

Le protocole JDBC (Java DataBase Connectivity)

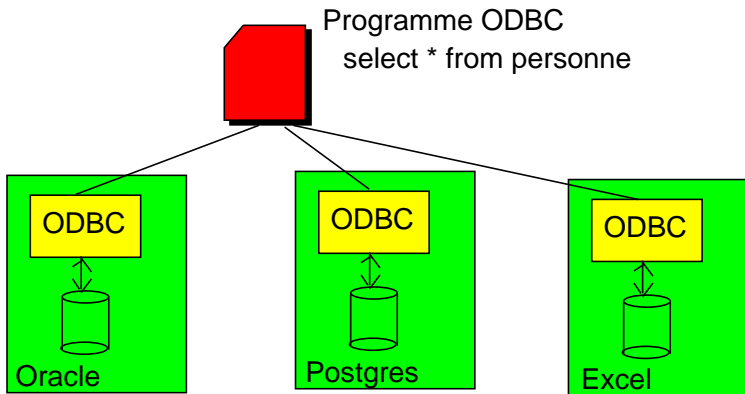
Olivier Caron

Polytech Lille
Avenue Paul Langevin Cité Scientifique Lille 1
59655 Villeneuve d'Ascq cedex

<http://ocaron.plil.fr>
Olivier.Caron@polytech-lille.fr



L'ancêtre ODBC



- Langage Java (multi-plateforme :-)

¹Un fichier `First.java` contenant les exemples JDBC du cours est disponible sur

- Langage Java (multi-plateforme :-)
- Interface de programmation (API)¹

¹Un fichier `First.java` contenant les exemples JDBC du cours est disponible sur

JDBC

- Langage Java (multi-plateforme :-)
- Interface de programmation (API)¹
- Adopté et implémenté par presque tous les constructeurs

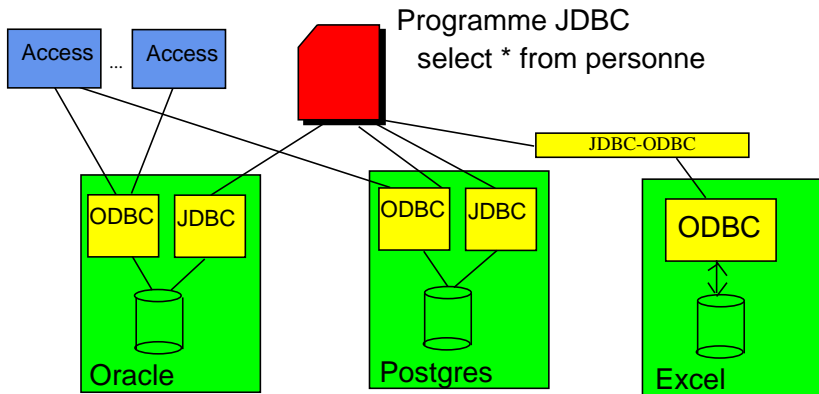
¹Un fichier `First.java` contenant les exemples JDBC du cours est disponible sur

JDBC

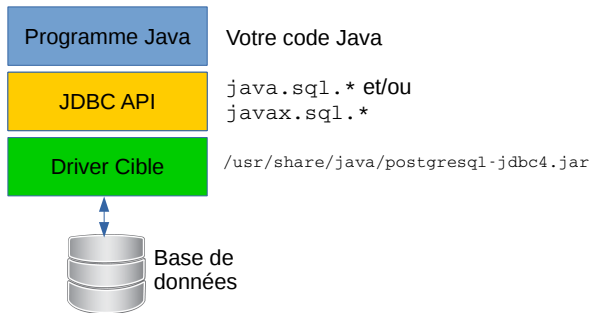
- Langage Java (multi-plateforme :-)
- Interface de programmation (API)¹
- Adopté et implémenté par presque tous les constructeurs
- Passerelle ODBC-JDBC

¹Un fichier `First.java` contenant les exemples JDBC du cours est disponible sur

Des utilisations possibles des protocoles



Structuration d'un programme JDBC



Exécution d'un programme Java-JDBC :

```
java -cp /usr/share/java/postgresql-jdbc4.jar:. progs.First
```


Déroulement d'un programme JDBC

- Déroulement classique :

