Programmation d'application Bases de données avec Java

INT

1

Plan du document

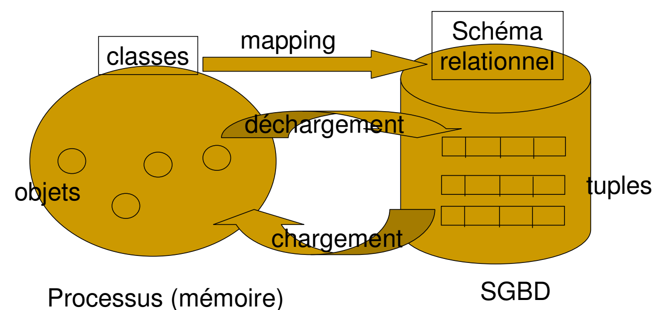
- Problématique slide 3
- Correspondance diagramme de classe UML vers schéma relationnel slide 4
- Programmation BD avec JDBC slide 16
- Gestion de la persistance en Java slide 35

- Ressources en ligne

  http://mica/~oracle/IO21

2

Problématique



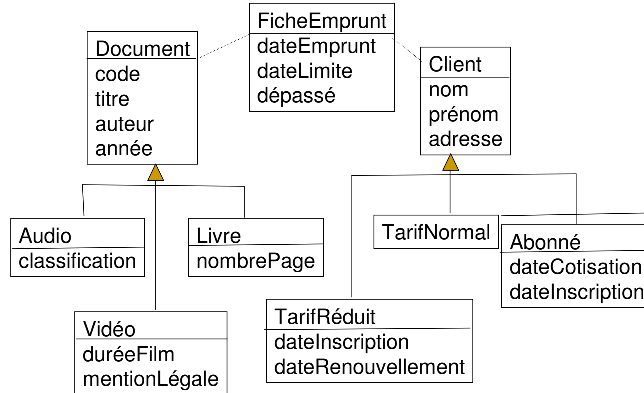
3

Correspondance de schémas

- Transformer un diagramme de classes UML (partie statique) en un schéma relationnel «équivalent»
  - Analogue à la transformation d'un schéma Entité-Association vers un schéma relationnel avec le problème des hiérarchies de classes en plus
- Plusieurs correspondances possibles

4

Exemple



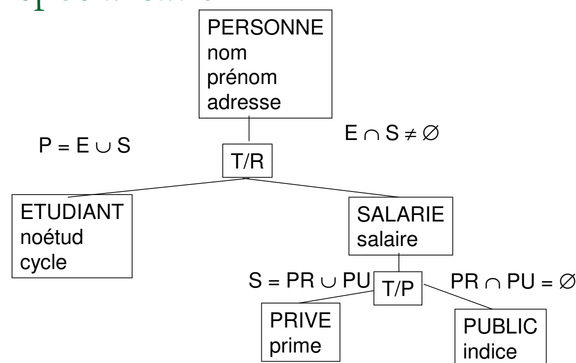
5

Sémantique de la généralisation / spécialisation

- Sous ensemble d'entité hérite des attributs de son super ensemble d'entité
- Spécialisation peut être :
  - totale (toute instance est spécialisée dans au moins un sous-ensemble) ou partielle
  - une partition (une instance ne peut être spécialisée dans plusieurs sous-ensembles) ou un recouvrement

6

Exemple de généralisation / spécialisation



7

Correspondance 1

- Une classe = une relation
- La liaison entre les relations se fait via la clé
- Fonctionne quelle que soit la sémantique d'héritage
- Nécessite de reconstruire les objets par jointure

8

Correspondance 1 pour Document

- Document(code, titre, auteur, année)
- Audio(code, classification)
- Vidéo(code, duréeFilm, mentionLégale)
- Livre(code, nombrePage)

Correspondance 2

- Chaque classe spécialisée = une relation
- Classe mère = une vue
- Pour sémantique partition et totale
- Évite les jointures pour reconstruire les objets
- Multiplie les relations (traduction d'associations modèle E/A)

Correspondance 2 pour Document

- Audio(code, titre, auteur, année, classification)
- Vidéo(code, titre, auteur, année, duréeFilm, mentionLégale)
- Livre(code, titre, auteur, année, nombrePage)
- Create View Document
  As Select code, titre, auteur, année from Audio
  Union Select code, titre, auteur, année from Vidéo
  Union Select code, titre, auteur, année from Livre
- 9 relations pour la classe FicheEmprunt !

