

# BUT Informatique 1<sup>ère</sup> année

## R2.02: Développement d'applications avec IHM

Cours 4 : accès aux bases de données

Sébastien Lefèvre  
[sebastien.lefevre@univ-ubs.fr](mailto:sebastien.lefevre@univ-ubs.fr)

# BD & Java : JDBC

JDBC = Java DataBase Connectivity

API Java qui permet la connexion à une BD et l'exécution de requêtes

Pour qu'un programme Java puisse accéder à une BD, **c'est simple !**

Il faut :

1. Un SGBD (Apache Derby 100% Java, MySQL, etc)
2. Un pilote (driver) JDBC pour ce SGBD particulier (inclus avec Derby, Connector/J for MySQL, etc)
3. Du code Java qui utilise le package java.sql

# Pilotes JDBC

4 types de pilotes :

- Type 1: implémentent JDBC sous la forme d'un mapping vers une autre API d'accès aux données (par exemple ODBC) ; dépendent d'une bibliothèque native, portabilité limitée ; ex: JDBC-ODBC Bridge
- Type 2: écrits partiellement en Java et en code natif ; utilisent une bibliothèque native spécifique à la BD ; portabilité limitée ; ex: Oracle OCI
- Type 3: pur Java et communiquent avec un serveur via un protocole indépendant de la BD ; le serveur fait suivre la requête du client vers la source de données
- Type 4: pur Java et implémentent le protocole réseau pour une source de données spécifiques ; le client se connecte directement à la source de données ; ex: MySQL Connector/J, JavaDB, etc

**Installer un pilote JDBC = télécharger le pilote + l'ajouter au CLASSPATH**

Si le pilote n'est pas de type 4, souvent besoin d'installer une API côté client

# Connexion à une BD

## Utilisation de DriverManager ou DataSource

```
Connection c = DriverManager.getConnection(String url)
```

ou

```
Connection c = DriverManager.getConnection(String url, String user,  
String password)
```

avec url qui précise le SGBD utilisé et son emplacement, par exemple

jdbc:mysql://localhost:3306/myDB

jdbc:derby:testdb;create=true

la forme exacte de l'URL dépend du SGBD / pilote JDBC !

# Exécution d'une requête

## Création d'un Statement à partir d'un objet Connection

```
Statement stmt = connection.createStatement();
```

**ou**

```
PreparedStatement stmt = connection.prepareStatement("SELECT  
NOM,PRENOM FROM TABLE WHERE ANNEE=?");  
stmt.setInt(1,2022);
```

## Exécution d'une requête

```
ResultSet rs = stmt.executeQuery(query)
```

où String query est une requête SQL de type SELECT

**ou** `stmt.executeUpdate(query)`

si la requête SQL est de type INSERT, DELETE, UPDATE

# Accéder aux résultats d'une requête

Le ResultSet contient un ensemble de résultats qu'on peut parcourir via un itérateur :

```
while (rs.next()) {  
    String nom=rs.getString("NOM");  
    int age=rs.getString("AGE");  
    System.out.println(nom+" "+age);  
}
```

# Fermer la connexion

Il faut veiller à appeler la méthode `close` quand on a fini d'utiliser les objets `Connection`, `Statement`, ou `ResultSet`.

On peut aussi utiliser un bloc `try-with-resources` :

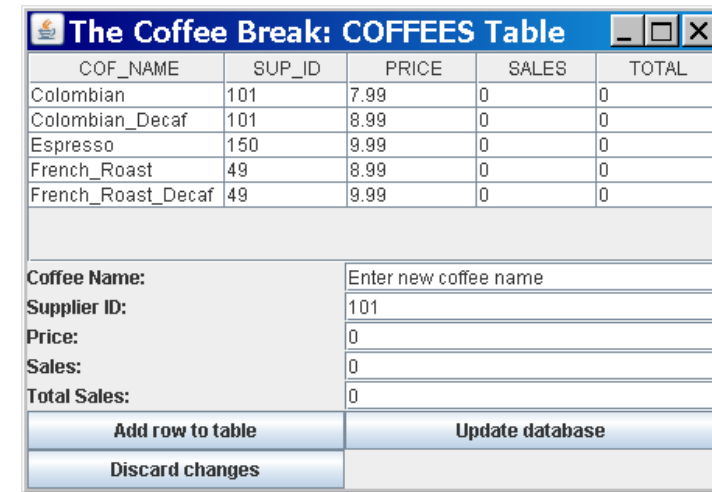
```
try(Statement stmt = connection.createStatement()) {  
    ...  
}  
catch(SQLException e) { ... }
```

qui déclare et fermera automatiquement la ressource `stmt` lorsque le bloc `try` se terminera (avec ou sans exception)

# BD et IHM

Swing permet facilement d'afficher le contenu d'une table ou le résultat d'une requête

<https://docs.oracle.com/javase/tutorial/jdbc/basics/jdbcswing.html>



COF_NAME	SUP_ID	PRICE	SALES	TOTAL
Colombian	101	7.99	0	0
Colombian_Decaf	101	8.99	0	0
Espresso	150	9.99	0	0
French_Roast	49	8.99	0	0
French_Roast_Decaf	49	9.99	0	0

Coffee Name:	Enter new coffee name
Supplier ID:	101
Price:	0
Sales:	0
Total Sales:	0

Add row to table	Update database
Discard changes	

Pour cela, on doit :

1. Implémenter l'interface `javax.swing.event.TableModel`
2. Implémenter l'interface `javax.sql.RowSetListener`
3. Disposer les composants Swing, ajouter les écouteurs, etc



# Un peu de pratique (tentative !)

<https://db.apache.org/derby/>

1. Téléchargement du SGBD
2. Compilation et exécution du fichier exemple WwdEmbedded.java
3. Ecriture d'un code spécifique

Exemple