Le protocole JDBC

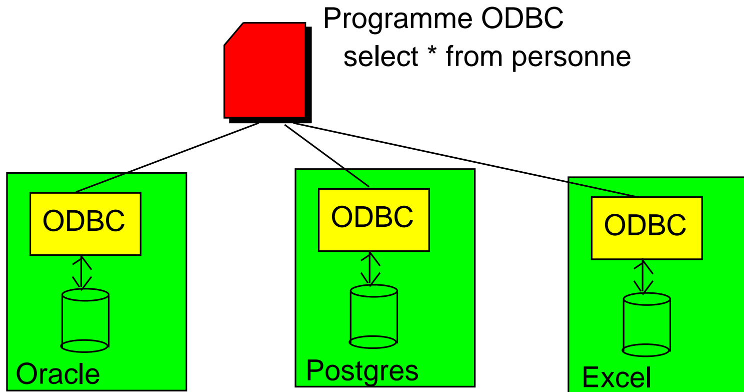
Olivier Caron

© Polytech'Lille

SGBD - GIS4

Polytech'Lille

L'ancêtre ODBC



Polytech'Lille

JDBC

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

Polytech'Lille

JDBC

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

Polytech'Lille

JDBC

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

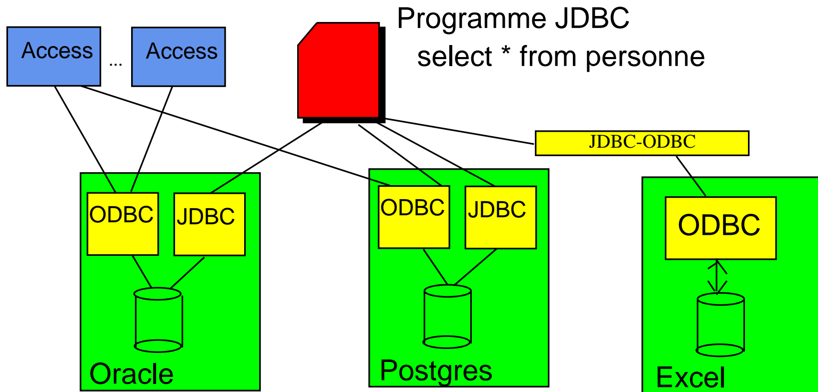
Polytech'Lille

JDBC

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

Polytech'Lille

Des utilisations possibles des protocoles



Polytech'Lille

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`

Polytech'Lille

Obtenir une connexion

- 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

Chargement driver, version 2 (1/2)

- 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
```

Polytech'Lille

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) ;
```

Polytech'Lille

L'interface Statement

- 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`

Polytech'Lille

Modification de la base

- 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,...

Polytech'Lille

- API : `ResultSet executeQuery(String requeteSQL)`

```
ResultSet rs = stmt.executeQuery(
                    "select * from tester") ;
```

Polytech'Lille

Récupération des données (1/2)

La classe `ResultSet` dispose des méthodes suivantes :

boolean next()	retourne true s'il reste un n-uplet à lire
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
...	...

Polytech'Lille

Récupération des données (2/2)

- 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

Polytech'Lille

Fermeture de la connexion

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

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

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

Polytech'Lille

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

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

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

Polytech'Lille

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

Polytech'Lille

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

Polytech'Lille

L'interface PreparedStatement

- 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 !

Polytech'Lille

Gérer les transactions

- 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) ;
```

Gestion des erreurs

- 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 Procédures stockées

- 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)

JDBC Procédures stockées et Postgresql

- 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")) ;
```

Gestion des warnings

- 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();
  }
}
```

Applet et JDBC

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

Polytech'Lille

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>
```

Polytech'Lille

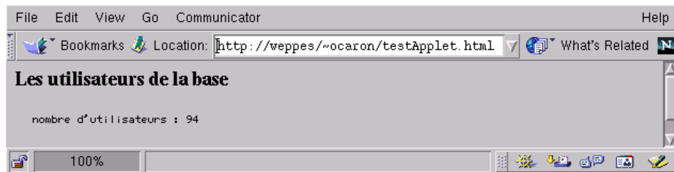
```
import java.sql.* ;
import java.applet.Applet ;
import java.awt.Label ;
public class AppletJDBC extends Applet {
 public void init() {
  Label label=new Label() ;
  Statement stmt = null ; Connection db = null ;
  ResultSet rs =null ;
  try {
    Class.forName("org.postgresql.Driver") ;
    db=DriverManager.getConnection(
    "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 : "+
```

Polytech'Lille

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)



JDBC 2.0

- 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`

Polytech'Lille

JDBC 2.0

- 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`

Polytech'Lille

JDBC 2.0

- 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`

Polytech'Lille

JDBC 2.0

- 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`

Polytech'Lille

JDBC 2.0

- 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`

Polytech'Lille

Scrollable Result Sets

- 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

Polytech'Lille

Les possibilités de déplacement

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

Polytech'Lille

Modification de lignes

- 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
```

Insertion de lignes

- 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
```

Suppression de lignes et Gérer les changements

- Des nouvelles méthodes pour `ResultSet` :
  `void deleteRow()`
  `void refreshRow()` (mode TYPE_SCROLL_SENSITIVE)

Programmes Batch

- 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) {...
```

Polytech'Lille

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")`)

Polytech'Lille

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")`)

Polytech'Lille

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")`)

Polytech'Lille

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")`)

Polytech'Lille

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)

Polytech'Lille

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)

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)

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)

Polytech'Lille

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)

Polytech'Lille

JDBC et le niveau Meta

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

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

Polytech'Lille

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 ;
     ...
  }
}
```

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)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable

Polytech'Lille

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)
- 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

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)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable

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)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable

Polytech'Lille

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)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable

Polytech'Lille

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)
- Alternative : les tables systèmes postgres
  - Simple
  - non portable

Polytech'Lille