Programmation JAVA – Interagir avec une base de données grâce à JDBC

Qu'est-ce que JDBC ?

JDBC (pour Java DataBase Connectivity) est une API (Application Programming Interface) permettant l'accès aux bases de données relationnelles et d'exécuter des instruction SQL au sein d'un programme JAVA. Il fait partie du JDK et les classes et interfaces se trouve dans le package java.sql et javax.sql

JDBC est indépendant du type de base utilisée (mySQL, Oracle, Postgres …). Cependant, lors de la connexion SGBDR, il faudra spécifier à quel type de base s'interfacer.

Déroulé type d'utilisation de JDBC dans un programme

Première étape

Dans un premier temps, il va falloir venir préciser/importer le driver que l'on souhaite utiliser. Il en existe un par type de base :

- ODBC pour Oracle
- MySQL Connector pour Mysql
- pgJDBC pour Postres
- …

Deuxième étape

Se connecter à un SGBD en fournissant le nom de la base ainsi que le login et mot de passe pour s'authentifier. En se connectant, nous allons récupérer un objet appelé « Connection » Qui va nous permettre d'interagir avec cette dernière.

Troisième étape (répétable n fois)

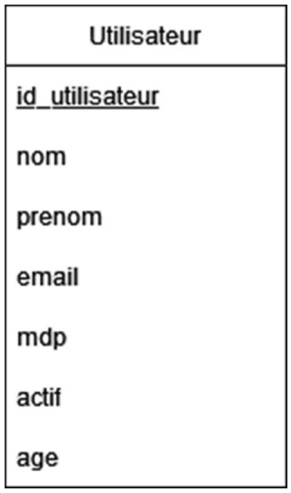
A partir de la connexion, nous allons pouvoir créer une requête (créer un objet « Statement »). Sur cette requête, lui donner les informations nécessaires venant du programme puis l'exécuter au niveau du SGBD. Une fois terminé avec cette requête, on peut venir clôturer la requête.

Dernière étape

Se déconnecter de la base de données en clôturant la connexion.

Mise en place dans un projet JAVA

Pour les exemples, nous allons Interagir avec une table nommé Utilisateur :



1. Connexion au SGBD (DriverManager)

Pour se connecter à la base de données, nous allons avoir besoin du DriverManager. Son rôle est de nous permettre de créer et de récupérer une connexion entre le code JAVA et notre base de données.

Nous allons utiliser principalement la fonction getConnection proposé par le DriverManager :

static Connection getConnection(String url, String user, String password)

Cette fonction nous renvoie en retour notre objet Connexion à la base de donnée et nécessite trois paramètres :

- url : identification de la base de donnée. Le format est dépendant de SGBD utilisé
  - Pour Mysql, l'url sera « jdbc:mysql://localhost:3306/nomDeLaBaseDeDonnée »
- user : nom de l'utilsiateur qui se connecte à la base
- password : le mot de passe de l'utilisateur

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class CoursJDBC {
    public static void main(String[] args) throws SQLException {
        // connexion à la BDD sle_exJDBC avec le compte root
        Connection maConnection = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/sle_exJDBC","root","");
    }
}
```

2. Gestion de la connexion (l'objet « Connexion »)

Une fois connecté, nous pouvons demander à notre objet Connexion de préparer deux types d'instructions :

- Une instruction simple : Celle-ci est utilisé plutôt pour des requêtes nécessitant aucun argument.
  - *Statement createStatement() <- fonction de création d'une requête simple*
    - Renvoie simplement un objet permettant d'exécuter une requête.
- Une instruction paramétrée : Permet de constituer des instructions génériques dont les champs sont non remplis. Cela permet une pré-compilation de l'instruction optimisant les performances. Avant chaque exécution, penser à fournir les arguments si nécessaire.
  - *PreparedStatement prepareStatement(String ordre)*
    - Retourne un objet permettant de réaliser une instruction paramétrée et pré-compilée. Le paramètre « ordre » correspond à la requête que l'on souhaite exécuter par la suite.
    - Dans l'ordre, les champs libres (au nombre quelconque) sont précisés par des « ? »
      - « SELECT * FROM Utilisateur WHERE email = ? »
      - Avant l'exécution de la requête, il faudra venir préciser ce que l'on souhaite à l'emplacement « ? ».

Pour ces deux types d'instructions, il existe deux types d'ordres possibles d'exécution :

- Update : Permettant de réaliser des opérations de mise à jour de la base ;
- Query : Permettant de réaliser des opérations de consultations (Select) des données de la base.

Pour clôturer la connexion à la base de donnée, on utilisera la fonction « close() ».

