

Cours  
**P**rogrammation **O**rientée  
**O**bjets **A**vancés

## Chapitre 5

# L'accès aux bases de données

Public Cible : L4DSI

Réalisé par : KHELIFA Afifa

Année universitaire 2021-2022



# PLAN

**1**

**Introduction à la persistance  
des données**

**2**

**JDBC: définition, architecture et  
types de drivers**

**3**

**Les classes de l'API JDBC**

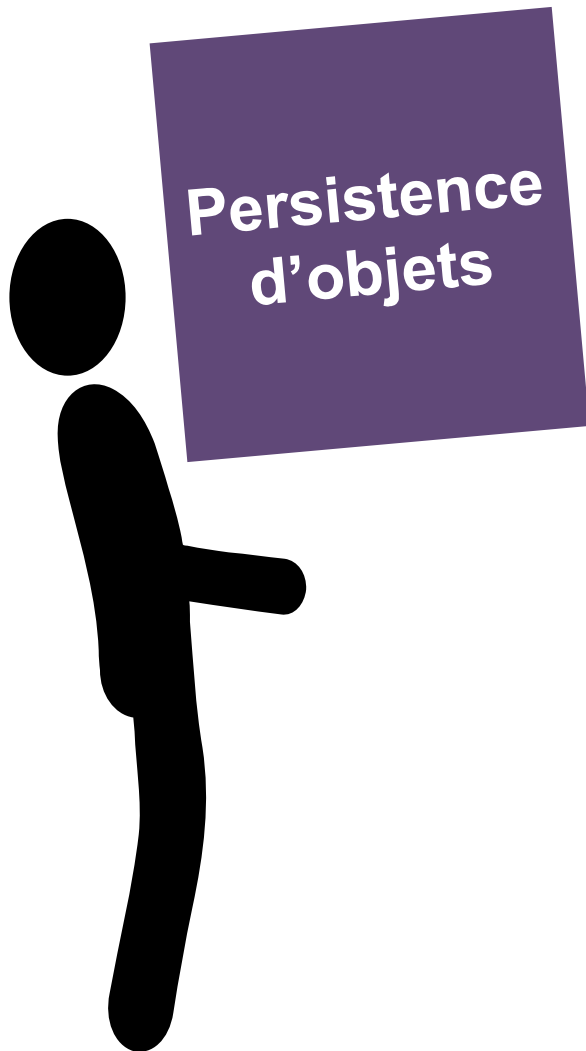
**4**

**Etapes d'accès à une BD à  
l'aide de JDBC**

**5**

**Exemple complet d'accès à  
une BD**

# Persistence d'objets



La **persistance** d'objets est un mécanisme pour la sauvegarde et la restauration des données.

Différentes solutions peuvent être utilisées pour la persistance des objets en Java :

- Sérialisation
- JDBC : Java Data Base Connectivity
- SQL/J
- Framework de mapping O/R (Object Relational Mapping)
- DAO
- Base de données objet (ODBMS)

# C'est quoi le JDBC?

## JDBC (Java Data Base Connectivity)

Cette API a été développée par SUN pour permettre à des applications Java d'accéder à des bases de données relationnelles quelconques.

- ❑ Une API permettant un accès uniforme à des BD relationnelles : **portable sur la plupart des OS.**
- ❑ **Indépendant du SGBD** (seule la phase de connexion est spécifique – driver)
- ❑ **Compatible avec la plupart des SGBDR** : Oracle, Postgres, MySQL, Informix, Sybase, MS SQL Server...

Toutes les classes et les interfaces de JDBC sont principalement dans le package **java.sql** mais certaines classes se trouvent dans **javax.sql**.

