DOCUMENTAȚIE PROIECT BAZE DE DATE DISTRIBUITE

## 1. Descrierea temei

Proiectul constă în proiectarea și implementarea unei baze de date relaționale pentru gestiunea activității unei ferme agricole. Sistemul permite administrarea fermelor, parcelelor, culturilor, angajaților, utilajelor, lucrărilor agricole, producției și stocării acesteia în depozite.

Pe baza datelor stocate sunt generate rapoarte complexe care oferă informații relevante privind costurile lucrărilor, performanța angajaților, gradul de ocupare al depozitelor și producția agricolă raportată la suprafață și cultură.

Aplicația software asociată permite interogarea bazei de date prin intermediul procedurilor stocate și afișarea rezultatelor sub formă tabelară și grafică.

## 2. Descrierea bazei de date

### 2.1 Diagrama bazei de date

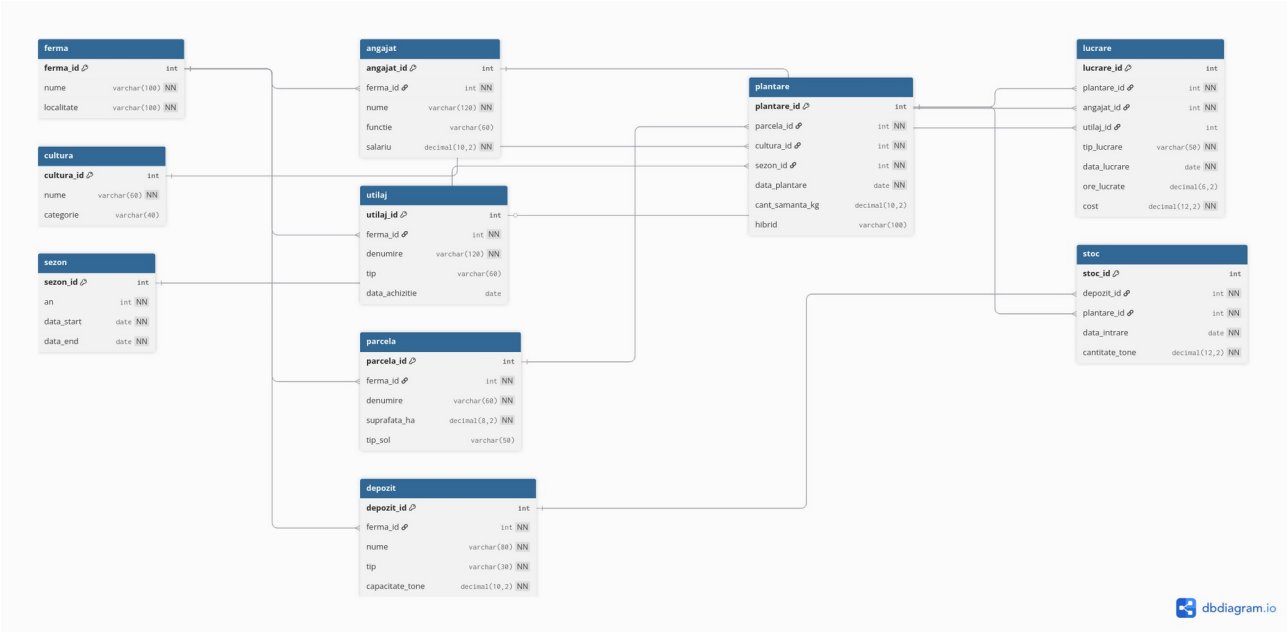



Figura 1. Diagrama bazei de date pentru gestiunea fermei agricole

Baza de date este proiectată folosind modelul relațional și este compusă din 10 tabele interconectate prin chei primare și chei externe. Diagrama evidențiază relațiile dintre entitățile principale ale aplicației: ferme, parcele, culturi, sezoane, plantări, angajați, utilaje, lucrări agricole, depozite și stocuri.

### 2.2 Structura tabelelor

**FERMA –** stochează informații generale despre fermele agricole (nume, localitate).

- **PARCELA** – reprezintă suprafețele agricole aparținând unei ferme, pe care se pot realiza plantări.
- **CULTURA** – conține tipurile de culturi agricole disponibile.
- **SEZON** – definește anii agricoli și intervalele calendaristice aferente.
- **PLANTARE** – face legătura între parcelă, cultură și sezon, descriind ce cultură este plantată pe o parcelă într-un anumit sezon.
- **ANGAJAT** – stochează date despre angajații fermelor.
- **UTILAJ** – conține utilajele agricole deținute de fiecare fermă.
- **LUCRARE** – reprezintă lucrările agricole efectuate pe o plantare, de către un angajat, folosind opțional un utilaj.
- **DEPOZIT** – definește spațiile de depozitare ale fermelor, cu capacitate limitată.

