Bază de date pentru
Programare la o clinică medicală locală

Barcan Nicoleta-Gabriela
Grupa 1310A

Profesor Coordonator
Mironeanu Cătălin

1. Titlul Proiectului

Proiectul este intitulat “Programare la o clinică medicală locală” și presupune analiza, proiectarea și implementarea unei baze de date și a relațiilor necesare între tabele pentru a modela informații despre programările pacienților și ceea ce a rezultatul în urma acestora. Baza de date este menită utilizării în cadrul clinicii, de angajații acesteia.

2. Descrierea proiectului

Baza de date propusă dorește ușurarea atât a procesului de a face o programare la clinică, cât și reținerea și accesarea mai ușoară a diagnosticului și tratamentului dat unui pacient. Printre acțiunile ce sunt permise se numără: inserarea, updatarea și ștergerea informațiilor ce țin de pacienți, programare, tratament, medici și specializările lor.

În realizarea baze de date, avem în vedere următoarele constrângeri:

- Clădirea clinicii are doar 3 etaje + parter, deci etajul poate fi între 0 și 3
- Sălile sunt 20 în total
- Numerele de telefon trebuie să aibă un anumit format, să înceapă cu 0 și să aibă maxim 10 cifre
- Email-ul unui pacient trebuie să aibă un anumit format, caractere înainte de un ‘@’ și după alte caractere urmate de un ’.’ și un alt șir de 2-4 caractere
- Numele, prenumele, descrierile sau denumirile pot conține doar litere și pot avea spații, deoarece fiind vorba de acordarea unui tratament, e bine ca fiecare informație să fie corectă
- Programările pot fi făcute doar după data curentă și nu poate fi o dată și oră identică cu alta
- Gen-ul unei persoane poate fi trecut cu o singură majusculă

Informațiile care sunt necesare pentru această bază de date sunt legate în primul rând de **Pacient** (nume, prenume, vârsta și telefon), dacă acesta are un **Istoric Inițial,** atunci îl notăm (boli anterioare, alergii, tratament curent). Pentru a realiza o **Programare** vom avea nevoie de dată, ora și informații legate de sală, etaj și dacă se știe, ce tip de serviciu va fi. De la **Medicul** care va fi la acea programare, este necesar numele și prenumele, împreună cu **Specializarea** sa.

În urma fiecărei programări, se determină un **Diagnostic** (denumire, detalii, simptome)**,** care face parte dintr-o categorie mare de diagnostic **CMD.** Un diagnostic poate avea mai multe **Tratamente** (numele pastilei, cantitatea, intervalul de administrare și durata tratamentului).

