

Cours de JAVA

Serge Rosmorduc
rosmord@iut.univ-paris8.fr

2000–2005

Table des matières

1	Java et les bases de données	1
1.1	Introduction à JDBC	1
1.2	Architecture	1
1.3	Un exemple : postgres	1
1.4	établir la connexion	2
1.4.1	Exemple :	2
1.5	Envoyer une requête	3
1.5.1	Méthodes	3
1.5.2	Méthodes applicables à un ResultSet	4
1.5.3	Execute	4
1.6	Commandes préparées	5
1.7	échappements SQL	5
1.8	Gestion des transactions	6
1.8.1	Niveau d'isolement	6
1.9	Capacités de la base de données : DataBaseMetaData	7
1.10	Exploration des tables	7
1.10.1	méthodes de ResultSetMetaData	7
1.11	Extensions du jdbc2.0	7
1.11.1	ResultSet navigables	8

Chapitre 1

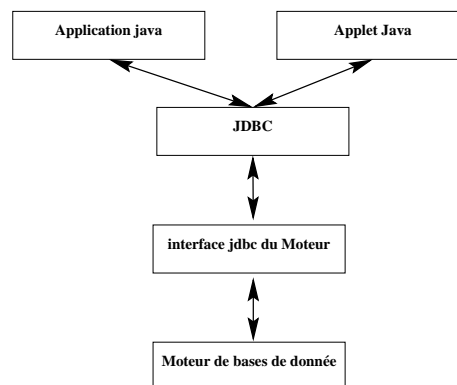
Java et les bases de données

Jdbc : Java Data Base Connectivity.

1.1 Introduction à JDBC

1. Nécessité d'utiliser un langage approprié pour interroger une base de donnée (SQL) ;
2. Nécessité d'un système client serveur à cause des restrictions des *applets*.

1.2 Architecture



1.3 Un exemple : postgres

- L'interface jdbc est distribuée avec les sources, dans `postgresql-6.2/src/interfaces/jdbc` ;
- une fois compilée, la bibliothèque est utilisable sous tout ordinateur ;
- Les sources qui l'utilisent doivent contenir la ligne :

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

– Pour spécifier le *driver* JDBC à utiliser, deux méthodes :

1. compiler les sources avec la ligne :

```
% java -Djdbc.drivers=org.postgresql.Driver monfic.java
```

2. include dans le code :

```
Class.forName("org.postgresql.Driver");
```

Attention !

Le driver doit se trouver dans le CLASSPATH; s'il s'agit d'un fichier jar, le fichier lui-même doit être dans le classpath :

```
export CLASSPATH=./home/titi/postgresql.jar:/usr/local/jdk1.2
```

1.4 établir la connexion

On utilise :

Connection

```
DriverManager.getConnection(String url,  
                             String login,  
                             String passwd);
```

La forme de l'URL est :

```
jdbc:sous_protocole:adresse
```

1.4.1 Exemple :

```
try {  
    Connection db;  
    String url= "jdbc:postgresql://localhost/guest";  
    db= DriverManager.getConnection(url, "guest", "toto");  
    // manipulations diverses :  
    ....  
    // on a fini :  
    db.close();  
} catch (SQLException e) {  
}
```

l'adresse est ici composée du nom du serveur postgres (localhost) suivi du nom de la base de donnée (ici, guest ; à l'IUT ce sera votre nom de login).

1.5 Envoyer une requête

- les **requêtes** sont représentées par la classe `Statement` ;
- les modifications de la base sont effectuées par la méthode `Statement.executeUpdate(String)` ;
- les requêtes sont effectuées par la méthode `Statement.executeQuery(String)` ;
- Le résultat d'une requête « Query » est un **ResultSet**

```
// On crée un canal de communication
Statement st= db.createStatement();
// On envoie une requête
ResultSet res= st.executeQuery("select * from Etud");
// Tant qu'il y a des lignes dans le résultat..
while (res.next()) {
    // on lit les valeurs des champs
    System.out.println("col 1 = " + rs.getString("Nom"));
}
res.close();
st.close();
```

Notes :

- on peut avoir plusieurs requêtes ouvertes sur la même connexion ;
- il n'est possible d'accéder à un champ qu'une fois et une seule ;
- il est nécessaire de fermer (`close`) les **Statement** et les **ResultSet**.

