

ENSM-SE

Axe Ingénierie des Systèmes Informatiques
Ecole Nationale Supérieure des Mines de Saint-Etienne

Connexions Java/BDD : JDBC
Développement de Systèmes Informatiques

© 2009, Laurent Vercoeur

Ecole Nationale Supérieure des Mines SAINT-ETIENNE

ENSM-SE

Java Database Connectivity
package (JDBC)

© 2009, Laurent Vercoeur

Ecole Nationale Supérieure des Mines SAINT-ETIENNE

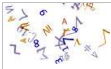
ENSM-SE

Introduction à JDBC

- JDBC est un package contenant des classes et interfaces permettant d'accéder à une base de données à partir d'un programme Java.
- L'API JDBC est contenu dans le package java.sql :
 - ♦ connexion à une base de données
 - ♦ expression de requêtes relationnelles en SQL
 - ♦ encapsulation des résultats des requêtes

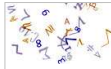
© 2009, Laurent Vercoeur

Ecole Nationale Supérieure des Mines SAINT-ETIENNE



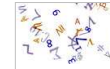
Drivers JDBC

- L'hétérogénéité des bases de données empêche une connexion directe entre une base de données et une application Java.
- Il faut utiliser un pilote qui "traduit" des commandes Java pour une base de données :
 - ♦ Chaque pilote est propre à une base de donnée
 - ♦ Un pilote implémente l'interface `java.sql.Driver`



`java.sql.DriverManager`

- La classe `java.sql.DriverManager` encapsule un ensemble de méthodes statiques pour :
 - ♦ enregistrer la liste des pilotes disponibles;
 - ♦ établir une connexion en trouvant le pilote approprié;
 - ♦ gérer des fichiers de log.
- L'utilisation de drivers est (presque) transparente.



Connexion à une DB

- L'interface `java.sql.Connection` est implémentée par des classes représentant une connexion à une base de données.
- L'objet connexion est ensuite utilisé pour toutes les interactions avec la BD :
 - ♦ accéder à des informations sur la base;
 - ♦ créer puis exécuter des requêtes;
 - ♦ récupérer les résultats des requêtes.

ENSM-SE

Ouverture d'une connexion


Une connexion est ouverte par la classe DriverManager :

```
Connection getConnection(String url);
```

L'URL JDBC a un format spécifique :

```
jdbc:<sub-protocol>:<sub-name>
```


- ♦ <sub-protocol> est généralement le nom du pilote;
- ♦ <sub-name> est le nom de la source de donnée (interprété par le pilote).

© 2009, Laurent Vercouter 

ENSM-SE

Exemple : connexion à une base Oracle

```
...
try {
    Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
    String url = "jdbc:oracle:thin:@educ.emse.fr:1521:DBEM";
    String user = "laurent";
    String passwd = "test";
    Connection c = DriverManager.getConnection(url,user,passwd);
    ...
} catch (Exception e) {
    e.printStackTrace();
}
...
```

© 2009, Laurent Vercouter 

ENSM-SE

Création de requêtes


L'objet connexion est utilisé pour créer des requêtes :

- ♦ Statement représente une requête simple


```
Connection c = DriverManager.getConnection("...");
Statement s = c.createStatement();
```
- ♦ PreparedStatement représente une requête souvent utilisée et/ou nécessitant des paramètres d'entrée


```
PreparedStatement ps = c.prepareStatement("...");
```
- ♦ CallableStatement représente un appel à une procédure


```
CallableStatement cs = c.prepareCall("...");
```

© 2009, Laurent Vercouter 

Exécution de requêtes simples

Une requête simple est exécutée par un objet Statement :

```
ResultSet executeQuery(String sql)
```

exemple :

```
Statement s = c.createStatement();
ResultSet r = s.executeQuery("SELECT a FROM table1");
```

pour les requêtes qui renvoient un résultat, et :

```
int executeUpdate(String sql)
```

exemple :

```
s.executeUpdate("INSERT INTO a VALUES (10, 'hello')");
```

pour les requêtes qui modifient la table.

Exécution de plusieurs requêtes simples

Plusieurs mises à jour consécutives peuvent être exécutées en groupe :

1. Le groupe de requêtes est construit

```
Statement s = c.createStatement();
s.addBatch("INSERT INTO a VALUES (10, 'hello')");
s.addBatch("INSERT INTO a VALUES (10, 'bonjour')");
...
```

2. Les requêtes sont toutes exécutées

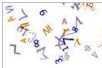
```
s.executeBatch();
```

Résultat d'une requête

Le résultat d'une requête de type SELECT est rangé dans un objet de la class *java.sql.ResultSet*.

L'objet ResultSet a la forme d'une table contenant :

- ♦ plusieurs lignes
- ♦ plusieurs colonnes typées
- ♦ un curseur sur la ligne courante



Résultat d'une requête (2)

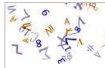
Les valeurs de chaque colonne de la ligne courante sont accédées :

- ♦ par leur nom ou leur index;
- ♦ en connaissant leur type.

Exemple :

```
ResultSet rs = stmt.executeQuery("SELECT * from
table1");
String nom = rs.getString("Intitule");
int valeur = rs.getInt(2);
```

On peut passer à la ligne suivante par la méthode next() qui renvoie le booléen faux s'il n'y a plus de ligne.
rs.next()



Transactions

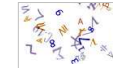
Par défaut une connexion est ouverte en mode "auto-commit".
Pour gérer la validation des changements, il faut appeler une méthode de la classe Connection :

```
public void setAutoCommit(boolean)
```

On peut ensuite appeler sur l'objet Connection les méthodes :

```
public void commit()
```

```
public void rollback()
```

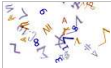


Requêtes préparées

Une requête préparée est créée avec une instruction SQL initiale qui peut être incomplète.

```
PreparedStatement ps = c.prepareStatement("SELECT ?
FROM table1 WHERE naissance=?");
```

- ♦ Les ? sont des paramètres qui doivent prendre une valeur avant d'exécuter la requête
- ♦ Une requête préparée peut être exécutée plusieurs fois (en changeant la valeur des paramètres si nécessaires)



Paramètres des requêtes préparées

Une valeur est assignée à un paramètre en :

- ♦ désignant sa position;
- ♦ appelant une méthode correspondant au type souhaité.

```
ps.setString(1, "age");  
ps.setDate(2, new Date(...));  
ResultSet rs = ps.executeQuery();
```

Remarques :

- ♦ Le résultat est du même type que pour les requêtes simples
- ♦ Les index des paramètres commencent à 1 (et non à 0) !
