Application "Zoo"

1 Position du problème

Le directeur d'un Zoo a informatisé la gestion de son établissement. Dans ce Zoo, on trouve des animaux répertoriés par type (lion, léopard, girafe, escargot, ...). Chaque animal possède un nom (Charly, Arthur, Enzo, ...) qui l'identifie de façon unique, un type (ou race), un type de cage (fonction) requis, une date de naissance et un pays d'origine. On retient également les maladies que chaque animal a contractées depuis son arrivée au Zoo, ainsi que le nombre de ses maladies.

Les animaux sont logés dans des cages. Chaque cage peut recevoir un ou plusieurs animaux. Certaines cages peuvent être inoccupées. Une cage correspond à une certaine fonctionnalité et ne permet de ne recevoir que des animaux compatibles. Une cage est identifiée par un numéro, elle est située dans une allée, identifiée aussi par un numéro. Des animaux de types différents ne peuvent pas cohabiter dans une même cage.

Les employées du Zoo entretienent les cages et soignent les animaux. Chaque personne est identifiée par son nom, et on connaît la ville où elle réside. Elle est aussi spécialiste de tel ou tel type de cages. Les personnes sont soit gardien, soit responsable. Les affectations, gardien ou responsable, doivent être compatible avec les spécialités de chacun. Un gardien s'occupe d'une ou de plusieurs cages, et un responsable a la charge de toutes les cages d'une ou de plusieurs allées. Une allée est supervisée par un seul employé et toute cage occupée par au moins un animal est gardée par au moins un gardien.

2 Schémas des relations

Pour modéliser cette application, on a défini le schéma relationnel donné ci-dessous. Les identifiants des relations sont les attributs notés en caractères soulignés :

LesAnimaux (nomA, sexe, type_an, fonction_cage, pays, anNais, noCage, nb_maladies)
{ <*n, s, t, f, p, a, c,nb*> ∈ *LesAnimaux* ⟺ *l'animal de nom n, sexe s,de type t pour une cage f est originaire du pays p. Son année de naissance est a. Il est logé dans la cage de numéro c. il a eu nb maladies* }

LesMaladies (nomA, nomM)
{ <*n, m*> ∈ *LesMaladies* ⟺ *l'animal de nom n a contracté au zoo, la maladie m.* }

LesCages (noCage, fonction, noAllée)
{ <*n, f, a*> ∈ *LesCages* ⟺ *la cage de numéro n est de type f, elle est située dans l'allée a.* }

LesEmployés (nomE, adresse)
{ <*e, a*> ∈ *LesEmployés* ⟺ *l'employé de nom e réside dans la ville a.* }

LesSpecialites (nomE, fonction_cage)
{ <*e, f*> ∈ *LesEmployés* ⟺ *l'employé de nom e est spécialisé pour les cages du type f.* }

LesResponsables (noAllée, nomE)
{ <*a, e*> ∈ *LesResponsables* ⟺ *l'allée de numéro a est sous la responsabilité de l'employé de nom e.* }

LesGardiens (noCage, nomE)
{ <*c, e*> ∈ *LesGardiens* ⟺ *l'employé de nom e est chargé de l'entretien de la cage de numéro c.* }

LesHistoiresAff (noCage, nomE, dateFin)
{ <*c, e, df*> ∈ *LesHistoiresAff* ⟺ *l'employé de nom e a gardé la cage de numéro c jusqu'à la date df.* }

La description des domaines est la suivante :
dom (adresse) = {"Nouméa", "Papeete", "Sartène", ...}

dom (anNais) = [1900, ∞[
dom (fonction, fonction_cage) = { "aquarium géant", "insectes", "fauves", ...}
dom (noAllée, nocage, nbmaladies) = [1, .., 999]
dom (nomA) = {"Charly", "Arthur", "Chloé", ...}
dom (nomE) = {"Adiba", "Calvary", "Jouanot", "Ledru", ...}
dom (nomM) = {"rage de dents", "grippe", "typhus", ...}
dom (pays) = {"Kenya", "Chine", "France", ...}
dom (sexe) = {"femelle", mâle", "hermaphrodite"}
dom (type_an) = {"lion", "léopard", "girafe", "escargot", ...}
dom(dateDebut) = dom (dateFin) = date { *données à la granularité du jour.* }

