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

BAZE DE DATE 2023-2024

Gestiunea unui magazin de articole școlare

Coordonator, asoc. ing. Avram Sorin
Student, Maftei-Guțui Robert Mihăiță
Grupa 1308A

Titlu proiect: Gestiunea activitatii unui magazin de articole școlare

Analiza, proiectarea si implementarea unei baze de date si a aplicatiei aferente care sa modeleze activitatea unui magazin de articole școlare cu privire la gestiunea comenzilor in functie de client.

Descrierea cerintelor si modul de organizare al proiectului

Volumul mare de solicitari pentru produsele ce pot fi furnizate de un magazin necesita digitalizarea activitatii, incepand cu preluarea comenzilor pana la livrarea acestora. Acest lucru presupune inregistrarea clientilor, a comenzilor si urmarirea acestora pana la momentul finalizarii. Informatiile de care avem nevoie sunt legate de :

- **Clienti**: avem nevoie sa stim nume si prenumele clientului, adresa (in cazul unei livrari la domiciliu), adresa de e-mail (pentru a putea trimite noutati cu privire la produsele si serviciile magazinului), data nasterii (pentru a trimite un e-mail de felicitare si eventuale discounturi cu ocazia zilei de nastere). S-au folosit doua tabele: **Client** si **Detalii client**. In vederea unei cautari rapide a clientului avem tabela Client cu numele clientului;

- **Comanda** : contine data efectuarii comenzii si pretul acesteia;

- **Angajati** : vrem sa stim cine se ocupa de comanda la un moment dat si valoarea comenzii, numarul de angajati ai magazinului si rolurile fiecarui angajat, totalul de vanzari per angajat;

- **Produse**: avem nevoie sa stim ce produse avem pe stoc la momentul efectuarii comenzii ;

- **Furnizor**: in cazul in care un produs se epuizeaza sau se primesc reclamatii sunt necesare datele despre furnizor. Se mentine evidenta a ce produse aduce fiecare furnizor si numarul de produse furnizate de acesta.

Tehnologii folosite:

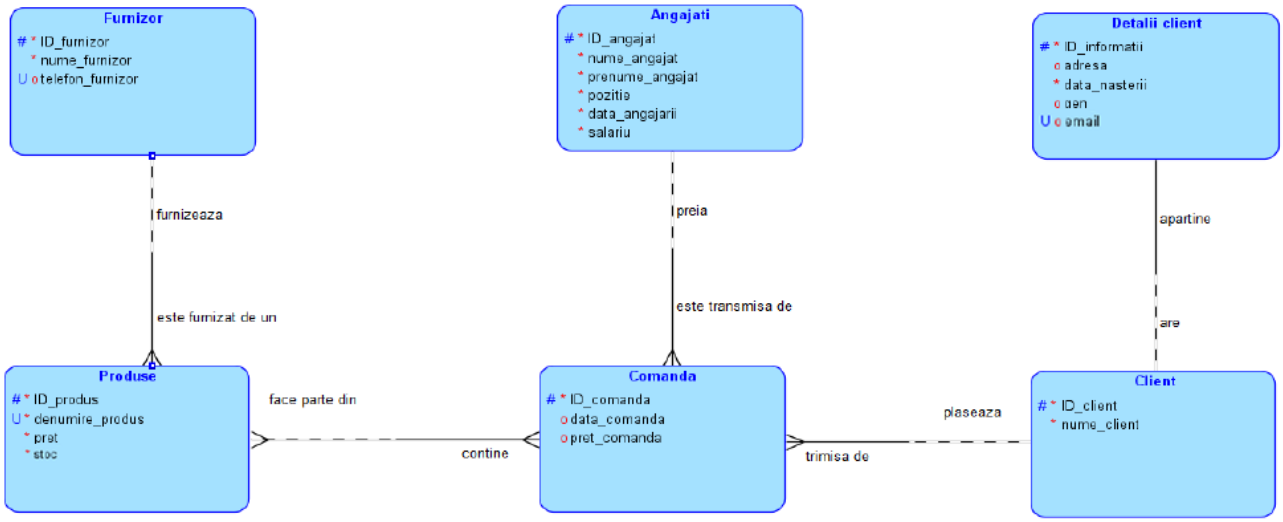
Pentru relizarea proiectului, pentru partea de front-end a aplicatiei s-a folosit HTML pentru realizarea paginilor si CSS pentru font. Pentru realizarea de back-end, SQL pentru realizarea bazei de date cu ajutorul aplicatiilor *DataModeler* si *SQLDeveloper* si Python prin intyermediul „*Flask*" si a modulului *„cx_Oracle"* pentru a realiza legatura dintre front-end si baza de date.

