Module 19 :
Application client/serveur

JDBC

Benkhayat Yassine

yasben11@gmail.com

Introduction à JDBC

- JDBC, Java Data Base Connectivity est un ensemble de classes (API – Application Programming Interface --JAVA) permettant de se connecter à une base de données relationnelle en utilisant des requêtes SQL ou des procédures stockées.

- L'API JDBC a été développée de manière à pouvoir se connecter à n'importe quelle base de données avec la même syntaxe; cette API est dite indépendante du SGBD utilisé.

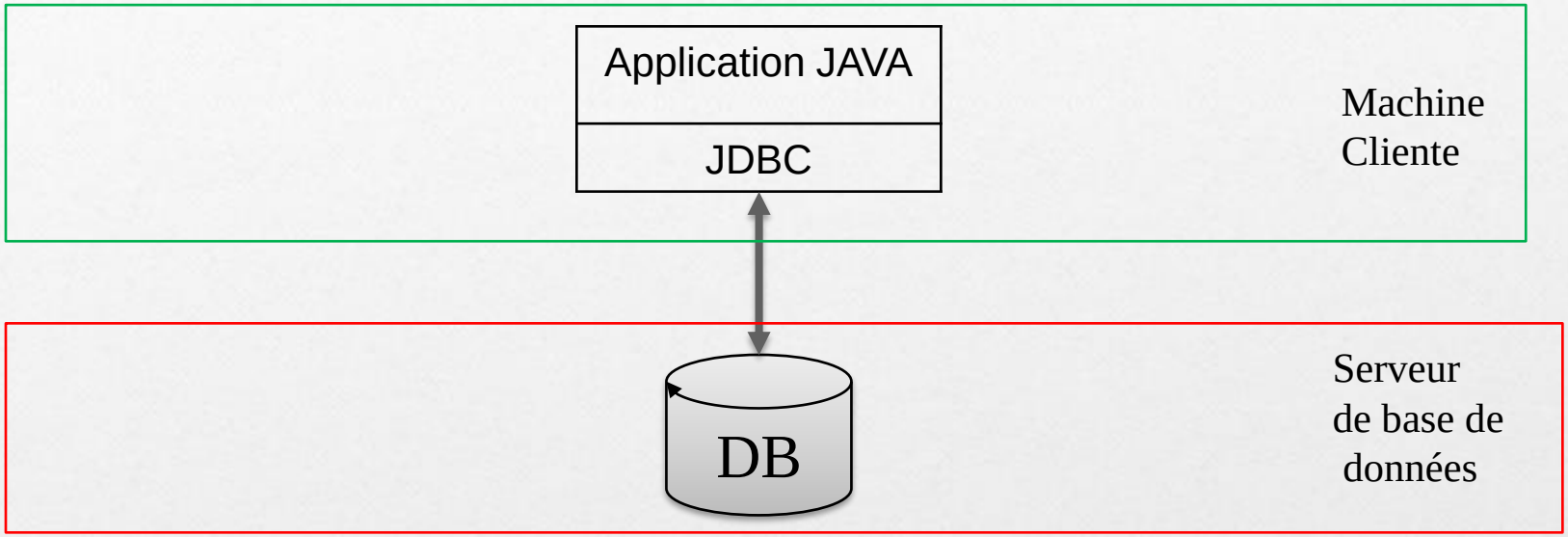
- Les classes JDBC font partie du package **java.sql** et **javax.sql**

- Pilote de bases de données ou driver JDBC :est un "logiciel" qui permet de convertir les requêtes JDBC en requêtes spécifiques auprès de la base de données.