Déroulement d'un programme JDBC

- Déroulement classique :
 - 1 Connexion à une base

Déroulement d'un programme JDBC

- Déroulement classique :
 - 1 Connexion à une base
 - 2 Exécution de requêtes et récupération des données

Déroulement d'un programme JDBC

- Déroulement classique :
 - 1 Connexion à une base
 - 2 Exécution de requêtes et récupération des données
 - 3 Déconnexion

Déroulement d'un programme JDBC

- Déroulement classique :
 - 1 Connexion à une base
 - 2 Exécution de requêtes et récupération des données
 - 3 Déconnexion
- Plein d'autres variantes (ex: accès multi-bases)

Établissement d'une connexion

- Pour se connecter à une base, il faut fournir les éléments suivants :

Établissement d'une connexion

- Pour se connecter à une base, il faut fournir les éléments suivants :
 - 1 L'adresse du serveur de base de données (son nom de machine)

Établissement d'une connexion

- Pour se connecter à une base, il faut fournir les éléments suivants :
 - 1 L'adresse du serveur de base de données (son nom de machine)
 - 2 Le type du serveur de base de données (postgresql, mysql, oracle, ...)

Établissement d'une connexion

- Pour se connecter à une base, il faut fournir les éléments suivants :
 - 1 L'adresse du serveur de base de données (son nom de machine)
 - 2 Le type du serveur de base de données (postgresql, mysql, oracle, . . .)
 - 3 Le nom de la base

Établissement d'une connexion

- Pour se connecter à une base, il faut fournir les éléments suivants :
 - 1 L'adresse du serveur de base de données (son nom de machine)
 - 2 Le type du serveur de base de données (postgresql, mysql, oracle, ...)
 - 3 Le nom de la base
 - 4 Les nom et mot de passe de l'utilisateur référencé du serveur

Établissement d'une connexion

- Pour se connecter à une base, il faut fournir les éléments suivants :
 - 1 L'adresse du serveur de base de données (son nom de machine)
 - 2 Le type du serveur de base de données (postgresql, mysql, oracle, ...)
 - 3 Le nom de la base
 - 4 Les nom et mot de passe de l'utilisateur référencé du serveur
- Les éléments 1 à 3 seront fournis via une URL de la forme :

`jdbc:typeServeur://adresseServeur/nomBaseDonnees`

Exemple : `jdbc:postgresql://houplin/videoclub`

Établissement d'une connexion

- Pour se connecter à une base, il faut fournir les éléments suivants :
 - 1 L'adresse du serveur de base de données (son nom de machine)
 - 2 Le type du serveur de base de données (postgresql, mysql, oracle, ...)
 - 3 Le nom de la base
 - 4 Les nom et mot de passe de l'utilisateur référencé du serveur
- Les éléments 1 à 3 seront fournis via une URL de la forme :
`jdbc:typeServeur://adresseServeur/nomBaseDonnees`
Exemple : `jdbc:postgresql://houplin/videoclub`
- Ces données seront idéalement externalisées du code source Java dans un fichier de propriétés.

Fichier de propriétés

- Un fichier de propriétés permet d'avoir un code évolutif sans recompilation
- Soit le fichier de propriétés de nom "database.properties":

```
jdbc.serverType=postgresql  
jdbc.serverHost=houplin  
jdbc.dbName=test  
jdbc.user=admin  
jdbc.password=secret
```

Exemple d'une connexion JDBC (1/2)

```
package progs ;

import java.sql.* ;
import java.io.* ;
import java.util.Properties ;

public class First {
    public static void main(String args[]) {
        try {
            Properties props= new Properties () ;
            FileInputStream in=new FileInputStream("database.properties") ;
            props.load(in) ;
            String url = "jdbc:"+
                props.getProperty("jdbc.serverType") + "://" +
                props.getProperty("jdbc.serverHost") + "/" +
                props.getProperty("jdbc.dbName") ;
            String user=props.getProperty("jdbc.user") ;
            String password=props.getProperty("jdbc.password") ;
```

Exemple d'une connexion JDBC (2/2)

```
Connection session = DriverManager.getConnection(url ,user ,password);
System.out.println("Connexion établie") ;
} catch (FileNotFoundException e1) {
    System.err.println("Fichier_de_propriétés non trouvé") ;
} catch (IOException e2) {
    System.err.println("Erreur_lecture_fichier_propriété") ;
} catch (SQLException e) {
    System.err.println("Erreur_connexion_base") ;
}
}
}
```

L'interface `Statement`

- Le programme précédent a permis d'obtenir une variable `session` de type `Connexion`.

