



UNIVERSITATEA TEHNICĂ “GH ASACHI” IAȘI
FACULTATEA AUTOMATICĂ ȘI CALCULATOARE
SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA
INFORMAȚIEI

DISCIPLINA: BAZE DE DATE

Gestiunea angajaților unei firme de curierat

**Coordonator,
Prof. Mironeanu Cătălin**

**Student,
Pintilie Răzvan-Nicolae
Grupa 1306B**

Iași, 2022

Titlul proiectului

Gestiunea angajaților unei firme de curierat

Proiectul își propune să ofere o modalitate de gestionare a angajaților unei firme de curierat și, mai mult, o modalitate de a ține evidența fiecărei adrese din oraș în parte pentru o bună organizare internă.

Descrierea proiectului

Numărul mare de angajați al unei astfel de firme și multitudinea zonelor orașului ce trebuie acoperite de aceștia determină necesitatea unei bune organizări și a unei bune cunoașteri (din partea superiorilor) a îndatoririlor fiecărui subordonat; numărul de străzi al orașului fiind foarte mare, e nevoie de o grupare a acestora pe zone, fiecărui curier revenindu-i, de obicei, o singură zonă. Informațiile de care avem nevoie corespund următorilor factori:

- **Managerii depozitului:** aceștia se ocupă de gestionarea coletelor la nivel de depozit, în vederea distribuirii lor fiecărui curier; dacă un manager este responsabil de o anumită zonă, atunci acesta îi este superior angajatului ce livrează pe acea zonă;
- **Curierii (angajații):** un angajat poate avea mai multe **roluri** (șofer, ajutor șofer, manipulator lift); angajaților le sunt atribuite zone;
- **Adresele și gruparea lor în zone:** mai multe adrese formează o „zonă mică”, iar mai multe „zone mici” formează o „zonă cargo” (aici revin coletele mari/grele); unele adrese pot avea **observații** legate de starea traficului, starea carosabilului, etc;

Descrierea tehnologiilor folosite – Front-end

Aplicația este una de tip N-Tier întrucât avem un client care se conectează la un server și care are în spate o baza de date. În realizarea părții de Front-end a proiectului s-a folosit ca limbaj de programare *Python 3.6*, mai exact biblioteca **TKInter** din acest limbaj, întrucât acesta are o gamă largă și ușor de utilizat de obiecte vizuale.

Astfel, interfața creată este una ușor de folosit, datele din fiecare tabelă fiind la puține click-uri distanță, iar operațiile în cadrul acestora fiind intuitive.

Descrierea tehnologiilor folosite – Back-end

Pentru partea de back-end a aplicației, a fost utilizat *Python 3.6*, mai exact biblioteca **CX_Oracle** din cadrul acestuia. Aceasta ne pune la dispoziție un mod relativ ușor de a utiliza comenzi din limbajul python împreună cu cele din SQL. Astfel, prin intermediul ei, am creat o conexiune la propria baza de date, apoi, bazându-ne pe obiectul creat în urm realizării cu succes a conexiunii, am reușit să trimitem diverse comenzi specifice limbajului SQL și afișarea rezultatului acestora în aplicația noastră.

Ca mediu de dezvoltare pentru partea de back-end cât și pentru partea de front-end, am folosit IDE *PyCharm Community Edition 2022.2*.

Pentru crearea tabelelor și a bazei de date, am folosit *Oracle SQL Developer* și *Oracle Data Modeler*.

Structura și relațiile dintre entități

Entitățile din această aplicație sunt:

- Angajați
- Zone
- Adrese
- Manageri_depozit
- Tipuri_rol
- Observații

În proiectarea acestei baze de date s-au identificat următoarele tipuri de relații:
1:1 (one-to-one), 1:n (one-to-many).

Între entitatea **Zone** și entitatea **Angajați** se stabilește o relație de 1:n. O zonă poate să corespundă mai multor angajați, dar un angajat va livra pe o singură zonă.

Între entitatea **Zone** și entitatea **Adrese** se stabilește o relație de 1:n. O zonă cuprinde mai multe adrese, iar o adresă aparține unei singure zone.

