Formation

JAVA

m2iformation.fr

Java Database Connectivity

JDBC

Plan

- Introduction aux bases de données et au JDBC
- Composants de l'API JDBC
- Création d'une connexion à une base de données
- Instructions(requêtes)
- Lecture des résultats
- Gestion des transactions

Introduction aux bases de données et au JDBC

- Les bases de données permettent le stockage des données
- En manipulant une base de donnée il faut prendre en considération les concepts suivants:
  - Création de la connexion avec la base
  - création des instructions à exécuter dans la base
  - Avoir un retour de données qui représentent les résultats

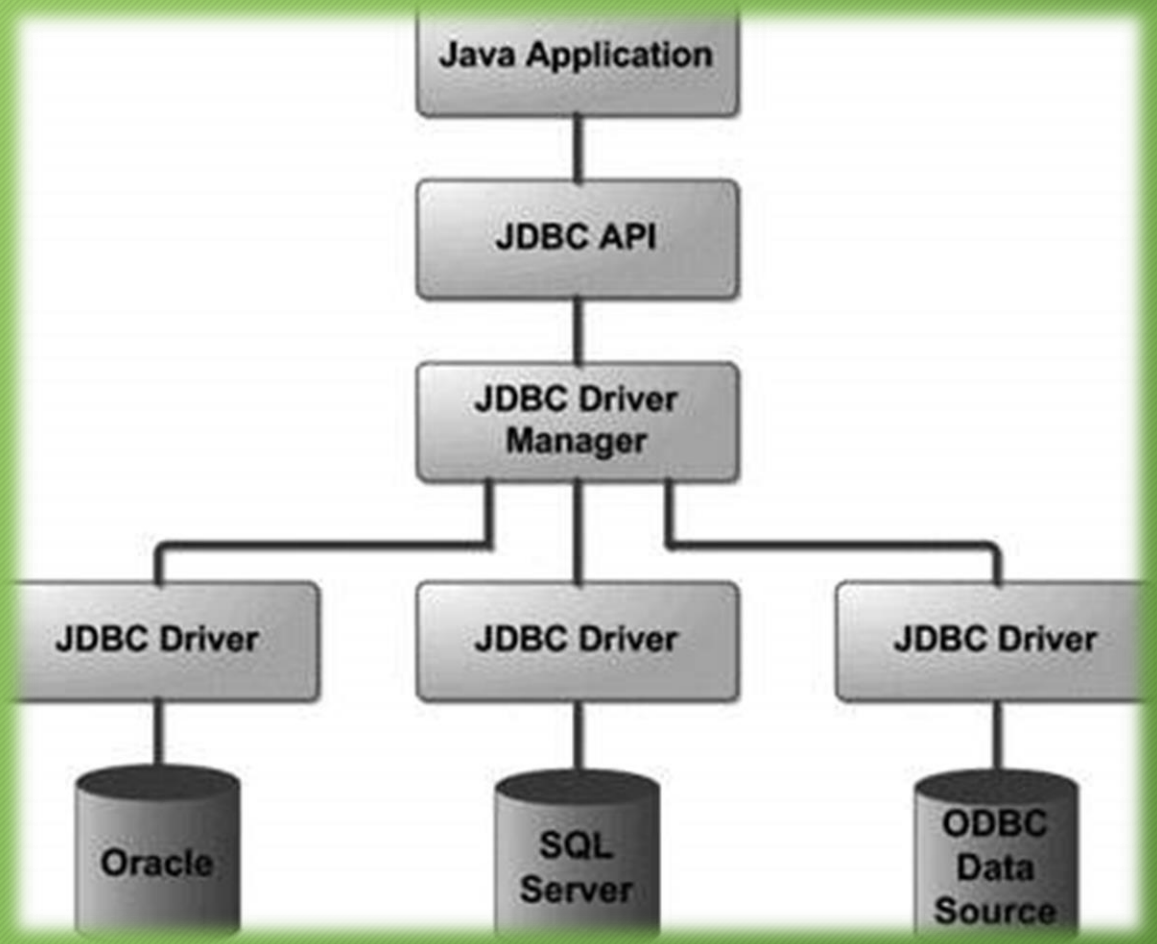
Introduction

Pour permettre d'établir une connexion avec la base de données et la création des requêtes SQL nous avons besoin d'un outil qui permet de relier l'application à la base, cet outil est une librairie d'APIs appelée JDBC qui fournit un support pour toutes les opérations effectuées sur la base

