TP : Billetterie en ligne d'un théâtre

1 Introduction

Ce TP vise à initier la construction une application Java de gestion simplifiée des réservations pour un théâtre. Le travail sera poursuit pendant le projet IBD. Cette application permet, par exemple, de consulter les spectacles à l'affiche et le programme des représentations associées, d'effectuer des réservations, d'ajouter des représentations, etc. Dans le cadre du TP, les interactions seront gérées par le biais des méthodes fournies dans la librairie jus.util.jar inclus dans l'archive distribué. La documentation se trouve dans le placard dans Javanaise/Jus/Doc.

2 Position du problème

Un directeur de théâtre désire informatiser son système de réservation de places pour les spectacles qui se déroulent dans son théâtre au cours d'une même saison (de septembre à juin). A la fin d'une saison, le contenu de la base de données est archivé et vidé. Pour les besoins du projet les données de l'application décrites ci-après ont été simplifiées.

Les spectacles Un spectacle est identifié par un numéro et on connaît son nom. Un spectacle fait généralement l'objet de plusieurs représentations proposées à des moments différents. Une représentation débute à un moment donné exprimé à la granularité de l'heure (par exemple 28/11/2007 20H).

La salle La salle est partitionnée en zones numérotées, regroupant chacune un ensemble de places. Une place est identifiée par un numéro de rang, et un numéro de place dans le rang. Une zone est associée à une catégorie tarifaire (orchestre, balcon, poulailler, etc). Toutes les places de la même zone sont dans la même catégorie. Le tarif de base associé à chaque catégorie est fixé pour l'ensemble de tous les spectacles.

Les ventes Chaque place vendue fait l'objet de l'émission d'un ticket identifié par un numéro de série et estampillé par la date au moment de la transaction (instant à la granularité de la seconde). Un achat peut concerner plusieurs places. Chaque achat se traduit par la création d'un dossier (identifié par un numéro) auquel est associé le prix global des places.

Hypothèses supplémentaires Le théâtre conserve toujours 70 places (toutes catégories confondues) qui seront vendues au guichet, dans l'heure qui précède le début du spectacle.

3 Modélisation relationnelle

Le schéma relationnel est donné ci-dessous. Les clefs candidates sont formées des attributs notés en caractères soulignés :

- LesZones (<u>numZ</u>, categorie)
  $\{< z, c >\in$ LesZones $\Leftrightarrow$ les places de la zone z sont dans la catégorie c. $\}$

- LesCategories (nomC, prix)
  $\{< c, p >\in$ LesCategories $\Leftrightarrow p$ est le prix des places se situant dans une zone de catégorie c. $\}$
- LesPlaces (noPlace, noRang, numZ)
  $\{< p, r, z >\in$ LesPlaces $\Leftrightarrow$ la place de numéro p dans le rang r est dans la zone z. $\}$
- LesSpectacles (numS, nomS)
  $\{< s, t >\in$ LesSpectacles $\Leftrightarrow$ le spectacle de numéro s a pour titre t. $\}$
- LesRepresentations (numS, dateRep)
  $\{< s, d >\in$ LesReprésentations $\Leftrightarrow d$ est la date d'une représentation pour le spectacle s. $\}$
- LesTickets (noSerie, numS, dateRep, noPlace, noRang, dateEmission)
  $\{< t, s, d, p, r, e >\in$ LesTickets $\Leftrightarrow$ le ticket dont le numéro de série est t correspondant la place $< p, r >$ pour la représentation $< s, d >$, a été émis à la date e, et $e < d$. $\}$

domaine(dateRep) = date(heure) { par ex. 24/11/2007 20H }
domaine(dateEmission) = date (seconde) { par ex. 24/11/2007 9H30MN14S }
domaine(catégorie) = {'orchestre', 'balcon', 'poulailler', ...}
domaine(numZ) = domaine(noPlace) = domaine(noRang) = domaine(numS)
= domaine(noDossier) = entier > 0
domaine(prix) = réels > 0

Les autres contraintes d'intégrité :

- Pour chaque représentation, il y a 70 places disponibles jusqu'à $t - 1$ heure (t est l'instant de début de la représentation).
- Dans la relation LesTickets : dateEmission < dateRep
- $\pi_{\text{catégorie}}(\text{LesZones}) \subset \pi_{\text{catégorie}}(\text{LesCategories})$
- $\pi_{\text{numZ}}(\text{LesPlaces}) \subset \pi_{\text{numZ}}(\text{LesZones})$
- $\pi_{\text{numS}}(\text{LesReprésentations}) \subset \pi_{\text{numS}}(\text{LesSpectacles})$
- $\pi_{\text{numS, dateRep}}(\text{LesTickets}) \subset \pi_{\text{numS, dateRep}}(\text{LesReprésentations})$
- $\pi_{\text{noPlace, noRang}}(\text{LesTickets}) \subset \pi_{\text{noPlace, noRang}}(\text{LesPlaces})$
- $\pi_{\text{noDossier}}(\text{LesTickets}) \subset \pi_{\text{noDossier}}(\text{LesDossiers})$

3.1 Question 1

1. Télécharger à partir du placard électronique de l'UE le script relations.sql pour créer la base de données.
2. Vérifier que les contraintes d'intégrité sont prises en compte. Ajouter des contraintes si besoin.
3. Peupler la base de données avec le script donnees.sql.

3.2 Question 2

1. Télécharger l'archive contenant le squelette de l'application à compléter.
2. Compiler et tester.

Astuce: *Cette archive contient un projet Eclipse (eclipse est disponible sur le serveur imasrv-java). Alternativement (par exemple sur jpp) on peut utiliser les commandes suivantes à partir du répertoire TP1:*

```
javac -d bin -classpath lib/jus.util.jar:lib/ojdbc14.jar -sourcepath src \
  > src/*/*.java
java -classpath lib/jus.util.jar:lib/ojdbc14.jar:bin application.Theatre
```

3.3 Question 3

Compléter l'application par la mise en œuvre des fonctions ci-dessous. En cas d'erreur, l'application doit annuler l'opération fautive, le cas échéant la réessayer, et ensuite revenir à un état normal.

1. Ajouter une catégorie : ajoute une catégorie avec son prix.
2. Modifier une catégorie : l'application doit proposer de modifier son prix ou son nom ou les deux.
3. Consulter les données de la base : fournit le résultat de l'évaluation d'une requête d'interrogation quelconque saisie par l'utilisateur.
4. Réaliser des émissions "groupées" de tickets (par exemple, un client veut 10 tickets à l'orchestre et 5 au poulailler).
5. Ajouter un spectacle. Pour chaque représentation de ce spectacle on réserve à des fin promotionnelles un certain nombre de tickets (à l'orchestre) destinés à des VIP invitées.

On veillera à appliquer de bonnes disciplines de programmation:

1. Chaque erreur doit produire un message compréhensible par l'utilisateur de l'application (échec de connexion, pilote manquant, mauvais mot de passe, entrées fausses etc.).
2. Utiliser le chaînage des exceptions pour faire remonter les exceptions des sous-fonctions (comme getConnexion) vers les fonctions de haut niveau qui gèrent les interactions avec l'utilisateur (création de messages d'erreur etc.).
3. Toujours fermer les connections (objets ResultSet, Statement, Connection) même après une erreur.