# C'est quoi le JDBC?

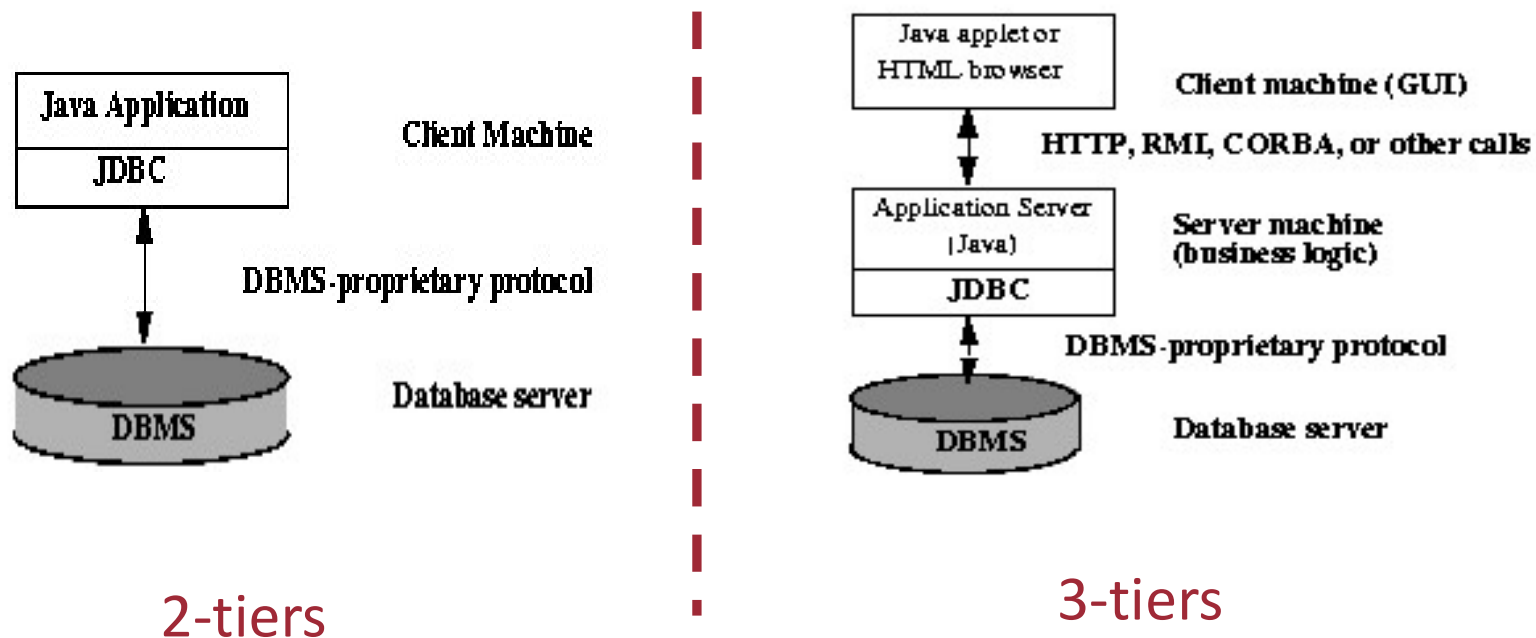
## ***L'API JDBC***

- ❑ Fait partie du **JDK** (Java Development Kit)
- ❑ Repose sur 8 classes et 8 interfaces définissant les objets nécessaires à:
  - ✓ la connexion à une BD distante
  - ✓ la création et l'exécution de requêtes SQL
  - ✓ la récupération et le traitement des résultats
  - ✓ l'accès au *méta-modèle*
- ❑ Tous les types des applications Java (applications de bureau, Applet, Servlets, JSP,...) sont capables d'utiliser cette API.

# JDBC : Architecture

6

L'API JDBC supporte deux modèles d'architectures pour l'accès aux BD:



