# JDBC (accès aux données)

## Principes

Quel que soit l'environnement, lorsque l'on souhaite accéder à des données stockées dans un SGBD-R, à partir d'un environnement de programmation, il est nécessaire de s'attacher à la démarche suivante :

1. Ouvrir une connexion à la BdD ;
2. Pour chaque requête à exécuter :
    1. Assembler la requête ;
    2. Demander l'exécution de la requête ;
    3. Traiter le résultat de la requête (souvent à l'aide d'une boucle) ;
3. Fermer la connexion à la BdD.

Java DataBase Connectivity (**JDBC**) est un logiciel intermédiaire (**middleware**) qui, dans une architecture logicielle, s'insère entre un client et un serveur de données.
Son but est de fournir des services normalisés pour cet accès aux données, indépendamment du SGBD utilisé. Dialecte SQL, jeu de caractères, méthodes d'accès, autant de caractéristiques qui présentent couramment des différences notables lorsque l'on passe d'un produit à un autre, d'un éditeur à un autre. Le middleware d'accès aux données prend en charge ces différences et veut rendre transparente l'implémentation effective.

Grace à JDBC, on peut donc en principe, écrire un programme d'accès aux données qui n'aura besoin que d'adaptations mineures si on décide de changer de SGBD, en cours de route, durant le cycle de vie d'une application. JDBC est un middleware spécifique au monde Java. Il a été nommé ainsi en référence à son homologue et prédécesseur dans le monde Microsoft : ODBC (Open DataBase Connectivity). ODBC a connu un grand succès. JDBC, dans son sillage, aussi.

## API courante JDBC

Toutes les spécificités décrites ici nécessitent de référencer le package java.sql

```
import java.sql.*;
```

### Ouverture d'une connexion

```
String url = "jdbc:mysql://turlututu.com/ma_base_de_données";
Connection connect = DriverManager.getConnection(url, user, passwd);
```

### Demande d'execution d'une requête

```
Statement stmt = connect.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
```

### Traitement du résultat

```
while (rs.next()) {
        int x = rs.getInt("a");
        String s = rs.getString("b");
        float f = rs.getFloat("c");
}
```

Note : il existe une méthode **get**xxx pour chaque type standard Java.

### Fermeture d'une connexion

```
connect.close();
```

## Exceptions Java

Une grande partie du code montré ci-dessus est susceptible de déclencher la survenue d'Exceptions Java du type SQLException. En conséquence l'environnement de développement refusera d'exécuter ce code si la survenue des exceptions n'est pas gérée dans le code lui-même. Pour ce faire, et dans une version simple, encadrer le bloc d'accès aux données d'un **try-catch** :

```
try {
  //... flot d'instructions pouvant générer une exception ...
}
catch (SQLException e) {
  // action en cas de survenue d'une Exception
  System.out.println (e.getMessage());
}
```

## Pilote

Selon le SGBD cible, pour que JDBC puisse fonctionner correctement, il faudra intégrer au projet le pilote JDBC adapté. On procèdera en deux temps :

1. D'abord, déposer le pilote adapté (sous la forme d'un fichier jar) dans un dossier du projet (dossier souvent nommé **lib**);
2. Ensuite, indiquer à l'IDE que ce pilote doit être intégré, comme une librairie, à la fabrication de l'exécutable.

Par exemple, pour MySql actuellement, il faut télécharger le fichier mysql-connector-java-8.0.30.jar (il est inclus dans l'archive téléchargeable ici https://dev.mysql.com/downloads/file/?id=513221) mais il vous sera aussi fourni directement dans vos ressources pédagogiques.

## Opérations à réaliser pour NetBeans

Voir NetBeans-Configuration - Pilote JDBC

From:
https://wiki.siochaptalqper.fr/ - **Wiki SIO Chaptal**

Permanent link:
**https://wiki.siochaptalqper.fr/doku.php?id=bloc2:prog:poo:jdbc**

Last update: **2023/10/12 11:04**