TP JDBC

1 Introduction

JDBC est une API (*Application Interface Programming*) JAVA dont vous trouverez la documentation à l'adresse :

`http://java.sun.com/j2se/6/docs/api/`

La programmation d'une application JAVA utilisant JDBC pour se connecter à une base de données va nécessiter l'utilisation de classes implémentant les interfaces suivantes :

Driver	Renvoie une connexion utilisée par le DriverManager
Connection	Connexion à une base
Statement	traitement d'une requête SQL
ResultSet	résultat d'un SELECT vers une table
ResultSetMetaData	description des lignes de ce SELECT
DatabaseMataData	informations du dictionnaire des données.

Chaque connexion à un SGBD se fait à l'aide du driver approprié qui est téléchargeable sur le site de l'éditeur du SGBD. Pour Oracle, ce dernier correspond à la classe : `oracle.jdbc.driver.OracleDriver`.

Une fois le driver chargé (par exemple avec un `Class.forName`), il faut ouvrir la connexion en instanciant un objet de connexion par :

```
Connection cnx = DriverManager.getConnection(url, uid, pswd);
```

où le paramètre `url` est une chaîne de la forme : *jdbc :sous-protocole :base-de-données*, `uid` étant le nom de l'utilisateur et `pswd` son mot de passe `oracle`

Pour le serveur`Oracle` du FIL , la valeur de `url` sera :

`jdbc:oracle:thin:@orval.fil.univ-lille1.fr:1521:filora10gr2`;

Le driver est `oracle-thin` (sous-protocole), le port oracle est `1521` et `filora10gr2` est le nom de la base sur le serveur `orval` du FIL.

2 Tester la connexion

2.1 Prise en main de JDeveloper

Vous allez utiliser l'outil de développement Java sur Oracle qui est `JDeveloper`. Celui-ci est accessible à partir d'un raccourci placé sur votre bureau et nommé :`jdevW1013`, qui fonctionne avec la version 1.6 du `jdk` implanté dans le répertoire `java`. (Il existe un autre raccourci pointant sur une version ancienne à ne pas utiliser).

Commencez, en utilisant le Poste de Travail ou l'explorateur de Windows, par créer, par exemple sur votre bureau, un répertoire `tpjdbc` contenant un sous-répertoire `Project1`, lui-même contenant un sous répertoire `src`. C'est dans ce dernier répertoire que vous implanterez le source de votre application.

Il vous faut ensuite, sous `JDeveloper`, créer un projet dans un espace de travail. Pour cela, allez dans l'onglet `Applications` situé en haut à gauche ; sélectionnez `Applications` puis, avec un *clic droit* : `new Application`. Le nom par défaut de votre application est `Application1`.

Choisissez pour être son répertoire le réperoire `tpjdbc` que vous venez de créer sur le bureau. En suivant les fenêtres de dialogue, vous allez créer un projet nommé `Project1` placé dans ce répertoire.

Sous `JDeveloper`, sélectionnez `Project1` dans la fenêtre de gauche, puis, avec un *clic-droit*, choisissez `Add to project content` et ajoutez le répertoire `src`.

Avant de pouvoir utiliser le pilote JDBC d'oracle, il vous faudra l'ajouter au `CLASSPATH` du projet. A cet effet, sélectionnez le projet, faites un *clic-droit*, choisissez `Project Properties`, puis la rubrique `Librairies`. Appuyez sur le bouton `Add jar/directory` et suivez les instructions pour ajouter le chemin du driver :

`C:\Oracle\product\10.2.0\client_1\jdbc\lib\ojdbc14.jar`

Votre environnement de travail est maintenant configuré.

2.2 Premier test

Commencez à télécharger le fichier suivant en le rangeant dans le répertoire `src` du projet :

`http://www.fil.univ-lille1.fr/~marti/bd/tp/0910/TestJDBC.java`

Modifier le contenu pour l'adapter à vos besoins : vos login et mot de passe, les paramètres de connexion et la requête d'interrogation.

