Introduction

Ce cours illustre les fonctionnalités de base du API JDBC (Java Database Connectivity). Nous allons discuter de comment créer des tables, insérer des valeurs, faire de requêtes sur les tables, mettre à jours de enregistrements, préparer des instructions, réaliser des transactions et intercepter des exceptions et des erreurs.

Les interfaces de type utilisateur tel que JDBC sont des interfaces de programmation permettant l'accès de façon externe aux bases de données SQL et d'opérer des commandes de mises à jour. Elles permettent l'intégration du langage SQL dans les langages de programmation tout en fournissant des librairies d'API pour l'interfaçage avec les DB. De façon plus spécifique, Java JDBC possède une riche collection de routines qui font d'elle une interface extrêmement simple et intuitive.

Voici en quelques étapes une représentation (flow) de comment on une API comme JDBC: Vous écrivez un programme Java normal. Quelque part dans le programme, vous avez besoin d'interagir avec une base de données. En utilisant des routines de la bibliothèque standard, vous ouvrez une connexion à la base de données. Vous utilisez ensuite JDBC pour envoyer votre code SQL à la base de données, et vous traitez les résultats renvoyés. A la fin, vous fermez la connexion.

Etablir une connexion

Avant d'accéder à une base de données avec JDBC, il faut ouvrir une connexion entre le programme et le server de base de données. Ceci se fait en deux étapes :

- Charger le pilote du type de base de données avec lequel on veut interagir : Le API JDBC a été conçu pour être indépendant du type de base de données.

  Class.forName("oracle.jdbc.driver.OracleDriver")

- Etablir la connexion

  Une fois que le pilote a été chargé, on peut créer une instance de connexion:

Connection con=DriverManager.getConnection

(

"`jdbc:oracle:thin:@192.168.1.1:1521:COURS`*", username, passwd*

)

L'instance de connexion créée permettra d'envoyer des requêtes SQL à la base de données.

Création d'une instruction JDBC

Une instruction JDBC est un objet utilisé pour envoyer des instructions SQL à la base de données, elle n'est pas à confondre avec une instruction SQL. Un objet instruction JDBC est associé à une connexion ouverte, et non pas à une seule instruction SQL. On peut assimiler une instruction JDBC à un canal permettant de faire passer une ou plusieurs requêtes SQL.

On a besoin d'une connexion active pour créer un objet de type *Statement :*

Statement stmt = con.createStatement() ;

L'objet *Statement* existe, mais aucune requête SQL n'y est encore associée.

Création d'une instance JDBC PreparedStatement

Parfois, il est plus commode ou plus efficace d'utiliser un objet de *PreparedStatement* pour envoyer des instructions SQL au SGBD. La principale caractéristique qui le distingue de la déclaration de la classe mère, est que, contrairement à la classe *Statement* qui envoie directement l'instruction SQL au SGBD, l'instruction de type *PreparedStatement* est pré-compilée avant d'être envoyée au SGBD.

L'avantage est que si vous avez besoin d'utiliser la même requête avec des paramètres différents à plusieurs reprises, l'instruction peut être compilée et optimisée juste une fois. Cela

contraste avec l'utilisation d'une déclaration normale où chaque utilisation de la même instruction SQL nécessite une nouvelle compilation. L'extrait suivant montre comment créer une instruction SQL paramétrée avec trois paramètres d'entrée:

```
PreparedStatement  prepareMAJ = con.prepareStatement
("UPDATE Ventes SET prix = ? WHERE bar = ? AND biere = ?" );
```

Avant d'exécuter un objet de type *PreparedStatement*, on doit attribuer des valeurs aux paramètres. Ceci peut se faire en appelant des méthodes de type setXXX définies dans la classe *PreparedStatement*. Les méthodes les plus utilisées sont setInt, setFloat, setDouble, setString etc.

```
prepareMAJ.setInt(1, 3);

prepareMAJ.setString(2, "Bar de Foo");

prepareMAJ.setString(3, "Guiness");
```

Exécuter les instructions CREATE/INSERT/UPDATE

L'exécution des instructions SQL avec JDBC varie en fonction de l'objectif ' de l'instruction SQL. Les instructions DDL telles que la création de table et la modification, ainsi que des mises à jour du contenu de la table, sont tous exécutés avec la méthode *executeUpdate*. Notez que ces instructions modifient l'état des données de la base de données, d'où le nom de la méthode contient « Update ».

Exemple d'instruction exécutant la méthode *executeUpdate*.

```
Statement stmt = con.createStatement();

stmt.executeUpdate("CREATE TABLE Ventes " +

        "(bar VARCHAR(40), biere VARCHAR(40), prix REAL)" );

stmt.executeUpdate("INSERT INTO Ventes " +

        "VALUES ('Bar de Foo', Guiness, 2.00)" );
```

String sqlString = "CREATE TABLE Bars " +

"(nom VARCHAR(40), adresse VARCHAR(80), licence INT)" ;

stmt.executeUpdate(sqlString);

Lorsqu'on utilise *executeUpdate* pour exécuter des instructions DDL, la valeur de retour est toujours zéro, alors que les exécutions d'instruction de modification de données renvoient une valeur supérieure ou égale à zéro, valeur correspondant au nombre de tuples affectés dans par l'opération.

Pendant qu'on utilise un *PreparedStatement*, nous devons assigner des valeurs aux différents paramètres d'appeler la méthode *executeUpdate*.

int n = prepareMAJ.executeUpdate ();

Exécution d'une instruction de sélection

Contrairement aux instructions de la section précédente, une requête devrait retourner un ensemble de tuples comme résultat, et elle ne pas changer l'état de la base de données. Il existe une méthode appelée executeQuery, qui renvoie les résultats sous forme d'objet ResultSet:

String bar, biere ;

float price ;

*ResultSet rs = stmt.executeQuery("SELECT * FROM Ventes");*

while (rs.next()) {

bar = rs.getString("bar");

beer = rs.getString("biere");

price = rs.getFloat("prix");

System.out.println(bar + " ventes " + biere + " pour un " + prix + " FCFA.");

}

On peut aussi spécifier le numéro de la colonne au lieu de mettre le nom de la colonne.

bar = rs.getString(1);

price = rs.getFloat(3);

beer = rs.getString(2);

Quelques exemples de méthodes de la classe ResultSet

La classe *ResultSet* possède plusieurs methodes qui offrent une meilleure gestion de la collection des résultats. Au nombre de celles-ci nous avons : *getRow, isFirst, isBeforeFirst, isLast, isAfterLast.*

rs.absolute(3); // Aller vers le troisième résultat

rs.previous(); // Retourner d'un enregistrement

rs.relative(2); // Avancer de deux enregistrements

rs.relative(-3); // Retourner de deux enregistrements

Transactions

Gestion des Exceptions