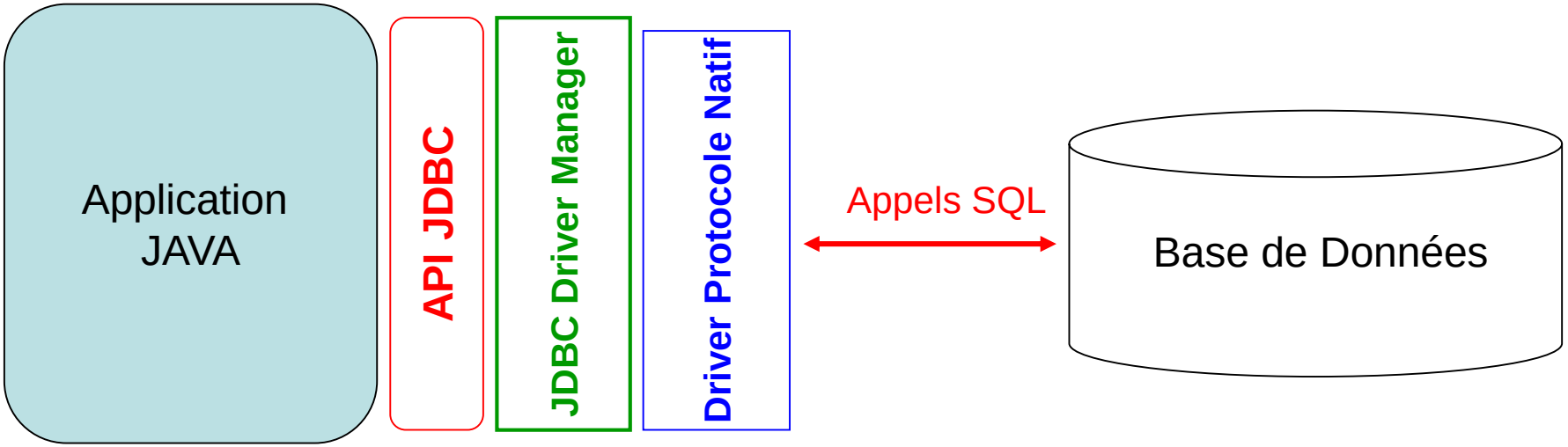
JDBC: manipuler une base de données en Java

Intermédiaire

- JDBC: Java database connectivity
  - Cette API à était développée par SUN pour permettre à des applications Java d'accéder à des bases de données relationnelles quelconques.
- Les étapes principales
  - Se connecter à une base de données
  - Envoyer une requête SQL
  - Manipuler le résultat
- JDBC: un driver (pilot) fournissant des outils pour ces fonctions



Définition

Le pilote...

établit le lien avec la base de donnees, en sachant “lui parler"

La connexion...

Elle peut s'etablir SI on donne l'adresse de la BD a laquelle se connecter...

Préparatif

- Installer un driver JDBC
  - E.g. SQL server 2000 de Microsoft

    http://msdn2.microsoft.com/en-us/sql/aa336272.aspx
  - pont ODBC/JDBC (Open DataBase Connectivity)

