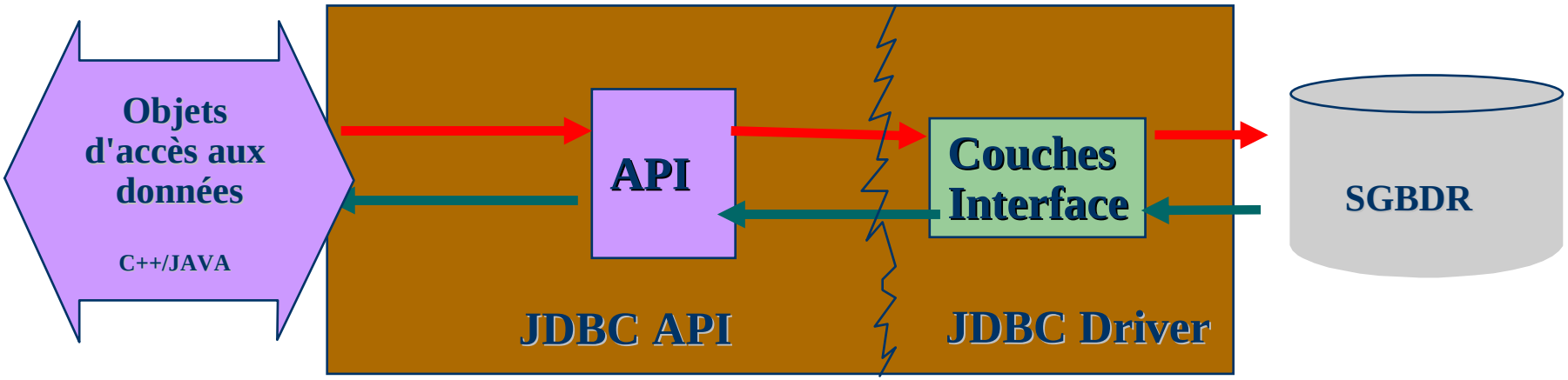
le cnam

JDBC
Java DataBase Connectivity

P. Graffion

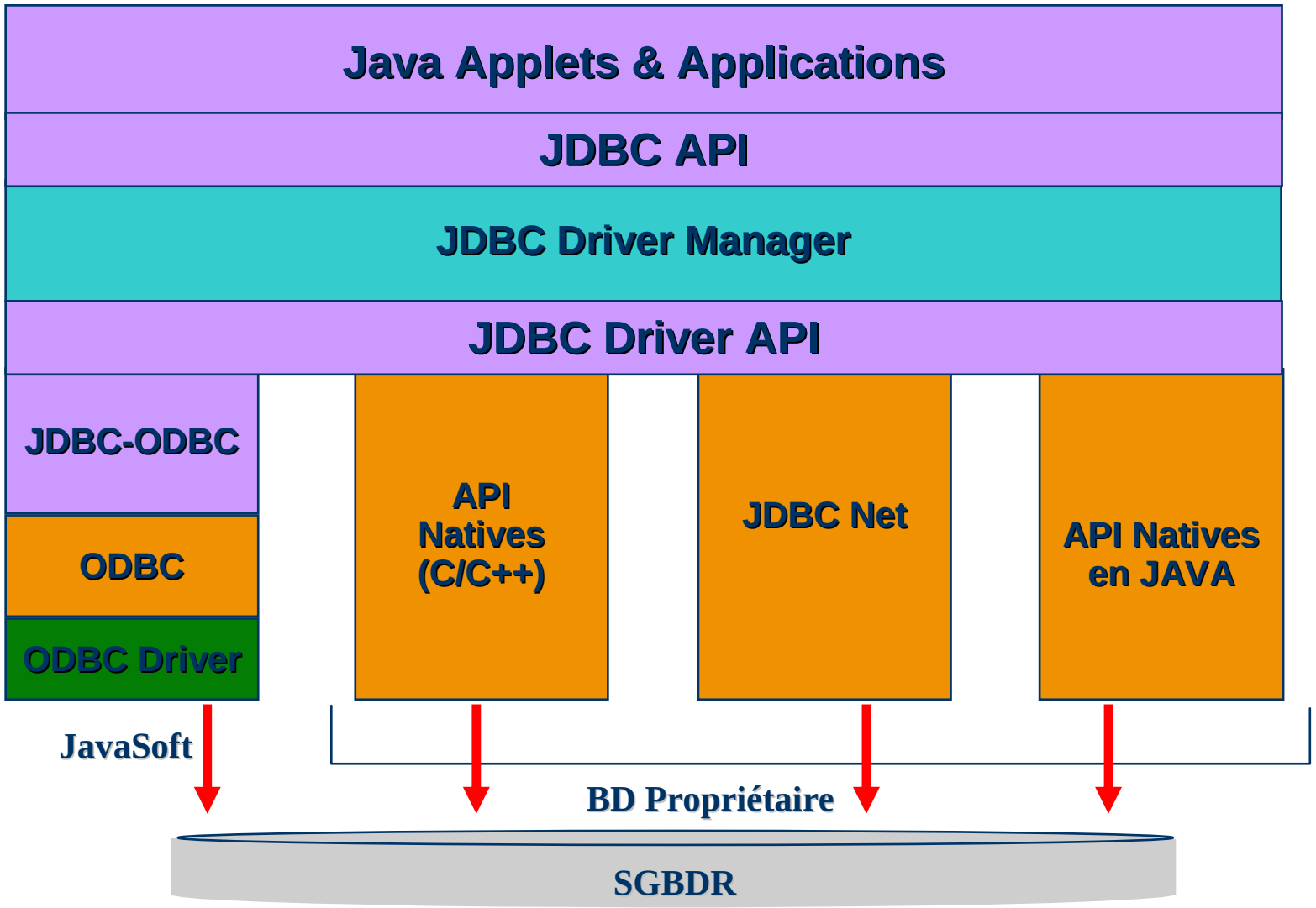
Définition

- Package Java pour l'accès aux SGDBR : java.sql
- API unique d'accès à tout SGBD conforme au standard SQL-3EntryLevel
- Permet d'envoyer des requêtes SQL (sur le réseau)

