Université de Sherbrooke
Faculté des sciences
Département d'informatique

IFT 287 – Exploitation de bases de données relationnelles et orientées objet
Travail pratique 3
L'approche client-serveur avec JDBC et Java

Gestionnaire des transactions

La problématique

Dans le premier TP vous avez développé un ensemble de requêtes SQL pour permettre le traitement d'une base de données relationnelle. Dans le deuxième TP vous avez développé un programme Java qui traitait avec un fichier texte comme entrepôt de données. Dans ce troisième TP nous allons utiliser la base de données du premier TP mais utilisée à partir d'un programme écrit en langage Java. Vous devez utiliser JDBC pour établir la connexion à la base de données et pouvoir faire l'exploitation des données.

Le travail à faire consiste en écrire un programme qui permet d'effectuer les transactions sur la base de données de la ligue de baseball utilisé lors du premier TP.

Votre programme doit lire une commande et ses paramètres associés. Les paramètres sont délimités par des espaces. Une transaction se tient toujours sur une seule ligne. Voici une liste des opérations que votre programme doit implémenter:

1. `creerEquipe <EquipeNom> [<NomTterrain> AdresseTerrain]`
Créer une nouvelle équipe. L'équipe est identifiée de manière unique par son `EquipeId`. Cette valeur devra être calculée par le programme. L'utilisateur doit fournir l'`<EquipeNom>`. Le nom de l'équipe est unique dans la ligue, il faut le valider. `<NomTterrain>` indique le nom du terrain dans lequel l'équipe joue ses match comme équipe locale, si le terrain n'existe pas, il faut l'ajouter à la base de données.

2. `afficherEquipes`
Afficher la liste des équipes. Il faut montrer l'`EquipeId` ainsi que l'`EquipeNom`. La liste doit être triée par `EquipeNom`.

3. `supprimerEquipe <EquipeNom>`
Supprimer l'équipe `< EquipeNom >`. L'équipe ne doit pas avoir de joueurs, sinon la transaction est refusée. Il faut avertir l'utilisateur avec un message d'erreur approprié. Valider aussi si `l'EquipeNom` n'existe pas.

4. `creerJoueur <JoueurNom> <JoueurPrenom> [<EquipeNom> <Numero> [<DateDebut>]]`
Créer un nouveau joueur, le programme doit calculer le `JoueurId`, et l'utilisateur doit fournir le `<JoueurNom>` et le `<JoueurPrenom>` . De manière optionnelle on peut donner les informations pour le joindre à une équipe. Si l'`<EquipeNom >` est donné il faut fournir le numéro du joueur dans cette équipe `<Numero>`. De manière optionnelle, on peut donner la `<DateDebut>`. Si elle n'est pas donnée, on peut prendre la date de la transaction (la date du système).

5. `afficherJoueursEquipe [<EquipeNom >]`
Afficher la liste de joueurs. Si le paramètre `< EquipeNom >` est fourni, le programme affiche seulement les joueurs de l'équipe correspondante. Si non, afficher tous les joueurs de toutes les équipes indiquant le nom de l'équipe.

6. `supprimerJoueur <JoueurNom> <JoueurPrenom>`
Supprimer le joueur et toute l'information stockée sur lui. S'il y a quelques données sur le joueur il faut les afficher et demander la confirmation de l'utilisateur avant procéder à la suppression. Valider aussi si les données du jouer à supprimer n'existent pas.

7. `creerMatch <MatchDate> <MatchHeure> <EquipeNomLocal> <EquipeNomVisiteur>`
Ajouter un match, en calculant le `MatchId` de manière automatique. Il faut vérifier que les équipes existent et qu'ils sont différents, une équipe ne peut pas jouer contre lui-même! Il faut vérifier aussi la date et l'heure. Le terrain doit être assigné à celui de l'équipe locale.

