UNIVERSITATEA TEHNICĂ "GHEORGHE ASACHI", IAŞI

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

SPECIALIZAREA CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI

DISCIPLINA BAZE DE DATE

TEMA DE CASĂ

Gestiunea materialelor din inventarul unei primării

Coordonator,

Avram Sorin

Student,

Percă Anca-Elena

Grupa 1309 B

An universitar 2021-2022

Cuprins

1. Descrierea proiectului

În acest proiect am dorit să evidențiez importanța unei baze de date la nivelul inventarului unei primării. Chiar dacă nu pare un lucru atât de complicat, un gestionar nu se ocupă doar de câteva cereri/achiziții într-o zi, ci mai are și alte lucruri de făcut pe lângă acestea. Consider că prin baza de date pe care am creat-o este mult mai ușor să ții evidența stocurilor, a cererilor făcute de angajați, mai ales în cazul celor care nu au fost încă rezolvate.

Baza de date stochează informații despre materialele prezente în inventar, tipurile acestora, furnizorii de la care se fac achiziții, angajații primăriei, detalii despre angajați, dar și date despre cererile făcute de angajați pentru a primi anumite bunuri sau informații despre achiziții efecuate. Gestiunea este cantitativă (nu se tratează problema prețurilor).

Primăria achiziționează cele necesare de la mai mulți furnizori, iar un anumit material poate fi achiziționat de la mai mulți furnizori;

Combustibilul se poate da doar în cantitate întreagă, iar agrafele de birou, maăștile și mănușile doar în cutii întregi;

Este foarte important faptul că gestionarul nu poate oferi mai mult decât există în stocul instituției.

Când un angajat dorește să primească ceva din inventarul primăriei, acesta trebuie să facă o cerere (se onorează imediat dacă stocul permite) în care să solicite cele necesare. Cererea va conține id-ul angajatului, id-ul materialului solicitat (sau id-urile materiale lor solicitate, deoarece se acceptă mai multe bunuri per cerere), numărul cererii, data solicitării, canitatea/fiecare material, iar gestionarul, după ce verifică stocurile, completează cererea cu un câmp ce indică dacă aceasta a fost rezolvată sau nu.

De exemplu, dacă un angajat cere pentru un anumit produs o cantitate mai mare decât cea din stoc, cererea va fi marcată cu "nu", iar gestionarul va efectua o comandă a acelui material și opțional și a altor materiale din cadrul aceluiași "tip de materiale". După ce furnizorul aduce comanda, angajatul va primi cantitatea solicitată, iar cererea va fi marcată ca fiind rezolvată.

Atunci când se fac achiziții, stocul crește, iar când o cerere este onorată, stocul scade.

Pe baza informațiilor din baza de date se vor putea obține diferite informații, precum:

- denumirea materialelor al căror stoc este mai mic decât unul dat;
- numele angajaților cu cereri care încă nu au fost rezolvate (+ materialele și cantitatea solicitată);
- numărul de cereri depuse de fiecare angajat;
- numărul de cereri depuse în fiecare lună ;
- materialele care nu au fost solicitate;
- numărul de angajați de gen masculin;
- numele angajaților născuți în luna ianuarie.

2. *Tehnologiile folosite*

Pentru partea de ***front-end*** am folosit Qt Designer, un program foarte util și ușor de folosit în crearea interfețelor grafice. Pentru fiecare tabel din baza de date am creat o fereastră "Dialog" , iar apoi, prin drag and drop, am poziționat diferite elemente:

