
* Competentele medicilor
* Serviciile medicale pe care le ofera un medic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

Mai mult trebuie sa permita si urmatoarele operatii:

* 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

Aplicatia Java trebuie sa permita prelucrarea informatiilor din baza de date, cautare si afisarea informatiilor si conectarea utilizatorilor inregistrati.





## 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

1. Angajati care sunt de 5 tipuri
2. 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

1. 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

1. Receptioneri

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

1. Asistenti medicali

* Completare rapoarte

1. Medic

* Completare raport
* Completare investigatie
* Parafare raport
* Modificare servicii medicale (proprii)

# Modelul de date si descrierea acestuia

## Entitati si atributele lor

**Unitati\_medicale**  ofera informatii despre fiecare unitate medicala. Atribute: denumire, adresa id\_program. Se leaga de Programe si cabinete.

**Programe**  retine informatii despre programul fiecarei policlinici. Atribute: id\_program, luni\_ora\_inceput, luni\_ora\_sfarsit, marti\_ora\_inceput, … duminica\_ora\_sfarsit.

**Utilizator** ofera informatii despre fiecare utilizator. Atribute: CNP, nume, prenume, adresa, nr\_telefon, email, IBAN, nr\_contract, data\_angajarii, tip, functie, salariu\_neg (salariu negociat pe ora), nr\_ore (pe luna), parola. 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. Atribute: id\_orar, CNP\_utilizator, zi (varchar poate fi luni marti miercuri…), ora\_inceput, ora\_sfarsit, unitate\_medicala.

**Orare\_specifice** ofera informatii despre orarul spcific al unui utilizator pentru o anumita data. Atribute: id\_orar, CNP\_utilizator, zi (data), ora\_inceput, ora\_sfarsit, unitate\_medicala.

**Concedii** ofera informatii despre concediile unui utilizator. Atribute: id\_concediu, CNP\_utilizator, data\_inceput, data\_sfarsit.

**Asistenti\_medicali** ofera informatii suplimentare pentru utilizatorii care au functia de asistent medical. Atribute: CNP\_utilizator, tip, grad.

**Medici** ofera informatii suplimentare despre utilizatorii care sunt si medici. Atribute: CNP+utilizator, titlu\_stiintific, cod\_parafa, postul\_didactic, procent\_adaugat.

**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. Atribute: id\_specialitate, denumire, grad, CNP\_medic.

**Competente** ofera informatii despre competentele unui medic. Atribute: id\_competenta, denumire, cnp\_medic.

**Servicii\_medicale** ofera informatii despre serviciile medicale pe care le poate face un medic in functie de specialitatile si competentele sale. Atribute: id\_serviciu\_medical, denumire, CNP\_medic, specialitate, competente, pret, durata. 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). Atribute: id\_programare, ziua, ora, durata, CNP\_medic, serviciu, unitate\_medicala, nume\_pacient, prenume\_pacient.