8. `creerArbitre <ArbitreNom> <ArbitrePrenom>`
Créer un nouvel arbitre en calculant de manière automatique `l'ArbitreId`. Assurez-vous de ne pas répéter les noms des arbitres, on suppose que dans la ligue il n'y a pas d'homonymes.

9. `afficherArbitres`
Afficher la liste des arbitres en ordre alphabétique d'`ArbitreNom`

10. `arbitrerMatch <MatchDate> <MatchHeure> <EquipeNomLocal> <EquipeNomVisiteur> <ArbitreNom> <ArbitrePrenom>`
Affecter des arbitres à un match. Valider que le match existe, ainsi que les `<ArbitreNom> <ArbitrePrenom>`. Un match peut avoir un maximum de quatre arbitres, il faut les compter.

11. `entrerResultatMatch <MatchDate> <MatchHeure> <EquipeNomLocal> <EquipeNomVisiteur> <PointsLocal> <PointsVisiteur>`
Entrer le résultat d'un match. Valider que la valeur utilisée pour les points soit toujours plus grande ou égale à zéro.

12. `afficherResultatsDate [<APartirDate>]`
Afficher les résultats de tous les matchs. Si le paramètre `<APartirDate>` est donné, il faut afficher seulement les résultats à partir de cette date. Afficher aussi les arbitres, s'ils existent pour le match. Les résultats doivent être triés par date.

13. `afficherResultats [<EquipeNom>]`
Afficher les résultats des matchs où l'équipe `<EquipeNom>` a participé, peu importe si c'était comme local ou comme visiteur. Afficher aussi les arbitres, s'ils existent pour le match. Les résultats doivent être triés par date.

Votre programme doit faire les validations nécessaires et afficher des messages d'erreur significatifs, c'est-à-dire, compréhensibles par l'utilisateur. Vous pouvez commencer vos messages par un niveau de message, par exemple : `USERWARNING – Paramètre manquant, application des paramètres par défaut`, ou bien `USERERREUR – Erreur de format, action non réalisée`.
En cas d'erreur fatale, le message affiché peut-être plus technique par exemple `SYSTEMERROR – Problème de connexion à la base de données.` Toutefois, vous pouvez supposer que tous les paramètres fournis seront du bon type, selon l'attribut de la table.

Tout le calcul des identifiants (clés primaires) peut être fait de deux façons :

1. Avant de faire une insertion à votre table, chercher la valeur maximale stockée dans la colonne clé. Cette manière de calculer fonctionne si on peut supposer qu'il y a peu d'utilisateurs concurrents.
2. Une deuxième manière est de créer une table auxiliaire nommée `Sequence` avec deux colonnes : `NomTable`, `NextCle`. Chaque tuple de cette table est associé à une table de votre base de données. Avant de faire une insertion dans une table qui demande l'utilisation d'une clé, il; faut faire une requête sur la table `Sequence` pour chercher la valeur de la clé à utiliser. Pour finir le

processus, après l'insertion il faut faire une mise à jour de la valeur de la colonne `NextCle` avec la valeur de la prochaine clé à utiliser. Une méthode un peu plus longue, mais plus efficace pour les utilisateurs concurrents.

Pour peupler votre base de données, vous pouvez utiliser le fichier `tp1_data.sql`, dans lequel les noms composés par deux string ont été changés, par exemple pour le nom du terrain Yankee Stadium a été changé pour Yankee_Stadium ou le nom des Bleu Jay a été changé par Blue_Jays.

