**UNIVERSITATEA POLITEHNICĂ BUCUREȘTI**

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

**Proiect**

Programarea interfețelor pentru baze de date - Python

București, 2023

Student:

Dobrin Liviu Andrei

Anul III, Grupa 432A

Profesor coordonator:

Ș.l. Dr. Ing. Pupezescu Valentin

1. **Cerință**

Creaţi două aplicaţii care să conţină o bază de date creată în sistemul de gestiune a bazelor de date MySql şi două interfeţe la aceasta (baza de date este comună). La crearea interfeţelor se vor folosi două tehnologii(la alegere - ex.: JSP, Hibernate, JPA, .NET, Python etc.).

Interfetele vor trebui sa permita utilizatorului sa execute urmatoarele operatii pe toate tabele: vizualizare, adaugare, modificare si stergere de date. Vizualizarea tabelelor de legatura va presupune vizualizarea datelor referite din celelalte tabele.

Pentru tema individuala primita, am ales 2 tehnologii diferite: JSP si Phyton. In aceasta prezentare ne vom orienta atentia catre tehnologia Python. Asocierea pentru tabelele din baza de date este de M:N.

1. **Introducere**

Ca temă am abordate gestionarea studentilor și a cursurilor, printr-o asociere M:N cu ajutorul unei tabele intermediare, cea a inscrierilor, creând o interfață Web ce operează baza de date în care sunt stocate toate datele. Printre funcționalitățile implementate în cadrul acestei etape a proiectului se enumeră vizualizarea datelor, adăugarea lor, modificarea și posibilitatea ștergerii acestora.

Un sistem de gestiune al bazelor de date (SGBD) este o aplicație care gestionează toate aspectele ce țin de o bază de date. Un SGBD are următoarele funcții:

- Asigură crearea bazelor de date;

- Asigură salvarea în siguranță a datelor;

- Conține un set de instrucțiuni minime pentru transpunerea în limbaj de programare a unui algoritm elementar;

- Conține un set de comenzi detaliate pentru gestionarea datelor salvate în baza de date;

- Asigură o interfață ce poate fi utilizată atât de programatori, cât și de utilizatorii neexperimentați;

- Conține un set de instrucțiuni ce permite programatorului să creeze o interfață cu utilizatorul.

În cadrul acestei părți finale a proiectului a fost implementată ca interfață pentru baza de date creată tehnologia **Python,** facand un **GUI (graphical user interface),** o interfață digitală în care un utilizator interacționează cu componente grafice, cum ar fi pictograme, butoane și meniuri.

**3.Implementare**

Tehnologia folosită pentru crearea bazei de date este **MySql**.

**3.1 Ce este MySQL?**

MySQL este un sistem de gestionare a bazelor de date relationale open source care este utilizat in principal pentru aplicatiile online. MySQL poate crea și gestiona baze de date foarte utile (cum ar fi informatii despre angajati, inventar și multe altele), la fel ca alte sisteme, cum ar fi popularul Microsoft Access. In timp ce Microsoft Access, MySQL și alte sisteme de gestionare a bazelor de date servesc scopuri similare (de a gazdui datele), utilizarea difera foarte mult.

MySQL este componenta integrata a platformelor LAMP sau WAMP (Linux/Windows-Apache-MySQL-PHP/Perl/Python). Popularitatea sa ca aplicatie web este strans legata de cea a PHP-ului care este adesea combinat cu MySQL și denumit Duo-ul Dinamic. In multe carți de specialitate este precizat faptul ca MySQL este mult mai usor de invatat și folosit decat multe din aplicatiile de gestiune a bazelor de date, ca exemplu comanda de iesire fiind una simpla și evidenta: „exit” sau „quit”.

**3.2 Cum administreaza MySQL bazele de date?**

Pentru a administra bazele de date MySQL se poate folosi modul linie de comanda sau, prin descarcare de pe internet, o interfata grafica: MySQL Administrator și MySQL Query Browser. Un alt instrument de management al acestor baze de date este aplicatia gratuita, scrisa în PHP, phpMyAdmin.

