Java et les SGBDR

Librairies JDBC

Yves Bekkers

Plan

- Bases de données, interface JDBC
  - Introduction, concepts
  - Pilote et gestionnaire de pilotes
    - Interface `java.sql.Driver`
    - Le cas d'ODBC
  - Accès aux données
    - Charger un pilote
    - Créer une connexion à la base
    - Créer une requête (Statement)
    - Exécuter une requête
    - Présenter les résultats
  - Accès aux méta-données
  - Traiter les erreurs

Introduction - concepts

Librairies JDBC

- JDBC : Java DataBase Connectivity
- Qui : Sun
- Quoi : deux librairies (accessibles dans Java 1.4)
  - `java.sql` : *JDBC 2.0 Core API*
  - `javax.sql` : *JDBC 2.0 Optional Package API*
- Objectif :
  - Accéder aux SGBD à l'aide de requètes SQL
  - Récupérer les résultats en Java
  - Le type de la base est transparent (Sybase, Oracle, MySql, Access …) *généricité, standardisation*

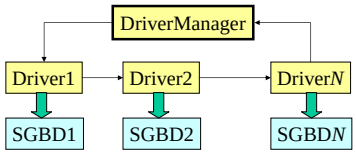
JDBC
Généricité - standardisation

- **Abstraction des SGBDRs** :
  - JDBC définit un ensemble d'interfaces
    - `Driver, Connection, Statement, ResultSet` …
  - JDBC définit une classe : `DriverManager`
- **Implementations** : Chaque base particulière possède son implémentation des interfaces : c'est la notion de "*pilote*"

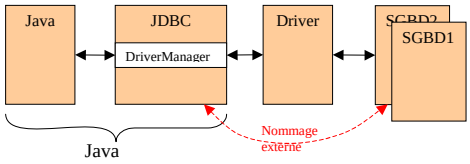
Pilote et gestionnaire de pilotes
"*Driver et driver manager*"

Pilotes (`Drivers`) et gestionnaire de pilotes (`DriverManager`)

- Classe `java.sql.DriverManager` :
  - Le gestionnaire de pilotes offert par la JVM
- Interface `java.sql.Driver`
  - Définit le point d'entrée de tout accès à une base



Un même driver peut piloter plusieurs bases



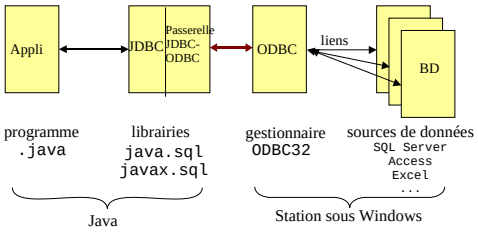
Interface `java.sql.Driver`

- Chaque vendeur fournit une implémentation
  - Certaines sont fournis avec le Java core
    - Accès à ODBC `sun.jdbc.odbc.JdbcOdbcDriver`
  - Les autres doivent être importées
    - postGresql : `org.postgresql.Driver`
- Généralement on trouve ces classes dans des fichiers jar dans les distributions du constructeur
- Dans le pire des cas :
  - On peut trouver une liste de plus de 160 pilotes à http://industry.java.sun.com/products/jdbc/drivers

Le cas d'ODBC

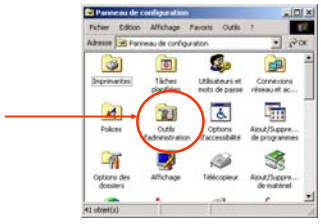
Accès à une base MS Access

- Passerelle Jdbc:odbc
- gestionnaire de données ODBC32



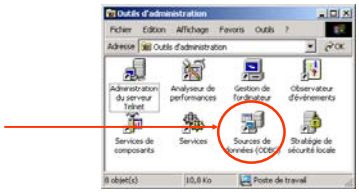
Administration des liens ODBC sous Windows 2000 (1)

- Outils d'administration (panneau de configuration)



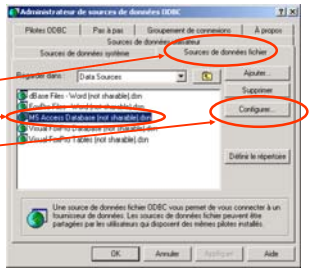
Administration des liens ODBC sous Windows 2000 (2)

- Source odbc (sous outils d'administration)



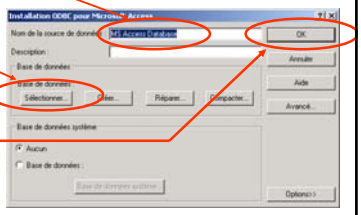
Administration des liens ODBC sous Windows 2000 (3)

