HMIN327 : une brique de base Java SE : JDBC

I. Mougenot isabelle.mougenot@umontpellier.fr

Faculté des Sciences - Université Montpellier

Semestre 1 2020

Plan du cours

- Préambule
- Généralités
- Contexte
- Architecture fonctionnelle
- Détails de l'API JDBC
- Paquetage Core (central) : java.sql

Java Standard Edition (Java SE)

distribution de la plateforme Java pour le développement/l'exécution d'une application sur un poste de travail

- Java Specification Requests (JSR), spécifications pour les briques de la version considérée
- Java Development Kit (JDK) briques de développement (compilateur, outil pour javadoc, debugger, etc)
- Java Runtime Environment (JRE), environnement d'exécution (présent dans le JDK et comprend le moteur d'execution : JVM)

Illustration Java SE

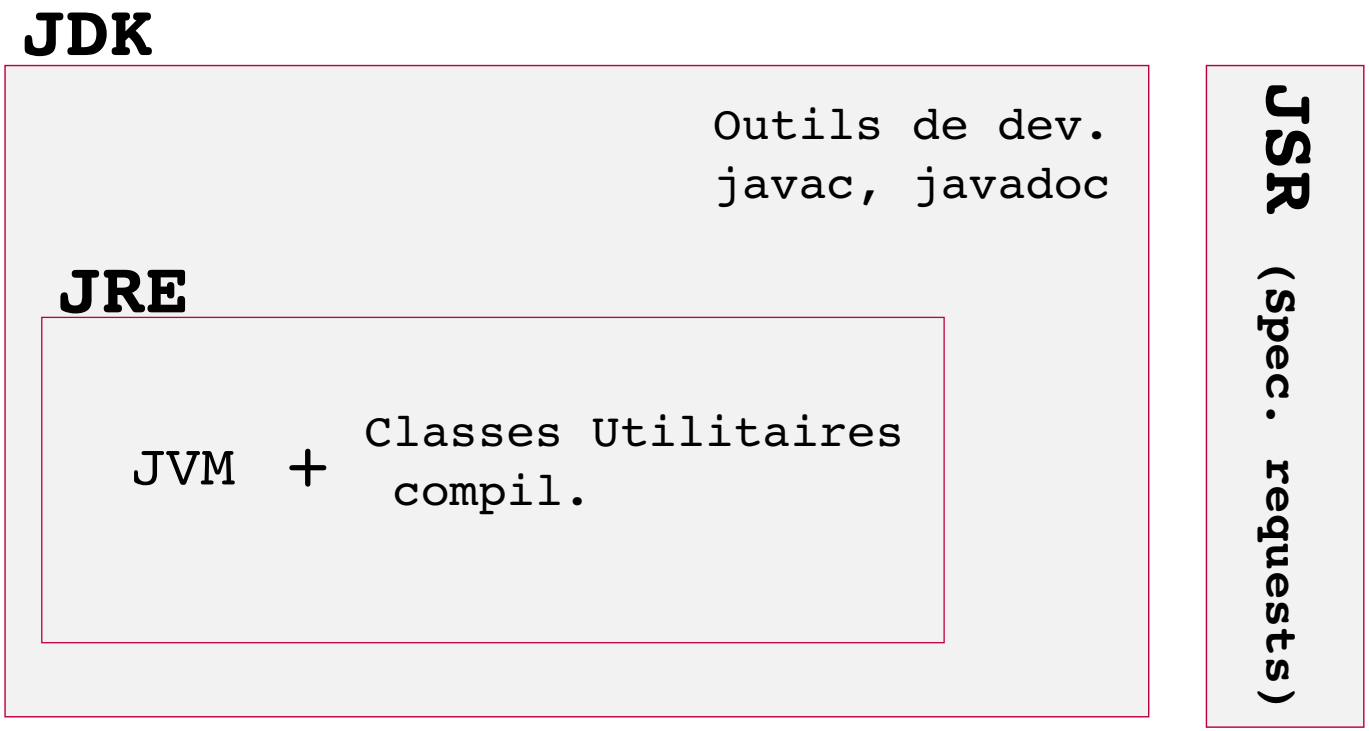

Figure: Briques de base de Java SE

Java Enterprise Edition (Java EE)