3. Structura si inter-relationarea tabelelor

- **Modelul Logic**

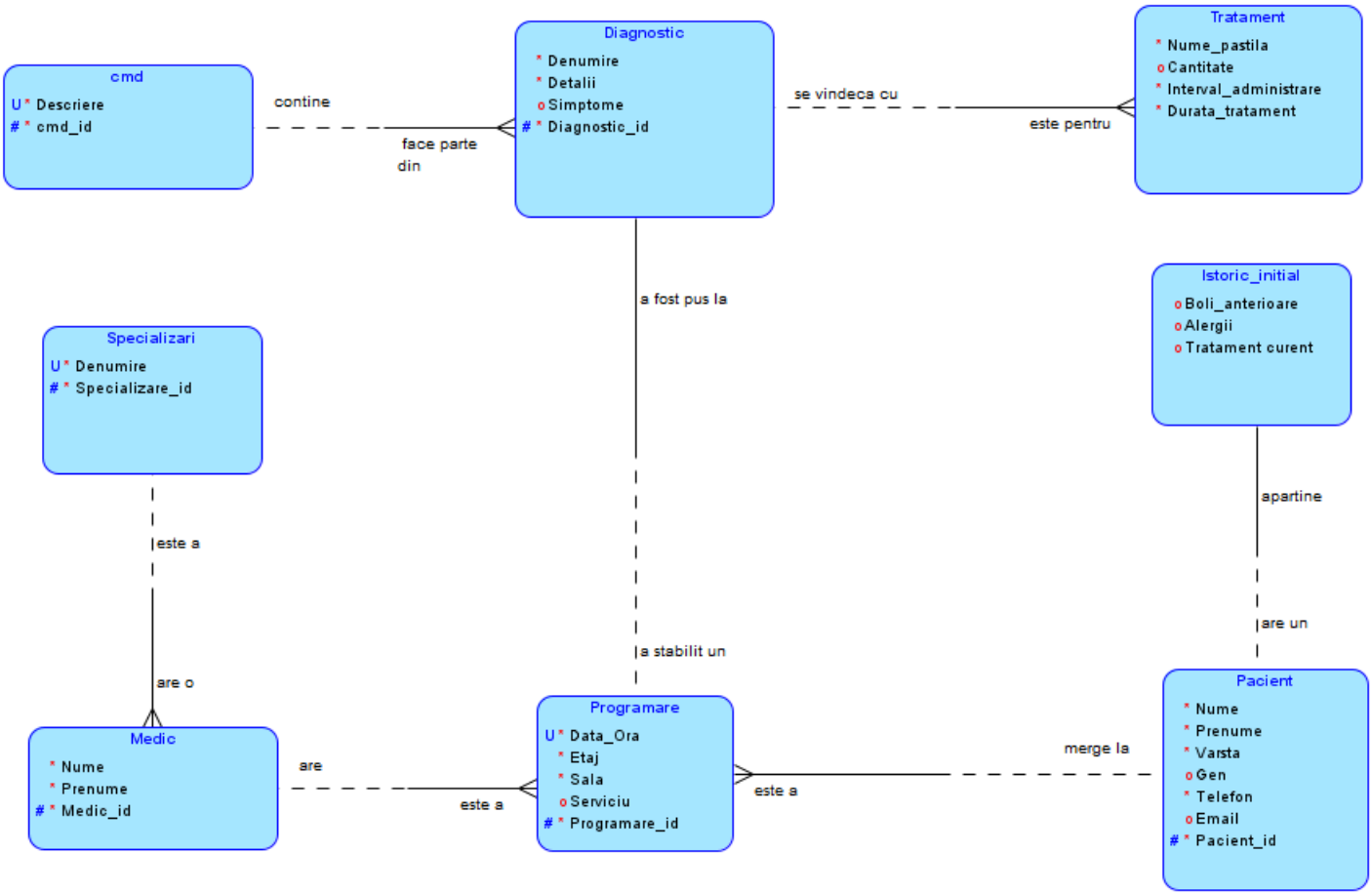

Fig1. Diagrama modelului logic

În acest model se regăsesc **Entitățile** cu **Atributele** lor:

- **cmd**
  - Descriere: de tip VARCHAR (100), unique, not null pentru trebuie să știm care e categoria și poate conține doar litere și spații ca să se înțeleagă exact care e categoria
  - cmd_id: de tip NUMERIC (5), primary-key implicit și not null

- **Diagnostic**
  - Denumire: de tip VARCHAR (100), not null și poate conține doar litere și spații, aici este cel mai important să se completeze corect
  - Detalii: de tip VARCHAR (100), not null
  - Simptome: de tip VARCHAR (100), poate fi null pentru că sunt persoane asimptomatice
  - Diagnostic_id: de tip NUMERIC (5), primary-key implicit și not null

- Programare_Programare_id: de tip NUMERIC (5), relation UID (foreign-key) implicit și not null (propagat din entitatea programare)
- cmd_cmd_id: de tip NUMERIC (5), relation UID (foreign-key) implicit și not null (propagat din entitatea cmd)

- **Tratament**
  - Nume_pastilă: de tip VARCHAR (20), not null și poate conține doar litere și spații
  - Cantitate: de tip NUMERIC (5)
  - Interval_administrare: de tip VARCHAR (50), not null
  - Durata_tratament: de tip VARCHAR (30), not null
  - Diagnostic_Diagnostic_id: de tip NUMERIC (5), relation UID (foreign-key) implicit și not null (propagat din entitatea Diagnostic)