L'interface `Statement`

- Le programme précédent a permis d'obtenir une variable `session` de type `Connexion`.
- Un objet de type `Statement` est nécessaire pour définir des requêtes SQL et recevoir des données issues de ces requêtes.

L'interface `Statement`

- Le programme précédent a permis d'obtenir une variable `session` de type `Connexion`.
- Un objet de type `Statement` est nécessaire pour définir des requêtes SQL et recevoir des données issues de ces requêtes.
- Les données issues de requêtes `select` sont stockées dans des objets de type `ResultSet`.

L'interface `Statement`

- Le programme précédent a permis d'obtenir une variable `session` de type `Connexion`.
- Un objet de type `Statement` est nécessaire pour définir des requêtes SQL et recevoir des données issues de ces requêtes.
- Les données issues de requêtes `select` sont stockées dans des objets de type `ResultSet`.
- La fermeture d'un `Statement` engendre la fermeture automatique des tous les `ResultSet` associés.

Statement : requêtes de modification

- **Toutes** les méthodes doivent prendre en compte `SQLException`
- La méthode `createStatement` permet d'obtenir un objet de type `Statement` à partir d'un objet de type `Connection` (lig. 1)
- La méthode `int executeUpdate(String requeteSQL)` exécute toute requête SQL (hors `select`) et retourne le nombre d'*éléments* concernés par la commande (ligs 4-5).

```
1 Statement stmt = session.createStatement();
2 stmt.executeUpdate("create_table_tester_"+
3     "(num_integer_primary_key, ch_text)");
4 int nb= stmt.executeUpdate("insert_into_tester_values_"+
5     "(1, 'dupont '), (2, 'durant ')");
6 System.out.println (nb+"lignes insérées");
```

Consultation de la base

- Le processus de consultation (et même de modification) est le suivant :

Consultation de la base

- Le processus de consultation (et même de modification) est le suivant :
 - 1 Création d'un objet de type `Statement`. Selon les paramètres fournis lors de cette étape, les possibilités de lecture des données reçues ainsi que d'écriture diffèrent.

Consultation de la base

- Le processus de consultation (et même de modification) est le suivant :
 - 1 Création d'un objet de type `Statement`. Selon les paramètres fournis lors de cette étape, les possibilités de lecture des données reçues ainsi que d'écriture diffèrent.
 - 2 Exécution d'une requête SQL de type `Select`, données stockées dans un objet de type `ResultSet`.

Consultation de la base

- Le processus de consultation (et même de modification) est le suivant :
 - 1 Création d'un objet de type `Statement`. Selon les paramètres fournis lors de cette étape, les possibilités de lecture des données reçues ainsi que d'écriture diffèrent.
 - 2 Exécution d'une requête SQL de type `Select`, données stockées dans un objet de type `ResultSet`.
 - 3 Consultation/modification des données d'un `ResultSet`

Paramétrage de Statement (1/2)

- La méthode générale de création d'un Statement est:

```
Statement createStatement(int resultSetType,  
int resultSetConcurrency)
```

Paramétrage de Statement (1/2)

- La méthode générale de création d'un Statement est:
`Statement createStatement(int resultSetType,
int resultSetConcurrency)`
- Les valeurs possibles de `resultSetType` :

Paramétrage de Statement (1/2)

- La méthode générale de création d'un Statement est:

```
Statement createStatement(int resultSetType,  
int resultSetConcurrency)
```
- Les valeurs possibles de `resultSetType` :
 - **ResultSet.Type_FORWARD_ONLY (par défaut)** la lecture des données issues de la requête ne pourra se faire que de la première ligne vers la dernière. Les données sont celles de l'état de la base au moment de l'exécution de la requête (pas de rafraîchissement possible).

Paramétrage de Statement (1/2)

- La méthode générale de création d'un Statement est:

```
Statement createStatement(int resultSetType,  
int resultSetConcurrency)
```
- Les valeurs possibles de `resultSetType` :
 - **ResultSet.Type_FORWARD_ONLY (par défaut)** la lecture des données issues de la requête ne pourra se faire que de la première ligne vers la dernière. Les données sont celles de l'état de la base au moment de l'exécution de la requête (pas de rafraîchissement possible).
 - **ResultSet.Type_SCROLL_INSENSITIVE** la lecture peut se faire en avant et/ou arrière. Pas de rafraîchissement possible des données.