**3.3 De ce este atat de util MySQL?**

Baza de date MySQL este folosita in principal ca mijloc de a stoca date pentru aplicatii mari, bazate pe web. Site-uri precum WordPress, iStock, GitHub, Facebook, NASA, Marina SUA, Tesla, Scholastic, Spotify, YouTube, Netflix, Glasses Direct, Symantec (si multe altele) folosesc baza de date MySQL ca mijloc de stocare a datelor din interiorul sau exteriorul site-urilor web și serviciilor interne.

Am realizat o bază de date în MySQL WORKBENCH , ea conținând 2 tabele : **studenti** și **cursuri**. Asocierea dintre ele este  de tipul M:N , ceea  ce presupune crearea unei tabele intermediare , pe care am numit-o **inscrieri.**

Tabela **studenti** contine urmatoare coloane:

* idstudent, de tip BIGINT , cheie primară, Not Null, Auto-Increment
* nume , de tip VARCHAR
* prenume, de tip VARCHAR
* cnp , de tip VARCHAR
* anstudiu , de tip BIGINT
* facultate , de tip VARCHAR

Tabela **cursuri** contine urmatoarele coloane :

* idcurs, de tip BIGINT, cheie primară, Not Null, Auto-Increment
* denumire, de tip VARCHAR
* anrecomandat, de tip BIGINT
* facultaterecomandata, de tip VARCHAR
* domeniu, de tip VARCHAR
* semestru, de tip BIGINT

Tabela intermediară **inscrieri** contine urmatoarele coloane :

* idinscriere, de tip BIGINT, cheie primară, Not Null, Auto-Increment
* idstudent, de tip BIGINT
* idcurs, de tip BIGINT
* datainscriere , de tip DATE

Datorită asocierii M:N ,  idstudent si idcurs sunt chei străine(FK) pentru tabela **inscrieri.** Cheile primare(PK) corespunzătoare fiecărei tabele au fost setate cu urmatoarele proprietăți:  not null (NN) si auto-increment (AI).

**Cheie primara**: Una sau mai multe coloane ale caror valori identifică în mod unic toate liniile unui tabel.

A screenshot of a computer

Description automatically generatedFigura 1.Diagrama asociată tabelelor

Aceasta ofera o reprezentare logica detaliata a datelor celor 3 tabele, a relatiilor dintre ele.

**Ce relatii exista intre cele trei tabele?**

* Intre studenti si inscrieri exista o asociere de tip 1:N
* Intre studenti si cursuri exista o asociere de tip M:N
* Intre cursuri si inscrieri exista o asociere de tip 1:N

Pentru realizarea interfeței dintre utilizator și bază de date am ales tehnologia **Python.**

Python este un limbaj de programare de nivel inalt, interpretat si generalist si a fost lansat pentru prima data in 1991, de catre Guido van Rossum. Este un limbaj simplu, care pune accent pe simplitatea codului. Python este cunoscut pentru simplitatea si claritatea sintaxei. De asemenea, acesta dispune de o bogata biblioteca standard. Cu toate acestea, a fost nevoie de a descarca in interiorul environment-ului biblioteci suplimentare pentru a putea sa ne conectam la baza de date si a executa comenzi asupra acesteia. A computer screen shot of a diagram

Description automatically generated

Figura 2.Diagrama UML aferenta fisierului main.py a interfetei

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Figura 3.Functiile aferente programului main.py

Fișierul main.py, reprezintă “motorul” care realizează operațiile pentru conectarea bazei de date și gestionarea tabelelor și a datelor. In acest fisier regasim si functiile pentru vizualizare tabelelor, inserare de date în tabele, ștergere de date existente, modificare date, dar și metode ce ne permit returnarea unei linii dintr-o tabelă folosind ID-ul acesteia.

Pagina principală a interfeței ne oferă acces la panoul de control al aplicației noastre prin care putem adăuga date noi, putem modifica datele existente sau le putem șterge din tabela pe care dorim să operăm.

