Programmation orientée objet en langage JAVA

Connexion à une base de données avec JDBC

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/~duvallet/

Connexion à une base de données avec JDBC

Introduction à JDBC

- Dans de nombreuses applications, il est nécessaire de se connecter à une base de données.
- Nous utiliserons dans notre cas de figure la base de données MySQL.
- En JAVA, pour se connecter à une base de données et effectuer des manipulations sur cette base, il existe l'API JDBC.
- L'api JDBC est disponible dans le package `java.sql` qui est nativement présent au sein de l'api JAVA.

Les principales classes

- **La classe** `DriverManager` charge et configure le driver de la base de données.
- **La classe** `Connection` réalise la connexion et l'authentification à la base de données.
- **La classe** `Statement` **(et** `PreparedStatement`**)** contient la requête SQL et la transmet à la base de données.
- **La classe** `ResultSet` permet de parcourir les informations retournées par la base de données dans le cas d'une sélection de données. Il contient les données de la ligne courante et un pointeur sur la ligne suivante.

Manipulation du JDBC

- La manipulation du JDBC sera vu au travers d'un exemple présentant les principales étapes à respecter.
- Au préalable, nous avons créée une base de données MySQL appelé `MTP-DB` et au sein de cette base, une table `Personne`.
- Nous allons passer en revue :
  - la connexion à une base de données avec le chargement du drivers,
  - l'insertion d'une nouvelle ligne dans la table `Personne`,
  - la sélection d'éléments dans la table `Personne`,
  - la suppression d'un élément dans la table.
- L'exemple complet vous permettra de tester effectivement la manipulation d'une base de données avec l'api JDBC.

Description générale de l'exemple

- La classe est nommée `TestConnexionMySQL`.
- Il faut importer le package `java.sql.*`.
- Elle possède deux attributs de type `Connection` et `Statement`.
- Le constructeur a pour objectif de charger le driver de la base de données puis d'ouvrir la connexion qui sera utilisée pour tout le reste des manipulations.
- Pour exécuter des requêtes de sélection, il faut utiliser la méthode `executeQuery()` de l'interface `java.util.Statement`.
- Pour exécuter des requêtes de mise à jour (modification/suppression), il faut utiliser la méthode `executeUpdate()` de l'interface `java.util.Statement`.

Chargement du driver et connexion

- Au préalable, il faut avoir récupérer le driver JDBC pour MySQL.
- Il se présente sous la forme d'un fichier jar.

```
public TestConnexionMySQL (){
    // Chargement du pilote
    try {
          Class.forName("com.mysql.jdbc.Driver").newInstance();
    } catch (Exception e) {
          e.printStackTrace();
          System.exit(99);
    }

    // Connexion à la base de données MySQL "MTP-DB" avec
    // le login "duvallet" et le mot de passe "duvallet"
    try {
        String DBurl = "jdbc:mysql://localhost:3306/MTP-DB";
        con = DriverManager.getConnection(DBurl,"duvallet","duvallet");
        stmt = con.createStatement();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Insertion d'une ligne au sein de la table `Personne`

- Utilisation de la méthode `executeUpdate()`.
- Création d'une requête SQL d'insertion.
- Le numéro qui représente la clef primaire est généré automatiquement puis récupéré.

```
public int insertPersonne (String nom, String prenom, int age){
    ResultSet resultats = null;
    int idGenere = -1;
    try {
        stmt.executeUpdate("INSERT INTO Personne (nom, prenom, age)
                            values ('"+nom+"', '"+prenom+"',"+age+")",
                            Statement.RETURN_GENERATED_KEYS);
        resultats = stmt.getGeneratedKeys();
        if (resultats.next()) {
            idGenere = resultats.getInt(1);
        }
        resultats.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return idGenere;
}
```

Afficher une personne à partir de sa clef primaire (numéro)

1. On effectue une requête permettant de récupérer la ligne de la table `Personne` dont le numéro est donné en paramètre de la méthode.
2. On affiche la ligne obtenue si il existe une ligne correspondant au numéro.

