M1 MIAGE FC/FA – 2010/2011

Bases de Données Avancées

octobre 2010

TP - JDBC

Première partie

Nous allons tester les fonctionnalités des `ResultSet` modifiables, et la gestion des transactions. Pour cela, nous utiliserons l'exemple des comptes bancaires largement exploité en cours. Nous allons donc considérer une table `COMPTE` :

```
create table compte(
id_compte VARCHAR2(20) constraint compte_pkey primary key,
solde NUMBER default 0.0 not null,
constraint solde_positif check (solde >= 0)
);
```

Nous allons écrire une petite application java qui permet de réaliser des virements bancaires. Nous testerons le comportement de cette application par rapport à la concurrence : vous lancerez donc plusieurs exécutions simultanées et essayerez de minimiser les problèmes posés par la concurrence. Je rappelle que dans un virement bancaire, il faut que les deux comptes soient modifiés, ou bien aucun des deux.
Cette application peut se décomposer de la façon suivante :

- Une classe `Lecture` qui permet de faire des saisies clavier. (cette classe est fournie)
- La classe `Base` écrite lors du TP précédent.
- Une classe `Compte` qui représente 1 compte bancaire. Cette classe possède un `ResultSet` contenant une seule ligne lue dans la table `Compte`
- Une classe `AppliCompte`, classe principale qui permet de réaliser des virements entre deux comptes.

Question 1 : Complétez la classe `Base` afin

1. de pouvoir créer des `ResultSet` qui n'ont pas le comportement par défaut (lecture seule et 1 seul sens de parcours).
2. de pouvoir valider et annuler la transaction courante.

Question 2 : Ecrivez la classe `Compte`. Chaque objet `Compte` possède un `ResultSet` qui contient la ligne qui lui correspond dans la table `compte`. Cette classe dispose des méthodes suivantes :

- double getSolde()
- String getIdCompte()
- void setSolde(double s)

ATTENTION sous Oracle, il faut écrire `select T.* from maTable T where ...` *pour que le ResultSet soit considéré comme modifiable.*

Question 3 : Ecrivez la classe `AppliCompte` qui permet de réaliser un virement d'une somme s entre deux comptes $c1$ et $c2$. Les valeurs de s, $c1$, $c2$ seront lues au clavier. Les lectures permettent également de marquer des pauses dans le déroulement du programme. A l'exécution de votre application, essayez de faire apparaître/disparaître des incohérences dues à la concurrence.

Question 4 : Pour voir l'évolution de deux `ResultSet` sur les mêmes lignes, vous pouvez refaire l'exercice avec un ResultSet par compte qui possède les lignes de tous les comptes (pas seulement la ligne le concernant). Vous afficherez à chaque modification d'un compte le contenu du `ResultSet` correspondant.

Seconde partie

Nous allons étudier maintenant l'accès à la Métabase via JDBC. Les informations sur la Métabase sont obtenues grâce à un objet instance de `DatabaseMetaData`, fourni par la méthode `getMetaData()` invoquée sur l'objet de type `Connection`.
Question 5 : On veut connaître certaines informations générales sur la base. Définissez une méthode `generalites` qui affiche :

- Le nom et la version du produit (SGBD) interrogé. (méthode `getDatabaseProductVersion()`)
- le niveau d'isolement des transactions. (méthode `getDefaultTransactionIsolation()`)
- le nom de l'utilisateur connecté (méthode `getUserName()`)

Question 6 : Ecrire une méthode `listeTables` qui affiche la liste des tables, vues, synonymes. Utiliser la méthode `getTables`. Pour les paramètres de cette méthode, le catalogue vaut `null` et le schéma est votre nom d'utilisateur oracle en majuscules.

Question 7 : Définir une méthode `desc`, équivalent de la commande SQLPlus de même nom, qui affiche pour une table donnée la liste des colonnes avec leur types, et si elles acceptent les valeurs `NULL`. On y ajoutera des informations sur les clefs primaires et les clefs étrangères qui concernent cette table. Voici l'affichage que l'on veut obtenir pour une relation, par exemple la table `VILLE` utilisée au TP précédent :

```
REF_VILLE NUMBER 22 NOT NULL PK
NOM VARCHAR2 30 NOT NULL
NB_HABITANTS NUMBER 22 NOT NULL
REF_PAYS NUMBER 22 NOT NULL
VILLE_PAYS_FKEY : REF_PAYS->PAYS.REF_PAYS
```

Pour cette question, utiliser les méthodes `getColumns`, `getPrimaryKeys` et `getImportedKeys`.

Question 8 : Modifier la méthode `listeTables` afin qu'elle affiche des précisions sur chaque relation, grâce à la méthode `desc`

Question 9 : Donner la liste des procédures. Utiliser la méthode `getProcedures` (la première colonne du ResultSet obtenu permet d'obtenir le nom du paquetage si c'est une procédure ou fonction faisant partie d'un paquetage). Pour simplifier, on n'affichera pas leur signature complète mais c'est faisable grâce à la méthode `getProcedureColumns`.

Question 10 : Ecrire une méthode `affichageSelect` qui permet d'afficher de façon formatée le résultat d'une requête. Cette méthode prend en paramètre un `ResultSet` On utilisera la méthode `getMetaData` de `ResultSet`, ainsi que les méthodes `getColumnCount`, `getColumnDisplaySize` et `getColumnLabel` de `ResultSetMetaData`. Pour tester cette méthode, on peut lire la requête sur la ligne de commande, ou bien utiliser la classe `Lecture` du TP précédent.