Paramétrage de Statement (1/2)

- La méthode générale de création d'un Statement est:

```
Statement createStatement(int resultSetType,  
int resultSetConcurrency)
```
- Les valeurs possibles de `resultSetType` :
 - **ResultSet.Type_FORWARD_ONLY (par défaut)** la lecture des données issues de la requête ne pourra se faire que de la première ligne vers la dernière. Les données sont celles de l'état de la base au moment de l'exécution de la requête (pas de rafraîchissement possible).
 - **ResultSet.Type_SCROLL_INSENSITIVE** la lecture peut se faire en avant et/ou arrière. Pas de rafraîchissement possible des données.
 - **ResultSet.Type_SCROLL_SENSITIVE** la lecture peut se faire en avant et/ou arrière. Rafraîchissement possible des données.

Paramétrage de Statement (2/2)

- Les valeurs possibles de `resultSetConcurrency` :

Paramétrage de Statement (2/2)

- Les valeurs possibles de `resultSetConcurrency` :
 - **`ResultSet.CONCUR_READ_ONLY` (par défaut)** les données du `ResultSet` ne sont pas modifiables

Paramétrage de Statement (2/2)

- Les valeurs possibles de `resultSetConcurrency` :
 - **`ResultSet.CONCUR_READ_ONLY` (par défaut)** les données du `ResultSet` ne sont pas modifiables
 - **`ResultSet.CONCUR_UPDATABLE`** les données du `ResultSet` sont modifiables et synchronisables avec la base.

Paramétrage de Statement (2/2)

- Les valeurs possibles de `resultSetConcurrency` :
 - **`ResultSet.CONCUR_READ_ONLY` (par défaut)** les données du `ResultSet` ne sont pas modifiables
 - **`ResultSet.CONCUR_UPDATABLE`** les données du `ResultSet` sont modifiables et synchronisables avec la base.
- La méthode `stmt = session.createStatement()` est un raccourci pour :

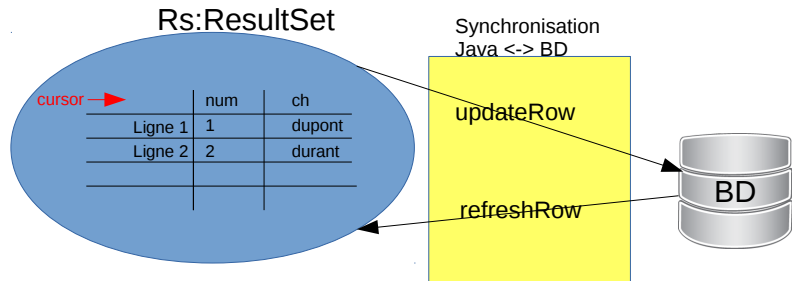
```
stmt = session.createStatement(ResultSet.TYPE_FORWARD_ONLY,  
ResultSet.CONCUR_READ_ONLY)
```

Exécution d'une requête `select`

- Utilisation de la méthode `executeQuery` de la classe `Statement`.

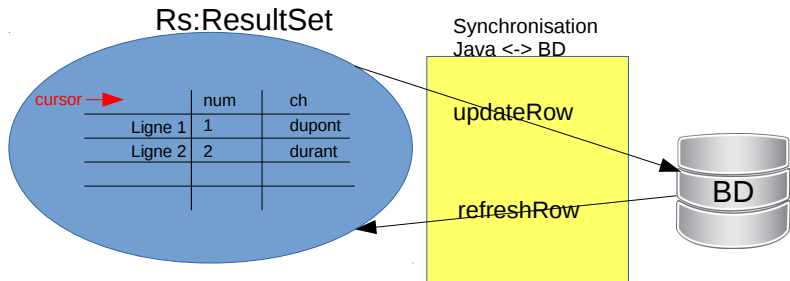
```
Statement stmt = session.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
                                           ResultSet.CONCUR_UPDATABLE) ;  
ResultSet rs = stmt.executeQuery("select _num, _ch_from _tester") ;
```

L'objet de type Result Set



- Gestion d'un curseur pour parcourir les lignes résultats de la requête.

