UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI” IAŞI
FACULTATEA AUTOMATICĂ ŞI CALCULATOARE
SPECIALIZAREA CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI

DISCIPLINA BAZE DE DATE

Gestiunea unui magazin cu articole pentru animale de companie

Coordonator,
Mironeanu Cătălin

Student,
Pandeleanu Cosmin-Constantin

Iaşi, 2021

INTRODUCERE

Titlu temă : Gestiunea unui magazin cu articole pentru animale de companie

Analiza, proiectarea și implementarea unei baze de date MySQL și a unei aplicații care să modeleze activitatea unui magazin cu privire la gestiunea produselor și a clienților.

Descrierea cerințelor

Baza de date cuprinde un număr de 7 tabele:

- ✓ clienți ;
- ✓ categorii_animale ;
- ✓ detalii_clienți ;
- ✓ furnizori ;
- ✓ produse ;
- ✓ tipuri_produse ;
- ✓ vânzări ;

Prin această aplicație am incercat să implementez vizual funcționalitatea principalelor funcții de interogare a unei baze de date. Astfel putem modifica/accesa o bază de date cu ușurință datorită uneltelor pe care python-ul le pune la dispoziție.

Principalele funcții folosite sunt:

- Select
- Insert
- Update
- Delete

Modul de organizare al proiectului

Volumul mare de informaţii existente în cazul unui magazin cu o gamă foarte variată de produse și numeroși clienţi determină necesitatea fluidizării fluxurilor de date ce traversează sistemele informaționale ale magazinului, gestiunea acestora fiind destul de grea.

Informațiile de care avem nevoie sunt cele legate de:

- **clienți:** datele referitoare la clienți se află în tabelele *clienți* și respectiv în *detalii_clienți*, datele care ne interesează cel mai mult sunt numele clientului, numărul cardului de fidelitate care se află in tabela *clienți*, iar datele referitoare la adresa de e-mail, data nașterii, genul, orașul și adresa la care locuiește se află in tabelă *detalii_clienți*.

Constrângerile folosite sunt:

- de tip primary key pe *nr_card* pentru a avea un număr de card unic și pentru a putea face legătura între cele două tabele, *clienți* și *detalii_clienți.*
- de tip check pe *nume_client* pentru a verifica că lungimea numelui clientului este mai mare de 3 litere și că numele introdus nu conține cifre.
- de tip unique pe *nume_client* pentru ca numele să fie unic, să nu existe două persoane cu același nume.
- de tip check pe *e-mail* pentru a verifica că o adresă de e-mail introdusă este validă.
- de tip unique pe *e-mail* pnetru a nu existe două persoane cu adrese de e-mail identice.
- de tip foreign key pentru *nr_card* pentru a face legăturile dintre tabelele *clienți* și *detalii_clienți.*

- **produse:** datele referitoare la produse se află in tabela *produse,* informațiile asociate sunt id-ul produsului, denumirea produsului, stocul produsului respectiv, prețul de vânzare, unitatea de măsură asociată produsului, tipul produsului și furnizorul.

Constrângerile folosite sunt:

- de tip primary key pe *id_produs*
- de tip check pentru *stoc* și *preț* pentru a nu avem valori negative.
- de tip check pe *denumire_produs* pentru a verifica că denumirea are mai mult de două caractere.
- de tip check pentru ca verifica că unitatea de măsură *um* se află in lista de valori impusă.
- de tip foreign key pentru *id_categorie, id_furnizor* și *id_tip* pentru a face legăturile cu tabele *categorii_animale, furnizori* și *tipuri_produse.*

- **furnizori:** datele referitoare la furnizorii magazinului se află in tabela *furnizori*, informațiile asociate sunt id-ul furnizorului și denumirea furnizorului.

Constrângerile folosite sunt:

- de tip primary key pe *id_furnizor.*
- de tip unique pe *denumire_furnizor,* pentru a nu avea mai mulți furnizori cu aceeași denumire.
- de tip check pe *denumire_furnizor* pentru a verifica că lungimea denumirii furnizorului este mai mare de 3 litere și că nu conține cifre.