Introduction

JDBC (Java Database Connectivity) est une API Java standard permettant d'interagir avec des bases de données relationnelles à partir de Java. **JDBC** dispose d'un ensemble de classes et d'interfaces pouvant être utilisées à partir d'une application Java et communiquer avec une base de données sans avoir à apprendre les détails du SGBDR ni à utiliser des pilotes JDBC spécifiques à une base de données.

Composants de l'API JDBC



Composants de l'API JDBC

❑**DriverManager:**

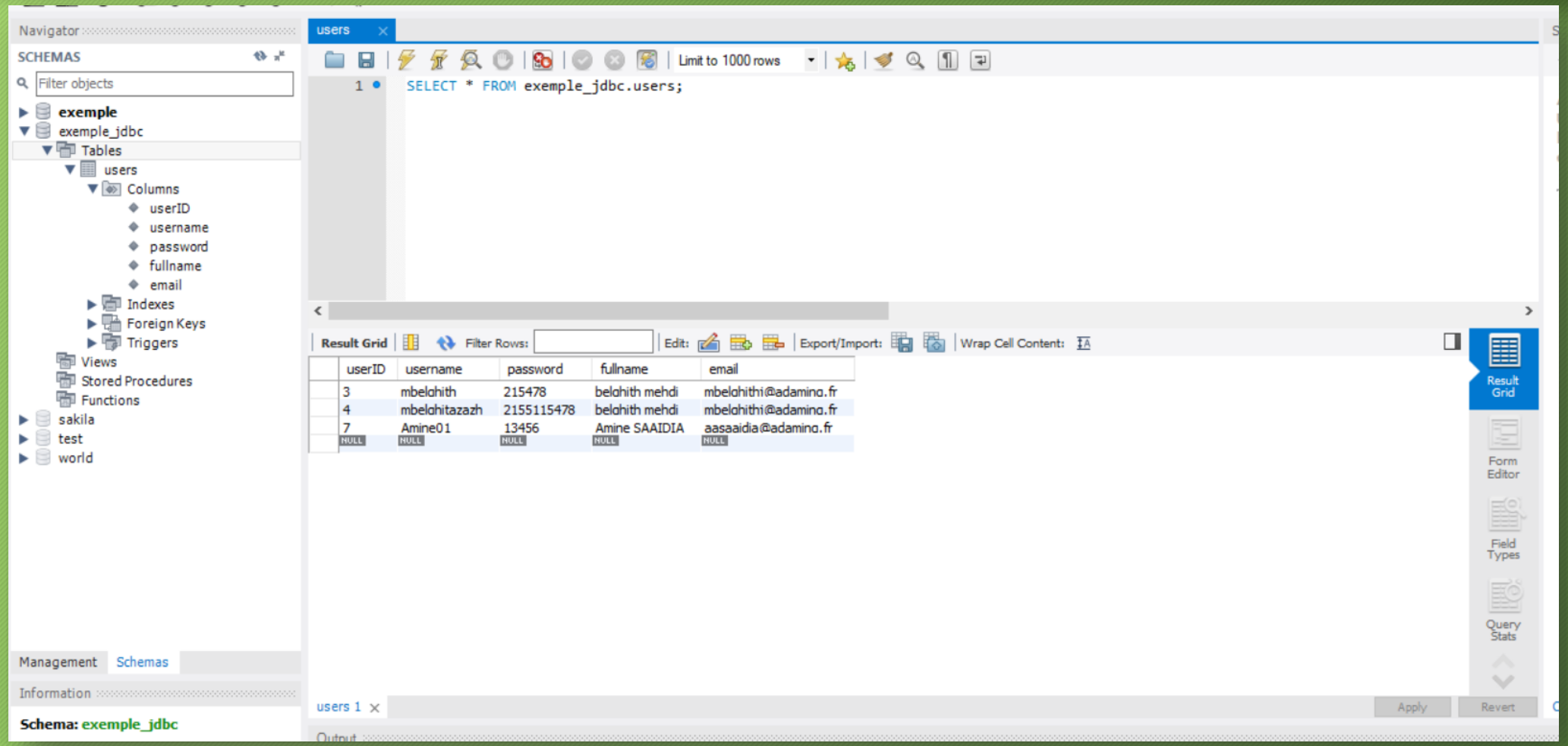
Cette classe gère une liste de pilotes pour la base de données, fait correspondre les requêtes de connexions en provenance de l'application Java avec le pilote propre à la base qui permet d'établir la connexion avec la base