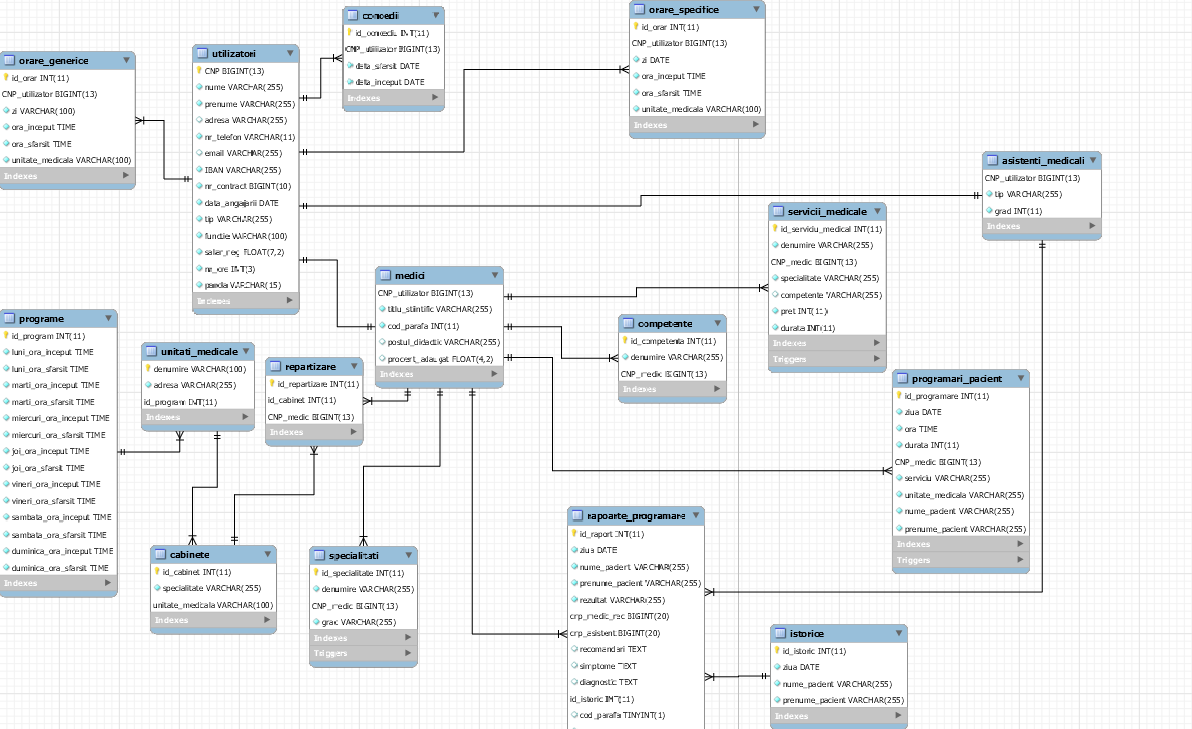
**Istorice** ofera informatii depsre istoricul unui pacient. Atribute: id\_istoric, ziua (in care a fost facut istoricul), nume\_pacient, prenume pacient.

**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. Atribute: id\_raport, ziua, nume pacient, prenume pacient, rezultat, cnp\_medic, cnp\_asistent, recomandari, simptome, diagnostic, id\_istorice, cod\_parafa, investigatie.

**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.) Atribute: id\_cabinet, specialitate, unitate\_medicala). 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. Atribute: id\_repartizare, id\_cabinet, CNP\_medic.

## Diagrama EER/UML pentru modelul de date complet



## Proceduri, 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.

## Normalizarea datelor

Definitia formei normale Boyce-Codd (FNBC) este: Fie R o schemă de relație si F multimea de dependențe funcționale asociată. Se spune ca R este în forma normală Boyce-Codd dacă și numai dacă

oricare ar fi o dependență netrivială X →Y din F , atunci X este supercheie pentru R.

Baza de date respectă. Atributele fiecărui tabel nu depind de alte atribute. Fiecare tabel are o

singură cheie primară după care sunt identificate înregistrările și este suficientă pentru a identifica în mod unic orice înregistrare din baza de date.

În fiecare tabel avem doar o cheie și toate dependențele au în partea stângă o supercheie (cheia primară a tabelei). De exemplu, pentru tabela Utilizator avem cheia primară CNP și toate dependențele au în stânga această supercheie, în tabelul Unitate medicala această supercheie este denumire, in tabelul Programe aceasta cheie este id\_program etc.

## Interogari MySql

**SELECT** i.nume\_pacient FROM Istorice AS iWHERE 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

## Cod MySql

### 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));

### 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.

In primul rand, prin parametrul CNP\_cadru\_medical se transmite CNP-ul utilizatorului

care doreste sa faca acest update, prin parametrul Raport ni se transmite id-ul raportului

ce se doreste a fi modific, prin parametrul Campul ni se specifica care atribut din tupla

repsectiva este modificat, iar prin parametrul Textul ni se pune exact ce se introduce in

acel camp.