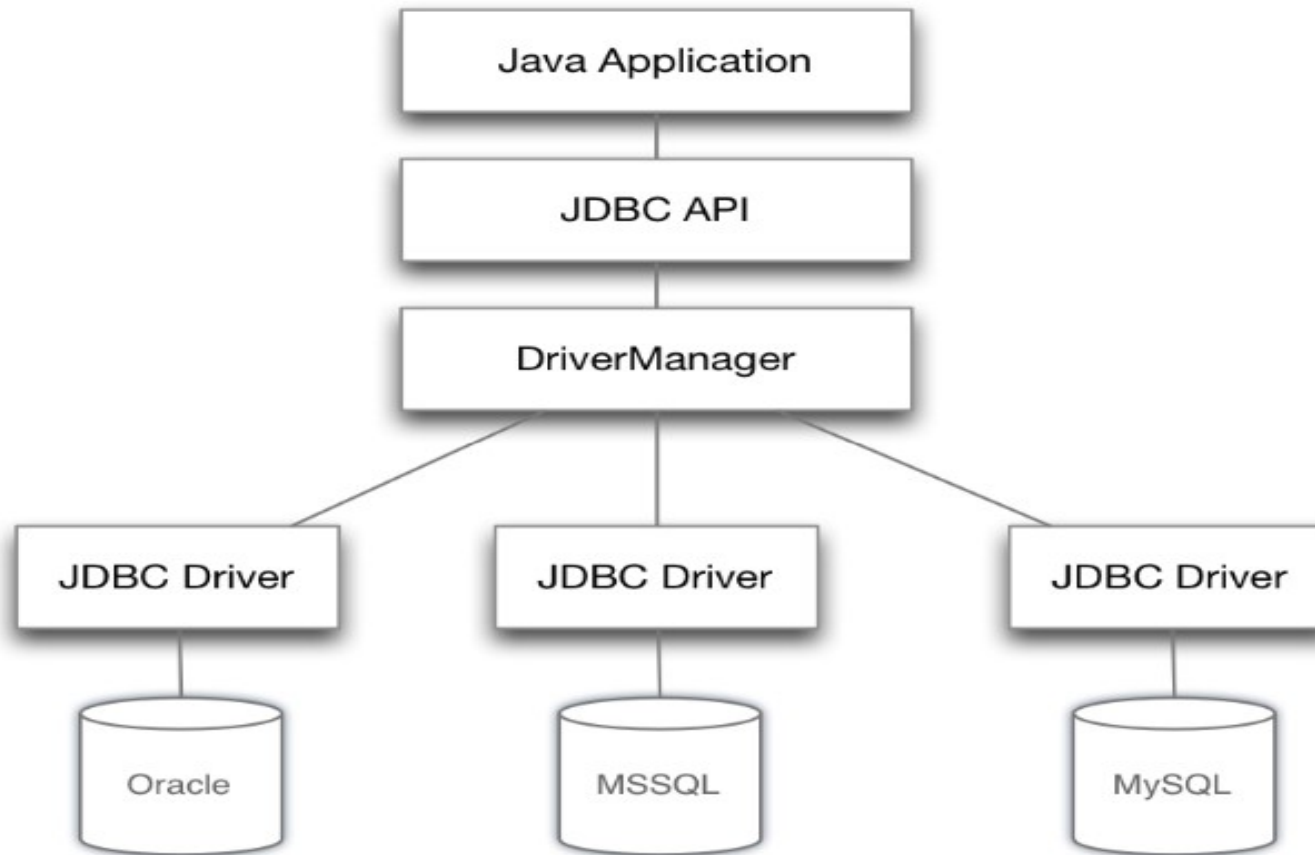
# JDBC : Architecture

7

- ❑ JDBC est une API qui permet d'établir une connexion entre une application java et une BD
- ❑ JDBC se comporte comme un **middleware** : toute instruction envoyée est traduite par l'API JDBC en langage compréhensible par la BD (sql).
- ❑ Quand la base de données répond aux instructions, l'API JDBC traduit les réponses en langage compréhensible par l'application.
- ❑ Plusieurs types de BD sont accessibles comme MySQL, SQLServer, PostgreSQL, Oracle, etc.
- ❑ Plusieurs versions avec des fonctionnalités plus évoluées (la dernière 4.2)

# JDBC : Architecture

8



2

JDBC: définition, architecture et types de drivers



# JDBC : Architecture

9

Architecture à 2 niveaux :

- ❑ **Niveau supérieur : API JDBC** : couche visible, nécessaire pour interfacer les applications Java et le SGBD (package java.sql)
- ❑ **Niveau inférieur : DRIVERS (pilotes)** : l'interface entre les accès bas niveau au moteur du SGBD et l'application
  - ✓ chaque SGBD utilise un **pilote (driver)** particulier
  - ✓ permettent de traduire les requêtes JDBC dans le langage du SGBD
  - ✓ constitués de **classes** implantant certaines **interfaces** de java.sql
  - ✓ plus de **200 drivers** sont actuellement disponibles

Un pilote ou driver JDBC est un "logiciel" qui permet d'établir une connexion entre un programme java et un système de gestion de bases de données. Ce "logiciel" est en fait une implémentation de l'interface Driver, du package java.sql.

