# FORMATION
# ACCÈS BASE DE DONNÉES EN JAVA

# ACCÈS AUX BASES DE DONNÉES



- **L'accès aux SGBD se fait grâce à l'API JDBC.**

- **L 'API JDBC permet un accès universel aux données et à la structure de ces données, contenues dans une Base de Données Relationnelle.**

- **L'API JDBC est contenue dans les packages**
  - java.sql (jSE)
  - javax.sql (jEE)

## Java.sql fournit toutes les fonctionnalités primaires pour l'accès aux BDD :

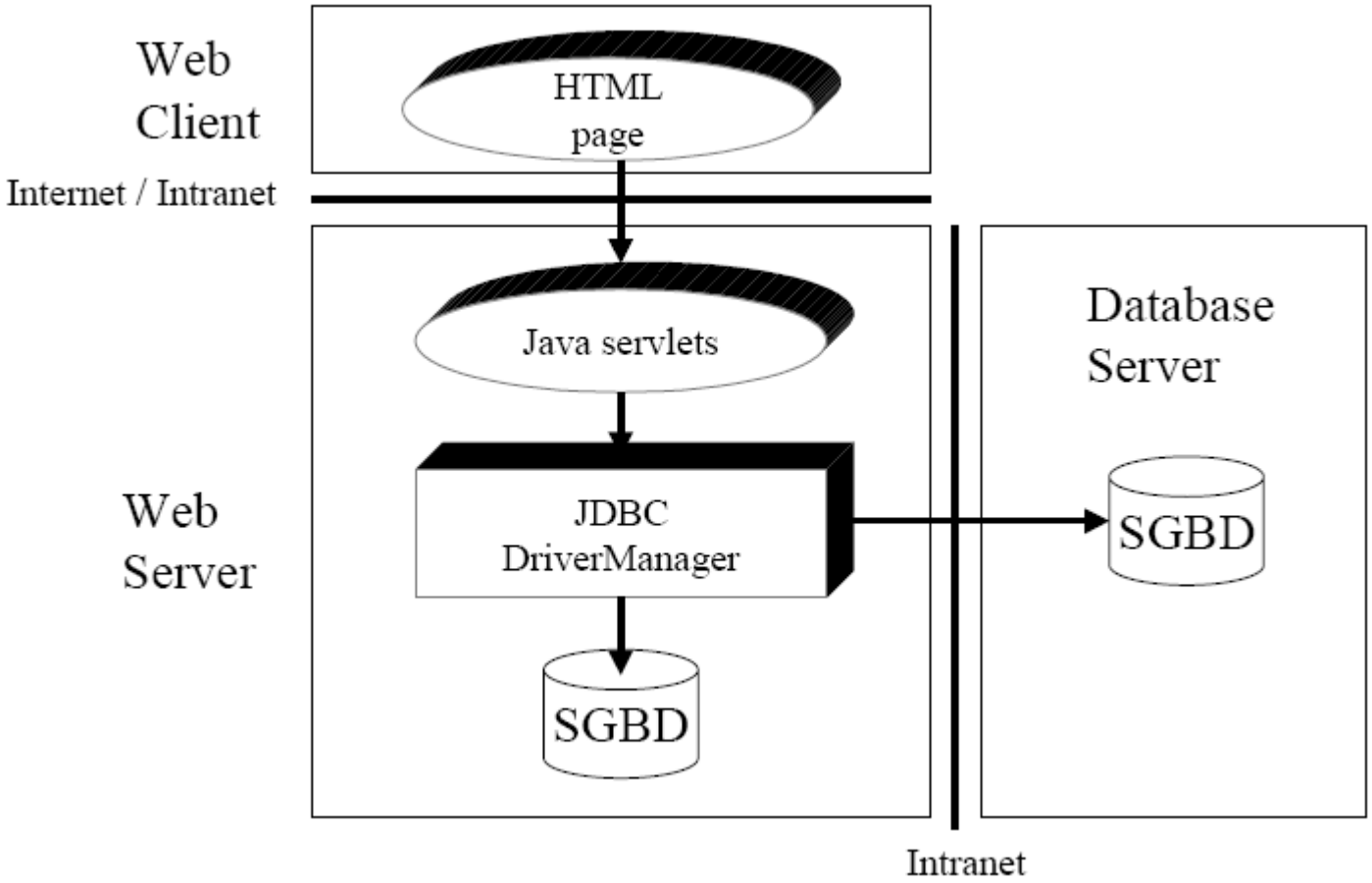
- Connexion à la base
- Envoi de résultat de l'exécution des requêtes
- Lecture des requêtes
- Traitement des méta-informations (structures de la base)