Pour tester votre programme, vous pouvez utiliser le fichier `testTP3.dat`

Structure du projet sur Eclipse

Il faut spécifier certains paramètres de configuration sous Eclipse :
Nom du projet : `tp3`
Nom du package : `ligueBaseball`
Nom de la classe qui contienne la méthode main : `Main`

De cette façon, pour pouvoir exécuter votre programme il faudra l'appeler comme suit :

```
java ligueBaseball.Main <userId> <motDePasse> <baseDeDonnees> [<fichier-entree>]
```

Le paramètre `baseDeDonnees` indique le nom de la base de données à utiliser.
Pour établir votre connexion à la base de données vous devez utiliser la chaîne de connexion suivante :

```
conn = DriverManager.getConnection ("jdbc :postgresql :"+baseDeDonnees, userId, motDePasse);
```

Le paramètre `<fichier-entrée>` est un fichier texte qui contient les transactions à exécuter. Si cet argument n'est pas présent, votre programme doit lire les transactions entrées de manière interactive au clavier. L'exemple de la bibliothèque est structuré exactement de cette façon. Utilisez cet exemple pour construire votre tp3.

Remise

La date prévue pour la remise de ce troisième TP est indiquée dans le plan cours. Le travail se fait en équipes de quatre personnes maximum.
Vous pouvez utiliser la même procédure que pour le tp1. Remettre les fichiers suivants :

Il faut créer un répertoire nommé tp3 qui contient tous les fichiers à remettre :

- `tp3/src/ligueBaseball/Main.java` (le classe qui contient la méthode main)
- `tp3/src/ligueBaseball/`… (Toutes les autres classes utilisées, si vous avez choisi de créer plus d'une classe.)
- `tp3/doc/JavaDoc` (la documentation en format JavaDoc de votre programme)
- `tp3/doc/12345678.pdf` (Un fichier par membre de l'équipe avec une petite réflexion sur la façon de développer le programme. La distribution des tâches, la participation des membres, l'apprentissage, etc. 12345678 est la matricule de l'étudiant.)
- `tp3/doc/diagramme/classes.pdf` (Le diagramme de classes de votre système)
- Le fichier `etudiants.xml` avec le nom et les matricules des membres de l'équipe.
- Ce n'est pas nécessaire de soumettre vos classes compilées (les fichiers .class).

Utilisez la commande turnin pour remettre votre travail.

```
turnin –c ift287 –p ift287TP3 tp3
```

Conseils sur les étapes à suivre pour développer votre programme

Le programme à développer n'est pas énorme, mais n'est pas trivial non plus. Il faut penser à développer de manière ordonnée et viser la qualité dès le départ de la conception de votre système.

Voici quelques conseils qui peuvent aider à mieux organiser le travail :

1. Identifier un chef d'équipe. Son rôle sera de coordonner le travail et de s'assurer que les travaux progressent comme prévu.
2. Planifier. Vous pouvez faire une phase de conception et ensuite diviser les tâches entre les membres de l'équipe pour assurer que tout le travail sera réalisé. Il faut toujours valider que les échéanciers soient respectés.
3. Définir toutes les classes et toutes les méthodes que votre programme a besoin avant de commencer à coder, vous pouvez utiliser la méthode des cartes CRC.
4. Définir des cas de test.
5. Coder et tester les classes et les fonctionnalités une par une.
6. Tester chaque instruction au moins une fois avant la remise.
7. Ne réinventer pas la roue. Inspirez-vous de l'exemple du système de gestion de bibliothèque vu en classe. Réutiliser autant de code que possible.

Évaluation

Le barème de correction pour ce troisième travail pratique est :

Caractéristique	Points
Exactitude des résultats : • Les bons messages d'erreurs sont émis • Les mises à jour sont correctement effectuées	30
Documentation et lisibilité (Voir la norme de programmation)	20
Architecture : • Correcte implémentation de l'architecture traitée en classe, ou si vous avez utilisé une nouvelle architecture elle doit porter des avantages. • Chaque classe doit avoir une fonction bien définie. • Chaque méthode a une fonction bien définie.	35
Utilisation efficace des ressources Allocation d'objets minimale (préserver la simplicité au niveau de la conception) Fermeture des ressources après son utilisation	10
Réflexion personnelle de tous les membres de l'équipe	5
TOTAL	100 points