Application "Zoo"

1 Objectifs

Le SGBD doit se charger de faire respecter la plupart des contraintes d'intégrité. Les déclencheurs sont utilisés pour programmer, lorsque c'est possible, les contraintes ne pouvant pas être spécifiées en SQL.

2 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, une 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.

Des personnes sont employées par le Zoo pour entretenir les cages et soigner les animaux. Chaque personne est identifiée par son nom, et on connaît la ville où elle réside. Chaque personne possède aussi un ensemble de spécialités sur tel ou tel type de cages. Les personnes sont affectées à un des deux postes, gardien ou 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 surpervisée par un seul employé et toute cage occupée par au moins un animal est gardée par au moins un gardien ; une cage inoccupée peut être affectée à un et un seul gardien pour entretien. Seule une allée dont au moins une cage contient des animaux peut avoir un responsable (pas de responsable d'une allée vide !).

Le directeur désire gérer de manière automatique les historiques des affectations des gardiens : avant d'enregistrer une affectation, une suppression ou une modification d'affectation d'un gardien à une cage, le système doit garder la trace de l'ancienne affectation. La date de la modification est fournie par le système (fonction SYSDATE).

3 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, 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) = {"lion", "léopard", "girafe", "escargot", ...}
dom(dateDebut) = dom (dateFin) = date { *données à la granularité du jour.* }

Les contraintes d'intégrité référentielle sont :
LesGardiens[noCage] ⊂ LesCages[noCage]
LesResponsables[nomE] ⊂ LesEmployés[nomE]
LesGardiens[nomE] ⊂ LesEmployés[nomE]
LesResponsables[nomE] ∩ LesGardiens[nomE] = ∅
LesResponsables[nomE] ∪ LesGardiens[nomE] ⊂ LesEmployés[nomE]
LesAnimaux * LesCages[noAllée] ⊂ LesResponsables[noAllée]
LesAnimaux[noCage] ⊂ LesGardiens[noCage]
LesMaladies[nomA] ⊂ LesAnimaux[nomA]
LesHistoiresAff[nomE] ⊂ LesEmployés[nomE]
LesHistoiresAff[noCage] ⊂ LesCages[noCage]

La contrainte LesAnimaux[noCage] ⊂ LesCages[noCage] est implicitement maintenue car :
LesAnimaux[noCage] ⊂ LesGardiens[noCage] et LesGardiens[noCage] ⊂ LesCages[noCage]
⟹ LesAnimaux[noCage] ⊂ LesCages[noCage]

4 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 trois parties: la première partie est orientée application avec le développement

de fonctionnalités dans un programme Java, la seconde partie se focalise sur la gestion de contraintes métiers en utilisant les triggers d'Oracle, la dernière partie réunie programme et triggers en s'intéressant à la capture d'exception Oracle au sein d'une application.

5 Transactions & JDBC

Dans cette section, le travail consiste à mettre en oeuvre quelques fonctionnalités très simples dans un programme Java utilisant l'API JDBC pour accéder à la base. Vous ne tiendrez pas compte des contraintes métiers à ce niveau, seul les accès aux données sont considérés. Pour cela vous pourrez utiliser soit la ligne de commande (javac), soit NetBeans, soit encore Eclipse pour développer votre programme. Vous aurez besoin du pilote JDBC sous la forme d'un jar **ojdbc6.jar** et accesoirement de l'API **LectureClavier** pour faciliter la construction d'une IHM très basique en mode ligne de commande. Un squelette d'application **squelette_appli.java** est également disponible pour vous guider dans le développement de cette petite application Java.

Question 1 :
Vous développerez au sein d'une petite application Java les fonctionnalités suivantes :

1. Afficher la liste des animaux
2. Déplacer un animal de cage.
3. Ajouter une maladie à un animal.

6 Gestion des contraintes

Nous intéressons dans cette section à la prise en compte de contraintes métiers qui ne sont pas exprimables directement en SQL dans le schéma de la base mais soit à l'aide de déclencheurs, soit dans l'application elle-même (Java/JDBC). Nous étudierons ici la gestion des contraintes à l'aide des triggers fournis par le SGBD Oracle.

Question 2 :
Vous développerez pour chacune des contraintes suivantes le trigger correspondant :

1. Le calcul du nombre de maladies pour chaque animal doit être automatisé en fonction de l'ajout ou de la suppression des maladies.
2. Des animaux ne peuvent pas être placés dans une cage dont la fonction est incompatible avec ces animaux. On prendra en compte le fait que des animaux peuvent être ajoutés, mais aussi déplacés d'une cage.
3. Des animaux ne peuvent pas être placés dans une cage non gardée. On prendra en compte le fait que des animaux peuvent être ajoutés, mais aussi déplacés d'une cage.
4. Des animaux de type différent ne peuvent pas cohabiter dans une même cage. On prendra en compte le fait que des animaux peuvent être ajoutés, mais aussi déplacés d'une cage.

Des jeux de tests pertinents devront illustrer le bon fonctionnement des triggers .

7 Contraintes & JDBC

Question 3 :
Reprenez les fonctionnalités développeés sous forme de programme Java et modifier le code pour prendre en compte les erreurs d'intégrités renvoyés par le SGBD via vos triggers. Il s'agit ici d'annuler les transactions qui violent certaines contraintes métiers.