
**UNIVERSITATEA TEHNICĂ “GHEORGHE ASACHI” DIN IAȘI**
**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI**

**DISCIPLINA EXTRAGEREA CUNOȘTIINȚELOR DIN BAZE DE DATE**

# PROIECAREA UNEI BAZE DE DATE SQL
# PENTRU
# MONITORIZAREA UNEI GRADINI ZOOLOGICE

**Coordonator,** **Student,**

**Prof. Ionut Baltariu** **Pasa Larisa, grupa 1410A**

# Cuprins

**Capitolul 1. Descrierea proiectului**

**Capitolul 2. Structura și inter-relaționarea tabelelor**

**Capitolul 3 Descrierea constrângerilor utilizate**

**Capitolul 4. Descrierea logicii stocate**

- **4.1 Pachete**
- **4.2 Triggeri**

**Capitolul 5. Testarea aplicatiei**

## Capitolul 1. Descrierea proiectului

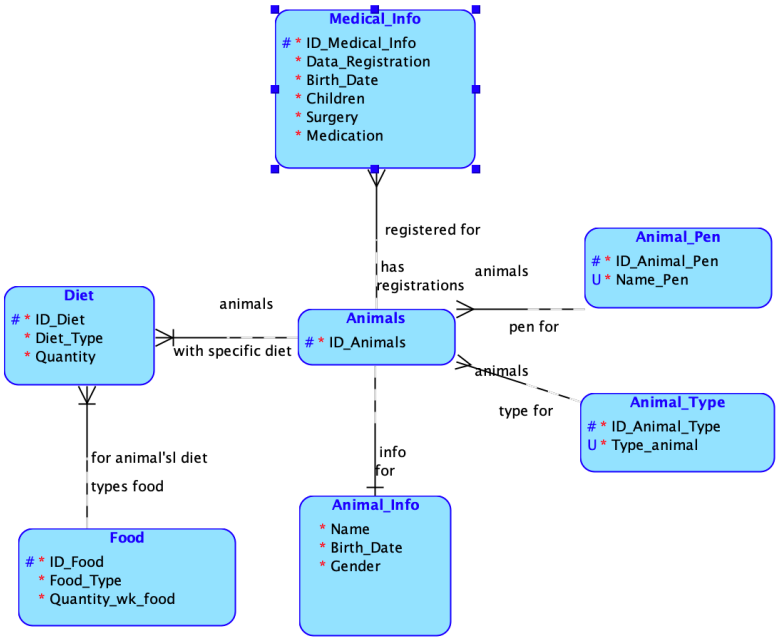
Sistemul propus are ca obiectiv gestionarea eficientă a bazei de date a animalelor din grădina zoologică. Acesta va conține înregistrări detaliate pentru fiecare animal, inclusiv informații generale precum numele și data de naștere. În plus, va exista o standardizare a tipurilor de animale pentru a se potrivi cu tarcurile existente. Fiecare tarc va fi definit în funcție de tipul de animal și ar putea include, în unele cazuri, corpuri anexe.