Descrierea functionala a aplicatiei:
Principalele functii care se pot intalni intr-un magazin sunt:
- Evidenta clientilor
- Evidenta comenzilor
- Evidenta personalului

Descrierea detaliata a entitatilor si a relatiilor dintre tabele
Tabelele din aceasta aplicatie sunt:
- Angajati
- Client
- Detalii client
- Produse
- Furnizor
- Comanda

In proiectarea acestei baze de date s-au identificat tipurile de relatii 1:1, 1:n si n:n.

Schema logica:



Intre tabelele **Detalii client** si **Client** exista o relatie de tip 1:1, deoarece un client are un singur cont unic (deci o singura instanta a „detaliilor acestuia”), iar un cont („detaliile clientului”) apartine unui singur client. Legatura se realizeaza prin ID_client.

Intre tabelele **Furnizor** si **Produse** exista o relatie de tip 1:n, deoarece un furnizor poate procura mai multe produse, dar, in general, un produs are un singur furnizor, legatura realizandu-se prin ID_furnizor.
Intre tabelele **Angajati** si **Comanda** exista o relatie de tip 1:n, deoarece un angajat poate prelua mai comenzi, insa o comanda poate fi transmisa unui singur angajat. Legatura se realizeaza prin coloana ID_angajat.
Intre tabelele **Client** si **Comanda** exista o relatie de tip 1:n, deoarece un client poate plasa mai multe comenzi, insa o comanda poate apartine unui singur client. Legatura se realizeaza prin coloana ID_client.
Intre tabelele **Produse** si **Comanda** exista o relatie de tip n:n, deoarece un produs poate face parte din mai multe comenzi, iar o comanda poate contine mai multe produse. Legatura se realizeaza prin tabela de legatura **Relation_prod_com** in care se retine id_produs si id_comanda, cat si cantitatea de produs ceruta.

Constrangeri folosite:
Pentru realizarea acestei baze de date, au fost utilizate constrangeri de tip:
- Primary key
- Foreign key
- Check
- Unique
- Not null

Exemple de utilizare constrangeri:

- **Primary key**: fiecare tabela contine un camp (id_*"nume tabel"*) de tip cheie primara pentru a numerota fiecare intrare unica in tabela. In tabelul **Relation_prod_com** avem chei primare Produse_ID_produs si Comanda_ID_comanda

- **Foreign key**: Este folosit pentru a ajuta la interogarile dintre tabele. Spre exemplu, in tabela:
  - **Detalii client** datorita relatiei 1:1 exista coloana client_ID_client de tip *foreign key.*
  - **Comanda** relatiile 1:n dintre angajati->comanda si client->comanda, avem 2 chei straine(angajati_id_angajat, client_id_client)
  - **Produse** 1:n furnizor->produse (furnizor_id_furnizor)
  - **Relation_prod_com** 1:n produse->relation_prod_com si comanda-> relation_prod_com (produse_id_produs, comanda_id_comanda)

- **Check**: este folosita in cele mai multe cazuri, ca si regex pentru verificarea corectitudinii stringurilor introduse, ca verificare a unor cantitati numerice sau dimenisuni de date(length), sau ca verificare a unui anumit cuvant dintr-o lista data.

Tabele:

Detalii clienti,

1. coloana **email**(regexp_like(email, '^[A-Za-z]+[A-Za-z0-9.]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$'));
2. adresa(length(adresa)>2 and regexp_like(adresa,'^[A-Za-z]*+ +[A-Za-z]*'))
3. data_nasterii(data_nasterii > to_date('01-01-1930', 'dd-mm-yyyy'))
4. gen(in(‚F’, ‚M’)).

Client: nume_client (length(nume_client)>2 and regexp_like(nume_client,'^[A-Za-z]*$')).

Angajati,

1. **pozitie** (CHECK (pozitie IN ('bucatar', 'manager', 'ospatar', 'spalator de vase'));
2. **data_nasterii**(data_nasterii > to_date('01-01-1930', 'dd-mm-yyyy'));
3. **nume/prenume_angajat** (length(prenume_angajat) > AND REGEXP_LIKE (prenume_angajat, '^[A-Za-z]*$'));
4. **salariu** (BETWEEN 1400 AND 99999)).