A screenshot of a computer

Description automatically generated

Figura 4.Pagina principala a interfeței

Codul aferent pentru această pagina este:

class StartPage(tk.Frame):  
 def \_\_init\_\_(self, parent, controller):  
 tk.Frame.\_\_init\_\_(self, parent)  
 self.controller = controller  
  
 label = tk.Label(self, text="PIBD CRUD App", font=("Helvetica", 24))  
 label.pack(pady=20)  
  
 label = tk.Label(self, text="Andrei Dobrin 432A", font=("Helvetica", 24))  
 label.pack(pady=20)  
  
 button\_frame = ttk.Frame(self)  
 button\_frame.pack(pady=10)  
  
 db\_buttons = []  
 db\_names = ['studenti', 'cursuri', 'inscrieri']  
 for db\_name in db\_names:  
 button = ttk.Button(button\_frame, text=db\_name[0].upper()+db\_name[1:len(db\_name)], command=lambda name=db\_name: self.show\_crud\_interface(name))  
 button.pack(side=tk.LEFT, padx=10)  
 db\_buttons.append(button)  
  
 def show\_crud\_interface(self, db\_name):  
 self.controller.show\_frame(CrudWindow, db\_name)

După cum se poate observa, utilizatorul are posibilitatea de a accesa pagina aferentă vizualizării și gestionării datelor pentru **studenti**, **cursuri** și **inscrieri**, stocate în tabelele cu aceleași nume.

Prin apăsarea unuia dintre butoanele:**Studenti, Cursuri, Inscrieri**vom ajunge in pagina fiecăreia dintre tabelele pe care dorim să le vizualizăm sau să le gestionăm. Paginile arată in mod similar, diferențele fiind datele care se adaugă/modifică/șterg.

Toate metodele pentru toate cele trei tabele sunt similare, asa ca voi prezenta doar pentru tabela studenti.

A screenshot of a computer

Description automatically generated

Figura 5.Pagina destinată vizualizării studentilor (tabela studenti)

Dacă se dorește adăugarea unui nou student, utilizatorul va apăsa pe butonul **Create** îl va redirecționa către pagina ce conține formularul ce necesită a fi completat pentru noua intrare.

A screenshot of a computer

Description automatically generated

Figura 6.Pagina destinată adăugării unui nou student

Codul aferent pentru această operatiune este:

def create(self):  
 data = self.show\_input\_dialog()  
 if data is not None:  
 if self.db\_name == 'studenti':  
 # validate data  
 if not data[0].isalpha():  
 messagebox.showerror("Error", "Name must contain only letters")  
 return  
 if not data[1].isalpha():  
 messagebox.showerror("Error", "Prenume must contain only letters")  
 return  
 if len(data[2]) != 13:  
 messagebox.showerror("Error", "CNP must have 13 characters")  
 return  
 if not data[3].isnumeric():  
 messagebox.showerror("Error", "An studiu must be a number")  
 return  
 if not data[4].isalpha():  
 messagebox.showerror("Error", "Facultate must contain only letters")  
 return  
  
 nume, prenume, cnp, anstudiu, facultate = data  
 new\_student = Student(nume=nume, prenume=prenume, cnp=cnp, anstudiu=anstudiu, facultate=facultate)  
 session.add(new\_student)  
 session.commit()  
 self.show\_data()

Implementarea codului pentru adăugarea unui nou curs este aproximativ identică, fiind diferite doar numele acestora.

Pentru ca toate datele sa fie corecte, am pus anumite restrictii pentru ca datele adaugate sa respecte anumite tipuri si conditii.

Codul aferent pentru această operatiune este:

def validate\_input\_data(self, input\_data):  
 if self.db\_name == 'studenti':  
 # Validare pentru tabelul 'studenti'  
 if not input\_data[0].isalpha():  
 messagebox.showerror("Error", "Name must contain only letters")  
 return False  
 if not input\_data[1].isalpha():  
 messagebox.showerror("Error", "Prenume must contain only letters")  
 return False  
 if len(input\_data[2]) != 13:  
 messagebox.showerror("Error", "CNP must have 13 characters")  
 return False  
 if not input\_data[3].isnumeric():  
 messagebox.showerror("Error", "An studiu must be a number")  
 return False  
 if not input\_data[4].isalpha():  
 messagebox.showerror("Error", "Facultate must contain only letters")  
 return False

Pentru celelalte tabele, funcțiile sunt asemanătoare, diferențele fiind întâlnite la ce se adaugă, numarul de coloane sau tipul acestora.