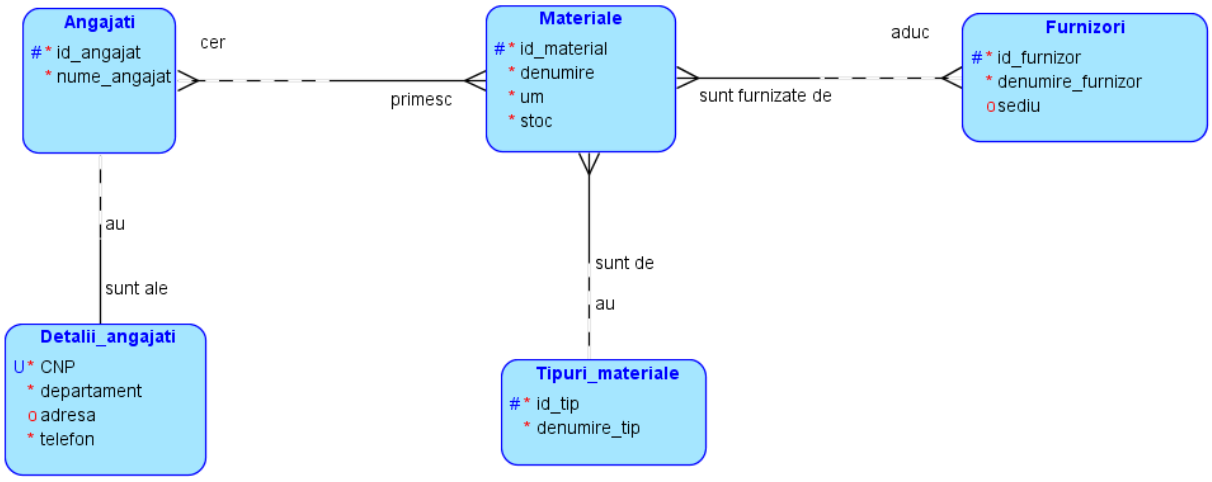
- *Label* - pentru secvențe de text;
- *LineEdit* - pentru acele casete în care utilizatorul introduce datele necesare inserării/modificării/ștergerii datelor din tabele;
- *ComboBox* - pentru acele casete din care utilizatorul poate alege o anumită valoare dintr-o listă de valori predefinite;
- *TextBrowser* - pentru locul în care se realizează afișarea datelor din tabel și pentru caseta în care utilizatorului îi este oferit un feedback;
- *PushButton* - pentru butoanele de pe interfață.

Pentru partea de ***back-end*** am lucrat în programul PyCharm și am folosit limbajul Python și modulul PyQt5.

Cu ajutorul comenzii *pyuic5 -x Nume.ui -o Nume.py* am generat un cod pentru fiecare interfață creată în Qt Designer. Am unit toate aceste coduri, de la cele 7 interfețe plus interfața de început și cea pentru testarea aplicației, și cu ajutorul funcțiilor din Python am reușit să creez o funcționalitate pentru fiecare buton de pe fiecare interfață.