Observez comment ce programme doit fonctionner : après création d'un objet de connexion, il instancie une requête (`stmt` de la classe implémentant l'interface `Statement`). La méthode `executeQuery` récupère le résultat de l'interrogation dans un objet `rs` de la "classe" `ResultSet`. L'exploration des données à afficher se fait par la méthode `next` qui manipule un curseur de façon transparante pour l'utilisateur.

Les *meta-données* (données issues du dictionnaire des données) associées à la table interrogée permettent de retrouver le nom et le type de ses colonnes (et aussi beaucoup d'autres informations). Elles sont exploitées dans ce programme par l'objet `rsmd` par l'intermédiaire des méthodes `getColumnCount()` , `getColumnName()` et `getColumnType()`

En effectuant un *clic-droit* sur le nom du projet, effectuez une compilation (`make`), puis, après corrections d'éventuelles erreurs, selectionnez de la même manière une exécution (`run`).

Le résultat de l'interrogation doit s'afficher dans la sous-fenêtre d'éxécution située en bas de la fenêtre de `JDeveloper`.

3 Application à écrire

Vous devez écrire, tester et rendre à l'issu du TP une application java permettant de construire l'export d'une table de la base (au choix). Ce peut être aussi bien l'une de vos propres tables ou une table sur laquelle vous disposez de droits de lecture.

Cet export prendra la forme d'un script SQL comportant aussi bien le `create table` que la série des `insert into` permettant l'insertion de ses données. Ce script devra comporter la définition de la clé primaire qui est présente dans les *meta-données*.

Voici les informations d'ordre techniques pour y parvenir :

1. L'accès aux *meta-données* se fait par l'intermédiaire de la classe `DataBaseMetaData`. Soit `dmd` un objet de cette classe, obtenu par la méthode `getMetaData()` de la la classe implémentant l'interface `Connection`.

2. L'invocation de `dmd.getTables` va fournir la liste des tables sous la forme d'un objet de type `ResultSet`.

Les paramètres de cette méthode sont : une chaîne correspondant au nom du dictionnaire (défini par la méthode `getCatalog()` de la "classe" `Connection`), une chaîne correspondant au schéma de la base (ne pas définir car il n'y en a pas), une chaîne précisant le modèle de nom des objets sur lesquels on veut des information (utilisant la syntaxe de l'opérateur SQL `like`) et enfin un tableau de chaînes précisant leur nature.

En déclarant par exemple : `String les_types={"TABLE","VIEW"}`, on limitera l'interrogation à ces deux type d'objets. On pourra alors écrire :

```
ResultSet les_tables=dmd.getTables(cnx.getCatalog(), null, "%",les_types)
```

L'objet `les_tables` permet l'exploitation d'une relation contenant les colonnes : `table_name` et `table_type` qui contiennent les noms et types (table ou vue) des objets de la métabase.

3. La méthode `getColumns` de la classe `DataBaseMetaData` permet de récupérer toutes les informations utiles sur les colonnes d'une table. Par exemple :

```
ResultSet les_colonnes=dmd.getColumns(cnx.getCatalog(), null, nomTable_ou_Vue,"%");
```

construit un objet encapsulant une relation disposant des colonnes : `column_name`, `data_type`, `column_size` et `is_nullable` qui précise si la colonne peut ou non contenir des valeurs nulles.

4. Enfin, la méthode `getPrimaryKeys` de `DataBaseMetaData` permet de récupérer la clé primaire de la table. Exemple :

```
ResultSet cle_primaire=dmd.getPrimaryKeys(cnx.getCatalog(), null, nomTable);
```

L'objet résultant encapsule une table ayant une colonne `column_name`, contenant le ou les noms des attributs composant la clé primaire, si elle existe (elle peut ne pas avoir été déclarée).

Il ne doit pas être trop difficile de réaliser l'application demandée. De là à construire un script correspondant à un export de toutes les tables présentes dans la base, il n'y a qu'un petit pas à franchir ... si vous avez le temps.