# Les classes de JDBC

10

Classe	Rôle
<b>DriverManager</b>	Charger et configurer le driver de la base de données.
<b>Connection</b>	Réaliser la connexion et l'authentification à la base de données.
<b>Statement (et PreparedStatement)</b>	Contenir la requête SQL et la transmettre à la base de données.
<b>ResultSet</b>	Parcourir les informations retournées par la base de données dans le cas d'une sélection de données

# Etapes d'accès à une BD à l'aide de JDBC

## Première étape

Préciser le type de driver que l'on veut utiliser

## Deuxième étape

Récupérer un objet « Connection » en s'identifiant auprès du SGBD et en précisant le nom de la base de données à utiliser

## Etapes suivantes

- A partir de la connexion, créer un « statement » (état) correspondant à une requête particulière
- Exécuter ce statement au niveau du SGBD
- Fermer le statement

## Dernière étape

Se déconnecter de la base en fermant la connexion

# Etape 1 : Chargement du pilote JDBC

12

Le chargement du pilote JDBC se fait à l'aide la méthode **Class.forName(String Nomdriver)**

**forName** permet de récupérer un objet de type java.lang.Class représentant l'objet passé en paramètre et donc de charger la classe du driver en mémoire.

De manière générale il est nécessaire de bien gérer les exceptions suivantes:

**ClassNotFoundException**: problème de chargement du driver (ex: driver introuvable)

## Exemples

*Class.forName("oracle.jdbc.driver.OracleDriver"); //Charger le pilote JDBC de Oracle*

*Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver"); //Charger le pilote de SQL Server.*

*Class.forName("com.mysql.cj.jdbc.Driver"); //Charger le driver de MySQL*

4

Etapes d'accès à une BD à l'aide de JDBC

# Étape 2: Connexion à une BD

13

Une connexion à une base de données avec JDBC est représentée par une instance de la classe **java.sql.Connection**.

Pour ouvrir une connexion vers une base de données, il suffit de spécifier l'**url** de connexion, le **login** et le **password**, à la méthode **getConnection** de DriverManager.

❑ Syntaxe:

```
String url = "url";
String login = "log";
String password = "pass";
try{
    Connection c =
    DriverManager.getConnection(url,login,password) ;

} catch (SQLException sqle)
{ // gérer l'erreur }
```

# Étape 2: Connexion à une BD

14

Les URL JDBC sont définies sous forme de String selon ce schéma :

String url = "jdbc:<subprotocol>:<subname>"

Avec:

**<jdbc>**: Le protocole dans une URL JDBC est toujours jdbc

**<subprotocol>**: Cela correspond au nom du driver ou au mécanisme de connexion à la base de données.

**<subname>** : Une manière d'identifier la source de données. Ce dernier élément dépend complètement du sous-protocole et du driver.

# Étape 2: Connexion à une BD

15

## Exemples d'URL

jdbc:odbc:maBase;CacheSize=30;ExtensionCase=LOWER

jdbc:mysql://localhost/maBase

jdbc:oracle:oci8@:maBase

dbc:oracle:thin:@localhost:1521:xe

jdbc:sybase:Tds:localhost:5020/maBase

4

Étapes d'accès à une BD à l'aide de JDBC

## Étape 2: Connexion à une BD

16

```
String driver = "oracle.jdbc.driver.OracleDriver";//driver de la BD Oracle
String url = "jdbc:oracle:thin:@localhost:1521:xe"; //url de la BD Oracle
String login = "system" ;
String password = "manager";
Connection conn = null;
try{
    Class.forName(driver);
    conn = DriverManager.getConnection(url,login,password);
}
catch(ClassNotFoundException cne)
{ System.out.println("Driver introuvable : "); cne.printStackTrace(); }
catch(SQLException sqle)
{ System.out.println("Erreur SQL : " );}
catch(Exception e)
{ System.out.println("Autre erreur : ");
e.printStackTrace(); }
```



