Stockage et traitement des données - TP5

Grâce à sqlite, nous pouvons travailler sur une base de données en lecture et en écriture. Cette base constitue la source de données pour le programme. Pour ce TP, nous reprenons la base bibliothèque du TP3. Nous allons consulter et modifier les données de cette base à partir de Java sous Eclipse comme dans le TP4.

Pour créer le nouveau projet, nous allons utiliser l'outil d'importation d'Eclipse :

- récupérez le fichier S7_TP5.tar.gz et sauvez-le sur votre compte
- allez dans File → Import → General → Existing Projects into Workspace
- sélectionnez "select archive file" et cliquez sur "Browse"
- sélectionnez le fichier `S7_TP5.tar.gz`

Un nouveau projet S7_TP5 doit apparaître dans votre espace de travail. Il comprend une seule classe : `Principal`, ainsi que les fichiers `creat_biblio.sql` (contenant le schéma de base et les valeurs) ainsi que `biblio.db` contenant les données codées au format sqlite. Le fichier `sqlitejdbc-v056.jar`, qui definit l'interface sqlite pour jdbc, fait egalement partie du projet.

Exécutez le programme principal et vérifiez que tout marche bien.

1. Requêtes

Testez la requête permettant d'afficher tous les livres de la base et n'afficher que l'auteur et le titre de chaque livre. Reprenez au choix une des requêtes 6, 7, 8 ou 9 du TP3 et affichez son résultat.

Attention. Pensez à lancer vos requêtes à l'aide d'un `try..catch` pour pouvoir lire les erreurs sans interrompre le programme

```
    try{
        r = m.executeQuery("SELECT ... ");
        }
    catch(Exception e){
        System.out.println("Probleme dans la requete :" + e.getMessage());
        }
```

(Si vous ne vous rappelez plus comment lire le résultat d'une requête, relisez le TP4)

2. Classes Membre

Nous allons maintenant définir une classe `Membre` (correspondant à la table Membre de la base) contenant les attributs de la table : `idMembre`, `nomMembre`, `adrMembre` et `cpMembre`. Nous ajouterons :

- un attribut emprunts servant à stocker les identifiants des livres empruntés par ce membre sous la forme d'une liste (de type chaînes de caractères) :

```
  private ArrayList<String> emprunts;
```

Commençons par définir quelques méthodes simples :

- Générez un constructeur (qui initialise les quatre premiers attributs avec des arguments fournis en paramètre et initialise emprunts avec une liste vide)
- Générez une méthode `toString`.
- Générez les getters et les setters (sauf pour l'attribut emprunts).
- Écrire la méthode emprunte qui ajoute un livre dans la liste des emprunts.

Testez cette classe dans le programme principal en créant un nouveau membre (dont la liste des emprunts est vide). Testez la méthode emprunte en empruntant le Dernier pharaon (de code `"7089PQIU"`), puis affichez le membre.

3. Lecture de la base

Nous souhaitons maintenant récupérer les informations concernant les membres stockés dans la base.

- Lancez dans le programme principal la requête qui permet de récupérer les informations sur le membre nommé Monet (d'identifiant 15), créez un objet membre correspondant, et affichez-le.
- Lancez dans le programme principal la requête permettant d'obtenir la liste des livres empruntés par Monet, ajoutez-les dans le champ emprunts du membre (grâce à la méthode emprunte) et affichez le membre.

4. Mise à jour

Les opérations de mise à jour (`INSERT`, `UPDATE`,...) s'effectuent avec la fonction `executeUpdate` (et non pas `executeQuery`) appliquées à un objet de type `Statement` ou `PreparedStatement`. La méthode ne retourne pas de `ResultSet`

INSERT

- Ajoutez à la table Membre le nouveau membre que vous avez créé à la question 2

remarque: pour copier les valeurs dans le texte de la requête, utilisez de préférence un `PreparedStatement` plutôt qu'un `Statement`.

Un `preparedSatement` est une requête contenant des "trous" à compléter.

```
PreparedStatement prep = c.prepareStatement("INSERT INTO Membre
VALUES (?,?,?,?)");
```

Pour entrer les valeurs dans la requête, on utilise `setInt` ou `setString` en précisant la position du trou à compléter (ici 1,2,3 ou 4)

Exemple :

```
prep.setInt(1,112);
prep.setString(2,”Durand”);
```

Pour exécuter la mise à jour, nous utiliserons executeUpdate (sans oublier le try..catch)

```
  try{
      prep.executeUpdate();
      }
  catch(Exception e){
       System.out.println("Probleme dans la mise à jour :" +
e.getMessage());
      }
```

Voir : jdbc

UPDATE

- Monet change d’adresse : il n’habite plus à Aubagne mais à Gardanne (code postal 13120). Utilisez un setter pour modifier l’adresse de Monet puis mettez à jour la base de données avec un `executeUpdate`, puis vérifiez que la base a bien été mise à jour.

5. Classe AccesseurMembre

Vous avez remarqué que ces différentes opérations sont assez “lourdes” à écrire. Il est préférable de définir une classe qui regroupe les opérations les plus courantes. Cette classe peut obéir aux spécifications ”DAO” (Data Access Object), voir: le-pattern-dao-1

Ici, un accesseur de Membre est un objet permettant de réaliser des opérations de lecture, d’écriture et de mise à jour sur des membres de la base, et plus généralement de synchroniser les informations contenues dans les objets avec les informations contenues dans les tables de la base.

La classe `AccesseurMembre` contient un seul attribut :

```
  private Connection connexion;
```

- Définissez le constructeur (prenant 1 unique paramètre) et testez-le en créant un `AccesseurMembre` dans le programme principal.
- Ajoutez une fonction `getMembreById` qui prend un numéro de membre et retourne le Membre correspondant à partir des informations lues dans la base. Une première requête permet de récupérer les informations issues de la table Membre. Une seconde requête doit récupérer la liste des livres empruntés par le membre. Que fait la fonction si le numéro n’est pas présent dans la base?

Remarque : il est nécessaire de “prendre en compte” les erreurs SQL :

```
public Membre getMembreById(int IdM) throws ClassNotFoundException,SQLException{
    ...
    }
```

Testez cette fonction dans le programme principal en créant un nouveau membre à partir de l'identifiant 30.

- Ajoutez une fonction `createMembre` qui ajoute un nouveau membre dans la base.

```
public void createMembre(Membre m) throws ClassNotFoundException,SQLException{
    ...
    }
```

Testez cette méthode dans le programme principal.

- Ajoutez une fonction `updateMembre` qui met à jour le membre dans la base. Que fait cette fonction si le membre n'est pas présent dans la base?

```
public void updateMembre(Membre m) throws ClassNotFoundException,SQLException{
    ...
    }
```

Testez cette méthode dans le programme principal.