Architecture de JDBC

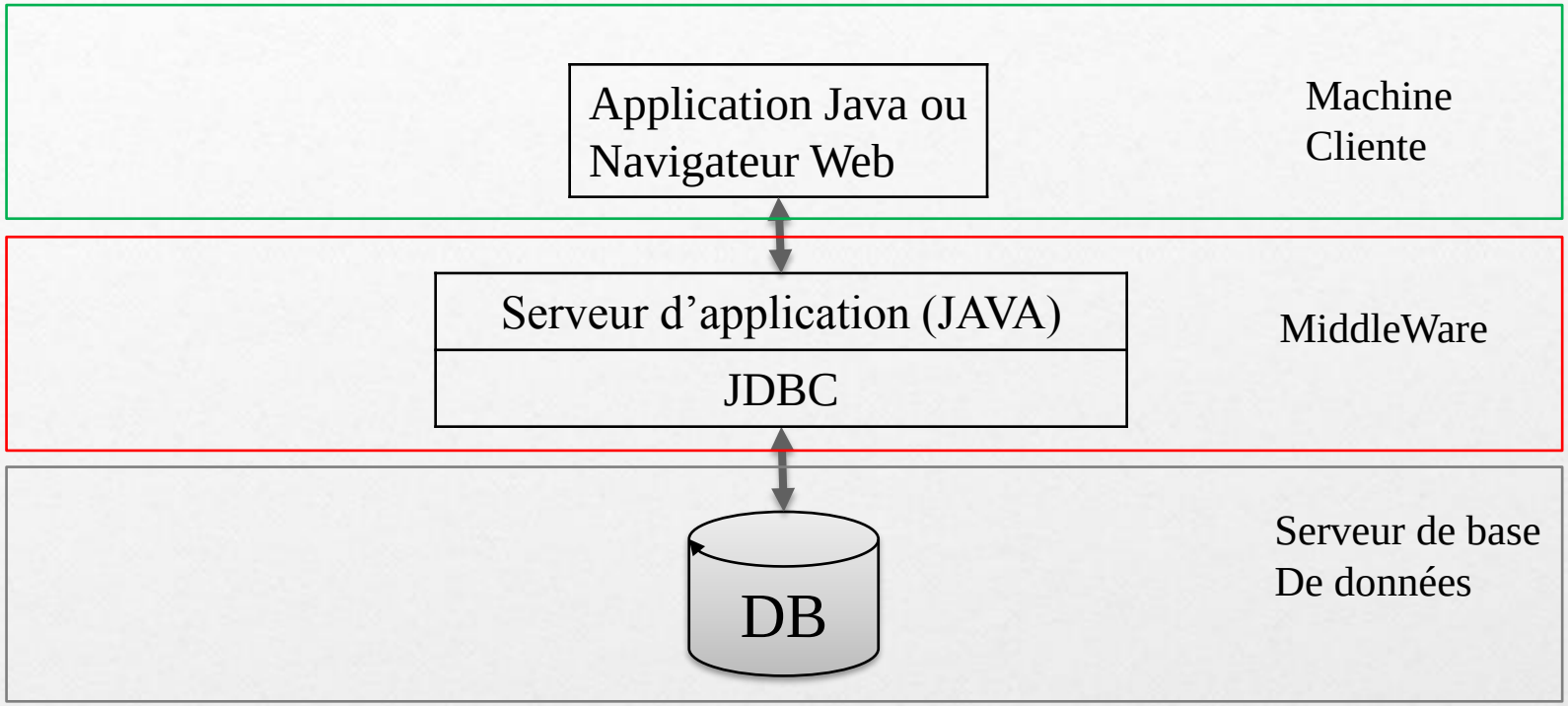
JDBC fonctionne selon les deux modèles : 2-tier et 3-tier

(2-tier)



Dans le modèle two-tier, une application JAVA (ou une applet) dialogue avec le SGBD par l'intermédiaire du pilote JDBC. L'application JAVA et le pilote JDBC s'exécutent sur l'ordinateur client tandis que le SGBD est placé sur un serveur.

Architecture de JDBC : Modèle 3-tier



Dans le modèle three-tier, l'applet (ou l'application JAVA) ne dialogue plus directement avec un SGBD : un MiddleWare fait le lien entre ces deux composants
Le SGBD exécute les requêtes SQL et envoie les résultats au middleware.
Ces résultats sont ensuite communiqués à l'applet sous forme d'appels http.