- **tipuri de produse:** datele referitoare la tipurile de produse disponibile in magazin se afla în tabela *tipuri_produse,* informațiile asociate sunt id-ul tipului și denumirea tipului de produs.

Constrângerile folosite sunt:

- de tip primary key pe *id_tip.*
- de tip unique pe *denumire_tip,* pentru a nu avea mai multe tipuri cu aceeași denumire.
- de tip check pe *denumire_tip* pentru a verifica că lungimea denumirii tipului este mai mare de 3 litere și că nu conține cifre.

- **categorii de animale:** datele referitoare la categoriile de animale pentru care sunt disponibile produsele din magazin se află in tabela *categorii_animale,* informațiile asociate tabelei *categorii_animale* sunt id-ul categoriei și denumirea categoriei de animale.

Constrângerile folosite sunt:

- de tip primary key pe *id_categorie.*
- de tip unique pe *denumire_categorie,* pentru a nu avea mai multe categorii cu aceeași denumire.
- de tip check pe *denumire_categorie* pentru a verifica că lungimea categoriei este mai mare de 3 litere și că nu conține cifre.

- **vânzări:** datele referitoare la produsele vândute se află in tabela *vânzări,* informațiile asociate sunt numărul bonului, numărul cardului de fidelitate, id-ul produsului, cantitate de produs cumpărată și data achiziției.

Constrângerile folosite sunt:

- de tip primary key pe *nr_bon.*
- de tip foreign key pentru *nr_card* și *id_produs* pentru a face legăturile cu tabele *clienți* și *produse.*
- constrângere implementată cu un trigger pentru verificarea datei calendaristice *data_achizitiei*, pentru care se emite bonul. Bonul nu se poate emite inainte de data înființării magazinului.

Toate primary key-urile sunt generate de baza de date printr-un mecanism de tip autoincrement. Majoritatea pornesc de la valoarea 1, cu excepția primary key-ului pentru *id_produs* care pornește de la 1000.

TEHNOLOGII FOLOSITE

Limbajul Python

Am folosit python versiunea 3.9.

Limbajele HTML și CSS pentru partea de front-end a aplicației

Framework-ul Flask

Flask este un API (Application Programming Interface) Python ce permite construirea de aplicații web.