## Exemple (lire les paramètres de connexion à partir d'un fichier de propriétés)

```
1 import java.sql.*;
2 public class TestConnection {
3     public static void main(String[] args) {
4         try
5         {
6             Connection c=Utilitaire.seConnecter("ConnectionPar.properties" );
7         }
8         catch (Exception e)
9         {
10             System.out.println (e.getMessage());
11         }
12     }
13 }
```

<terminated> TestConnection (1) [Jav  
Connexion établie

## Étape 3: Accès à la BD

### Comment créer un statement?

L'interface **Statement** représente une instruction SQL. L'obtention d'une instance de cette interface se fait à partir de la Connection :

```
try{  
    //conn est un objet de type Connection déjà ouvert  
    Statement state = conn.createStatement();  
    state.execute("SELECT * FROM ...");  
}  
catch(Exception e)  
    { //gestion de l'exception }
```

## Étape 3: Accès à la BD

### Comment exécuter un statement?

Méthode d'exécution	Rôle	Type de retour
<b>execute</b>	Générique pour n'importe quelle expression SQL.	un boolean true si l'instruction renvoie un ResultSet, false sinon
<b>executeQuery</b>	SELECT	un ResultSet contenant les résultats
<b>executeUpdate</b>	INSERT, UPDATE, DELETE, CREATE, etc	int indiquant le nombre de tuples (lignes) modifiés
<b>executeBatch</b>	Exécution d'un groupe de requêtes.	int[] : un tableau d'entiers indiquant le nombre de tuples modifiés pour chaque requête contenue dans le batch.

## Étape 3: Accès à la BD

### Quelques exemples

```
Statement state = conn.createStatement();
```

```
boolean result = state.execute("SELECT * FROM  
MATABLE");
```

```
ResultSet resultSet = state.executeQuery("SELECT  
ATTRIBUT1, ATTRIBUT2 FROM MATABLE");
```

```
int nb = state.executeUpdate("INSERT INTO MATABLE  
VALUES (15, 'bonjour', 7.0)");
```

# Étape 3: Accès à la BD

## La classe resultSet

C'est une classe qui représente une abstraction d'une table qui se compose de plusieurs enregistrements constitués de colonnes qui contiennent les données.

Les principales méthodes pour obtenir des données sont :

- **getInt(int/String)** : retourne sous forme d'entier le contenu de la colonne dont le numéro/nom est passé en paramètre.
- **getFloat(int/String)** : retourne sous forme d'un nombre flottant le contenu de la colonne dont le numéro/nom est passé en paramètre.
- **getDate(int/String)** : retourne sous forme de date le contenu de la colonne dont le numéro/nom est passé en paramètre.
- **next()/previous()/last()/First()/beforeFirst()/afterLast()**: déplacer le curseur sur l'enregistrement en question.
- **close()**: ferme le ResultSet
- **getMetaData()**: retourne un objet de type ResultSetMetaData associé au ResultSet.

## Étape 3: Accès à la BD

### Le parcours d'un resultSet

```
Connection conn =  
DriverManager.getConnection(url,user,password);  
Statement state= conn.createStatement();  
ResultSet resultat = state.executeQuery("SELECT *  
FROM MaTable");  
System.out.println(resultat.isBeforeFirst()); //true  
resultat.next(); //on se retrouve ici sur la première ligne  
//traitement de la première ligne ...  
while(resultat.next()) { //traitement des autres lignes }  
resultat.first();  
//on a remplacé ici le curseur sur la première ligne
```

