Applications orientées données (NSY135)

9 – Règles avancées de mapping

Auteurs: Raphaël Fournier-S'niehotta et Philippe Rigaux
(philippe.rigaux@cnam.fr,fournier@cnam.fr)

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Introduction

- Nous abordons dans ce chapitre quelques aspects avancés des règles de **mapping**, et notamment la gestion de **l'héritage** orienté-objet.
- On est ici au point le plus divergent des représentations relationnelles et OO, puisque l'héritage n'existe pas pas dans la première, alors qu'il est au coeur des modélisations avancés dans la seconde.
- Il n'est pas très fréquent d'avoir à gérer une situation impliquant de l'héritage dans une base de données.
- Un des cas les plus courants est celui où on effectue la modélisation objet d'une application dont on souhaite rendre les données persistantes.
- Dans tous les cas les indications qui suivent vous montrent comme réaliser très simplement avec JPA la persistance d'une hiérarchie de classes Java.

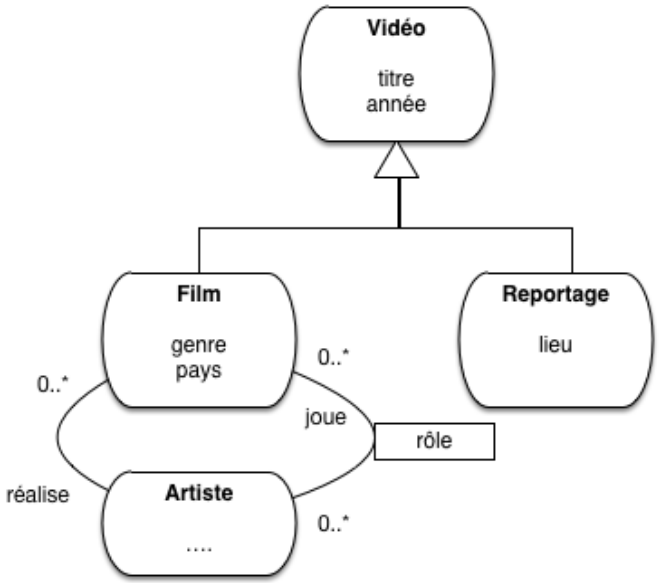
Plan du cours

1 Gestion de l'héritage
- Les solutions possibles
- Application: une table pour chaque classe
- Implantation JPA/Hibernate

2 Résumé: savoir et retenir

Héritage

- Nous prenons comme exemple illustratif le cas très simple d'un raffinement de notre modèle de données.
- La notion plus générale de **vidéo** est introduite, et un film devient un cas particulier de vidéo, comme le **reportage**
- Dans la super-classe, on trouve le titre et l'année. Un film se distingue par l'association à des acteurs et un metteur en scène; un reportage en revanche a un lieu de tournage et une date.
- Cas assez simplifié, mais suffisant pour étudier nos règles de **mapping**.



Solutions possibles

- Comme indiqué ci-dessus, il n'existe pas dans le modèle relationnel de notion d'héritage (d'ailleurs la notion d'objet en général est inconnue).
- Il faut donc trouver un contournement.
- Voici les trois solutions possibles :
  - **Une table pour chaque classe**. C'est la solution la plus directe, menant pour notre exemple à créer des tables **Vidéo**, **Film** et **Reportage**. Remarque très importante: on doit **dupliquer** dans la table d'une sous-classe les attributs persistants de la super-classe. Le titre et l'année doivent donc être dupliqués dans, respectivement, **Film** et **Reportage**. Cela donne des tables indépendantes, chaque objet étant complètement représenté par une seule ligne.
  - Remarque annexe: si on considère que **Vidéo** est une classe abstraite qui ne peut être instantiée directement, on ne crée pas de table **Vidéo**.