In urma adaugarilor, daca toate restrictiile sunt respecatate pagina de adaugare se va inchide si vom vedea din nou tabelul, cu date actualizate. In cazul in care una dintre restrictii nu este respectata, va aparea un mesaj de eroare pe ecran si fereastra de adaugari se va inchide, fara ca adaugarile sa fie efectuate.

A screenshot of a computer error

Description automatically generated A screenshot of a computer error

Description automatically generated

Figura 7.Tipuri de mesaje de eroare daca nu sunt respectate restrictiile

Pentru modificarea unui student din tabel vom selecta linia pe care dorim să o modificăm. După selectarea liniei, apăsăm butonul **Update** care ne va duce la o pagină cu un form ce conține detaliile liniei selectate, în acest mod putând sa modificăm ce dorim. După ce se modifică datele dorite, se va apăsa butonul **OK** și vom fi redirecționați către o pagină care ne confirmă ca modficările au fost efectuate.

A screenshot of a computer

Description automatically generated

Figura 8.Pagina aferentă modificării unei linii

Codul aferent pentru modificarea în tabelă:

def update(self):  
 selected\_ids = self.get\_selected\_item\_ids()  
 if selected\_ids:  
 # Obține datele existente  
 existing\_data = self.get\_existing\_data(selected\_ids[0])  
  
 # Afișează dialogul cu datele existente completate  
 data = self.show\_input\_dialog(existing\_data)  
  
 if data is not None:  
 # Validare date introduse  
 if not self.validate\_input\_data(data):  
 return  
  
 if self.db\_name == 'studenti':  
 for item\_id in selected\_ids:  
 student = session.query(Student).get(item\_id)  
 if student:  
 student.nume, student.prenume, student.cnp, student.anstudiu, student.facultate = data  
 session.commit()  
 self.show\_data()

In urma modificarilor efectuate, daca toate restrictiile sunt respecatate pagina de modificare se va inchide si vom vedea din nou tabelul, cu date actualizate. In cazul in care una dintre restrictii nu este respectata, va aparea un mesaj de eroare pe ecran si fereastra de modificari se va inchide, fara ca modificarile sa fie efectuate.

A screenshot of a computer error

Description automatically generated A screenshot of a computer error

Description automatically generated

Figura 9.Tipuri de mesaje de eroare daca nu sunt respectate restrictiile

Datorită faptului că funcția are ca variabilă db\_name-ul tabela, această funcție este utilizata și pentru modificarea celelorlate tabele, **cursuri** și **inscrieri**. Prin acel db\_name se recunoaște tabela din care se realizează modificările.

În schimb, dacă utilizatorul dorește să șteargă date, acesta va trebui să selecteze linia prin selectarea acesteia, urmând apoi să apese butonul **Delete**.

A screenshot of a computer

Description automatically generated

Figura 10.Modul prin care se selectează studentul care dorește a fi șters

Codul aferent pentru ștergerea din tabelă:

def delete(self):  
 selected\_ids = self.get\_selected\_item\_ids()  
 if selected\_ids:  
 confirm = messagebox.askyesno("Delete", "Are you sure you want to delete the selected items?")  
 if confirm:  
 for item\_id in selected\_ids:  
 if self.db\_name == 'studenti':  
 student = session.query(Student).get(item\_id)  
 if student:  
 session.delete(student)

La fel ca în cazul funcției de modificare din tabelă, db\_name-ul tabela este cel care ne indica din ce tabelă se vor șterge datele corespunzătoare id-ului dat de checkbox-ul bifat. Astfel, variabila și vectorii de tip string parametrizează tabela și câmpurile, putându-se astfel modifica un număr variabil de valori în funcție de tabela dată ca parametru.

Cum paginile aferente pentru tabela cursuri sunt asemănătoare, la fel și în majoritatea cazurilor tabelei inscrieri, în continuare voi prezenta doar pagina aferentă vederii tabelei **Inscrieri** și codul acesteia.

A screenshot of a computer

Description automatically generated

Figura 11.Pagina pentru a vedea datele din tabela inscrieri

Diferența față de celelalte 2 pagini, este că fiind tabelă intermediara, am atașat și câteva dintre detaliile studentului și a cursului care au inscrierea, prin preluarea acestora din tabelele corespunzătoare.