# Étape 3: Accès à la BD

## Le parcours d'un ResultSet

eclipse-workspace - BDTest/src/AccessFenetre.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Utilitaire.java testConnection.java connectionPar.properties TestConnection.java TestResultSet.java AccessFenetre.java

```

1 import javax.swing.*;
2 import javax.swing.table.DefaultTableModel;
3 import java.sql.*;
4 public class AccessFenetre extends JFrame{
5     public AccessFenetre()
6     {ResultSet resultat;
7     try {Connection c=Utilitaire.seConnecter("ConnectionPar.properties" );
8         Statement st = c.createStatement();
9         resultat = st.executeQuery(" select * from departments order by department_id");
10        String col[] = { "ID", "Nom", "Manager", "Localisation" };
11        String cont[][] = new String[35][4];
12        int i = 0;
13        while (resultat.next()) {
14            cont[i][0] = resultat.getString(1); cont[i][1] = resultat.getString(2);
15            cont[i][2] = resultat.getString(3); cont[i][3] = resultat.getString(4);
16            i++;
17        }
18        DefaultTableModel model = new DefaultTableModel(cont, col);
19        JTable table = new JTable(model);
20        JScrollPane pane = new JScrollPane(table);
21        JFrame frame = new JFrame("Affichage JTable");
22        JPanel panel = new JPanel(); panel.add(pane); frame.add(panel);
23        frame.setSize(500, 150); frame.setVisible(true);
24    }
25    catch (SQLException e)
26    { System.out.println(e.getMessage()); }
27    public static void main(String[] args) {

```

Declaration Console Progress

AccessFenetre [Java Application] C:\Program Files\Java\jre-9\bin\javaw.exe  
Connexion établie

Affichage JTable

ID	Nom	Manager	Localisation
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700
140	Control And Credit		1700
150	Shareholder Servi...		1700
160	Benefits		1700
170	Manufacturing		1700
180	Construction		1700
190	Contracting		1700
200	Operations		1700
210	IT Support		1700

# Étape 3: Accès à la BD

## La classe ResultSetMetaData

La méthode `getMetaData()` d'un objet `ResultSet` retourne un objet de type `ResultSetMetaData`. Cet objet permet de connaître le nombre, le nom et le type des colonnes:

- **`int getColumnCount()`**: Retourne le nombre de colonnes du `ResultSet`
- **`String getColumnName(int)`**: Retourne le nom de la colonne dont le numéro est donné
- **`String getColumnLabel(int)`**: Retourne le libellé de la colonne donnée
- **`boolean isReadOnly(int)`**: Retourne true si la colonne est en lecture seule



## Étape 3: Accès à la BD

### Exemple d'exécution d'un groupe de requêtes

```
public static void main(String args[]) throws Exception{  
    Class.forName("oracle.jdbc.driver.OracleDriver");  
    Connection c=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");  
    Statement stmt=con.createStatement();  
    stmt.addBatch("insert into User values(190,'Sleh',40000)");  
    stmt.addBatch("insert into User values(191,'Oussema',50000)");  
    stmt.executeBatch();//executing the batch  
    con.commit();  
    con.close();  } }
```

## Étape 3: Accès à la BD

### Qu'est ce qu'un PreparedStatement?

L'interface **PreparedStatement** étend **Statement** et représente une instruction paramétrée. Cette interface diffère de **Statement** par deux points principaux :

- Les instances de PreparedStatement contiennent une **instruction SQL déjà compilée**. D'où le terme *prepared*. Cela améliore notamment les performances si cette instruction doit être appelée **plusieurs fois**.
- Les instructions SQL des instances de PreparedStatement contiennent **un ou plusieurs paramètres d'entrée**, non spécifiés lors de la création de l'instruction. Ces paramètres sont représentés par des **points d'interrogation(?)**. Ces paramètres doivent être spécifiés avant l'exécution.

## Étape 3: Accès à la BD

### Création d'un PreparedStatement

```
PreparedStatement prep1 = conn.prepareStatement("SELECT *  
FROM Annuaire WHERE nom = ?");  
PreparedStatement prep2 = conn.prepareStatement("UPDATE  
Annuaire SET noTel = ? WHERE nom = ?");  
PreparedStatement prep3 = conn.prepareStatement("SELECT  
Attribut1, Attribut2 FROM MaTable");
```

Le passage des paramètres d'entrée des PreparedStatement se fait grâce à l'ensemble des méthodes **setXXX()**, et la récupération de ces données se fait grâce aux méthodes **getXXX()**.