- **Specializări**
  - Denumire: de tip VARCHAR (50), not null și poate conține doar litere și spații
  - Specializare_id: de tip NUMERIC (5), primary-key implicit și not null

- **Medic**
  - Nume: de tip VARCHAR (10), not null și poate conține doar litere și spații
  - Prenume: de tip VARCHAR (10), not null și poate conține doar litere și spații
  - Medic_id: de tip NUMERIC (5), primary-key implicit și not null
  - Specializari_Specializare_id: de tip NUMERIC (5), relation UID (foreign-key) implicit și not null (propagat din entitatea Specializare)

- **Programare**
  - Data_Ora: de tip Timestamp (0), not null, trebuie să fie mai mare decât data curentă și să fie unique
  - Etaj: de tip NUMERIC (2), not null și se află în intervalul [0,3]
  - Sala: de tip NUMERIC (2), not null și e mai mic strict decat 21
  - Serviciu: de tip VARCHAR (10) și poate conține doar litere și spații
  - Programare_id: de tip NUMERIC (5), primary-key implicit și not null
  - Pacient_Pacient_id: de tip NUMERIC (5), relation UID implicit și not null (propagat din entitatea Pacient)
  - Medic_Medic_id: de tip NUMERIC (5), relation UID implicit si not null (propagat din entitatea Medic)

- **Pacient**
  - Nume: de tip VARCHAR (20), not null, poate conține doar litere și spații
  - Prenume: de tip VARCHAR (20), not null și poate conține doar litere și spații
  - Vârsta: de tip NUMERIC (3), not null
  - Gen: de tip VARCHAR (1) și poate conține doar majuscule
  - Telefon: de tip VARCHAR (10), not null, începe doar cu 0 și continuă cu 7, 2 sau 3, continuat de orice secvență de cifre

- Email: de tip VARCHAR (100), trebuie să conțină după o secvență aleatoare de litere, cifre și caracterele "_" și ".", un "@" și după o altă secvență aleatoare "." și o secvență de litere de lungime între 2 și 4, ca să ne asigurăm pe cât posibil că este o adresă validă
- Pacient_id: de tip NUMERIC(5), primary-key implicit și not null

- **Istoric_inițial**, nu este mandatory pentru că pot fi persoane care au fost sănătoase înainte sa vină la clinică
  - Boli_anterioare: de tip VARCHAR (100)
  - Alergii: de tip VARCHAR (100)
  - Tratament_curent: de tip VARCHAR (100)
  - Pacient_Pacient_id: de tip NUMERIC (5), foreign-key si primary-key implicit și not null

După stabilirea entităților și a atributelor, am stabilit relațiile între acestea. Se vor identifica 2 relații de 1:1 și 5 relații de 1:n.

Între entitățile **Pacient** și **Istoric_initial** este relație de unu la unu, deoarece o persoană poate avea un singur istoric inițial atunci când vine la clinică. Acesta descrie ce boli, alergii sau tratament curent are pacientul, pentru ca medical să poata lua o decizie mai informată asupra diagnosticului.

A doua relație de unu la unu se afla între entitățile **Programare** și **Diagnostic**, deoarece în urma unei întâlniri între pacient și medic, poate rezulta un singur rezultat, oricât de complex și detaliat ar fi.

Între entitățile **Pacient** și **Programare** avem o relație de unu la mai mulți, un om având posibilitatea de a face mai multe programări la clinică, ori de câte ori are nevoie, doar să fie la o dată care nu se regăsește în baza de date și mai mare decât data curentă.

Între **Medic** și **Programare** există tot o relație de unu la mai mulți, deoarece un medic poate avea mai multe programări într-o zi, la ore diferite, dar o programare nu poate să aparțină mai multor medici.

O altă relație de unu la mulți este între **Medic** și **Specializări**, mai mulți medici de la clinica locală putând avea aceeași specializare.