Codul aferent pentru vederea tabelei folder în acest mod:

class Inscriere(Base):  
 \_\_tablename\_\_ = 'inscrieri'  
 idinscriere = Column(Integer, primary\_key=True)  
 idstudent = Column(Integer, ForeignKey('studenti.idstudent'))  
 idcurs = Column(Integer, ForeignKey('cursuri.idcurs'))  
 data\_inscriere = Column(Date)  
 student = relationship('Student', back\_populates='inscrieri')  
 curs = relationship('Curs', back\_populates='inscrieri')

Pentru ca toate inscrierile sa fie corecte si sa nu existe studenti care se inscriu la cursuri care nu exista, sau sa existe inscrieri la curs ale unor studenti inexistenti am adaugat si aici cateva restrictii.

A screenshot of a computer error

Description automatically generated A screenshot of a computer error

Description automatically generated

Figura 12.Tipuri de mesaje de eroare daca nu sunt respectate restrictiile

Codul aferent pentru această operatiune este:

def validate\_input\_data(self, input\_data):  
 if self.db\_name == 'studenti':  
 # Validare pentru tabelul 'studenti'  
 if not input\_data[0].isalpha():  
 messagebox.showerror("Error", "Name must contain only letters")  
 return False  
 if not input\_data[1].isalpha():  
 messagebox.showerror("Error", "Prenume must contain only letters")  
 return False  
 if len(input\_data[2]) != 13:  
 messagebox.showerror("Error", "CNP must have 13 characters")  
 return False  
 if not input\_data[3].isnumeric():  
 messagebox.showerror("Error", "An studiu must be a number")  
 return False  
 if not input\_data[4].isalpha():  
 messagebox.showerror("Error", "Facultate must contain only letters")  
 return False  
  
 elif self.db\_name == 'cursuri':  
 # Validare pentru tabelul 'cursuri'  
 if not input\_data[0].isalpha():  
 messagebox.showerror("Error", "Denumire must contain only letters")  
 return False  
 if not input\_data[1].isnumeric():  
 messagebox.showerror("Error", "An recomandat must be a number")  
 return False  
 if not input\_data[4].isnumeric():  
 messagebox.showerror("Error", "Semestru must be a number")  
 return False  
  
 elif self.db\_name == 'inscrieri':  
 # Validare pentru tabelul 'inscrieri'  
 if not input\_data[0].isnumeric() or not input\_data[1].isnumeric():  
 messagebox.showerror("Error", "ID Student and ID Curs must be numbers")  
 return  
 if session.query(Student).get(input\_data[0]) is None:  
 messagebox.showerror("Error", "ID Student does not exist")  
 return  
 if session.query(Curs).get(input\_data[1]) is None:  
 messagebox.showerror("Error", "ID Curs does not exist")  
 return  
 return True

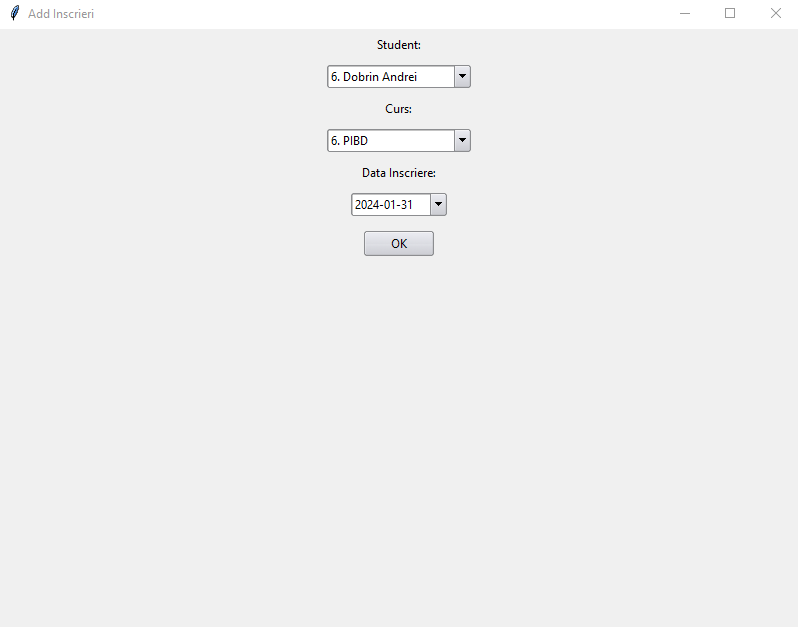
Si in plus am facut ca studentii si cursurile sa poata fii selectati din doua liste, pentru a nu exista greseli si astfel incat inscrierile sa poata fi realizate mai usor.