- Les Pilotes JDBC est une bibliothèque de classes java qui permet, à une application java, de communiquer avec un SGBD via le réseau en utilisant le protocole TCP/IP
- Chaque SGBD possède ses propres pilotes JDBC

Composants de l'API JDBC

```
package com.adaming.utils;

import java.sql.Connection;

public class Connexion {
    private final static String dbURL = "jdbc:mysql://localhost:3306/exemple_jdbc";
    private final static String username = "root";
    private final static String password = "root";

    public static Connection conn;

    public static void seConnecter() {
        // Se connecter à la BD
        try {
            conn = DriverManager.getConnection(dbURL, username, password);
            if (conn != null) {
                System.out.println("Connecté");
            } else {
                System.out.println("Connexion échouée");

            }
```

Composants de l’API JDBC



Composants de l'API JDBC

- **Driver**:Cette interface gère la communication avec le serveur de la base de données, en pratique il n'y a pas d'interaction directe avec les objets **Driver** mais plutôt avec des objets **DriverManager** qui se chargent de ce type d'objet. Normalement, une fois le pilote chargé, le développeur n'a pas besoin de l'appeler explicitement.

- **Connection**:Cette interface fournit un support de toutes les méthodes de manipulation d'une base, toute communication avec la base se fait à travers un objet de type **Connection**

Composants de l'API JDBC

- **Statement**:

Les objets crées à partir de cette interface sont utilisés pour soumettre les requêtes SQL à la base de donnée. Encapsule une instruction SQL qui est transmise à la base de données pour être analysée, compilée, planifiée et exécutée.

- **ResultSet**:

Ces objets permettent de sauvegarder et manipuler les données récupérées depuis la base après l'exécution d'une requête SQL à travers des objets **Statement**

Le ResultSet représente un ensemble de lignes extraites en raison de l'exécution de la requête.

Création d’une connexion avec la BD

1. Préciser le type de driver que l'on veut utiliser: le Driver permet de gérer l'accès à un type particulier de SGBD
2. Récupérer un objet « Connection » en s'identifiant auprès du SGBD et en précisant la base utilisée
3. A partir de la connexion, créer un « statement » (état) correspondant à une requête particulière puis exécuter ce statement au niveau du SGBD et finalement fermer le statement
4. Se déconnecter de la base en fermant la connexion

Instruction Simple

❑Classe Statement

- ResultSet executeQuery(String ordre)
  - Exécute un ordre de type SELECT sur la base
  - Retourne un objet de type ResultSet contenant tous les résultats de la requête
- int executeUpdate(String ordre)
  - Exécute un ordre de type INSERT, UPDATE, ou DELETE
  - void close()

- Ferme l'état

Instruction paramétrée

❑Classe PreparedStatement

- Avant d'exécuter l'ordre, on remplit les champs avec
  - void set[Type](int index, [Type] val)
  - Remplit le champ en ième position définie par index avec la valeur val de type [Type]
  - [Type] peut être : String, int, float, long ...
  - Ex : void setString(int index, String val)
- ResultSet executeQuery()
  - Exécute un ordre de type SELECT sur la base
  - Retourne un objet de type ResultSet contenant tous les résultats de la requête
- int executeUpdate()
  - Exécute un ordre de type INSERT, UPDATE, ou DELETE

