Sistem de Gestiune a Vehiculelor și Amenzilor Rutiere

Administrarea Bazelor de Date

Proiect PostgreSQL

Cuprins

1 Introducere

Acest proiect reprezintă un sistem de gestiune pentru vehicule și amenzi rutiere, implementat în PostgreSQL. Sistemul permite evidența vehiculelor, proprietarilor (persoane fizice și juridice), amenzilor emise și plăților efectuate, cu respectarea legislației rutiere din România.

1.1 Obiectivele Proiectului

- Gestiunea entităților (persoane fizice și juridice)
- Evidența vehiculelor și istoricul proprietarilor
- Administrarea amenzilor rutiere și plăților
- Calcularea automată a punctelor de penalizare
- Suspendarea automată a permiselor de conducere
- Aplicarea reducerilor pentru plăți în termen
- Auditare completă a operațiunilor

1.2 Tehnologii Utilizate

- **SGBD:** PostgreSQL
- **Limbaj:** Bash/pgSQL
- **Containerizare:** Docker și Docker Compose
- **Administrare:** PgAdmin

2 Arhitectura Sistemului

2.1 Diagrama Entitate-Relație

Sistemul este organizat în jurul următoarelor entități principale:

- `entitati` - Persoane fizice și juridice (proprietari)
- `vehicule` - Vehiculele înmatriculate
- `istoric_proprietari` - Istoricul deținătorilor pentru fiecare vehicul
- `amenzi_emise` - Amenzile rutiere emise
- `nomenclator_amenzi` - Catalogul infracțiunilor și sancțiunilor
- `agentii_emitente` - Instituțiile care emit amenzi
- `plati` - Plățile efectuate pentru amenzi
- `log_operatiuni` - Jurnal de audit

2.2 Tipuri de Date Personalizate

Sistemul utilizează tipuri ENUM pentru:

```
CREATE TYPE tip_persoana AS ENUM ('Fizica', 'Juridica');
CREATE TYPE status_amenda AS ENUM ('Neplatit', 'Platit');
CREATE TYPE status_permis AS ENUM ('Activ', 'Suspendat');
```

Listing 1: Tipuri de date personalizate

3 Structura Bazei de Date

3.1 Tabelul `entitati`

Stochează informații despre proprietarii vehiculelor.

```
CREATE TABLE entitati (
    id SERIAL PRIMARY KEY,
    nume VARCHAR(100) NOT NULL,
    cnp_cui VARCHAR(20) UNIQUE NOT NULL,
    adresa TEXT,
    telefon VARCHAR(15),
    tip tip_persoana NOT NULL,
    stare_permis status_permis DEFAULT 'Activ',
    CONSTRAINT chk_cnp_lungime CHECK (length(cnp_cui) >= 6)
);
```

Listing 2: Structura tabelului entitati

Caracteristici importante:

- CNP/CUI unic pentru fiecare entitate
- Diferențiere între persoane fizice și juridice
- Gestionarea stării permisului de conducere
- Validare minimă a lungimii CNP/CUI

3.2 Tabelul `vehicule`

Conține informații despre vehiculele înregistrate în sistem.

```
CREATE TABLE vehicule (
    id SERIAL PRIMARY KEY,
    numar_inmatriculare VARCHAR(15) UNIQUE NOT NULL,
    serie_sasiu VARCHAR(17) UNIQUE NOT NULL,
    marca VARCHAR(50) NOT NULL,
    model VARCHAR(50) NOT NULL,
    data_expirare_itp DATE NOT NULL,
    data_expirare_rca DATE NOT NULL
);
```

Listing 3: Structura tabelului vehicule

3.3 Tabelul amenzi_emise

Registru central al amenzilor rutiere.

```
CREATE TABLE amenzi_emise (
    id SERIAL PRIMARY KEY,
    vehicul_id INTEGER REFERENCES vehicule(id),
    entitate_id INTEGER REFERENCES entitati(id),
    agentie_id INTEGER REFERENCES agentii_emitente(id),
    nomenclator_id INTEGER REFERENCES nomenclator_amenzi(id),
    data_emitere DATE DEFAULT CURRENT_DATE,
    suma_initiala DECIMAL(10, 2),
    status status_amenda DEFAULT 'Neplatit'
);
```

Listing 4: Structura tabelului amenzi_emise

3.4 Tabelul nomenclator_amenzi

Catalogul oficial al infracțiunilor rutiere.

```
INSERT INTO nomenclator_amenzi
    (cod_articol, descriere, valoare_standard, puncte_penalizare)
VALUES
    ('VIT-50', 'Depasire viteza 50+ km/h', 1305.00, 9),
    ('TEL-01', 'Folosirea telefonului la volan', 580.00, 6),
    ('RCA-LIPSA', 'Lipsa asigurare RCA', 2000.00, 0);
```

Listing 5: Exemplu nomenclator amenzi

4 Trigger-e și Automatizări

4.1 Trigger 1: Setarea Automată a Sumei Amenzi

La emiterea unei amenzi, suma este preluată automat din nomenclator.

```
CREATE OR REPLACE FUNCTION fn_setare_suma_amenda()
RETURNS TRIGGER AS $$
BEGIN
    SELECT valoare_standard INTO NEW.suma_initiala
    FROM nomenclator_amenzi WHERE id = NEW.nomenclator_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```
CREATE TRIGGER trg_insert_amenda_suma
BEFORE INSERT ON amenzi_emise
FOR EACH ROW EXECUTE FUNCTION fn_setare_suma_amenda();
```

Listing 6: Trigger pentru suma automată

4.2 Trigger 2: Audit pentru Modificări

Orice modificare a statutului amenzii este înregistrată în jurnalul de audit.

```
CREATE OR REPLACE FUNCTION fn_audit_amenzi()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'UPDATE' THEN
        INSERT INTO log_operatiuni
            (nume_tabel, utilizator_db, tip_operatie, detalii)
        VALUES ('amenzi_emise', CURRENT_USER, 'UPDATE',
                jsonb_build_object('id_amenda', OLD.id,
                                   'status_vechi', OLD.status,
                                   'status_nou', NEW.status));
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Listing 7: Trigger de auditare

4.3 Trigger 3: Suspendarea Automată a Permisului

Când o persoană fizică acumulează 20+ puncte în 180 zile, permisul este suspendat automat.

```
CREATE OR REPLACE FUNCTION fn_check_puncte()
RETURNS TRIGGER AS $$
DECLARE
    total_puncte INT;
    v_tip tip_persoana;
BEGIN
    SELECT tip INTO v_tip
    FROM entitati WHERE id = NEW.entitate_id;

    IF v_tip = 'Fizica' THEN
        SELECT COALESCE(SUM(n.puncte_penalizare), 0)
        INTO total_puncte
        FROM amenzi_emise a
        JOIN nomenclator_amenzi n ON a.nomenclator_id = n.id
        WHERE a.entitate_id = NEW.entitate_id
          AND a.data_emitere > (CURRENT_DATE - INTERVAL '180 days
              ');

```

```
        IF total_puncte >= 20 THEN
            UPDATE entitati
            SET stare_permis = 'Suspendat'
            WHERE id = NEW. entitate_id;

            RAISE NOTICE 'ATENTIE: Permisul suspendat!  Puncte:
               %',
                        total_puncte;
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Listing 8: Trigger pentru suspendare permis

5 Proceduri Stocate

5.1 Procedură de Înregistrare Plată cu Reducere

Aplică reducere de 50% dacă plata se face în primele 15 zile.

```
CREATE OR REPLACE PROCEDURE prc_inregistreaza_plata(
    p_amenda_id INT,
    p_suma_oferita DECIMAL
)
LANGUAGE plpgsql AS $$
DECLARE
    v_data_emitere DATE;
    v_suma_standard DECIMAL;
    v_suma_redusa DECIMAL;
BEGIN
    SELECT data_emitere, suma_initiala
    INTO v_data_emitere, v_suma_standard
    FROM amenzi_emise WHERE id = p_amenda_id;

    v_suma_redusa := v_suma_standard / 2;

    IF (CURRENT_DATE - v_data_emitere) <= 15 THEN
        IF p_suma_oferita >= v_suma_redusa THEN
            INSERT INTO plati(amenda_id, suma_achitata)
            VALUES (p_amenda_id, p_suma_oferita);

            UPDATE amenzi_emise SET status = 'Platit'
            WHERE id = p_amenda_id;

            RAISE NOTICE 'Plata acceptata cu reducere! ';
        ELSE
            RAISE EXCEPTION 'Suma insuficienta.  Necesar:  %',
```

```
                                v_suma_redusa;
        END IF;
    ELSE
        IF p_suma_oferita >= v_suma_standard THEN
            INSERT INTO plati(amenda_id, suma_achitata)
            VALUES (p_amenda_id, p_suma_oferita);

            UPDATE amenzi_emise SET status = 'Platit'
            WHERE id = p_amenda_id;

            RAISE NOTICE 'Plata integrala acceptata. ';
        ELSE
            RAISE EXCEPTION 'Termen reducere depasit. Necesar: %'
               ,
                                v_suma_standard;
        END IF;
    END IF;
END;
$$;
```

Listing 9: Procedură plată cu reducere

Exemplu de utilizare:

```
-- Plata cu reducere (in primele 15 zile)
CALL prc_inregistreaza_plata(1, 145.00);
```

5.2 Procedură cu Cursor - Raport Restanțieri

Generează un raport cu toți debitorii.

```
CREATE OR REPLACE PROCEDURE prc_raport_restantieri()
LANGUAGE plpgsql AS $$
DECLARE
    cur_datornici CURSOR FOR
        SELECT e.nume, a.suma_initiala, a.data_emitere
        FROM amenzi_emise a
        JOIN entitati e ON a.entitate_id = e.id
        WHERE a.status = 'Neplatit';
    rec RECORD;
BEGIN
    OPEN cur_datornici;
    LOOP
        FETCH cur_datornici INTO rec;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Datornic: % | Suma: % | Data: %',
                     rec.nume, rec.suma_initiala, rec.
                        data_emitere;
    END LOOP;
    CLOSE cur_datornici;
```

```
END;
$$;
```

Listing 10: Procedură cursor pentru raportare

5.3 Procedură Job Zilnic - Verificare ITP

Procedură automată pentru verificarea vehiculelor cu ITP expirat.

```
CREATE OR REPLACE PROCEDURE prc_job_verificare_itp_zilnic()
LANGUAGE plpgsql AS $$
DECLARE
    v_nr_expirate INT;
BEGIN
    SELECT COUNT(*) INTO v_nr_expirate
    FROM vehicule
    WHERE data_expirare_itp < CURRENT_DATE;

    IF v_nr_expirate > 0 THEN
        INSERT INTO log_operatiuni
            (nume_tabel, utilizator_db, tip_operatie, detalii)
        VALUES (
            'vehicule',
            'SYSTEM_JOB',
            'ALERT',
            jsonb_build_object(
                'mesaj', 'Job zilnic:  Vehicule cu ITP expirat',
                'cantitate', v_nr_expirate
            )
        );

        RAISE NOTICE 'Job finalizat: % vehicule cu ITP expirat. '
           ,
                    v_nr_expirate;
    END IF;
END;
$$;
```

Listing 11: Job de verificare ITP

6 Vederi (Views)

6.1 View: Dosarul Autoturismului

Oferă o imagine de ansamblu asupra stării fiecărui vehicul.

```
CREATE OR REPLACE VIEW v_dosar_auto AS
SELECT
    v.numar_inmatriculare,
```

```
    v.marca,
    e.nume AS proprietar_curent,
    CASE
        WHEN v.data_expirare_itp < CURRENT_DATE
        THEN 'ITP EXPIRAT'
        ELSE 'ITP VALID'
    END AS stare_itp,
    COUNT(a.id) AS total_amenzi
FROM vehicule v
JOIN istoric_proprietari ip
    ON v.id = ip.vehicul_id AND ip. data_sfarsit IS NULL
JOIN entitati e ON ip. entitate_id = e.id
LEFT JOIN amenzi_emise a ON v.id = a.vehicul_id
GROUP BY v. id, e.nume, v.data_expirare_itp;
```

Listing 12: View dosar auto

Exemplu interogare:

```
SELECT * FROM v_dosar_auto WHERE stare_itp = 'ITP EXPIRAT';
```

7 Securitate și Roluri

7.1 Definirea Rolurilor

Sistemul definește două roluri principale:

```
CREATE ROLE ofiter_politie;
CREATE ROLE casier;
```

Listing 13: Crearea rolurilor

7.2 Atribuirea Privilegiilor

```
-- Ofiter de politie:  poate vedea si modifica amenzi
GRANT SELECT, INSERT, UPDATE ON amenzi_emise, entitati
TO ofiter_politie;

-- Casier: poate vedea amenzi si inregistra plati
GRANT SELECT ON amenzi_emise TO casier;
GRANT INSERT ON plati TO casier;
```

Listing 14: Granturi pentru roluri

7.3 Trigger DDL - Protecție împotriva Ștergerii

```sql
CREATE OR REPLACE FUNCTION fn_prevent_drop()
RETURNS event_trigger AS $$
BEGIN
    RAISE EXCEPTION
        'Este interzisa stergerea tabelelor in productie! ';
END;
$$ LANGUAGE plpgsql;

-- Activare (comentat implicit pentru dezvoltare)
-- CREATE EVENT TRIGGER trg_no_drop
--     ON ddl_command_start
--     WHEN TAG IN ('DROP TABLE')
--     EXECUTE FUNCTION fn_prevent_drop();
```

Listing 15: Event trigger pentru protecție

8 Instalare și Configurare

8.1 Cerințe Preliminare

- Docker
- Docker Compose
- Client PostgreSQL sau PgAdmin

8.2 Pași de Instalare

8.2.1 Pasul 1: Clonarea Repository-ului

```
git clone https://github.com/sorynturda/ABD_proiect.git
cd ABD_proiect
```

8.2.2 Pasul 2: Pornirea Containerelor

```
docker compose up -d --build
```

Această comandă va:

- Crea containerul PostgreSQL
- Expune portul 5432
- Executa automat scripturile de inițializare
- Crea structura tabelelor, trigger-elor, procedurilor și vederi

8.2.3 Pasul 3: Conectarea la Baza de Date

Parametri de conexiune:

- **Host:** localhost
- **Port:** 5432
- **Database:** (verificați docker-compose.yml)
- **User:** (verificați docker-compose.yml)

8.2.4 Pasul 4: Popularea Bazei de Date

După conectare, executați scriptul de populare:

```
psql -h localhost -U <user> -d <database>
     -f database/populare_db.sql
```

Sau din PgAdmin, deschideți și executați `populare_db.sql`.

9 Scenarii de Testare

9.1 Test 1: Verificare Structură Bază de Date

```
SELECT table_name FROM information_schema.tables
WHERE table_schema = 'public';
```

Listing 16: test_1.sql

9.2 Test 2: Emitere Amendă cu Calcul Automat

Testează trigger-ul de setare automată a sumei.

```
INSERT INTO amenzi_emise
    (vehicul_id, entitate_id, agentie_id, nomenclator_id)
VALUES (1, 1, 1, 1);

-- Verificare suma populata automat
SELECT suma_initiala FROM amenzi_emise
WHERE id = (SELECT MAX(id) FROM amenzi_emise);
```

Listing 17: test_2.sql

9.3 Test 3: Plată cu Reducere

```sql
-- Plata in primele 15 zile (cu reducere)
CALL prc_inregistreaza_plata(1, 145.00);

-- Verificare status
SELECT status FROM amenzi_emise WHERE id = 1;
```

Listing 18: test_3.sql

9.4 Test 4: Suspendare Automată Permis

```sql
-- Emitere multiple amenzi cu puncte
-- pentru a depasi pragul de 20 puncte

-- Verificare stare permis
SELECT stare_permis FROM entitati WHERE id = 3;
-- Ar trebui sa returneze 'Suspendat'
```

Listing 19: test_4.sql

9.5 Test 5: Verificare Audit Log

```sql
SELECT * FROM log_operatiuni ORDER BY data_operatie DESC;
```

Listing 20: test_5.sql

9.6 Test 6: Raport Dosare Auto

```sql
SELECT * FROM v_dosar_auto;
```

Listing 21: test_6.sql

10 Caracteristici Avansate

10.1 Tipuri de Date JSONB

Log-ul de audit folosește JSONB pentru flexibilitate:

```sql
CREATE TABLE log_operatiuni (
    ...
    detalii JSONB
);

-- Exemplu inserare
INSERT INTO log_operatiuni(..., detalii) VALUES
    (..., jsonb_build_object('key', 'value'));
```

10.2 Constraint-uri Complexe

```
-- Validare perioada istoric proprietari
CONSTRAINT chk_perioada CHECK
    (data_sfarsit IS NULL OR data_sfarsit >= data_start)

-- Validare lungime CNP
CONSTRAINT chk_cnp_lungime CHECK (length(cnp_cui) >= 6)
```

10.3 Cascade și Restrict

```
-- Stergere cascade pentru plati
amenda_id INTEGER REFERENCES amenzi_emise(id)
    ON DELETE CASCADE

-- Prevent stergere entitate cu vehicule
entitate_id INTEGER REFERENCES entitati(id)
    ON DELETE RESTRICT
```

11 Diagrame și Fluxuri

11.1 Fluxul de Emitere Amendă

1. Agent rutier identifică infracțiune
2. Se inserează înregistrare în `amenzi_emise`
3. **Trigger** `fn_setare_suma_amenda()` setează automat suma
4. **Trigger** `fn_check_puncte()` calculează punctele totale
5. Dacă puncte ≥ 20 în ultimele 180 zile $\rightarrow$ suspendare permis
6. Înregistrare în log audit

11.2 Fluxul de Plată

1. Cetățean dorește să plătească amenda
2. Apel procedură `prc_inregistreaza_plata(id, suma)`
3. Verificare: Emitere < 15 zile? $\rightarrow$ Reducere 50%
4. Validare sumă
5. Inserare în tabelul `plati`
6. Update status amendă la 'Platit'
7. **Trigger audit** înregistrează modificarea

12 Întreținere și Monitorizare

12.1 Backup

```
# Backup complet
pg_dump -h localhost -U <user> <database> > backup.sql

# Backup doar structura
pg_dump -h localhost -U <user> --schema-only <database>
    > schema.sql

# Backup doar datele
pg_dump -h localhost -U <user> --data-only <database>
    > data.sql
```

12.2 Restore

```
psql -h localhost -U <user> <database> < backup.sql
```

12.3 Interogări Utile de Monitorizare

```
-- Numar total amenzi pe luna
SELECT DATE_TRUNC('month', data_emitere) AS luna,
       COUNT(*) AS total_amenzi
FROM amenzi_emise
GROUP BY luna ORDER BY luna DESC;

-- Incasari totale
SELECT SUM(suma_achitata) AS total_incasari
FROM plati;

-- Top 5 contravenientisti
SELECT e.nume, COUNT(a.id) AS numar_amenzi
FROM entitati e
JOIN amenzi_emise a ON e. id = a.entitate_id
GROUP BY e.id, e.nume
ORDER BY numar_amenzi DESC
LIMIT 5;

-- Persoane cu permisul suspendat
SELECT nume, stare_permis FROM entitati
WHERE stare_permis = 'Suspendat';
```

Listing 22: Statistici generale

13 Îmbunătățiri Viitoare

13.1 Funcționalități Propuse

1. **Notificări automate:**
   - Email/SMS la emiterea amenzii
   - Reminder înainte de expirarea ITP/RCA
   - Alertă la apropierea de pragul de suspendare
2. **API REST:**
   - Interfață web pentru cetățeni
   - Aplicație mobilă pentru agenți
   - Integrare cu alte sisteme guvernamentale
3. **Analytics:**
   - Dashboard-uri pentru manageri
   - Rapoarte predictive
   - Identificare zone cu multe infracțiuni
4. **Plăți online:**
   - Integrare gateway de plată
   - Generare chitanță automată
   - Istoric plăți pentru utilizator

13.2 Optimizări Database

```sql
-- Index pentru cautari frecvente
CREATE INDEX idx_amenzi_status
    ON amenzi_emise(status);

CREATE INDEX idx_amenzi_entitate
    ON amenzi_emise(entitate_id);

CREATE INDEX idx_vehicul_numar
    ON vehicule(numar_inmatriculare);

-- Index pentru calcul puncte
CREATE INDEX idx_amenzi_data_entitate
    ON amenzi_emise(entitate_id, data_emitere);
```

Listing 23: Indexuri propuse

14 Concluzie

Acest sistem demonstrează implementarea unui proiect complex de bază de date cu:

- **12 tabele** interconectate
- **3 trigger-e DML** pentru automatizare
- **1 event trigger DDL** pentru securitate
- **3 proceduri stocate** cu logică business complexă
- **1 cursor** pentru parcurgere date
- **View-uri** pentru raportare
- **Roluri și privilegii** pentru control acces
- **Audit complet** al operațiunilor

Sistemul este containerizat cu Docker, ușor de instalat și configurat, și respectă principiile de design modern ale bazelor de date relaționale.

14.1 Resurse Adiționale

- Repository GitHub: https://github.com/sorynturda/ABD_proiect
- Documentație PostgreSQL: https://www.postgresql.org/docs/
- PL/pgSQL Guide: https://www.postgresql.org/docs/current/plpgsql.html