```
public void affichePersonne (int key){
    ResultSet resultats = null;
    try {
        boolean encore = false;
        resultats = stmt.executeQuery("Select Nom, Prenom, Age
                                       From Personne Where Numero="+key);
        if (resultats != null)
          encore = resultats.next();

        if (encore)
            System.out.println (resultats.getString(1) + " "
                                + resultats.getString(2)
                                + " (" + resultats.getInt(3) + " ans)");
        else
          System.out.println ("Il n'y a personne avec ce numero !");
        resultats.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Afficher toutes les lignes de la table

1. On effectue une requête permettant de récupérer la liste de tous les enregistrements présents dans la table `Personne`.
2. On parcourt l'ensemble des résultats obtenus (les lignes de la table) puis on les affiche.

```
public void afficheToutesLesPersonnes (){
  ResultSet resultats = null;
  try {
    resultats = stmt.executeQuery("Select nom, prenom, age, numero from Personne");
    boolean encore = resultats.next();

    while (encore) {
        System.out.println (resultats.getInt(4) + " :  " + resultats.getString(1)
                            + " " + resultats.getString(2) + " ("
                            + resultats.getInt(3) + " ans)");
        encore = resultats.next();
    }
    resultats.close();
  } catch (SQLException e) {
      e.printStackTrace();
  }
}
```

Supprimer des personnes

- Utilisation de la méthode `executeUpdate()`.
- Création d'une requête SQL de suppression.

```
public void supprimeToutesLesPersonnes (){
    try {
        stmt.executeUpdate("Delete From Personne");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void supprimePersonne (String nom){
    try {
        stmt.executeUpdate("Delete From Personne Where nom='"+nom+"'");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Créer une table

- Création d'une table avec un seul champs de type `VarChar(25)`.

```
public void creerTable (String nomTable, String champs){
    ResultSet resultats = null;
    try {
        stmt.executeUpdate("CREATE TABLE "+nomTable
                           +" ("+champs+" VARCHAR(25))");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Tester le fonctionnement de l'exemple

- Nous commençons par vider la table ce qui évitera que, sur plusieurs tests successifs, il n'y ait trop d'entrées dans la table.
- Insertion de trois personnes.
- Affichage de la première personne insérée.
- Affichage de toutes les personnes.
- Suppression de 'Duvallet' puis affichage de toutes les personnes.
- Création d'une table "Etudiant" avec un champs "Identifiant" : on ne peut le faire qu'une seule fois.

Le code complet de la méthode principale

```
public static void main(java.lang.String[] args) {
  TestConnexionMySQL test = new TestConnexionMySQL ();
  test.supprimeToutesLesPersonnes ();

  System.out.println ("Insertion de trois personnes");
  int id1 = test.insertPersonne ("Duvallet", "Claude", 36);
  int id2 = test.insertPersonne ("Amanton", "Laurent", 40);
  int id3 = test.insertPersonne ("Sadeg", "Bruno", 48);

  System.out.println ("Affichage de la personne de clef primaire = "+id1)
  test.affichePersonne (id1);

  System.out.println ("Affichage de toutes les personnes");
  test.afficheToutesLesPersonnes ();

  System.out.println ("Suppression de Duvallet et affichage
                                    de toutes les personnes");
  test.supprimePersonne ("Duvallet");
  test.afficheToutesLesPersonnes ();
  // Création de la table Etudiant
  test.creerTable("Etudiant", "Identifiant");
}
```

Les méthodes de la classe `Statement`

- La méthode `executeQuery()` :
  - Elle permet d'exécuter des requêtes de type "`Select`" écrites en langage SQL.
  - L'objet retourné est de type `ResultSet` et il n'est jamais `null` mais éventuellement vide.
- La méthode `executeUpdate()` :
  - Elle permet d'exécuter des requêtes de mise-à-jour écrites en langage SQL.
  - Les requêtes SQL sont alors de type `INSERT`, `UPDATE` ou `DELETE`.

Les méthodes de la classe `ResultSet` (1/2)

- La méthode `next()` permet d'accéder à la ligne suivante. Au départ, le point est placé avant la première ligne.
- Les méthodes `getType()` :
  - Elles permettent de lire le résultat d'une colonne suivant son type.
  - C'est ainsi que l'on a des méthodes `getInt()`, `getString()`, `getDouble()`, `getDate()`,....
  - Deux paramètres sont possibles pour ces méthodes : (1) un entier représentant l'index de la colonne, (2) une chaine représentant le nom de la colonne.

Les méthodes de la classe `ResultSet` (2/2)

- La mise à jour des données :
  - Durant le parcours d'un objet de type `ResultSet`, il est possible d'effectuer des mises à jour sur la ligne courante du curseur.
  - Pour cela, il faut déclarer l'objet `ResultSet` comme acceptant les mises à jour :

```
Statement statement =
    connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                               ResultSet.CONCUR_READ_ONLY);
```

  - Avec les versions précédentes de JDBC, il fallait utiliser la méthode `executeUpdate()` avec une requête SQL.
  - Maintenant pour réaliser ces mises à jour, JDBC 2.0 propose de les réaliser via des appels de méthodes plutôt que d'utiliser des requêtes SQL.