Flask-MySQL

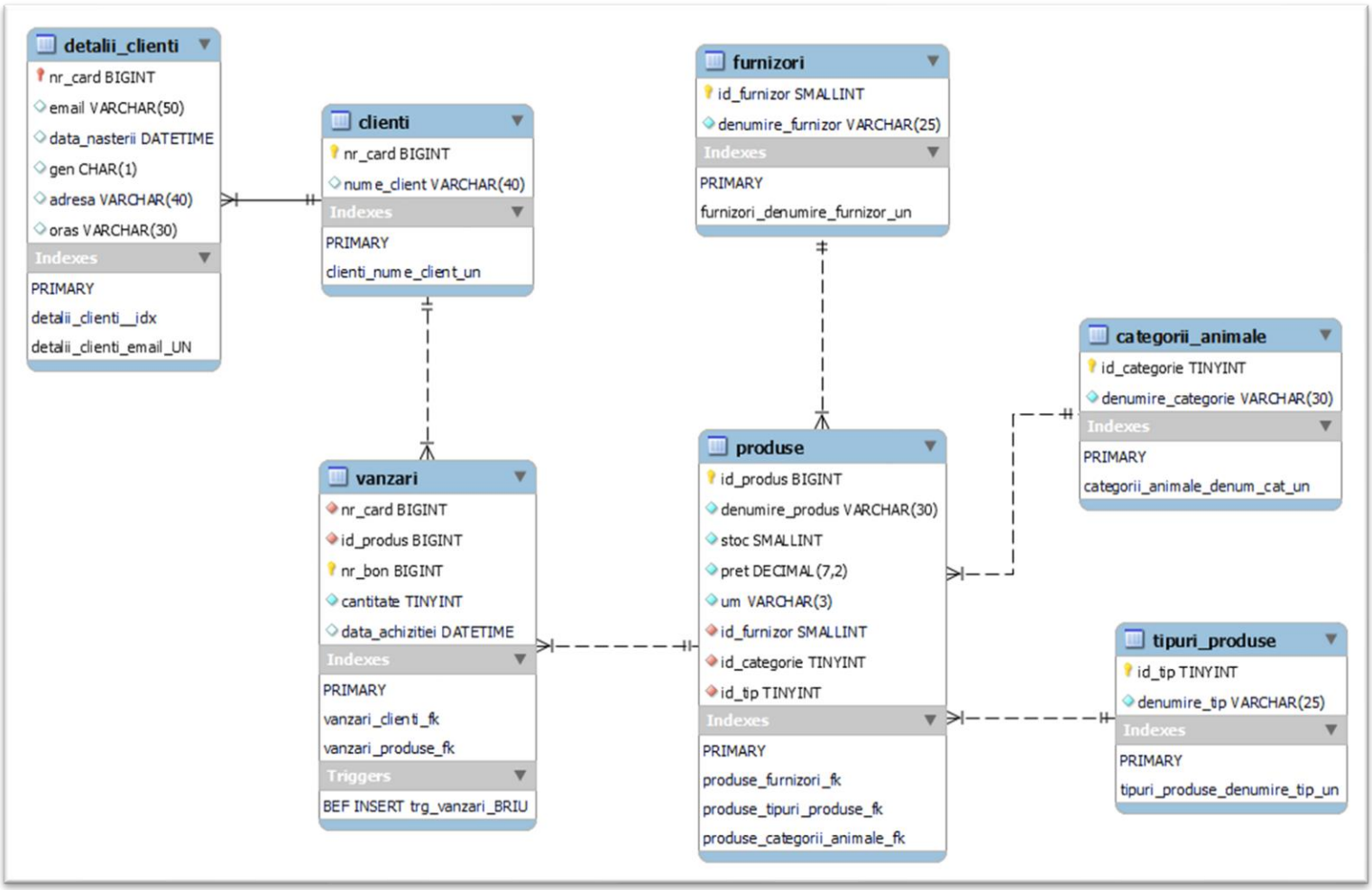
Este o extensie a Flask-ului care permite accesul la o bază de date MySQL.
Aceasta este secvența de cod prin care s-a făcut conectarea aplicației la baza de date.

```
app = Flask(__name__)
mysql = MySQL()
app.config['MYSQL_DATABASE_USER'] = 'root'
app.config['MYSQL_DATABASE_PASSWORD'] = 'TemaBD0.'
app.config['MYSQL_DATABASE_DB'] = 'test'
app.config['MYSQL_DATABASE_HOST'] = 'localhost'
mysql.init_app(app)
conn = mysql.connect()
```

MySQL Workbench

Am folosit aplicația *MySQL Workbench* pentru a crea o bază de date și pentru a rula script-ul de creare a tabelelor cu constrângerile necesare și cel de inserat valori în tabele.