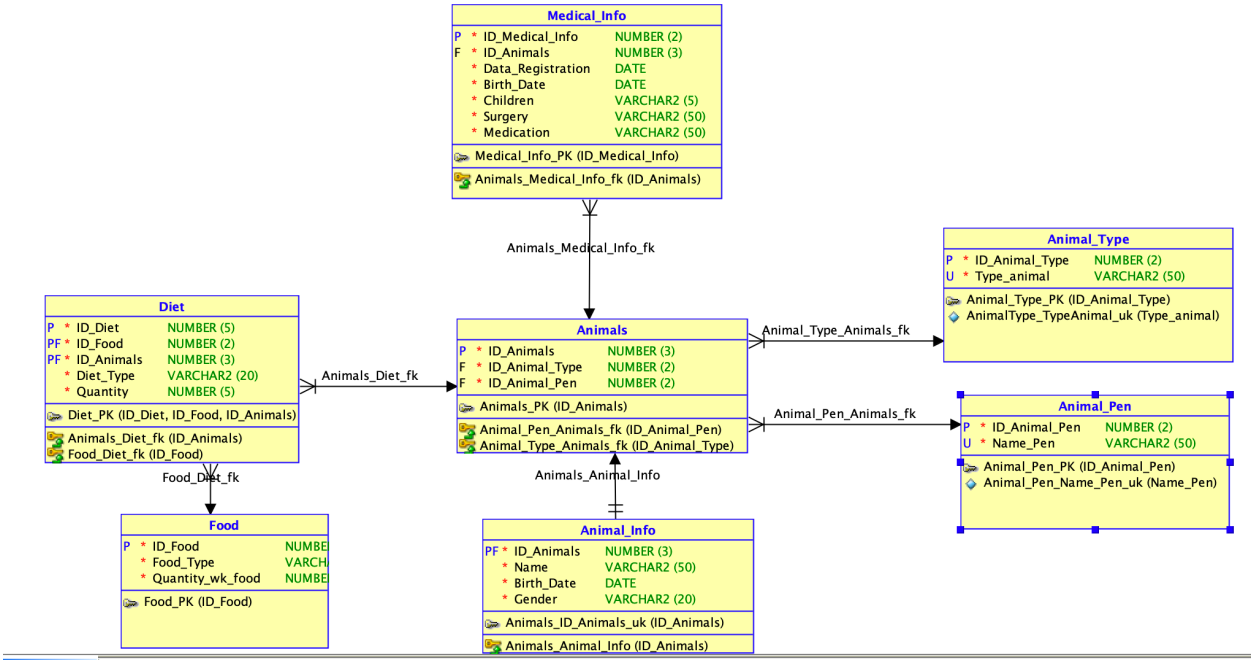
Un aspect important al sistemului va fi gestionarea dietei animalelor. Astfel, pentru fiecare animal se vor înregistra detalii despre tipul alimentației, componentele acesteia și cantitățile necesare .

De asemenea, în cadrul grădinii zoologice, există un cabinet veterinar specializat care va avea propriul său registru pentru înregistrarea vizitelor și tratamentelor animalelor. Acest registru va conține informații medicale relevante, inclusiv diagnostice, tratamente, posibile operații și medicamente prescrise, alături de data de naștere a animalului și alte detalii care pot influența îngrijirea acestuia.

Prin implementarea acestui sistem de evidență, se va asigura o monitorizare a animalelor, facilitând comunicarea între departamente .

## Capitolul 2. Structura și inter-relaționarea tabelelor






**Tabelele folosite sunt:**

- **Animals**: elementul principal al unei gradini Zoologice sunt animalele. Avem nevoie sa stocam animalele din zoo intr-o entitate Animals
- **Animal_Info:** stocarea informatiilor referitoate la fiecare animal in parte se va realiza in entitatea Animal_Info in care date de interes sunt numele si data de nastere ale animalului.
- **Animal_Type:** avem nevoie sa stocam datele despre tipurile de animale existente in zoo pentru o imagine de ansamblu asupra gradinii zoologice
- **Animal_Pen:** trebuie sa stim tarcurile in care animalele pot locui pentru a le organiza corespunzator
- **Medical_Info:** ne intereseaza sa avem o evidenta asupra starii de sanatate a animalelor pentru tratament si dieta potrivite
- **Food:** este de interes stocul de mancare din gradina zoologica. In aceasta entitate se stocheaza ce tipuri de mancare sunt disponibile.
- **Diet:** vom folosi entitatea diet pentru stocarea datelor despre nutritia fiecarui animal

**În proiectarea acestei baze de date se identifică urmatoarele tipuri de relatii:**

- **One – to – One :** “Animal_Info” are o legatura de 1:1 cu entitatea “Animals” . In cea de-a douatabela mentionata se afla doar numarul de identificare al animalului, iar in “Animal Info” este o inregistrare cu datele sale generale. Legatura 1:1 ajuta la mentinereaconstrangerii pentru a nu aparea inregistrari multiple si diferite pentru un

singur animal. Aceasta s-a realizat prin setarea drept cheie primara si unica a “ID_Animals”.

- **One – to – Many:** Entitatea “Animals” este entitatea principala care contine toate animalelepe baza carora sunt create si au sens si celelalte entitati. Aceasta detine o relatie1:ncu entitatile “Animal_Type” si “Animal_Pen”. Mai multe animale pot fi de acelasi tip(canin, felin,rozator) si mai multe animale pot sta in acelasi tarc, dar un animal nupoate sta in >=2 tarcuri simultan. Entitatea “Medical_Info” poate avea mai multe inregistrari cu acelasi animal din entitatea “Animals”, dar un animal nu poate fi repartizat decat la un singur cabinet, cel al gradinii zoologice. Astfel se implementeaza o relatie 1:n

- **Many – to -Many :** Entitatea “Diet” creeaza o legatura n:n cu entitatea “Animals” din care preiaID_Animals pentru identificare si cu entitatea“Food” de unde extrage tipuriledemancare cu ID_Food . Un animal poate aparea in mai multe inregistrari din entitata“Diet” si un tip de macarea poate fi mancat de mai multe animale. Deci entitatea“Diet” este intermediara in vederea stabilirii nutritiei animalelor.

