Mapping Objet/Relationnel

I. Mougenot (isabelle.mougenot@umontpellier.fr)

Faculté des Sciences Université Montpellier

2020

Les limites du relationnel

1. Première forme normale (1FN) : contorsions en terme de modélisation pénalisantes. Un seul type de données (table), qui correspond à un ensemble de tuples. Le tuple est construit comme une concaténation de valeurs de types primitifs (numérique, alphanumérique, date . . .).
2. la normalisation éclate les objets complexes du monde réel en de multiples relations et oblige à des jointures pour recomposer l'information.

```
e1.department().manager().salary()

select e2.salary from emp e1, emp e2 where e1.n_sup=e2.num;
```

Relationnel et programmation externe

1. la distorsion des langages (impedance mismatch) : les LDD et LMD (déclaratifs et orientés ensemble) versus les langages de programmation *dit hôtes* (impératifs et orientés élements) présentent des incompatibilités qui nécessitent des artifices de programmation (curseurs par exemple).
2. Les langages de programmation sont souvent plus complets en ce qui concerne la prise en charge de la complexité des objets du monde considéré.

Difficultés de mise en place de couches applicatives implantées autour des SGBD relationnels

Notion de type de données abstrait

ADT ou TAD = ensemble de fonctions qui cache la représentation d'un objet et contraint les interactions de l'objet avec les autres objets. Un ADT est naturellement implanté sous forme de classe (collection d'objets ayant mêmes structure et opérations). Les objets sont donc des instances d'ADT. Les SGBDOO (orientés objet) comme les SGBDOR (objet relationnel) vont tirer parti des ADT

1. SGBDOO Db4O, Objectivity (http://www.objectivity.com/), Poet Software, Matisse, EyeDB (http://www.eyedb.org/), ...
2. SGBDOR Oracle, PostgreSQL et DB2 admettent une surcouche objet qui les range dans la catégorie SGBDOR

Exemple de ADT dans le contexte d'Oracle (SQL3)

```
 CREATE TYPE proprietaire_t AS OBJECT
(nomP char(20),
adresseP varchar(20))
/
 CREATE TABLE client OF proprietaire_t
(constraint client_pk primary key (nomP));
```

Patron Data Access Object

Patron de conception pour la persistance des objets . Un ensemble de classes intermédiaires (suffixées par DAO) vont venir s'articuler entre les classes Java définies sans traitements spécifiques (POJOs [a]) et les tables de la base de données

[a] Plain Object Java Objects

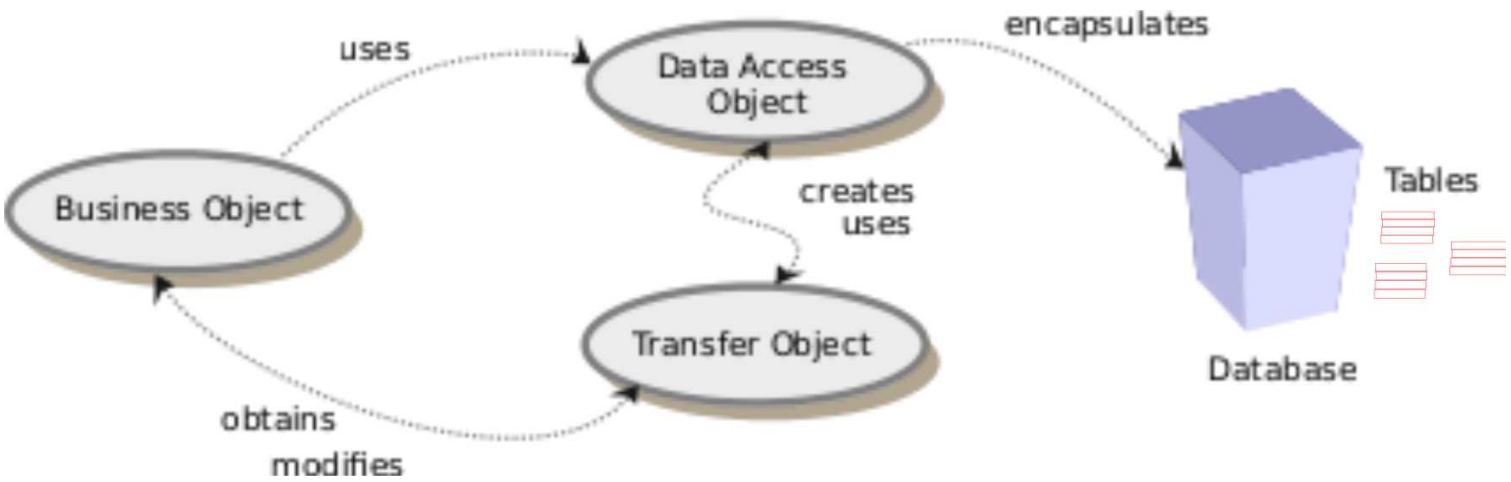

Figure: Patron de conception DAO

Patron Data Access Object

Ces classes *DAO* vont prendre en charge les méthodes CRUD (Create, Retrieve, Update, Delete) = prendre en charge les opérations de consultation, d'insertion et de mise à jour permises sur les tuples des tables de la bd

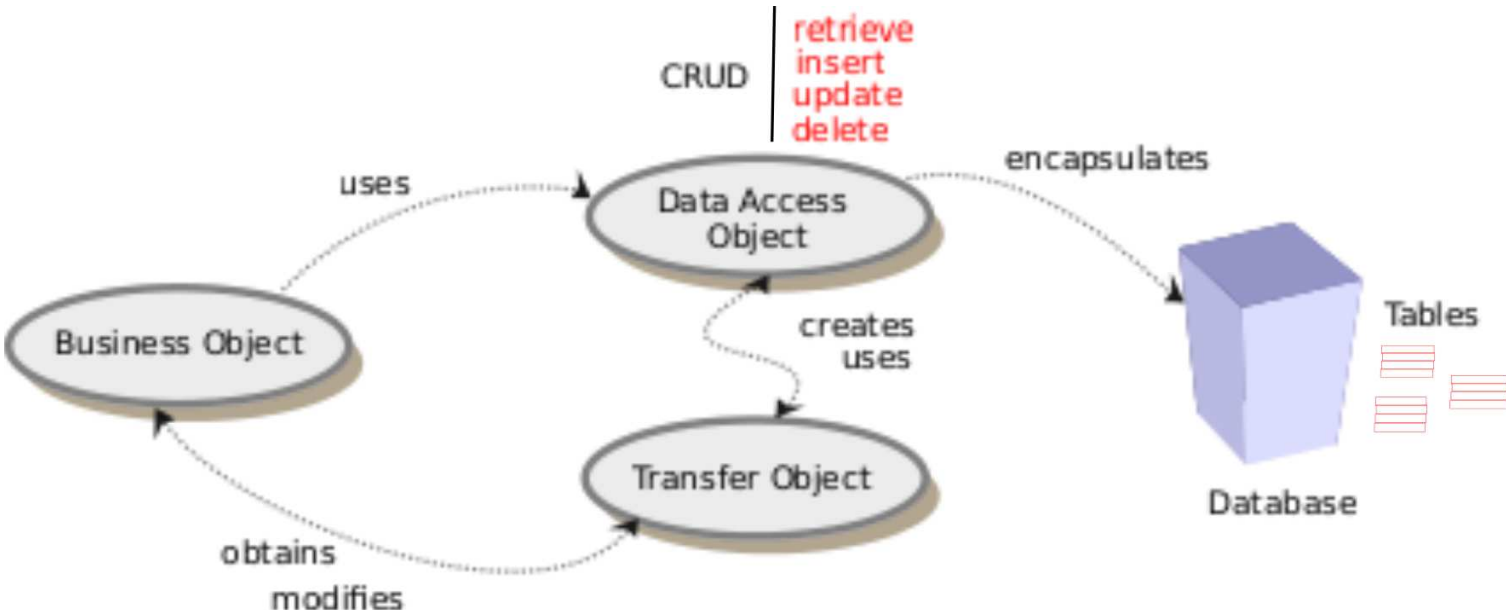

Figure: Patron de conception DAO

Patron Data Access Object

Diagramme de classes simplifié

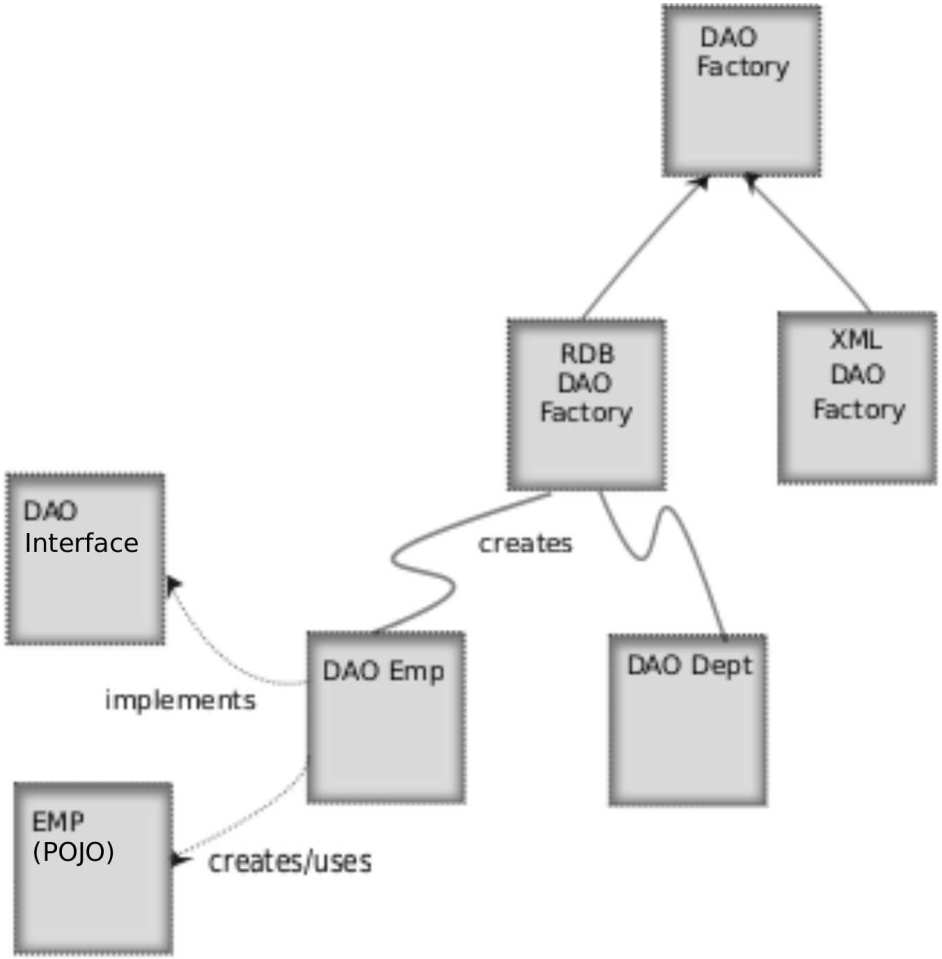

Figure: Diagramme de classes DAO

Classes POJO EMP et DEPT

Diagramme de classes descriptives

Figure: Diagramme de classes du modèle exemple

La plus grosse difficulté : information provenant de la recomposition de tables

- EMP (num, nom, fonction, n_sup, embauche, salaire, comm, n_dept)
- DEPT (n_dept, nom, lieu, budget)

référence sur la valeur de l'attribut clé versus référence sur l'objet

Classe DAO Employé : ordres CRUD pris en charge

- put/create
- get (retrieveOne)
- getWithFilter (retrieveSome)
- list (retrieveAll)
- delete
- truncate (deleteAll)
- update

Classe DAO Employé : exemples d'ordres pris en charge

```
import java.sql.*;

public class EmployeDAO {

final private String create = "insert into EMP (nom, num,
    fonction,"
+ "n_sup, embauche, salaire, comm, n_dept)"
            + " values(?, ?, ?, ?, ?, ?, ?, ?)";

final private String update = "update EMP set nom = ?,"
              + " fonction = ?, n_sup = ?, embauche = ?,"
            + " salaire = ?, comm = ?, n_dept = ? where num =
               ? ";

final private String delete = "delete from EMP where num = ? ";
```

Listing 1: Quelques opérations DAO

Classe DAO Employé (portion 2) : construction des méthodes

```
public Employe create(int num,..., Date embauche, Departement
    dept)
          throws SQLException, Exception
             {Connection c = null;
           try {c = getConnection();
            PreparedStatement ps = c.prepareStatement(create);
                  ps.setInt(1, num);
                   ...
                  ps.setDate(7, embauche);
                  ps.setInt(8, dept.getN_dept());
                  ps.executeUpdate();
                   c.commit(); }
          catch (SQLException e)
                   { c.rollback(); System.err.println("Erreur
                       createEmploye!"); return null; }
           finally{ c.close();}
return new Employe(num,nom, fonction, n_sup, salaire, comm,
    embauche, dept);}
```

Classe DAO Employé (portion 3) : construction des méthodes

```
public ArrayList<Employe> retrieveAll()
throws SQLException, Exception
   {Connection c = null;
   ArrayList<Employe> collection = new ArrayList<Employe>();
   try {
   c = getConnection();
   Statement pst = c.createStatement();
   ResultSet rset = pst.executeQuery(retrieveAll);
      while (rset.next()) {
      Employe e = new Employe(rset.getInt("num"), rset
  .getString("nom"), rset.getString("fonction"), rset
                .getInt("n_sup"),...
                rset.getDate("embauche"), new
                    Departement(rset.getInt("n_dept")));
         collection.add(e);
      }}
   ...catch et finally
   return collection;}
```

Gestion de l'accès à la BD (extension jdbc)

```
private static Connection getConnection() throws Exception {
        OracleDataSource ods = new OracleDataSource();
        ods.setDriverType("thin");
        ods.setServerName("serverName");
        ods.setDatabaseName("dbName");
        ods.setNetworkProtocol("tcp");
        ods.setPortNumber(1521);
        ods.setUser("id");
        ods.setPassword("mdp");
        return ods.getConnection();
    }
```

Listing 4: Connexion

Classe Test

```
EmployeDAO ed = new EmployeDAO();

ArrayList<Employe> v = ed.retrieveAll();

for (Employe e : v) {
System.out.println(e);
      }
Departement d = new Departement(10);
Employe emp = ed.create(22222, "Hessel", "sociologue",
    24533, 5000, 0, new Date(50), d);
d.addEmploye(emp);
```

Listing 5: Test d'utilisation

Solutions ORM pour Java

Tirer parti dans des applications distribuées de données pouvant provenir de diverses sources de données. Par principe, il s'agit aussi de prendre en charge les évolutions et les extensions possibles des applications.

Les solutions assurant la persistance des données au sein d'applications distribuées sont multiples :

- JDBC / DAO / DbUtils
- CMP (Container-Managed persistance) EJB
- frameworks ORM : Hibernate, TopLink, OpenJPA, KODO
- Architectures Orientées Services (SOA)

Framework Hibernate

Adossement à un modèle de métadonnées qui va guider la correspondance entre les classes Java et les tables de la base de données.
Dans le contexte d'Hibernate, ce modèle va être exprimé :

1. soit au travers de descripteurs XML
2. soit au travers d'annotations Java 5. Hibernate s'appuie sur la librairie standard JPA (Java persistance Api)

Dans les deux cas, Hibernate propose une gamme de fonctionnalités suffisamment riches pour prendre par exemple, en charge la persistance de plusieurs entités POJOs dans une seule table.