DIAGRAMA E-R


detalii_clienti
nr_card BIGINT
email VARCHAR(50)
data_nasterii DATETIME
gen CHAR(1)
adresa VARCHAR(40)
oras VARCHAR(30)
Indexes
PRIMARY
detalii_clienti__idx
detalii_clienti_email_UN
clienti
nr_card BIGINT
nume_client VARCHAR(40)
Indexes
PRIMARY
clienti_nume_client_un
furnizori
id_furnizor SMALLINT
denumire_furnizor VARCHAR(25)
Indexes
PRIMARY
furnizori_denumire_furnizor_un
categorii_animale
id_categorie TINYINT
denumire_categorie VARCHAR(30)
Indexes
PRIMARY
categorii_animale_denum_cat_un
vanzari
nr_card BIGINT
id_produs BIGINT
nr_bon BIGINT
cantitate TINYINT
data_achizitiei DATETIME
Indexes
PRIMARY
vanzari_clienti_fk
vanzari_produse_fk
Triggers
BEF INSERT trg_vanzari_BRIU
produse
id_produs BIGINT
denumire_produs VARCHAR(30)
stoc SMALLINT
pret DECIMAL(7,2)
um VARCHAR(3)
id_furnizor SMALLINT
id_categorie TINYINT
id_tip TINYINT
Indexes
PRIMARY
produse_furnizori_fk
produse_tipuri_produse_fk
produse_categorii_animale_fk
tipuri_produse
id_tip TINYINT
denumire_tip VARCHAR(25)
Indexes
PRIMARY
tipuri_produse_denumire_tip_un

Descrierea detaliată a tabelelor și a relațiilor dintre ele

În proiectarea acestei baze de date s-au identificat tipurile de **relații** 1:1, 1: n și n : n.

Între tabelele **clienți** și **detalii_clienți** se întânește o relație de tip **one-to-one** deoarece unui client din tabela *clienți* îi corespunde un singur set de detalii din tabela *detalii_clienți.* Legătura dintre cele două tabele este realizată prin câmpul *nr_card.*

Între tabelele **furnizori** și **produse** se întâlnește o relație de tip **one-to-many** deoarece un furnizor poate furniza mai multe produse, iar un produs este furnizat de către un singur furnizor. Legătura dintre cele două tabele este realizată prin câmpul *id_furnizor.*

Între tabelele **tipuri_produse** și **produse** se întâlnește o relație de tip **one-to-many** deoarece un produs poate fi asociat unui singur tip, iar un tip de produs are mai multe produse asociate. Legătura dintre cele două tabele este realizată prin câmpul *id_tip.*

Între tabelele **categorii_animale** și **produse** se întâlnește o relație de tip **one-to-many** deoarece un produs poate fi destinat unei singure categorii de animale, iar o categorie de animale are mai multe produse asociate. Legătura dintre cele două tabele este realizată prin câmpurile *id_categorie.*

Între tabelele **clienți** și **produse** se întâlnește o relație de tip **many-to-many** deoarece un client poate cumpăra mai multe produse, iar un produs poate fi cumpărat de mai mulți clienți. Legătura dintre cele două tabele este realizată prin câmpurile *nr_card* și *id_produs.*

Între tabelele *clienți* și *produse*, mai apare tabela *vânzări,* prin urmare relația de **many-to-many** se transformă în relație de **one-to-many** între tabela *clienți* și tabela *vânzări* și în relație de **many-to-one** între tabela *vânzări* și tabela *produse.*

Pentru simplificarea proiectului s-au luat in considerare următoarele aspecte

- Un furnizor aprovizioneaă maxim o dată pe zi;
- Nu este obligatoriu ca vanzarea să se poată face doar pe baza de card de fidelitate ;
- Nu se poate vinde mai mult decât este in stoc;
- Nu se gestionează termenul de garanție al produselor;
- Vânzarea produselor se face doar in cantități intregi;
- Nu sunt promoții;
- Prețul unui produs nu variază în timp;

FUNCȚIONALITATEA APLICAȚIEI

Prin această aplicație am incercat să implementez vizual funcționalitatea principalelor funcții de interogare a unei baze de date. Astfel putem modifica/accesa o bază de date cu ușurință datorită uneltelor pe care python-ul le pune la dispoziție.

1. Funcția Select

Sintaxa este :

SELECT * FROM tabela;

Este comanda cea mai utilizata. Este folosita pentru obtinerea datelor din bazele de date.