## Capitolul 3 Descrierea constrângerilor utilizate

Constrangerile de orice tip (not null, check, primary key etc) sunt necesarepentru a asigura integritatea si corectitudinea datelor introduse.

- Animals:
  - Primary Key: id_animals
  - Foreign Keys: id_animal_type (animal_type) si id_animal_pen (animal_pen)
- Animal_info:
  - Primary Key: id_animals (animals)
  - Foreign Keys: id_animals (animals)
  - Constrangere Unique: id_animals (animals)
  - Constrangere check:
    - name (regexp_like(Name, '^[a-zA-Z_ ]*$’) -> numele nu trebuiesacontina cifre sau caractere speciale precum acolade etc. Numele poatecontine spatii
    - Gender ( lista de valori ) -> sunt doua tipuri , in acordcu realitatea, de oameni pe acest criteriu
- Animal_pen:
  - Primary Key: id_animal_pen

    - Constrangere Unique: name_pen -> numele tarcului trebuie sa fie unic pentruoidentificare mai usoara a zonelor locuibile si specifice pentru animale si tipurile lor
    - Constrangere check: name_pen ( regexp_like (Name_Pen, '^[a-zA-Z_ ]*$')) ->numele nu trebuie sa contina caractere speciale care sa duca la crearea unui numede tarc irelevant
- Animal_type:
    - Primary Key: id_animal_type
    - Constrangere Unique: type_animal -> exista cateva categorii de animale privite in ansamblul lor, de aceea datele introduse vor fi unice. Daca exista tipul canin, numai e nevoie de o alta definire a sa
    - Constrangere check: type_animal -> lista valori ()
- Medical_info:
    - Primary Key: id_medical_info
    - Constrangere check:Children -> lista de valori() -> raspuns care ajutaladeterminarea posibilelor probleme de sanatate
- Food:
    - Primary Key: id_food
    - Constrangere check: food_type ( regexp_like ( Food_Type, '^[a-zA-Z_ ]*$' )) ->mancarea nu trebuie sa contina numere sau alte caractere speciale
- Diet:
    - Primary Key: id_diet
    - Foreign Keys: id_food (food), id_animals (animals)
    - Constrangere check:
        - diet_type -> lista de valori () -> sunt 3 tipuri care cuprind particularitati care ajuta veterinarul in privintaconsultatiilor
        - quantity ( regexp_like ( Cantity,'[+-]?([0-9]*[.])?[0-9]+')) -> cantitatea trebuie sa contina doar numere. Reprezinta numarul de bucati mancate de animal intr-o zi.

# Capitolul 4. Descrierea logicii stocate

## 4.1 Pachete

Pentru a pastra o abordare unitara si concisa, am creat pachete care implementeaza operatiile CRUD pentru fiecare dintre tabele:

- animal_pen_package:
    - add_animal_pen - adaugă un nou tarc în baza de date. Parametrul p_name_pen reprezintă numele tarțului pe care dorim să-l adăugăm.

Înainte de a efectua operația de inserare în baza de date, se efectuează următoarele verificări: se verifică dacă p_name_pen nu este nul sau gol. Dacă această condiție nu este îndeplinită, se generează o eroare.Se verifică dacă un tarț cu același nume (p_name_pen) nu există deja în baza de date. Dacă există un conflict, se anulează tranzacția și se generează o eroare corespunzătoare.