Extensions pour le développement/l'exécution d'applications distribuées dans un contexte d'environnement de production en entreprise

- Collection d'API à exploiter ensemble ou bien de manière séparée
- Ensemble de spécifications qui définissent un serveur d'application au sein duquel des composants vont s'exécuter

Architecture du serveur d'application modulaire avec un découpage logique des fonctionnalités : le développement est donc facilité

Illustration Java EE

Crédit image : http://uws-software-service.com

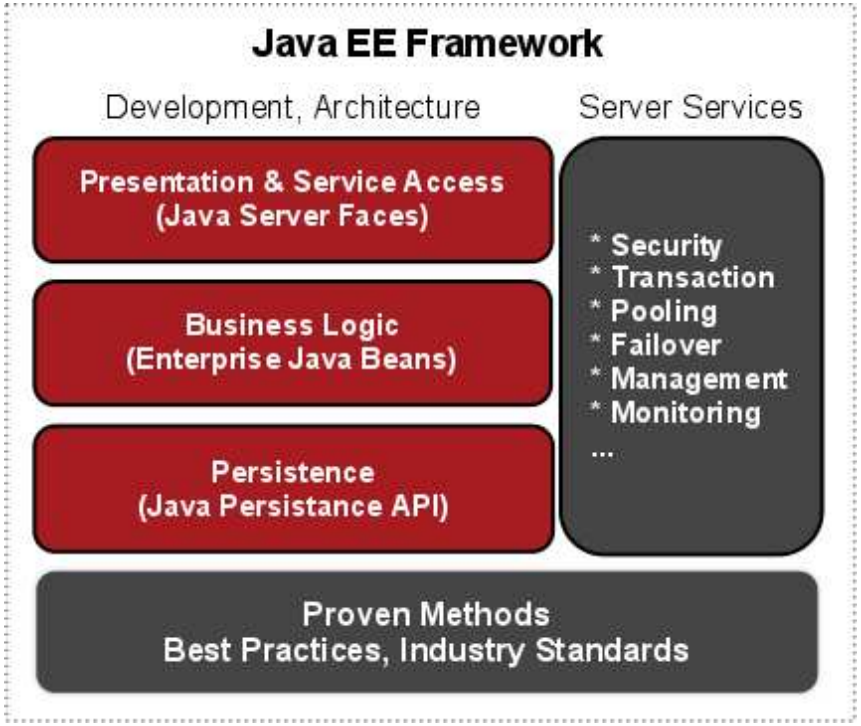

Figure: Principes Java EE

La brique JDBC

Crédit image : https://netbeans.org

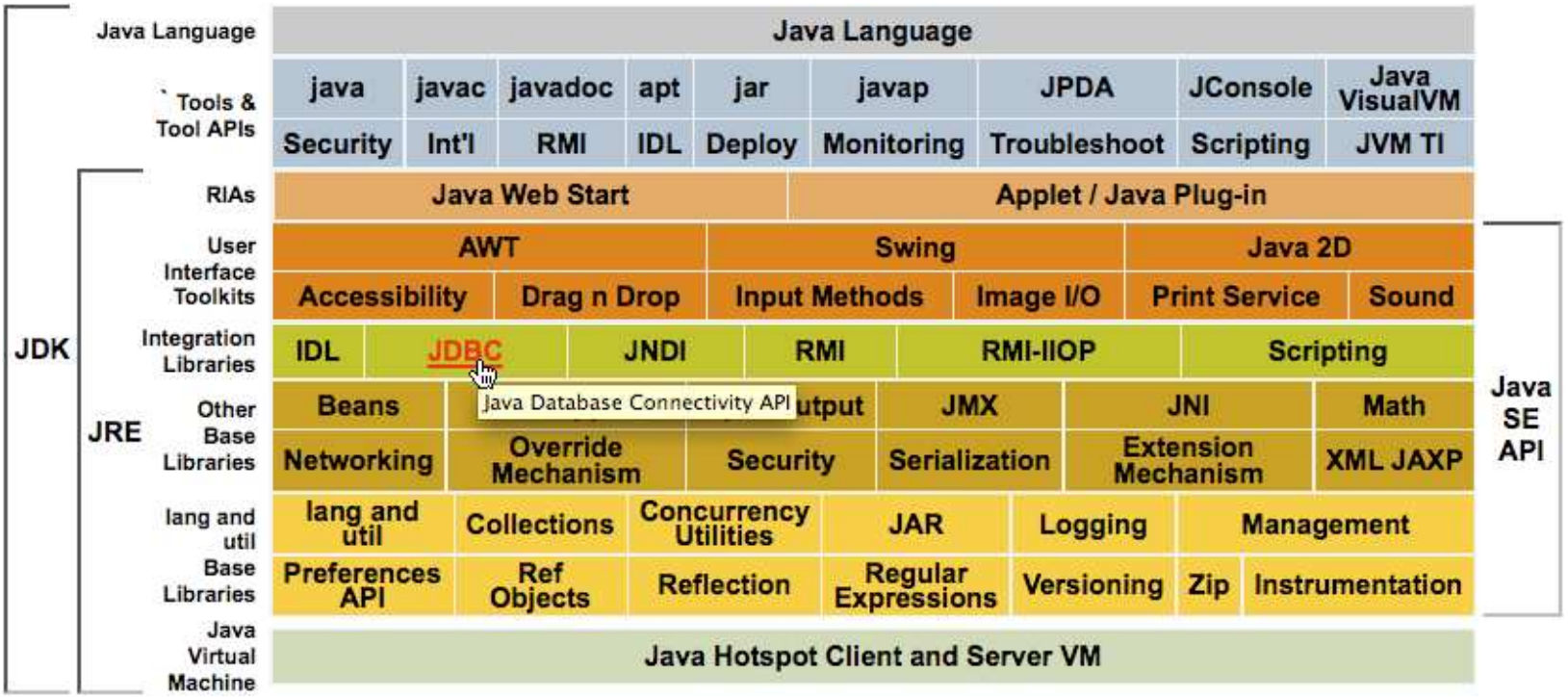

Figure: JDBC et Java SE

Positionnement général JDBC

JDBC : un socle pour différentes approches

- ORM : Hibernate (modèle objet à "mapper", IBatis (solution plus légère)
- Le standardJPA (Java Persistence API) : ex. JPA /Hibernate
- JOOQ (Java Object Oriented Querying) : solution Java centrée sur la réécriture de la requête SQL
- JDBC avec Apache DbUtils et DAO (pattern d'accès aux données)

Vision plug and play

JDBC est une API de Java SE (Java Standard Edition) et par extension de Java EE (Java Enterprise Edition)

- Plug : les fournisseurs de SGBDR intègrent leurs produits avec les environnements standards de Java