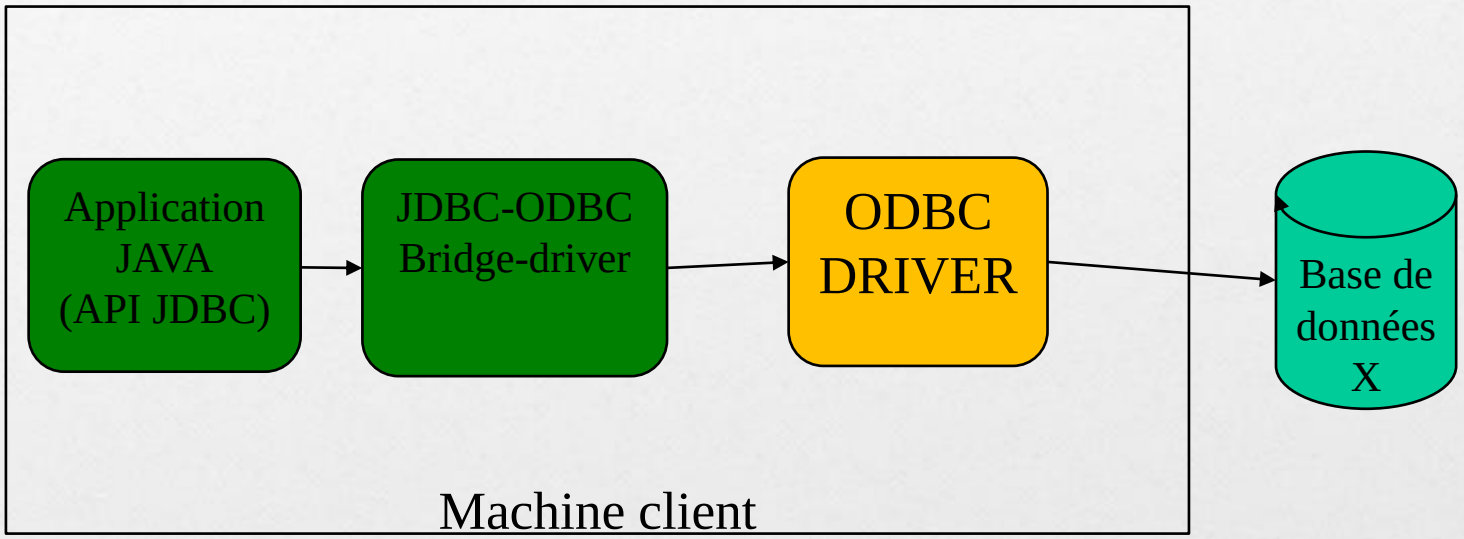
Les types de pilotes JDBC

Le driver JDBC est un composant logiciel qui permet a un programme java d'interagir avec une base de donnée, il y a 4 types de pilote JDBC (Java Data Base Conectivity) :

1. Le pont JDBC-ODBC (JDBC-ODBC bridge driver)
2. Pilote natif(Native-API driver "partially java driver")
3. Le pilote de protocole réseau (Network Protocol driver "fully java driver")
4. Pilote entièrement écrit en java(Thin driver)

Pilote JDBC type 1: Le pont JDBC-ODBC

Les appelles JDBC sont traduit en ODBC via le pilote JDBC-ODBC et utilisent le pilote ODBC pour communiquer avec le base de données X.



