TP JDBC & Transactions

L'objectif de ce TP est double :

- d'une part, appréhender l'interface JDBC qui permet à une application Java d'interagir avec un SGBD relationnel
- d'autre part, compléter les notions déjà acquises en ce qui concerne la gestion des transactions au sein du SGBD Oracle

Partie 1 : Accès à une BD à travers JDBC

JDBC (*Java DataBase Connectivity*) est un ensemble de classes et d'interfaces permettant de réaliser des connexions vers des bases de données et d'effectuer des requêtes.

L'API JDBC est représentée par le package **java.sql** dont les principales interfaces sont les suivantes:

- **java.sql.DriverManager** : Elle gère le chargement des pilotes et permet de créer de nouvelles connexions à des bases de données. Elle tient à jour la liste principale des pilotes JDBC recensés du système.
- **java.sql.Connection** : Elle représente une connexion à une base de données.
- **java.sql.Statement :** C'est une classe que l'application emploie pour transmettre des instructions à la base, elle représente une requête SQL. La fermeture d'un *Statement* engendre l'invalidation automatique des tous les *ResultSet* associés.
- **java.sql.ResultSet** : Cette classe symbolise un ensemble de tuples correspondant au résultat d'une requête d'interrogation. L'application se déplace séquentiellement dans l'ensemble des tuples du résultat.

Il existe un pilote spécifique pour chaque SGBD (Oracle, MySQL, ...). Chaque produit a un pilote JDBC adapté.

La suite de cette partie décrit la méthode d'accès à une BD via JDBC:

1) Chargement du pilote JDBC pour le SGBD,
2) Connexion à la base (identique à une session sqlplus, mais sans la console),
3) Construction et soumission d'une requête au SGBD,
4) Exploitation des résultats renvoyés par le SGBD,
5) Libération des ressources et fermeture de la connexion.

Importation des Classes JDBC

```
import java.sql.*;
```

Enregistrement du pilote

```
// Enregistrement du driver Oracle pour pouvoir accéder à la base via JDBC
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

Connexion à une base de données

```
// Etablissement de la connection
Connection conn = DriverManager.getConnection(CONN_URL, USER, PASSWD);
```

`CONN_URL` est la chaine de connexion :
`"jdbc:oracle:thin:@hopper.e.ujf-grenoble.fr:1521:ufrima"`
`USER` est le login de l'utilisateur et `PASSWD` son mot de passe (identiques à ceux utilisés pour une session sqlplus).

Requêtes

On commence par créer un descripteur de requête sur la BD :

1. On crée un objet de type `Statement`,
2. On récupère le descripteur de requête en invoquant la méthode `createStatement()` sur le descripteur de la BD

   Ex : `Statement requete = conn.createStatement();`

   Ensuite, les méthodes `executeQuery(String)` et `executeUpdate(String)` appelées sur le descripteur de requête permet d'exécuter la requête SQL passée en paramètre. `executeQuery` concerne les requêtes d'interrogation de la base et `executeUpdate` est utilisée pour les requêtes qui modifient le contenu de la base.

   Ex : `ResultSet resultat = requete.executeQuery("select * from Participant'");`

Il est également possible de prépare une requête à l'avance (requête paramétrée) à l'aide d'un objet de type `PreparedStatement`

```
// Creation de la requete
PreparedStatement stmt = conn.prepareStatement(PRE_STMT);
```

où `PRE_STMT` à été initialisée avec une requête SQL, par exemple:

```
String PRE_STMT = "select * from Participant";

// Execution de la requete
ResultSet rset = stmt.executeQuery();

// Gestion des résultats...

//  Fin de la requête ; liberation des ressources
rset.close();
stmt.close();
```

Comment gérer des paramètres dans une requête ?

1. Utiliser le caractère `?` pour repérer un paramètre,

2. Invoquer la méthode `setXXX` (`setString`, `setInt`, etc.) sur le descripteur de requête en précisant le numéro du paramètre à remplacer et sa valeur.

Ex: On désire paramétrer une requête:

```
PreparedStatement stmt = conn.prepareStatement("select * from thetable
where age>? And nom=?");
stmt.setInt(1, Integer.parseInt(in.readLine());
stmt.setString(2,in.readLine());
```

`Integer.parseInt(uneChaine)` permet de transformer une chaîne (représentant un entier) en un entier.

Gestion des résultats d'une requête

La méthode `executeQuery()` retourne les réponses dans un objet de type `ResultSet`. Les méthodes `next()` et `previous()` appelées sur un objet de type `ResultSet` permettent de parcourir les instances (réponses) de la requête.

1. `next()` passe à l'instance suivante, `previous()` passe à l'instance précédente,
2. `next()` ou `previous()` renvoie false s'il n'y a plus de tuples à lire.

La méthode `getString(String)` retourne la valeur du champ (passé en paramètre) d'un objet `ResultSet` (instance courante) sous la forme d'un objet de type `String` pour Java. Une méthode identique `getString(integer)` existe, le paramètre représente ici le numéro de la colonne.

Ex :
```
while(resultat.next()) {
System.out.println("Nom = " + resultat.getString("Nom")
+ ", Prenom = " + resultat.getString("Prenom")
+ ", Travail = " + resultat.getString(4)); }
```

D'autres fonctions getXXX() existent, consulter la documentation de l'API.

Attention : Un objet ResultSet est implicitement associé à l'objet Statement correspondant. Il ne faut donc pas réutiliser un objet Statement pour effectuer une nouvelle requête si l'on a encore besoin du ResultSet de la requête précédente.

Déconnexion

```
// Fermeture de la connexion
conn.close();
```

Traitement des exceptions

Le code Java JDBC doit récupérer les exceptions :

```
try {…} catch (SQLException e) {
            System.err.println("failed");
            System.out.println("Affichage de la pile d'erreur");
            e.printStackTrace(System.err);
            System.out.println("Affichage du message d'erreur");
            System.out.println(e.getMessage());
            System.out.println("Affichage du code d'erreur");
            System.out.println(e.getErrorCode()); }
```

Partie 2 : Travaux pratiques

On considère la même application de gestion de comptes bancaires que dans le TP n°1 (et la même relation `COMPTES`). Vous créerez une classe Banque en Java pour simuler une application bancaire.
La classe `LectureClavier` est disponible pour simplifier les saisies au clavier.
Les programmes Java seront exécutés sur hopper pour simplifier la gestion de JDBC.

a) Mise en place de la partie fonctionnelle du programme

1. Créez la relation `COMPTES` sous sqlplus si ce n'est pas déjà fait, et insérez-y au moins un tuple.
2. Ajoutez dans le squelette de la classe Banque les opérations `selection`, `debit`, `credit` et `insert`. `selection` affiche le contenu de la relation COMPTES, `credit` (respectivement `debit`) augmente (diminue) le solde d'un compte du montant `m` (laissé à votre choix). `insert` correspond à l'insertion d'un tuple de votre choix.

b) Gestion transactionnelle

JDBC offre les mêmes fonctionnalités que la console sqlplus. Les méthodes `setAutoCommit`(true/false), `commit`() et `rollback`() sont définies pour la classe `Connection`.

Ouvrez deux programmes `BANQUE` qui réaliseront des opérations différentes, on a donc deux utilisateurs que l'on nommera `A` et `B`. Dans la suite, nous noterons `operationI` l'exécution de l'opération `operation` par l'utilisateur `i`. `setAutoCommit` est placé à false.

1. Effectuez les opérations suivantes sur le même compte bancaire :
   `selectionA`, `selectionB`, `debitB`, `selectionA`, `creditB`, `selectionA`
   Que constatez-vous ? Annulez ensuite les deux transactions précédentes.

2. Introduisons maintenant des points de validation :
   `selectionA, selectionB, debitB, commitB, selectionA, creditB, commitB, selectionA`
   Que constatez-vous ?

3. Exécutez maintenant les opérations suivantes :
   `selectionA, selectionB, debitB, creditA, selectionA, selectionB`
   Que constatez-vous ? Annulez ensuite les deux transactions précédentes.

4. Exécutez les opérations suivantes :
   `selectionA, debitA, selectionB, commitA, selectionB`
   Que constatez-vous ?

5. Que déduisez-vous de ces expériences concernant :
   - le niveau d'isolation par défaut d'Oracle ?
   - l'utilisation éventuelle de verrous pour les opérations d'écriture ?
   - l'utilisation éventuelle de verrous pour les opérations de lecture ?

c) Configuration des paramètres transactionnels via JDBC

La classe `Connection` possède plusieurs méthodes pour manipuler le niveau d'isolation des transactions :

- `public abstract int getTransactionIsolation()` Retourne le niveau d'isolation courant dont la valeur est explicite en utilisant les constantes suivantes (`TRANSACTION_NONE` `TRANSACTION_READ_COMMITTED`, `TRANSACTION_SERIALIZABLE`, `TRANSACTION_READ_UNCOMMITTED`, `TRANSACTION_REPEATABLE_READ`). Remarque : seuls les niveaux `READ_COMMITED` et `SERIALIZABLE` sont gérés par Oracle.
- `public abstract void setTransactionIsolation(int)` Place le niveau d'isolation sur l'une des valeurs listées ci-dessus.
- `public abstract boolean isReadOnly()` Retourne vrai si la transaction est en mode `READ ONLY` ; retourne faux si le mode est `READ WRITE`.

Commencez par vérifier que vos conclusions précédentes sur le niveau d'isolation par défaut étaient correctes.

Exécutez les opérations suivantes : `selectionA`, `insertB`, `commitB`, `debitA`, `selectionA` en utilisant le mode d'isolation `SERIALIZABLE`.
Que constatez-vous ?

Exécutez ensuite les opérations suivantes (toujours en mode `SERIALIZABLE`) : `selectionA`, `selectionB`, `debitB`, `commitB`, `debitA`, `commitA`
Que constatez-vous ?

Qu'en concluez-vous sur les mécanismes employés par Oracle pour l'ordonnancement des transactions en mode `SERIALIZABLE` ?

d) Gestion des erreurs

Dans certaines situations, une transaction peut rencontrer une erreur levée par le SGBD. Avec Oracle, cela se produit notamment dans les deux cas suivants :

- la détection d'un interblocage lors de la demande d'obtention d'un verrou (code d'erreur Oracle 60) ;
- en mode `SERIALIZABLE`, la détection d'un problème de sérialisation lors de la demande de validation d'une transaction (code d'erreur Oracle 8177).

Le programmeur d'application doit prévoir ce type de problèmes. Il faut, au minimum, annuler la transaction en cours.

Il est possible d'appeler la méthode `getErrorCode` sur une exception levée par le SGBD (`SQLException`) afin de déterminer la cause précise du problème.

Modifiez le code de votre application pour prendre en charge les cas d'erreur mentionnés ci-dessus et proposez des séquences de test pour en vérifier le bon fonctionnement.