```
import java.sql.*;

public class CoursJDBC {
    public static void main(String[] args) throws SQLException {
        // connexion à la BDD sle_exJDBC avec le compte root
        Connection maConnection = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/sle_exJDBC","root","");

        // création d'une requête simple
        Statement requeteSimple = maConnection.createStatement();

        //Création d'une requête paramétrée
        PreparedStatement requetePrepare = maConnection.prepareStatement(
                "SELECT * FROM Utilisateur WHERE email = ?"
        );

        /**
         * exécution des requêtes
         * traitement des résultats
         * ....
         */

        // Clôture la connexion à la base de donnée
        maConnection.close();
    }
}
```

3. Réalisation d'une requête simple (l'objet Statement)

Pour exécuter une requête simple, nous avons les deux possibilités suivantes :

- *int executeUpdate(String ordre)*
  - Permet d'exécuter une requête (le paramètre « ordre ») de type INSERT, UPDATE, ou DELETE.
- *ResultSet executeQuery(String ordre)*
  - Permet d'exécuter une requête (le paramètre « ordre ») de type SELECT sur la base et retourne un objet contenant l'ensemble des résultats de la requête (l'objet ResultSet)

```
import java.sql.*;

public class CoursJDBC {
    public static void main(String[] args) throws SQLException {
        // connexion à la BDD sle_exJDBC avec le compte root
        Connection maConnection = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/sle_exJDBC","root","");

        // création d'une requête simple
        Statement requeteSimple = maConnection.createStatement();


        // requête opérant une MAJ dans la base de donnée
        requeteSimple.executeUpdate(
                            "DELETE FROM Utilisateur WHERE actif=0");

        // requête Permettant d'extraire des données de la base de donnée
        ResultSet resultatsRequetes = requeteSimple.executeQuery(
                            "SELECT * FROM Utilisateur");


        // Clôture la connexion à la base de donnée
        maConnection.close();
    }
}
```

4. Réalisation d'une requête paramétrée (l'objet PreparedStatement)

Pour exécuter une requête paramétrée, il est nécessaire de fournir les champs manquants (s'il y en a) avant l'exécution :

- void set[Type](int index, [Type] val)
  - Remplit le champ en ième position définie par index avec la valeur val de type [Type]
  - Le type peut être String, int, long, float …
  - Exemple : *setString(int index, String val)*

Une fois les champs manquants rempli, il nous est possible d'exécuter la requête. Comme pour une requête simple, nous avons deux possibilités d'exécution :

- *int executeUpdate()*
  - Permet d'exécuter la requête paramétrée de type INSERT, UPDATE, ou DELETE.
- *ResultSet executeQuery()*
  - Permet d'exécuter la requête paramétrée de type SELECT sur la base et retourne un objet contenant l'ensemble des résultats de la requête (l'objet ResultSet)

```
import java.sql.*;

public class CoursJDBC {
    public static void main(String[] args) throws SQLException {
        // connexion à la BDD sle_exJDBC avec le compte root
        Connection maConnection = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/sle_exJDBC","root","");

        //requête opérant une MAJ dans la base de donnée
        PreparedStatement requetePrepareInsert =
                maConnection.prepareStatement(
                "INSERT INTO Utilisateur (nom,prenom,email,mdp,actif,age)
VALUES (?,?,?,?,?,?)");
        // avant l'exécution, fournir les champs manquant
        // 1 - nom => String
        requetePrepareInsert.setString(1,"Franck");
        // 2 - prenom => String
        requetePrepareInsert.setString(2,"Dubosc");
        // 3 - email => String
        requetePrepareInsert.setString(3,"f.dubosc@test.fr");
        // 4 - mdp => String
        requetePrepareInsert.setString(4,"LesFlotsBleu");
        // 5 - actif => boolean
        requetePrepareInsert.setBoolean(2,true);
        // 6 - age => int
        requetePrepareInsert.setInt(2,59);

        //Execution de la requête
        requetePrepareInsert.executeUpdate();