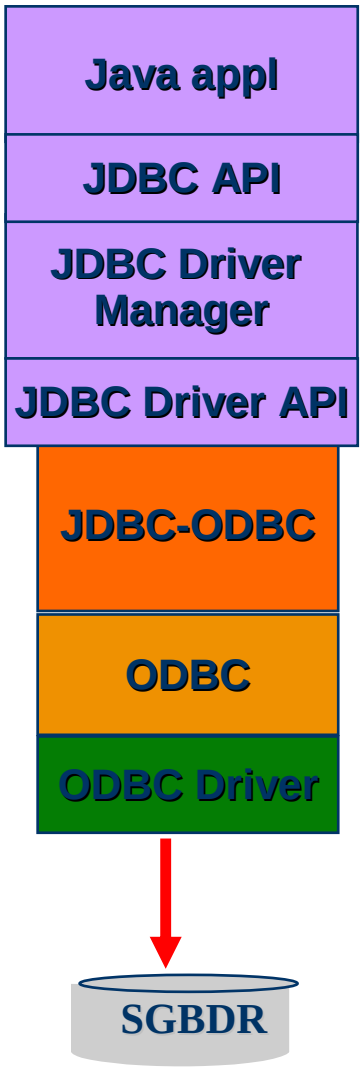


Middleware JDBC

Architecture



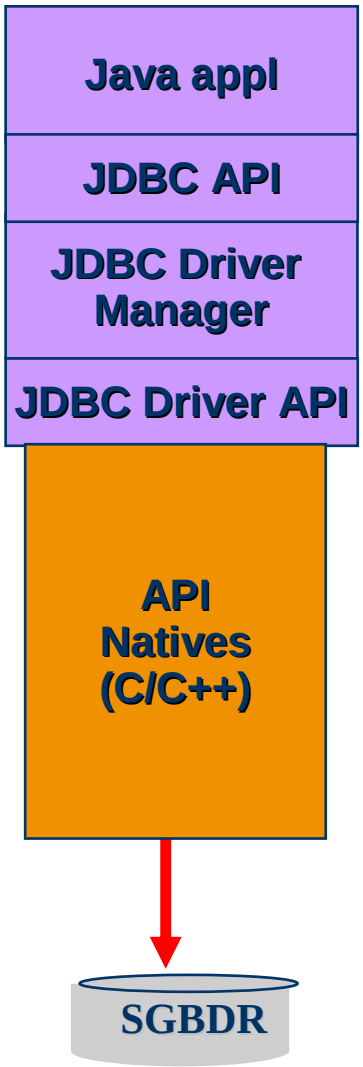
Driver JDBC de type 1



- **Pont JDBC-ODBC :**
  appels JDBC traduits en appels ODBC
- **fourni par Sun :**
  sun.jdbc.odbc.JdbcOdbcDriver
- **requiert l'installation d'un driver ODBC sur le client**
- **code ODBC natif (C)**

  => ne peut pas être utilisé par une applet "untrusted"

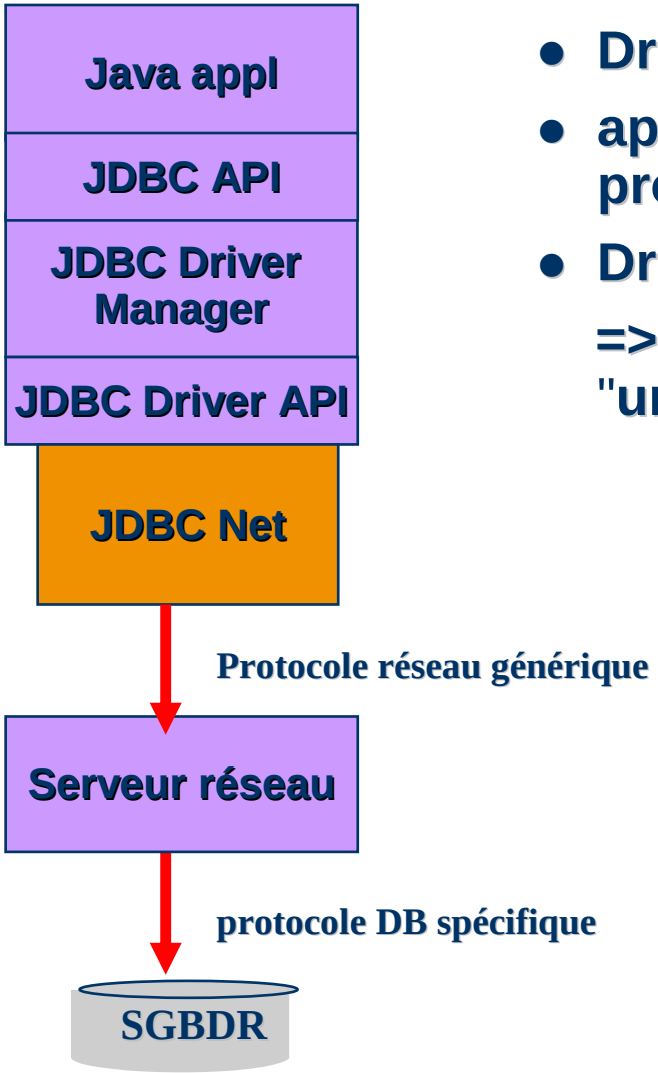
Driver JDBC de type 2



- Pont natif (C/C++)

  => ne peut pas être utilisé par une applet "untrusted"
- moins ouvert que le pont JDBC-ODBC
- plus performant