3. *Structura si inter-relaționarea tabelelor*

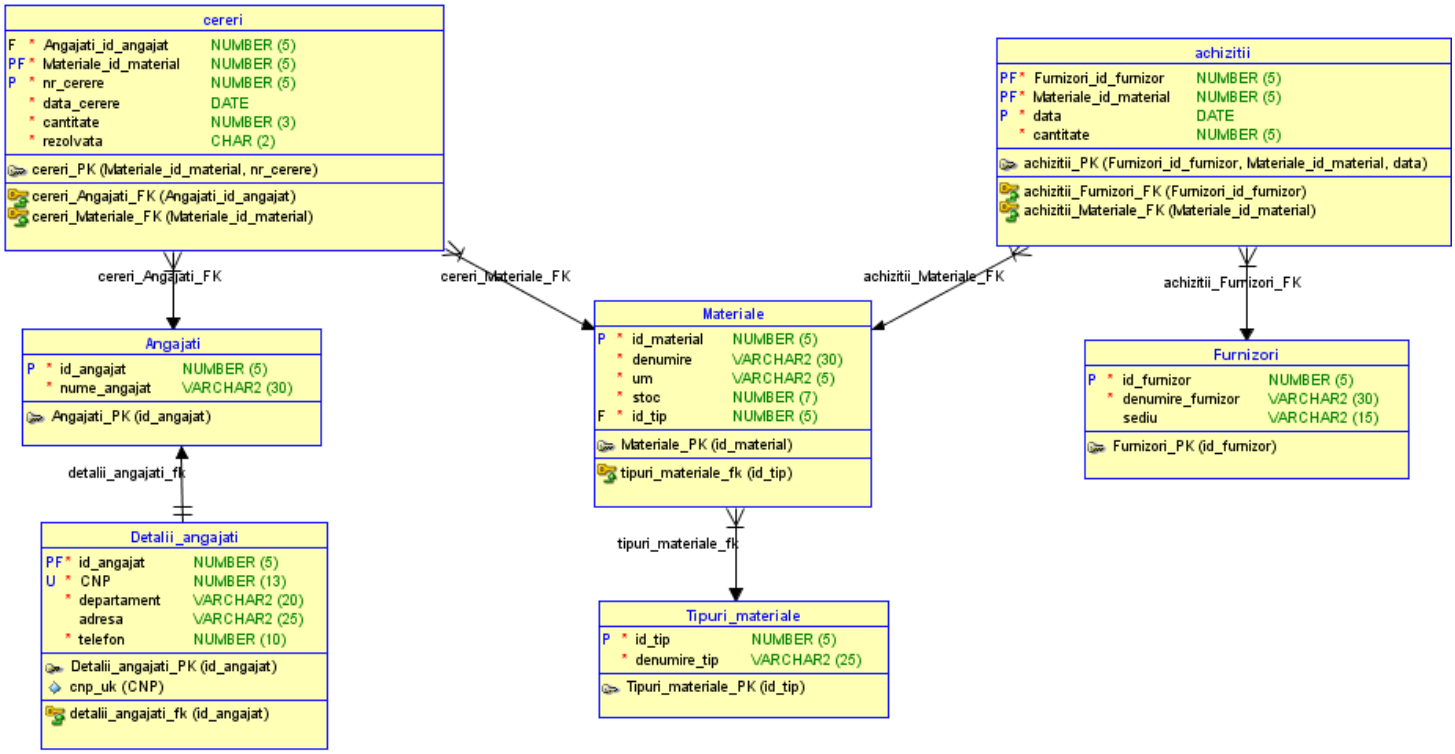
3.1 Modelul Logic



Tabelele din cadrul acestei baze de date sunt:

1. Materiale;
2. Tipuri_materiale;
3. Furnizori;
4. Angajati;
5. Detalii_angajati;
6. cereri;
7. achizitii;

3.2 Modelul Relațional



3.3 Relațiile dintre tabele

În proiectarea bazei de date s-au identificat toate tipurile de relații:

one-to-many, one-to-one si many-to-many.

- Între **materiale** si **tipuri_materiale** se stabilește o relație de tip **one-to-many** deoarece un tip de material se poate regăsi în mai multe materiale, însă reciproca nu este adevărată deoarece un anumit material nu poate fi de două tipuri. Legătura dintre cele două tabele este realizată prin câmpul *id_tip*;

- Între **angajați** și **detalii_angajați** avem o relație de tip **one-to-one** întrucât un angajat are un singur set de detalii, iar un set de detalii este specific unui singur angajat. Legătura dintre cele două tabele este realizată prin câmpul *id_angajat*;

- Între **angajați** și **materiale** există o relație **many-to-many** deoarece un angajat poate face cereri pentru mai multe materiale, iar un material poate fi prezent pe cererile mai multor angajați (și va fi dat în limita stocului) .
  În modelul relațional, în urma normalizării, legătura many-to-many dintre cele două entitati este "spartă" în două legături one-to-many, rezultând tabelul **cereri** care face corespondența între tabelele *angajati* si *materiale*.

- Între **materiale** și **furnizori** se stabilește o relație **many-to-many** deoarece un material poate fi achiziționat de la mai mulți furnizori, iar de la același furnizor se pot cumpăra mai multe materiale .
  Și aici, în modelul relațional, legătura many-to-many dintre cele două entitati este "spartă" în două legături one-to-many, rezultând tabelul **achiziții** care face corespondența între tabelele *materiale* si *furnizori*.

4. Descrierea constrângerilor folosite

4.1 Achiziții

Câmpurile din tabela **achiziții** sunt:

- furnizori_id_furnizor = un cod specific unui singur furnizor (CUI).
  Acest câmp este de tip *not null* deoarecere trebuie specificat id-ul furnizorului de la care se face achiziția și de tip *foreign_key* pentru că face legătura între tabela achiziții și tabela furnizori;
- materiale_id_material = codul de bare al unui anumit material.
  Acest câmp este *not null* deoarecere trebuie specificat id-ul materialului achiziționat și *foreign_key* deoarece face legătura între tabela achiziții și tabela materiale;
- data = data la care se face achiziția.
  Se consideră *not null* deoarece gestionarul trebuie să cunoască datele la care s-au făcut modificări în stoc;
- cantitate = cantitatea achiziționată.
  Se consideră *not null* deoarece gestionarul trebuie să știe cu cât se modifică stocul.

4.2 Angajați

Aici avem câmpurile:

- id_angajat = un cod specific unui singur angajat .
  Întrucât nu pot exista mai mulți angajați cu același cod, acest câmp este *primary_key (not null & unique);* **A fost generat printr-un mecanism de tip autoincrement;**
- nume_angajat = numele și prenumele angajatului
  Aici se întâlnesc două constrângeri:
  - *not null,* pentru că la fiecare angajat trebuie să se cunoască numele și prenumele său;
  - *check (nume_ck)* unde interzicem ca numele să conțină cifre.