Correspondance 3

- Ensemble des classes de la hiérarchie = une seule relation
- Éventuellement chaque classe = une vue
- Introduit un attribut discriminant et des valeurs nulles
- Pour sémantique de partition
- Évite les jointures

Correspondance 3 pour Document

- Document(code, titre, auteur, année, typeDocument, classification, duréeFilm, mentionLégale, nombrePage)
- Create view Audio as
  Select code, titre, auteur, année, classification
  From Document
  Where typeDocument='Audio'

13

Correspondance 4

- Variante de la correspondance 3 avec utilisation d'un attribut booléen supplémentaire par sous classe
- Pour sémantique de recouvrement

14

Correspondance 4 pour Document

- Document(code, titre, auteur, année, AudioB, classification, VidéoB, duréeFilm, mentionLégale, LivreB, nombrePage)
- Create view Audio as
  Select code, titre, auteur, année, classification
  From Document
  Where AudioB is true

15

JDBC

- API Java pour manipuler des relations via SQL (dans des fichiers locaux ou via un SGBD)
- Une seule API uniforme (même niveau que SQL CLI de X/open)
- Indépendance / SGBD cible (via des pilotes)
- Code portable de bout en bout
- Pas forcément construit au dessus de ODBC

16

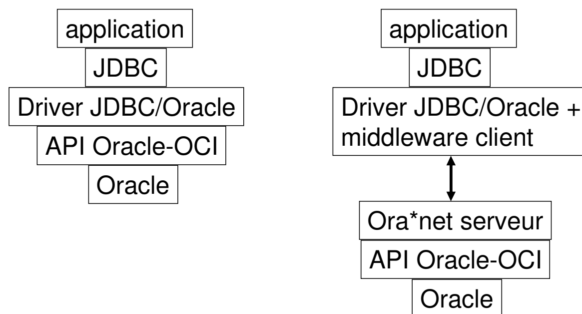
Interfaces SQL ⇔LPG

Approche	Java	Autres langages
Embedded SQL	SQLJ	Pro*C
SQL/CLI	JDBC	ODBC
PSM	PL/SQL ou Java	

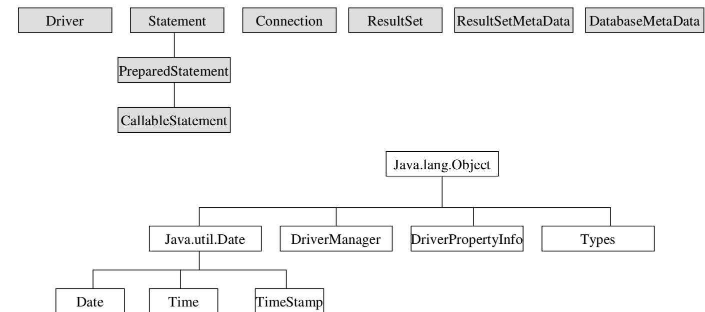
Pilotes JDBC

- JDBC non supporté en natif par les SGBD du commerce (ou les systèmes de fichiers)
- Transformations des appels JDBC en appels natifs
- Un pilote pour chaque SGBD
- 4 catégories de pilotes en fonctions de :
  - La présence ou non de pilote SGBD (non java) sur le client
  - Protocole de communication entre le client Java et le serveur

Les pilotes (suite)



Classes et interfaces du "paquage"java.sql



Connexion JDBC

- Classe java.sql.Connection
- URL d'une source de données
  jdbc:<subprotocol>:<subname>

```
private final String URLDBORACLE=
   "jdbc:oracle:thin:@tanna:1521:TPIO";
private final String URLDBMYSQL=
   "jdbc:mysql//localhost/mediatheque";
```

Connexion à la base (suite)

```
Class.forName("oracle.jdbc.driver.OracleDriver");
   chargement dynamique de la classe implémentant le pilote Oracle
String dburl = "jdbc:oracle:thin:@tanna:1521:TPIO";
   construction de l'url pour Oracle INT
Connection conn = DriverManager.getConnection(dburl, "toto", "titi");
   connexion à l'url avec un (user, passwd)=(toto, titi)
```

Création d'un *Statement*

- Un objet Statement symbolise une instruction SQL
- 3 types de *statement* :
  - Statement : requêtes simples
  - PreparedStatement : requêtes précompilées
  - CallableStatement : procédures stockées
- Création d'un *Statement* :

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

Exécution d'une requête (1/2)

- 3 types d'exécutions :
  - `executeQuery` : pour les requêtes qui retournent un `ensemble (SELECT)`
  - `executeUpdate` : pour les requêtes `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE` et `DROP TABLE`
  - `execute` : pour quelques cas rares (procédures stockées)

Exécution d'une requête (2/2)

- Exécution de la requête :

```
String myQuery = "SELECT code, titre, auteur " +
                 "FROM Document " +
                 "WHERE (auteur LIKE 'L%') " +
                 "ORDER BY titre";

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

Récupération des résultats (1/2)

- `executeQuery()` renvoie un `ResultSet`
- Le RS se parcourt itérativement *ligne* par *ligne*
- Les colonnes sont référencées par leur numéro ou par leur nom
- L'accès aux valeurs des colonnes se fait par les méthodes
  - `getXXX()` où `xxx` représente le type de l'objet
  - ou bien par un `getObject` suivi d'une conversion explicite
- Pour les types longs, on peut utiliser des *streams*.

Récupération des résultats (2/2)

```
java.sql.Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT code, titre,
   auteur from Document where auteur like 'L%'");
while (rs.next())
{
   // print the values for the current row.
   int i = rs.getInt("code");
   String s1 = rs.getString("titre");
   String s2 = rs.getString("auteur");
   System.out.println("ROW = " + i + " " + s1 + " " +s2);
}
```

Exemple de SQL dynamique

```
class Document { ...
public static int updateDocument(int num, String
   nom) {
   try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      String dburl ="jdbc:oracle:thin:@tanna:1521:TPIO";
      Connection conn = DriverManager.getConnection(dburl,
         "toto", "titi");
      conn.setAutoCommit(false);
      PreparedStatement pstmt = conn.preparedStatement("UPDATE
         Document SET auteur=? WHERE code=?");
      pstmt.setString(1, nom);
      pstmt.setInt(2, num);
      int nbLignesModifiees = pstmt.executeUpdate();
      if (nbLignesModifiees == 1) conn.commit();
      else conn.rollback();
   } catch (Exception e) {e.printStackTrace();}
}
```

28

Accès aux méta-données

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

Exemple de MetaData

```
class HTMLResultSet {
   private ResultSet rs;
   public HTMLResultSet (ResultSet rs){this.rs=rs;}
   public String toString() {
      StringBuffer out = new StringBuffer();
      out.append("<TABLE>");
      ResultSetMetaData rsmd = rs.getMetaData();
      int numcols = rsmd.getColumnCount();
      out.append("<TR>");
      for (int i=0;i<numcols;i++) {
         out.append("<TH>").append(rsmd.getColumnName(i));
      }
      out.append("</TR>");
      ...
   }
}
```


Exemple d'interrogation JDBC

```
import java.sql.*;
class TestDoc {
   public static void main (String args []) throws SQLException,
      ClassNotFoundException  {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection conn = DriverManager.getConnection
         ("jdbc:oracle:thin:@tanna:1521:TPIO", "toto", "titi");
      Statement stmt = conn.createStatement();
      ResultSet rset = stmt.executeQuery("select code, titre from
         Document");
      while (rset.next()) {
         System.out.print (rset.getInt(1));
         System.out.print ("     ");
         System.out.println (rset.getString(2));
      }
   }
}
```


Exemple d'insertion JDBC

```
import java.sql.*;
class InsereDocument{
   public static void main (String args [])throws SQLException,
      ClassNotFoundException {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection conn =
         DriverManager.getConnection("jdbc:oracle:thin:@tanna:152
         1:TPIO", "toto", "titi");
      int codeDoc = 100;
      String titreDoc = "un long dimanche de fiancailles";
      String auteurDoc =  "Japrisot";
      Statement stmt = conn.createStatement();
      int res = stmt.executeUpdate ("INSERT INTO Document(code,
         titre, auteur) VALUES(" + codeDoc + ", ' " + titreDoc +
         "', '" + auteurDoc + "') ");
      if (res == 1) {
         System.out.println("\nInsertion reussie\n");
         conn.commit();
      }else conn.rollback();
   }
}
```

Correspondance de type SQL/Java

Type SQL	Type Java	Type Java retourné par getObject()	Méthode recommandée au lieu de getObject()
numeric	Java.Math.BigDecimal	Java.Math.BigDecimal	Java.Math.BigDecimal GetBigDecimal()
integer	int	Integer	Integer getInt()
float	double	Double	Double getDoubel()
char, varchar	String	String	String getString()
date	Java.sql.Date	Java.sql.Date	Java.sql.Date getDate()

33

Evolution de JDBC

- JDBC 1.0 (janvier 1996)
- JDBC 2.0 (Mars 1998)
- JDBC 2.1 (2000)
- Extensions de JDBC 2.x
  - parcours avant/arrière d'un ResultSet
  - mise à jour depuis un ResultSet
  - batch de plusieurs ordres SQL
  - support des types SQL3
  - validation 2 phases via XA
  - notion de DataSource et utilisation de JNDI
  - pool de connexion, cache sur le client

34

Introduction de la persistance

- Comment intégrer le code JDBC qui va assurer chargement et déchargement des objets Java depuis la BD
  - En structurant au mieux le code (limiter le nombre d'opérations à modifier)
  - Quel(s) objet(s) charger et quand ?
  - quel(s) objet(s) décharger et quand ?

35

Structuration du code

- Constructeur : chargement d'objet
- « destructeur » : suppression d'objet
- Opérations de mise à jour : modification d'objets (attention à la granularité pour les transactions)

36

Chargement / déchargement

- Chargement
  - Statique : constructeur de « collections »
  - Dynamique : constructeur d'objet (nécessité d'une clé pour sélectionner l'objet)
- Déchargement
  - Statique : à la fin du programme (attention aux pertes d'infos)
  - Dynamique : dans les opérations de mise à jour

37

Chargement / déchargement (2)

- Statique
  - Adapté aux petits volumes de données
  - Faible taux de mises à jour
  - Simple à mettre en œuvre
- Dynamique
  - Adapté aux volumes de données importants
  - Fort taux de mises à jour
  - Meilleure résistance aux fautes
  - Plus compliqué à programmer

38