Instruction paramétrée

```
String sql = "INSERT INTO Users (username, password, fullname, email) "
        + " VALUES (?, ?, ?, ?)";

//Statement statement = Connexion.conn.createStatement();
PreparedStatement statement = Connexion.conn.prepareStatement(sql);

statement.setString(1, util.getUsername());
statement.setString(2, util.getPassword());
statement.setString(3, util.getFullname());
statement.setString(4, util.getEmail());

int rows = statement.executeUpdate();
if (rows > 0) {
    System.out.println("A new user was inserted successfully!");
}
```

Lecture des résultats

- Pour parcourir un **ResultSet**, on utilise sa méthode **next()** qui permet de passer d'une ligne à l'autre. Si la ligne suivante existe, la méthode **next()** retourne true. Si non elle retourne false.
- Pour récupérer la valeur d'une colonne de la ligne courante du **ResultSet**, on peut utiliser les méthodes **getInt**(colonne),**getString**(colonne), **getFloat**(colonne), **getDouble**(colonne), **getDate**(colonne), etc… colonne représente le numéro ou le nom de la colonne de la ligne courante

Lecture des résultats

- **TYPE_FORWARD_ONLY :** Constante indiquant le type d'un objet ResultSet dont le curseur peut uniquement avancer.
- **TYPE_SCROLL_INSENSITIVE** : Constante indiquant le type d'un objet ResultSet qui peut défiler, mais n'est généralement pas sensible aux modifications des données sous-jacentes du ResultSet.
- **TYPE_SCROLL_SENSITIVE** : Constante indiquant le type d'un objet ResultSet pouvant défiler et généralement sensible aux modifications des données sous-jacentes à ResultSet.
- **CONCUR_READ_ONLY :** La constante indiquant le mode de concurrence pour un objet resultSet qui ne peut pas être mis à jour.
- **CONCUR_UPDATABLE :** Constante indiquant le mode d'accès simultané d'un objet ResultSet pouvant être mis à jour.

Lecture des résultats

```
    Connexion.seConnecter();
    String sql = "SELECT * FROM Users where userID= ?";

    PreparedStatement statement = Connexion.conn.prepareStatement(sql);
    statement.setLong(1, idUser);
    ResultSet result = statement.executeQuery();

    while (result.next()) {


        u = new Users(result.getLong(1), result.getString(2), result.getString(3), result.getString(4), result.getString(5));
    }

    Connexion.seDeconnecter();

} catch (SQLException e) {
```

Gestion des transactions

Une **transaction** est un ensemble d'actions qui est effectué en entier ou pas du tout.

Les transactions permettent de contrôler quand est ce que les changements effectués dans la base seront pris en compte en traitant des groupes de requêtes comme étant une seule unité logique (dans le sens où une instruction échoue, toute la transaction échoue).

- **Commit & Rollback**

Après avoir fini avec les changements, pour les commettre , on fait appel de la méthode commit() sur un objet de type connnection

conn.**commit();**

Dans le cas contraire , si nous voulons annuler les changements effectués avec l'objet conn de connection, on fait appel à la méthode rollback()

conn.**rollback();**

Gestion des transactions

La **transaction** est un concept important en SQL.

Exemple : une personne envoie une somme de 1 000 euro sur son compte, deux actions se produisent donc dans la base de données:

Débit de 1 000 euro sur le compte d'une personne
Créditez 1 000 euro sur le compte de la personne B.

Et la transaction est considérée comme réussie si les deux étapes ci-dessus sont implémentées avec succès. Au contraire, si l'une des deux étapes échoue, la transaction doit être considérée comme infructueuse et nous devrons procéder à un retour en arrière par rapport au statut antérieur.

