Chapitre 6

JDBC

Introduction

- JDBC (Java DataBase Connector) est une API chargée de communiquer avec les bases de données en Java.
- Les classes et interfaces de l'API JDBC figurent dans le package java.sql : `import java.sql.*;`
- JDBC peut être utilisé pour accéder à n'importe quelle base de données à partir de:
  - Simple application Java
  - Une servlet
  - Page JSP, …

Travail avec une base de données

- JDBC permet de travailler avec les base de données de la même façon quelque soit leur fournisseur (Oracle, SQL Server, MySQL, PostgreSQL,…).
- Il suffit de télécharger la bibliothèque qui assure la communication entre Java et cette base de donnée.
- Cette bibliothèque s'appelle Driver ou Pilote ou Connecteur.
- Elle figure sur le site du fournisseur du SGBDR utilisé.

Etapes d'interaction avec une BDD

1. Chargement du pilote
2. Etablissement de la connexion
3. Création des objets encapsulant les requêtes
4. Exécution des requêtes
5. Parcours des résultats dans le cas d'une requête de sélection
6. Fermeture des objets résultats, requêtes et connexion

Chargement du pilote

- Dans ce cours nous allons prendre MySQL comme exemple.
- Le connecteur MySQL pour Java se nomme comme cet exemple : " **mysql-connector-java-8.0.27.jar**"
- Pour se connecter à une base de données il faut charger son pilote.
- La documentation de la Bdd utilisée fournit le nom de la classe à utiliser.

342

Chargement du pilote

- Le chargement se fait comme suit :
  Class.forName(nom_classe_connecteur);
- Exemple :
  - Dans le cas de la Bdd MySQL, ce chargement est comme suit : **Class.forName(com.mysql.cj.jdbc.Driver)**
- Une fois chargée, la classe JDBC qui se nomme **DriverManager** prend en charge le driver pour communiquer avec la base de donnée.

343

Classes de l'API JDBC

- Les classes et interfaces les plus usuelles sont les suivantes:
  - **DriverManager** (classe): charge et configure le driver de la base de données.
  - **Connection** (interface): réalise la connexion et l'authentification à la base de données.
  - **Statement** (interface): contient la requête SQL et la transmet à la base de données.
  - **PreparedStatement** (interface): représente une requête paramétrée
  - **ResultSet** (interface): représente les résultat d'une requête de sélection.

Etablissement de la connexion

- Pour se connecter à une base de données, il faut disposer d'un objet **Connection** créé grâce au DriverManager en lui passant :
  - l'URL de la base à accéder , Le login, Le mot de passe
- Exemple avec MySQL:
  - String url="jdbc:mysql://localhost/mydb";
  - String login="root";
  - String password="motdepasse";
  - **Connection** con=**DriverManager**.**getConnection**(url, login, password);

Exécution de requêtes SQL

- L'interface **Statement** permet d'envoyer des requêtes SQL à la base de données.
- Un objet Statement est créé grâce à un objet Connection de la façon suivante : **Statement** st = **con**.createStatement();
- Il possède deux méthodes :
  - **executeUpdate()** : Insertion, suppression, mise à jour.
    - int n= st.executeUpdate("**INSERT INTO Etudiant VALUES (3452,'Taha','Ali')**");
  - **executeQuery()** : Selection.
    - **ResultSet** res= st.**executeQuery**(**"SELECT * FROM Etudiant"**);

Requêtes avec paramètres

- L'interface **preparedStatement** permet d'envoyer des requêtes SQL à la base de données en prenant des paramètres.
- Ces paramètres sont représentés par des points d'interrogation(**?**) et doivent être spécifiés avant l'exécution.
- Exemple :
  - **PreparedStatement** p= con.**prepareStatement**("select* from Etudiant where cne=? And nom= ? ");

Requêtes avec paramètres

```
p.setInt(1, 3452345);
p.setString(2, "Alaoui");
ResultSet resultats = p.executeQuery();
```


348

Résultat d'une requête de sélection

- Une requête de sélection retourne un **ResultSet**
- ResultSet est un ensemble d'enregistrements constitués de colonnes qui contiennent les données.
- Les principales méthodes :
  - **next()** : se déplace sur le prochain enregistrement : retourne false si la fin est atteinte. Le curseur pointe initialement juste avant le premier enregistrement.
  - **getString(**int/String**)** : retourne le contenu de la colonne dont le numéro (resp. le nom) est passé en paramètre sous forme d'une chaine de caractère.

349

Résultat d'une requête de sélection

- **getInt(**int/String**)** : retourne le contenu de la colonne dont le numéro (resp. le nom) est passé en paramètre sous forme d'entier.
- **getFloat(int/String)** : retourne le contenu de la colonne sous forme de nombre flottant.
- **getDate(int/String)** : retourne le contenu de la colonne sous forme de date.
- **Close()** : ferme le ResultSet

Résultat d'une requête de sélection

- Exemple :

```
ResultSet res= st.executeQuery("SELECT * FROM Etudiant");
while (res.next()) {
    System.out.println("CNE= "+res.getString(1)+" Nom= " +
        res.getString(2)+" Prénom= "+res.getString(3));
}
res.close();
```

Exemple complet

```
import java.sql.*;
public class Main {
   public static void main(String[] args) {
      String url="jdbc:mysql://localhost/etudiantsDB";
      String driver = "com.mysql.cj.jdbc.Driver";
      try { Class.forName(driver);
            Connection con=DriverManager.getConnection(url,"root","");
            Statement st = con.createStatement();
            ResultSet res= st.executeQuery("SELECT * FROM Etudiant");
```


Exemple complet

```
  while (res.next()) {
    System.out.println("CNE= "+res.getString(1)+" Nom= "
                  +res.getString(2)  + " Prénom= "+res.getString(3 ));
  }
  res.close();
  st.close();
  con.close();
 }
 catch (Exception e) {       e.printStackTrace();   }
 }
}
```

Métadonnées sur la base de données

- Classes pour obtenir des métadonnées:
  - **DatabaseMetaData** : informations à propos de la base de données : nom des tables, index, version ...
  - **ResultSetMetaData** : informations sur les colonnes (nom et type) d'un ResultSet
- Exemple :

```
ResultSetMetaData meta = res.getMetaData();
int nbCols = meta.getColumnCount();
while (res.next()) {
  for (int i = 1; i <= nbCols; i++) {
     System.out.print(meta.getColumnName(i)+" = "+res.getString(i) + " ");
  }
  System.out.println();
}
```


Transactions

- Une transaction est un ensemble de requêtes qui doivent s'exécuter d'un seul bloc.
- Si une requête de cet ensemble échoue alors toutes les autres sont annulées
- Par contre si toutes les requêtes réussissent alors l'ensemble est validés.

Validation automatique

- Par défaut, les Connexion sont en mode de validation automatique (auto-commit) où chaque instruction SQL est considérée comme une transaction.
- La validation automatique peut être désactivée grâce à la méthode setAutoCommit()
- Exemple :
  - conn.setAutoCommit(false);// conn est un objet Connection

Commit & Rollback

- Une fois les modifications sont terminées, ils sont validées grâce à la méthode **commit ()** sur l'objet de connection
- Exemple :
  - conn.commit( );
- Si un problème se produit dans la suite des requêtes exécutées, alors l'ensemble peut être annulée grâce à la méthode rollback()
- Exemple :
  - Conn.rollback()

Commit & Rollback

- Exemple :

```
try{
  conn.setAutoCommit(false);
  Statement stmt = conn.createStatement();
 //requête correcte
  String requete1 = "INSERT INTO Etudiant VALUES (26, 'Alaoui', 'Ali')";
  stmt.executeUpdate(requete1);
  //requête fausse
  String requete2 = "INSERT INTOO Etudiant VALUES (27,'Omari', 'Omar')";
  stmt.executeUpdate(requete2);
  conn.commit(); // si il n y a pas d'erreur
}
catch(SQLException se){
  conn.rollback(); // si il  y a des erreurs
}
```


Points de sauvegarde

- Un point de sauvegarde, est un point de restauration logique dans une transaction.
- Si une erreur se produit après un point de sauvegarde, la méthode de restauration peut:
  - Annuler soit toutes les modifications,
  - Annuler uniquement les modifications apportées après le point de sauvegarde.

Points de sauvegarde

- L'objet Connection a deux nouvelles méthodes qui vous aident à gérer les points de sauvegarde :
  - setSavepoint(String savepointName) - Définit un nouveau point de sauvegarde et renvoie également un objet Savepoint.
  - releaseSavepoint(Savepoint savepointName) - Supprime un point de sauvegarde.

Points de sauvegarde

- Exemple :

```
try{
  conn.setAutoCommit(false);
  Statement stmt = conn.createStatement();
 // définition du point de sauvegarde
  Savepoint savepoint1 = conn.setSavepoint("Savepoint1");
 //requête correcte
  String requete1 = "INSERT INTO Etudiant VALUES (26, 'Alaoui', 'Ali')";
  stmt.executeUpdate(requete1);
  //requête fausse
  String requete2 = "INSERT INTOO Etudiant VALUES (27,'Omari', 'Omar')";
  stmt.executeUpdate(requete2);
  conn.commit(); // si il n y a pas d'erreur
}
catch(SQLException se){
  conn.rollback(savepoint1); // retour au point de sauvegarde en cas d'erreur
}
```