Între entitatea **Manageri_depozit** și entitatea **Zone** se stabilește o relație de 1:n. Un manager este responsabil de mai multe zone, iar o zonă îi va reveni unui singur manager.

Între entitatea **Tipuri_rol** și entitatea **Angajați** se stabilește o relație de 1:n. Mai mulți angajați pot avea același rol, iar un angajat este responsabil doar de o singură sarcină generală.

Între entitatea **Adrese** și entitatea **Observații** se stabilește o relație de 1:1. O observație corespunde unei singure adrese, iar o adresă poate avea o singură observație (desigur, aceasta poate să cuprindă mai multe informații).

Descrierea constrângerilor

Constrângerile de tip **check** se găsesc în toate entitățile. Cu ajutorul lor verificăm dacă valorile introduse pentru nume de persoane sunt corecte sau dacă valorile numerice respectă intervalele impuse.

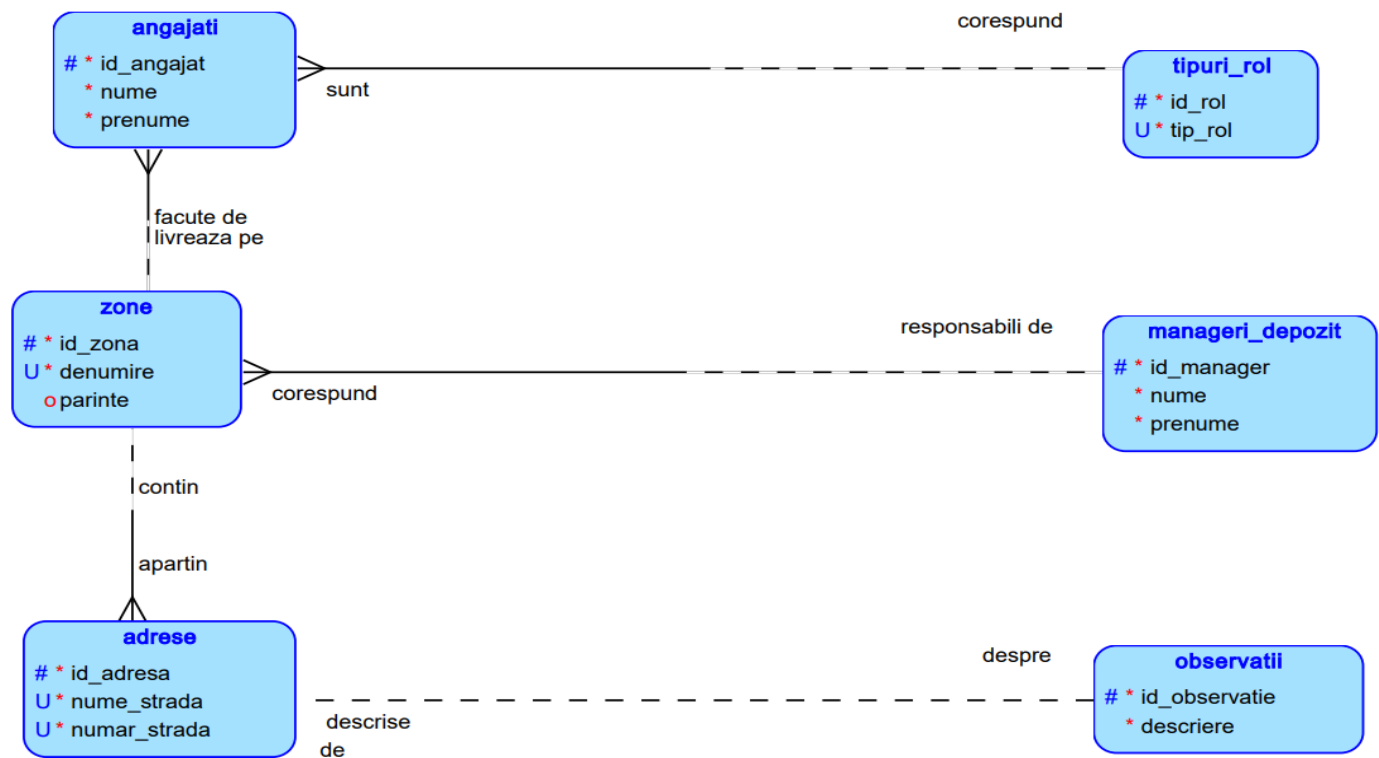
Constrângerile de tip **unique** sunt folosite pentru attributele *tip_rol*, *denumire* (în cadrul unei zone), dar și în gruparea de attribute *nume_strada* // *numar_strada* (pentru a nu avea două adrese identice).