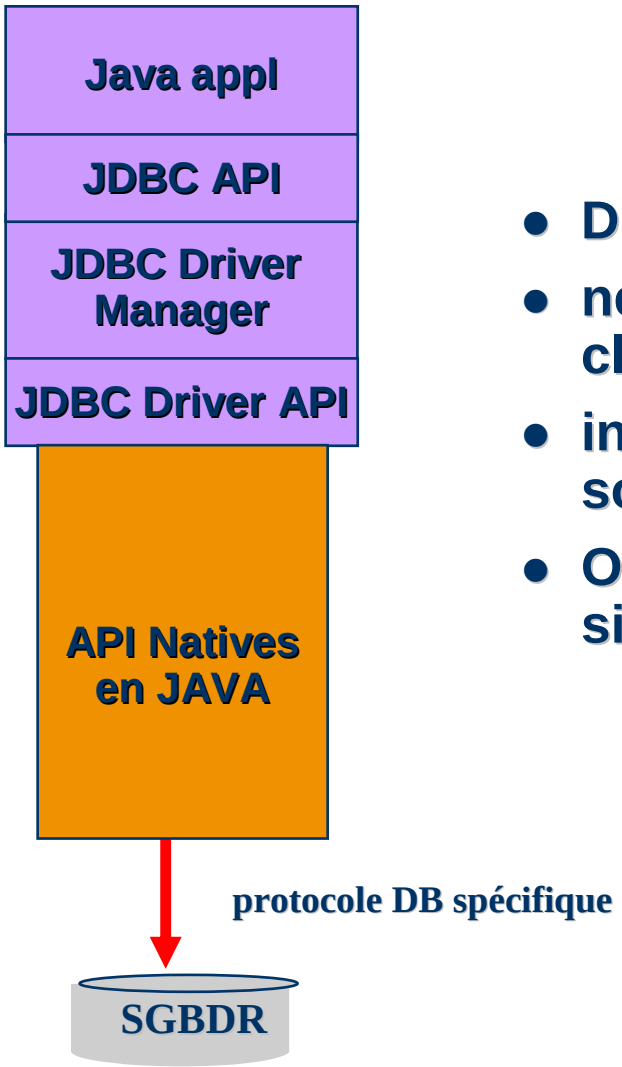
Driver JDBC de type 3



- Driver JDBC-Net
- appels JDBC convertis suivant un protocole réseau générique
- Driver 100% Java

  => peut être utilisé par une applet "untrusted"

Driver JDBC de type 4

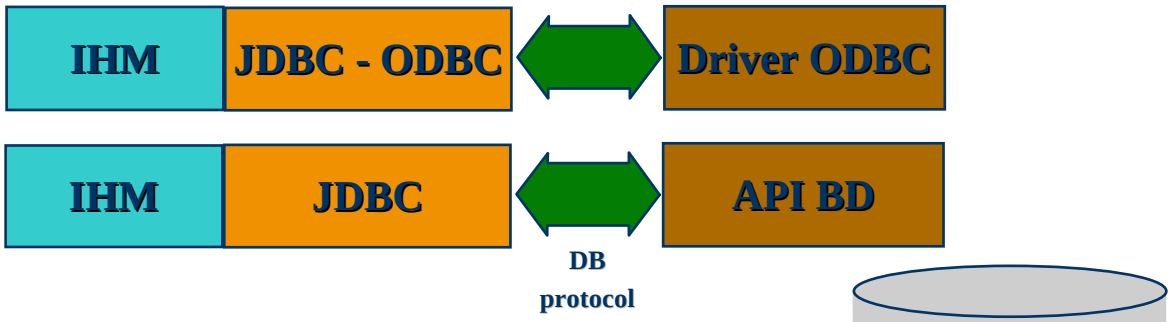


- Driver 100% Java
- ne requiert aucune configuration sur le client
- interaction directe avec le SGBD via sockets
- OK pour applet "untrusted" ...
  si le SGBD est sur le serveur web