Exemplu de cod utilizat în aplicație:

```
###---------------------------------------- VIZUALIZARE CLIENTI ----------------------------------------------------###
@app.route('/clienti')
def clienti():
    clienti_ = []
    cursor = conn.cursor()
    cursor.execute('SELECT clienti.nr_card , nume_client , email, data_nasterii, gen, oras, adresa '
                   'FROM clienti, detalii_clienti '
                   'WHERE clienti.nr_card = detalii_clienti.nr_card '
                   'ORDER BY clienti.nr_card')
    for result in cursor:
        client = {'nr_card': result[0], 'nume_client': result[1], 'email': result[2],
                  'gen': result[4], 'oras': result[5], 'adresa': result[6]}
        if str(result[3]) != 'None':
            client['data_nasterii'] = datetime.strptime(str(result[3]), '%Y-%m-%d %H:%M:%S').strftime('%d.%m.%Y')
        else:
            client['data_nasterii'] = result[3]
        clienti_.append(client)
    cursor.close()
    return render_template('Clienti/clienti.html', clienti=clienti_)
```

Pagina pentru select este următoarea:

Clienți

Adaugă clienți

Nr.card	*Nume client*	*Adresă de e-mail*	*Data nașterii*	*Gen*	*Oraș*	*Adresă*	**Acțiuni**	
1	FĂRĂ CARD	None	None	None	None	None	Editează client	Șterge client
2	Robert Popa	robert1605@yahoo.com	16.05.2000	M	Iași	None	Editează client	Șterge client
3	Ana Maria	anaa-maria@gmail.com	15.01.2001	F	Vaslui	str.Ștefan cel Mare, 23	Editează client	Șterge client
4	Bianca Popa	biia-popa@ymail.com	10.09.1999	F	Botoșani	str.Mihai Viteazul, 25	Editează client	Șterge client
5	Ionuț Marian	ionutz-marian@gmail.com	11.10.1989	M	București	str.Voiezorilor, 75	Editează client	Șterge client
6	Petru Ionică	petru.ionica89@gmail.com	21.11.1998	M	București	str.Apelor, 175	Editează client	Șterge client
7	Răzvan Mitică	razvan_mitica@gmail.com	25.11.2000	M	Iași	str.Florilor, 165	Editează client	Șterge client

2. Funcția Insert

Adaugă o nouă înregistrare în tabelă.

Sintaxa este :

INSERT INTO tabela [(coloana [, coloana . . .])]
VALUES (valoare [, valoare . . .]);

Exemplu de cod utilizat in aplicație:

```
###---------------------------------------- ADAUGARE CLIENTI ----------------------------------------###
@app.route('/add_client', methods=['GET', 'POST'])
def add_client():
    error = " "
    try:
        if request.method == 'POST':
            cursor = conn.cursor()
            values = []
            values.append("'" + request.form['nume_client'] + "'")
            fields = ['nume_client']
            query = 'INSERT INTO %s (%s) VALUES (%s)' % ('clienti', ', '.join(fields), ', '.join(values))
            cursor.execute(query)
            values = []
            cursor.execute('SELECT max(nr_card) FROM clienti')
            last_nr_card = cursor.fetchone()
            values.append(str(last_nr_card)[1:-2])
            values.append("'" + request.form['email'] + "'")
            if request.form['data_nasterii'] != "":
                values.append("STR_TO_DATE('" + request.form['data_nasterii'] + "', '%d.%m.%Y')")
            else:
                values.append("null")
            values.append("'" + request.form['gen'] + "'")
            values.append("'" + request.form['oras'] + "'")
            values.append("'" + request.form['adresa'] + "'")
            fields = ["nr_card", 'email', 'data_nasterii', 'gen', 'oras', 'adresa']
            query = 'INSERT INTO %s (%s) VALUES (%s)' % ('detalii_clienti', ', '.join(fields), ', '.join(values))
            cursor.execute(query)
            cursor.execute('commit')
            cursor.close()
            return redirect('/clienti')
    except Exception as err:
        print("Something went wrong: {}".format(err), file=sys.stderr)
        error = err
    finally:
        return render_template('Clienti/add_client.html', error=error)
```

Pagina pentru insert este următoarea:

3. Funcția Update

Modifică datele existente cu alte date valide.

