

# Travaux pratiques avec JDBC

---

## 1 Installer MySQL (option ISL)

### 1.1 Installation.

Commencez par chercher et installer le système de gestion de base de données MySQL (commandes ci-dessous). Une fois installé, lancez les démons de ce S.G.B.D. et finalement créez vous une base de données.

```
# yum install mysql mysql-server mysql-connector-odbc
# /etc/init.d/mysqld start
# mysqladmin create dbessai
```

### 1.2 Création de la base.

Connectez-vous en root sur la base « mysql » et donnez vous des droits sur la base d'essai :

```
# mysql
mysql> use mysql;
mysql> grant all privileges on dbessai.*
    to 'votre_login_unix'@'localhost'
    identified by 'votre_mot_de_passe_base_de_données'
    with grant option ;
mysql> flush privileges ;
mysql> exit ;
```

**Remarque** : MySQL identifie les utilisateurs par un triplet (login, machine-cliente, mot-de-passe). Le caractère % dénote toutes les machines.

Abandonnez le compte root et prenez contact avec mysql en vous connectant (commande `mysql -p`), en créant une ou plusieurs tables simples, en ajoutant des données et en programmant quelques requêtes.

## 2 Utiliser MySQL (options ARO et TAL)

Un serveur MySQL est installé sur le serveur du département (machine sol) et quinze bases de données sont disponibles (bases de db01 à db15 et utilisateurs user01 à user15). Vous pouvez utiliser le client MySQL (commande `mysql`), à partir de votre machine pour vous connecter sur le serveur sol.

```
pcXX $ mysql -u userXX -h sol -p
password: ... le mot de passe MySQL de userXX ..
mysql> use dbXX;
mysql> ... création d'une table ...
```

Vous pouvez maintenant créer une ou plusieurs tables simples, en ajoutant des données et en programmant quelques requêtes.

### 3 Pilotes JDBC.

Allez sur le site [www.mysql.com](http://www.mysql.com)<sup>1</sup> et récupérez les pilotes JDBC 5.x (aussi disponible ici<sup>2</sup>). Préparez un nouveau projet Java dans Eclipse et ajoutez le fichier .jar du pilote au *buildpath* de votre projet.

### 4 Premiers programmes JDBC

Pour organiser notre travail, nous allons commencer par créer deux classes :

- Une classe `JdbcTools` qui va contenir le code générique indépendant de notre domaine d'application.
- Une classe `Dao` (*Data Access Object*) qui va contenir le code JDBC qui a la charge de sauvegarder et de récupérer les données (javaBeans) de notre application. Cette classe étend la première.

Nous allons **immédiatement**, créer deux tests unitaires. Le premier qui étend la classe `JdbcTools` (`JdbcToolsTest`) et le second qui teste la classe `Dao` (`DaoTest`).

#### 4.1 Préparer la couche des outils

L'objectif de cette classe est d'offrir une série d'outils afin de faciliter l'écriture des classes DAO.

- Commencez par créer dans la classe `JdbcTools` les propriétés, les setters et les getters pour représenter le nom du pilote, l'URL de connexion, l'identifiant et le mot de passe.
- Créez ensuite dans la classe `JdbcTools` les deux méthodes ci-dessous et prévoyez un test unitaire de ces méthodes (dans `JdbcToolsTest`).

```
public void init() { ... }  
public void close() { ... }
```

- Créez ensuite dans la classe `JdbcTools` les deux méthodes de création/destruction des connexions. Prévoyez un test unitaire de ces méthodes (dans `JdbcToolsTest`).

```
public Connection newConnection() throws SQLException { ... }  
public void quietClose(Connection c) { ... }
```

- Vous pouvez maintenant ajouter la méthode `executeUpdate` qui va vous offrir un moyen simple de détruire ou créer des tables et des lignes.

```
public int executeUpdate(String query) throws SQLException { ... }
```

Cette méthode doit avoir la structure ci-dessous. Il est **très important** de bien capter les exceptions afin de libérer la connexion.

---

<sup>1</sup><http://www.mysql.com>

<sup>2</sup>ress-jdbc

```

try {
    // créer une connexion
    // préparer l'instruction
    // exécuter l'instruction
} catch (SQLException e) {
    // renvoyer cette exception
}
} finally {
    // fermer la connexion
}

```

- Nous aurons souvent besoin de paramétrer l'exécution d'une mise à jour. Pour ce faire, ajoutez à la méthode précédente une série d'arguments. Cette nouvelle version est basée sur l'utilisation des PreparedStatement.

```

public int executeUpdate(String sql, java.io.Serializable... parameters)
    throws SQLException { ... }

```

Attention : vous devrez, pour chaque paramètre, étudier son type pour utiliser la méthode setType la plus appropriée (contentez-vous d'étudier les cas numériques et chaînes de caractères).