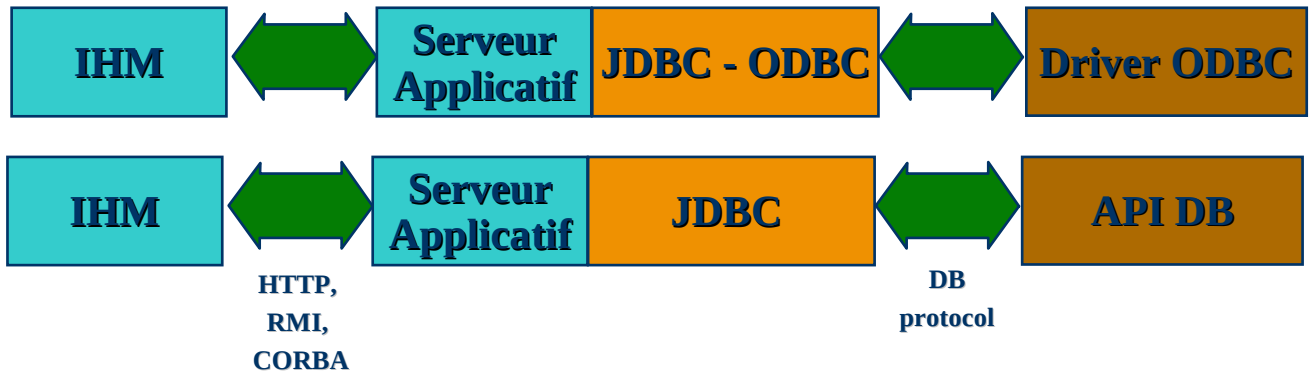
Architectures 2-tier et 3-tier

- **Architecture 2-tier**
  - JDBC est téléchargé
  - Le Driver reste sur le serveur

JDBC-ODBC
- Traduit les appels JDBC en ODBC
- Appel à un driver local qui n'est pas écrit en Java



- **Architecture 3-tier**
  - JDBC est sur le serveur applicatif
  - Le Driver reste sur le serveur



API JAVA

- **java.sql.DriverManager**
- **java.sql.Connection**

Classes de connexion

- **java.sql.Statement**
- **java.sql.PreparedStatement**
- **java.sql.CallableStatement**
- **java.sql.ResultSet**

Interfaces de requête sql

- **java.sql.DatabaseMetaData**

Classe de traitement des résultats

Classe de récupération d 'info sur la structure de la BD

Mise en oeuvre

1. Enregistrer le driver JDBC
2. Ouvrir une connexion sur la base
3. Créer la requête (un Statement)
4. Exécuter la requête
5. Traiter le résultat retourné (un ResultSet)
6. Fermer les objets

Enregistrement du driver

Fait en instanciant une classe implémentant ce driver

2 possibilités :

- DriverManager.registerDriver(new com.mysql.jdbc.Driver());

ou

- Class.forName("com.mysql.jdbc.Driver");

URL JDBC

- Identifie une BD en spécifiant :
  - un protocole : jdbc
  - un sous-protocole : nom d'un driver (ex.: odbc)
  - (un URL distant ://host:port si connection réseau)
  - identificateur de BD (nom logique défini par l 'administrateur de BD)

- Exemple : jdbc:odbc://www.xyz.com/employeDB

Connexion à la base

```
String dbUrl = "jdbc:mysql://localhost:3306/yaps";
String user = "root",  password = "";
Connection conn ;
conn = DriverManager.getConnection(dbUrl, user, password);
```

Exemple de schéma SQL

```
CREATE TABLE employe (
    id INT UNSIGNED PRIMARY KEY,
    nom VARCHAR(50),
    prenom VARCHAR(40),
    age INT UNSIGNED
);

INSERT INTO employe VALUES (1, 'Gates', 'Bill', 50);
INSERT INTO employe VALUES (2, 'Jobs', 'Steve', 50);
```