Rappel : ODBC

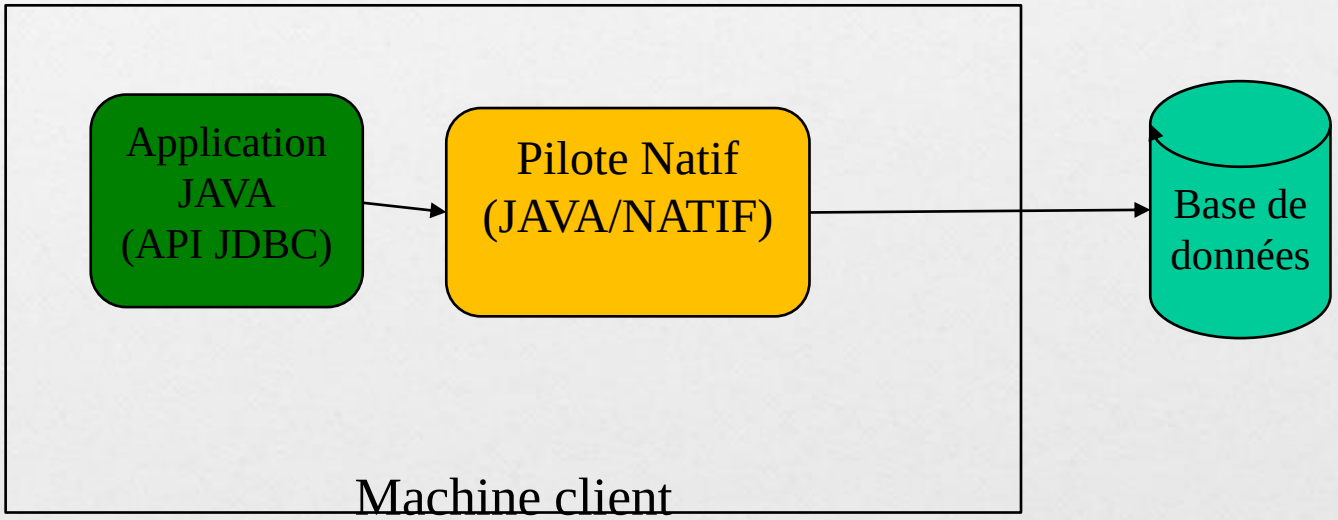
- ODBC signifie Open DataBase Connectivity. Il s'agit d'un format défini par Microsoft permettant la communication entre des clients bases de données fonctionnant sous Windows et les SGBD du marché. Le gestionnaire ODBC est présent sur les systèmes Windows.
- La technologie ODBC permet d'interfacer de façon standard une application à n'importe quel serveur de bases de données, pour peu que celui-ci possède un driver ODBC (la quasi-totalité des SGBD possèdent un tel pilote, dont tous les principaux SGBD du marché).

Inconvénients de la technologie ODBC:

- Cette technologie reste une solution propriétaire de Microsoft. Cela se traduit par une dépendance de la plateforme (ODBC ne fonctionne que sur les plateformes Microsoft Windows).
- Difficile à mettre en œuvre directement dans les programmes.