  - Pour MySQL, recuperer le .jar correspondant

    https://dev.mysql.com/downloads/connector/j/

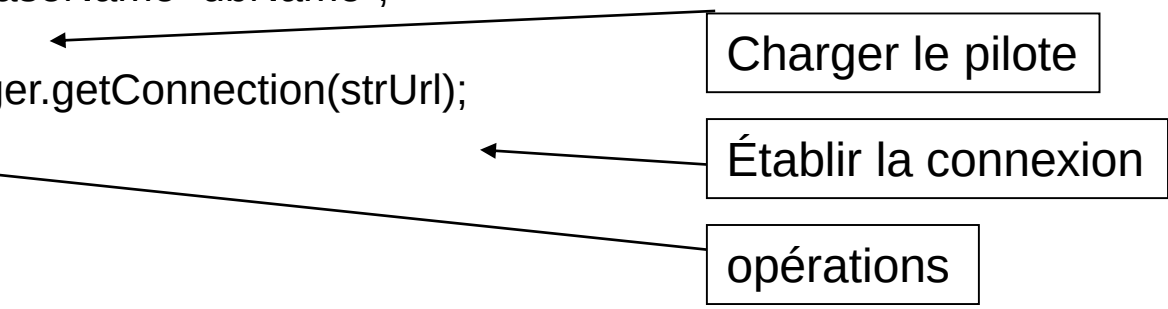
Étape 1: charger le pilote

- Charger le pilote (driver)
  - Pilote: contient toutes les classes nécessaire pour communiquer avec une base de données
  - il faut utiliser la méthode forName de la classe Class
  - E.g.
    - SQL Server 2000:
      Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
    - Pour MySQL
      Class.forName(« com.mysql.jdbc.Driver");
  - Cette méthode charge en mémoire la classe demandée et exécute son éventuel bloc static.
    - static { BaseDriver.registerDriver(new SQLServerDriver()); }
  - Pour que cela fonctionne, il faut définir la variable d'environnement CLASSPATH pour inclure le répertoire contenant les classes du driver

Étape 2: établir une connexion

- Pour établir la connexion avec SQL Server, il faut préciser
  - le nom de la machine (ou son numéro IP),
  - le port où le service SQL est démarré (quasiment toujours 1433),
  - le nom de la base de données,
  - le login utilisé ainsi que son mot de passe.

```
try { String strClassName = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
      String strUrl = "jdbc:microsoft:sqlserver://hostname:1433;" +
      "user=sa;password=pass;DatabaseName=dbName";
      Class.forName(strClassName);
      Connection conn = DriverManager.getConnection(strUrl);
      // . . .
conn.close();
}
catch(ClassNotFoundException e) {
      System.err.println("Driver non chargé !");
      e.printStackTrace();
} catch(SQLException e) {
      // . . .
}
```



Étape 2: établir une connexion

- Établir la connexion avec MySQL
  - DriverManager: la méthode statique getConnection va créer un objet de connexion
  - Paramètre: le protocole et le sous-protocole:
    - jdbc:odbc:DsnName
    - DSN (Data Source Name)

```
Connection conn = null;
try {
  Class.forName("com.mysql.jdbc.Driver");
  conn = DriverManager.getConnection("jdbc:mysql://localhost");
  // . . .
  conn.close();
 } catch(ClassNotFoundException e) {
   System.err.println("Driver non chargé !");
   e.printStackTrace();
 } catch(SQLException e) {
   // . . .
}
```

Étape 3: Requête SQL

- L’exécution d’une requête SQL passe par l'utilisation d'une classe, spécifique au pilote utilisé, implémentant l'interface **Statement**
- Un objet de type **Statement** se doit d'être adapté à la base manipulée. JDBC ne fournit que l'interface Statement, qui est implantée par différentes classes dans un pilote
- Obtenir un objet Statement: avec la méthode **createStatement**.

Exemple

```
try { String strClassName = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
String strUrl = "jdbc:microsoft:sqlserver://hostname:1433;"
                + "user=sa;password=pass;DatabaseName=dbName";
String strInsert = "INSERT INTO T_Users "
      + "(Login, Password, ConnectionNumber) "
      + "VALUES ('Toto', 'Titi', 0);";
Class.forName(strClassName);
Connection conn = DriverManager.getConnection(strUrl);

Statement stAddUser = conn.createStatement();
stAddUser.executeUpdate(strInsert);

conn.close();
}
catch(ClassNotFoundException e) {
      // . . .
} catch(SQLException e) {
      // . . .
}
```

Créer un Statement

Exécuter un ordre SQL

Données de la table « T_Users » dans « PangeeN...

