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, 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.

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).

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 de fonction 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 possède la fonction 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 :
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]

3 Mise en place

Vous devez créer la base de données Zoo à l'aide du fichier **zoo.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 zoo** permet le chargement et l'exécution de ce script. Vous pouvez aussi simplement executer le script qui se trouve déjà sur le répertoire temporaire du serveur de fichiers (zone fourre-tout partagée): **start /tmp/zoo**. La suite du TP se décompose en deux parties: la première partie se focalise sur la définition de fonctionnalités en mode transactionnel au sein d'une application Java, la seconde partie est orientée gestion de contraintes et son application dans un contexte transactionnel.

4 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. Aucune IHM ou intégration n'est demandée, chaque fonctionnalité ci-après peut être implémentée sous la forme d'un programme Java. Chaque fonctionnalité / programme prendra la forme d'une transaction. Un squelette de programme JAVA pour implémenter les fonctionnalités est disponible: **squelette_appli.java** 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.

Vous pouvez utiliser NetBeans ou Eclipse pour développer vos programmes mais la meilleure solution reste pour ce TP la ligne de commande (dans deux shell séparé) : vous devez exécuter deux programmes en parallèle pour simuler la concurrence et l'exécution de deux programmes sous Eclipse est rédibitoire (et peu confortable) dans le contexte. En ligne de commande vous devrez entrez la commande **javac -c .:./ojdbc6.jar squelette_appli.java** pour compiler et la commande **java -cp .:./ojdbc6.jar squelette_appli** pour exécuter le programme. Vous pouvez aussi modifier la variable d'environnement **CLASSPATH** pour simplifier les commandes précédentes : **setenv CLASSPATH $CLASSPATH:.:./ojdbc6.jar**

1. Changer la fonction d'une cage. La cage et la fonction sont données par l'utilisateur.
2. Modifier l'affectation d'un gardien. Le nom du gardien est donné par l'utilisateur. La liste des cages auxquelles il est affecté s'affiche. L'utilisateur choisit la cage où il sera retiré. Les cages compatibles avec les spécialités du gardien sont listées, puis l'utilisateur choisit la cage de destination.
3. Afficher des statistiques. Cette fonction affiche deux tableaux: le nombre de gardien par cage et le nombre de cage par gardien.

Ces fonctionnalités devront respecter (dans le code Java) les contraintes énoncées ci-dessous :

1. Un gardien ne peut pas être retiré de la surveillance d'une cage si les animaux qu'elle contient se retrouvent non gardés.
2. Un seul et unique gardien peut garder une cage vide (pour entretien).

Question 1 :
Vous donnerez plusieurs cas d'utilisation de transactions en concurrence pour illustrer l'impact du niveau d'isolation choisi. Pour cela vous montrerez les résultats obtenus en fonction du choix de différents niveaux d'isolation d'une transaction cible dans un même ordonnancement. Des exemples de transactions en concurrence: Changer la fonction d'une cage avec le changement d'affectation d'un gardien, Plusieurs modifications d'un même gardien en concurrence, Afficher les statistiques pendant le changement d'affectation d'un gardien, etc. Votre code Java devra bien entendu intégrer des pauses pour permettre d'interfolier les opérations de différentes transactions à convenance, et simuler ainsi l'odonnancement désiré. L'objectif ultime est de trouver une solution pour éviter les incohérences. Vous avez à votre disposition les niveaux d'isolations READ COMMITTED & SERIALIZABLE, ainsi que la clause FOR UPDATE.

5 Gestion des contraintes

Nous nous intéressons dans cette section à la gestion de contraintes métiers au sein du SGBD à l'aide de déclencheurs (triggers).

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

1. Lorsqu'un gardien voit l'une de ses affectations modifée, son ancienne affectation doit être conservée dans la table LesHistoiresAff.
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. Un gardien ne peut pas être retiré de la surveillance d'une cage si les animaux qu'elle contient se retrouvent non gardés. On prendra en compte le fait que des gardiens peuvent être retirés, mais aussi affectés à une autre cage.

Dans un premier temps, pendant le TP, vous testerez ces triggers avec des instructions SQL basiques. Dans un second temps, hors TP sous forme d'un DM, vous testerez ces triggers en reprenant la question précédente mais en modifiant le code java des différentes fonctionnalités: la gestion des contraintes est réalisée par des triggers et plus dans l'application qui devra alors réagir aux exceptions Oracle liées à la violation de ces contraintes.

Des jeux de tests pertinents devront être produits.