- fichier Access
  - Onglet données fichier
  - Type "MS Access"
  - Configurer



Administration des liens ODBC sous Windows 2000 (4)

- fichier Access
  - Nom donnée au lien
  - Emplacement du fichier
  - Confirmer



Accès aux données

Accès aux données - 5 étapes

1. Charger un pilote
2. Créer une connexion à la base
3. Créer une requête (Statement)
4. Exécuter une requête
5. Présenter les résultats

Première étape *"charger un pilote"*

Charger un pilote

- Classe `java.sql.DriverManager`
  - Fournit les utilitaires de gestion des connexions et des pilotes
  - Elle ne contient que des méthodes statiques
- Les pilotes sont chargés dynamiquement
  - *implicitement* en instanciant la "system property" `jdbc.drivers`
  - *explicitement* par la méthode statique `forName()` de la classe `Class` (voir ci-après)

Conventions à propos du chargement des pilotes

- *Classe `DriverManager`*
  - Elle inspecte la propriété système `jdbc.drivers` au moment de son instanciation
- *Classes implémentant l'interface `Driver`*
  - elles doivent s'enregistrer d'elles mêmes au près du `DriverManager` au moment de leur instanciation

Exemples de chargement

- Initialisation avant appel de l'API
  `jdbc.drivers=`*`nom-de-classe-Driver`*
- Initialisation au moment de l'appel de l'API
  `java -Djdbc.drivers=`*`nom-de-classe`*
- Initialisation dans le programme

```
try {
   Class.forName("nom-de-classe");
} catch(ClassNotFoundException ex)
   {...}
```


Remarques

- L'utilisation de propriétés (systèmes ou privées à l'API) permet de ne pas "*cabler*" le nom de classes dans le code java
- La propriété `jdbc.drivers` peut contenir plusieurs noms complets de classes, il suffit de séparer les noms par des caractères "`:`"
- Exemple:
  - `foo.bah.Driver`:`wombat.sql.Driver`
    nom1 *nom2*

Seconde étape
"*Créer une connexion à la base*"

Classe `java.sql.Connection`

- Demande de connexion
  - méthode statique `getConnection(String)`classe `DriverManager`
- Le driver manager essaye de trouver un driver approprié d'après la chaîne passée en paramètre
- Structure de la chaîne décrivant la connexion
  `jdbc:`*`protocole`*`:`*`URL`*

  Exemples

```
jdbc:odbc:epicerie
jdbc:msql://athens.imaginaire.com:4333/db
jdbc:oracle:thin:@blabla:1715:test
```

Classe `java.sql.Connection` (suite)

- Connexion sans information de sécurité

```
Connection con =
  DriverManager.getConnection
  ("jdbc:odbc:epicerie");
```

- Connexion avec informations de sécurité

```
Connection con =
  DriverManager.getConnection
  ("jdbc:odbc:epicerie", user, password);
```

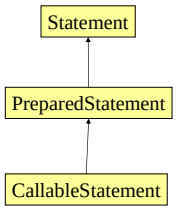
- Dans tous les cas faut récupérer l'exception `java.sql.SQLException`

Troisième étape
"*Créer une requète*"

Trois types de requètes

- Requètes simples
  - `java.sql.Statement`
- Requètes Précompilées
  - `java.sql.PreparedStatement`
  - Pour une opération réalisée à plusieurs reprises
- Procédures stockées
  - `java.sql.CallableStatement`
  - Pour lancer une procédure du SGBD

Héritage entre interfaces requètes



Interface `java.sql.Statement`

- C'est le type d'objet
  - pour exécuter une requète SQL simple
  - pour recevoir ses résultats
- Exemple de création

```
Connection con =
  DriverManager.getConnection
  ("jdbc:odbc:epicerie");

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


`Statement`
La réquete sql n'est pas liée à l'objet Statement

- Initialisation

```
Connection con = getConnection();
Statement stmt = con.createStatement();
```

- Exécuter une requête

```
String sql = "SELECT * FROM fournisseur";
ResultSet rs = stmt.executeQuery(sql);
```

PreparedStatement

La réquete sql est liée
à l'objet PreparedStatement

- Requètes paramétrées

```
String s = "UPDATE EMPLOYEES" +
" SET SALARY = ? WHERE ID = ?"
PreparedStatement pstmt =
  con.prepareStatement(s);

pstmt.setBigDecimal(1, 153833.00);
pstmt.setInt(2, 110592);
```


Quatrième étape

"*Exécuter une requête*"

Interrogation de la base

- Instruction SQL « select »

```
String sql = "SELECT * FROM
  fournisseur";
ResultSet rs =
  stmt.executeQuery(sql);
```


Mise à jour : effacer une table

- Instruction SQL « drop table »

```
Statement stmt =
  con.createStatement();
int i = stmt.executeUpdate(
   "drop table fournisseur");
```


Statement - Deux types d'exécution

- Requètes "select" retournant un tableau

```
Statement stmt = con.createStatement();
String sql = "SELECT * FROM fournisseur";
ResultSet rs = stmt.executeQuery(sql);
```

- Requètes de mise à jour et de création

```
Statement stmt = conn.createStatement();
String sql = "INSERT INTO Customers " +
    "VALUES (1001, 'Simpson', 'Mr.', " +
    "'Springfield', 2001)";
int i = stmt.executeUpdate(sql);
```


executeQuery ()

- Pour lancer une requète "SELECT" statique

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

String s = "select * from employes";
ResultSet rs = stmt.executeQuery(s);
```

- Un ResultSet est un objet qui modélise un tableau à deux dimensions
  - Lignes (*row*)
  - Colonnes (valeurs d'attributs)

executeUpdate() - 1 INSERT INTO

- Pour lancer une requête INSERT

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

String s = "INSERT INTO test (code,val)" +
   "VALUES(" + valCode + ", '" +val +"')";
int i = stmt.executeUpdate(s);
```

- Le résultat est un entier donnant le nombre de lignes créées

executeUpdate() - 2 UPDATE

- Pour lancer une requête UPDATE

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

String s = "UPDATE table
  SET column = expression
  WHERE predicates";
int i = stmt.executeUpdate(s);
```

- Le résultat est un entier donnant le nombre de mises-à-jour

executeUpdate() - 3 DELETE

- Pour lancer une requête DELETE

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

String s = "DELETE FROM table
  WHERE predicates";
int i = stmt.executeUpdate(s);
```

- Le résultat est un entier donnant le nombre de d'effacements

PreparedStatement

```
Class.forName(driverClass);
java.sql.Connection connection =
  java.sql.DriverManager.getConnection(url,user
  ,password);

 PreparedStatement pstmt =
connection.prepareStatement(
   "insert into users values(?,?,?,?)");
    pstmt.setInt(1,user.getId());
    pstmt.setString(2,user.getName());
    pstmt.setString(3,user.getEmail());
    pstmt.setString(4,user.getAddress());
    pstmt.execute();
```


Cinquième étape

"*Présenter les résultats*"

Récupération des résultats

```
java.sql.Statement stmt =
  conn.createStatement();
String s = "SELECT a, b, c FROM Table1";
ResultSet rs = stmt.executeQuery(s);
while (rs.next()) {
   int i = rs.getInt("a");
   String s = rs.getString("b");
   byte b[] = rs.getBytes("c");
   System.out.println("ROW = " + i +
      " " + s + " " + b[0]);
}
```

Objets `ResultSet`

```
ResultSet rs = stmt.executeQuery(s);
```

- Se parcourt itérativement *row* par *row*
- Les colonnes sont référencées par leur numéro ou leur nom
- L'accès aux valeurs des colonnes se fait par des méthodes getXXX() où XXX représente le type de l'attribut se trouvant dans la colonne

Objets `ResultSet` (suite)

- Une rangée *(row)* est un tuple
- un RS contient un curseur pointant sur une rangée du résultat
- au départ le pointeur est positionné a*vant la 1ère rangée*
- La méthode next() fait passer à l'enregistrement suivant.
- Le premier appel resultat.next() positionne sur la première rangée
- next() renvoie un booléen true en général, et false lorsque l'on a dépassé le dernier enregistrement.

types SQL - méthodes JDBC

BIGINT	getLong()
BINARY	getBytes()
BIT	getBoolean()
CHAR	getString()
DATE	getDate()
DECIMAL	getBigDecimal()
DOUBLE	getDouble()
FLOAT	getDouble()
INTEGER	getInt()
LONGVARBINARY	getBytes()

types SQL - méthodes JDBC (bis)

LONGVARCHAR	getString()
NUMERIC	getBigDecimal()
OTHER	getObject()
REAL	getFloat()
SMALLINT	getShort()
TIME	getTime()
TIMESTAMP	getTimestamp()
TINYINT	getByte()
VARBINARY	getBytes()
VARCHAR	getString()

Accès aux méta-données d'un `ResultSet`

Méthode `getMetaData()`

- La méthode `getMetaData()` permet d'obtenir les méta-données d'un `ResultSet`.
- Elle renvoie des `ResultSetMetaData`.
- On peut connaître :
- Le nombre de colonnes : `getColumnCount()`
- Le nom d'une colonne : `getColumnName(int col)`
- Le type d'une colonne : `getColumnType(int col)`
- ...

Utilisation des méta-données

```
private static void affiche (ResultSet resultat)
    throws java.sql.SQLException {
  int i;
  ResultSetMetaData rs = resultat.getMetaData ();
  int nbcol = rs.getColumnCount ();

  for (i=1; i<=nbcol; i++) {
    if (i > 1) System.out.print("\t");
    System.out.print(rs.getColumnLabel(i));
  }
```

Entêtes de colonne

Utilisation des méta-données (bis)

```
  boolean encore = resultat.next();
  while (encore) {
     for (i=1; i<=nbcol; i++) {
        if (i > 1) System.out.print("\t");
        System.out.print(resultat.getString(i));
     }
     System.out.println(""); // ligne suivante
     encore = resultat.next();
  }
```

Sortir une ligne

Accès aux méta-données d'une base

Interface `DatabaseMetaData`

```
Connection con = DriverManager.getConnection
   ("jdbc:odbc:epicerie");
// Envoi d'une demande d'information
DatabaseMetaData meta = con.getMetaData();
String[] types = {"table"};
ResultSet rs =
  meta.getTables(null,null,"%",types);
afficher(rs);
```

- le joker "%" permet d'obtenir toutes les tables sans connaître leur nom

Exemples de résultats

CAT	*SCHEME*	*NAME*	*TYPE*	*REMARKS*
..épicerie	null	fournisseur	TABLE	null
..épicerie	null	mafourniture	TABLE	null
..épicerie	null	produit	TABLE	null

Traitement par lot

```
con.setAutoCommit(false);
stmt.addBatch("INSERT INTO fournisseur " +
    "VALUES('f9', 'Cochoux', 0.035, 'Brest')");
stmt.addBatch("INSERT INTO fournisseur " +
    "VALUES('f10', 'Enaf', 0.05, 'rennes')");
int[] updateCounts = stmt.executeBatch();
con.commit();
con.setAutoCommit(true);
stmt.clearBatch();
```

Traiter les erreurs

Erreurs et avertissement

Deux niveaux :

- *Erreurs* : La plupart des méthodes JDBC sont susceptibles de lever des exceptions `java.sql.SQLException`.
- *Warnings* : avertissements

```
SQLWarning warn = con.getWarnings ();
```


Prévenir les fuites de connections - 1

```
Connection conn = null;
 Statement stmt = null;
 ResultSet rs = null;
 try {
   conn = ... obtenir une connexion dans un pool ...
   stmt = conn.createStatement("select ...");
   rs = stmt.executeQuery();
   ... iterer à travers le result set ...
   rs.close();
   rs = null;
   stmt.close();
   stmt = null;
   conn.close(); // retourner la connexion
   conn = null;  // pour s'assurer qu'elle
                 // n'est pas fermée deux fois
 } catch (SQLException e) {
```


Ramasse miette

```
} catch (SQLException e) {
    ... traiter le erreurs ...
} finally { // ne pas oublier de rendre les connexions
  // fermer resultset, statement et connection
  if (rs != null) {
    try { rs.close(); } catch (SQLException e) { ; }
    rs = null;
  }
  if (stmt != null) {
    try { stmt.close(); } catch (SQLException e) { ; }
    stmt = null;
  }
  if (conn != null) {
    try { conn.close(); } catch (SQLException e) { ; }
    conn = null;
  }
}
```


Récupérer les avertissements

```
if (warn != null) {
   System.out.println ("\n *** Warning ***\n");
   rc = true;
   while (warn != null) {
     System.out.println ("SQLState: " +
          warn.getSQLState ());
     System.out.println ("Message:  " +
          warn.getMessage ());
     System.out.println ("Vendor:   " +
          warn.getErrorCode ());
     System.out.println ("");
     warn = warn.getNextWarning ();
   }
}
```


Conclusion

- Ce que vous avez vu dans ce cours
  - Bases de données, interface JDBC
    - Introduction, concepts
    - Pilote et gestionnaire de pilotes
      - Interface `java.sql.Driver`
      - Le cas d'ODBC
    - Accès aux données
      - Charger un pilote
      - Créer une connexion à la base
      - Créer une requête (Statement)
      - Exécuter une requête
      - Présenter les résultats
    - Accès aux méta-données
    - Traiter les erreurs