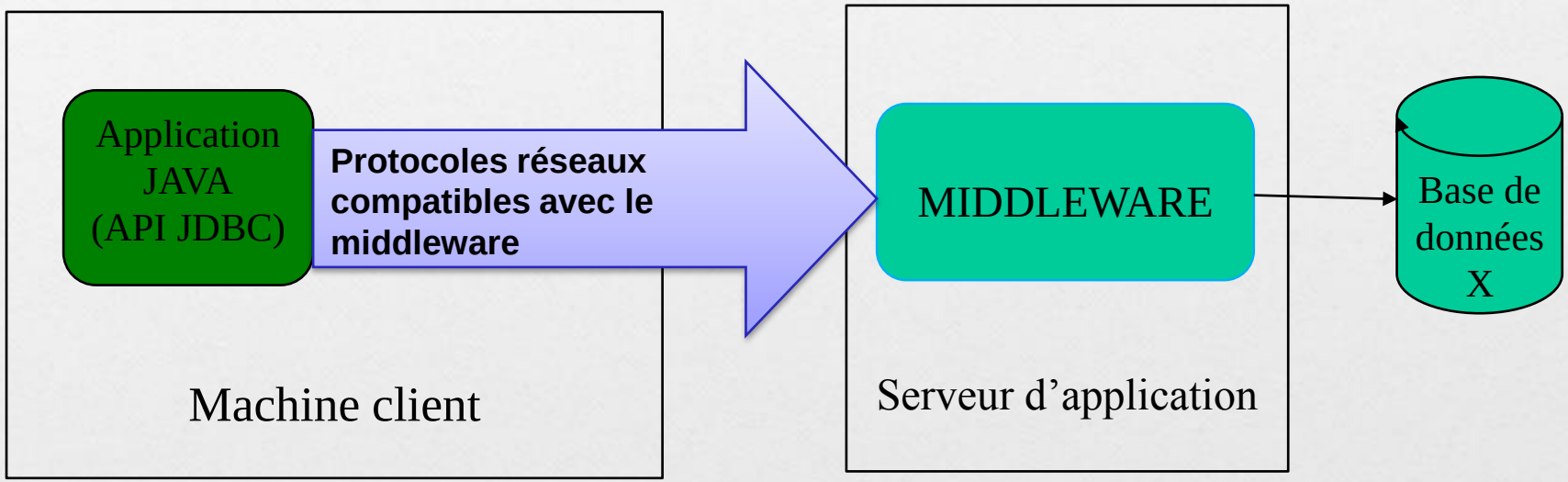
Pilote JDBC type 2: Pilote natif

Pilote(Driver) n'est pas entièrement écrit en Java. La partie de ce pilote écrite en Java effectue simplement des appels vers des fonctions du pilote natif(C/C++), ce dernier convertit les méthodes JDBC pour appeler directement la base de données via un pilote natif sur le client.



Pilote JDBC type 3:
Le pilote de protocole réseau

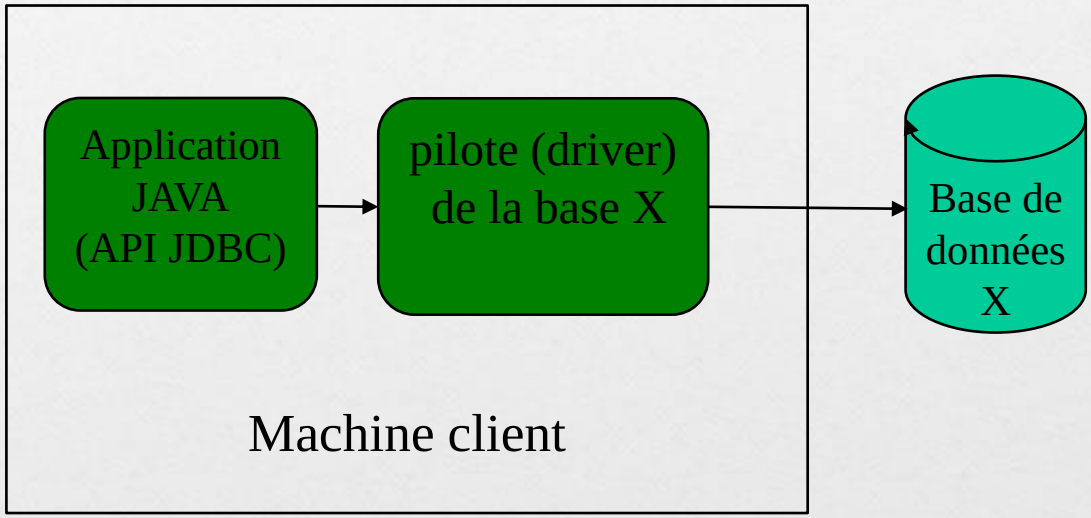
Ce type de driver traduit les appelles jdbc en un protocole réseau compréhensible par le middleware , ce dernier fournit la connectivité à une base données via des protocoles spécifique de cette base de données.



Pilote JDBC type 4:
Pilote entièrement écrit en java(Thin driver)

Ce type de pilote(driver) écrit en java, se connecte directement au SGBD. Il est fourni par l'éditeur de la base de données.

Ce type de driver est le plus optimale puisqu'il est en java et la majorité des SGBD le fournie.



Fonctionnement

Tout programme Java exploitant l'API JDBC fonctionne selon les étapes suivantes :

1. Connexion à la base de données
   - Chargement du pilote de la BDD
   - Demande de connexion: s'identifiant auprès du SGBD et en précisant la base utilisée
2. Traitement des commandes SQL
3. Traitement des résultats
4. Fermeture de la connexion.

