Java Swing

Atelier III: Interrogation de la base de données

Objectif

- Vers la fin de cet atelier vous devrez être capable de lier les interfaces *Java Swing* avec une base de données ;
- Enrichir votre application par d'autres formulaires utiles ;

Introduction : Java et les bases de données

Lors de développement des applications avec un langage donné, le besoin de stocker ou de récupérer les données surgit et persiste comme une étape inévitable. Pour ceci, tous les langages de programmation offrent des interfaces pour l'interrogation des bases de données.

Dans le cas du langage Java, l'utilisation d'une base de données n'est pas très différente des autres langages. La démarche est similaire à peu près pour tous les langages en respectant ces étapes :

- Chargement du pilote ;
- Création de la chaine de connexion ;
- Ouverture de la base ;
- Création et exécution des requêtes ;
- Récupération des résultats.

Etape I : Chargement du pilote

Les langages de la programmation n'ont pas un moyen natif pour communiquer avec une base de données. Ceci est dû à plusieurs raisons, parmi elles on trouve la nature hétérogène d'une base et d'un langage, multitude de base fournies par des entreprises différentes…

Pour cela, le langage en général, et Java en particulier a besoin d'une couche logicielle intermédiaire pour surpasser l'hétérogénéité entre les deux parties. Cette couche intermédiaire est appelée souvent *Pilote ou Driver.* Le pilote est utilisé par Java afin d'exécuter des requêtes *SQL* et récupérer les résultats. Le développeur a un large

choix de trouver le pilote qui le convient le plus. Ce choix peut se baser sur plusieurs critères : adapté au langage, adapté au SGBD, gratuit, indépendant, multi-langage…etc

Le pilote le plus utilisé dans le monde est l'*ODBC (open database conncectivity)*. Il support plus de 700 pilotes compatible avec plusieurs SGBD (avec leurs versions) et langage. Dans JAVA, l'API JDBC permet de coopérer et/ou utiliser le pilote ODBC pour se connecter à une base de données. Le développeur doit télécharger le pilote JDBC compatible pour un SGBD (attention aux versions) qui sera utilisé dans l'application. Dans le cas de *mySQL*, le téléchargement pilote est disponible de l'URL suivante : https://dev.mysql.com/downloads/connector/j/. Une fois le pilote installé, il est utilisable dans votre application sachant que vous disposez déjà d'une base données mySQL. L'instruction pour charger le driver est comme suit :

- Class.forName("com.mysql.jdbc.Driver");

Remarque : si vous avez un problème du chargement, ajouter le jar comme external librery (chemin par defaut : C:\Program Files (x86)\MySQL\Connector J 5.1.26\).

Toutes les classes de JDBC sont dans le package java.sql. Il faut donc l'importer dans tous les programmes devant utiliser JDBC :

- Import java.sql.*

Etape II : création d'une connexion

Le développeur doit exprimer l'ouverture d'une ou plusieurs bases de données par la méthode *getconnection()* en précisant une variable de type Connection comme un identifiant de la base dans le programme. L'exemple suivant illustre une création d'une connexion avec une base mySQL :

```
try {
   Connection con =
       DriverManager.getConnection("jdbc:mysql://localhost/ensa?" +
                                   "user=root&password=");

    System.out.println(" connexion reussie " );


} catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
```

Etape III : création et exécution d'une requête

Une requête sur Java est interprétée comme une chaine des caractères. Pour l'interpréter comme un Script SQL il faut l'introduire à une variable de type *statement*. Cette dernière va être entretenu par le pilote comme une requête SQL, surtout lors de son execution avec la fonction *exexuteQuety().* Le resultat de la requête est stocké dans une variable capable de supporter une grande quantité d'information et de deux dimensions : ResultSet.

Une fois les données récupérées, il faut les utiliser dans le programme. A priori, le développeur doit lire une ligne et lire cellule par cellule. La fonction *Next()* appliquée sur un ResultSet permet de lire une ligne et pointer sur la prochaine. La fonction getType() (getString(), getInt()) permet de récupérer une valeur en indiquant entre les parenthèse le nom ou le numéro de la colonne ciblée.

```
try{
                    Statement st;
                    st = con.createStatement();
                    ResultSet res = null;
                    res  = st.executeQuery("select * from etudiant");
                    while(res.next())
                          System.out.println("le resultat est " +
res.getString("nom"));
              }
              catch (Exception ex){

              }
```

Remarque :

Dans le cas d'une requete d'insertion ou de suppression la fonction executeQuesry() est inutile. Il faut utiliser la fonctions ExecuteUpdate(requete) :

statement.executeUpdate("delete from etudiant where prenom='ahmed' ")

Travailler demandé :

Le formulaire de l'atelier dernière doit ajouter les étudiants dans la base ;

Ajouter un autre formulaire pour la recherche d'un étudiant par plusieurs critères (nom, prénom…) ;

Ajouter un formulaire pour la modification des informations à propos d'un étudiant ou le supprimer carrément de la table.