Il est important de connaître les **correspondances** entre les **types SQL** et les **types java**

# Étape 3: Accès à la BD

## Création d'un PreparedStatement

Type SQL	Type Java	Méthode <i>getter</i>
CHAR VARCHAR	String	getString()
INTEGER	int	getInt()
TINYINT	byte	getByte()
SMALLINT	short	getShort()
BIGINT	long	getLong()
BIT	boolean	getBoolean()
REAL	float	getFloat()
FLOAT DOUBLE	double	getDouble()
NUMERIC DECIMAL	java.math.BigDecimal	getBigDecimal()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getString()

## Étape 3: Accès à la BD

### Exemple de traitement d'une requête précompilée

```
String sql = "UPDATE Stocks SET prix = ?, quantite = ? WHERE nom = ?";
```

```
PreparedStatement p1 = conn.prepareStatement(sql);
```

```
p1.setFloat(1, 15.6);
```

```
p1.setInt(2, 256);
```

```
p1.setString(3,"café");
```

```
p1.executeUpdate();
```

# Étape 3: Accès à la BD

## La gestion de SQLException

La classe `SQLException` représente les erreurs émises par la base de données. Elle contient trois attributs qui permettent de préciser l'erreur :

- **message** : contient une description de l'erreur
- **SQLState** : code défini par les normes X/Open et SQL99
- **ErrorCode** : le code d'erreur du fournisseur du pilote

La classe `SQLException` possède une méthode `getNextException()` qui permet d'obtenir les autres exceptions levées durant la requête. La méthode renvoie null une fois la dernière exception renvoyée.

Exemple:

```
try {... }  
catch (SQLException e) { System.out.println("SQLException");  
do {  
    System.out.println("SQLState : " + e.getSQLState());  
    System.out.println("Description : " + e.getMessage());  
    System.out.println("code erreur : " + e.getErrorCode());  
    e = e.getNextException();  
} while (e != null);}
```

## Étape 4: Déconnexion

31

La méthode ***close()*** permet de libérer les ressources prises par la création d'objets de type ***ResultSet***, ***Statement***, et ***Connection***. Elle existe pour chacune de ces interfaces. Elle est le plus souvent employée pour une ***Connection***, car elle libère en même temps toutes les ressources qui lui sont associées.

Il faut commencer par libérer le ***ResultSet***, puis le ***Statement*** et enfin la ***Connection***.