Connexion à la base de données

1. Chargement du pilote (driver)

Pour établir une connexion, il faut d'abord charger le driver en utilisant la méthode forName de la classe Class comme suit :

```
Class.forName(string driver);
```

Pour MySQL, l'instruction est la suivante :
Class.forName("com.mysql.jdbc.Driver");

2. Demander une connexion

Une fois le pilote chargé, alors on peut demander une connexion à la base de données. Cette connexion est obtenue grâce à la méthode **getConnection** de la classe DriverManager

Cette méthode retourne la connexion qui est en fait, un **objet** implémentant l'interface «Connection». Voila la syntaxe :

```
Connection connexion = DriverManager.getConnection(url,user,pass);
```

url désigne la chaine de connexion , **user** est le nom d'un utilisateur ayant l'autorisation sur la base ,et **pass** c'est le mot de passe de cet utilisateur.

Exemple de Connexion à une base de données MySQL

```
String url = "jdbc:mysql://localhost/magasin";
String user = "root";
String motdepasse = "";
try {
Class.forName("com.mysql.jdbc.Driver");
System.out.println("Pilote chargé");
} catch (ClassNotFoundException cnfe) {
System.out.println("ERREUR : Driver manquant.");
}
try {
Connection connexion = DriverManager.getConnection(url, user, motdepasse);
System.out.println("connécté");
} catch (SQLException se) {
System.out.println("ERREUR : bd manquante ou connexioninvalide.");
}
```

Déclarations de Requêtes SQL :Interfaces : Statement, PreparedStatement, CallableStatement

Cette étape consiste à obtenir une déclaration ou «*statement » au* travers de laquelle les requêtes SQL seront exécutées.
Il existe 3 types de déclarations ou «*statement »* :
1. Statement: instruction simple : permet d'exécuter directement et une fois l'action sur la base de données :
Statement declaration1= connexion.createStatement();
2. PreparedStatement: instruction paramétrée. (cas des requêtes avec paramètres)
• L'instruction est générique, des champs sont non remplis
• Permet une précompilation de l'instruction optimisant les performances
• Pour chaque exécution, on précise les champs manquants
PreparedStatement declaration2= connexion.PreparedStatement (requetesql);
3. CallableStatement:
Une déclaration de type « CallableStatement» permet l'accès complet aux fonctions contenues dans la base de données.(cas des procédures stockées)

Exécution de Requêtes SQL : executeUpdate

La méthode **ExecuteUpdate** est utilisée pour les **Requêtes** de manipulation des enregistrements(INSERT, DELETE, UPDATE) et returne le nombre des mis ajour et une exception dans le cas contraire.

Syntaxe:

objetStatement.**executeUpdate**(String Requête_SQL]);
ou
objetPreparedStatement.**executeUpdate**(String Requête_SQL]);

Exemple :

```
String requete= "INSERT INTO employes (numemp, nom) VALUES (1,'Mounir')";
Connection connexion = DriverManager.getConnection(url1,usager,motdepasse);
Statement stm = connexion.createStatement();
stm.executeUpdate(requete);
System.out.println("insertion complétée");
```

Exercice 1 :

Écrire un programme JAVA qui permet de :
1-Se connecter à notre base de données magasin
2-Supprimer le client dont le code = 1
3-Afficher le nombre d'enregistrements supprimés

Indications :

✓ Servez vous des variables ci-dessous :

```
String url = "jdbc:mysql://localhost/magasin";
String user = "root";
String password = "";
String Driver_Name= "com.mysql.jdbc.Driver"
```

✓ La figure ci-contre montre la structure de la table client

Exécution de Requêtes SQL : executeQuery

- La méthode **executeQuery,** permet d'exécuter une instruction SQL de type SELECT
- Elle retourne un objet de type **ResultSet** contenant tous les résultats de la requête .