Între entitățile **cmd** și **Diagnostic** este o relație de unu la mai mulți, într-o categorie mare de diagnostic putând fi mai multe tipuri de diagnostic. Cum ar fi diagnosticul de "bronșită" sau de "pneumonie", care fac parte din categoria de "boli ale sistemului respirator". Un diagnostic nu poate face parte din mai multe categorii.

Între **Diagnostic** și **Tratament** am ales să fie relație de unu la mai mulți, deoarece pentru un diagnostic pot exista mai multe tratamente, pentru cazurile când un tratament este alcătuit din 2 părți, adică se schimbă pastila sau modul de administrare după o perioadă. Un tratament nu funcționează pentru mai multe diagnostice.

Este de notat existența unui proces de organizare a bazei de date, prin care se minimizează potențialele erori de la manipularea datelor și redundanța datelor, numit **normalizare**.

Se observă prima formă normală **1NF**, prin faptul că în fiecare tabela, fiecare înregistrare este identificată în mod unic printr-o cheie primară. Nu există repetiții cum ar fi mai multe valori semnificative în același câmp sau mai multe coloane care să reprezinte același lucru.

Toate cheile primare din tabelele bazei de date sunt formate dintr-un singur atribut. Ceea ce face ca tabele să se afle în a doua formă normală **2NF**.

- **Modelul Relational/Fizic**

În acest model am implementat mecanismul de autoincrement cu trigger pentru cheile primare: **Diagnostic_id**, **Medic_id**, **Programare_id**, **Pacient_id;** întrucât tabelele acestor coloane ar fi cele care ar primi date cel mai des. Cheile primare din tabelele cmd și Specializări, deoarece conțin date care s-ar schimba rar, au fost alese să se introducă manual.

Am realizat un trigger pentru datele din coloana **Data_Ora,** ca data și ora la care se face o programare să fie mai mari decat data curentă.

Fig2. Diagrama modelului fizic

4. Interfața cu utilizatorul

Pentru o gestionare mai ușoară a datelor din tabele, a fost costruită o interfață grafică. Aceasta a fost creată în **python 3**, cu ajutorul bibliotecii **flask**.

Conexiunea la baza de date și efectuarea schimbărilor din aceasta s-a realizat cu biblioteca **cx_Oracle.** De asemenea, a fost nevoie de descărcarea **Oracle Instant Client,** de unde am preluat fișierele ddl.

Aceasta este linia de comandă prin care s-a realizat conexiunea.

```
con = cx_Oracle.connect("bd125", "bd125", "bd-dc.cs.tuiasi.ro:1539/orcl")
```

Interfața a fost realizată în HTML și CSS. A are o structură simplă, făcând posibilă interacțiunea cu fiecare tabelă în parte. În bara de navigare se află butoane, care deschid o pagină nouă corespondentă tabelei cu același nume cu cel de pe buton.