Id	Login	Password	ConnectionNumber
1	SkyWalker	Luc	30
2	Plisken	Snake	18
3	Ripley	Helen	19
4	Gordon	Flash	12
5	Kent	Clark	1
*			

Exécuter une requête SELECT

- l'ordre SQL "SELECT * FROM T_Users;"
- L'appel à "executeQuery" renvoie au final un objet de type **ResultSet**

```
try {
    String strClassName = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
    String strUrl = "jdbc:microsoft:sqlserver://hostname:1433;"
            + "user=sa;password=pass;DatabaseName=dbName";
    String strQuery = "SELECT * FROM T_Users;";
    Class.forName(strClassName);
    Connection conn = DriverManager.getConnection(strUrl);
    Statement stLogin = conn.createStatement();

    ResultSet rsLogin = stLogin.executeQuery(strQuery);
    // . . . Utilisation du ResultSet . . .
    conn.close();
}
catch(ClassNotFoundException e) {
    // . . .
} catch(SQLException e) {
    // . . .
}
```

requête

Exécuter la requête et stocker le résultat

Manipuler le résultat

- On peut identifier chaque colonne de la base de donnée
  - Par son index
  - Par son nom

```
String strQuery = "SELECT * FROM T_Users;";
ResultSet rsUsers = stUsers.exexcuteQuery(strQuery);
while(rsUsers.next()) {
    System.out.print("Id[" + rsUsers.getInt(1) + "]"
          + rsUsers.getString(2)
          + "[" + rsUsers.getString("Password") + "] "
          + rsUsers.getInt("ConnectionNumber") ); }
rsUsers.close();
```

Plusieurs mode de parcours

st = conn.createStatement(type, mode);

Type ==

ResultSet.TYPE_FORWARD_ONLY

ResultSet.TYPE_SCROLL_SENSITIVE

ResultSet.TYPE_SCROLL_INSENSITIVE

Mode ==

ResultSet.CONCUR_READ_ONLY

ResultSet.CONCUR_UPDATABLE

Modifier le résultat ou la base

- Se positionne sur le premier enregistrement
  - rsUsers.first();
- Ou avancer jusqu’à l’élément voulu
- Modifie la valeur du Password dans le résultat
  - rsUsers.updateString("Password", "toto");
- Pour appliquer les modifications dans la base de données:
  - rsUsers.updateRow();

Autres opérations

- Exécuter des requêtes SQL pré-compilées

java.sql.PreparedStatement

- Exécuter des procédures stockées java.sql.CallableStatement
- Utiliser des transactions
- Accéder aux méta-données (schéma) de la base

java.sql.DatabaseMetaData

- Manipuler des BLOBs

JDBC

Quelques notions de Meta Données (Metadata)

1. Les meta données sont des données sur les données.

2. Chaque ResultSet possède ses propres MetaDonnées.

3. Elles sont utilisées pour obtenir les noms des colonnes dans un ResultSet ainsi que le type des données qui se trouvent dans chacune d'elles.

```
try
{
   // Obtenir les en-têtes de colonne.
   ResultSetMetaData rsmd = rs.getMetaData();

   for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
      enTeteColonne.addElement( rsmd.getColumnName( i ) );
```

Obtenir les meta-données

Utiliser les meta-données

Les méta données sont stockées dans un ResultSetMetaData.

Requetes préparées

```
String anSQLquery = "SELECT * FROM employe WHERE age > ?
";
// Création d'un PreparedStatement
PreparedStatement pst = conn.prepareStatement(anSQLquery );
// => requête SQL compilée par le SGBD
// Passage des paramètres par setXXX
pst.setInt(1, 55); // => age > 55
//Exécution de la requête
```

Exercice

- Créer une classe Java J2SE
- Etablir une connexion MySQL pour :
- - créer une table PERS(id, login, pwd)
- - insérer une personne dedans
- - afficher toutes les personnes présentes

- Utiliser le même mécanisme dans des servlets pour créer un accès à une personne, vérifier les acces aux forms, …