Les contraintes d'intégrité référentielle sont :
$\pi_{\text{noCage}}(\text{LesGardiens}) \subset \pi_{\text{noCage}}(\text{LesCages})$
$\pi_{\text{nomE}}(\text{LesResponsables}) \subset \pi_{\text{nomE}}(\text{LesEmployés})$
$\pi_{\text{nomE}}(\text{LesGardiens}) \subset \pi_{\text{nomE}}(\text{LesEmployés})$
$\pi_{\text{nomE}}(\text{LesResponsables}) \cap \pi_{\text{nomE}}(\text{LesGardiens}) = \emptyset$
$\pi_{\text{nomE}}(\text{LesResponsables}) \cup \pi_{\text{nomE}}(\text{LesGardiens}) \subset \pi_{\text{nomE}}(\text{LesEmployés})$
$\pi_{\text{noAllée}}(\text{LesAnimaux} \bowtie \text{LesCages}) \subset \pi_{\text{noAllée}}(\text{LesResponsables})$
$\pi_{\text{noCage}}(\text{LesAnimaux}) \subset \pi_{\text{noCage}}(\text{LesGardiens})$
$\pi_{\text{nomA}}(\text{LesMaladies}) \subset \pi_{\text{nomA}}(\text{LesAnimaux})$
$\pi_{\text{nomE}}(\text{LesHistoiresAff}) \subset \pi_{\text{nomE}}(\text{LesEmployés})$
$\pi_{\text{noCage}}(\text{LesHistoiresAff}) \subset \pi_{\text{noCage}}(\text{LesCages})$

La contrainte $\pi_{\text{noCage}}(\text{LesAnimaux}) \subset \pi_{\text{noCage}}(\text{LesCages})$ est implicitement maintenue car :
$\pi_{\text{noCage}}(\text{LesAnimaux}) \subset \pi_{\text{noCage}}(\text{LesGardiens})$ et $\pi_{\text{noCage}}(\text{LesGardiens}) \subset \pi_{\text{noCage}}(\text{LesCages})$
$\Longrightarrow \pi_{\text{noCage}}(\text{LesAnimaux}) \subset \pi_{\text{noCage}}(\text{LesCages})$

3 Mise en place

Vous devez créer la base de données Zoo à l'aide du fichier **script.sql** qui contient à la fois le schéma de la base sous la forme d'un script SQL et quelques données pour initialiser la base sous la forme d'insertion de tuples.
La commande SQLPLUS **start script.sql** permet le chargement et l'exécution de ce script. La suite du TP se décompose en deux parties: la première partie est orientée gestion de contraintes, la seconde partie se focalise sur la définition de fonctionnalités en mode transactionnel au sein d'une application Java, avec prise en compte des contraintes développées dans la partie précédente.

4 Gestion des contraintes

Nous nous intéressons dans cette section à la prise en compte de contraintes métiers qui sont ou pas exprimables directement en SQL dans le schéma de la base.

Question 1 :
Vérifier que les contraintes d'intégrité sont bien effectives :

1. tester l'insertion de "mauvais" tuples.
2. quelle est la contrainte que vérifie le trigger.

3. donner un autre type de contrainte qui devrait être vérifiée à l'aide d'un trigger.

Des jeux de tests pertinents devront illustrer le bon fonctionnement des triggers. Ainsi il vous sera souvent nécessaire d'organiser vos opérations de mise à jour correctement pour tester une contrainte (une cage gardée doit déjà être gardée avant de recevoir des animaux, sinon Erreur !!!).

5 Fonctionnalités et transactions

Dans cette section, le travail consiste à mettre en oeuvre quelques fonctionnalités simplifiées de cette application dans des petits programmes Java utilisant l'API JDBC pour accéder à la base. Vous devrez prendre en compte les exceptions potentielles des triggers développés dans la section précédente. Aucune IHM ou intégration n'est demandée, chaque fonctionnalité ci-après peut être implémentée à l'aide du squelette d'application qui vous est fournie. Chaque fonctionnalité prendra la forme d'une transaction et **vous devrez justifier le mode d'isolation choisi**.

Vous aurez besoin du pilote JDBC sous la forme d'un jar: **ojdbc.jar** et accessoirement de l'API **LectureClavier** pour faciliter les interactions en mode ligne de commande.

Question 2 :
Compléter **squelette_appli** pour implémenter les fonctionnalités/transactions suivantes :

1. Changer la fonction d'une cage. La cage et la fonction sont données par l'utilisateur.
2. Demander un numéro de cage à l'utilisateur et lister les animaux occupant cette cage.
3. Ajouter un nouvel animal : demander les caractéristiques de l'animal à l'utilisateur. Lister l'ensemble des cages compatibles. L'utilisateur choisit parmi ces dernières la cage où l'animal sera logé.
4. Déplacer un animal de cage : le nom de l'animal est donné par l'utilisateur. La nouvelle cage est choisie par l'utilisateur dans la liste des cages compatibles.

Question 3 :
Vous donnerez deux cas d'utilisation pour illustrer la bonne gestion transactionnelle de votre travail et le choix du niveau d'isolation des fonctionnalités mis en oeuvre.