- Play : les applications Java exploitent uniformément les SGBDRs

Connecteur JDBC (Java DataBase Connectivity)

Librairie Java d'accès aux serveurs de bases de données relationnels

- Faire abstraction des SGBDRs (Oracle mais aussi Postgresql, MySql, Informix, Access, Derby, Berkeley-DB, ...) pour fournir un accès homogène aux données quel que soit le SGBD cible.
- Facilités pour les applications Java (par ex. basées sur une architecture Java EE) qui vont bénéficier de :
  1. la consultation et mise à jour des données
  2. l'évolution des schémas de données
  3. la gestion des transactions (essentiel dans un environnement concurrent)
  4. l'exploitation de données provenant de différents serveurs de BD

Quatre types de pilote

Quelques éléments distinctifs

1. Pont JDBC-ODBC : en présence d'un pilote ODBC et en absence d'un pilote JDBC
2. Client natif : une installation d'une bibliothèque propriétaire (écrite par ex. en C comme OCI pour Oracle) est nécessaire sur le poste client
3. Serveur lorsque
   - accès à un SGBDR derrière un pare-feu
   - recours à un pilote JDBC qui ne supporte pas la mutualisation de connexions
   - accès à des SGBDR différents
4. Client Java, le plus souple, ne nécessite pas la maîtrise du poste client