# PACKAGES DE JDBC

- **Javax.sql fournit des fonctionnalités liées à la vocation « entreprise » des applications :**
  - Accès aux données étendu à toutes les sources de données, représentées par l'objet DataSource
  - L'objet DataSource est créé à l'aide d'un service d'annuaire JNDI ( Java Naming Directory Interface) qui contient toutes les informations nécessaires à l'établissement d'une connexion
  - L'objet DataSource gère un pool de connexions permettant d'optimiser les temps de traitement des opérations

# CONNEXION À UNE BASE DE DONNEES





# INTERFACE JAVA.SQL.DRIVER



- **Chaque vendeur fournit une implémentation**
  - Certaines sont fournis avec le Java core
  → Accès à ODBC sun.jdbc.odbc.JdbcOdbcDriver
  - Les autres doivent être importées
  → postGresql : org.postgresql.Driver

- **Généralement on trouve ces classes dans des fichiers jar dans les distributions du constructeur**
- **Dans le pire des cas :**
  - On peut trouver une liste de plus de 160 pilotes à http://industry.java.sun.com/products/jdbc/dri

# ACCÈS AUX DONNÉES - 5 ÉTAPES



1. Charger un pilote
2. Créer une connexion à la base
3. Créer une requète (Statement)
4. Exécuter une requête
5. Présenter les résultats

# PREMIÈRE ÉTAPE "*CHARGER UN PILOTE*"

# CHARGER UN PILOTE



- **Classe `java.sql.DriverManager`**
  - Fournit les utilitaires de gestion des connexions et des pilotes
  - Elle ne contient que des méthodes statiques

- **Les pilotes sont chargés dynamiquement**
  - par la méthode statique `forName()` de la classe `Class`

# EXEMPLES DE CHARGEMENT

- **Initialisation dans le programme**

```
try {
    Class.forName("nom-de-classe");
} catch(ClassNotFoundException ex)
    {...}
```

# SECONDE ÉTAPE "*CRÉER UNE CONNEXION À LA BASE*"

# CLASSE JAVA.SQL.CONNECTION



- **Demande de connexion**
  - méthode statique `getConnection(String)` classe `DriverManager`
- **Le driver manager essaye de trouver un driver approprié d'après la chaîne passée en paramètre**
- **Structure de la chaîne décrivant la connexion**

  `jdbc:`*`protocole`*`:`*`URL`*

  Exemples

  `jdbc:odbc:epicerie`

  `jdbc:mysql://localhost:3306/db`

  `jdbc:oracle:thin:@blabla:1715:test`

# CLASSE JAVA.SQL.CONNECTION (SUITE)



- **Connexion sans information de sécurité**

```
Connection con =
  DriverManager.getConnection
  ("jdbc:odbc:epicerie");
```

- **Connexion avec informations de sécurité**

```
Connection con =
  DriverManager.getConnection
  ("jdbc:odbc:epicerie", user, password);
```

- **Dans tous les cas faut récupérer l'exception** `java.sql.SQLException`

# TROISIÈME ÉTAPE "*CREER UNE REQUÈTE*"

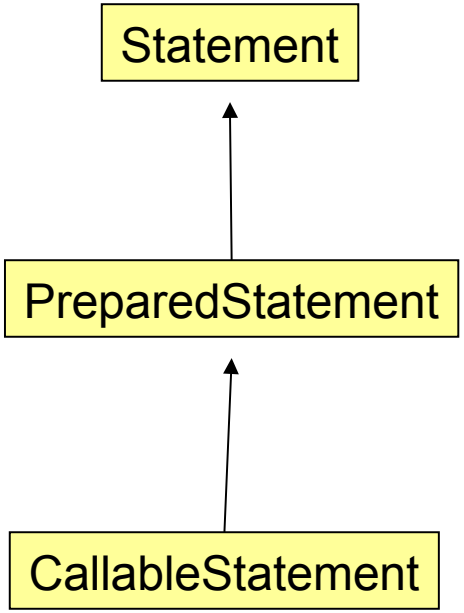
# TROIS TYPES DE REQUÊTES



- **Requètes simples**
  - `java.sql.Statement`
- **Requètes Précompilées**
  - `java.sql.PreparedStatement`
  - Pour une opération réalisée à plusieurs reprises
- **Procédures stockées**
  - `java.sql.CallableStatement`
  - Pour lancer une procédure du SGBD

# HÉRITAGE ENTRE INTERFACES REQUÊTES





# INTERFACE JAVA.SQL.STATEMENT



- **C'est le type d'objet**
  - pour exécuter une requète SQL simple
  - pour recevoir ses résultats
- **Exemple de création**

```
Connection con =
  DriverManager.getConnection
  ("jdbc:odbc:epicerie");

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

# STATEMENT LA REQUETE SQL N'EST PAS LIEE A L'OBJET STATEMENT

- **Initialisation**

```
Connection con = getConnection();
Statement stmt = con.createStatement();
```

- **Exécuter une requête**

```
String sql = "SELECT * FROM fournisseur";
ResultSet rs = stmt.executeQuery(sql);
```

# PREPAREDSTATEMENT
# LA RÉQUETE SQL EST LIÉE À L'OBJET PREPAREDSTATEMENT

- Requètes paramétrées

```
String s = "UPDATE EMPLOYEES" +
" SET SALARY = ? WHERE ID = ?"
PreparedStatement pstmt =
  con.prepareStatement(s);


pstmt.setBigDecimal(1, 153833.00);
pstmt.setInt(2, 110592);
```

# QUATRIÈME ÉTAPE "*EXÉCUTER UNE REQUÊTE*"

## Instruction SQL « select »

```
String sql = "SELECT * FROM fournisseur";

ResultSet rs = stmt.executeQuery(sql);
```

# MISE À JOUR : EFFACER UNE TABLE



## Instruction SQL « drop table »

```
Statement stmt = con.createStatement();
int i = stmt.executeUpdate(
   "drop table fournisseur");
```

# STATEMENT DEUX TYPES D'EXÉCUTION



## Requètes "select" retournant un tableau

```
Statement stmt = con.createStatement();
String sql = "SELECT * FROM fournisseur";
ResultSet rs = stmt.executeQuery(sql);
```

## Requètes de mise à jour et de création

```
Statement stmt = conn.createStatement();
String sql = "INSERT INTO Customers " +
             "VALUES (1001, 'Simpson', 'Mr.', " +
```

# EXECUTEQUERY ()



- **Pour lancer une requète "SELECT" statique**

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


String s = "select * from employes";

ResultSet rs = stmt.executeQuery(s);
```

- **Un `ResultSet` est un objet qui modélise un tableau à deux dimensions**
  - Lignes (*row*)
  - Colonnes (valeurs d'attributs)

# EXECUTEUPDATE () - 1 INSERT INTO



- **Pour lancer une requète `INSERT`**

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


String s = "INSERT INTO test (code,val)" +
    "VALUES(" + valCode + ", '" +val +"')";
int i = stmt.executeUpdate(s);
```

- **Le résultat est un entier donnant le nombre de lignes créées**

# EXECUTEUPDATE () - 2 UPDATE



- **Pour lancer une requète UPDATE**

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


String s = "UPDATE table
  SET column = expression
  WHERE predicates";
int i = stmt.executeUpdate(s);
```

- **Le résultat est un entier donnant le nombre de mises-à-jour**

# EXECUTEUPDATE () - 3 DELETE



- **Pour lancer une requète DELETE**

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


String s = "DELETE FROM table
  WHERE predicates";
int i = stmt.executeUpdate(s);
```

- **Le résultat est un entier donnant le nombre de d'effacements**

```
Class.forName(driverClass);
java.sql.Connection connection =
  java.sql.DriverManager.getConnection(url,user
  ,password);


 PreparedStatement pstmt =
connection.prepareStatement(
   "insert into users values(?,?,?,?)");
    pstmt.setInt(1,user.getId());
    pstmt.setString(2,user.getName());
    pstmt.setString(3,user.getEmail());
    pstmt.setString(4,user.getAddress());
    pstmt.execute();
```

# CINQUIÈME ÉTAPE "*PRÉSENTER LES RÉSULTATS*"

# RÉCUPÉRATION DES RÉSULTATS

```
java.sql.Statement stmt =
  conn.createStatement();
String s = "SELECT a, b, c FROM Table1";
ResultSet rs = stmt.executeQuery(s);
while (rs.next()) {
    int i = rs.getInt("a");
    String s = rs.getString("b");
    byte b[] = rs.getBytes("c");
    System.out.println("ROW = " + i +
        " " + s + " " + b[0]);
}
```

# OBJETS RESULTSET