        //requête permettant d'extraire des données de la base de donnée
        PreparedStatement requPrepareSelect =
maConnection.prepareStatement(
                "SELECT * FROM Utilisateur WHERE email = ?"
        );
        // 1 - email => String
        requPrepareSelect.setString(1,"f.dubosc@test.fr");
        //Execution de la requête
        ResultSet resultatsRequetes = requPrepareSelect.executeQuery();
    }
}
```

5. Lecture des résultats (l'objet ResultSet)

L'objet ResultSet est l'équivalent d'un tableau de tableau :

- Le premier tableau correspont au nombre de lignes récupéré en résultats
  - Exemple : Tous les Inscrits
- Le deuxième tableau représente les colonnes décrivant chaque ligne
  - Le détail d'un inscrit (nom, prenom etc…)

Sauf que ce n'est pas un tableau « ordinaire », on ne peux pas utiliser les [] pour intéragir avec les résultats. Il faut utiliser certaines fonctions pour les lignes et pour les colonnes :

Changement de ligne

- *boolean next()*
  - Se place à la ligne suivante s'il y en a une
  - Retourne true si le déplacement a été fait, false s'il n'y avait pas d'autre ligne
- *boolean previous()*
  - Se place à la ligne précédente s'il y en a une
  - Retourne true si le déplacement a été fait, false s'il n'y avait pas de ligne précédente
- *boolean absolute(int index)*
  - Se place à la ligne numérotée index
  - Retourne true si le déplacement a été fait, false sinon
  - La première ligne commence par 1 (et non 0 comme les tableaux)
- *boolean first()*
  - Se place à la première ligne de résultat
  - Retourne true si le déplacement a été fait, false sinon
- *boolean last()*
  - Se place à la dernière ligne de résultat
  - Retourne true si le déplacement a été fait, false sinon
- int getRow()
  - Récupère le nombre de résultat possible
  - 0 si pas de ligne, > 0 si résultat

Accès aux données concernant la ligne courante

- [type] get[Type](int col)
  - Retourne le conteny de la colonne dont l'élément est de type [type].
  - Le type peut être String, int, long, float …
  - Exemple : *setString(int index, String val)*

```
import java.sql.*;

public class CoursJDBC {
    public static void main(String[] args) throws SQLException {
        // connexion à la BDD sle_exJDBC avec le compte root
        Connection maConnection = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/sle_exJDBC","root","");

        //requête permettant d'extraire des données de la base de donnée
        PreparedStatement requetePrepareSelect =
maConnection.prepareStatement(
                "SELECT * FROM Utilisateur WHERE actif = ?"
        );
        // 1 - email => String
        requetePrepareSelect.setBoolean(1,true);
        //Execution de la requête
        ResultSet resultatsRequetes =
requetePrepareSelect.executeQuery();

        /**
         * Pour l'exemple : Il y a 4 Utilisateux actifs dans la table
         */
        System.out.println(resultatsRequetes.getRow());

        // tant que j'ai un élément suivant, je me place à cette
emplacement
        while (resultatsRequetes.next()){
            // pour la ligne courante, j'affiche le nom et prénom
            System.out.println(resultatsRequetes.getString(2)
                    +" "+ resultatsRequetes.getString(3));
        }
        // Mon curseur est actuellement sur la dernière position
        // Je retourne à la première ligne de mon résultat
        resultatsRequetes.first();
        // j'affiche le nom prénom de la première ligne
        System.out.println(resultatsRequetes.getString(2)
                           +" "+ resultatsRequetes.getString(3));
        // Je me replace sur la dernière ligne
        resultatsRequetes.last();
        // j'affiche le nom prénom de la dernière ligne
        System.out.println(resultatsRequetes.getString(2)
                +" "+resultatsRequetes.getString(3));

        // J'essaye de me placer au 7e élément du résultat
        boolean mouvementReussi = resultatsRequetes.absolute(7);
        // faux, le mouvement n'a pas pu se réaliser, il n'y a que 4
résultats
        // on reste donc à la dernière position
        System.out.println(mouvementReussi);

        // J'essaye de me placer au 3e élément du résultat
        mouvementReussi = resultatsRequetes.absolute(3);
        // vrai, le mouvement a pu se réaliser
        // Je suis donc sur le 3e résultat
        System.out.println(mouvementReussi);
        // j'affiche l'email et l'age de l'Utilisateur à la position 3
        System.out.println(resultatsRequetes.getString(4)+"
"+resultatsRequetes.getInt(7));

    }
}
```