L'objet de type Result Set



- Gestion d'un curseur pour parcourir les lignes résultats de la requête.
- Après initialisation de l'objet, le curseur se trouve **avant** la première ligne

ResultSet : gestion du curseur

- Plusieurs méthodes booléennes pour déplacer le curseur

ResultSet : gestion du curseur

- Plusieurs méthodes booléennes pour déplacer le curseur
- Les méthodes retournent `false` si opération impossible (ex: fin de table), la première ligne est la ligne 1.

méthode	action
<code>next()</code>	déplacement vers la ligne suivante
<code>previous()</code>	déplacement vers la ligne précédente
<code>absolute(i)</code>	$i > 0$, aller à la $i^{\text{ème}}$ ligne
<code>absolute(-i)</code>	$i > 0$, aller à la $i^{\text{ème}}$ ligne en partant de la dernière
<code>relative(i)</code>	descendre de i lignes
<code>relative(-i)</code>	remonter de i lignes
<code>afterlast()</code>	aller en fin de la table (après la dernière ligne)
<code>srs.isAfterLast()</code>	retourne vrai si fin de table
<code>last()</code>	aller à la dernière ligne

ResultSet : lecture des lignes

- Plusieurs méthodes pour lire les données de la ligne courante (celle pointée par le curseur)

ResultSet : lecture des lignes

- Plusieurs méthodes pour lire les données de la ligne courante (celle pointée par le curseur)
- Deux modes : n^o de colonne (indicée à partir de 1) ou nom de colonne

méthode	action
Date getDate(int i)	retourne l'objet Date de la colonne i
Date getDate(String col)	retourne l'objet Date de la colonne col
int getInt(int i)	retourne un entier de la colonne i
int getInt(String col)	retourne un entier de la colonne col
String getString(int i)	retourne l'objet String de la colonne i
String getString(String col)	retourne l'objet String de la colonne col
...	...

Exemple de consultation

```
int i = 1;
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`)
- En mode `FORWARD_ONLY`, `next()` est la seule méthode possible.

ResultSet : modification de lignes

- Des nouvelles méthodes pour ResultSet :

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

```
void updateRow() ,void refreshRow()
```

```
void cancelRowUpdates()
```

- Un exemple :

```
rs.first() ; // première ligne
```

```
rs.refreshRow() ; // synchronisation BD → Java
```

```
rs.updateString("ch", "dubois") ;
```

```
rs.updateRow() ; // synchronisation Java → BD
```

```
rs.updateString("ch", "Caron") ;
```

```
rs.cancelRowUpdates(); // précédente ligne invalidée
```

ResultSet : insertion de lignes

- Des nouvelles méthodes pour ResultSet :

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

- Un exemple :

```
// phase d'insertion  
rs.moveToInsertRow();  
rs.updateInt("num",3);  
rs.updateString("ch", "garcia");  
rs.insertRow(); // synchronisation Java → BD  
rs.moveToCurrentRow(); // retour au curseur avant insertion
```

Préparation des requêtes

- Lorsqu'elles sont exécutées plusieurs fois

Préparation des requêtes

- Lorsqu'elles sont exécutées plusieurs fois
- Les SGBD ont une version compilée de la requête (performances)

Préparation des requêtes

- Lorsqu'elles sont exécutées plusieurs fois
- Les SGBD ont une version compilée de la requête (performances)
 - dépend des SGBD

Préparation des requêtes

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

Préparation des requêtes

- 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

Préparation des requêtes

- 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

L'interface PreparedStatement

- Création d'une requête préparée :

```
PreparedStatement req = session.prepareStatement(  
    "select * from _tester where _num=? and _ch=?" );
```

- Paramétrer la requête :

```
req.setInt(1, 3) ; req.setString(2, "garcia" ) ;
```

- Exécution de la requête :