Vue d'ensemble des pilotes

Interactions Applications Java - Serveurs de BD (fig. extraite de http://wikivince.roiheenok.fr)

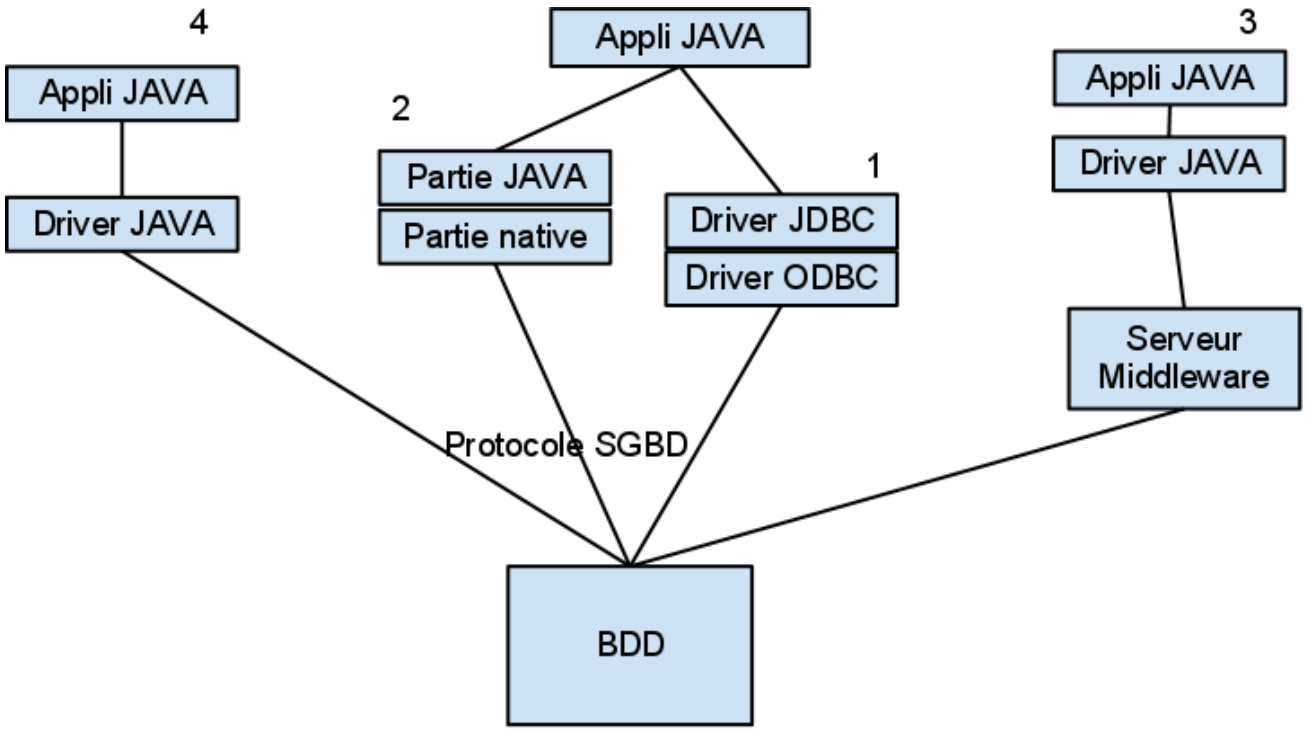

Figure: Typologie des pilotes

Rôle de JDBC

Spécifie l'architecture fonctionnelle à laquelle les pilotes doivent se conformer (fig. extraite de `http://manu.kegtux.org`)

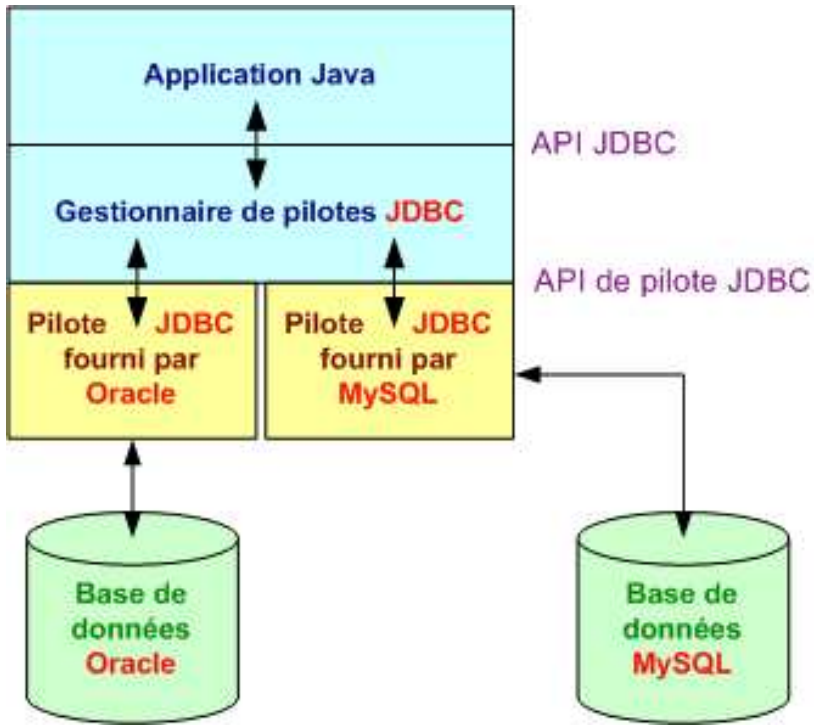

Figure: Architecture fonctionnelle

Détails de JDBC

L'API JDBC comprend un paquetage de base (core) et un paquetage d'extension

1. core
   - notion de connexion
   - notion de requête élémentaire, précompilée, appel à une procédure stockée
   - notion de résultat qui peut être parcouru et mis à jour
   - notion de transaction locale associée à la requête
2. extension
   - notion de source de données abstraite
   - notion de mutualisation de connexions (pools de connexions)
   - notion de transaction globale ou distribuée (protocole XA)

Eléments structurels de JDBC

L'API JDBC (java.sql de base) : principales classes et interfaces

- interface connexion : Connection
- interfaces requête : Statement, PreparedStatement, CallableStatement
- interface résultat : ResultSet
- interface métadonnées sur le résultat : ResultSetMetadata
- classe exception : SQLException
- classe gestionnaire de pilotes : DriverManager

Modélisation conceptuelle

Représentation du contenu du paquetage core de JDBC

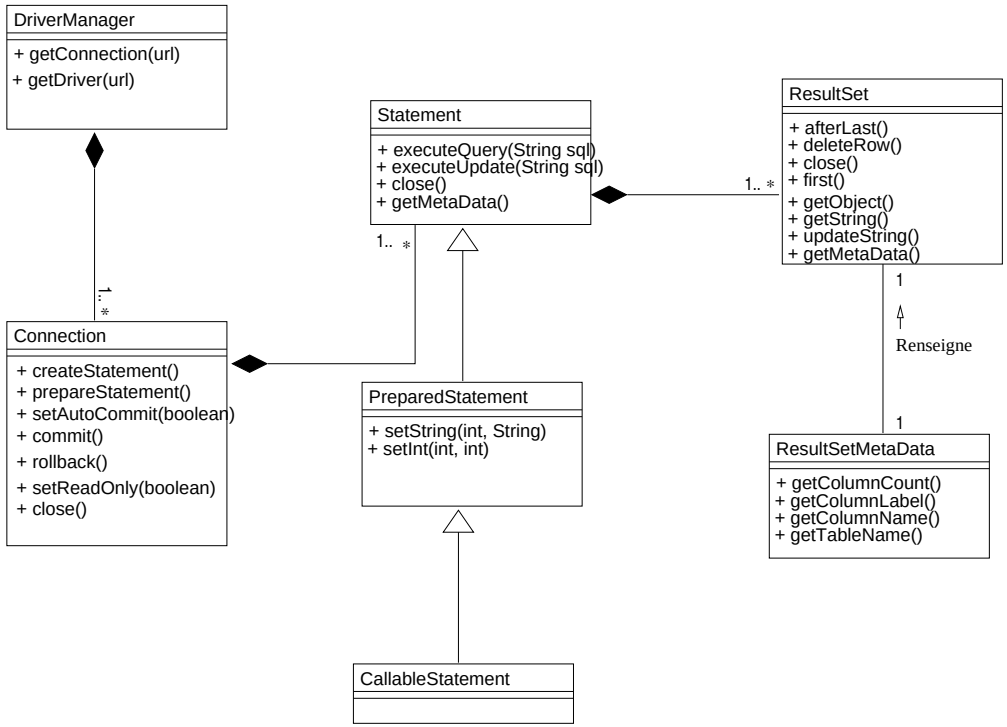

Figure: Diagramme de classes

DriverManager

Création des connexions - méthode getConnection(url, user, password) qui retourne une référence d'objet de type Connection

```
import java.sql.*;
..
 public static void main (String[] args) {
    Connection c = null;
    Statement st = null;
    ResultSet rset = null;
    try {
 c =
DriverManager.getConnection("jdbc:mysql://mysql.etu.umontpelli
     +
                        "user=p00000009432&password=XXX");
 st = c.createStatement();
...
```

Statement : plus ou moins une requête imbriquée

```
rset = st.executeQuery ("select nom from Monument ");
    ResultSetMetaData rsetSchema = rset.getMetaData();
    int nbCols = rsetSchema.getColumnCount();
    for (int i=1; i<=nbCols;i++)
    { System.out.print(rsetSchema.getColumnName(i)+ "
        | "); }
    System.out.println();
    while (rset.next ())
    {for (int i=1; i<=nbCols;i++)
    {System.out.print(rset.getObject(i)+ " | ");;  }
    System.out.println(); }
```

Listing 2: ordre SQL et traitement

Prise en charge des exceptions possibles

```
  } catch (SQLException ex) {
    // gestion des erreurs
    System.out.println("SQLException: " +
        ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " +
        ex.getErrorCode());
}
}
```

Listing 3: Etat sur la connexion

Transaction

Une connexion gère aussi les transactions - Par défaut, le mode activé est le mode commit automatique

```
void setAutoCommit(boolean autoCommit)
void setReadOnly(boolean readOnly)
void setTransactionIsolation(int level)
void commit()
void rollback()
void close()
```

Listing 4: Prise en charge transactions

Statement

Trois méthodes : executeQuery() pou les consultations (SELECT), executeUpdate() pour les mises à jour ou les ordres de LDD et execute()

```
stmt =
    c.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
      ResultSet.CONCUR_UPDATABLE);
rset = stmt.executeQuery ("select nom ,"
+ " codeinsse from monument");
ResultSetMetaData rsetSchema = rset.getMetaData();
...
```

Listing 5: Consultation

Statement

Exemple d'appel executeUpdate()

```
c.setAutoCommit(false);
stmt = c.createStatement();
String sql = "update emp set nom='Martin' where num=4";
int result = stmt.executeUpdate(sql);
if (result == 1) {
System.out.println("Table Updated
    Successfully.......");}
```

Listing 6: Mise à jour

PreparedStatement, Instruction paramétrée

Avec ordre SQL précompilé et paramètres d'entrée valués avant l'exécution avec méthodes setXXX via points d'interrogation - méthodes execute dépourvues d'arguments

```
PreparedStatement pstmt = null;
System.out.println("Entrez un num de departement");
saisie = new Scanner(System.in); int dep =
    saisie.nextInt();
String sql = "update D set lieu=?, budg=? where
    n_Dep=?";
pstmt=c.prepareStatement(sql,
ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
pstmt.setInt(3,dep);
int etat = pstmt.executeUpdate();
System.out.println("nbre de tuples modifies "+etat);
```

ResultSet

ResultSet gère l'accès aux tuples d'un résultat en offrant des services de navigation et d'extraction des données dans ce résultat - L'accès aux colonnes se fait soit par le nom de colonne, soit par l'index au travers notamment des méthodes getString() et getObject()

```
rset = stmt.executeQuery ("select ... from cols where
    table_name='EMP'");
ResultSetMetaData rsetSchema = rset.getMetaData();
int nbCols = rsetSchema.getColumnCount();
for (int i=1; i<=nbCols;i++)
{ System.out.print(rsetSchema.getColumnName(i)+ " | ");}
while (rset.next ())
{for (int i=1; i<=nbCols;i++)
{System.out.print(rset.getObject(i)+ " | ");;}
}
```

Listing 8: Manipulation des résultats

Test de niveaux d'isolation

Notion de métadonnées sur le schéma

```
 Connection c = getConnection();
 DatabaseMetaData dbMd = c.getMetaData();
   if (
dbMd.supportsTransactionIsolationLevel(
   Connection.TRANSACTION_READ_COMMITTED))
{
  System.out.println("Transaction Isolation level "
   + "TRANSACTION_READ_COMMITTED is supported.");
 c.setTransactionIsolation(
   Connection.TRANSACTION_READ_COMMITTED);
 System.out.println("Number for transaction
     read-committed "
     + c.getTransactionIsolation());
   }
...
```

Modèle architectural

Modèle à deux couches

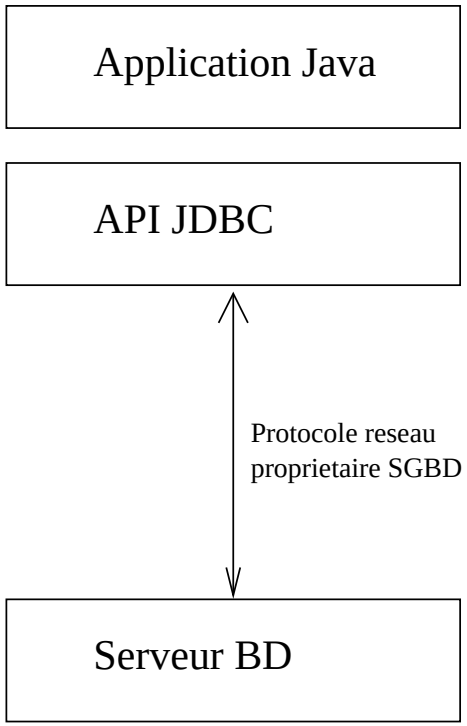

Figure: Architecture client-serveur

Modèle architectural

Modèle à trois couches

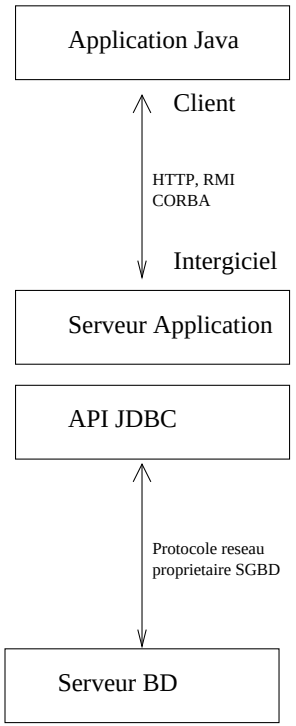

Figure: Couche intergicielle

Modèle architectural

fig. extraite du site CommentCaMarche

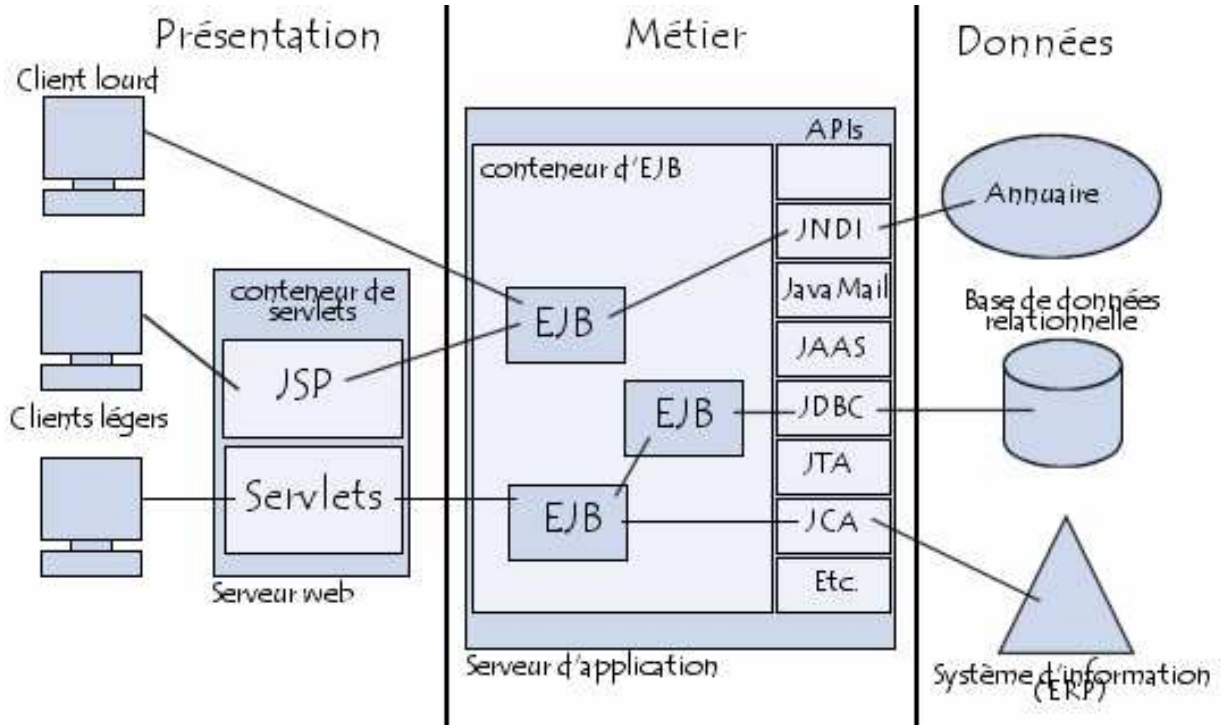

Figure: Vision J2EE

Extensions JDBC

Les intérêts de l'extension de JDBC et des packages

1. *javax.sql* de manière générale
2. *oracle.jdbc.pool* dans le contexte d'Oracle

reposent sur différentes notions qui facilitent l'exploitation à distance et le partage de sources de données

Les notions sous-jacents aux extensions

1. Le concept de **DataSource** qui permet de désigner une base de données ou encore tout autre type de source de données et va également en simplifier l'accès et l'exploitation.
2. **Pool et cache de connexions (connection pooling et caching)** : les créations/fermetures de connexions physiques sur un SGBD, à travers un réseau, sont des opérations lourdes et donc longues. JDBC propose des extensions *Connection Pooling* et *Connection Caching* qui distinguent connexion logique et physique.
3. **Transactions distribuées** Cette notion va être à la base de mécanismes d'intégration entre sources de données. Il sera en effet possible de consulter/mettre à jour plusieurs bases de données en même temps.

Illustration Datasource (sur Oracle)

Flexiblité sur l'accès à une source de données

```
import java.sql.*;
import oracle.jdbc.pool.*;
public class DataSourceTest {
   public static void main(String[] args)
   throws SQLException,IOException {
  OracleDataSource ods = new OracleDataSource();
  ods.setDriverType("thin");
  ods.setServerName("venus");
  ods.setDatabaseName("master");
  ods.setNetworkProtocol("tcp");
      ods.setPortNumber(1521);
  ods.setUser("mast1"); ods.setPassword("mast1");
  Connection c = ods.getConnection();
  Connection c1 = ods.getConnection("user1","user1");
```

Illustration Datasource

Flexiblité sur l'accès à une source de données

```
Statement stmt = c.createStatement();
ResultSet rset = stmt.executeQuery ("select num, nom
    , fonction from emp");
 while (rset.next ())
 System.out.println (rset.getString (1) + " " +
    rset.getString (2)
 + " " + rset.getString(3));
 Statement stmt1 = c1.createStatement();
 ResultSet rset1 = stmt1.executeQuery ("select num,
    nom , fonction from emp");
  while (rset1.next ())
  System.out.println (rset1.getString (1) + " " +
     rset1.getString (2)
  + " " + rset1.getString(3));
c.close(); ...
```