Syntaxe:

objetResultSet=objetStatement.**executeQuery(**String ordreSQL);
ou
objetResultSet=objetPreparedStatement.**executeQuery(**String ordreSQL)

Exploitation des résultats des requêtes SQL

- L'interface **ResultSet** représente une table de lignes et de colonnes.
  - Une ligne représente un enregistrement
  - Une colonne représente un champ particulier de la table
  - Un objet de type ResultSet possède un pointeur sur l'enregistrement courant. À la réception de cet objet, le pointeur se trouve devant le premier enregistrement.
  - On y accède ligne par ligne, puis colonne par colonne dans la ligne

- Pour pouvoir récupérer les données contenues dans l'instance de **ResultSet**, celui-ci met à disposition des méthodes permettant de :
  - Positionner le curseur sur l'enregistrement suivant :
  - public boolean **next();** Renvoi un booléen indiquant la présence d'un élément suivant.
  - Accéder à la valeur d'un champ (par indice ou par nom) de l'enregistrement actuellement pointé par le curseur avec les méthodes getInt(),getString(), …..
- À la création du ResultSet, le curseur de parcours est positionné avant la première occurrence à traiter. **Le premier indice étant 1**

Exemple de parcours d'enregistrement d'une table

```
//on suppose que la connexion est deja etablie
//creation d'un statment
Statement stm = connexion.createStatement();
//creation de la requete SQL
String requete = "select * from article;";
//stoker le result de l'execution de la requite dans un ResultSet
ResultSet result = stm.executeQuery(requete);
//parcours du ResultSey
        while (result.next()) {
        String designation = result.getString("designation");
        String prix = result.getString("prix");
        System.out.println("le produit :" + designation + "coute : " + prix);

        }
```

Exercice 2:

Écrire un programme JAVA qui permet de :
2-Afficher toutes les informations (id_client,nom) de la table client

Indications :

✓ Servez vous des variables ci-dessous :

String url = "jdbc:mysql://localhost/magasin";

String user = "root";

String password = "";

String Driver_Name= "com.mysql.jdbc.Driver"

✓ La figure ci-contre montre la structure de la table client

Fermeture d’une connexion :

La connexion est fermée avec la méthode close de l’objet **connexion** crée.

Exemple :

```
String url = "jdbc:mysql://localhost/magasin";
String user = "root";
String motdepasse = "";
Class.forName("com.mysql.jdbc.Driver");
Connection connexion ;
connexion= DriverManager.getConnection(url, user, motdepasse);
// Traitement de sélections et des mises à jour
// Fermeture de la connexion
connexion.close()
```

Utilisation du PreparedStatement

Ce type d'interface est utilisé pour des requêtes paramétrées. **PreparedStatement** est utilisé dans le cas où la requête va être exécutée plusieurs fois. De plus Les requêtes sont précompilées.

Remarques:

1. Dans le requête le paramètre est représenté par **?**
2. Les paramètres sont passés dans l'ordre de leur présentation de la requête
3. Les paramètres et les valeurs sont passés comme suit :
   [Objet PreparedStatement].setString([index],[objet String]);
   [Objet PreparedStatement].setBoolean([index],[valeur]);
   [Objet PreparedStatement].setInt([index],[valeur]);
   [Objet PreparedStatement].setFloat([index],[valeur]);
4. La requête est exécuté par executeUpdate() ou executeQuery

Exemples : PreparedStatement

Exemple 1 : executeQuery

```
String requete1 = "select *  employes where prenom = ? And nom = ? "
// On suppose que la connexion est pre-etablie
PreparedStatement stm1 = connexion.prepareStatement(requete);
stm1.setString(1, "Blabla");
stm1.setString(2, "Alpha");
stm1.executeQuery();
```

Exemple 2: executeUpdate

```
String requete2 = "update employes set prenom = ? And nom = ? ";
// On suppose  que la connexion est pre-etablie
PreparedStatement stm2 = connexion.prepareStatement(requete2);
stm2.setString(1, "Blabla");
stm2.setString(2, "Alpha");
stm2.executeUpdate();
```

L'interface ResultSetMetaData

Interface **ResultSetMetaData** permet d'obtenir toute sorte d'information sur les colonnes d'un ResultSet.

La méthode **getMetaData**() d'un objet **ResultSet** retourne un objet de type **ResultSetMetaData**. Cet objet permet d'obtenir des informations sur les colonnes par exemple :le nombre, le nom et le type des colonnes etc…

Exemple:

```
ResultSet rs = stm.executeQuery(requete);
ResultSetMetaData rsmd;
rsmd = rs.getMetaData();
nbCols = rsmd.getColumnCount();
String columnName = rsmd.getColumnName(1) ;
int columnType = rsmd.getColumnType(1) ;
```

Le nombre de colonnes peut être obtenu grâce à la méthode **getColumnCount**() de cet objet.

