ORM

Object Relational Mappings

ou

Comment stocker des objets dans une BD relationnelle

Emmanuel Coquery

emmanuel.coquery@liris.cnrs.fr

http://liris.cnrs.fr/~ecoquery/

Persistance et objets

- Comment faire pour:
  - Utiliser des objets au niveau métier
  - Pouvoir sauver et récupérer des objets
  - Mettre à jour des objets de manière persistante

- Utiliser un SGBD pour stocker les informations
- Ecrire du code pour:
  - Sauver un objet dans la BD
  - Récupérer un objet à partir des données de la BD

Exemple

```
private String nom;
private String prenom;

public void save(Connection conn) throws SQLException {
    PreparedStatement stat =
            conn.prepareStatement(
                    "INSERT into personne(nom,prenom) "
                    + "VALUES (?,?)");
    stat.setString(1,nom);
    stat.setString(2,prenom);
    stat.executeUpdate();
}

public void update(Connection conn) throws SQLException {
    PreparedStatement stat =
            conn.prepareStatement(
                    "UPDATE personne SET prenom=? "
                   +"WHERE nom=?");
    stat.setString(1, prenom);
    stat.setString(2,nom);
    stat.executeUpdate();
}
```

Exemple - suite

```
    public static Personne getByNom(Connection conn, String nom) throws SQLException {
        PreparedStatement stat =
                conn.prepareStatement("SELECT nom,prenom FROM personne"
                                      +" WHERE nom=?");
        stat.setString(1, nom);
        ResultSet rs = stat.executeQuery();
        if (rs.next()) {
            return new Personne(rs.getString(1),rs.getString(2));
        } else {
            return null;
        }
    }

}
```

Avantages/Inconvénients

- Avantages:
  - Bonne maîtrise de ce qui se passe
  - Bonnes performances
- Inconvénients
  - Code lourd à maintenir
  - Certaines fonctionnalités sont pénibles à implémenter (transactions, caches, parcours d'un graphe d'objets, ...)
  - Pas de langage de haut niveau pour interroger les objets stockés

Idée: cadre applicatif dédié

- Attaquer le problème dans sa généralité
- Limiter la quantité de code à écrire
- Proposer des optimisations (e.g. caches)
- Fournir un langage de haut niveau
- Encapsuler les interactions avec la source de données
  - Pouvoir changer ainsément la source de données

Implémentations en Java

- Java Persistence API (JPA)
  - Annotations d'objets
  - Gestionnaire d'entités
- Fournisseurs (implémentations) de JPA
  - Hibernate (JBoss)
  - TopLink (Oracle)
  - OpenJPA (Apache)

JPA

- Un moyen déclaratif de décrire comment les objets sont stockés dans la BD
  - avec de bonnes valeurs par défaut
- Interfaces implémentées par les fournisseurs pour:
  - Gérer la persistance des objets
  - Récupérer des objets
    - via un langage de haut niveau
    - sous forme aisément exploitable (e.g. listes)

Correspondance objet/relationnel

- Cas simple:
  - Une classe ↔ une table
  - Un objet ↔ un n-uplet dans une table
  - Un champ ↔ un attribut
    - type SQL ↔ type Java
  - Important: définir une clé primaire
    - identifier un attribut (voir un groupe d'attributs) servant d'identifiant
    - possibilité de laisser la gestion de l'identifiant au cadre applicatif (autoincrement, sequences)
  - Certains champs ne sont pas destinés à être sauvés: marqués "transcients"

Exemple

```
package orm1;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Etudiant {

    @Id
    private long numEtu;
    private String nom;
    private String prenom;

    public Etudiant() {}

    public Etudiant(long num, String nom, String prenom) {
        setNumEtu(numEtu);
        setNom(nom);
        setPrenom(prenom);
    }

    public String getNom() {
        return nom;
    }
```

Exemple - suite

```
EntityManager em =
      Persistence.createEntityManagerFactory("nomUnitePersistance")
      .createEntityManager();
Etudiant etu = em.find(Etudiant.class, nEtudiant);
em.getTransaction().begin();
etu.setNom("Toto");
em.persist(etu);
em.getTransaction().commit();
```

- Table:
  - Etudiant(numEtu integer,
    nom varchar(255),
    prenom varchar(255),
    numEtu PRIMARY KEY)

Collections de valeurs

- Champ contient une collection de valeurs
  - Ensemble, liste, etc...
- Création d'une table pour ce champ
  - Ajout éventuel d'un index (pour les tableaux)
  - Spécification d'un ordre éventuel (pour les OrderedSets)

Héritage

- Plusieurs stratégies pour représenter une classe qui hérite d'une autre:
  - Une table par hiérarchie
  - Une table par classe avec jointure
  - Une table par classe concrète

Héritage: table par hiérarchie

- Une seule table pour toute une hiérarchie
  - Les attributs spécifiques à certaines classes sont inutilisés dans les n-uplets des objets qui sont des instances d'autres classes
  - Un attribut spécial permet de distinguer une instance d'une classe des instances d'un autre classe.

