JDBC et l'accès aux bases de données

Présentation

JDBC (Java DataBase Connectivity) = ensemble d'API (« interface de programmation ») permettant à un programmeur Java, d'accéder aux informations stockées dans une **base de données** (la plupart du temps **relationnelles**).

Démarche d'utilisation :

1. Connexion à la base de données visée
2. Envoi de commandes SQL
3. Réception, pour traitement, des résultats des requêtes

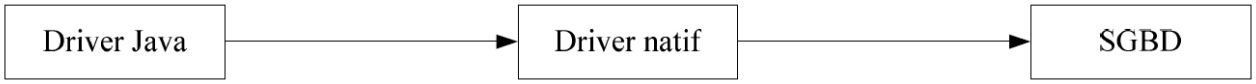
Il est donc nécessaire de connaître suffisamment le **SQL** pour pouvoir utiliser **JDBC** → on dit qu'il s'agit d'une interface de « **bas niveau** » (contrairement à **JPA** qui est une interface de « **haut niveau** » faisant directement le pont entre « tuple » et « objet java »).

Notion de driver

Driver = élément qui sert de pont entre **l'application Java** et le **SGBD** (la « base de données »)

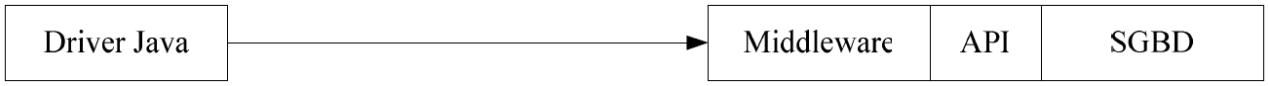
Il en existe 4 types :

- **Type 1** : le driver Java mappe simplement un driver natif (donc par exemple écrit en C) :



- **Type 2** : le driver est partiellement écrit en Java et partiellement avec du code natif, et utilise en fait une librairie native (C ou C++ par exemple) :

- **Type 3** : le driver est à présent « full Java » et utilise un middleware qui réalise l'accès au SGBD (la communication driver/middleware est indépendante du SGBD et utilise un protocole indépendant de la base données ; le middleware convertit ensuite les appels JDBC dans le protocole du SGBD) :



- **Type 4** : le driver, entièrement écrit en Java, implémente complètement le protocole d'accès à la base de données (il est donc dédié à un type de SGBD) :

Il existe de nombreux drivers JDBC adaptés aux SGBD les plus courant : **MySQL**, **Oracle**, **Microsoft SQL Server**, **CSV Files**, **PostgreSQL**, …

Etablir une connexion

Les classes Java relatives à JDBC se trouvent dans le JDK dans le package **java.sql**.

Le chargement du driver : la classe Class

Il faut tout d'abord localiser le driver correspondant au SGBD visé. Celui-ci est représenté par une classe Java. Par exemple, la classe associée au **driver MySQL** est

com.mysql.cj.jdbc.Driver

Pour les autres drivers, il faut se renseigner dans la documentation fournie.

Pour charger le driver en mémoire, on doit utiliser la classe **Class** qui possède notamment les méthodes :

- **public static native Class forName(String className) throws ClassNotFoundException** → permet de localiser et de charger la définition d'une classe dont on passe le nom en paramètre → il s'agit d'un mécanisme d'**introspection** → retourne un objet instance de **Class**

- **getName()**, **getDeclaredFields()**, **getDeclaredMethods()**, … → permettent d'analyser la classe

Pour charger en mémoire le driver JDBC associé à **MySQL**, if faut donc taper :

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

La classe **com.mysql.cj.jdbc.Driver** est

- chargée en mémoire, et aussi
- enregistrée dans la liste des drivers disponibles

Bien sûr, cette classe doit être trouvée quelque part… Et c'est la plupart du temps le fournisseur de la base de données qui la fournit. Pour MySQL, il faut télécharger cette classe sous forme d'un **jar** (« **mysql-connector-j-8.0.32.jar** ») et le monter dans votre projet.