4.3 Cereri

Câmpurile acestui tabel sunt:

- angajati_id_angajat = un cod specific unui singur angajat
  Este *not null* deoarecere trebuie specificat id-ul angajatului care face cererea și cheie străină (*foreign_key*), face legătura dintre tabelul cereri și tabelul angajați;
- materiale_id_material = codul de bare al unui anumit material.
  Acest câmp este de tip *not null* deoarecere trebuie specificat id-ul materialului pentru care se face cererea și de tip *foreign_key* pentru că face legătura între tabelul cereri și tabelul materiale;
- nr_cerere = numărul cererii
  Este de tip *not null* pentru că pe fiecare cerere trebuie scris acest număr, care este și unic fecărei cereri; **A fost generat printr-un mecanism de tip autoincrement;**
- data_cerere = data la care a fost depusă cererea.
  Este *not null* deoarece pe fiecare cerere trebuie menționată neapărat data;
- cantitate = cantitatea salocitată pentru un anumit material.
  Este de tip *not null* pentru că pe fiecare cerere, la fiecare id de material trebuie specificată și cantitatea solicitată;
- rezolvata = o stare a cererii care indică dacă angajatul a primit sau nu ceea ce a notat.
  Și aici se întâlnesc două constrângeri:
  - *not null,* pentru că pe fiecare cerere trebuie menționată starea;
  - *check (cerere_rezolvata),* unde permitem ca acest câmp să ia doar valorile da și nu.

Tupla (nr_cerere, Materiale_id_material) reprezintă *primary_key* -ul acestui tabel deoarece pe aceeași cerere nu poate apărea același id de material de mai multe ori.

Nu putea fi doar câmpul nr_cerere chia primară deoarece este ceva redundant. Dacă un angajat face o singură cerere în care specifică trei id-uri de materiale, în tabela cereri se fac trei înregistrări, dar toate au același număr de cerere.

4.4 Detalii_angajați

Întâlnim câmpurile:

- id_angajat = un cod specific unui singur angajat .
  Este *not null* deoarecere trebuie specificat id-ul angajatului care face cererea și cheie străină (*foreign_key*), face legătura dintre tabelul cereri și tabelul angajați;
- cnp = CNP-ul angajatului.
  Și aici se întâlnesc două constrângeri:
  - *not null,* pentru că la fiecare angajat trebuie menționat CNP-ul;
  - *unique (cnp_uk),* deoarece nu există 2 sau mai multe persoane care să aibă același cod numeri personal.
- departament = departamentul în care lucrează fiecare angajat.
  Este *not null* deoarecere trebuie menționat la fiecare angajat;
- adresa = adresa angajatului (nu are nicio constrângere);
- telefon = numărul de telefon.
  Avem o constrângere de tip *check (telefon_ck)* unde permitem ca numărul de telefon să conțină doar cifre.

4.5 Furnizori

Aici avem câmpurile:

- id_furnizor = un cod specific unui singur furnizor (CUI).
  Acest câmp este *primary_key (not null & unique) deoarece* nu pot exista mai mulți furnizori cu același CUI*;*
- denumire_furnizor = numele societății de la care primăria face achiziții.
  Este de tip *not null* deoarecere trebuie menționat la fiecare furnizor;
- sediu= sediul furnizorului (nu are nicio constrângere).

4.6 Materiale

Câmpurile din **materiale** sunt:

- id_material = codul de bare al unui anumit material.
  Acesta constitue și **cheia primară** *(primary_key)* a tabelului, deoarece nu pot exista două materiale cu același cod de bare.
- denumire = denumirea materialului
  La fiecare material trebuie să se știe denumirea sa, de aceea am folosit constrângerea *not null*;

- um = unitatea de masură a materialului
  Aici se întâlnesc mai multe constrângeri:
  - *not null* deoarece la fiecare material trebuie să cunoaștem acest detaliu;
  - *check* (um_ck) întrucât um se poate încadra doar într-o anumită listă de valori și anume: buc (bucata), l(litru), cutie sau top.
- stoc = cantitatea din inventar
  Și aici vom avea două constrângeri:
  - *not null* pentru că gestionarul trebuie să știe stocul fiecărui material;
  - *check* (stock_ck) deoarece stocul nu poate fi negativ.
- id_tip = categoria din care face parte materialul.
  Este cheie străină (*foreign_key*), face legătura dintre tabelul materiale și tabelul tipuri_materiale

4.7 Tipuri_materiale

Aici avem câmpurile:

- id_tip = codul categoriei din care face parte materialul
  Acest câmp reprezintă **cheia primară** *(primary_key)* a tabelului, deoarece un material nu se poate încadra în mai multe categorii. De asemenea, face legătura dintre tabelul materiale și tabelul tipuri_materiale;
  A fost generat printr-un mecanism de tip autoincrement.
- denumire_tip = denumirea unei anumite categorii de materiale
  Fiecare categorie trebuie să aibă un anumit nume și de aceea am folosit constrângerea *not null*;

5. Modalitatea de conectare la baza de date

Pentru a mă conecta la baza de date, am avut nevoie de cx_Oracle, un modul de extensie Python care permite accesul la Oracle Database.

Codul folosit pentru conectare a fost:

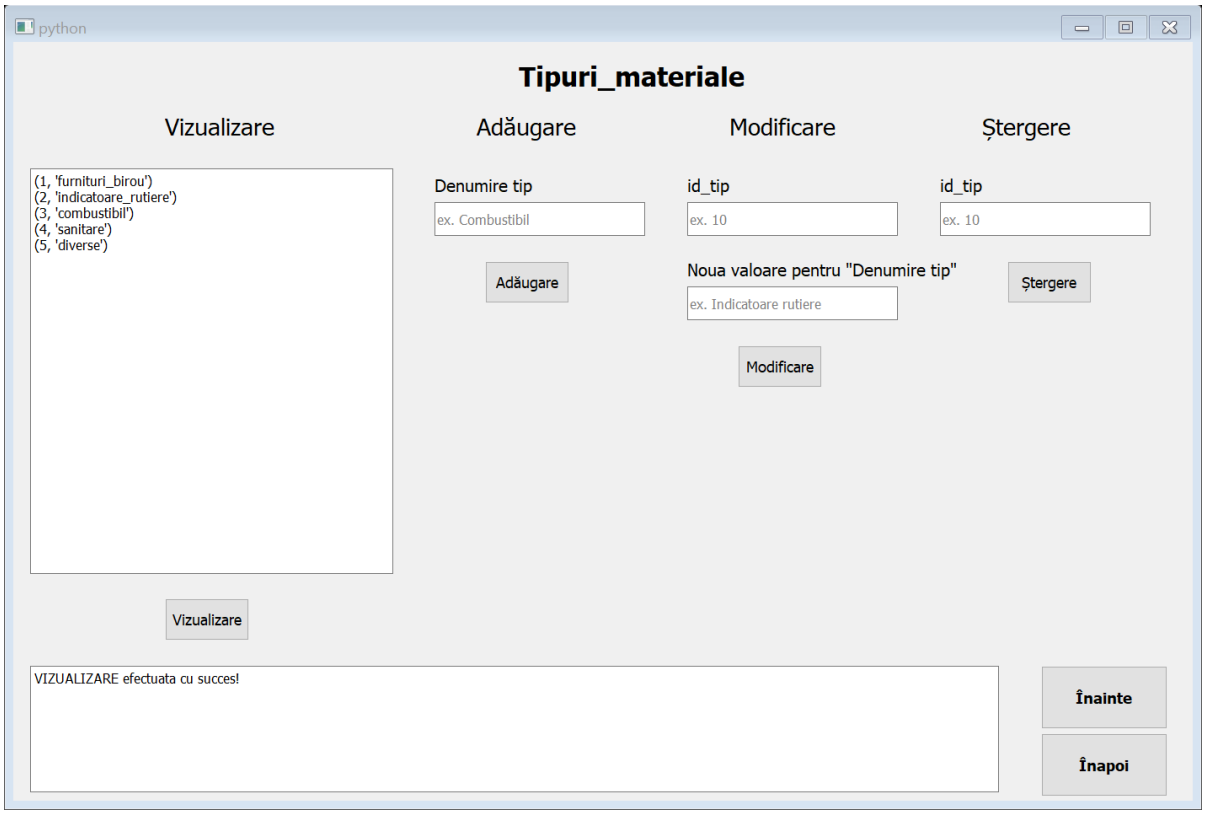
```
import cx_Oracle

con = cx_Oracle.connect(user="bd124", password="bd124",  dsn="bd-dc.cs.tuiasi.ro:1539/orcl")
```