Solutions possibles

- Comme indiqué ci-dessus, il n'existe pas dans le modèle relationnel de notion d'héritage (d'ailleurs la notion d'objet en général est inconnue).
- Il faut donc trouver un contournement.
- Voici les trois solutions possibles :
  - **Une seule table pour toute la hiérarchie d'héritage**. On créerait donc une table **Vidéo**, et on y placerait tous les attributs persistants de **toutes** les sous-classes. La table **Vidéo** aurait donc un attribut **id_realisateur** (venant de **Film**), et un attribut **lieu** (venant de **Reportage**).
  - Les instances de **Vidéo**, **Film** et **Reportage** sont dans ce cas toutes stockées dans la même table **Vidéo**, ce qui nécessite l'ajout d'un attribut, dit **discriminateur**, pour savoir à quelle classe précise correspondent les données stockées dans une ligne de la table. L'inconvénient évident, surtout en cas de hiérarchie complexe, est d'obtenir une table fourre-tout contenant des données difficilement compréhensibles.

Solutions possibles

- Comme indiqué ci-dessus, il n'existe pas dans le modèle relationnel de notion d'héritage (d'ailleurs la notion d'objet en général est inconnue).
- Il faut donc trouver un contournement.
- Voici les trois solutions possibles :
  - Enfin, la troisième solution est un mixte des deux précédentes, consistant à créer une table par classe (donc, trois tables pour notre exemple), tout en gardant la spécialisation propre au modèle d'héritage: chaque table ne contient que les attributs venant de la classe à laquelle elle correspond, et une **jointure** permet de reconstituer l'information complète.
  - Par exemple: un film serait représenté partiellement (pour le titre et l'année) dans la table **Vidéo**, et partiellement (pour les données qui lui sont spécifiques, comme **id_realisateur**) dans la table **Film**.

Discussion des solutions

- Aucune solution n'est totalement satisfaisante.
- La duplication introduite par la première solution semble source de problèmes à terme, elle est plutôt déconseillée.
- Tout changement dans la super-classe devrait être répliqué dans toutes les sous-classes, ce qui donne un schéma douteux et peu contrôlable.
- Tout placer dans une même table se défend, et présente l'avantage de meilleures performances puisqu'il n'y a pas de jointure à effectuer.
- On risque de se retrouver en revanche avec une table dont la structure est peu compréhensible.
- Enfin la troisième solution (table reflétant exactement chaque classe de la hiérarchie, avec jointure(s) pour reconstituer l'information) est la plus séduisante intellectuellement.
- Il n'y a pas de redondance, et il est facile d'ajouter de nouvelles sous-classes.
- L'inconvénient principal est la nécessité d'effectuer autant de jointures qu'il existe de niveaux dans la hiérarchie des classes pour reconstituer un objet.

Approche proposée

- Nous aurons donc trois tables :
  - **Video (id_video, titre, annee)**
  - **Film (id_video, genre, pays, id_realisateur)**
  - **Reportage(id_video, lieu)**

- Les identifiants sont nommés **id_video** afin de mettre en évidence une contrainte : **comme un même objet est représenté dans les lignes de plusieurs tables, son identifiant est une valeur de clé primaire commune à ces lignes**
- exemple (un film et un reportage)

id_video	titre	année
1	Gravity	2013
2	Messner, profession alpiniste	2014

- Rien n'indique dans cette table quelle est la catégorie particulière des objets représentés.
- C'est conforme à l'approche objet : selon le point de vue on peut très bien se contenter de voir les objets comme instances de la super-classe. De fait, **Gravity** et **Messner** sont toutes deux des vidéos.