La gestion des drivers : la classe DriverManager

La classe **DriverManager**

- gère la liste des drivers disponibles pour l'application Java en cours d'exécution
- fait le lien entre le programmeur Java et les drivers JDBC
- va permettre d'établir une connexion entre la base de données visée et le driver associé

La méthode (de classe) de **DriverManager** qui permet d'établir cette connexion est la suivante :

- **public static synchronized Connection getConnection(String url, String user, String password) throws SQLException**

Cette méthode reçoit, outre le login et le mot de passe de l'utilisateur du SGBD, une URL précisant la base de données sur laquelle on désire se connecter. Plus précisément, cette URL est de la forme

jdbc:<nom du sous-protocole>:<nom de la base pour ce protocole>

Dans le cas d'une connexion à une base de données MySQL, appelée « **PourStudent** » et située sur une machine distante d'adresse IP 192.168.228.167, cette URL est

jdbc:mysql://192.168.228.167/PourStudent

La connexion à cette base de données se réalise donc en tapant

```
Connection con =
DriverManager.getConnection("jdbc:mysql://192.168.228.167/PourStudent"
,"Student","PassStudent1_");
```

Pour un utilisateur dont le login est « **Student** » et le mot de passe est « **PassStudent1_** ».

La connexion : l'interface Connection

L'objet retourné par la méthode **getConnection()** de **DriverManager** implémente l'interface **Connection**, il matérialise pour le programmeur Java la connexion établie avec la base de données → il faudrait plutôt parler de « **session** » car cet objet mémorise également les commandes SQL en cours et leurs résultats.

Parmi les méthodes de l'interface **Connection**, on peut citer :

- **public abstract void close() throws SQLException** → permet de fermer la session
- **public abstract void setAutoCommit(bool autoCommit) throws SQLException** → par défaut, toute opération de mise à jour est l'objet d'un commit. Cette méthode permet d'imposer qu'il n'en soit pas ainsi. La gestion des transactions est alors à charge du programmeur :
  - **public abstract void commit() throws SQLException**
  - **public abstract void rollback() throws SQLException**
- **public abstract Statement createStatement() throws SQLException** → permet de créer l'instruction SQL à exécuter sur la base de données

Exécution d'une commande SQL

Une fois que l'on dispose de l'objet instance de **Connection**, il est alors possible d'instancier un objet représentant une instruction SQL, un tel objet implémente l'interface **Statement** :

```
Statement instruction = con.createStatement();
```

Cet objet va permettre d'envoyer des commandes au SGBD. L'interface **Statement** dispose essentiellement des méthodes

- **public abstract ResultSet executeQuery(String sql) throws SQLException** → qui permet d'envoyer une requête de sélection, dont le résultat est une instance implémentant l'interface **ResultSet**
- **public boolean execute(String sql) throws SQLException** → qui permet d'exécuter une commande SQL quelconque → retourne true si la requête a fourni un ResultSet, false si la requête n'a fourni aucun résultat.
- **public abstract int executeUpdate(String sql) throws SQLException** → qui permet d'effectuer une commande SQL de mise à jour

La requête SQL est fournie aux méthodes via une simple String **sql**. Dans le cas d'un **execute**, il est possible de récupérer un **ResultSet** via la méthode

- public **Resultset getResultset**()

Tout objet implémentant l'interface **ResultSet**

- a pour but de contenir tous les tuples correspondant aux résultats d'une requête
- correspond à un curseur dans le jargon « SQL »
- correspond donc à une espèce de table dont les lignes correspondent aux tuples obtenus et les colonnes dépendent de la requête réalisée
- est un conteneur de tuples à déplacement séquentiel et à sens unique (du début à la fin)

et cet interface **ResultSet** dispose des méthodes

- **public abstract boolean next()** → qui permet de passer d'un tuple au suivant → retourne false lorsque l'on a dépassé le dernier tuple → à comparer à la notion d'itérateur du C++
- **public ResultSetMetaData getMetaData()** → qui permet d'obtenir les méta-données du ResultSet sous la forme d'un objet implémentant l'interface **ResultSetMetaData**
- **public Object getObject(int col)** → qui permet de récupérer, sous forme d'Object, le champ d'indice **col** du tuple courant du ResultSet
- … d'autres méthodes permettant de récupérer chaque champ de chaque tuple sous la forme d'un int, float, String, … (voir plus loin)