Figura 13. Pagina destinată adăugării unei noi inscrieri

Codul aferent pentru această operatiune este:

elif self.db\_name == 'inscrieri':  
 label\_names = ['Student', 'Curs', 'Data Inscriere']  
 dropdown\_values\_student = [str(student.idstudent) + '. ' + student.nume + ' ' + student.prenume for student  
 in session.query(Student).all()]  
 dropdown\_values\_curs = [str(curs.idcurs) + '. ' + curs.denumire for curs in session.query(Curs).all()]

Pentru a nu inchide aplicatia si sa pierdem unele modificari nesalvate, sau sa nu o inchidem din greseala, am adaugat o atentionare atunci cand apasam pe butonul de inchidere.

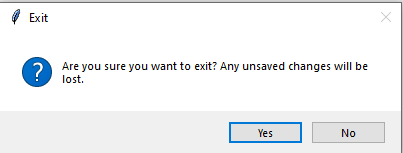


Figura 14.Mesajul de atentionare aparut in urma apasarii butonului de inchidere

Codul aferent pentru această operatiune este:

def event\_delete(self, event):  
 confirm = messagebox.askyesno("Exit", "Are you sure you want to exit? Any unsaved changes will be lost.")  
 if confirm:  
 self.destroy()

**4.Concluzii**

În prezent, bazele de date sunt utilizate în mod extensiv în toate domeniile, iar fiecare companie, în special cele cu caracter de producție sau comercializare, implementează cu siguranță un sistem de bază de date. Pentru utilizarea fiabilă și corectă a unei baze de date, este esențial să se urmărească realizarea unei arhitecturi care să permită separarea funcționalităților, respectiv a părții de back-end și a celei de front-end.

Cursul de Programarea Interfețelor pentru Baze de Date ne-a oferit o percepție mai amplă asupra conceptului de baze de date și o înțelegere solidă a conceptelor și tehnicilor necesare pentru a gestiona și utiliza bazele de date eficient. Am avut oportunitatea de a învăța cum putem crea și administra o bază de date folosind SGBD-ul MySQL.

În ceea ce privește tehnologiile MySQL si Python putem spune că reprezintă un mijloc prin care se poate crea o astfel de interfață într-o manieră elegantă, avânt totodată și avantajul unei implementări ușoare și exacte.

Folosind interfata creata, o universitate, de exemplu, isi poate gestiona cursurile, adaugand noi studenti, noi cursuri si inscrieri noi. Nu le trebuie un angajat special pentru gestionarea datelor, orice angajat existent poate face acest lucru folosind aceasta interfata.

Totodata, aplicatia realizata in tehnologia Python si prezentata in cadrul acestui proiect, permite utilizatorului sa efectueze operatiile cerute pe baza de date creata in MySQL: afisare, adaugare, modificare si stergere.

**5. Bibliografie**

|  |  |
| --- | --- |
|  | Wikipedia, "MySQL," 05 02 2022. [Online]. Available: https://ro.wikipedia.org/wiki/MySQL. |
|  | D. Avasilica, "Introducere in SQL," [Online]. Available: http://etutoriale.ro/articles/100/1/Introducere-in-SQL/. [Accessed 05 02 2022]. |

Surse folosite pentru implementarea proiectului:

1. <https://www.geeksforgeeks.org/web-development/>
2. Curs de Programare a Interfetelor pentru Baze de Date (V. Pupezescu)
3. -https://www.w3schools.com/python/python\_mysql\_getstarted.asp
4. -https://stackoverflow.com/questions/372885/how-do-i-connect-to-a-mysql-database-in-python