UNIVERSITATEA TEHNICĂ “GH ASACHI” IAŞI

FACULTATEA AUTOMATICĂ ŞI CALCULATOARE

CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI

DISCIPLINA BAZE DE DATE

Gestiunea activității unui service auto

Coordonator,
Avram Sorin

Studenți,
Cănuci Răzvan-Andrei
Simionescu Darius-Ioan

Grupa 1307B

Tema proiectului: Gestiunea activității unui service auto

Acest proiect constă in analiza, proiectarea si implementarea unei baze de date si a aplicației aferente care să modeleze activitatea unui service auto, dar și funcționarea internă a afacerii ce constă în angajații și lista serviciilor prestate.

Descrierea cerințelor și a funcțiilor aplicației

În cadrul aplicației se tratează problema unui singur service auto deschis la începutul anul 2021. În cadrul acestuia, clienții își pot aduce mașinile pentru ITP, revizii periodice sau reparații. Nu se pune problema stocului, piesele fiind achiziționate online la plasarea comenzii, iar prețul acestora este inclus în prețul manoperei. Nu sunt oferite reparații mașinilor produse înainte de 1990. Clienții își pot face programare pentru o dată ulterioară zilei curente. Prețul unui serviciu nu variază în timp și nu se gestionează perioada de garanție a pieselor.

Aplicația poate fi utilizată pentru:

- Obținerea de informații despre mașini, angajați și servicii.
- Gestionarea reparațiilor și a reviziilor tehnice.
- Asigurarea valabilității ITP-ului.
- Reținerea costului fiecărei comenzi.

Tehnologiile folosite în cadrul aplicației

Front-end:

- **HTML5** - Este a cincea revizuire a standardului **HyperText Markup Language** care furnizează mijloacele prin care conținutul unui document poate fi adnotat cu diverse tipuri de metadate și indicații de redare. Indicațiile de redare pot varia de la decorațiuni minore ale textului, cum ar fi specificarea faptului că un anumit cuvânt trebuie subliniat sau că o imagine trebuie introdusă, până la scripturi sofisticate, hărți de imagini și formulare. Metadatele pot include informații despre titlul și autorul documentului, informații structurale despre cum este împărțit documentul în diferite segmente, paragrafe, liste, titluri etc. și informații cruciale care permit ca documentul să poată fi legat de alte documente pentru a forma astfel hiperlink-uri (sau web-ul).
- **CSS** - Cascading Style Sheets este un standard pentru formatarea elementelor unui document HTML. Stilurile se pot atașa elementelor HTML prin intermediul unor fișiere externe sau în cadrul documentului.. CSS se poate utiliza și pentru formatarea

elementelor XHTML, XML și SVGL. CSS este unui dintre tehnologiile de bază utilizate în procesul de dezvoltare web, împreună cu HTML și JavaScript.

```css
/* Set height of body and the document to 100% */
body, html {
    height: 100%;
    margin: 0;
    font-family: 'SansationLight' !important;
    background: rgb(22,22,22);
}

/* Style tab links */
.tablink {
    font-weight: bold;
    font-family: 'SansationLight' !important;
    background-color: #557;
    color: azure;
    float: left;
    border: none;
    outline: none;
    cursor: pointer;
    padding: 12px 12px;
    font-size: 17px;
    width: 20%;
}
```

Back-end:

- **Python -** este un limbaj de programare dinamic multi-paradigmă (poate servi ca limbaj pentru software de tipul object-oriented, dar permite și programarea imperativă, funcțională sau procedurală) folosit pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment programate parțial sau în întregime în Python. Sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C.
- **Flask –** este un micro-framework (nu necesită alte unelte sau biblioteci) scris în Python. Nu are strat de validare sau abstractizare a bazelor de date sau alte componente în care bibliotecile deja existente oferă funcții commune.

```python
@app.route("/")
#-------------------DETALII MASINI-------------------
@app.route("/detaliimasini")
def detaliimasini():
    details=[]
    cur=connection.cursor()
    cur.execute('SELECT * FROM detaliimasini')
    curr=cur.fetchall()
    for rez in curr:
        detail={}
        detail['nr_inmatriculare']=rez[0]
        detail['model']=rez[1]
        detail['an_fabricatie']=rez[2]
        detail['serie_sasiu']=rez[3]
        detail['valabilitate_itp']=rez[4]
        details.append(detail)
    cur.close()
    return render_template('detaliimasini.html',details=details)
```