Produse,

1. **pret**(CHECK (pret BETWEEN 2 AND 999));
2. **denumire_produs** (length(...) and regexp_like(...));
3. **stoc** (>= 0)

Furnizor,

1. **Nume_furnizor** ((length(...) and regexp_like(...));
2. **Telefon_furnizor** (REGEXP_LIKE (telefon_furnizor, '^[0-9]') AND length(telefon_furnizor) = 10).

Relation_prod_com, Cantitate(BETWEEN).

Comanda,

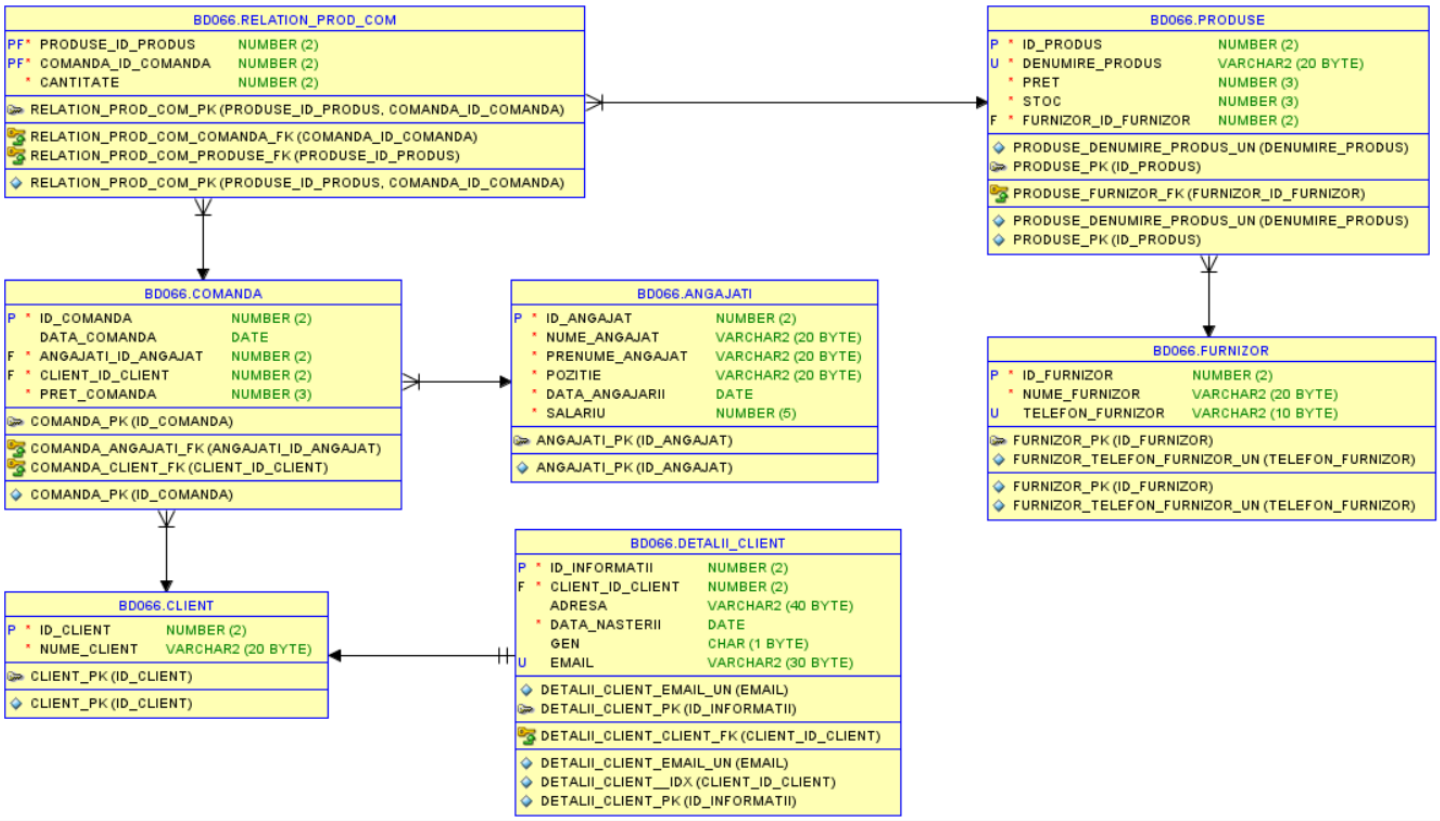
1. Data_comanda
2. Pret_comanda(>=0)