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). Am ales sa nu facem 2 proceduri diferite pentru a nu supraincarca interfata expertului financiar.

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 pentru a nu parcurge toata luna. 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 sis a le insumam, iar totalul salariilor parcurgand toti utilizatorii, veriicand daca sunt angajati la policlinica respective si calculandu-le salariul. Astfel am ales un cursor cu care sa parcurgem toti utilizatorii. Pentru a verifica daca utilizatorul este angajat la aceasta policlinica, numaram cate id\_orar exista in orare generice care sa aiba numele unitatii medicale corespunzator si care sa fie ale utilizatorului current, daca are 0 inseamna ca nu e angajat. Apoi ii calculam salariul cu procedura de calculare salariu si il scadem din profit, iar apoi ii cautam functia. Daca este medic, ii calculam profitul pe care l-a adus unitatii medicale si il adaugam la profit.

### 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 ;

# Detalii de implementare

## Structura claselor in Java

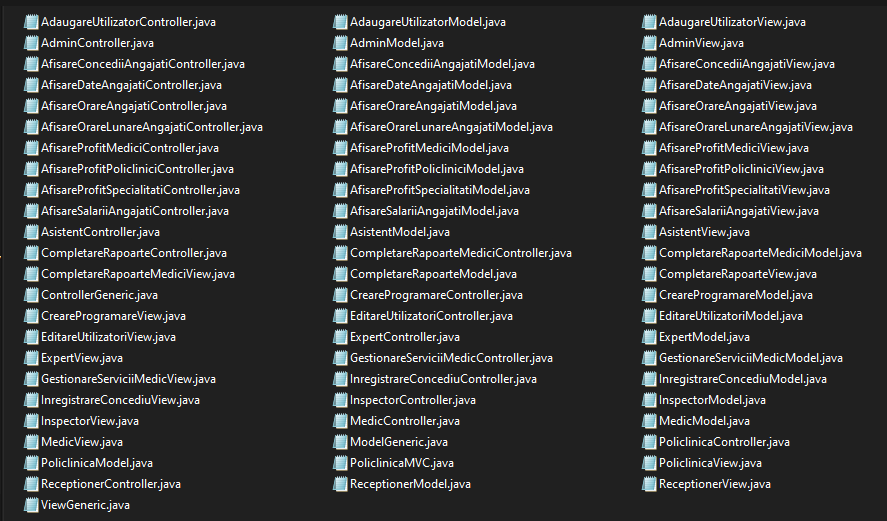


Diagrama UML - <https://didatec.sharepoint.com/:i:/s/ViardsofUTCN/EZD8tlrZep1BlNTUNhdT0f8BM83LOoJZfX7ZyxCCsqF2hA?e=AkYUyk>

## Manual de utilizare/instalare

Pentru început, asigurați-vă că dețineți un server MySQL și pe acesta aveți implementată toată baza de date descrisă mai sus (tabele + procedurile + triggerele + popularea).

După ce v-ați asigurat că totul a rulat cu succes, deschideți fisierul „PoliclinicaMVC.java”, iar acolo trebuie să modificați, de pe rândul 35, conținutul stringului care stabilește conexiunea cu baze de date prin scrierea corectă a username-ului și a parolei serverului deținut.