## 4.2 Lister des lignes

- Partons du principe que vous avez une table qui représente une personne (vous pouvez, bien entendu prendre n'importe quel autre exemple). Prenez soin de placer une clef primaire dans cette table (et dans le javaBean). Créez dans la classe Dao la méthode

```

public Collection<Person> findPersons() throws DaoException { ... }

```

Cette méthode doit avoir la structure ci-dessous. il est **très important** de bien capter les exceptions afin de libérer la connexion.

```

try {
    // 1. créer une connexion
    // 2. préparer l'instruction
    // 3. exécuter la requête
    // 4. lire le résultat
} catch (SQLException e) {
    // 5. construire l'exception DAO
    // 6. renvoyer cette exception
}
} finally {
    // 7. fermer la connexion
}
// renvoyer le résultat

```

Écrivez le test unitaire de cette méthode (vous pouvez utiliser la méthode executeUpdate pour créer plusieurs personnes).

- Vous pouvez remarquer que dans cette méthode, seule la ligne 4 est spécifique à notre application. Nous pouvons généraliser cette méthode en créant (dans JdbcTools) la version suivante :

```

public <T> Collection<T> findBeans(String sql, ResultSetToBean<T> mapper)
    throws SQLException { ... }

```

dans laquelle l'interface `ResultSetToBean` est définie par

```
// construire un bean de type T à partir d'un ResultSet
interface ResultSetToBean<T> {
    T toBean(java.sql.ResultSet rs) throws SQLException;
}
```

La méthode `findBeans` ainsi définie permet d'interroger n'importe quelle table pour construire le `JavaBean` équivalent.

- Dans un deuxième temps vous pouvez ajouter à la méthode `findBeans` une série de paramètres.

### 4.3 Insérer une ligne

Vous pouvez maintenant enrichir votre classe `Dao` pour offrir une méthode d'ajout d'une personne. Prévoyez un test unitaire de cette méthode ainsi qu'un enrichissement du test de la méthode précédente (vous pouvez maintenant ajouter des lignes).

```
public void addPerson(Person p) throws DaoException { ... }
```

Testez notamment les cas d'erreur dus à la contrainte de clef primaire.

**Conseil** : Utilisez la version `UPDATABLE` de la classe `java.sql.ResultSet`. C'est plus simple.

### 4.4 Détruire ou modifier des lignes

- Écrivez dans la classe `Dao` la méthode d'effacement d'une personne :

```
public void removePerson(int id) throws DaoException { ... }
```

- Toujours dans la classe `Dao`, écrivez la méthode de mise à jour d'une personne. Utilisez la version `UPDATABLE` de la classe `java.sql.ResultSet`.

```
public void updatePerson(Person p) throws DaoException { ... }
```

Ralentissez votre méthode (avec un `sleep` entre la requête et un éventuel `commit`), et testez les mises à jour concurrentes avec l'interface ligne de commande de `mysql`. **Attention** : dans sa version de base, `MySQL` ne gère ni les transactions, ni la pose de verrous. Pour avoir accès à ces fonctionnalités, vous devez utiliser un codage particulier (appelé `InnoDB`) pour vos tables. Pour ce faire, utilisez la clause ci-dessous :

```
ALTER TABLE votre_table TYPE=InnoDB;
```

- Dans un deuxième temps, vous pouvez concevoir dans la classe `JdbcTools` une version générale de la méthode `updatePerson` :

```
public <T> void updateBean(String sql,
    BeanToResultSet<T> mapper,
    T theBean,
    java.io.Serializable... parameters)
    throws SQLException { ... }
```

## 5 Utiliser les métas informations et les RowSet

Écrire (dans la classe `JdbcTools`) une méthode générale qui prend en paramètre une requête et qui imprime une page HTML contenant la requête et le résultat de la requête.

```
public void makeTable(String sql, PrintWriter stdout)
    throws SQLException { ... }
```

Vous pouvez utiliser les `RowSet`<sup>3</sup> pour traiter cette question. Pour ce faire, vous avez besoin d'une implantation des interfaces `RowSet`. Depuis la version 1.5, ces implantations sont disponibles dans la JVM. Pour les trouver, vous pouvez essayer :

```
jar tvf $JAVA_HOME/jre/lib/rt.jar | grep -i RowSetImpl
```

Il est **fortement conseillé** de s'inspirer de ce guide<sup>4</sup>.

---

<sup>3</sup><http://java.sun.com/developer/Books/JDBCTutorial/chapter5.html>

<sup>4</sup><http://download.oracle.com/javase/tutorial/jdbc/basics/jdbcrowset.html>