Constrângerile de tip **not null** se găsesc pe marea majoritate a atributelor din entități.

Primary key-urile sunt generate de baza de date pe baza unui mecanism de **autoincrement**.

Modelul logic

Pe baza entităților mai sus descrise, a rezultat următorul model logic:



O descriere amănunțită a atributelor fiecărei entități se va face la prezentarea *modelului relational*, după ce vor fi fost convertite în coloane ale tabelelor.

Aspecte legate de normalizare

Baza de date a fost normalizată, deoarece îndeplinește următoarele condiții:

A) Tabelele respectă condițiile primei forme normale:

- un atribut conține valori atomice din domeniul său (și nu grupuri de astfel de valori)
- nu conține grupuri care se repetă

B) Tabelele respectă condițiile celei de-a doua forme normale:

- este în prima formă normală
- toate attributele non-cheie depind în totalitate de toate cheile candidat

C) Tabelele respectă a treia formă normală:

- este în a doua formă normală
- toate attributele non-cheie sunt direct (non-tranzitiv) dependente de toate cheile candidat.

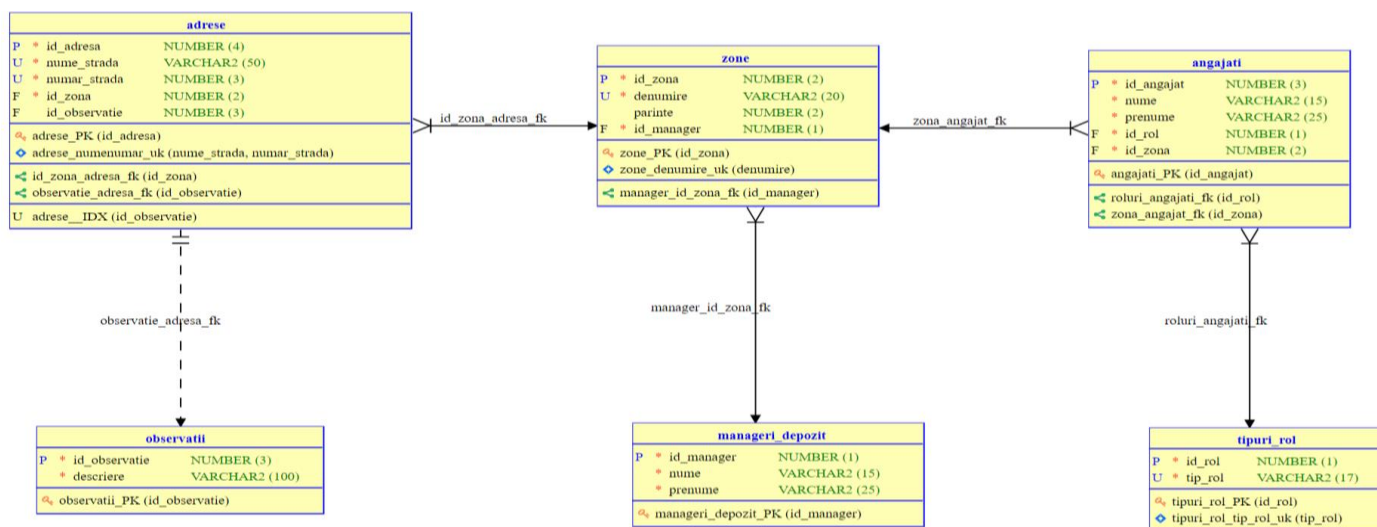
Prima formă normală este îndeplinită în cazul tuturor tabelelor; în fiecare tabelă, un câmp poate conține doar o singură valoare din domeniul acestuia. De exemplu, pentru tabela **Angajați**, dacă s-ar dori inserarea a încă o zonă unui angajat, acest lucru este imposibil și se va recurge la o înregistrare nouă în baza de date, cu două zone diferite.