Invocation d'une requête SQL de sélection

```
String anSQLquery =  "SELECT prenom, nom, age FROM employe";
// Création d'un statement
Statement st = conn.createStatement();
// Exécution de la requête
ResultSet r = st.executeQuery(anSQLquery);
```

```
conn.createStatement();
<=> conn.createStatement(TYPE_FORWARD_ONLY,CONCUR_READ_ONLY);
        - le curseur se déplace uniquement vers l'avant (par next())
        - les données dans le ResultSet ne seront pas modifiables
```

Parcours du résultat - ResultSet 1/2

- executeQuery( ) retourne une instance de ResultSet
- permet de parcourir l'ensemble des lignes de la table :
  - Une seule possibilité avec JDBC 1.x : next()
  - plus riche en JDBC 2.0 : previous(), first(), last(), absolute(int row)
- les colonnes peuvent être référencées par leur nom ou leur numéro (origine à 1)
- obtention de la valeur d'une colonne par get<JavaType>(index|nom)

Parcours du résultat - ResultSet 2/2

```
String query = "SELECT nom, age ...";
ResutSet r = st.executeQuery(query);
while(r.next()) {
     String nom = r.getString("nom");
    int age = r.getInt(2);
    // ...
}
```

Exemple complet

```
import java.sql.*;
public class PrintAllEmployees {
  public static void main(String[] args)
      throws SQLException, ClassNotFoundException {
    String dbUrl = "jdbc:mysql://localhost:3306/yaps";
    String user = "root", password = "";
    // Chargement dynamique du driver
    Class.forName("com.mysql.jdbc.Driver");
    // Etablissement de la connexion
    Connection conn;
    conn = DriverManager.getConnection(dbUrl, user, password);
    // Création d'un statement
    Statement st = conn.createStatement();
    String anSQLquery = "SELECT prenom, nom, age FROM employe ";
    // Exécution de la requête
    ResultSet r = st.executeQuery(anSQLquery);
    // Parcours du résultat
    while(r.next()) {
      String nom = r.getString("nom");
      int age = r.getInt("age");
      System.out.println(nom + ", " + age + " ans");
    }
    r.close(); st.close(); conn.close(); }
}
```

Correspondance de types SQL - Java

Type Java	méthode	Type SQL
boolean	getBoolean	BIT
String	getString	CHAR
double	getDouble	FLOAT
float	getFloat	REAL
int	getInteger	INTEGER
long	getLong	BIG INT
byte	getByte	TINYINT
java.math.BigDecimal	getBigDecimal	NUMERIC
byte[]	getBytes	BINARY
java.sql.Date	getDate	DATE
java.sql.Time	getTime	TIME

Invocation d'une requête SQL de mise à jour

Dans le cas d ’une modification de donnée (**INSERT, UPDATE**, **DELETE**), on utilise la méthode executeUpdate(…) qui renvoie un entier spécifiant le nombre d’enregistrements modifiés

```
String reqSql =   "DELETE FROM   ...";
int  n = st.executeUpdate(reqSql)
```

Avec JDBC, on peut aussi ...

Entre autres:

- **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**

Invocation d'une requête SQL précompilée

```
// requête SQL dynamique paramétrée par des ?
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
ResultSet r = pst.executeQuery();
```

Invocation d'une procédure stockée

```
// procédure stockée paramétrée par des ?
String aCall =   "{ ? = call getNumberOfAgeGreaterThan( ? ) }  ";
// Création d'un CallableStatement
CallableStatement cst = conn.prepareCall(aCall);
cst.setInt(2, 55); // => age > 55
// Passage des paramètres d'output
cst.registerOutParameter(1, java.sql.TYPES.INTEGER, 0);
cst.executeQuery(); // Exécution de la procédure
int nb = cst.getInt(1); // Récupération du résultat
```

Transactions

3 services déclarés dans l 'interface Connection :

- **setAutoCommit(boolean b)**
- **commit() valide une transaction**
- **rollback() annulle une transaction**

- **Les nouvelles connexions sont initialement en mode auto-commit => commit implicite après chaque requête SQL**
- **Pour définir une transaction composée de plusieurs requêtes SQL, il faut désactiver l 'auto-commit : conn.setAutoCommit(false)**
- **Un appel à commit() ou rollback() va alors créer implicitement une nouvelle transaction**

Transactions – Exemple

```
// Désactiver l'auto-commit
 conn.setAutoCommit(false);
 try {
    // les requêtes SQL suivantes constituent
    // une seule transaction
     st.executeUpdate("INSERT  ...");
     st.executeUpdate("DELETE  ...");
     st.executeUpdate("UPDATE  ...");
     // valider la transaction
     conn.commit();
     st.close();
 }
 catch(java.sql.SQLException e) { conn.rollback(); }
```

Accès au méta-modèle

Connection.getMetaData() retourne un DatabaseMetaData permettant de connaître :

- **la structure de la base :** getCatalogs(), getTables(), …
- **les types SQL supportés :** getTypeInfo()
- une description des procédures stockées : getProcedures()
- ...

ResultSet.getMetaData() retourne un ResultSetMetaData permettant de connaître :

- **le nombre de colonnes d'un ResultSet :** getColumnCount
- **le label d'une colonne :** getColumnLabel(n)
- **le type SQL d 'une colonne :** getColumnType(n)
- ...

Accès au méta-modèle - Exemple

```
List getColumnNames(ResultSet rs)
                    throws SQLException  {
   ArrayList list = new ArrayList();
   ResultSetMetaData rsmd = rs.getMetaData();
   int numcols = rsmd.getColumnCount();
   for (int i = 1 ; i <= numcols; i++)
   {
     String s = rsmd.getColumnLabel(i);
     list.add(s);
   }
   return list;
}
```

CLOB et BLOB

2 types d'objets larges :

- CLOB : Character large object
- BLOB: Binary large

Une colonne de type BLOB est en fait un pointeur vers un fichier

```
create table userImages (
        user varchar(50),
        image BLOB
)
```

Insérer un BLOB

```
String q =
 "insert into userImages values('dewez', ?)";
Statement pstmt = con.prepareStatement(q);

File file = new File("dewez.jpg");
InputStream fin = new FileInputStream(file);

pstmt.setBinaryStream (1, fin, file.length());
pstmt.executeUpdate();
```

Lire un BLOB

```
String q = "select image from userImages"
ResultSet rs = stmt.executeQuery(q);
while (rs.next) {
  Blob b = rs.getBlob("image");
  InputStream stream = b.getBinaryStream();
  // ...
}
```

Versions JDBC

JDBC	JDK	nouveaux packages
1.0	1.1	java.sql
2.0	1.2	javax.sql
3.0	1.4	javax.sql.rowset
4.0	1.6	
4.1	1.7	

cf **http://docs.oracle.com/javase/1.5.0/docs/api/java/sql/package-summary.html#package_description**

JDBC 2.0

- **Améliorations des ResultSet**
  - **SCROLLABLE : previous(), first(), last(), absolute(), …**
  - **UPDATABLE : updateInt(), …, updateRow()**
- **Type de données SQL3**

JDBC 3.0

- Connection pooling
- Transactions distribuées
- interface DataSource
- interface RowSet

JDBC 4.0

- chargement automatique des drivers JDBC
- Nouvelles exceptions :
  SQLSyntaxErrorException,
  SQLIntegrityConstantViolationException …
- Support XML
- Pas d'~~Utilisation des annotations~~ finalement!!?

Points clés

- JDBC permet d ’accéder à un SGBDR
- Une application basée sur JDBC est indépendante du SGBDR utilisé
- JDBC permet un accès SQL mais n’automatise pas la gestion de la persistance des objets java