Exemplu de cod utilizat in aplicație:

```
###---------------------------------------- EDITEAZA CLIENTI ----------------------------------------------------###
@app.route('/edit_client', methods=['GET', 'POST'])
def edit_client():
    try:
        if request.method == 'POST':
            ## update into clienti
            cursor = conn.cursor()
            values = []
            values.append("'" + request.form['nume_client'] + "'")
            fields = ['nume_client']
            query = 'UPDATE clienti SET %s=%s WHERE nr_card = %s' % (fields[0], values[0], _nr_card)
            cursor.execute(query)
            ## update into detalii_clienti
            values = []
            values.append("'" + request.form['email'] + "'")
            if request.form['data_nasterii'] not in ["", "None"] :
                values.append("STR_TO_DATE('" + request.form['data_nasterii'] + "', '%d.%m.%Y')")
            else:
                values.append("null")
            values.append("'" + request.form['gen'] + "'")
            values.append("'" + request.form['oras'] + "'")
            values.append("'" + request.form['adresa'] + "'")
            fields = ['email', 'data_nasterii', 'gen', 'oras', 'adresa']
            query = "UPDATE detalii_clienti SET %s=%s, %s=%s, %s=%s, %s=%s, %s=%s WHERE nr_card = %s" %(
                fields[0], values[0], fields[1], values[1],fields[2], values[2], fields[3],values[3], fields[4],
                values[4], str(_nr_card))
            cursor.execute(query)
            cursor.execute('commit')
            cursor.close()
        return redirect('/clienti')
    except Exception as err:
        print("Something went wrong: {}".format(err), file=sys.stderr)
        return render_template('Clienti/clienti.html', error=err)
```

La efectuarea unui update, datele existente sunt citite din tabelă și completate automat în câmpurile necesare pentru a putea modifica doar una sau mai multe câmpuri fără a fi nevoie de a se completa câmpurile nemodificate.

Pagina update arată astfel:

Funcția de completare automată a câmpurilor este următoarea:

```
###------------------------------------ PRELUARE INFORMATII CLIENTI ------------------------------------------###
@app.route('/get_client', methods=['POST'])
def get_client():
    nr_card = request.form['nr_card']
    global _nr_card
    _nr_card = nr_card
    cursor = conn.cursor()
    cursor.execute('SELECT nume_client, email, data_nasterii, gen, oras, adresa '
                   'FROM clienti, detalii_clienti '
                   'WHERE clienti.nr_card = detalii_clienti.nr_card AND clienti.nr_card = %s '
                   'ORDER BY clienti.nr_card' %(nr_card) )
    result = cursor.fetchone()
    client = {'nume_client': result[0], 'email': result[1],
              'gen': result[3], 'oras': result[4], 'adresa': result[5]}
    if str(result[2]) != 'None':
        client['data_nasterii'] = datetime.strptime(str(result[2]), '%Y-%m-%d %H:%M:%S').strftime('%d.%m.%Y')
    else:
        client['data_nasterii'] = result[2]
    cursor.close()
    return render_template('Clienti/edit_client.html', client=client)
```

4. Funcția Delete

Șterge o înregistrare din tabelă.

Exemplu de cod utilizat în aplicație:

```
###---------------------------------------- STERGERE CLIENTI ----------------------------------------------------###
@app.route('/delete_client', methods=['POST'])
def delete_client():
    try:
        client = request.form['nr_card']
        cursor  = conn.cursor()
        cursor.execute('DELETE FROM detalii_clienti WHERE nr_card = ' + client)
        cursor.execute('commit')
        cursor.execute('DELETE FROM clienti WHERE nr_card = ' + client)
        cursor.execute('commit')
        cursor.close()
        return redirect('/clienti')
    except Exception as err:
        print("Something went wrong: {}".format(err), file=sys.stderr)
        return render_template('Clienti/clienti.html', error=err)
```