# Exemple complet

32

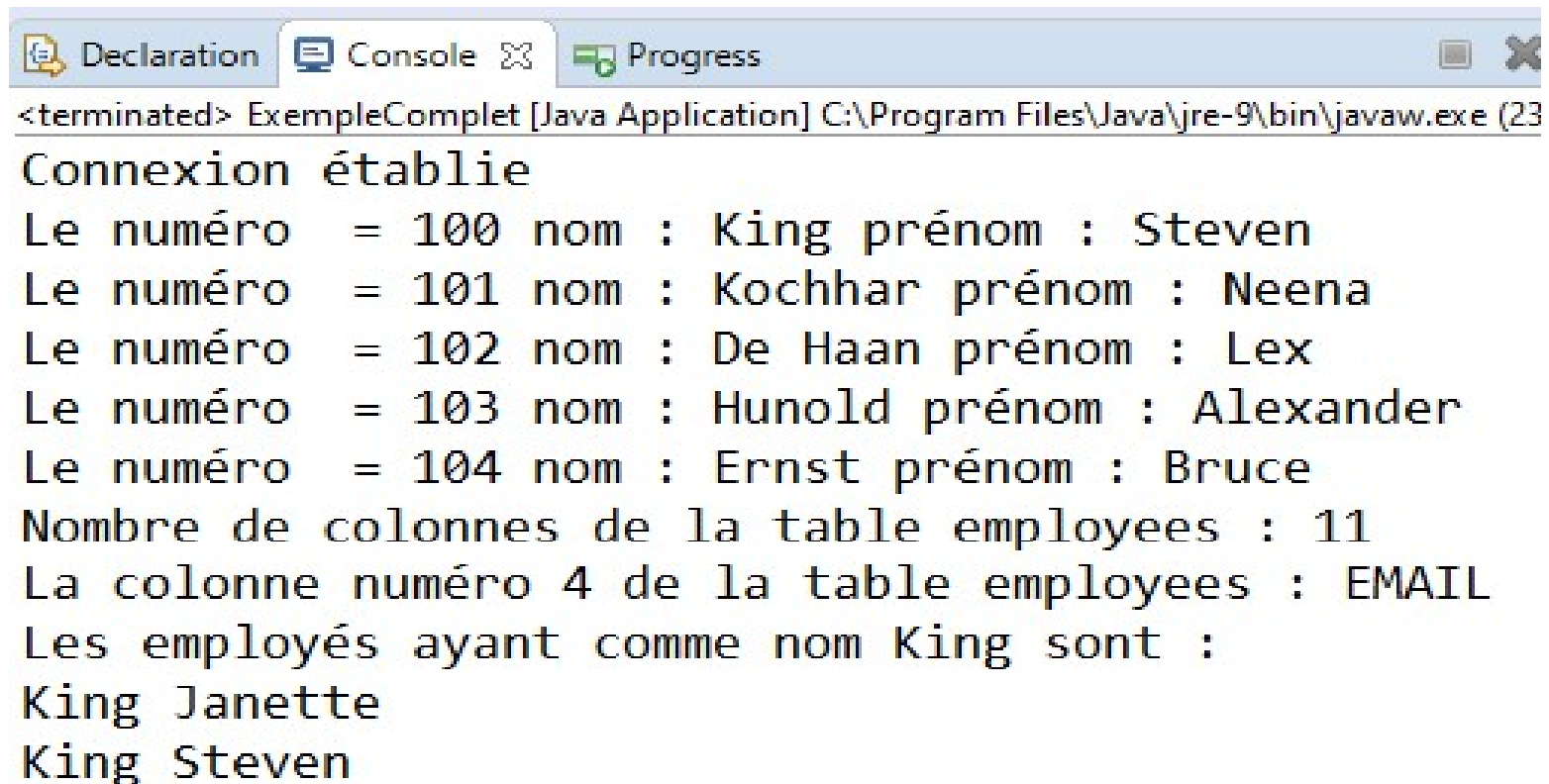
```
eclipse-workspace - BDTest/src/ExempleComple.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

1 import java.sql.*;
2 public class ExempleComple {
3     public static void main(String[] args) throws SQLException {
4         Connection c = null;
5         ResultSet res = null;
6         c = Utilitaire.seConnecter("ConnectionPar.properties");
7         res = Utilitaire.OuvrirReq( c, "select * from employees");
8         for (int i=0; i<5; i++)
9         {
10             res.next();
11             System.out.println ("Le numéro = " + res.getInt(1) + " nom : " +
12             res.getString("Last_Name") + " prénom : " + res.getString("First_Name"));
13         }
14
15         System.out.println("Nombre de colonnes de la table employees : " + res.getMetaData().getColumnCount());
16         System.out.println("La colonne numéro 4 de la table employees : " + res.getMetaData().getColumnLabel(4));
17
18         PreparedStatement s = c.prepareStatement("select last_name, first_name from employees where last_name = ?");
19         s.setString(1, "King");
20         ResultSet r = s.executeQuery();
21         System.out.println ("Les employés ayant comme nom King sont :");
22         while (r.next())
23         {
24             System.out.println (r.getString("Last_Name") + " " + r.getString("First_Name"));
25         }
26     }
27 }
```



# Exemple complet

33



The screenshot shows a Java IDE window with three tabs: Declaration, Console, and Progress. The Console tab is active, displaying the output of a Java application. The output text is as follows:

```
<terminated> ExempleComplet [Java Application] C:\Program Files\Java\jre-9\bin\javaw.exe (23
Connexion établie
Le numéro = 100 nom : King prénom : Steven
Le numéro = 101 nom : Kochhar prénom : Neena
Le numéro = 102 nom : De Haan prénom : Lex
Le numéro = 103 nom : Hunold prénom : Alexander
Le numéro = 104 nom : Ernst prénom : Bruce
Nombre de colonnes de la table employees : 11
La colonne numéro 4 de la table employees : EMAIL
Les employés ayant comme nom King sont :
King Janette
King Steven
```