UNIVERSITATEA TEHNICĂ “GH ASACHI” IAŞI FACULTATEA AUTOMATICĂ ŞI CALCULATOARE SPECIALIZAREA CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI

DISCIPLINA BAZE DE DATE

Adăpost pentru persoane

Coordonator,
Prof. Ș.l.dr. Cristian Buțincu

Ionita Bogdan Marian
1306B

Titlu Tema: Adapost pentru persoane

Acest proiect conține o implementare a unei baze de date care modelează activitatea unui centru pentru persoane fără adăpost.

Descrierea proiectului și scopul aplicației

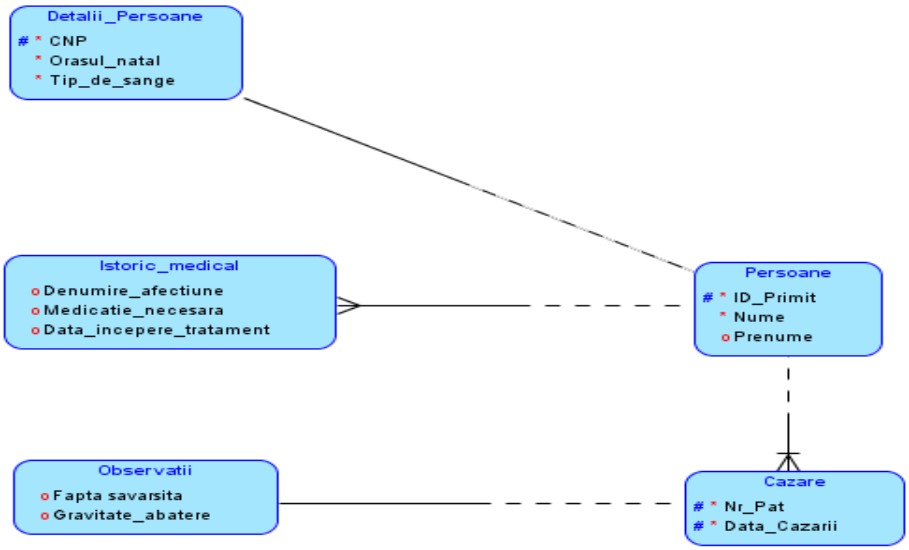
Un adapost pentru persoane se ocupă cu primirea și înregistrarea oamenilor care au nevoie de acest serviciu. Înregistrarea persoanelor intr-o baza de date, in care salvam informatiile personale ale acestora, este benefica, atat pentru a le oferi servicii mai bune, dar și pentru a realiza anumite statistici la nivel de tara/judet. De asemenea, astfel se asigura faptul ca o persoana nu profita de pe urma acestora.

Pentru o gestiune cat mai buna a bazei de date sunt necesare următoarele informații:
-Datele personale ale celor cazați, precum cnp-ul, numele,prenumele, tipul de sange dar si un istoric medical scurt.

-Restul informațiilor, care țin de desfășurarea procesului de cazare, precum data în care persoana vine în adăpost, sau ce observații negative primește, vor fi introduse de cate angajații centrului.

Structura tabelelor și inter-relationarea entităților

Nivelul logic



În modelul logic se observa 5 entități și campurile specifice acestora.
În proiectarea acestei baze de date s-au folosit relații 1:1 si 1:n.

Între **Persoane** și **Istoric medical** exista o relație **1:n**, cheia primara ID propagandu-se in tabela copil. Aceasta relatie este necesara, deoarece pentru fiecare persoana, pot exista mai multe afectiuni care trebuie notate.

Fiecare persoana, după ce se inregistreaza, în caz ca nu a mai fost în acest adapost, va primi un pat și se va consemna data in care a venit. Acest lucru se realizeaza printr-o relație tot de **1:n** între **Persoane** și **Cazare**, relație care asigura faptul ca un om poate fi cazat de mai multe ori, in zile diferite.