Le nom de la 1ére colonne peut être obtenu par la methode **getColumnName**(1)

Le type de la 1ére colonne peut être obtenu par la methode **getColumnType**(1)

L'interface DatabaseMetaData

Un objet de la classe DatabaseMetaData permet d'obtenir des informations sur la base de données dans son ensemble : nom des tables, nom des colonnes dans une table, la version du SGBD , l'url d'acces à la base de données etc…

Méthode	Rôle
ResultSet getTables()	Retourner des descriptions sur une ou plusieurs tables dans la base de données
ResultSet getColumns()	Retourner des descriptions sur une ou plusieurs colonnes dans table.
String getURL()	Retourner l'URL de la base à laquelle on est connecté
String getDriverName()	Retourner le nom du driver utilisé
String getProductName()	Retourner le nom du SGBD

Exemple

```
DatabaseMetaData dmd = connexion.getMetaData();
System.out.println("DatabaseProductName: " + dmd.getDatabaseProductName());
System.out.println("DatabaseProductVersion: " + dmd.getDatabaseProductVersion());
System.out.println("getDriverName() : " + dmd.getDriverName());
System.out.println("getURL() : " + dmd.getURL());
```

Exercice 3:

Écrire un programme JAVA qui permet à l'aide d'une requête paramétré :

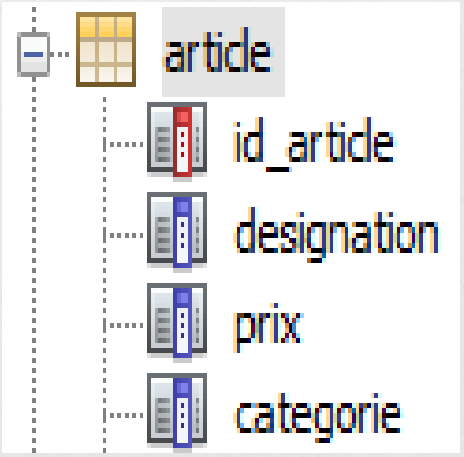
1-Afficher toutes les informations sur les articles de la catégorie: photo

2-et avec la même requête d'afficher toutes les informations sur les articles de la catégorie: informatique.

Indications :

On suppose que la connexion est préétablie (la variable de connexion est **connexion3**)

✓ La figure ci-contre montre la structure de la table client



Gestions des transactions

- Une **transaction** est un **ensemble d'instructions** devant s'exécuter d'un seul bloc. C'est à dire que les instructions sont soit toutes exécutées, soit toutes annulées. Si une seule instruction du bloc échoue la transaction ne prends pas effet.
- Par défaut, les **Connection** sont en mode **auto-commit**. C'est à dire que chaque instruction SQL est considérée comme une transaction. Vous pouvez changer ce comportement en utilisant la méthode **setAutoCommit(boolean)**.
- Dans le cas ou l'auto-commit est desactivé, vous devrez appeller la méthode **commit** pour effectivement valider la transaction.
- Pour annuler une transaction vous devez utiliser la méthode **rollback**.

Exemple :

```
Connection connection = null;
connection = ...;
connection.setAutoCommit(false);
//instruction N° 1 de la modification de la base de donnés
//instruction N° 2 de la modification de la base de donnés
//instruction N° 3 de la modification de la base de donnés
connection.commit();// c'est ici que l'on valide la transaction
connection.setAutoCommit(true);
//instruction N° 4 de la modification de la base de donnés
//instruction N° 5 de la modification de la base de donnés
//instruction N° 6 de la modification de la base de donnés
```

Dans cet exemple si une de ces 3 instructions (1,2et 3) ne s'est pas bien déroulé aucune modification ne va être apporter à la base . Par contre les autres instructions (4,5 et 6) l'echec de l'une n'impacte pas les autres.

TP