1.5.1 Méthodes

```
ResultSet executeQuery (String requete)
    throws SQLException
```

Envoie une requête SQL, (normalement de type « select »), et renvoie le résultat sous forme d'un `ResultSet`. Le résultat n'est *jamais* null.

```
int executeUpdate (String requete)
    throws SQLException
```

Exécute une requête de modification des données ou de la base (bref, tout ce qui n'est pas `select`). La valeur retournée normalement le nombre de lignes modifiées, ce qui a un sens pour `insert`, `delete`, `update`. Pour les autres opérateurs, le résultat est 0.

```
boolean execute (String requete)
    throws SQLException
```

Envoie une requête SQL qui peut même envoyer plusieurs résultats. Le résultat est `true` si la première valeur renvoyée est un `ResultSet`. Nous détaillons plus avant la méthode `execute` en 1.5.3.

1.5.2 Méthodes applicables à un ResultSet

Les méthodes suivantes permettent d'accéder à la valeur d'une colonne, soit en passant comme argument le numéro de colonne (commençant à 1), soit le nom de la colonne : `getBytes` `getShort` `getInt` `getLong` `getFloat` `getDouble` `getBigDecimal` `getBoolean` `getString` `getBytes` `getDate` `getTime` `getTimestamp` `getAsciiStream` `getUnicodeStream` `getBinaryStream` `getObject`

Par ailleurs, après appel d'une de ces méthodes, la méthode `wasNull()` permet de savoir si en fait la valeur était NULL.

1.5.3 Execute

La méthode `execute()` permet d'envoyer une requête, qu'elle soit de type « select » ou qu'elle soit une modification d'une base. Les méthodes utilisées dans l'exemple suivant permettent de récupérer des informations sur la requête. Bien entendu, dans la plupart des cas, le programmeur sait quelle est la requête, et donc utilise `executeQuery` ou `executeUpdate()`. La méthode `execute()` sera, par exemple, utilisée dans un programme où l'utilisateur pourra saisir une requête SQL quelconque.

```

----- Utilisation générale de Execute -----
stmt.execute(queryStringWithUnknownResults);
while(true) {
    int rowCount = stmt.getUpdateCount();
    if(rowCount > 0) {
        // Des données ont été modifiées
        System.out.println("Rows changed = " + count);
        stmt.getMoreResults();
        continue;
    }
    if(rowCount == 0) {
        // Modification de la Structure,
        // ou pas de changement.
        System.out.println(" Pas de ligne modifiée,
                            ou la ligne est une commande DDL");
        stmt.getMoreResults();
        continue; }
    // Si on arrive ici, il s'agit d'une requête
    ResultSet rs = stmt.getResultSet();
    if(rs != null) {
        ...
        // Il faut utiliser les métadatas pour connaître
        // la liste des colonnes
        while(rs.next())
            {
                ...
                // Traiter le résultat
                stmt.getMoreResults();
                continue;
            }
        break;
    }
}

```



```

    // there are no more results
  }
}

```

1.6 Commandes préparées

- classe `PreparedStatement` ;
- typiquement, commande utilisée plusieurs fois en changeant la valeur de certains paramètres ;
- les paramètres qui changent sont remplacés dans la commande par des « ? » ;
- les commandes `setXXX` (où `XXX` est le type de la variable) permettent de spécifier la valeur des paramètres.

```

PreparedStatement pstmt =
    connec.prepareStatement("UPDATE table4 SET m = ?
                            WHERE x = ?");

...
pstmt.setString(1, "Hi");
for (int i = 0; i < 10; i++)
{
    pstmt.setInt(2, i);
    int rowCount = pstmt.executeUpdate();
}

```

NULL : pour que la valeur d'un paramètre soit `NULL`, il suffit d'utiliser la commande `setNull`

1.7 échappements SQL

But : avoir une plus grande portabilité, et faciliter la création de commandes SQL.