Între **Cazare** și **Observații** exista o relație de **1:1**, deoarece pentru fiecare cazare existentă, se pot însemna anumite fapte negative savarsite de persoana în cauză. In anumite cazuri, o persoana poate primi interdictie de a mai veni în adăpost.

Constrangerile folosite:

Pentru a evita introducerea datelor eronate de către utilizatori, în cadrul bazei de date s-au stabilit diferite constrangeri:

1.Constrangeri de tip check:

- Pentru CNP s-a folosit un regex pentru a asigura integritatea acestuia.

- Campurile Orasul_natal, Nume, Prenume au o constrângere de tip check pentru a nu permite introducerea numerelor.

- Campurile de mai sus, precum și denumire afectiune, medicatie necesara si fapta savarsita au o constrangere de lungime.

- S-au folosit constrângeri de tip check pentru a asigura corectitudinea domeniului de valori pentru campurile tip de sange, gravitate abatere si nr pat.

2.Constrangeri de tip Primary Key(+Not Null,Unique)

Majoritatea entitatilor au un camp definit ca P.K. În afara de tabela care contine detaliile personale ale celor cazați, care are ca P.K cnp-ul. Tabela Persoane are P.K cu auto increment.

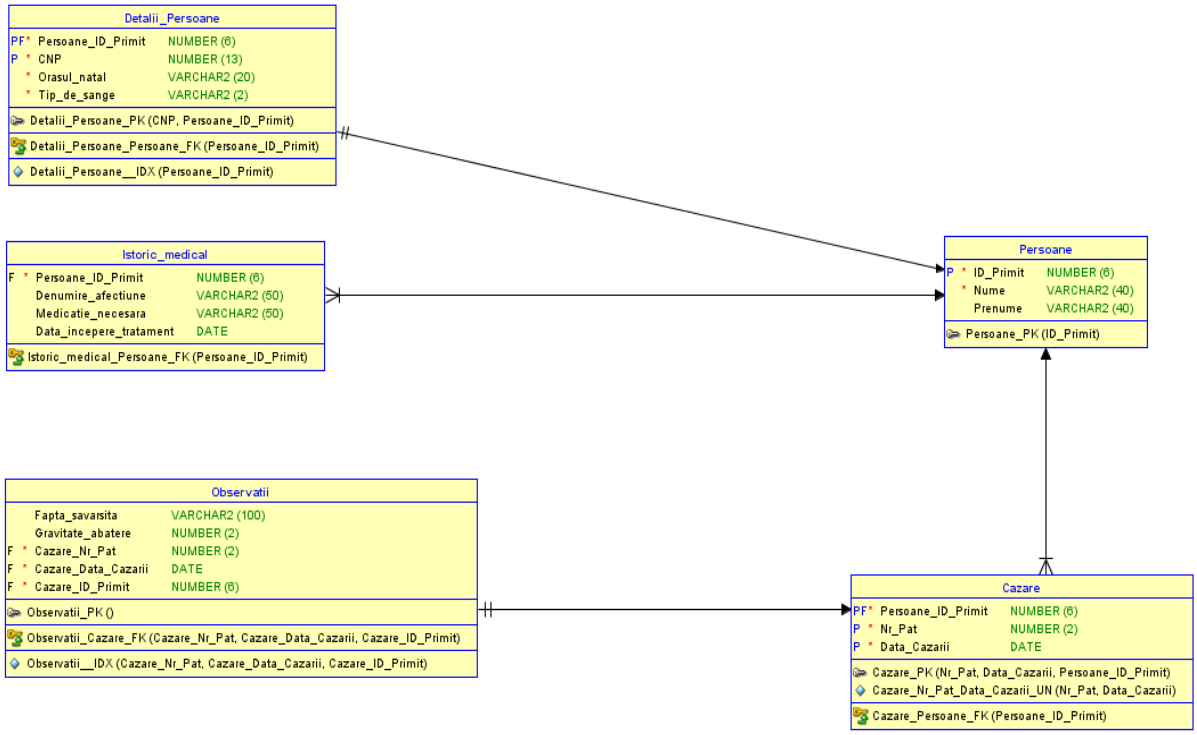
3.Constrangeri de tip NOT NULL

Multe campuri sunt necesare funcționării corecte și logice a bazei de date, astfel acestea sunt definite ca fiind not null, pentru a asigura existenta unei valori. În afara de valorile din **Istoric medical** și **Observații**, toate campurile au proprietatea de NOT NULL.

