Base de donnée

Programmation avancée – Java

F5 – ISIMA 2020/2021

Olivier Goutet
o.goutet@openium.fr

10 novembre 2020

Plan

JDBC (Java DataBase connectivity)

Présentation

- JDBC (Java Data Base Connectivity) est l'API de base pour l'accès à des bases de données relationnelles avec le langage SQL, depuis un programme en Java
- Il est fourni par le paquetage `java.sql`

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

Interfaces principales

- `Driver` : renvoie une instance de Connection
- `Connection` : connexion à une base
- `Statement` : ordre SQL
- `PreparedStatement` : ordre SQL paramétré
- `CallableStatement` : procédure stockée sur le SGBD
- `ResultSet` : lignes récupérées par un ordre SELECT
- `ResultSetMetaData` : description des lignes récupérées par un SELECT
- `DatabaseMetaData` : informations sur la BDD

Classes principales

- `DriverManager` : gère les drivers, lance les connexions aux bases
- `Date` : date SQL
- `Time` : heures, minutes, secondes SQL
- `TimeStamp` : date et heure avec une précision à la microseconde

Mise en oeuvre de JDBC

1. Importer le package java.sql
2. Enregistrer le driver JDBC
3. Établir la connexion au SGBD
4. Créer une requête (ou instruction SQL)
5. Exécuter la requête
6. Traiter les données retournées
7. Fermer la connexion

Chargement du Driver

- Pour se connecter à une base de données via ODBC, il faut tout d'abord charger le pilote JDBC-ODBC qui fait le lien entre les deux.

```
Class.forName(nomDudriver);
Class.forName(com.mysql.jdbc.Driver);
```

Connexion

```
String DBurl = "jdbc:odbc:testDB";
Connection con = DriverManager.getConnection(DBurl,
                                             "myLogin",
                                             "myPassword");
```

Exemple

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

public class Bdd {
  public static void main(String[] argv) {
    try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn = DriverManager.getConnection
      ("jdbc:mysql://localhost/dbname","user","mdp");
      ...
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Accès à la base de données

- Ordre SQL
  - `DatabaseMetaData` : informations à propos de la base de données : nom des tables, index, version ...
  - `ResultSet` : résultat d'une requête et information sur une table. L'accès se fait enregistrement par enregistrement.
  - `ResultSetMetaData` : informations sur les colonnes (nom et type) d'un ResultSet

Exécution requêtes SQL

- Les requêtes d'interrogation SQL sont exécutées avec les méthodes d'un objet Statement que l'on obtient à partir d'un objet Connection

```
ResultSet resultats = null;
String requete = "SELECT␣*␣FROM␣client";
try {
  Statement stmt = conn.createStatement();
  resultats = stmt.executeQuery(requete);
} catch (SQLException e) {
  //traitement de l'exception
}
```

ResultSet

- Au début, `ResultSet` est positionné avant la premiere ligne
- avancer en appelant la méthode `next()`
- Quand ResultSet est positionné sur une ligne les méthodes `getXXX` permettent de récupérer les valeurs des colonnes de la ligne
  - `getXXX(int numeroColonne)`
  - `getXXX(String nomColonne)`

```
while(rs.next()) {
  // Traitement de chaque ligne
}
```

Exemple

```
Statement st = connexion.createStatement();
ResultSet rs = st.executeQuery("SELECT␣a,b,c,␣FROM␣Table1");
while(rs.next()) {
  int i = rs.getInt("a");
  String s = rs.getString(2);
  byte[] b = rs.getBytes("c");
}
```

SQL <–> java

CHAR, VARCHAR	getString()
LONGVARCHAR	getAsciiStream()
NUMERIC, DECIMAL	getBigDecimal()
BINARY, VARBINARY	getBytes()
LONGVARBINARY	getBinaryStream()
BIT	getBoolean()
INTEGER	getInt()
BIGINT	getLong()
SMALLINT	getShort()
TINYINT	getByte()
REAL	getFloat()
DOUBLE, FLOAT	getDouble()
DATE	getDate()
TIME	getTime()
TIMESTAMP	getTimeStamp()

Mise à jour : UPDATE

- La méthode executeUpdate() retourne le nombre d'enregistrements qui ont été mis à jour

```
//insertion d'un enregistrement dans la table client
requete ="INSERT INTO client VALUES (3,client 3,prenom 3)";
try {
   Statement stmt = con.createStatement();
   int nbMaj = stmt.executeUpdate(requete);
   affiche("nb mise a jour = "+nbMaj);
} catch (SQLException e) { ... }
```

Valeurs NULL

- Pour repérer les valeurs NULL de la base
  - utiliser la méthode wasNull() de ResultSet
    - renvoie true si l'on vient de lire un NULL, false sinon
- les méthodes `getXXX()` de `ResultSet` convertissent une valeur NULL SQL en une valeur acceptable par le type d'objet demandé
  - String -> NULL java
  - Numérique : 0
  - Boolean : False