- **Unique**: conditia ca intr-o coloana fiecare intrare sa fie unica.
Exemple:

- **Furnizor**, coloana **telefon_furnizor**: fiecare numar de telefon este unic, deci fiecare furnizor va avea un numar unic de telefon;
- **Detalii clienti**, coloana **email**: emailurile sunt unice, deci fiecare cilent va avea un email unic.
- **Produse**, coloana denumire_produs: numele produselor nu se poate repeta.

- **Not null**: conditia ca un camp dintr-o coloana sa nu fie null.
Coloane care **POT FI NULE**: telefon_furnizor(Furnizor), data_comanda si pret_comanda(Comanda), adresa, gen, email(Detalii_client).

Schema relationala:



In schema logica se observa legaturile dintre tabele prin intermediul cheilor straine(FK), cheile primare din tabele(PK), Constrangerile de tip UNIQUE(UN).

Pentru a stabili relatia de tip n:n intre Produse si Comanda se formeaza o tabela noua denumita **Relation_prod_com** care are rolul de a facilita legatura intre cele doua tabele.

Cu ajutorul schemei relationale, s-a dezvoltat modelul fizic, prin intermediul caruia s-au implementat *Triggers*-urile pentru a:

- Verifica ca data introdusa sa nu depaseasca data curenta (**Trg_angajati_data**);
- Pentru a actualiza stocul dupa fiecare adaugare/actualizare intr-o comanda si pentru a verifica ca o comanda sa nu depaseasca stocul unui produs(**Trg_update_stoc_ins/up**);
- Pentru a actualiza pretul unei comenzi dupa fiecare adaugare/actualizare a unui produs pe bon, dupa verificarea stocului(**Trg_get_price_ins/up**).

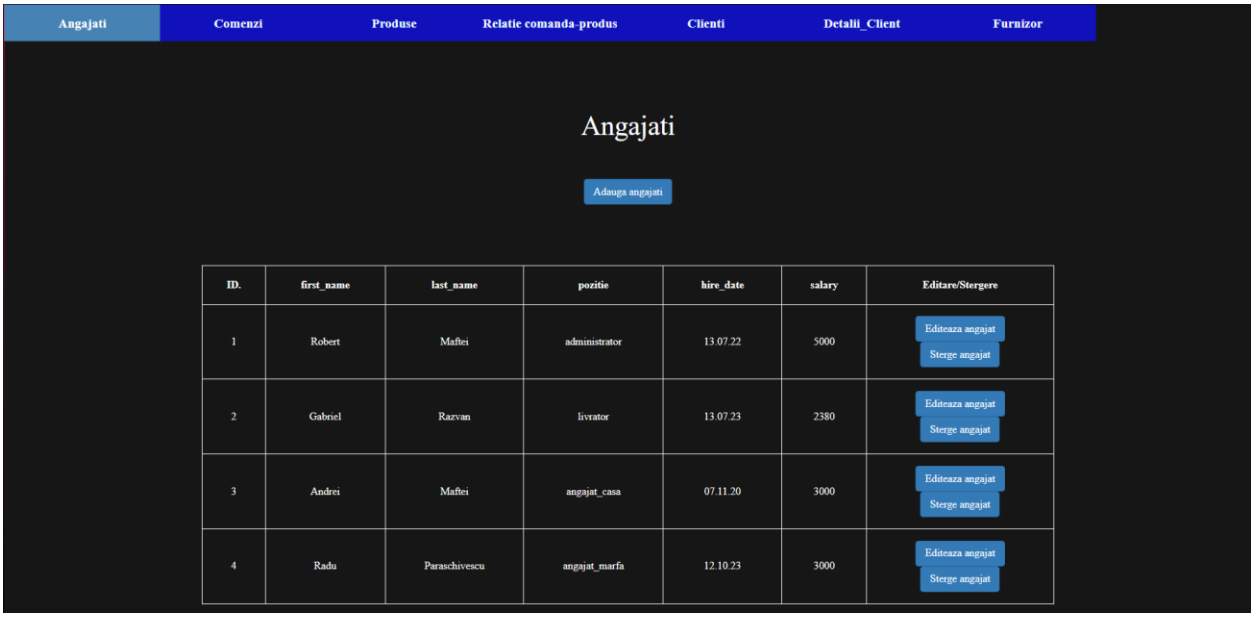
De asemenea, pentru anumite chei primare(PK), s-a implementat autoincrementul, fara generare de trigger si prin folosirea modelului fizic *Oracle Database 12c*:
- Tabela **Comanda**, coloana **ID_comanda;**
- Tabela **Client**, coloana **ID_client**.