- **STOC** – evidențiază cantitățile de producție depozitate, asociate unei plantări și unui depozit.

**2.3 Constrângeri de integritate**

Pentru asigurarea corectitudinii și consistenței datelor, baza de date utilizează următoarele tipuri de constrângeri:

- **Chei primare (PRIMARY KEY)** pentru identificarea unică a fiecărei înregistrări.
- **Chei externe (FOREIGN KEY)** pentru definirea relațiilor dintre tabele.
- **Constrângeri de unicitate (UNIQUE)**, de exemplu:
    - o parcelă are o denumire unică în cadrul aceleiași ferme;
    - o plantare este unică pentru o parcelă într-un sezon.
- **Constrângeri de validare (CHECK)**, precum:
    - suprafața parcelei și salariul angajatului trebuie să fie valori pozitive;
    - capacitatea unui depozit trebuie să fie mai mare decât zero.
- **Triggere**, utilizate pentru implementarea regulilor de business care nu pot fi exprimate direct prin constrângeri SQL, precum:
    - verificarea apartenenței angajatului la ferma corespunzătoare plantării;
    - prevenirea depășirii capacității maxime a unui depozit.

**2.4 Proceduri și funcții (proceduri stocate pentru rapoarte)**

**Pachetul PKG_RAPOARTE_INTERACTIVE conține rapoarte parametrizabile, concepute pentru a fi apelate din aplicație. Acestea oferă informații agregate și analize economice relevante pentru managementul unei ferme agricole.**

**R1 – Grad de ocupare depozite (raport_ocupare_depozite)**

**Scop:**
Analizează cât de ocupate sunt depozitele fermei într-un anumit sezon.

**Ce face:**
Pentru fiecare depozit, raportul calculează:

- cantitatea totală de producție stocată (tone),
- gradul de ocupare exprimat procentual,
- cantitatea de spațiu liber rămasă.

**Cum se calculează:**
Valorile sunt obținute prin agregarea intrărilor din tabelul STOC, raportate la capacitatea fiecărui depozit. Se folosesc funcții de tip SUM, NVL, ROUND și NULLIF pentru a obține valori corecte chiar și în lipsa stocurilor.

**Parametri importanți:**

- anul sezonului,
- ferma (opțional),
- un prag minim de ocupare (%) pentru filtrare.

---

**R2 – Profitabilitatea grâului pe soi (raport_profit_grau_soi)**

**Scop:**
Evaluează care soiuri de grâu sunt cele mai profitabile într-un sezon.

**Ce face:**
Raportul compară soiurile de grâu în funcție de:

- producția obținută (tone),
- costurile totale (lucrări + sămânță),
- profitul total și profitul raportat la hectar,
- raportul profit / cost (indicator de eficiență).

**Cum se calculează:**

- Producția este calculată din STOC;
- Costurile lucrărilor sunt agregate din LUCRARE;
- Costul seminței se calculează folosind cantitatea de sămânță și un preț introdus din aplicație;
- Profitul este diferența dintre venit (tone × preț/tonă) și investiție.

**Parametri importanți:**

- prețul de vânzare al grâului (lei/tonă),
- prețul seminței (lei/kg),
- numărul maxim de soiuri afișate (Top N).

---

**R3 – Parcelele cele mai productive pentru o cultură (raport_parcele_productive_cultura)**

**Scop:**
Identifică cele mai productive parcele pentru o cultură selectată.

**Ce face:**
Raportul afișează parcelele care au obținut cele mai bune rezultate pentru o anumită cultură, ordonate după productivitate (tone/hectar). Opțional, poate calcula și profitul.

**Cum se calculează:**

- Producția totală este agregată din intrările în siloz (STOC);
- Costurile sunt calculate din lucrările efectuate (LUCRARE);
- Productivitatea se exprimă în tone/hectar;
- Dacă se introduce un preț de vânzare, se calculează venituri și profit.