- **Conectarea la baza de date: SQLite -** este o mică bibliotecă C care implementează un motor de baze de date SQL încapsulat (nu are dependențe externe) care oferă posibilitatea de a-l introduce în diverse sisteme, necesită zero-configurare, implementează o mare parte a SQL92, este mai rapid decât baze de date client/server cunoscute pentru majoritatea operațiilor obișnuite și are un API simplu, ușor de folosit.

```
def __init__(self):
    with sqlite3.connect(DATABASE_PATH) as db:
        db.create_function("REGEXP", 2, regex)
        cursor = db.cursor()
        for cmd in self.CREATE_SCRIPT:
            cursor.execute(cmd)
        for cmd in self.INSERT_SCRIPT:
            cursor.execute(cmd)
        cursor.close()
```

Modul de organizare al aplicației

Tabelele utilizate în această aplicație sunt:

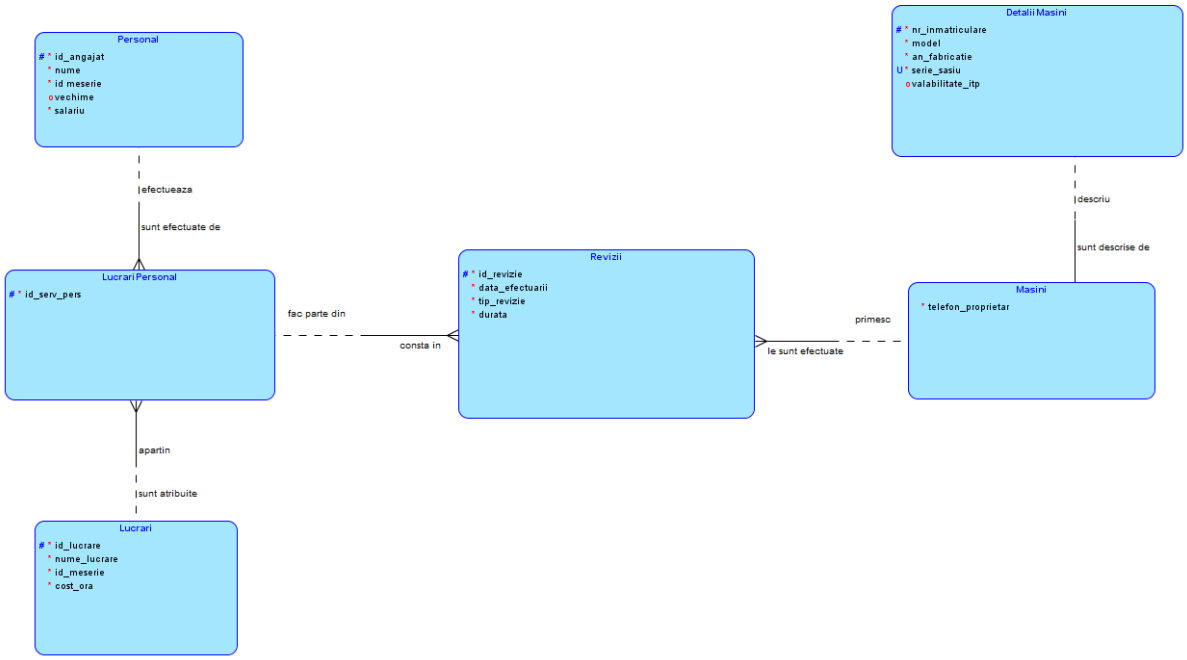
1. **Mașini**
2. **DetaliiMașini**
3. **Personal**
4. **Lucrări**
5. **LucrăriPersonal**
6. **Revizii**

Informațiile folosite în cadrul fiecărei tabele sunt:

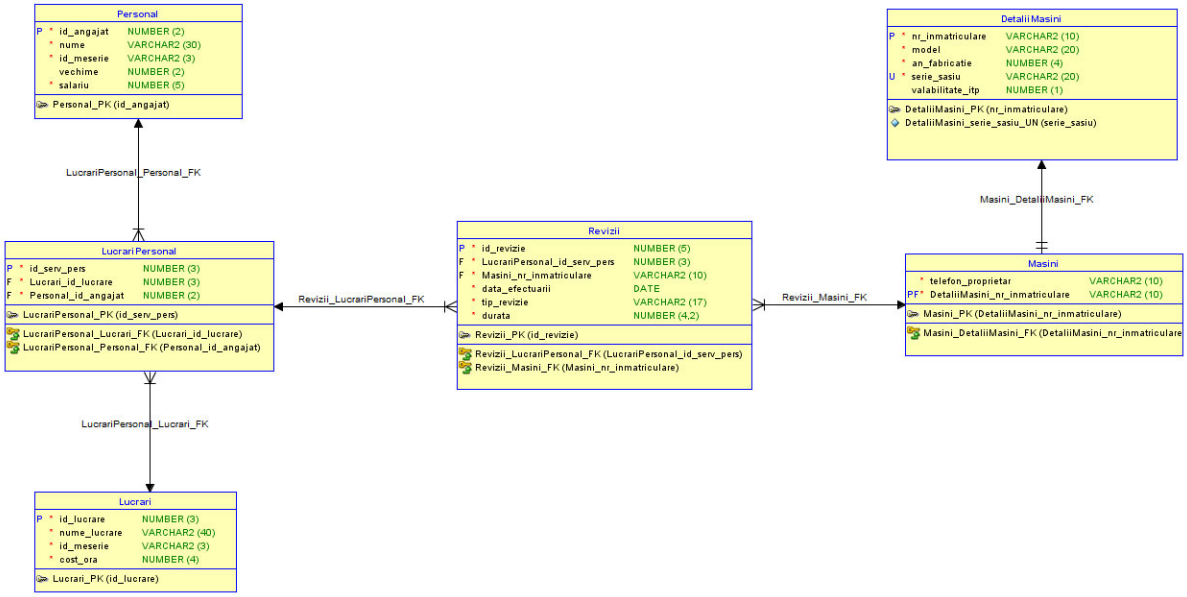
- În tabela **Mașini** – identificate prin numărul de înmatriculare și numărul de telefon al proprietarului.
- În tabela **DetaliiMașini** – extinde tabela Mașini, adăugând informații precum modelul mașinii, anul fabricației, seria șasiului și valabilitatea ITP-ului în ani.
- În tabela **Personal** - lista cu angajații service-ului; conține id-ul unic al angajatului, numele, id-ul meseriei în funcție de care îi sunt atribuite lucrările pe care le poate practica, salariul și vechimea în domeniu.
- În tabela **Lucrări** – lista cu toate serviciile pe care le pot presta angajații service-ului; conține id-ul unic al lucrării, denumirea ei, id-ul meseriei de care este efectuată și costul pe oră al acesteia.
- În tabela **LucrăriPersonal** – produsul cartezian între id-urile angajaților și id-urile lucrărilor, grupate în funcție de id-ul meseriei
- În tabela **Revizii** – lista de lucrări efectuate de un anumit angajat unei anumite mașini identificate prin numărul de înmatriculare, alături de data serviciului și durata acestuia.

Diagrame Entitate-Relație

Modelul logic



Modelul Relațional



Relațiile dintre tabele

În proiectarea modelului fizic al bazei de date au fost utilizate relații de tipul 1:1 și 1:n, rezultate ale simplificării unor relații m:n.

Între **Mașini** și **DetaliiMașini** este o relație de 1:1 deoarece unei mașini i se atribuie un singur set de detalii, iar un set de detalii ale mașinii i se atribuie doar unei singure mașini, deoarece seria șasiului este unică pentru fiecare autovehicul, de aceea în tabela DetaliiMașini ***serie_sasiu*** este și cheie unică. Cheia primară din DetaliiMașini, ***nr_inmatriculare***, este atât cheie primară cât și cheie străină în tabela Mașini.

Între **Personal** și **Lucrări** se află o relație de tip m:n (many to many). Un angajat poate efectua mai multe lucrări în cadrul mai multor revizii, iar o lucrare poate fi efectuată de mai mulți angajați, în funcție de câmpul id_meserie care se regăsește atât în tabela Personal, cât și în tabela Lucrări. Tabela **LucrăriPersonal** este rezultatul relației many to many dintre **Personal** și **Lucrări**. Aceasta conține id-ul angajatului din Personal și id-ul lucrării din Lucrări, grupate în funcție de câmpul id_lucrare menționat anterior. Așadar, această tabelă are câte o relație 1:n cu **Personal** și **Lucrări**, astfel încât baza de date să fie de tipul 3FN. Tabela conține un id unic autoincrementat pentru fiecare pereche a cheilor primare ale celor două tabele, ***id_angajat*** și ***id_lucrare***.

Între **LucrăriPersonal** și **Mașini** există o relație de tip m:n. În cadrul tabelei **Revizii**, care este rezultantă relației dintre aceste două tabele, aceeași lucrare efectuată de un anumit angajat poate fi aplicată mai multor mașini, iar aceeași mașină poate primi mai reparații de tipuri diferite de la angajați diferiți. La fel ca în cazul de mai sus, între tabela Revizii și tabelele LucrăriPersonal și Mașini apar relații de tipul 1:n pentru ca baza de date să se încadreze în tipul 3FN. În cadrul tabelei Revizii există o cheie primară autoincrementată care reprezintă id-ul de pe foaia de lucru al fiecărui serviciu prestat, iar perechile de chei primare are tabelelor LucrăriPersonal și Mașini, ***id_serv_pers*** și ***nr_inmatriculare***, care în această tabelă devin chei străine, nu sunt unice, deoarece unei mașini i se poate aplica aceeași reparație efectuată de același angajat, dar la o dată ulterioară.

Lista de constrângeri

- Tabela **Mașini**:
  - Constrângeri de integritate referențială: nr_inmatriculare este folosit atât ca și cheie primară, cât și cheie străină din tabela DetaliiMașini
  - Constrângeri Check: numărul de telefon să nu aibă litere și să respecte standardul din România (începe cu 07)
- Tabela **DetaliiMașini**:
  - Constrângeri de integritate referențială: nr_inmatriculare este folosit atât ca și cheie primară
  - Constrângeri Unique și Not Null: câmpul ***model*** nu poate fi nul pentru a identifica mașina, pe baza ***an_fabricatie*** se calculează valabilitatea ITP-ului, iar ***seria_sasiu*** este unică pentru fiecare mașină

- Constrângeri Check: numărul de înmatriculare să fie de forma IS 01 ABC, numele modelului să nu conțină caractere speciale, anul fabricației să fie între 1990 și 2021.

- Tabela **Personal**:
  - Constrângeri de integritate referențială: ***id_angajat*** este cheie primară și se autoincrementează
  - Constrângeri Unique și Not Null: ***nume, salariu*** nu acceptă valori nule din motive de contabilitate, ***id_meserie*** este necesar pentru atribuirea serviciilor fiecărui angajat
  - Constrângeri Check: numele angajatului să nu conțină caractere speciale și să fie de forma Nume Prenume, id-ul meseriei poate fi doar REC – recepționer, MEC – mecanic, ELE – electrician, salariul să fie mai mare decât salariul minim pe economie, vechimea în domeniu să fie între 0 și 40 de ani
- Tabela **Lucrări**:
  - Constrângeri de integritate referențială: ***id_lucrare*** este cheie primară și se autoincrementează
  - Constrângeri Unique și Not Null: ***nume_lucrare*** nu poate fi null pentru a putea oferi detalii clientului în legătură cu serviciul aplicat mașinii, prin ***id_meserie*** se face legătura cu tabela Personal, ***cost_ora*** este necesar pentru a calcula costul unei revizii
  - Constrângeri Check: denumirea serviciului să nu conțină caractere speciale, costul pe oră al lucrării să fie mai mare ca 0, id-ul meseriei poate fi doar REC – recepționer, MEC – mecanic, ELE – electrician
- Tabela **LucrăriPersonal**:
  - Constrângeri de integritate referențială: ***id_serv_pers*** este cheia primară asociată fiecărei perechi de chei străine ***id_angajat, id_lucrare***, aceste câmpuri nu pot fi nule
- Tabela **Revizii**:
  - Constrângeri de integritate referențială: ***id_revizie*** este cheie primară care se autoincrementează, ***id_serv_pers*** din LucrariPersonal și ***nr_inmatriculare*** din Masini sunt chei străine, nu pot avea valori nule
  - Constrângeri Unique și Not Null: ***data_efectuarii, durata*** nu pot avea valori nule deoarece sunt folosite la calculul costului total al reviziei, ***tip_revizie*** este nenul pentru a calcula câte reparații de o anumită natură au avut loc într-un interval de timp
  - Constrângeri Check: data efectuării reparației să fie mai mare decât data deschiderii service-ului (1 Ian 2021), durata reparației să fie pozitivă, tipul serviciului să facă parte din lista ITP, Reparatie, Revizie Periodica

Funcționalitatea aplicației

1. **Funcția Select -** este folosită pentru obținerea datelor din baza de date
   Sintaxa operației este SELECT * FROM EMPLOYEES;
   Exemplu de cod folosit:

```
@app.route("/getLucrari",methods=['POST'])
def get_lucrari():
    pk="'"+request.form['id_lucrare']+"'"
    cur=connection.cursor()
    cur.execute("SELECT * FROM lucrari WHERE id_lucrare="+pk)
    lucr=cur.fetchone()
    work={}
    work['id_lucrare']=lucr[0]
    work['nume_lucrare']=lucr[1]
    work['id_meserie']=lucr[2]
    work['cost_ora']=lucr[3]
    cur.close()
    return render_template("lucrariEdit.html",work=work)
```

2. **Funcția Insert –** adaugă o nouă înregistrare.
   Sintaxa operației este:
   INSERT INTO tabela [(coloana [, coloana . . .])]
   VALUES (valoare [, valoare . . .]);
   Exemplu de cod folosit:

```
@app.route('/addDetails', methods=['POST'])
def add_details():
    if request.method=='POST':
        cur=connection.cursor()
        values=[]
        values.append("'"+request.form['nr_inmatriculare']+"'")
        values.append("'"+request.form['model']+"'")
        values.append("'"+request.form['an_fabricatie']+"'")
        values.append("'"+request.form['serie_sasiu']+"'")
        values.append("'"+request.form['valabilitate_itp']+"'")
        fields=['nr_inmatriculare','model','an_fabricatie','serie_sasiu','valabilitate_itp']
        query='INSERT INTO {} ({}) VALUES ({})'.format('detaliimasini',', '.join(fields),', '.join(values))
    try:
        cur.execute(query)
    except:
        cur.close()
        return redirect('/detaliimasini')
    #cur.execute("COMMIT")
    cur.close()
    return redirect('/detaliimasini')
```

3. **Funcția Update -** modifică datele existente cu alte date valide.
   La efectuarea unui update, datele sunt citite din tabelă și completate automat în câmpurile necesare pentru a putea modifica doar câmpurile dorite fără a repeta aceleași informații în celelalte câmpuri.
   Exemplu de cod:

```
@app.route("/editMasini",methods=['POST'])
def edit_masini():
    detaliimasini_nr_inmatriculare="'"+request.form['detaliimasini_nr_inmatriculare']+"'"
    telefon_proprietar="'"+request.form['telefon_proprietar']+"'"
    cur=connection.cursor()
    query="UPDATE masini SET telefon_proprietar={} WHERE detaliimasini_nr_inmatriculare={}".format(telefon_proprietar,detaliimasini_nr_inmatriculare)
    try:
        cur.execute(query)
    except:
        cur.close()
        return redirect('/masini')
    #cur.execute("COMMIT")
    cur.close()
    return redirect("/masini")
```

   Exemplu de interfață pentru update:

4. **Funcția Delete -** Șterge o înregistrare din tabelă.
   Sintaxa operației este DELETE FROM [tabelă] WHERE [condiție];
   Exemplu de cod utilizat:

```
@app.route("/deleteRevizii",methods=['POST'])
def delete_revizii():
    pk="'"+request.form['id_revizie']+"'"
    cur=connection.cursor()
    cur.execute("DELETE FROM revizii WHERE id_revizie="+pk)
    #cur.execute("COMMIT")
    cur.close()
    return redirect("/revizii")
```