Syntaxe : Dans la chaîne de commande SQL :

```
{commande arguments}
```

Spécifier un caractère d'échappement :

```
stmt.executeQuery("SELECT name FROM Identifiers
                  WHERE Id LIKE '{ } _%' {escape '\'}");
```

Spécifier une date :

```
{d 'YYYY-mm-dd'}
```

1.8 Gestion des transactions

- par défaut, chaque requête forme une transaction ;
- pour changer ce comportement, on manipule l'objet `Connection` lié à la base de donnée :

```
maconnexion.setAutoCommit(false);
```

- ensuite :

```
maconnexion.commit();
```

valide les requêtes déjà effectuées lors de cette transaction ;

```
maconnexion.rollback();
```

annule les requêtes déjà effectuées ;

1.8.1 Niveau d'isolement

But : une transaction doit « voir » un *état* de la base. Dans le cas d'accès concurrents à la base : on peut changer le type d'accès concurrent avec la méthode de `Connection` :

```
public void {setTransactionIsolation}(int level)
        throws SQLException
```

où `level` peut valoir :

TRANSACTION_READ_UNCOMMITTED

TRANSACTION_READ_COMMITTED

TRANSACTION_REPEATABLE_READ

TRANSACTION_SERIALIZABLE

TRANSACTION_READ_UNCOMMITTED on peut lire des modifications dès qu'elles sont faites. En cas de `ROLLBACK`, postérieur, les valeurs lues peuvent être fausses ;

TRANSACTION_READ_COMMITTED on ne peut pas lire une rangée sur laquelle il y a des modifications non validées (par `commit`) ;

TRANSACTION_REPEATABLE_READ idem ; de plus, évite le cas où la transaction lit une rangée, une autre transaction la modifie, et la première relit la rangée modifiée ; la lecture donne toujours le même résultat, d'où le nom ;

TRANSACTION_SERIALIZABLE le comportement est similaire à celui obtenu avec un traitement séquentiel. Empêche le cas où

1. la transaction fait un `select` avec une condition ;
2. une seconde transaction crée des lignes qui satisfont la condition ;
3. la première transaction refait le même `select`.

1.9 Capacités de la base de données : DataBaseMetaData

- Se récupère grâce à la méthode `getMetaData()` de `Connection`
- les méthodes permettent de connaître les capacités de la base. Par exemple :

`supportsSelectForUpdate()` renvoie vrai si la base permet d'utiliser un `select` dans un `update` (cf. le cours de SQL !)

des méthodes permettent d'obtenir le catalogue de la base et la liste des tables :

```
ResultSet getTables (String catalog,
                    String schemaPattern, String tableNamePattern,
                    String[] types)
```

throws **SQLException**

revoie un `ResultSet` décrivant les tables et les index de la base. Par exemple, pour afficher la liste des tables et index :

```
rs= meta.getTables(null , null, null, null);
while (rs.next()) {
    System.out.println(rs.getString("TABLE_NAME"));
}
rs.close();
```

Les arguments de `getTables` peuvent être nuls. Les plus intéressants sont :

types : un tableau de chaînes de caractère, donnant le type des tables à récupérer, entre autres : `TABLE` pour les tables stricto sensu, `VIEW` pour les vues.

1.10 Exploration des tables

la méthode `getMetaData()` de l'interface `ResultSet` permet de récupérer le `ResultSetMetaData` associé.

1.10.1 méthodes de `ResultSetMetaData`

`int getColumnCount()` nombre de colonnes;

`String getColumnName(int column)` : nom de la i^e colonne;

`String getColumnLabel(int column)` : titre de la colonne pour affichage;

`int getColumnType(int column)` : type SQL de la colonne ; les valeurs possibles pour le résultat sont décrites dans `java.sql.Types`.