Conectarea bazei de date la aplicatie.

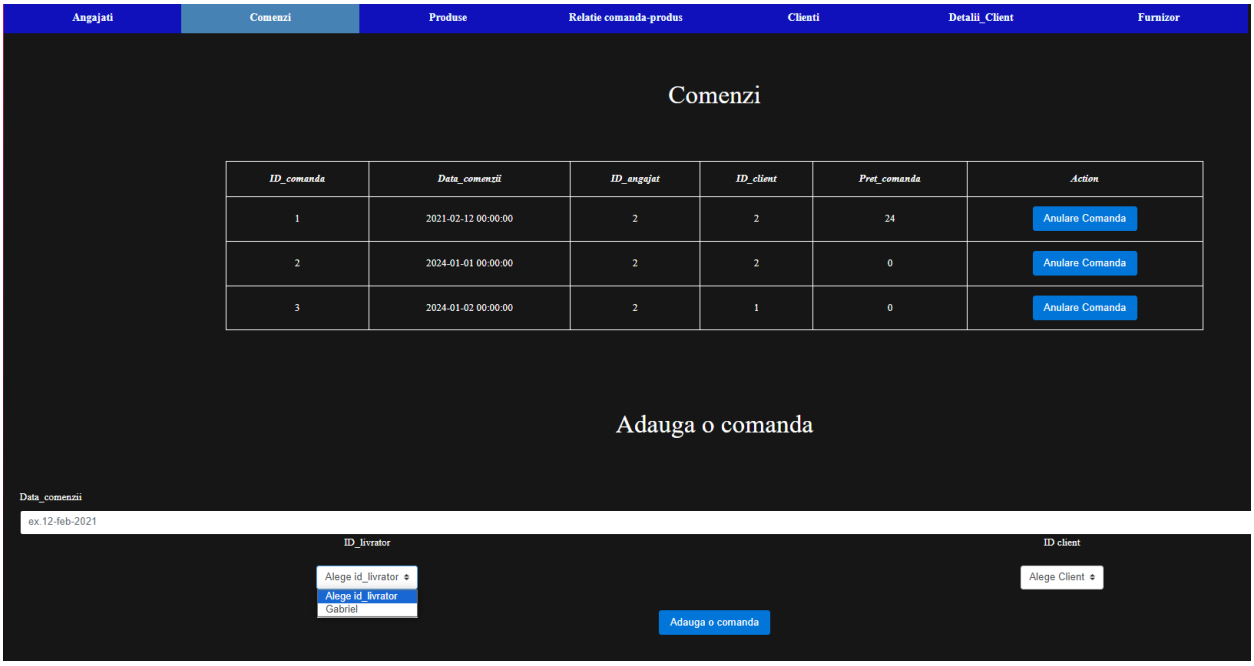
Conectarea la baza de date se realizeaza cu ajutorul modulului cx_Oracle din Python. Conexiunea se realizeaza prin comanda *cx_Oracle.connect(...)* ce are ca si parametru user-ul, parola de la baza de date, cat si dsn-ul necesar conectarii la baza de date respectiva.

Interfata:

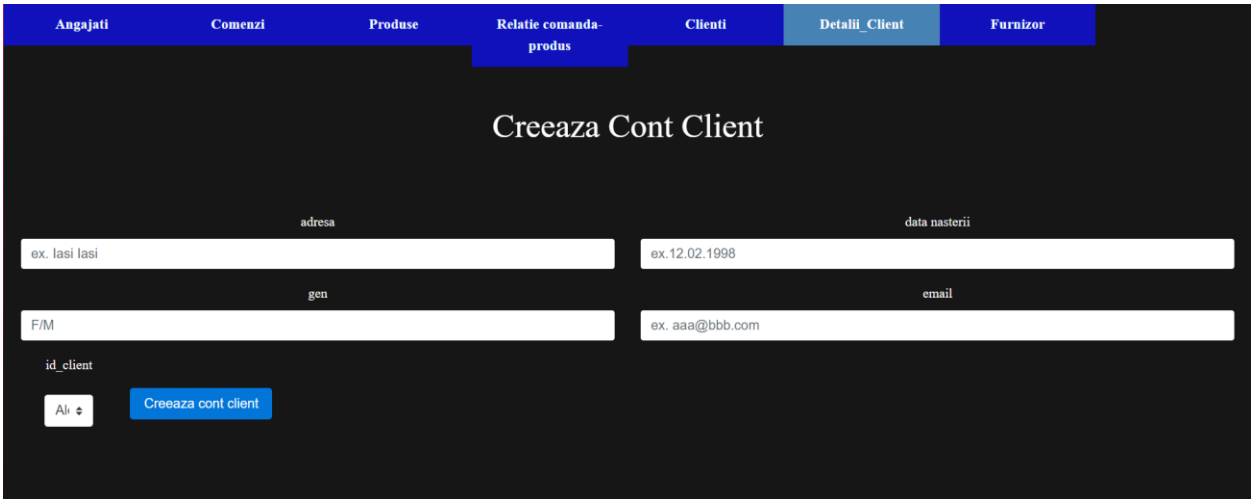
Interfata tabela angajati



Interfata tabela comenzi



Interfata creare cont client



Instructiuni SQL:

- **SELECT**: selectarea datelor necesare din baza de date pentru diverse operatii.

```python
@app.route('/')
@app.route('/angajati')
def angaj():
    employees = []

    cur = con.cursor()
    cur.execute('select * from angajati')
    for result in cur:
        print(result)
        employee = {}
        employee['employee_id'] = result[0]
        employee['first_name'] = result[1]
        employee['last_name'] = result[2]
        employee['pozitie'] = result[3]
        employee['hire_date'] = datetime.strptime(str(result[4]), '%Y-%m-%d %H:%M:%S').strftime('%d.%m.%y')
        employee['salary'] = result[5]

        employees.append(employee)
    cur.close()
    return render_template('employees.html', employees=employees)
```

- **INSERT**: pentru inserarea unor noi inregistrari in tabele

```python
@app.route('/addEmployee', methods=['GET', 'POST'])
def add_angaj():
    error = None
    if request.method == 'POST':
        emp = 0
        cur = con.cursor()
        cur.execute('select max(ID_angajat) from Angajati')
        for result in cur:
            emp = result[0]
        cur.close()
        emp = emp + 1 if emp is not None else 1
        cur = con.cursor()
        values = []
        values.append("'" + str(emp) + "'")

        values.append("'" + request.form['first_name'] + "'")
        values.append("'" + request.form['last_name'] + "'")
        values.append("'" + request.form['pozitie'] + "'")
        values.append("'" + datetime.strptime(str(request.form['hire_date']), '%d.%m.%Y').strftime('%d-%b-%y') + "'")
        values.append("'" + request.form['salary'] + "'")

        fields = ['ID_angajat', 'nume_angajat', 'prenume_angajat', 'pozitie', 'data_angajarii', 'salariu']
        query = 'INSERT INTO %s (%s) VALUES (%s)' % ('angajati', ', '.join(fields), ', '.join(values))

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

- **UPDATE**: pentru actualizarea unor inregistrari din tabela(pentru angajati si tabela de relatie dintre produse si comanda)

```
@app.route('/modRel', methods=['POST'])
def mod_rel():
    prod = "'" + request.form['id_produs'] + "'"
    comanda = "'" + request.form['id_com'] + "'"
    quant = "'" + request.form['cantitate'] + "'"
    cur = con.cursor()
    query = "UPDATE relation_prod_com SET cantitate=%s where comanda_id_comanda=%s and produse_id_produs=%s" % (
        quant, comanda, prod)
    cur.execute(query)
    cur.execute('commit')
    return redirect('/relatie')
```

-DELETE: pentru stergera unor inregistrari din tabele

```
80  @app.route('/delEmployee', methods=['POST'])
81  def del_emp():
82      emp = request.form['employee_id']
83      cur = con.cursor()
84      cur.execute('delete from angajati where ID_angajat=' + emp)
85      cur.execute('commit')
86      return redirect('/angajati')
87
88
```

- **COMMIT**: salveaza modificarile facute asupra bazei de date

```
64
65          cur.execute(query)
66          cur.execute('commit')
67          return redirect('/angajati')
```