L'interface **ResultSetMetaData** dispose des méthodes

- **public int getColumnCount()** → qui retourne le nombre de colonnes du ResultSet
- **public String getColumnName(int col)** → retourne le nom de la colonne d'indice **col** du ResultSet
- **public String getTableName(int col)** → retourne le nom de table correspondant à la colonne d'indice **col** du ResultSet

Exemple de requête de sélection « brute » et de comptage de tuples

Supposons que nous disposons d'une base de données MySql appelée **PourStudent** qui contient la table **Personnes** où chaque tuple présente les champs suivants :

- **id** : INT(4) : clé primaire auto-incrémentée
- **Nom** : VARCHAR(20)
- **Prenom** : VARCHAR(20)
- **Poids** : FLOAT(4)
- **NbEnfants** : INT(4)
- **AnneeNaissance** : DATE

Cette table comporte actuellement 3 tuples :

```
# mysql -u Student -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.21 Source distribution

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use PourStudent;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from Personnes;
+----+---------+--------+-------+-----------+----------------+
| id | Nom     | Prenom | Poids | NbEnfants | AnneeNaissance |
+----+---------+--------+-------+-----------+----------------+
|  1 | Issier  | Paul   |  75.4 |         2 | 1975-05-11     |
```

```
|  2 | Coptere | Eli    |  81.7 |         0 | 2001-12-21     |
|  3 | Tombale | Pierre |  67.5 |         1 | 1983-02-15     |
+----+---------+--------+-------+-----------+----------------+
3 rows in set (0.00 sec)

mysql>
```

Dans l'exemple qui suit, le programme récupère et affiche les métadonnées de la table, tous les tuples, ainsi que le nombre de tuples présents.

Le code du programme de test (fichier **TestJDBC_MySql_Personnes.java**) est

```
import java.sql.*;

public class TestJDBC_MySql_Personnes
{
    public static void main(String[] args)
    {
        try
        {
            // Chargement du driver
            Class leDriver = Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Obtention du driver OK...");

            // Connexion a la BD
            Connection con =
DriverManager.getConnection("jdbc:mysql://192.168.228.167/PourStudent","Student","Pas
sStudent1_");
            System.out.println("Connexion à la BD PourStudent OK...");

            // Creation d'un objet Statement permettant d'executer une requete
            Statement stmt = con.createStatement();
            System.out.println("Creation du Statement OK...");

            // Execution d'une requete
            ResultSet rs = stmt.executeQuery("SELECT * FROM Personnes");
            System.out.println("ExecuteQuery OK...");
            System.out.println("Instruction SELECT * FROM Personnes");

            // Affichage des meta-donnees : nombre de colonnes
            System.out.println("Nombre de colonnes = " +
rs.getMetaData().getColumnCount());

            // Affichage des meta-donnees : noms des colonnes
            for (int j = 1; j <= rs.getMetaData().getColumnCount(); j++)
                System.out.print(rs.getMetaData().getColumnName(j) + "\t");
            System.out.println();

            // Affichage du contenu du ResulSet (du curseur)
            while (rs.next())
            {
                for (int j = 1; j <= rs.getMetaData().getColumnCount(); j++)
                    System.out.print(rs.getObject(j) + "\t");
```

```
                System.out.println();
            }
            System.out.println();

            // Autre exemple de requete
            // Comptage du nombre de tuples
            Statement instruc = con.createStatement();
            boolean retour = instruc.execute("select count(*) from Personnes");
            System.out.println("Instruction SELECT COUNT(*) sur Personnes");
            System.out.println("retour = " + retour);
            ResultSet rsc = instruc.getResultSet();
            if (rsc==null) System.out.println("Pas de resultset");
            else
            {
                System.out.println("Resultset récupéré");
                if (rsc.next())
                {
                    short nbre = rsc.getShort(1);
                    System.out.println("Nombre de tuples = " + nbre);
                }
            }

            // Fermeture de la connexion
            rs.close();
            rsc.close();
            con.close();
        }
        catch (ClassNotFoundException ex)
        {
            System.out.println("Erreur ClassNotFoundException: " + ex.getMessage());
        }
        catch (SQLException ex)
        {
            System.out.println("Erreur SQLException: " + ex.getMessage());
        }
    }
}
```