A doua formă normală este îndeplinită de toate tabelele; nicio cheie candidat nu este de tipul cheie multiplă, deci fiecare tabelă are doar o cheie primară.

A treia formă normală este îndeplinită în cazul relației dintre tabela **Tipuri_rol** și tabela **Angajați**; atunci când specificăm un *id_rol*(FK) pentru un angajat, rolul acestuia va fi preluat cu ajutorul cheii primare din prima tabelă menționată.

Modelul relațional

Pe baza modelului logic, respectiv și a relațiilor dintre entitățile acestuia, dar și în urma procesului de normalizare, a rezultat următorul model relațional:



Descriere a tabelor finale și a coloanelor acestora

Tabelele rezultate sunt:

- Angajați
- Zone
- Adrese
- Manageri_depozit
- Tipuri_rol
- Observații

Tabela **Angajați** are următoarele coloane:

- *id_angajat*: primary-key;
- *nume*;
- *prenume*;
- *id_rol*: foreign-key; realizează legătura 1:n dintre tabela **Tipuri_rol** și tabela

Angajați;

- *id_zona*: foreign-key; specifică zona pe care va livra curierul și realizează legătura

1:n dintre tabela **Zone** și tabela **Angajați**; pot avea același *id_zona* doar angajați cu roluri diferite.

Tabela **Tipuri_rol** are următoarele coloane:

- *id_rol*: primary_key;
- *tip_rol*: roluri ce vor fi atribuite angajaților.

Tabela **Adrese** are următoarele coloane:

- *id_adresa*: primary-key;
- *nume_strada*;
- *numar_strada*;
- *id_zona*: foreign-key; specifică în ce zonă este localizată o adresă și realizează

legătura 1:n dintre tabela **Zone** și tabela **Adrese**;

- *id_observație*: foreign-key specifică ce observație are o anumită adresă (dacă are) și realizează legătura 1:1 dintre tabela **Adrese** și tabela **Observații**.

Tabela **Observații** are următoarele coloane:

- *id_observație*: primary-key;
- *descriere*: specifică informații despre starea traficului, starea carosabilului, etc.

Tabela **Zone** are următoarele coloane:

- *id_zona*: primary-key;
- *denumire*;
- *părinte*: dacă o zonă are părinte, este de tip „zonă mică”; dacă această coloană

rămâne *NULL*, zona este de tip „Cargo”;

- *id_manager*: foreign-key; managerul responsabil de coletele ce trebuie livrate la adresele dintr-o anumită zonă și, implicit, responsabil și de angajații ce livrează pe acea zonă; realizează legătura 1:n dintre tabela **Manager_depozit** și tabela **Zone**.

Tabela **Manageri_depozit** are următoarele coloane:

- *id_manager*: primary-key;
- *nume*;
- *prenume*.

Conectarea la baza de date

Așa cum aminteam anterior conexiunea la baza de date este făcută prin intermediul bibliotecii *cx_Oracle* din Python 3.6. Aceasta permite crearea unui obiect de tip conexiune pe care îl putem salva într-o variabilă. În momentul în care operația de conectare s-a realizat cu succes, vom avea la îndemână un obiect de tip conexiune pe care îl vom putea folosi pentru a crea un cursor prin intermediul căruia vom putea executa diverse comenzi specifice SQL.

Pentru fiecare tabelă, au fost create funcții pentru fiecare dintre operațiile de *select*, *insert*, *update* și *delete*.