Tables Film et reportage

- Table **Film** avec la description de **Gravity** spécifique à sa nature de film

id_video	genre	pays	id_realisateur
1	Science-fiction	USA	59

- L'identifiant de **Gravity** (la valeur de **id**) est le **même** que pour la ligne contenant le titre et l'année dans **Vidéo** : il s'agit du même objet.
- Dans **Film**, **id** est **à la fois** la clé primaire, et une clé étrangère référençant une ligne de la table **Video**.
- On voit facilement quelle requête SQL permet de reconstituer l'ensemble des informations de l'objet :

```
1    SELECT * FROM Video AS v, Film AS f WHERE v.id=f.id AND titre='Gravity'
```

- C'est cette requête qui est engendrée par Hibernate quand l'objet doit être instancié.
- Avec cette approche, l'information relative à un même objet est donc éparpillée entre différentes tables.
- Particularité : la clé primaire d'une table pour une sous-classe est **aussi** clé étrangère référençant une ligne dans la table représentant la super-classe.

Création avec MySQL

- Voici les commandes de création des tables sous MySQL (nous nommons la seconde **FilmV** pour éviter la collision avec la table existante dans notre schéma).

```
1     CREATE TABLE Video (id_video INT AUTO_INCREMENT,
2             titre VARCHAR(255) NOT NULL,
3             annee INT NOT NULL,
4             PRIMARY KEY (id_video)
5            );
6    CREATE TABLE FilmV (id_video INT,
7             genre VARCHAR(40) NOT NULL,
8             pays VARCHAR(40) NOT NULL,
9             PRIMARY KEY (id_video),
10            id_realisateur INT NOT NULL,
11            FOREIGN KEY(id_realisateur) REFERENCES Artiste(id),
12            FOREIGN KEY (id_video) REFERENCES Video(id_video)
13           );
14   CREATE TABLE Reportage (id_video INT,
15            lieu VARCHAR(40) NOT NULL,
16            PRIMARY KEY (id_video),
17            FOREIGN KEY (id_video) REFERENCES Video(id_video)
18           );
```

- La clé primaire de la classe racine est en **AUTO_INCREMENT**, mais pas celles des autres classes. Un seul identifiant doit en effet être généré, pour la table-racine, et propagé aux autres.

Annotations dans la super-classe

- Avec le schéma qui précède, il est facile d'annoter les classes du modèle Java

```
package modeles.webscope;
import javax.persistence.*;

@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Video {

  @Id
  @GeneratedValue
  @Column(name = "id_video")
  private Long idVideo;

  @Column
  private String titre;

  public void setTitre(String t) {
    titre = t;
  }

  public String getTitre() { return titre; }

  @Column
  private Integer annee;
  public void setAnnee(Integer a) { annee = a; }
  public Integer getAnnee() { return annee; }

  public Video() {  }
}
```

La nouvelle annotation **@Inheritance** indique que la hiérarchie d'héritage dont cette classe est la racine est évaluée par **jointure**.

FilmV (squelette)

```java
package modeles.webscope;

import javax.persistence.*;

@Entity
@PrimaryKeyJoinColumn(name="id_video")
public class FilmV extends Video {
  @ManyToOne
  @JoinColumn(name = "id_realisateur")
  private Artiste realisateur;
  public void setRealisateur(Artiste a) {
    realisateur = a;
  }

  public Artiste getRealisateur() {
    return realisateur;
  }
}
```

@PrimaryKeyJoinColumn spécifie que **id_video** est à la fois clé primaire et clé étrangère pour la résolution de la jointure qui reconstitue un objet de la classe.

Plan du cours

1 Gestion de l'héritage
- Les solutions possibles
- Application: une table pour chaque classe
- Implantation JPA/Hibernate

2 Résumé: savoir et retenir

Résumé

- Le **mapping** JPA/Hibernate des situations d'héritage objet est relativement simple,
- le seul "problème" étant le choix de la structure de la base de données.
- Plusieurs solutions sont possibles, aucune n'étant idéale.
- Nous avons développé la plus lisible
- Retenez la particularité de la représentation relationnelle dans ce cas: la clé primaire des tables, à l'exception de celle représentant la classe-racine, est **aussi** clé étrangère.
- Il est important de spécifier cette contrainte dans le schéma pour assurer la cohérence des données.