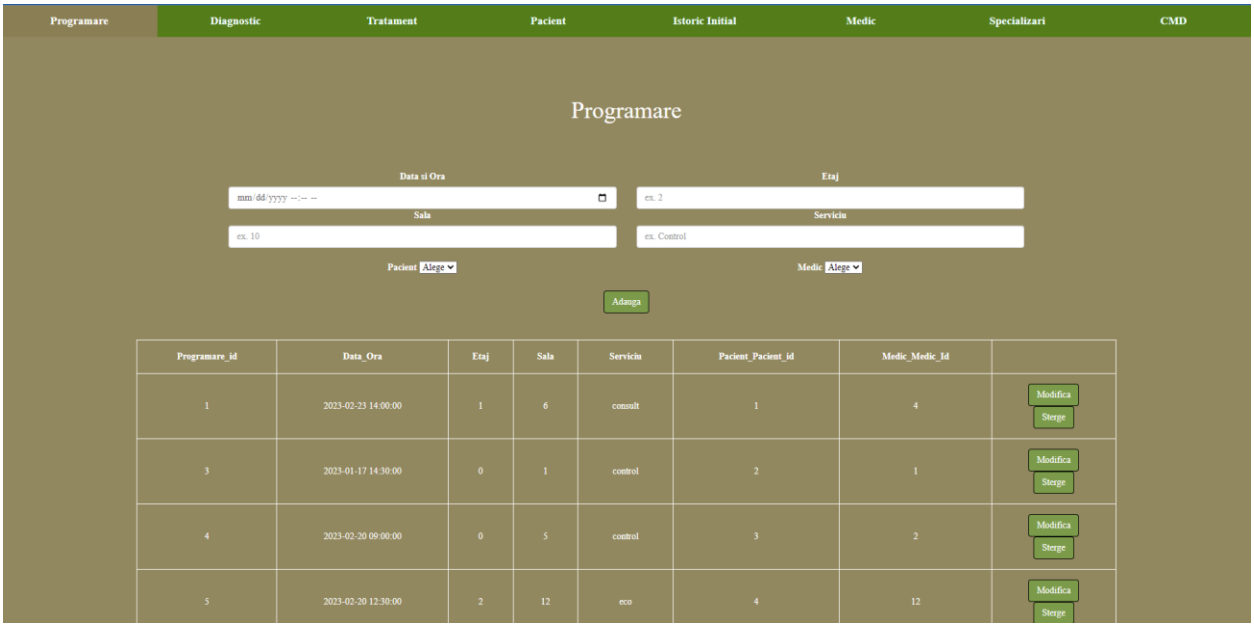

Fig3. Pagina corespunzătoare tabelei Programare

Pe fiecare pagină, pe lângă vizualizarea tabelei, există butoane în plus (“Modifică”, ”Șterge”, ”Adaugă” sau “Nou pacient”) sau câmpuri, ce ne oferă posibilitatea de a modifica/șterge/insera date din/în baza de date. Luând în calcul aplicația practică a interfeței, operațiile enumerate diferă de la o tabelă la alta, după cum urmează:

- **Programare**:
  -Inserare: se inserează data și ora, etajul, sala, serviciul și se selectează id-ul pacientului și al medicului. Id-ul programării este incrementat automat.
  -Ștergere: se șterg aceste date, în funcție de id-ul programării
  -Modificare: în funcție de id-ul pacientului, se selectează programarea asupra căreia se doresc făcute modificări

- **Diagnostic**:
  -Inserare: se inserează denumirea, detaliile, simptomele și se aleg id-ul programării și cel al cmd-ului, id-ul diagnosticului este autoincrementat
  -Ștergere: se șterg datele în functie de id-ul diagnosticului
  -Modificare: în funcție de denumire, se selectează linia corespunzătoare acelui id și se editează datele, se apasă pe butonul de “Trimite” ca să se facă commit

- **Tratament**:
  -Inserare: se inserează date despre numele pastilei, cantitatea si intervalul de administrare, durata tratamentului, iar id-ul diagnosticului pentru care este tratamentul se selectează
  -Ștergere: în funcție de id-ul diagnosticului pentru care este tratamentul (diagnostic_diagnostic_id), se șterg datele de pe acea linie
  -Modificare: pentru această tabelă nu a fost implementată editarea, pentru că într-o clinică medicală este necesar să existe toate tratamentele date unui pacient în formatul initial

  Exemplu de funcții în python care afișează și șterg datele din tabela Tratament.

```python
# tratament start code
@app.route('/tratament')
def tratament():
    tratam = []

    cur = con.cursor()
    cur.execute('select * from tratament order by diagnostic_diagnostic_id')
    for result in cur:
        tra = {}
        tra['diagnostic_diagnostic_id'] = result[4]
        tra['nume_pastila'] = result[0]
        tra['cantitate'] = result[1]
        tra['interval_administrare'] = result[2]
        tra['durata_tratament'] = result[3]
        tratam.append(tra)
    cur.close()

    com = []
    cur = con.cursor()
    cur.execute('select diagnostic_id from diagnostic')
    # import pdb;pdb.set_trace()
    for result in cur:
        com.append(result[0])
    cur.close()

    return render_template('tratament.html', tratam=tratam, tratament=com)
```

```python
@app.route('/delTratament', methods=['POST'])
def del_tratament():
    c = "'" + request.form['diagnostic_diagnostic_id'] + "'"
    cur = con.cursor()
    cur.execute('delete from tratament where diagnostic_diagnostic_id=' + c)
    cur.execute('commit')
    return redirect('/tratament')
```