dont un exemple d'exécution fournit

```
# java TestJDBC_MySql_Personnes
Obtention du driver OK...
Connexion à la BD PourStudent OK...
Creation du Statement OK...
ExecuteQuery OK...
Instruction SELECT * FROM Personnes
Nombre de colonnes = 6
id    Nom         Prenom      Poids       NbEnfants   AnneeNaissance
1     Issier      Paul        75.4        2           1975-05-11
2     Coptere     Eli         81.7        0           2001-12-21
3     Tombale     Pierre      67.5        1           1983-02-15

Instruction SELECT COUNT(*) sur Personnes
retour = true
Resultset récupéré
Nombre de tuples = 3
```

```
#
```

On remarque que l'accès aux champs de chaque tuple se réalise à l'aide de la méthode **getObject(int col)** du **ResultSet**. Cette méthode retourne simplement un objet correspondant au tuple en cours pour la colonne **col** passée en paramètre.

Une lecture plus fine du ResultSet

Il est cependant possible de faire mieux en précisant le **nom de la colonne**, ainsi que le **type de données** que l'on désire récupérer pour chaque champ. Pour cela, la classe **ResultSet** possèdes les méthodes

- **String getString(String columnName)** throws SQLException
- **boolean getBoolean(String columnName)** throws SQLException
- **int getInt(String columnName)** throws SQLException
- **long getLong(String columnName)** throws SQLException
- **float getFloat(String columnName)** throws SQLException
- **double getDouble(String columnName)** throws SQLException
- **java.sql.Date getDate(String columnName)** throws SQLException
- **java.sql.Time getTime(String columnName)** throws SQLException
- …

Lors de l'exécution de ces méthodes, le driver JDBC tentera de **convertir** les données lues dans les tables en un format utilisable pour le type Java attendu.

En particulier, on remarque l'existence de la classe **Date** du package **java.sql** qui assure la correspondance entre les classes SQL et la classe **LocalDate** du package **java.time**.

Exemple de requête de sélection « fine »

Nous reprenons ici l'exemple précédent, à la différence que le ResultSet est « parsé » de telle sorte que l'on récupérer les champs de chaque tuple selon le format de données java correspondant.