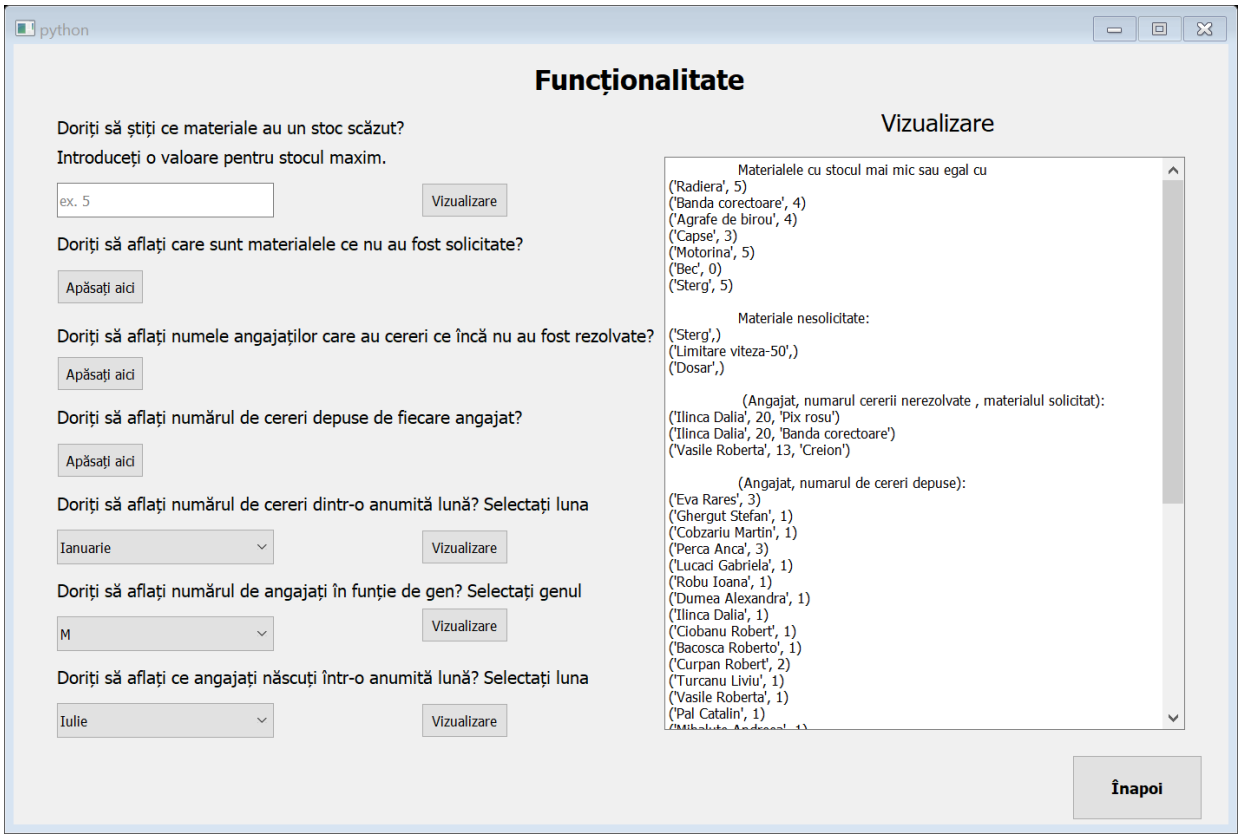
Pentru a testa conexiunea, am rulat următorul cod

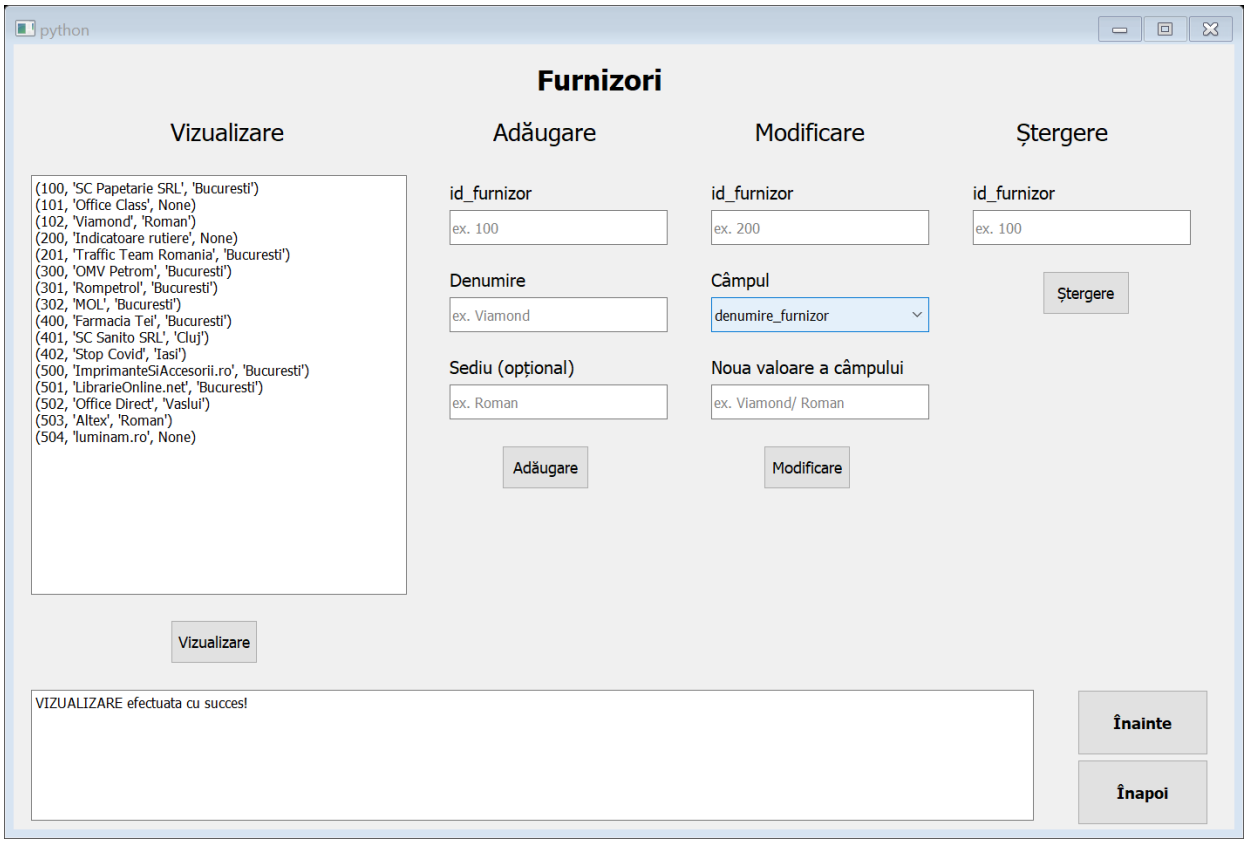
```
import cx_Oracle
con = cx_Oracle.connect(user="bd124", password="bd124", dsn="bd-dc.cs.tuiasi.ro:1539/orcl")
cursor = con.cursor()
sql_command = "SELECT * FROM Tipuri_materiale"
cursor.execute(sql_command)
for result in cursor:
    print(result)
```