Acum că totul este complet aranjat, puteți să deschideți aplicația prin aplicarea comenzii „Run & Build” a mainului din fișierul „PoliclinicaMVC.java”. Odată cu aceasta se va deschide pagina de logare, aferentă conturilor utilizatorilor introduși până la acel moment în baza de date. Pe această pagină aveți posibilitatea de a merge pe fereastra corespunzătoare funcției pe care o deține utilizatorul care se loghează (username-ul = CNP-ul acestuia, password-ul = parola aleasă la creare contului) unde dispuneți de facilitățile propriei funcții prin apăsarea butonului de „Login” sau, dacă vă aflați în postura de administrator sau super-administrator, vă puteți loga pe pagina specială acestora cu butonul de „Login admin”, unde, în funcție de ierarhia tipurilor de utilizatori, puteți exercita adăugarea sau ștergerea acestora (super-administrator-»administrator-»angajat)

Paginile dedicate fiecarei funcții conțin butoanele ce duc utilizatorul pe funcționalitățile specifice acestuia, dar, totodată, fiecarea dintre acestea conțin interogări personale legate de concedii, salariu și programul de lucru. Tot pe aceste pagini, se poate găsi butonul de delogare, care va duce utilizatorul pe prima pagină unde a avut loc logarea; orice buton apăsat de pe aceste pagini, va deschide o altă (fără a închide acest meniu principal personal), care, după terminarea operațiilor executate pe ea trebuie închisă de la butonul de exit.

Recomand ca de fiecare dată când se introduc date sau când se modifică altele existente, să fie facute cu atenție, deoarece modificările nu vor avea loc și nu vor actualiza baza de date cu noile informații.

## Elemente de securitate ale aplicatiei

Aplicatia prezinta un nivel de securitate in momentul conectarii la baza de date dar si in momentul logarii propriu zise.

In functie de modul de conectare ales(angajat sau administrator sunt disponibile urmatoarele operatii):

* Angajat: poate accesa baza de date pentru a isi vedea datele si pentru a introduce si modifica date in functie de functia pe care o are
* Administrator: poate accesa, modifica datele oricui si de asemenea sterge si insera date.

Datele pe care le poate modifica si accesa un angajat in functie de functia pe care o are sunt descrise in detaliu la capitolul 3.3.

De asemenea datele sunt securizate prin faptul ca nimeni nu poate vedea parola niciunui utilizator, aceatsa cand este introdusa este inlocuita cu \*\*\*\*.

# 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.

# Concluzii. Limitari si dezvoltari ulterioare

Toate cerintele au fost respectate si indeplinite, acestea ducand la un mediu placut prin care angajatii pot interactiona cu o un lant de policlinici modern si la un mod convenabil de gestiune a datelor de catre acestia.

Aplicatia ofera suport deplin pentru angajati pentru a-si opera toate interactiunile cu o unitate medicala acestea fiind personalizate pentru fiecae 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, foaie si pix, 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.

Ca si dezvoltari ulterioare s-ar putea face:

* O securizare a datelor mai buna, deja nu pot accesa decat anumite functii anumite date, dar pot exista si date pe care ar trebui sa le vada numai posesorul datelor sau persoana respectiva
* Implementarea unui istoric al analizelor
* Criptarea parolelor pentru a augmenta securizarea datelor
* Preluarea datelor unui pacient direct de pe cardul de sanatate cu un cititor de carduri
* Printarea automata a bonului fiscal, momentan bonul fiscal apare numai digital.

1. Tipul unui asistent medical poate fi: generalist, laborator, radiologie. [↑](#footnote-ref-1)
2. Gradul unui asistent medical poate fi secundar, principal. [↑](#footnote-ref-2)
3. Gradul unui medic poate fi specialist, primar. [↑](#footnote-ref-3)
4. Astfel de proceduri pot fi ecografie, endoscopie digestivă, ecocardiografie, cardiologie intervențională, bronhoscopie, EEG, EMG, dializă, chirurgie laparoscopică, chirurgie toracică, chirurgie spinală, chirurgie spinală, litotriție extracorporeală, explorare computer tomograf / imagistică prin rezonanță magnetică și altele. [↑](#footnote-ref-4)
5. Nu este obligatoriu ca un medic să dețină un titlu științific. În cazul în care medicul deține un astfel de titlu, acesta poate fi: doctorand sau doctor în științe medicale. [↑](#footnote-ref-5)