Héritage: une table par sous-classe

- Chaque classe a sa propre table
  - Seules les valeurs pour les champs spécifiques à la sous-classe sont stockés dans la table
  - Les champs des super-classes sont stockées dans leur propre table
  - Pour reconstituer toutes les valeurs liées à un objet: jointure avec les tables de toutes les super-classes
  - Un système de clés étrangères permet de faire la jointure

Héritage: table par classe concrète

- Une table par classe concrète
  - Similaire au cas sans héritage
  - Un objet est sauvé dans une table en fonction de sa classe réelle
  - Il est possible de récupérer les instances d'une classe et de ses sous-classes via une union des résultats pour chaque classe
    - Nécessite un systèmes d'identifiants cohérents

Associations entre classes

- Associations de type 1-1 (OneToOne)
  - Clé étrangère correspondant au champ indiquant l'objet associé
- Associations de type 1-n (OneToMany) et n-1 (ManyToOne)
  - Clé étrangère dans la table côté "Many"
  - Collection dans la classe côté "One"
- Associations de type n-n (ManyToMany)
  - Table intermédiaire
  - Stratégie utilisable pour 1-1, 1-n, n-1

Bidirectionnalité

- Cas où une association est représentée par un champ dans chacune des deux classes
  - Permet de "naviguer" entre les objets dans les deux directions
- Une des classes est privilégiée:
  - les modifications faites dans cette classe seront prises en compte
  - les modifications dans l'autre classe
    - doivent être cohérentes avec celles de la première classe
    - ne sont pas utilisées pour le stockage

Champs composés

- Champ contenant un objet qui n'est pas une entité en soit:
  - Identifiants complexes
  - Objets n'ayant pas de signification hors de leur objet "contenant"
    - ex: un nom (nom de famille + prénom)
- Ensembles d'attributs dans la table de l'objet "contenant"
- Fonctionne avec les collections de tels objets
  - en créant une table dédiée, comme pour les types simples

Gestionnaire de persistance

- Gère les connections à la BD
- Génère les requêtes SQL pour sauver/ modifier/lire les données des objets
  - Génère au besoin des identifiants pour les objets nouvellement sauvegardés
  - Fourni des méthodes de récupération d'objet
    - via l'identifiant
    - via des critères simples (↔ condition WHERE simple en SQL)
- Gère les transactions

Récupération et sauvegarde paresseuses

- Si la BD contient un nombre importants d'objets liés les uns aux autres
  - Ne pas tout charger d'un coup
    - ex: Elèves ↔ UEs ↔ Enseignants
  - Charger à la demande les objets associés
    - Implémentation dédiée des APIs de collections
  - Peut parfois être moins efficace
    - Multiplication des requêtes
- La sauvegarde peut également être paresseuse
  - On précise les associations conduisant à des sauvegardes automatiques

Graphes d'objets

- Le gestionnaire de persistance garde trace des objets gérés
  - Ne charge pas deux fois un objet
    - Important pour les mappings n-n
    - Ex: Eleves ↔ UEs, ne pas charger deux fois la même UE pour deux élèves (l'objet UE doit être partagé)
  - Les identifiants sont importants dans ce cadre
  - Optimise les accès à la base
  - Sauvegarde d'un graphe d'objet sans duplication ni boucle

Attachement/détachement

- Notion liée aux transactions/sessions
  - Attaché: lié à une session
    - modifications sauvegardées
  - Détaché: non lié à une session
    - modifications non sauvées
  - Nouveau: non lié à une session
    - Pas d'identifiant

Interroger des données objets

- Langage proche du SQL, mais portant sur des objets au lieu des n-uplets
- Le SELECT peut retourner des objets
- La jointure peut être utilisée pour parcourir le graphe des objets
  - Elle peut être implicaite en cas d'utilisation directe des champs
- Possibilités d'aggrégations (GROUP BY)
- Possibilités de tri (ORDER BY)

Exemple

```
SELECT etu, ue.titre
FROM Etudiant as etu left join etu.ues as ue
WHERE ue.parcours = "M1IF"
OR ue.parcours = "ISTIL2AINFO"
```

(étudiants et ues en M1INFO ou en 2eme année ISTIL INFO)

```
SELECT etu
FROM Etudiant as etu join fetch etu.ues
```

(chargement des UEs avec les étudiants)

Références

- http://docs.jboss.org/hibernate/stable/core/reference/en
- http://www.jcp.org/en/jsr/detail?id=220
- http://java.sun.com/javaee/5/docs/api/