1.11 Extensions du `jdbc2.0`

La version 2.0 du `jdbc` propose un certain nombre d'extensions.

1.11.1 ResultSet navigables

Par défaut, on ne peut parcourir un un `ResultSet` que d'une manière : du premier au dernier élément. Le `jdbc` version 2 permet de se déplacer librement dans un `ResultSet`, et éventuellement d'en modifier les éléments. Ces options ne sont pas forcément implémentées par les drivers `jdbc`. Par exemple, le driver `postgresql` permet les déplacements, mais pas la modification.

Pour disposer de `ResultSets` modifiables, il faut le demander au moment de créer un `Statement`, en utilisant la méthode `createStatement` de la classe `Connection` :

```
Statement createStatement (int resultSetType, int
    resultSetConcurrency)
    throws SQLException
```

resultSetType trois valeurs possibles :

ResultSet.TYPE_FORWARD_ONLY : seul les déplacements vers l'avant sont possibles

ResultSet.TYPE_SCROLL_INSENSITIVE : tout déplacement est possible. Par contre, si les données sont modifiées, et que l'on revient sur une ligne déjà visitée, la valeur visible sera la valeur d'origine et non la valeur modifiée.

ResultSet.TYPE_SCROLL_SENSITIVE : tout déplacement est possible. , Si les données sont modifiées, et que l'on revient sur une ligne déjà visitée, la valeur visible sera la valeur modifiée.

resultSetConcurrency : règle le comportement du `ResultSet` en cas par rapport aux transactions.

ResultSet.CONCUR_READ_ONLY : lecture seule ;

ResultSet.CONCUR_UPDATABLE : modifiable.

Déplacement dans le ResultSet

Un `ResultSet` fonctionne comme un tableau dont les cases sont numérotées de **1** à **n**. Il dispose de plus de deux positions spéciales, `beforeFirst` et `afterLast`, aux deux extrémités du tableau. Le curseur est à l'origine placé sur `beforeFirst`.

```
void last ()
    throws SQLException
    se place au dernier enregistrement.
```

```
void first ()
    throws SQLException
    se place au premier enregistrement.
```

```
void afterLast ()
    throws SQLException
    se place après le dernier enregistrement.
```

```
void beforeFirst ()
    throws SQLException
    se place avant le premier enregistrement.
```

```
void next ()
    throws SQLException
    avance à l'enregistrement suivant.
```

```
void previous ()
    throws SQLException
    avance à l'enregistrement précédent.
```

```
boolean absolute (int i)
    throws SQLException
    se place sur l'enregistrement numéro i. Si i vaut 1, c'est l'équivalent de first. Si
    i est négatif, on numérote à partir du dernier enregistrement.
    La fonction renvoie true si le curseur pointe sur un enregistrement valide.
```

```
boolean relative (int delta)
    throws SQLException
    Déplacement relatif à la position courante. relative(-1) est équivalent à previous,
    et relative(1) à next().
    La fonction renvoie true si le curseur pointe sur un enregistrement valide.
```

```
int getRow ()
    throws SQLException
    renvoie l'indice de la ligne courante.
```

Modification d'un ResultSet

Un `ResultSet` n'est modifiable que si on l'a demandé et que le driver le gère.

Les méthodes principales sont (remplacer `XXX` par `int`, `String`...) :

```
void updateXXX (int i, XXX a)
    throws SQLException
    modifie le ie champ, de type XXX, en lui donnant la valeur a.
```

```
void updateXXX (String name, XXX a)
    throws SQLException
    modifie le champ nommé name, de type XXX, en lui donnant la valeur a.
```

```
void deleteRow ()
    throws SQLException
    détruit la ligne courante.
```

```
void moveToInsertRow ()
    throws SQLException
    se place sur une ligne spéciale, qui sert aux insertions de nouvelles données.
```

```
void moveToCurrentRow ()  
    throws SQLException  
    après un appel à moveToInsertRow, revient à sa position initiale.
```

```
void insertRow ()  
    throws SQLException  
    insère le contenu de la ligne d'insertion dans la base.
```