Principes d'utilisation des systèmes de gestion de bases de données

JDBC

L3 Informatique

Emmanuel Waller, LRI, Orsay

JDBC

- le mode programme
- JDBC : qu'est-ce que c'est ? Avantages ?
- devant les machines
- interface, portabilité, connexion
- gestion erreurs BD (intuition), transactions
- exécution d'ordres BD : principe, avec/sans paramètres, curseurs, SQL dynamique
- JDBC et l'architecture en couches
- Ref cursor, gestion communication programme-serveur, métadonnées

pbs MP : exécution d'ordres BD

- principe et déroulement
- ordres sans paramètre
- ordres avec paramètres
- appel procédure stockée
- SQL dynamique
- curseurs

principe et déroulement

- création d'un « objet ordre » générique (ordre SQL non fixé)
- envoi de l'ordre au serveur par méthode de cet objet
- si select : récupération du résultat (curseur)
- gestion erreurs

- 3 catégories :
  - ordres BD sans ou avec paramètres
    - paramètre : valeur d'une colonne
  - appel procédure stockée PL/SQL

pbs MP : exécution d'ordres BD

- principe et déroulement
- ordres sans paramètre
- ordres avec paramètres
- appel procédure stockée
- SQL dynamique
- curseurs

ordres SQL avec paramètres

- ordres SQL DML (insert, update, delete, select)
- remarque : même paramètres que dans PL/SQL (rappel : variables table ou colonne impossible)
- déroulement :
  - création « objet ordre » pour
    - un ordre SQL fixé
    - avec des paramètres formels

    + précompilation (« préparation »)
  - affectation des valeurs aux paramètres
  - envoi au serveur, etc.

exemple

- situation : on veut enregistrer l'anniversaire d'une personne (= age++ dans BD)
- principe : where paramétré par le nom de la personne
- code :

```
PreparedStatement stmt = conn.prepareStatement(
    « update personne set age = age+1 where nom = ? »);
stmt.setString(1, « Jeanne »);
stmt.executeUpdate();
```

PreparedStatement

- création par méthode de Connection :

  PreparedStatement prepareStatement(String sql)

  throws SQLException

- String sql :
  - contient un symbole « ? » pour chaque paramètre
  - repérés par leur position de gauche à droite (en commençant à 1)

- affectation par setXXX de PreparedStatement
  - où XXX est un type primitif de Java
  - une méthode pour chaque tel type
- ex :
  - void setString(int indiceParametre, String x)
    throws SQLException
  - x converti en varchar par le SGBD

- exécution par :
  - executeUpdate si sql non requête
  - executeQuery si sql requête
- utilisable même si aucun paramètre (mais inutilement coûteux)
- rem : nombre et type des paramètres doivent être connus

avantage PreparedStatement sur Statement

- ex : même situation, mais plusieurs anniversaires
- possible avec Statement :

```
String[] p = {« Jeanne », « Jules »};
Statement stmt = conn.createStatement();
for (int i=0; i<p.length; i++)
    stmt.executeUpdate(
        « update personne set age = age+1 where nom = ' »
        + p[i] + «' » );
```

- compilation et exécution à chaque fois
- (rq : il faut les apostrophes si chaîne de car. Oracle)

- PreparedStatement :

```
String[] p = {« Jeanne », « Jules »};
PreparedStatement stmt = conn.prepareStatement(
    « update personne set age = age+1 where nom = ? »);
for (int i=0; i<p.length; i++) {
    stmt.setString(1, p[i]);
    stmt.executeUpdate(); }
```

  - précompilation une seule fois
  - exécution plusieurs fois avec nouvelles valeurs des paramètres
  - (rq : pas d'apostrophe)

pbs MP : exécution d'ordres BD

- principe et déroulement
- ordres sans paramètre
- ordres avec paramètres
- appel procédure stockée
- SQL dynamique
- curseurs

appel de procédure stockée

- exemple :
  - on a déjà une procédure stockée : anniversaire(n varchar)
  - incrémente de 1 l'âge de la ligne de nom n
  - on veut l'appeler depuis notre programme Java
- code :

```
String n = « Jeanne »;
CallableStatement stmt = conn.prepareCall(
    « {call anniversaire(?)} »);
stmt.setString(1, « Jeanne »); // comme PreparedStatement
stmt.executeUpdate(); // comme PreparedStatement
```

- CallableStatement hérite de PreparedStatement

- lecture de
  - paramètre OUT d'une procédure stockée
  - valeur renvoyée par une fonction stockée

  après exécution :
  - registerOutParameters : « déclarer » le type du paramètre (parmi java.sql.Types)
  - getXXX

- appel de la fonction âge :

```
CallableStatement stmt = conn.prepareCall(
    « {? = call age(?)} »);
stmt.setString(2, « Jeanne »);
stmt.registerOutParameter(1, Types.INTEGER);
stmt.execute();
System.out.println(stmt.getInt(1));
```

- il existe
  - variantes syntaxe
  - « bloc anonymes »

  mais spécifiques à Oracle
- ci-dessus : portable (Sybase, etc.)

récapitulatif

- ordres SQL sans paramètres (schéma, instance, y compris select) : Statement
- ordres SQL avec paramètres (instance, y compris select) : PreparedStatement
- appel procédure stockée (avec ou sans paramètres) : CallableStatement

pbs MP : exécution d'ordres BD

- principe et déroulement
- ordres sans paramètre
- ordres avec paramètres
- appel procédure stockée
- SQL dynamique
- curseurs

JDBC

- le mode programme
- JDBC : qu'est-ce que c'est ? Avantages ?
- devant les machines
- interface, portabilité, connexion
- gestion erreurs BD (intuition), transactions
- exécution d'ordres BD : principe, avec/sans paramètres, curseurs, SQL dynamique
- JDBC et l'architecture en couches
- Ref cursor, gestion communication programme-serveur, métadonnées

pbs MP : SQL dynamique

- vu en PL/SQL (ex : table inconnue lors compilation)
- JDBC :
  - immédiat car ordre est une String : concaténation par + (subtilité si nombre/type attributs retour select inconnu)
  - Impossible faire autrement : JDBC ne fait que SQL dynamique
- contrepartie :
  - ordres non connus lors compilation
  - => vérification statique impossible (ex : table n'existe pas)
  - programmes moins robustes
- (vérification statique : SQLJ)

JDBC et l'architecture en couches

- même principe qu'en objet :
  - les données sont encapsulées
  - accès par procédures stockées (sauf rares cas)
- Java :
  - interagit peu avec la base :
    - appelle les procédures stockées
    - gère commit et rollback
    - si nécessaire : curseur pour affichage
  - est plutôt dédié à la partie non BD de l'application

Le type Oracle ref cursor

- Import oracle.jdbc.driver.*;
- Une fonction stockée peut
  - Remplir un curseur
  - Renvoyer un pointeur sur ce curseur : type ref cursor
- Ce pointeur se récupère en JDBC :

```
CallableStatement cstmt = conn.prepareCall("{ ? = call f() }");
...registerOutParameters(1, OracleType, CURSOR)...execute...
ResultSet rset = (ResultSet) cstmt.getObject(1);
                                  // 1 : la valeur de retour de la fonction PL
. . .  // parcours normal de rset
```

JDBC

- le mode programme
- JDBC : qu'est-ce que c'est ? Avantages ?
- devant les machines
- interface, portabilité, connexion
- gestion erreurs BD (intuition), transactions
- exécution d'ordres BD : principe, avec/sans paramètres, curseurs, SQL dynamique
- JDBC et l'architecture en couches
- Ref cursor, gestion communication programme-serveur, métadonnées