4.Constrangeri de tip Foreign Key

Acestea au rezultat în urma relațiilor dintre entități, cheia primara propagandu-se de la parinte la copil.

Nivelul relational



La acest nivel apar alte constrangeri, precum cele de tip trigger.

5.Trigger

Constrangerile care nu s-au putut realiza prin check-uri, s-au facut prin triggere la nivel relational.
Acestea sunt:

- Trigger pentru data, pentru a nu insera în niciun camp de tip data o valoare mai mare decat sysdate.
- Un trigger pentru campul de tip data din tabela **Cazare** pentru a se asigura faptul ca o persoana nu poate fi cazata de 2 ori în aceeași zi.
- Un trigger în tabela **Cazare** care verifica dacă persoana care urmează sa fie cazata a avut o abatere cu o gravitate mai mare decat 7. În caz pozitiv, aceasta nu va fi lăsată în adăpost.

Descrierea tehnologiilor folosite

- **PyQT5**

Aplicatia este creata pe baza bibliotecii PyQT5 din python. Interfata a fost creata cu ajutorul tool-ului QT Designer, interfata care a fost convertita ulterior in cod python.

```
pip install PyQt5
```

- **cx_Oracle**

Cx_Oracle este un modul de extensie Python care permite accesul la Oracle Database. Pentru a utiliza cx_Oracle 7 cu Python și Oracle Database aveți nevoie de:
Python 2.7 sau 3.5 și mai mult. Versiunile mai vechi ale cx_Oracle pot funcționa cu versiuni mai vechi ale Python.
Standardul de interoperabilitate Oracle standard pentru client-server permite cx_Oracle să se conecteze la bazele de date mai vechi și mai noi.
Instalare

```
Python -m pip install cx_Oracle - upgrade
```

Funcțiile principale oferite de acest modul sunt:
-connect: primește ca parametru user,parola și un server pentru a se conecta la baza de date Oracle dorita.
-execute: primește ca parametru un string pentru a realiza o comanda sql la nivelul bazei de date pe care s-a conectat.
-commit: salvează modificările realizate până în acel moment.

Funcțiile principale ale aplicației

Aplicația contine 4 tipuri de funcții principale:
-**Comenzi de selectare a datelor pentru afișarea lor pe interfata**: showPersoane,showDetaliiP,showIstoric,showCazare si showObservatii.
Aceste funcții sunt reunite în functia showAllValues pentru o utilizare mai usoara.

```
def showPersoane(self):
    persoane = []
    cur = con.cursor()
    cur.execute('select * from persoane order by ID_Primit')
    for result in cur:
        persoana = {}
        persoana['ID'] = result[0]
        persoana['Nume'] = result[1]
        persoana['Prenume'] = result[2]
        persoane.append(persoana)
    cur.close()
    row = 0
    self.tablePersoane.setRowCount(0)
    self.tablePersoane.setRowCount(len(persoane))
    for person in persoane:
        self.tablePersoane.setItem(row, 0, QtWidgets.QTableWidgetItem(str(person["ID"])))
        self.tablePersoane.setItem(row, 1, QtWidgets.QTableWidgetItem(person["Nume"]))
        self.tablePersoane.setItem(row, 2, QtWidgets.QTableWidgetItem(person["Prenume"]))
        row = row + 1
```

-Comenzi de inserare pentru fiecare tabel:

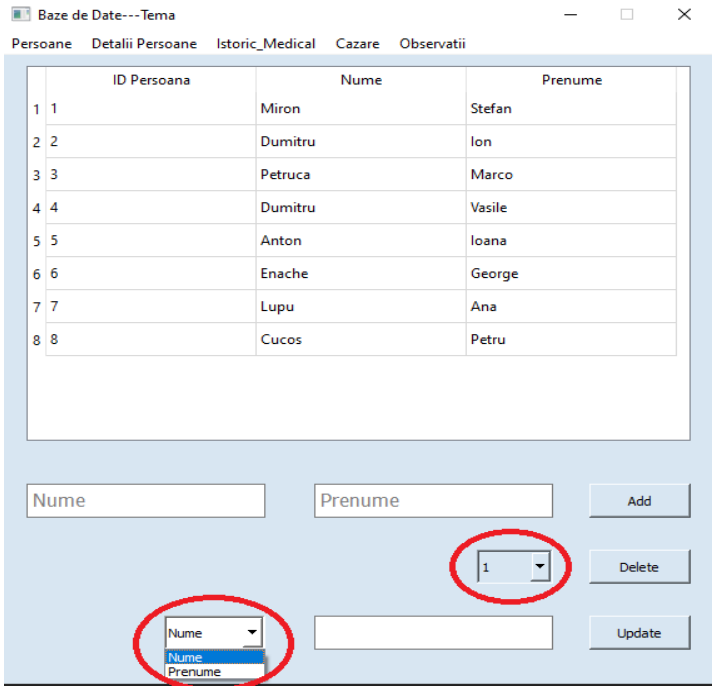
```
def addEntryPersoana(self):
    nume = self.PersinsertNume.toPlainText()
    nume = nume.capitalize().strip()
    prenume = self.PersInsertPrenume.toPlainText()
    prenume = prenume.capitalize().strip()
    if (len(nume) < 3 or len(prenume) < 3):
        error_dialog = QtWidgets.QErrorMessage()
        error_dialog.showMessage('Nume/Prenume introdus incorect!')
        error_dialog.exec_()
        return
    try:
        cur = con.cursor()
        sql = """ INSERT INTO Persoane
                            (Nume,Prenume) VALUES ('{}','{}')""".format(nume,prenume)
        cur.execute(sql)
        con.commit()
        showAllValues(self)

    except cx_Oracle.Error as e:
        error_dialog = QtWidgets.QErrorMessage()
        error_dialog.showMessage(str(e))
        error_dialog.exec_()
    finally:
        cur.close()
```

Aceste funcții de inserare, precum și funcțiile de delete si update, se realizate în blocuri de tip try și except. În caz ca o comanda nu se poate realiza din cauza

incalcarii unei constrangeri făcute la nivelul bazei de date, o eroare de tip pop-up va afișa ce s-a intamplat.

-Comenzi de update:

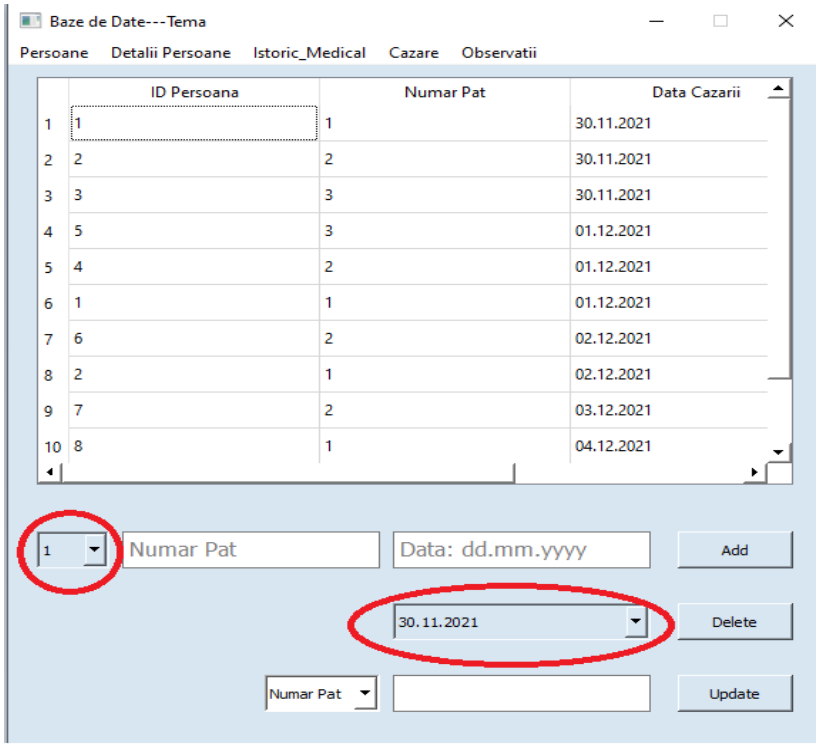
Utilizatorul va alege de la nivelul interfeței P.K randului pe care vrea să-l modifice, împreuna cu campul care urmează a fi modificat.




```python
def updatePersoane(self):
    camp=str(self.PerscomboChooseUpdate.currentText())
    campIntrodus = str(self.PersinsertUpdate.toPlainText()).strip().capitalize()
    ID = str(self.PersIDcomboBox.currentText())
    if (len(campIntrodus)<3):
        error_dialog = QtWidgets.QErrorMessage()
        error_dialog.showMessage('Nume/Prenume introdus incorect!')
        error_dialog.exec_()
        return
    try:
        cur = con.cursor()
        if(camp=="Nume"):
            sql = """ Update Persoane set Nume='{}' where ID_Primit={}""".format(campIntrodus, ID)
        else:
            sql = """ Update Persoane set Prenume='{}' where ID_Primit={}""".format(campIntrodus, ID)

        cur.execute(sql)
        con.commit()
        showAllValues(self)

    except cx_Oracle.Error as e:
        error_dialog = QtWidgets.QErrorMessage()
        error_dialog.showMessage(str(e))
        error_dialog.exec_()
    finally:
        cur.close()
```

-Comenzi de delete:

Utilizatorul va alege de asemenea PK-ul(urile) necesare pentru ștergerea unei singure valori din baza de date. De exemplu, pentru a sterge o valoarea din tabela Cazare, utilizatorul trebuie sa aleaga persoana prin campul ID, si data la care aceasta a fost cazata.




```
def deleteEntryCazare(self):
    ID=str(self.CazareIDcomboBox.currentText())
    Data = str(self.CazareDatacomboBox.currentText())
    try:
        cur = con.cursor()



        cur.execute(
                """ Delete from Cazare where Persoane_ID_Primit={} and Data_Cazarii=to_date('{}','dd.mm.yyyy')""".format(
                    ID, Data))
        con.commit()
        showAllValues(self)

    except cx_Oracle.Error as e:
        error_dialog = QtWidgets.QErrorMessage()
        error_dialog.showMessage(str(e))
        error_dialog.exec_()
    finally:
        cur.close()
```

Campurile de tip foreign key, precum ID si data, au fost generate automat, pentru a nu fi introduse date eronate de către utilizatori. Funcția getAllID genereaza ID-urile persoanelor din tabela Persoane. Funcția IDChanged detectează cand un ID este schimbat de către utilizator, și pune în campul necesar data la care persoana cu ID-ul ales s-a cazat.

```
def IDChanged1(self):
    Data = []
    try:
        cur = con.cursor()
        cur.execute('select Data_Cazarii from Cazare where Persoane_ID_Primit={} order by Data_Cazarii'.format(str(self.ObservatiiIDcomboBox.currentText())))
        for result in cur:
            entry = result[0]
            Data.append(entry)
    except:
        return
    finally:
        cur.close()
    self.ObservatiiDatacomboBox.clear()
    for date in Data:
        self.ObservatiiDatacomboBox.addItem(date.strftime("%d.%m.%Y"))
```