Applications orientées données (NSY135)

11 – Le langage HQL

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

Plan du cours

Introduction

- HQL est un langage de requêtes **objet** qui sert à interroger le graphe (virtuel au départ) des objets java constituant la vue orientée-objet de la base.
- On interroge **un ensemble d'objets java** liés par des associations, et pas directement la base relationnelle qui permet de les matérialiser.
- Hibernate se charge d'effectuer les requêtes SQL pour matérialiser la partie du graphe qui satisfait la requête.
- Cela donne une syntaxe et une interprétation parfois différente de SQL, même si les deux langages sont en apparence très proches.
- Attention : Contrairement à un "vrai" langage d'interrogation orienté-objet, HQL ne permet pas d'appeler les méthodes des classes du modèle.

La clause **from**

- La forme la plus basique de requête HQL comprend la clause “from“, et indique la classe dont l'extension est sujet de la recherche :

```
from Film
```

- en cas de hiérarchie d'héritage, il est possible d'interroger les super- et sous-classes.

```
from Video
```

- et la liste des films (sous-classe de “Video“) par :

```
from FilmV
```

La clause **from** (2)

- On peut rechercher tous les objets persistants avec :

```
from java.lang.Object
```

- On peut créer des **alias** pour désigner les objets
- on peut ensuite leur appliquer des traitements de sélection ou projection

```
select film.titre
from Film as film
where film.annee < 2000
```

- Le mot-clé “as“ est optionnel (comme la clause “select“), mais on le garde dans un souci de clarté

♠ Préfixer un attribut par l'alias n'est indispensable que s'il y a risque d'ambiguité, ce qui n'est pas le cas dans l'exemple ci-dessus.

Clause **where**

- Les opérateurs de comparaison sont très proches de ceux de SQL :
  `= < > between in not between`
- Les opérateurs “like“ et “not like“ s'utilisent comme en SQL et expriment la même chose.
- Les opérateurs logiques sont les mêmes qu'en SQL: “and“, “or“ et “not“, avec ordre de priorité défini par le parenthésage.
- il y a aussi une clause **order by** :

```
from Film as film
where (titre like '%er%' and annee between 1970 and 2000)
or film.genre='Drame'
order by film.titre
```

Clause **where**

- Attention à la valeur “NULL“ ou “null“.
- Comme en SQL, elle indique **l'absence** de valeur.
- On la teste avec l'expression “is null“, comme en SQL:

```
from Film as film
where  film.resume is null
```

- il existe beaucoup d'autres fonctions pré-définies qui peuvent être appliquées aux propriétés des objets (ex : size(), minElement(), trunc()) (cf doc)

Jointures implicites

- Contrairement à SQL où les critères de sélection ne peuvent concerner que les propriétés de la table interrogée, HQL permet la navigation vers des objets associés (et donc vers d'autres tables), avec une syntaxe abrégée et intuitive.
- On peut par exemple chercher les films dirigés par Clint Eastwood:

```
from Film as film
where film.realisateur.nom = 'Eastwood'
```

- Pas de magie ici : cette syntaxe nécessite une **jointure** au niveau de la requête SQL générée par Hibernate
- cela correspond à une manière implicite d'exprimer une jointure
- voici la requête générée :

```
select *
from Film film0_ cross join Artiste artiste1_
where film0_.id_realisateur=artiste1_.id
and artiste1_.nom='Eastwood'
```

Jointures implicites (2)

- Cela ne fonctionne **que** pour les associations *many-to-one* et *one-to-one*.
- Vous ne pouvez pas écrire :

```
from Film as film
where film.roles.acteur.nom = 'Eastwood'
```

- On peut bien entendu exprimer **explicitement** des jointures, avec des options assez riches comme on le verra dans la section suivante.