**Parametri importanți:**

- cultura dorită (ex: Grâu, Porumb),
- prețul de vânzare (opțional),
- numărul de parcele afișate (Top N).

---

**R4 – Risc de rotație necorespunzătoare a culturilor (raport_risc_rotatie)**

**Scop:**
Detectează probleme de rotație a culturilor pe parcele.

**Ce face:**
Raportul identifică parcelele unde aceeași cultură a fost cultivată în ani consecutivi, situație considerată un risc agronomic.

**Cum se calculează:**
Se analizează istoricul plantărilor folosind funcții analitice (LAG) pentru a compara cultura curentă cu cea din sezonul anterior.

**Parametri importanți:**

- anul analizat,
- ferma (opțional).

### 3. Descrierea aplicației

Aplicația dezvoltată are ca scop analiza și vizualizarea datelor agricole pentru o fermă, oferind rapoarte interactive legate de producție, costuri, profitabilitate și utilizarea resurselor. Utilizatorul poate introduce parametri direct din interfața web, iar aplicația generează automat tabele și grafice pe baza datelor existente în baza de date Oracle.

#### 3.1 Diagrama de clase

**Diagrama de clase**

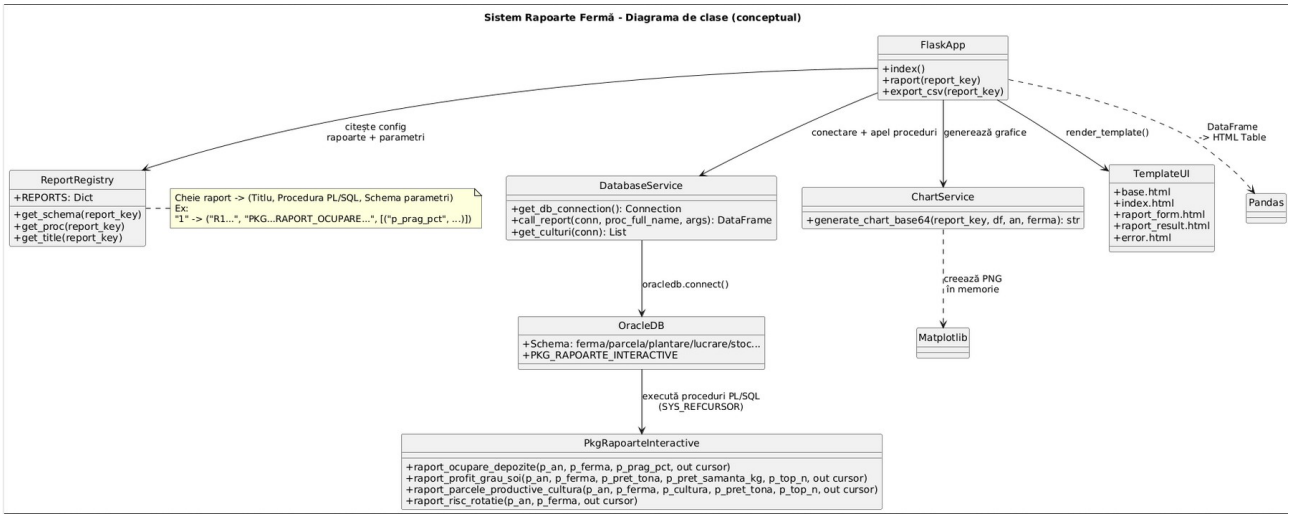



Diagrama de clase descrie componentele aplicației și modul în care acestea colaborează pentru generarea rapoartelor.

**Clase / componente (conceptual)**

- **FarmReportsWebApp (Flask Controller)**
    - gestionează rutele (/, /raport/<key>, /export/<key>)
    - preia parametrii din formular și coordonează execuția raportului
- **ReportRegistry (Catalog rapoarte)**
    - structura REPORTS
    - definește: titlu raport, procedură PL/SQL, schema de parametri
- **DatabaseService (Acces Oracle)**
    - creează conexiunea la DB
    - apelează procedurile care întorc SYS_REFCURSOR

    - transformă rezultatele în pandas.DataFrame
- **ChartService (Vizualizare)**
    - generează grafice Matplotlib în funcție de raport
    - codifică imaginea în Base64 pentru afișare în HTML
- **Views (Template-uri Bootstrap)**
    - index.html, raport_form.html, raport_result.html, error.html

**3.2 Structura claselor**

Structura aplicației este organizată conceptual în următoarele clase:

## FarmReportsWebApp (logica Flask)

**Rol:** controlul aplicației (Controller în MVC)

**Responsabilități:**

- route handling (GET/POST)
- validare și conversie parametri:
    - int, float, None pentru opționale
    - verificare parametri obligatorii (ex: p_pret_tona în R2)
- construirea listei args în ordinea corectă pentru callproc
- trimiterea rezultatelor către template

**Rute principale:**

- GET / → listă rapoarte (meniul)
- GET /raport/<key> → formular parametri
- POST /raport/<key> → execuție raport + afișare rezultate

---

## ReportRegistry (REPORTS)

**Rol:** configurare unitară a rapoartelor

Pentru fiecare raport se definește:

- titlu (pentru UI)
- procedură PL/SQL (ex: PKG_RAPOARTE_INTERACTIVE.RAPORT_PROFIT_GRAU_SOI)
- schema de parametri:
    - nume parametru (ex: p_top_n)
    - tip (int, float, string, *_or_none)
    - default
    - descriere afișată în form

---

## DatabaseService

**Rol:** acces la baza de date Oracle

**Funcții:**

- get_db_connection() – creează conexiunea folosind user/pass/dsn
- call_report(conn, proc, args):
    - creează un cursor de output
    - apelează procedura PL/SQL
    - citește coloanele + rândurile
    - returnează DataFrame

---

## ChartService

**Rol:** generare grafic pentru raport (dacă există coloane potrivite)

**Exemple de coloane folosite:**

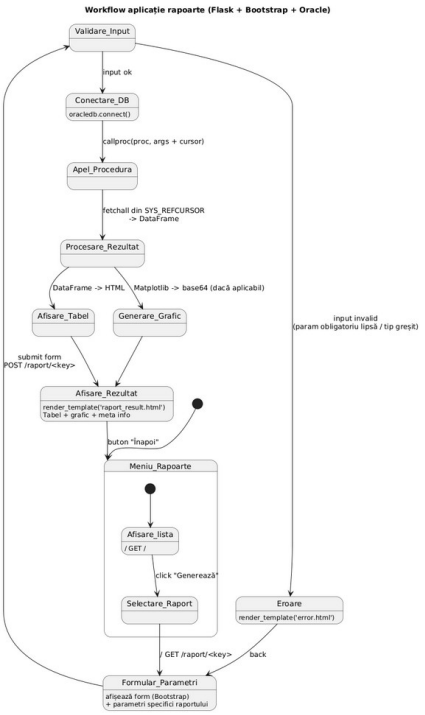
- R1: DEPOZIT, GRAD_OCUPARE_PCT
- R2: SOI_GRAU, RAPORT_PROFIT_COST / PROFIT_PE_HA
- R3: PARCELA, TONE_PE_HA / PROFIT_PE_HA
- R4: PARCELA, ESTE_REPETARE

Graficul este salvat în memorie și returnat ca Base64 pentru inserare în HTML.

### 3.3 Diagrama de stări și fluxul de lucru (workflow)

Stări principale

1. Home (meniul rapoartelor)
2. ReportSelected (alegerea raportului)
3. ParameterInput (completare parametri)
4. Validation (validare + conversie tipuri)
5. DBExecution (apel procedură PL/SQL)
6. ResultRender (tabel + grafic)
7. Error (mesaj de eroare)




**Workflow (pas cu pas)**

1. Utilizatorul intră pe pagina principală
2. Alege raportul dorit
3. Primește formular pentru parametri (Bootstrap)
4. Trimite formularul
5. Backend-ul validează parametrii și construiește argumentele
6. Aplicația se conectează la Oracle și rulează procedura PL/SQL
7. Preia rezultatul din SYS_REFCURS0R

8. Afișează tabelul + generează grafic (dacă e relevant)

### 3.4 Prezentarea modului în care se face conexiunea cu baza de date

**Configurare (din variabile de mediu)**

- DB_USER, DB_PASS, DB_HOST, DB_PORT, DB_SERVICE
- se formează DSN:
    - DSN = f"{DB_HOST}:{DB_PORT}/{DB_SERVICE}"

**Pașii de conectare și execuție**

1. **Conectare:**

```
conn = oracledb.connect(user=DB_USER, password=DB_PASS, dsn=DSN)
```

2. **Apel procedură (SYS_REFCURSOR):**

```
with conn.cursor() as cur:
    out_cur = conn.cursor()
    cur.callproc(proc_full_name, [*args, out_cur])
```

3. **Citire rezultate:**

```
cols = [d[0] for d in out_cur.description]
rows = out_cur.fetchall()
df = pd.DataFrame(rows, columns=cols)
```

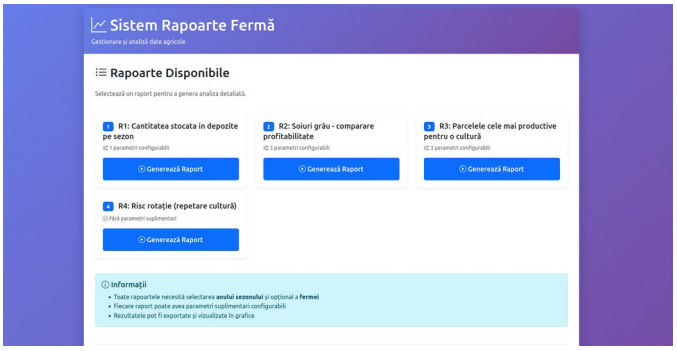
4. **Închidere conexiune:**
   Conexiunea este închisă după execuție (finally: conn.close()), ca să nu rămână sesiuni deschise.

**Gestionarea erorilor**

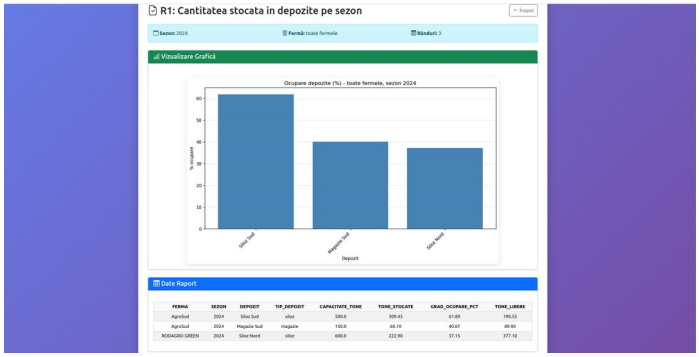
Execuția e încapsulată în try/except, iar aplicația returnează o pagină de eroare (error.html) cu mesaj relevant, fără să se oprească serverul.

### 4. Capturi de ecran pentru interfețe și rapoarte

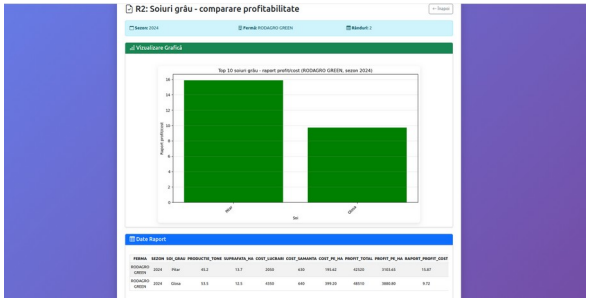
4.1 Pagina principala



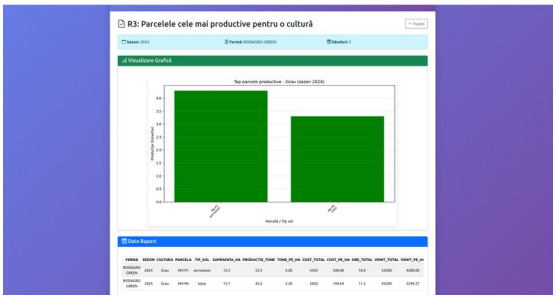

4.2 R1-Ocupare Depozite



4.3 R2: Soiuri grâu - comparare profitabilitate




4.4 R3:Parcelele cele mai productive




## 5. Concluzii

Aplicația oferă o soluție eficientă și flexibilă pentru analiza datelor agricole, permițând generarea de rapoarte complexe pe baza unor parametri introduși de utilizator. Prin integrarea Flask, Oracle Database și a procedurilor PL/SQL, sistemul asigură performanță, claritate în rezultate și suport decizional pentru managementul fermei.

## 6. Bibliografie

1. Oracle Database Documentation – PL/SQL Packages and Procedures

2. Oracle Database Documentation – SQL Language Reference

3. Python Official Documentation – https://docs.python.org

4. Pandas Documentation – https://pandas.pydata.org

5. Matplotlib Documentation – https://matplotlib.org