```
ResultSet rs = stmt.executeQuery(s);
```

- Se parcourt itérativement *row* par *row*
- Les colonnes sont référencées par leur numéro ou leur nom
- L'accès aux valeurs des colonnes se fait par des méthodes getXXX() où XXX représente le type de l'attribut se trouvant dans la colonne

# OBJETS RESULTSET (SUITE)



- Une rangée *(row)* est un tuple
- un RS contient un curseur pointant sur une rangée du résultat
- au départ le pointeur est positionné a*vant la 1ère rangée*
- La méthode next() fait passer à l'enregistrement suivant.
- Le premier appel resultat.next() positionne sur la première rangée
- next() renvoie un booléen true en général, et false lorsque l'on a dépassé le dernier enregistrement.

# POOLS DE CONNEXION ET DATASOURCE JDBC



L'utilisation d'un pool de connexion possède plusieurs avantages

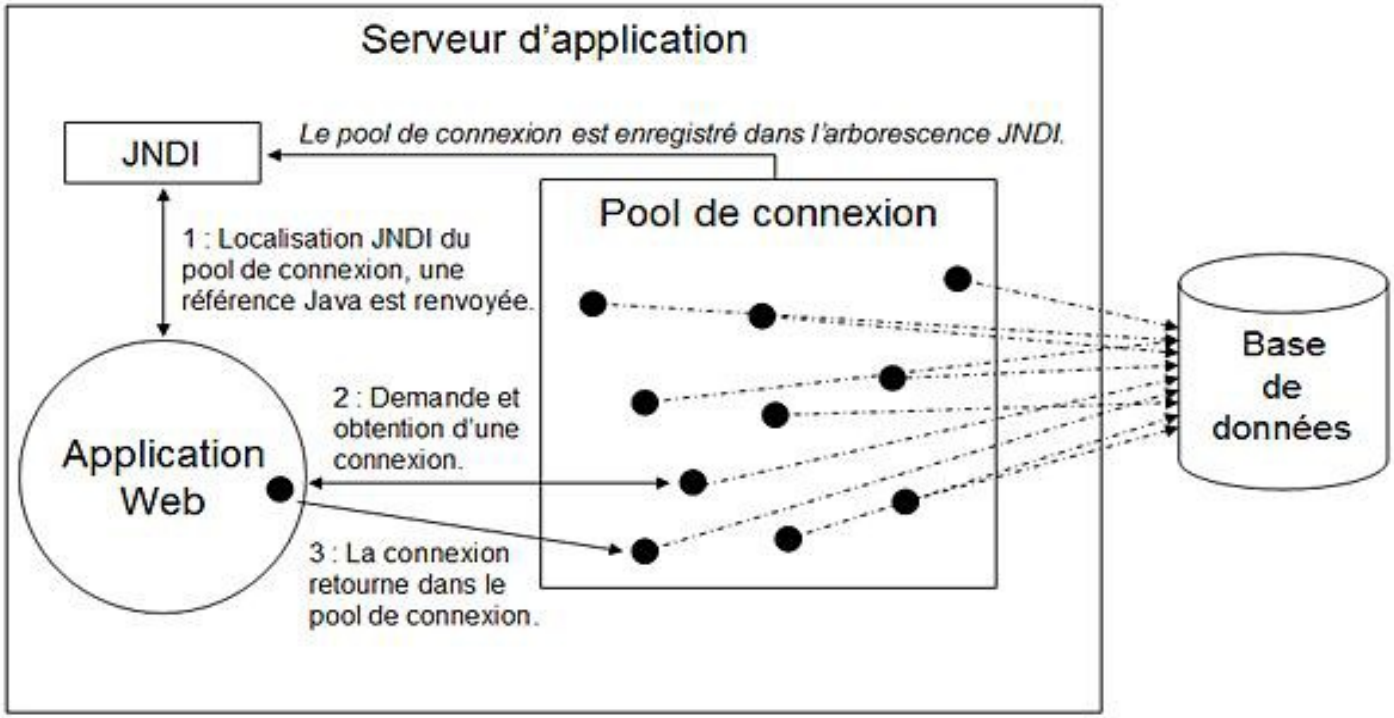
- les applications n'ont plus la responsabilité de créer et de détruire les connexions
- les connexions ne sont pas détruites à la fin de leur utilisation
- l'abstraction de l'application par rapport à la base de données
- un même pool de connexion est utilisable par plusieurs applications simultanément

# LE POOLING DE CONNEXION JDBC



## Pour obtenir une connexion à la base, l'application doit :

- localiser le pool de connexion en faisant une recherche JNDI :
- (



## TP: Intégration Base de données