- delete_animal_pen - șterge un tarc existent din baza de date. Parametrul p_name_pen reprezintă numele tarțului pe care dorim să-l ștergem. Procedura include următoarele verificări: se verifică dacă p_name_pen nu este nul sau gol. Dacă această condiție nu este îndeplinită, se generează o eroare. Se verifică dacă tarțul cu numele specificat există în baza de date. Dacă tarțul nu există, se anulează tranzacția și se generează o eroare corespunzătoare.
- update_animal_pen - actualizează detaliile unui tarț existent în baza de date. Parametrii p_id_animal_pen, p_choice, și p_updated reprezintă ID-ul tarțului, opțiunea aleasă pentru actualizare și valoarea actualizată. Procedura include următoarele verificări: se verifică dacă opțiunea aleasă (p_choice) este validă. În exemplul dat, singura opțiune validă este 'name_pen'. Dacă opțiunea este validă, se verifică dacă noua valoare (p_updated) nu este nulă. Se efectuează actualizarea în baza de date și se verifică dacă actualizarea a avut loc cu succes. În caz contrar, se generează o eroare.
- display_animal_pen - afișează toate tarcurile existente din baza de date. Nu sunt necesare verificări speciale pentru această procedură, deoarece doar extrage datele din tabela animal_pen și le afișează.

- animals_package:
  - add_animal - adaugă un animal nou în baza de date folosind parametrii: ID-ul tipului de animal, ID-ul tarțului, numele, data nașterii și genul animalului. Verifică dacă toți parametrii sunt validați și adaugă înregistrările în tabelele animals și animal_info. Se foloseste un singur pachet pentru operatii CRUD pentru tabelele animals si animal_info.
  - delete_animal - sterge un animal din baza de date folosind ID-ul animalului furnizat. Înainte de ștergere, verifică existența animalului și șterge înregistrările asociate din diet, medical_info, și animal_info.
  - update_animal - actualizează detaliile unui animal existent folosind ID-ul animalului, atributul de actualizat și noua valoare. Verifică validitatea atributelor și efectuează actualizarea în animals și animal_info.
  - display_animal - afișează informații despre animalele din baza de date combinând datele din animals, animal_type, animal_pen, și animal_info.

Fiecare tabel din baza de date are asociat un pachet distinct și specific, implementând logica necesară pentru operațiunile CRUD corespunzătoare. Aceste pachete respectă constrângerile, cheile străine și asigură integritatea datelor în baza de date, contribuind la un sistem integrat și eficient de gestionare a informațiilor.

## 4.2 Triggeri:

Triggerii sunt utilizati pentru monitorizarea și gestionarea automata a informațiilor despre animale. Aceștia sunt folosiți pentru a asigura actualizări corespunzătoare atunci când se adaugă, actualizează sau șterg informații despre animale, diete și registrele medicale. Triggerii sunt necesari pentru a menține consistența datelor și pentru a aplica regulile specifice de gestionare a informațiilor în cadrul grădinii zoologice.

Am folosit urmatorii triggeri:

- animal_info_birth_date_check_trg - declanșat înainte de inserarea sau actualizarea unei înregistrări în tabela **Animal_Info**. Funcționalitatea principală a acestui trigger este de a verifica dacă data de naștere (**birth_date**) furnizată pentru un animal este în viitor comparativ cu data curentă (**SYSDATE**). Dacă data de naștere este în viitor, triggerul va lansa o excepție, prevenind astfel inserția sau actualizarea informațiilor cu o dată de naștere nevalidă.
- animal_info_gender_check_trg - este creat pentru a asigura că valoarea introdusă sau actualizată în coloana "gender" din tabela "Animal_Info" este fie "Male", fie "Female". Dacă valoarea nu este una dintre acestea, triggerul va arunca o eroare. În esență, acest trigger menține integritatea datelor în ceea ce privește genul animalelor înregistrate.
- diet_quantity_check_trg - verifica cantitatea de hrană introdusă sau actualizată în coloana "quantity" din tabela "Diet". Regula specifică verificării este că aceasta cantitate nu poate fi mai mică sau egală cu zero. Dacă valoarea introdusă nu respectă această condiție, triggerul va arunca o eroare .
- food_quantity_check_trg - verifica cantitatea de hrană introdusă sau actualizată în coloana "quantity_wk_food" din tabela "Food". Regula specifică verificării este că aceasta cantitate nu poate fi mai mică sau egală cu zero. Dacă valoarea introdusă nu respectă această condiție, triggerul va arunca o eroare.
- medical_info_date_check_trg - verifica corectitudinea datelor introduse în coloanele "data_registration" și "birth_date" din tabela "Medical_Info". Regula de verificare este că data de înregistrare trebuie să fie ulterioară datei de naștere și să nu fie o dată viitoare față de data curentă (SYSDATE).
- update_food_quantity_trg - actualizeaza cantitatea de hrană disponibilă în tabela "Food" după inserarea sau actualizarea unei înregistrări în tabela "Diet"
    - Obține cantitatea existentă de hrană din tabela "Food" pentru ID-ul de hrană specificat în înregistrarea nouă sau actualizată din "Diet".
    - Calculează diferența dintre noua și vechea cantitate de hrană.