și am observat că datele corespund cu cele rezultate din rularea comenzii „SELECT * FROM Tipuri_materiale;” în sqldeveloper.

6. Capturi de ecran din interfața aplicației




python
Funcționalitate
Doriți să știți ce materiale au un stoc scăzut?
Introduceți o valoare pentru stocul maxim.
ex. 5
Vizualizare
Doriți să aflați care sunt materialele ce nu au fost solicitate?
Apăsați aici
Doriți să aflați numele angajaților care au cereri ce încă nu au fost rezolvate?
Apăsați aici
Doriți să aflați numărul de cereri depuse de fiecare angajat?
Apăsați aici
Doriți să aflați numărul de cereri dintr-o anumită lună? Selectați luna
Ianuarie
Vizualizare
Doriți să aflați numărul de angajați în funție de gen? Selectați genul
M
Vizualizare
Doriți să aflați ce angajați născuți într-o anumită lună? Selectați luna
Iulie
Vizualizare
Vizualizare
Materialele cu stocul mai mic sau egal cu
('Radiera', 5)
('Banda corectoare', 4)
('Agrafe de birou', 4)
('Capse', 3)
('Motorina', 5)
('Bec', 0)
('Sterg', 5)
Materiale nesolicitate:
('Sterg',)
('Limitare viteza-50',)
('Dosar',)
(Angajat, numarul cererii nerezolvate , materialul solicitat):
('Ilinca Dalia', 20, 'Pix rosu')
('Ilinca Dalia', 20, 'Banda corectoare')
('Vasile Roberta', 13, 'Creion')
(Angajat, numarul de cereri depuse):
('Eva Rares', 3)
('Ghergut Stefan', 1)
('Cobzariu Martin', 1)
('Perca Anca', 3)
('Lucaci Gabriela', 1)
('Robu Ioana', 1)
('Dumea Alexandra', 1)
('Ilinca Dalia', 1)
('Ciobanu Robert', 1)
('Bacosca Roberto', 1)
('Curpan Robert', 2)
('Turcanu Liviu', 1)
('Vasile Roberta', 1)
('Pal Catalin', 1)
Înapoi