Exemple funcții:

```
def selectAllAngajati():
    for item in angajati.get_children():
        angajati.delete(item)
    conn.commit()
    cur = conn.cursor()
    command = 'select a.id_angajat, a.nume, a.prenume, z.id_manager, a.id_rol, a.id_zona ' \
              'from ANGAJATI a, ZONE z ' \
              'where a.id_zona = z.id_zona order by id_angajat'
    cur.execute(command)
    for result in cur:
        angajati.insert(parent='', index='end', iid=0, text='',
                        values=(result[0], result[1], result[2], result[3], result[4], result[5]))
    cur.close()
```

```
def selectJoinAdresa(street, number, cargo):
    for item in angajati.get_children():
        angajati.delete(item)
    if street != '' and number != '':
        conn.commit()
        cur = conn.cursor()
        if cargo == '' or cargo == 'nu' or cargo == 'Nu' or cargo == 'NU':
            command = 'select a.id_angajat, a.nume, a.prenume, z.id_manager, a.id_rol, a.id_zona ' \
                      'from ANGAJATI a, ZONE z, ADRESE ad ' \
                      'where a.id_zona = z.id_zona and ' \
                      'ad.nume_strada = \'' + street + '\' and ' \
                      'ad.numar_strada = ' + number + ' and ' \
                      'ad.id_zona = z.id_zona '
        if cargo == 'da' or cargo == 'DA' or cargo == 'Da':
            command = 'select a.id_angajat, a.nume, a.prenume, z1.id_manager, a.id_rol, a.id_zona ' \
                      'from ANGAJATI a, ZONE z1, ADRESE ad, ZONE z2 ' \
                      'where ad.nume_strada = \'' + street + '\' and ' \
                      'ad.numar_strada = ' + number + ' and ' \
                      'ad.id_zona = z1.id_zona and ' \
                      'z1.parinte = z2.id_zona and ' \
                      'a.id_zona = z2.id_zona '
        cur.execute(command)
        for result in cur:
            angajati.insert(parent='', index='end', iid=0, text='',
                            values=(result[0], result[1], result[2], result[3], result[4], result[5]))
        cur.close()
```

```
def insertIntoZone(denumire, parinte, id_manager):
    if denumire != '' and id_manager != '':
        conn.commit()
        cur = conn.cursor()
        command = 'insert into Zone (denumire, parinte, id_manager) ' \
                  'VALUES (\'' + denumire + '\', \'' + parinte + '\', ' + id_manager + '))'
        cur.execute(command)
        cur.close()
```

```

def updateManagerZona(manager, denumire):
    if nume != '' and zona != '':
        conn.commit()
        cur = conn.cursor()
        command = 'update ZONE ' \
                   'set id_manager = ' + manager + ' ' \
                   'where denumire = \'' + denumire + '\''
        cur.execute(command)
        cur.close()

```

```

def delIdAdresa(idAddr):
    if idAddr != '':
        conn.commit()
        cur = conn.cursor()
        command = 'delete ADRESE where id_adresa = ' + idAddr + ' '
        cur.execute(command)
        cur.close()

```

Capturi de ecran cu interfața aplicației

DB Interface

Angajati Tipuri_Rol Manageri_Depozit **Zone** Adrese Observatii

Adauga Angajat Editeaza Angajat Sterge Angajat Vezi tot

id	Nume	Prenume	id_manager	id_rol	id_zona
13	Ciubotaru	Stefan	3	1	5
14	Pasa	David	3	2	5
15	Selicu	Ionut-Dumitru	3	3	5


Strada Nr. Cargo?

Aleea Romana 90 da Vezi

Manager (Nume) Manager (Prenume) Vezi

Zona (denumire) Vezi

Nume Vezi

 DB Interface

Editeaza Zona

Zona (denumire)	Parinte (denumire)	Update Parinte
<input type="text"/>	<input type="text"/>	
Zona (denumire)	Manager (id)	Update Manager
<input type="text"/>	<input type="text"/>	
Back		

 DB Interface

Sterge Observatie

Observatie (id)	Sterge
<input type="text"/>	
	Back