Exemple NULL

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT␣nomE,comm␣FROM␣emp");
while (rs.next()) {
  nom = rs.getString(1);
  commission = rs.getDouble(2);
  if (rs.wasNull()){
    System.out.println(nom + ":␣pas␣de␣comm");
  } else {
    System.out.println(nom + "␣a␣"+ commission
                       + "euros␣de␣commission");
  }
}
```

Fermer les connexions

- Pour terminer proprement un traitement, il faut fermer les différents espaces ouverts
  - sinon le garbage collector s'en occupera mais moins efficace
- Chaque objet possède une méthode close()

```
resultset.close();
statement.close();
connexion.close();
```

Création d'un Statement

- L'interface `Statement` possède les méthodes nécessaires pour réaliser les requêtes sur la base associée à la connexion dont il dépend
- 3 types de Statement
  - `Statement` : requêtes statiques simples
  - `PreparedStatement` : requêtes dynamiques précompilées (avec paramètres d'entrée/sortie)
  - `CallableStatement` : procédures stockées

Requêtes pré-compilées

- L'objet `PreparedStatement` envoie une requête sans paramètres à la base de données pour précompilation et spécifiera le moment voulu la valeur des paramètres
- plus rapide qu'un Statement classique
  - le SGBD n'analyse qu'une seule fois la requête
  - pour de nombreuses exécutions d'une même requête SQL avec des paramètres variables
- tous les SGBD n'acceptent pas les requêtes précompilées

Création d'une requête pré-compilée

- La méthode `prepareStatement()` de l'objet `Connection` crée un `PreparedStatement`

```
PreparedStatement ps =
c.prepareStatement("SELECT␣*␣FROM␣Clients␣"+
  "WHERE␣name␣=␣?␣");
```

  - les paramètres sont spécifiés par un " ?"
  - ils sont ensuite instanciés par les méthodes `setInt()`, `setString()`, `setDate()`...
  - ces méthodes nécessitent 2 arguments (`setInt(n, valeur)`)

Exemple

```
PreparedStatement ps =
c.prepareStatement("UPDATE␣emp␣SET␣sal␣=␣?␣"
                   + "WHERE␣name␣=␣?");
...

for(int i = 0; i < 10; i++) {
  ps.setFloat(1, salary[i]);
  ps.setString(2, name[i]);
  int count = ps.executeUpdate();
}
```

Accès aux méta-données

- JDBC permet de récupérer des informations
  - sur le type de données que l'on vient de récupérer par un `SELECT` (interface `ResultSetMetaData`)
  - mais aussi sur la base elle-même (interface `DatabaseMetaData`)

ResultSetMetaData

- Méthode `getMetaData()` permet d'obtenir des informations sur les types de données du `ResultSet`
- Elle renvoie des instances de `ResultSetMetaData`
- On peut connaître entre autres
  - le nombre de colonne : `getColumnCount()`
  - le nom d'une colonne : `getColumnName(int col)`
  - le nom de la table : `getTableName(int col)`
  - si un NULL SQL peut être stocké dans une colonne : `isNullable()`

Exemple ResultSetMetaData

```
ResultSet rs = st.executeQuery("SELECT␣*␣FROM␣emp");
ResultSetMetaData rsmd = rs.getMetaData();
int nbColonnes = rsmd.getColumnCount();
for (int i = 1; i <= nbColonnes; i++) {
  String typeColonne = rsmd.getColumnTypeName(i);
  String nomColonne= rsmd.getColumnName(i);
  System.out.println("Colonne␣" + i
  + "␣de␣nom␣" + nomColonne
  + "␣de␣type␣" + typeColonne);
}
```

DatabaseMetaData

- informations sur la base de données
- méthode `getMetaData()` de l'objet `Connection`
- elle renvoie des instances de `DatabaseMetaData`
- on peut connaître entre autres
  - les tables de la base : `getTables()`
  - les colonnes de la table : `getString()`
  - ...

Exemple DatabaseMetaData

```
private DatabaseMetaData metaData;
private java.awt.List listTables= new List(10);
...
String[] types = { "TABLE", "VIEW" };
String nomTables;
...
metaData = conn.getMetaData();
ResultSet rs =
metaData.getTables(null, null, "\%", types);
while (rs.next()) {
  nomTable = rs.getString(3);
  listTables.add(nomTable);
}
```

Synthèse sur JDBC

- Interface pour un accès homogène
  - le concept de Driver masque au maximum les différences des SGBD
  - API de bas niveau : il faut connaître SQL
- Tous les éditeurs proposent un driver JDBC