- **Pacient**:
  -Inserare: pentru a insera un nou pacient se apasă pe butonul “Nou pacient”, care ne va redirecționa spre o pagină unde vom putea insera date despre numele, prenumele, vârsta, gen-ul, telefonul și email-ul noului pacient, id-ul fiind autoincrementat
  -Ștergere: în funcție de id-ul pacientului se șterge linia corespunzătoare
  -Modificare: în funcție de numele pacientului, se selectează linia cu id-ul acelui pacient, asupra căreia se pot schimba date

- **Istoric Initial**:
  -Inserare: se pot insera date despre boli anterioare, alergii și tratamentul curent al unui pacient existent

-Ștergere: în functie de pacientul cu respectivul istoric (pacient_pacient_id), se șterg datele de pe acea linie
-Modificare: în acest caz, modificarea unui istoric inițial nu avea sens, deoarece acesta nu se schimbă în general

- **Medic**:
  -Inserare: se inserează date despre numele, prenumele și se selectează specializarea. Id-ul medicului este inserat automat
  -Ștergere: în funcție de id-ul medicului se șterg datele de pe respectiva linie
  -Modificare: un medic poate avea mai multe specializări, iar acest lucru se poate obține prin adăugare, în rest datele care ar rămâne de modificat sunt numele și prenumele

- **Specializări**:
  -Inserare: se poate insera denumirea unei specializari, id-ul acesteia apărând prin autoincrement
  -Ștergere: în funcție de id-ul specializării, se șterge linia
  -Modificare: nu a fost implementată această opțiune, deoarece specializările pe care le poate avea un doctor sunt standard și pot doar apărea unele noi

- **CMD**:
  -Inserare: în secțiunea din josul paginii se poate insera descrierea unei noi categorii. Id-ul categorii va fi inserat prin autoincrement
  -Ștergere: în funcție de id-ul categoriei, se șterge linia
  -Modificare: nu a fost implementată această opțiune, deoarece categoriile mari de diagnostic nu se schimbă, pot doar apărea unele noi

Așa arată pagina de modificare a unei linii dintr-o coloană din tabela Programare. În exemplu este selectată programarea cu id=1.

Tranzacțiile în această aplicație sunt implementate prin instrucțiunea “commit”, cu ajutorul limbajului python. După o serie de instucțiuni de SQL (selectarea datelor, actualizarea sau inserarea acestora), se execută commit-ul pentru încheierea tranzacției și se salvează modificările în baza de date.

```
@app.route('/editPacient', methods=['POST'])
def edit_emp():
    emp = 0
    cur = con.cursor()
    nume = "'" + request.form['nume'] + "'"

    cur.execute('select pacient_id from pacient where nume=' + nume)
    for result in cur:
        emp = result[0]
    cur.close()
    prenume = "'" + request.form['prenume'] + "'"
    varsta = request.form['varsta']
    gen = "'" + request.form['gen'] + "'"
    telefon = "'" + request.form['telefon'] + "'"
    email = "'" + request.form['email'] + "'"

    cur = con.cursor()
    query = "UPDATE pacient SET nume=%s, prenume=%s, varsta=%s, gen=%s, telefon=%s, email=%s where pacient_id=%s" % (
    nume, prenume, varsta, gen, telefon, email, emp)

    cur.execute(query)
    cur.execute('commit')
    return redirect('/pacient')
```

5. Bibliografie

- Pentru realizarea bazei de date am folosit: Oracle SQL Developer pentru scrierea scripturilor de inserare și testare, iar pentru diagrama ER, am utilizat Oracle SQL Developer Data Modeler