python
Furnizori
Vizualizare
Adăugare
Modificare
Ștergere
(100, 'SC Papetarie SRL', 'Bucuresti')
(101, 'Office Class', None)
(102, 'Viamond', 'Roman')
(200, 'Indicatoare rutiere', None)
(201, 'Traffic Team Romania', 'Bucuresti')
(300, 'OMV Petrom', 'Bucuresti')
(301, 'Rompetrol', 'Bucuresti')
(302, 'MOL', 'Bucuresti')
(400, 'Farmacia Tei', 'Bucuresti')
(401, 'SC Sanito SRL', 'Cluj')
(402, 'Stop Covid', 'Iasi')
(500, 'ImprimanteSiAccesorii.ro', 'Bucuresti')
(501, 'LibrarieOnline.net', 'Bucuresti')
(502, 'Office Direct', 'Vaslui')
(503, 'Altex', 'Roman')
(504, 'luminam.ro', None)
Vizualizare
id_furnizor
ex. 100
Denumire
ex. Viamond
Sediu (opțional)
ex. Roman
Adăugare
id_furnizor
ex. 200
Câmpul
denumire_furnizor
Noua valoare a câmpului
ex. Viamond/ Roman
Modificare
id_furnizor
ex. 100
Ștergere
VIZUALIZARE efectuata cu succes!
Înainte
Înapoi

Exemple de cod

Voi lua ca exemplu Tabela “Furnizori”.

După ce am generat codul cu comanda amintită la punctul **2.Tehologiile folosite,** am creat câte o funcționalitate pentru fiecare buton.

În setupUi:

```
self.VizualizarBtt.clicked.connect(self.View)
self.AdaugareBtt.clicked.connect(self.Insert)
self.ModificareBtt.clicked.connect(self.Update)
self.StergereBtt.clicked.connect(self.Delete)
self.Furnizori_Inainte.clicked.connect(self.Inainte)
self.Furnizori_Inapoi.clicked.connect(self.Inapoi)
```

În retranslateUi:

```
def View(self):
    sql_command = "SELECT * FROM Furnizori"
    cursor = con.cursor()
    cursor.execute(sql_command)
    self.Viz.clear()
    self.feedback.clear()
    self.feedback.append("VIZUALIZARE efectuata cu succes!")
    for result in cursor:
        self.Viz.append(str(result))

def Insert(self):
    if self.id_furnizor_a.text() != "" and self.denumire.text() != "":
        if self.sediu.text() == "":
            sql_command = "INSERT INTO Furnizori VALUES(" + self.id_furnizor_a.text() + ",\'" +
self.denumire.text()+"\', NULL)"
        else:
            sql_command = "INSERT INTO Furnizori VALUES(" + self.id_furnizor_a.text() + ",\'" +
self.denumire.text()+"\'" +",\'" + self.sediu.text()+"\')"
        cursor = con.cursor()
        try:
            cursor.execute(sql_command)
            self.id_furnizor_a.clear()
            self.denumire.clear()
            self.sediu.clear()
            cursor.execute('commit')
            self.feedback.clear()
            self.feedback.append("INSERARE efectuata cu succes! Pentru a vedea modificarile din tabela
apasati pe Vizualizare.")

        except Exception as a:
            self.feedback.append(str(a))
    else:
        self.feedback.append("Un camp a ramas necompletat!")

def Update(self):
```

```
    cursor = con.cursor()
    if self.id_furnizor_m.text() !="":
        cursor.execute("SELECT id_furnizor FROM Furnizori")
        ok=False
        for id in cursor:
            if self.id_furnizor_m.text() == str(id[0]):
                ok=True

        if ok == True:
            if self.newVal.text() !="":
                sql_command = "UPDATE Furnizori SET "+ self.camp.currentText() +"= \'" +
self.newVal.text() +"\' WHERE id_furnizor=" + self.id_furnizor_m.text()
                cursor.execute(sql_command)
                cursor.execute('commit')
                self.id_furnizor_m.clear()
                self.newVal.clear()
                self.feedback.clear()
                self.feedback.append("MODIFICARE efectuata cu succes! Pentru a vedea modificarile din
tabela apasati pe Vizualizare.")

            else:
                self.feedback.clear()
                self.feedback.append("Campul pentru noua valoare a ramas necompletat!")
        else:
            self.feedback.clear()
            self.feedback.append("Acest id_furnizor nu exista in baza de date!")
    else:
        self.feedback.clear()
        self.feedback.append("Un camp a ramas necompletat!")

def Delete(self):
    sql_command = "DELETE FROM Furnizori WHERE id_furnizor=" + self.id_furnizor_s.text()
    cursor = con.cursor()
    try:
        self.id_furnizor_s.clear()
        cursor.execute(sql_command)
        cursor.execute('commit')
        self.feedback.clear()
        self.feedback.append("STERGERE efectuata cu succes! Pentru a vedea modificarile din tabela
apasati pe Vizualizare.")

    except Exception as a:
        self.feedback.append(str(a))

def Inainte(self):
    widget.setCurrentIndex(widget.currentIndex() + 1)

def Inapoi(self):
    widget.setCurrentIndex(widget.currentIndex() - 1)
```