- Verifică dacă cantitatea existentă de hrană este suficientă pentru a acoperi diferența calculată.
- Dacă cantitatea existentă este suficientă, actualizează cantitatea de hrană în tabela "Food" prin scăderea diferenței calculată din cantitatea existentă.
- Dacă nu există o înregistrare cu ID-ul de hrană specificat în tabela "Food", triggerul va arunca o eroare.
- Dacă cantitatea de hrană existentă este insuficientă pentru a acoperi diferența, triggerul va arunca o eroare cu mesajul corespunzător.

## Capitolul 5. Testarea aplicatiei

**Tranzactii**:

Pentru a asigura consistența și fiabilitatea operațiunilor efectuate în baza de date, fiecare procedură utilizează tranzacții. În acest context, tranzacțiile permit gruparea unor instrucțiuni SQL într-o singură unitate logică care poate fi apoi confirmată sau anulată ca un întreg. În cazul în care o eroare apare în timpul execuției unei tranzacții, aceasta poate fi anulată și revertită la un punct de salvare (savepoint) anterior.

De exemplu, procedura add_animal_pen implementeaza o tranzactie astfel:

1. Se stabilește un punct de salvare (**SAVEPOINT**) la începutul tranzacției cu numele **add_animal_pen_savepoint**.
2. Se efectuează verificări pentru validarea datelor.
3. În cazul în care datele sunt valide, se efectuează operația dorită (inserare în cazul acestei proceduri).
4. Dacă apar erori specifice (de exemplu, duplicarea unei valori unice), tranzacția este anulată până la punctul de salvare stabilit anterior și se afișează un mesaj corespunzător.
5. În cazul în care totul decurge fără probleme, tranzacția este confirmată cu **COMMIT**.

Mai mult decat atat, am creat un bloc PL/SQL anonim in care folosesc o tranzactie completa in vederea testarii procedurilor create in pachetele anterioare. Se utilieaza procedura **add_animal** din pachetul **animals_package**, care include un trigger pentru validarea genului animalului. Acest trigger asigură că doar genurile valide sunt introduse în baza de date, respectând astfel regulile de afaceri. De asemenea, această procedură beneficiază de un trigger care verifică cantitatea de hrană disponibilă înainte de adăugare, asigurând că există suficientă hrană pentru consumul animalului. Pe parcursul tranzacției, se montiorizeaza și afiseaza rezultatele așteptate pentru a valida operațiunile efectuate. În final, se confirma și incheie tranzacția folosind

comanda COMMIT, asigurându-ne că toate modificările efectuate în cadrul tranzacției sunt permanent salvate în baza de date.

**Teste:**

În cadrul dezvoltării și validării aplicației, a fost vizata funcționarea corecta a fiecărui trigger și procedură implementată, astfel incat pentru a verifica și valida funcționalitatea acestora, au fost create  diverse teste.

Un exemplu concret de testare este pentru triggerul **animal_info_birth_date_check_trg**. În acest test, s-a încercat actualizara datei de naștere a unui animal la o dată viitoare. În cazul în care triggerul funcționează corect, actualizarea ar trebui să fie respinsă, asigurându-se astfel că datele introduse sunt valide.

De asemenea, fiecare pachet dispune de testele proprii. De exemplu, aufost creat un set de teste pentru procedurile asociate pachetului **diet_package** pentru a valida funcționalitatea corectă a acestora în diferite scenarii: adăugarea unei diete noi, ștergerea unei diete existente și actualizarea unei diete existente.

În testul **test_add_diet_valid**, s-a validat capacitatea procedurii **add_diet** de a adăuga o nouă dietă în baza de date cu parametrii validați, inclusiv tipul de hrană, identificatorul animalului, tipul dietei și cantitatea de hrană.

În testul **test_delete_diet_valid**, s-a verificat corectitudinea funcționării procedurii **delete_diet**, care ar trebui să elimine dieta cu identificatorul specificat din baza de date.

În testul **test_update_diet_valid**, s-a evaluat funcționalitatea procedurii **update_diet**, care ar trebui să actualizeze tipul sau cantitatea dietei pentru un identificator dat.