```
ResultSet rs2 = req.executeQuery() ;
```

Fermeture de la connexion

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

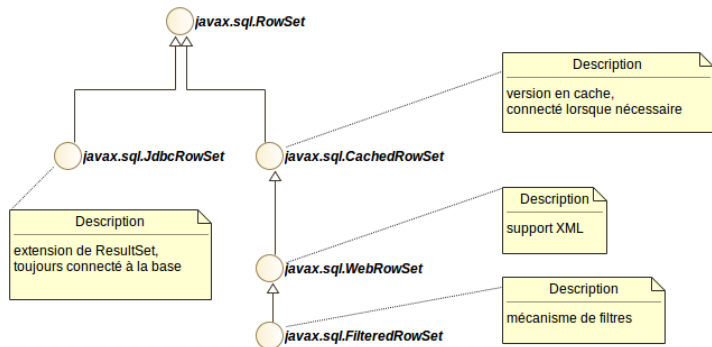
```
stmt.close () ;  
connect.close () ;
```

Les objets RowSet : extension de ResultSet

- JDBC 4 : fusion de Connection, PreparedStatement et ResultSet

Les objets RowSet : extension de ResultSet

- JDBC 4 : fusion de Connection, PreparedStatement et ResultSet
- Intègre mécanisme d'observation des données modifiées.



Exemple RowSet

```
JdbcRowSet jrs = RowSetProvider.newFactory().createJdbcRowSet();  
/* Connexion base distante : */  
jrs.setUrl(url);  
jrs.setUsername(user);  
jrs.setPassword(password);  
/* établissement requête (PreparedStatement) : */  
jrs.setCommand("SELECT * FROM tester WHERE ch=?");  
jrs.setString(1, "garcia");  
/* exécution requête (équivalent à executeQuery) : */  
jrs.execute();  
while (jrs.next()) ...
```

Gestion des transactions

- Le mode par défaut est "auto-commit" : chaque requête SQL forme une transaction
- Initier une transaction, choix du mode d'isolation standard :

```
session.setAutoCommit(false) ;  
session.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE) ;
```

- 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") ;  
session.commit() ;  
session.setAutoCommit(true) ;
```

Annuler une transaction

- méthode `rollback()`

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


Prise en compte de SQL 3

- SQL 3 : unification de concepts orienté-objet et le monde relationnel

Prise en compte de SQL 3

- SQL 3 : unification de concepts orienté-objet et le monde relationnel
- Des colonnes de type non simple :

Prise en compte de SQL 3

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

Prise en compte de SQL 3

- 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")`)

Le niveau Meta

Une meta-donnée est une donnée qui décrit une donnée

- Des Exemples :

Le niveau Meta

Une meta-donnée est une donnée qui décrit une donnée

- Des Exemples :
 - Modèle Relationnel: `table(colonne1 type1, ...)`

Le niveau Meta

Une meta-donnée est une donnée qui décrit une donnée

- Des Exemples :
 - Modèle Relationnel: `table(colonne1 type1, ...)`
 - Postgres `pg_database, pg_user, pg_class, ...`

Le niveau Meta

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()`, ...

Le niveau Meta

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)

JDBC et le niveau Meta

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

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

Un exemple d'analyse dynamique de table

```
rs = stmt.executeQuery(uneRequeteSQL);
rs.last(); int nbLignes = rs.getRow(); rs.beforeFirst();
ResultSetMetaData rsmd = rs.getMetaData() ;
String colName[] = new String[rsmd.getColumnCount()] ;
for (int i=0;i<rsmd.getColumnCount();i++) { // gestion des noms de col
    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 ;
            case Types.VARCHAR: resultat[i][lig]=new String(rs.getString(colName[i]));
                break ;
            ...
        }
    }
}
```

Analyse dynamique de la base

- L'interface DatabaseMetaData

Analyse dynamique de la base

- L'interface DatabaseMetaData
 - API très riche (et donc complexe)

Analyse dynamique de la base

- L'interface DatabaseMetaData
 - API très riche (et donc complexe)
 - Permet de connaître les possibilités du SGBD

Analyse dynamique de la base

- L'interface DatabaseMetaData
 - API très riche (et donc complexe)
 - Permet de connaître les possibilités du SGBD
 - Portable (de nombreux outils génériques)

Analyse dynamique de la base

- L'interface DatabaseMetaData
 - API très riche (et donc complexe)
 - Permet de connaître les possibilités du SGBD
 - Portable (de nombreux outils génériques)
- Permet de développer des outils génériques (aka PhpMyAdmin)

Conclusion

- API très riche, une petite partie de JDBC est présentée dans ce cours.

Conclusion

- API très riche, une petite partie de JDBC est présentée dans ce cours.
- Une API unique quelque soit le SGBD

Conclusion

- API très riche, une petite partie de JDBC est présentée dans ce cours.
- Une API unique quelque soit le SGBD
- En constante évolution : gestion de "row sets", gestion de blobs, support de SQL XML, programmation par annotations Java, . . . ,