Le code du programme de test (fichier **TestJDBC_MySql_Select.java**) est

```
import java.sql.*;
import java.time.format.DateTimeFormatter;
```

```
public class TestJDBC_MySql_Select
{
    public static void main(String[] args)
    {
        try
        {
            // Chargement du driver
            Class leDriver = Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Obtention du driver OK...");

            // Connexion a la BD
            Connection con =
DriverManager.getConnection("jdbc:mysql://192.168.228.167/PourStudent","Student","Pas
sStudent1_");
            System.out.println("Connexion à la BD PourStudent OK...");

            // Creation d'un objet Statement permettant d'executer une requete
            Statement stmt = con.createStatement();
            System.out.println("Creation du Statement OK...");

            // Execution de la requête
            ResultSet rs = stmt.executeQuery("SELECT * FROM Personnes");
            System.out.println("ExecuteQuery OK...");
            System.out.println("Instruction SELECT * FROM Personnes");

            // Affichage du contenu du Resulset (du curseur)
            while (rs.next())
            {
                // Récupération des données de chaque champ
                int id = rs.getInt("id");
                String nom = rs.getString("Nom");
                String prenom = rs.getString("Prenom");
                float poids = rs.getFloat("Poids");
                int nbEnfants = rs.getInt("NbEnfants");
                java.sql.Date dateSql = rs.getDate("AnneeNaissance");
                java.time.LocalDate date = dateSql.toLocalDate();

                // Formatage pour l'affichage de la date
                DateTimeFormatter formateur =
DateTimeFormatter.ofPattern("dd/MM/yyyy");
                String dateFormatee = date.format(formateur);

                // Affichage des données du tuple
                System.out.println("id=" + id + "\tNom=" + nom + "\tPrenom=" + prenom
+ "\tPoids=" + poids + "\tnbEnfants=" + nbEnfants + "\tAnneeNaissance=" +
dateFormatee);
            }

            // Fermture de la connexion
            rs.close();
            con.close();
        }
        catch (ClassNotFoundException ex)
        {
            System.out.println("Erreur ClassNotFoundException: " + ex.getMessage());
```

```
        }
        catch (SQLException ex)
        {
            System.out.println("Erreur SQLException: " + ex.getMessage());
        }
    }
}
```

dont un exemple d'exécution fournit

```
# java TestJDBC_MySql_Select
Obtention du driver OK...
Connexion à la BD PourStudent OK...
Creation du Statement OK...
ExecuteQuery OK...
Instruction SELECT * FROM Personnes
id=1  Nom=Issier  Prenom=Paul       Poids=75.4  nbEnfants=2 AnneeNaissance=11/05/1975
id=2  Nom=Coptere Prenom=Eli        Poids=81.7  nbEnfants=0 AnneeNaissance=21/12/2001
id=3  Nom=Tombale Prenom=Pierre     Poids=67.5  nbEnfants=1 AnneeNaissance=15/02/1983
#
```

On remarque que la conversion entre une classe **Date** du package **java.sql** et une classe **Date** du package **java.time** se réalise simplement avec la méthode **toLocalDate()** de la classe java.sql.Date.

Exemple de requête d'ajout, de suppression et de mise à jour

Nous reprenons à nouveau l'exemple précédent. Dans l'exemple ci-dessous, nous réalisons un ajout, une suppression et une mise à jour.

Le code du programme de test (fichier **TestJDBC_MySql_AddDeleteUpdate.java**) est

```
import java.sql.*;

public class TestJDBC_MySql_AddDeleteUpdate
{
    public static void main(String[] args)
    {
        try
        {
            // Chargement du driver
            Class leDriver = Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Obtention du driver OK...");

            // Connexion a la BD
            Connection con =
DriverManager.getConnection("jdbc:mysql://192.168.228.167/PourStudent","Student","Pas
sStudent1_");
            System.out.println("Connexion à la BD PourStudent OK...");
```

```
            // Creation d'un objet Statement permettant d'executer une requete
            Statement stmt = con.createStatement();
            System.out.println("Creation du Statement OK...");

            // Execution requete d'ajout
            String requete = "insert into Personnes values
(NULL,'Dupont','Tom','56.2','4','1991-1-28');";
            int nbLignesAffectees= stmt.executeUpdate(requete);
            System.out.println("ExecuteUpdate OK... nbLignesAffectees = " +
nbLignesAffectees);
            System.out.println("--> insert into Personnes values
(NULL,'Dupont','Tom','56.2','4','1991-1-28')");

            // Execution requete de suppression
            requete = "delete from Personnes where nom='Coptere';";
            nbLignesAffectees= stmt.executeUpdate(requete);
            System.out.println("ExecuteUpdate OK... nbLignesAffectees = " +
nbLignesAffectees);
            System.out.println("--> delete from Personnes where nom='Coptere';");

            // Execution requete de mise à jour
            requete = "update Personnes set NbEnfants='10' where nom='Issier';";
            stmt.execute(requete);
            System.out.println("Execute OK...");
            System.out.println("--> update Personnes set NbEnfants='10' where
nom='Issier';");

            // Fermture de la connexion
            con.close();
        }
        catch (ClassNotFoundException ex)
        {
            System.out.println("Erreur ClassNotFoundException: " + ex.getMessage());
        }
        catch (SQLException ex)
        {
            System.out.println("Erreur SQLException: " + ex.getMessage());
        }
    }
}
```

Avant exécution du programme le contenu de la table **Personnes** est le suivant :

```
mysql> select * from Personnes;
+----+---------+--------+-------+-----------+----------------+
| id | Nom     | Prenom | Poids | NbEnfants | AnneeNaissance |
+----+---------+--------+-------+-----------+----------------+
|  1 | Issier  | Paul   |  75.4 |         2 | 1975-05-11     |
|  2 | Coptere | Eli    |  81.7 |         0 | 2001-12-21     |
|  3 | Tombale | Pierre |  67.5 |         1 | 1983-02-15     |
+----+---------+--------+-------+-----------+----------------+
3 rows in set (0.00 sec)

mysql>
```

tandis qu'un exemple d'exécution du programme est

```
# java TestJDBC_MySql_AddDeleteUpdate
Obtention du driver OK...
Connexion à la BD PourStudent OK...
Creation du Statement OK...
ExecuteUpdate OK... nbLignesAffectees = 1
--> insert into Personnes values (NULL,'Dupont','Tom','56.2','4','1991-1-28')
ExecuteUpdate OK... nbLignesAffectees = 1
--> delete from Personnes where nom='Coptere';
Execute OK...
--> update Personnes set NbEnfants='10' where nom='Issier';
#
```

Après exécution du programme, voici le contenu de la table **Personnes** :

```
mysql> select * from Personnes;
+----+---------+--------+-------+-----------+----------------+
| id | Nom     | Prenom | Poids | NbEnfants | AnneeNaissance |
+----+---------+--------+-------+-----------+----------------+
|  1 | Issier  | Paul   |  75.4 |        10 | 1975-05-11     |
|  3 | Tombale | Pierre |  67.5 |         1 | 1983-02-15     |
|  4 | Dupont  | Tom    |  56.2 |         4 | 1991-01-28     |
+----+---------+--------+-------+-----------+----------------+
3 rows in set (0.00 sec)

mysql>
```

On remarque que :

- Si on veut récupérer une information sur le résultat de la requête (ici le nombre de lignes affectées), il faut utiliser la méthode **executeQuery()**
- Si on ne veut récupérer aucune information sur le résultat de la requête exécutée, on utilise la méthode **execute()**

Encapsulation d'une connexion JDBC

Dans une application, il est courant de la scinder en « couches » ayant chacune leur rôle. Il serait donc intéressant d'encapsuler la connexion JDBC à une base de données dans un objet dédié et dont le rôle serait de gérer une connexion à une base de données.

Appelons par exemple **DatabaseConnection** la classe instanciant un tel type d'objet. Voici un exemple de code de ce genre de classe :

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

```
import java.util.Hashtable;

public class DatabaseConnection
{
    private Connection connection;

    public static final String MYSQL = "MySql";

    private static Hashtable<String,String> drivers;

    static
    {
        drivers = new Hashtable<>();
        drivers.put(MYSQL,"com.mysql.cj.jdbc.Driver");
    }

    public DatabaseConnection(String type,String server,String dbName,String
user,String password) throws ClassNotFoundException, SQLException
    {
        // Chargement du driver
        Class leDriver = Class.forName(drivers.get(type));

        // Création de l'URL
        String url = null;
        switch(type)
        {
            case MYSQL: url = "jdbc:mysql://" + server + "/" + dbName;
                        break;
        }

        // Connexion à la BD
        connection = DriverManager.getConnection(url,user,password);
    }

    public synchronized ResultSet executeQuery(String sql) throws SQLException
    {
        Statement statement = connection.createStatement();
        return statement.executeQuery(sql);
    }

    public synchronized int executeUpdate(String sql) throws SQLException
    {
        Statement statement = connection.createStatement();
        return statement.executeUpdate(sql);
    }

    public synchronized void close() throws SQLException
    {
        if (connection != null && !connection.isClosed())
            connection.close();
    }
}
```

On remarque

- qu'il est possible **d'ajouter d'autres drivers** à cette classe afin de pouvoir se connecter à un autre type de base de données que MySql
- la présence du mot clé « **synchronized** » devant les méthodes exécutant les requêtes. Cela permet à plusieurs threads de manipuler proprement la connexion unique à la base de données.

Un exemple de programme de test utilisant cette classe (fichier **TestDatabaseConnection.java**) est le suivant :

```
import java.sql.*;

public class TestDatabaseConnection
{
    public static void main(String[] args)
    {
        try
        {
            DatabaseConnection dbConnect;
            dbConnect = new DatabaseConnection(DatabaseConnection.MYSQL,
                                               "192.168.228.167",
                                               "PourStudent",
                                               "Student",
                                               "PassStudent1_");

            // Exécution d'une requête de sélection
            String requete = "select * from Personnes;";
            ResultSet rs = dbConnect.executeQuery(requete);

            while(rs.next())
                System.out.println("Nom = " + rs.getString("Nom"));

            // Exécution d'une requête de mise à jour
            requete = "update Personnes set NbEnfants='10' where nom='Issier';";
            int nbLignes = dbConnect.executeUpdate(requete);

            System.out.println("nbLignes = " + nbLignes);

            // Fermeture de la connexion
            rs.close();
            dbConnect.close();
        }
        catch (ClassNotFoundException | SQLException ex)
        {
            System.out.println("Erreur DatabaseConnection: " + ex.getMessage());
        }
    }
}
```

Avant exécution du programme le contenu de la table **Personnes** est le suivant :

```
mysql> select * from Personnes;
+----+---------+--------+-------+-----------+----------------+
| id | Nom     | Prenom | Poids | NbEnfants | AnneeNaissance |
```

```
+----+---------+--------+-------+-----------+----------------+
|  1 | Issier  | Paul   |  75.4 |         2 | 1975-05-11     |
|  2 | Coptere | Eli    |  81.7 |         0 | 2001-12-21     |
|  3 | Tombale | Pierre |  67.5 |         1 | 1983-02-15     |
+----+---------+--------+-------+-----------+----------------+
3 rows in set (0.00 sec)

mysql>
```

tandis qu'un exemple d'exécution du programme est

```
# java TestDatabaseConnection
Nom = Issier
Nom = Tombale
Nom = Dupont
nbLignes = 1
#
```

Remarque :

On pourrait encore imaginer d'encapsuler cet objet (instanciant **DatabaseConnection**) dans un objet « **DAL (Data Access Layer)** » propre à l'application afin de rendre la manipulation de la base de données complètement indépendante du reste de l'application.

Exemple :

```
Personne p = new Personne(« Durant », « Paul », 62.3, 2, « 21-5-2001 ») ;

dal.enregistre(p) ;

ArrayList<Personne> liste = dal.getPersonnes() ;
```

L'objet **dal**, contenant une instance de **DatabaseConnexion**, pourrait alors

- **enregistre** : récupérer les champs de l'objets p afin d'en créer une requête SQL d'ajout et exécuter cette requête sur la base de données
- **getPersonnes** : réaliser une requête SQL de sélection en base de données afin de récupérer toutes les personnes et retourner un ArrayList contenant l'ensemble de ces personnes sous forme d'objets instanciant la classe Personne.

Il existe des libraires permettant de faire ce genre de chose de manière plus automatique. En Java, on peut par exemple utiliser **JPA**.