Composants de l’API JDBC

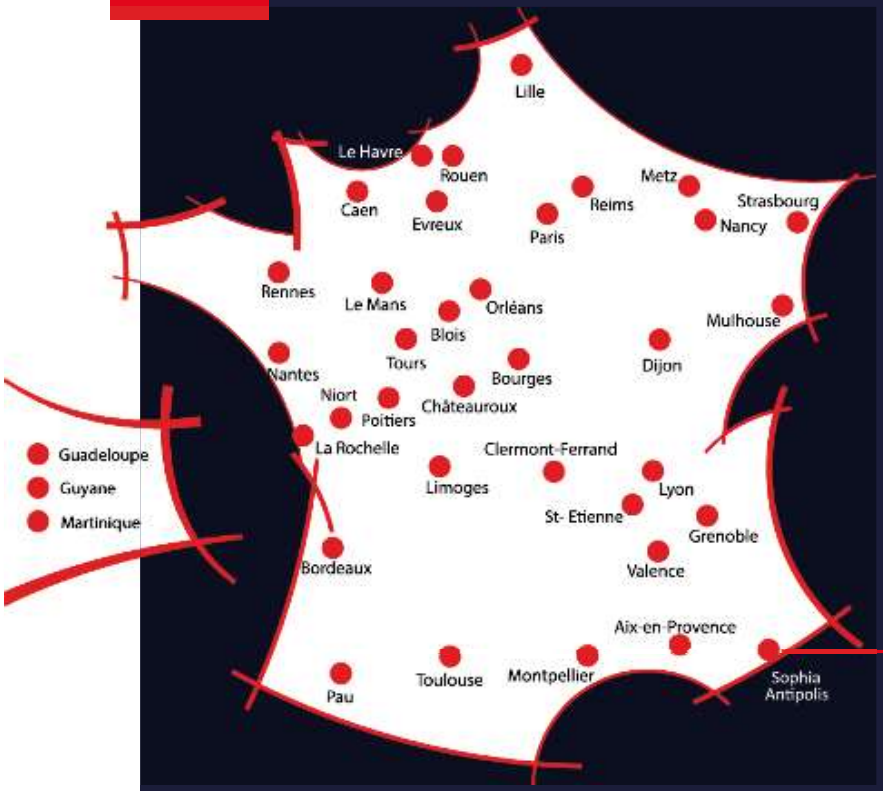
```
public class UsersDAO {

    public void inserer(Users util) {
        try {
            Connexion.seConnecter();

            String sql = "INSERT INTO Users (username, password, fullname, email) "
                    + " VALUES (?, ?, ?, ?)";

            PreparedStatement statement = Connexion.conn.prepareStatement(sql);

            statement.setString(1, util.getUsername());
            statement.setString(2, util.getPassword());
            statement.setString(3, util.getFullname());
            statement.setString(4, util.getEmail());

            int rows = statement.executeUpdate();
            if (rows > 0) {
                System.out.println("A new user was inserted successfully!");
            }

            Connexion.seDeconnecter();
        } catch (SQLException e) {

            e.printStackTrace();
```

Composants de l'API JDBC

```
public List<Users> findAll() {
    List<Users> liste = new ArrayList<>();
    try {

        Connexion.seConnecter();
        String sql = "SELECT * FROM Users";

        PreparedStatement statement = Connexion.conn.prepareStatement(sql);

        ResultSet result = statement.executeQuery();

        while (result.next()) {


            liste.add(new Users(result.getLong(1), result.getString(2), result.getSt
        }

        Connexion.seDeconnecter();

    } catch (SQLException e) {

        e.printStackTrace();
    }
    return liste;
```

Création d'une connexion avec la BD

```
public Users findOne(Long idUser) {
    Users u = null;
    try {

        Connexion.seConnecter();
        String sql = "SELECT * FROM Users where userID= ?";

        PreparedStatement statement = Connexion.conn.prepareStatement(sql);
        statement.setLong(1, idUser);
        ResultSet result = statement.executeQuery();

        while (result.next()) {


            u = new Users(result.getLong(1), result.getString(2), result.getString(3), result.getString(4), result.getString(5));
        }

        Connexion.seDeconnecter();

    } catch (SQLException e) {

        e.printStackTrace();
    }
```



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr