

# Architectures Logicielles

**Olivier Caron**

© Polytech'Lille

Architectures 2-tiers

- Les applications logicielles deviennent de plus en plus complexes :
  - L'application ne fonctionne pas sur une machine mais sur plusieurs.  
→ programmation réseau, efficacité, performances, sécurité, tolérance aux pannes, . . .
  - Intéropérabilité des langages et plates-formes.
  - L'application utilise/ou réutilise des applications existantes.  
→ applications patrimoines, bases de données

- Les applications logicielles deviennent de plus en plus complexes :
  - L'application ne fonctionne pas sur une machine mais sur plusieurs.  
→ programmation réseau, efficacité, performances, sécurité, tolérance aux pannes,...
  - Intéropérabilité des langages et plates-formes.
  - L'application utilise/ou réutilise des applications existantes.  
→ applications patrimoines, bases de données

- Les applications logicielles deviennent de plus en plus complexes :
  - L'application ne fonctionne pas sur une machine mais sur plusieurs.  
→ programmation réseau, efficacité, performances, sécurité, tolérance aux pannes,...
  - Intéropérabilité des langages et plates-formes.
  - L'application utilise/ou réutilise des applications existantes.  
→ applications patrimoines, bases de données

- Les applications logicielles deviennent de plus en plus complexes :
  - L'application ne fonctionne pas sur une machine mais sur plusieurs.  
→ programmation réseau, efficacité, performances, sécurité, tolérance aux pannes,...
  - Intéropérabilité des langages et plates-formes.
  - L'application utilise/ou réutilise des applications existantes.  
→ applications patrimoines, bases de données

- Face à cette complexité, on veut, en outre :
  - Améliorer la qualité et la production de code
  - Simplifier le travail du programmeur
  - Réutiliser au mieux le logiciel

- Face à cette complexité, on veut, en outre :
  - Améliorer la qualité et la production de code
  - Simplifier le travail du programmeur
  - Réutiliser au mieux le logiciel

- Face à cette complexité, on veut, en outre :
  - Améliorer la qualité et la production de code
  - Simplifier le travail du programmeur
  - Réutiliser au mieux le logiciel



- Face à cette complexité, on veut, en outre :
  - Améliorer la qualité et la production de code
  - Simplifier le travail du programmeur
  - Réutiliser au mieux le logiciel

- Définir en premier lieu l'architecture logicielle la mieux adaptée :  
*A software architecture is an abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces and component-component interconnections". (Hayes-Roth, 1994)*
- Disposer de briques logicielles réutilisables : les composants.
- Disposer d'infrastructures d'accueil de ces composants : les serveurs d'applications.

- Définir en premier lieu l'architecture logicielle la mieux adaptée :  
*A software architecture is an abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces and component-component interconnections". (Hayes-Roth, 1994)*
- Disposer de briques logicielles réutilisables : les composants.
- Disposer d'infrastructures d'accueil de ces composants : les serveurs d'applications.

- Définir en premier lieu l'architecture logicielle la mieux adaptée :  
*A software architecture is an abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces and component-component interconnections". (Hayes-Roth, 1994)*
- Disposer de briques logicielles réutilisables : les composants.
- Disposer d'infrastructures d'accueil de ces composants : les serveurs d'applications.

- **Les architectures 2-tiers**

- Code et Bases de données (applications JDBC)

- Des objets aux composants

- Le modèle de composants Java EE

- Des objets aux objets répartis

- Le modèle Java RMI

- Le modèle CORBA

- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
  - Des objets aux composants
    - Le modèle des Composants EJB
  - Des objets aux objets répartis
    - Le modèle Java RMI
    - Le modèle CORBA

- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
  - Des objets aux composants
    - Le modèle des Composants Java EE
  - Des objets aux objets répartis
    - Le modèle Java RMI
    - Le modèle CORBA

- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
- Des objets aux composants
  - Le modèle de composants Java Beans
- Des objets aux objets répartis
  - Le modèle Java RMI
  - Le modèle CORBA



- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
- Des objets aux composants
  - Le modèle de composants Java Beans
- Des objets aux objets répartis

- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
- Des objets aux composants
  - Le modèle de composants Java Beans
- Des objets aux objets répartis

- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
- Des objets aux composants
  - Le modèle de composants Java Beans
- Des objets aux objets répartis
  - Le modèle Java RMI
  - Le modèle CORBA

- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
- Des objets aux composants
  - Le modèle de composants Java Beans
- Des objets aux objets répartis
  - Le modèle Java RMI
  - Le modèle CORBA

- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
- Des objets aux composants
  - Le modèle de composants Java Beans
- Des objets aux objets répartis
  - Le modèle Java RMI
  - Le modèle CORBA

- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
- Des objets aux composants
  - Le modèle de composants Java Beans
- Des objets aux objets répartis
  - Le modèle Java RMI
  - Le modèle CORBA

- Les architectures 2-tiers
  - Code et Bases de données (applications JDBC)
- Des objets aux composants
  - Le modèle de composants Java Beans
- Des objets aux objets répartis
  - Le modèle Java RMI
  - Le modèle CORBA

- Des objets aux composants répartis
  - Les architectures clients/serveurs 3-tiers
    - L'architecture J2EE (composants Web, composants EJB)
    - Introduction aux web services
    - *Le modèle de composants CORBA*
- Conception d'applications 3-tiers



- Des objets aux composants répartis
  - Les architectures clients/serveurs 3-tiers
    - L'architecture J2EE (composants Web, composants EJB)
    - Introduction aux web services
    - *Le modèle de composants CORBA*
  - Conception d'applications 3-tiers

- Des objets aux composants répartis
  - Les architectures clients/serveurs 3-tiers
    - L'architecture J2EE (composants Web, composants EJB)
      - Introduction aux web services
      - *Le modèle de composants CORBA*
  - Conception d'applications 3-tiers

- Des objets aux composants répartis
  - Les architectures clients/serveurs 3-tiers
    - L'architecture J2EE (composants Web, composants EJB)
    - Introduction aux web services
      - *Le modèle de composants CORBA*
  - Conception d'applications 3-tiers

- Des objets aux composants répartis
  - Les architectures clients/serveurs 3-tiers
    - L'architecture J2EE (composants Web, composants EJB)
    - Introduction aux web services
    - *Le modèle de composants CORBA*
  - Conception d'applications 3-tiers

- Des objets aux composants répartis
  - Les architectures clients/serveurs 3-tiers
    - L'architecture J2EE (composants Web, composants EJB)
    - Introduction aux web services
    - *Le modèle de composants CORBA*
- Conception d'applications 3-tiers

- Découpage d'une application en deux entités.
- Niveau données :  
fichiers, SGBD
- Niveau code métier logiciel :
- Gère l'IHM et le code métier
- Dispose d'une API pour gérer les données
- Code plus ou moins complexe :  
en réseau ?, sécurité ?, accès concurrents ?, ...

- Découpage d'une application en deux entités.
- Niveau données :  
fichiers, SGBD
- Niveau code métier logiciel :
- Gère l'IHM et le code métier
- Dispose d'une API pour gérer les données
- Code plus ou moins complexe :  
en réseau ?, sécurité ?, accès concurrents ?, ...

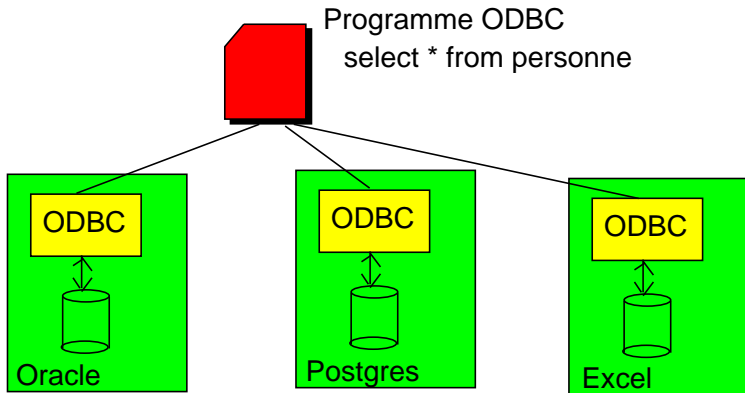
- Découpage d'une application en deux entités.
- Niveau données :  
fichiers, SGBD
- Niveau code métier logiciel :
  - Gère l'IHM et le code métier
  - Dispose d'une API pour gérer les données
  - Code plus ou moins complexe :  
en réseau ?, sécurité ?, accès concurrents ?, ...



- Découpage d'une application en deux entités.
- Niveau données :  
fichiers, SGBD
- Niveau code métier logiciel :
- Gère l'IHM et le code métier
- Dispose d'une API pour gérer les données
- Code plus ou moins complexe :  
en réseau ?, sécurité ?, accès concurrents ?, ...

- Découpage d'une application en deux entités.
- Niveau données :  
fichiers, SGBD
- Niveau code métier logiciel :
- Gère l'IHM et le code métier
- Dispose d'une API pour gérer les données
- Code plus ou moins complexe :  
en réseau ?, sécurité ?, accès concurrents ?, ...

- Découpage d'une application en deux entités.
- Niveau données :  
fichiers, SGBD
- Niveau code métier logiciel :
- Gère l'IHM et le code métier
- Dispose d'une API pour gérer les données
- Code plus ou moins complexe :  
en réseau ?, sécurité ?, accès concurrents ?, ...



- Langage Java (multi-plateforme :-)
- API
- Adopté par presque tous les constructeurs
- Passerelle ODBC-JDBC

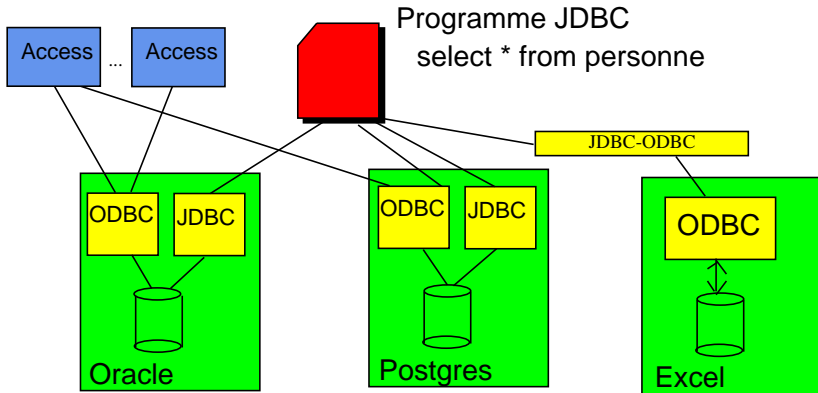
- Langage Java (multi-plateforme :-)
- API
- Adopté par presque tous les constructeurs
- Passerelle ODBC-JDBC

- Langage Java (multi-plateforme :-)
- API
- Adopté par presque tous les constructeurs
- Passerelle ODBC-JDBC

- Langage Java (multi-plateforme :-)
- API
- Adopté par presque tous les constructeurs
- Passerelle ODBC-JDBC



# Des utilisations possibles des protocoles



# Chargement du driver

- Disposer d'un driver propriétaire  
(Réseau Polytech : /usr/share/java/postgresql.jar)
- Charger le driver :

```
try {  
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver") ;  
} catch (ClassNotFoundException e) {  
    System.err.println("Exception:_" + e.toString());  
}
```

- Système basé sur le mécanisme d'interfaces Java
- Driver postgres : `org.postgresql.Driver`

- Utilisation d'une méthode static de `DriverManager`

```
Connection connect =  
    DriverManager.getConnection(urlBase ,  
        "nomLogin" , "nomPassword" );
```

- Il faut définir dans l'url :
  - le protocole et sous-protocole
  - l'adresse du serveur de base de données et le nom de la base

```
String urlBase = "jdbc:odbc:// nomServeur/nomBase" ;  
String urlBase = "jdbc:postgresql://weppes/biblio" ;
```

- Connexion réseau conforme internet

- Objectif : avoir un programme 100% réutilisable
- Définition d'un fichier de propriétés,

fichier `database.properties` :

```
jdbc.driver=org.postgresql.Driver
```

```
jdbc.url=jdbc:postgresql://weppes.studserv/base
```

```
jdbc.username=admin
```

```
jdbc.password=secret
```

## Chargement driver, version 2 (2/2)

```
Properties props= new Properties() ;  
FileInputStream in=new FileInputStream(fileName) ;  
props.load(in) ;  
String driver=props.getProperty("jdbc.driver") ;  
String url=props.getProperty("jdbc.url") ;  
String username=props.getProperty("jdbc.username") ;  
String password=props.getProperty("jdbc.password") ;  
  
Class.forName(driver) ;
```

- permet de définir les requêtes SQL à envoyer à la base connectée

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

- Permet d'exécuter deux types de requêtes :

- Les requêtes de modification de la base
- Les requêtes de consultation de la base

- **Toutes** les méthodes doivent prendre en compte l'exception `SQLException`

- **API:** `int executeUpdate(String requeteSQL)`  

```
stmt.executeUpdate("create_table_tester_"+  
                  "(num_integer ,ch_text)");  
stmt.executeUpdate("insert_into_tester_values_(1,'dupont')");  
stmt.executeUpdate("insert_into_tester_values_(2,'durant')");
```
- Valable pour toutes les commandes SQL DDL (Data Definition Language)
- Valable pour les requêtes du type `select * into ...`
- retourne le nombre de lignes créées, modifiées,...

- **API**: `ResultSet executeQuery(String requeteSQL)`  
`ResultSet rs = stmt.executeQuery(`  
`"select_*_from_tester" ) ;`



La classe `ResultSet` dispose des méthodes suivantes :

<code>boolean next()</code>	retourne true s'il reste un n-uplet à lire
<code>Date getDate(int i)</code>	retourne l'objet Date de la colonne i
<code>Date getDate(String col)</code>	retourne l'objet Date de la colonne col
<code>int getInt(int i)</code>	retourne un entier de la colonne i
<code>int getInt(String col)</code>	retourne un entier de la colonne col
<code>String getString(int i)</code>	retourne l'objet String de la colonne i
<code>String getString(String col)</code>	retourne l'objet String de la colonne col
...	...

- Un exemple :

```
int i = 0;
while (rs.next()) {
    int num = rs.getInt("num");
    String ch = rs.getString("ch");
    System.out.println("ligne_" + i + " :_" +
                       num + ",_" + ch);
    i++;
}
```

- `next()` est obligatoire même pour lire une seule ligne (ex : `count`)
- JDBC 1.0 oblige à parcourir la table résultat

- Fermer le dialogue SQL
- Fermer la session avec la base

```
stmt.close() ;  
connect.close() ;
```

- Lorsqu'elles sont exécutées plusieurs fois
- Les SGBD ont une version compilée de la requête (performances)
  - dépend des SGBD
- JDBC offre une API pour :
  - Préparer une requête
  - Paramétrer une requête

- Lorsqu'elles sont exécutées plusieurs fois
- Les SGBD ont une version compilée de la requête (performances)
  - dépend des SGBD
- JDBC offre une API pour :
  - Préparer une requête
  - Paramétrer une requête

- Lorsqu'elles sont exécutées plusieurs fois
- Les SGBD ont une version compilée de la requête (performances)
  - dépend des SGBD
- JDBC offre une API pour :
  - Préparer une requête
  - Paramétrer une requête

- Lorsqu'elles sont exécutées plusieurs fois
- Les SGBD ont une version compilée de la requête (performances)
  - dépend des SGBD
- JDBC offre une API pour :
  - Préparer une requête
  - Paramétrer une requête

- Lorsqu'elles sont exécutées plusieurs fois
- Les SGBD ont une version compilée de la requête (performances)
  - dépend des SGBD
- JDBC offre une API pour :
  - Préparer une requête
  - Paramétrer une requête



- Lorsqu'elles sont exécutées plusieurs fois
- Les SGBD ont une version compilée de la requête (performances)
  - dépend des SGBD
- JDBC offre une API pour :
  - Préparer une requête
  - Paramétrer une requête

- Création d'une requête préparée :

```
PreparedStatement cmdSQL =  
    connect.prepareStatement("select _*_ "+  
        "from _personne _where _nom=? _and _age _> _?")
```

- Paramétrer la requête :

```
cmdSQL.setString(1, "dupont") ; cmdSQL.setInt(2,17) ;
```

- Exécution de la requête :

```
ResultSet rs = cmdSQL.executeQuery() ;
```

- La concaténation Java marche aussi !

- Mode par défaut :auto-commit  
Chaque requête SQL forme une transaction
- Modifier le mode par défaut :

```
connect.setAutoCommit(false) ;
```

- Valider une transaction :

```
stmt.executeUpdate("delete _from _emprunter _where _numl=2") ;  
stmt.executeUpdate("delete _from _reserver _where _numl=2") ;  
stmt.executeUpdate("delete _from _livres _where _numl=2") ;  
connect.commit() ;  
connect.setAutoCommit(true) ;
```

- méthode `rollback()`

```
try {
    connect.setAutoCommit(false) ; // début transaction
    cmdeSQL="insert_into_emprunter_values_( "+numl+" ,_" +numu+" )" ;
    stmt.executeUpdate(cmdeSQL) ;
    if (nbEmprunts >= 3) connect.rollback() ;
    else connect.commit() ; //fin transaction
    connect.setAutoCommit(true) ;
} catch (SQLException e) {
    if (connect != null) {
        try { connect.rollback() ; connect.setAutoCommit(true) ;
        } catch (SQLException ex) {
            System.err.println("on_est_mal_!" ) ; }
    }
}
```

- JDBC propose une nouvelle interface `CallableStatement`  

```
CallableStatement cs =  
    connect.prepareCall (" { call_now } " ) ;
```
- Invocation de la procédure :  

```
ResultSet rs = cs.executeQuery () ;
```
- Les paramètres sont gérés comme une requête préparée
- Utile pour être indépendant du SGBD (portabilité)

- Utilisation classique (expr. select)
- Non portable

```
stmt = connect.createStatement() ;  
rs=s.executeQuery("select_now()") ;  
rs.next() ;  
System.out.println("La_date_du_jour:_"+  
                    rs.getDate("now")) ;
```

- disposer d'informations provenant d'erreurs non graves d'exécution  
(déconnexion, une instruction grant ne s'est pas réalisée,...)

```
SQLWarning warning = stmt.getWarnings();
if (warning != null) {
    System.out.println("\n---Warning---\n");
    while (warning != null) {
        System.out.println("Message:_" + warning.getMessage());
        System.out.println("SQLState:_" + warning.getSQLState());
        System.out.print("Vendor_error_code:_");
        System.out.println(warning.getErrorCode());
        warning = warning.getNextWarning();
    }
}
```

- A éviter lorsque la sécurité est importante
- Accès réseau limité
- Nécessite de charger le driver



# Applet et JDBC : un exemple (1/3)

```
<html>
<head>
  <title>test applet et JDBC</title>
</head>
<body>
  <h1>Les utilisateurs de la base</h1>
  <APPLET code = "AppletJDBC.class"
    archive = "postgresql.jar"
    codebase="http://weppes.studserv.deule.net/~ocaron/"
    width=200 height=50>
  </APPLET>
</body>
</html>
```

## Applet et JDBC : un exemple (2/3) I

```
import java.sql.* ;
import java.applet.Applet ;
import java.awt.Label ;
public class AppletJDBC extends Applet {
    public void init() {
        Label label=new Label() ;
        Statement stmt = null ; Connection db = null ;
        ResultSet rs =null ;
        try {
            Class.forName("org.postgresql.Driver") ;
            db=DriverManager.getConnection(
                "jdbc:postgresql://weppes.studserv.deule.net/template1"
                ,"nomLogin" ,"motdepasse") ;
            stmt=db.createStatement() ;
            rs=stmt.executeQuery("select_count(*)_from_pg_user") ;
            rs.next() ;
            label.setText("nombre_d'utilisateurs :_"+
```

## Applet et JDBC : un exemple (2/3) II

```
                rs.getInt("count")) ;  
    stmt.close() ; db.close() ;  
} catch (ClassNotFoundException e1) {  
    label.setText(e1.getMessage()) ;  
} catch (SQLException e2) {  
    label.setText(e2.getMessage()) ;  
    try {  
        if (stmt != null) stmt.close() ;  
        if (db != null) db.close() ;  
    } catch (SQLException e3) {}  
}  
this.add(label) ;  
}  
}
```

# Applet et JDBC : un exemple (3/3)



- Déplacement amélioré dans une table
- Mise à jour de la base en utilisant des méthodes Java au lieu de commandes SQL
- Envoi de plusieurs commandes SQL (batch)
- Quelques adaptations à SQL 3
- Package java : `javax.sql`

- Déplacement amélioré dans une table
- Mise à jour de la base en utilisant des méthodes Java au lieu de commandes SQL
- Envoi de plusieurs commandes SQL (batch)
- Quelques adaptations à SQL 3
- Package java : `javax.sql`

- Déplacement amélioré dans une table
- Mise à jour de la base en utilisant des méthodes Java au lieu de commandes SQL
- Envoi de plusieurs commandes SQL (batch)
- Quelques adaptations à SQL 3
- Package java : `javax.sql`

- Déplacement amélioré dans une table
- Mise à jour de la base en utilisant des méthodes Java au lieu de commandes SQL
- Envoi de plusieurs commandes SQL (batch)
- Quelques adaptations à SQL 3
- Package java : `javax.sql`



- Déplacement amélioré dans une table
- Mise à jour de la base en utilisant des méthodes Java au lieu de commandes SQL
- Envoi de plusieurs commandes SQL (batch)
- Quelques adaptations à SQL 3
- Package java : `javax.sql`

- Nouvelle définition pour obtenir une table résultat :

```
Statement stmt ; ResultSet srs ;  
stmt = db.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
                          ResultSet.CONCUR_UPDATABLE);  
srs = stmt.executeQuery("SELECT_*_from_joueur");
```

- Des constantes définies au niveau de `ResultSet` :

TYPE_FORWARD_ONLY	parcours dans un seul sens
TYPE_SCROLL_SENSITIVE	sensible aux changements
TYPE_SCROLL_INSENSITIVE	non sensible aux changements
CONCUR_READ_ONLY	non modifiable
CONCUR_UPDATABLE	modifiable

# Les possibilités de déplacement

<code>srs.next()</code>	ligne suivante
<code>srs.previous()</code>	ligne précédente
<code>srs.absolute(i)</code>	aller à la $i^{\text{ème}}$ ligne
<code>srs.absolute(-i)</code>	aller à la $i^{\text{ème}}$ ligne en partant de la dernière
<code>srs.relative(i)</code>	descendre de $i$ lignes
<code>srs.relative(-i)</code>	remonter de $i$ lignes
<code>srs.afterlast()</code>	fin de la table
<code>srs.isAfterLast()</code>	retourne vrai si fin de table
<code>srs.last()</code>	aller à la dernière ligne

- Des nouvelles méthodes pour `ResultSet` :

```
void updateType(String nomColonne, type  
nouvelleValeur)  
void updateRow()  
void cancelRowUpdates()
```

- Un exemple :

```
srs.updateString("nom", "Benzema") ;  
srs.updateRow() ; // changement validé  
srs.updateString("nom", "Caron") ;  
srs.cancelRowUpdates(); // ici, seule la ligne  
// précédente est invalidée
```

- Des nouvelles méthodes pour `ResultSet` :

```
void moveToInsertRow()  
void moveToCurrentRow()  
void insertRow()
```

- Un exemple :

```
// phase d'insertion  
srs.moveToInsertRow();  
srs.updateInt("num", 5);  
srs.updateString("nom", "Mexes");  
srs.insertRow();  
srs.moveToCurrentRow(); // retour au curseur  
                        // avant phase d'insertion
```

- Des nouvelles méthodes pour `ResultSet` :

```
void deleteRow()
```

```
void refreshRow() (mode TYPE_SCROLL_SENSITIVE)
```

- Permet de diminuer les échanges entre client/serveur (performances)
- Un exemple :

```
db.setAutoCommit(false);  
try {  
    Statement stmt = db.createStatement();  
    stmt.addBatch("insert_into_joueur_values(6,'Anelka')")  
    stmt.addBatch("insert_into_joueur_values(7,'Ribery')")  
    stmt.addBatch("insert_into_joueur_values(8,'Viera')") ;  
    int [] updateCounts = stmt.executeBatch();  
} catch (BatchUpdateException e) {...
```

- SQL 3 : unification de concepts orienté-objet et le monde relationnel
- Des colonnes de type non simple :
  - Des tableaux (`rs.getArray("nomColonne")`)
  - Des objets (`rs.getBlob("nomColonne")`)



- SQL 3 : unification de concepts orienté-objet et le monde relationnel
- Des colonnes de type non simple :
  - Des tableaux (`rs.getArray("nomColonne")`)
  - Des objets (`rs.getBlob("nomColonne")`)

- SQL 3 : unification de concepts orienté-objet et le monde relationnel
- Des colonnes de type non simple :
  - Des tableaux (`rs.getArray("nomColonne")`)
  - Des objets (`rs.getBlob("nomColonne")`)

- SQL 3 : unification de concepts orienté-objet et le monde relationnel
- Des colonnes de type non simple :
  - Des tableaux (`rs .getArray ("nomColonne")`)
  - Des objets (`rs .getBlob ("nomColonne")`)

Une meta-donnée est une donnée qui décrit une donnée

- Des Exemples :
  - Modèle Relationnel : table(colonne1 type1, ...)
  - Postgres pg\_database, pg\_user, pg\_class, ...
  - Java : getClass() , getMethods, getFields(), ...
- JDBC propose une API pour analyser une base (introspection)

Une meta-donnée est une donnée qui décrit une donnée

- Des Exemples :
  - Modèle Relationnel : `table(colonne1 type1, ...)`
  - Postgres `pg_database, pg_user, pg_class, ...`
  - Java : `getClass()` , `getMethods`, `getFields()`, ...
- JDBC propose une API pour analyser une base (introspection)

Une meta-donnée est une donnée qui décrit une donnée

- Des Exemples :
  - Modèle Relationnel : table(colonne1 type1, ...)
  - Postgres pg\_database, pg\_user, pg\_class, ...
    - Java : getClass() , getMethods, getFields(), ...
- JDBC propose une API pour analyser une base (introspection)

Une meta-donnée est une donnée qui décrit une donnée

- Des Exemples :
  - Modèle Relationnel : table(colonne1 type1, ...)
  - Postgres pg\_database, pg\_user, pg\_class, ...
  - Java : getClass() , getMethods, getFields(), ...
- JDBC propose une API pour analyser une base (introspection)

Une meta-donnée est une donnée qui décrit une donnée

- Des Exemples :
  - Modèle Relationnel : table(colonne1 type1, ...)
  - Postgres pg\_database, pg\_user, pg\_class, ...
  - Java : getClass() , getMethods, getFields(), ...
- JDBC propose une API pour analyser une base (introspection)



- Analyser dynamiquement la structure d'une table résultat  
l'interface `ResultSetMetaData`

<code>int getColumnCount()</code> <code>int getColumnDisplaySize(int column)</code> <code>String getColumnLabel(int column)</code> <code>String getColumnName(int column)</code> <code>int getColumnTypes(int column)</code> <code>String getColumnTypeName(int column)</code>	le nombre de colonnes taille d'affichage d'une col nom suggéré d'une colonne nom de colonne type (cste) de la colonne nom du type de la colonne
---	--

# Un exemple d'analyse dynamique de table I

```
rs = stmt.executeQuery("select _count(*) _from _joueur");
rs.next() ; int nbLignes=rs.getInt("count") ;
rs = stmt.executeQuery("select _*_ _from _joueur");
ResultSetMetaData rsmd = rs.getMetaData() ;
// stockage des noms de colonne
String colName[] = new String[rsmd.getColumnCount()] ;
for (int i=0;i<rsmd.getColumnCount();i++) {
    colName[i]= rsmd.getColumnLabel(i+1);
}
Object resultat[][]=
    new Object[rsmd.getColumnCount()][nbLignes] ;
int lig=0 ;
while (rs.next()) {
    for (int i=0;i<colName.length;i++)
        switch (rsmd.getColumnType(i+1)) {
            case Types.INTEGER :
                resultat[i][lig]=new Integer(rs.getInt(colName[i])) ;
                break ;
```

# Un exemple d'analyse dynamique de table II

```
case Types.VARCHAR :  
    resultat[i][lig]=new String(rs.getString(colName[i])) ;  
    break ;  
    ...  
}  
}
```

- L'interface DatabaseMetaData
  - API très riche (et donc complexe)
  - Permet de connaître les possibilités du SGBD
  - Portable (de nombreux outils génériques)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable

- L'interface DatabaseMetaData
  - API très riche (et donc complexe)
    - Permet de connaître les possibilités du SGBD
    - Portable (de nombreux outils génériques)
  - Alternative : les tables systèmes postgres
    - Simple
    - non portable

- L'interface DatabaseMetaData
  - API très riche (et donc complexe)
  - Permet de connaître les possibilités du SGBD
  - Portable (de nombreux outils génériques)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable

- L'interface DatabaseMetaData
  - API très riche (et donc complexe)
  - Permet de connaître les possibilités du SGBD
  - Portable (de nombreux outils génériques)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable

- L'interface DatabaseMetaData
  - API très riche (et donc complexe)
  - Permet de connaître les possibilités du SGBD
  - Portable (de nombreux outils génériques)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable



- L'interface DatabaseMetaData
  - API très riche (et donc complexe)
  - Permet de connaître les possibilités du SGBD
  - Portable (de nombreux outils génériques)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable

- L'interface DatabaseMetaData
  - API très riche (et donc complexe)
  - Permet de connaître les possibilités du SGBD
  - Portable (de nombreux outils génériques)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable