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

3 Les jointures
- Jointures HQL et jointures SQL
- Le rôle de la clause “select“
- La clause “join“ de HQL

Jointures HQL et jointures SQL

- Jusqu'à présent, nous prenons les objets d'une classe, ce qui correspond donc à une recherche de ligne dans une seule table pour les instancier.
- Comme en SQL, il est nécessaire d'exprimer des **jointures** si l'on veut exprimer des critères de recherche sur les objets associés.
- Comme en SQL, une jointure s'exprime avec le "join" :
  - en HQL, on exprime des jointures par **navigation** en exploitant les associations d'objets définies dans le **mapping**,
  - en SQL, on applique (conceptuellement) le produit cartésien de deux tables, et on restreint le résultat à des combinaisons de lignes satisfaisant certains critères.
- **une jointure en HQL qui, en apparence, accède directement aux objets associés, est en fait implantée par Hibernate comme une jointure SQL**

Jointures HQL et jointures SQL (suite)

- Exemple : on veut tous les films mis en scène par Clint Eastwood :

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

- On accède donc directement à l'objet associé "realisateur", sans mention explicite de la classe "Artiste".
- En sous-main, Hibernate engendre et exécute la requête SQL suivante.

```
select film.*
from Film film0_  inner join  Artiste artiste1_
on film0_.id_realisateur=artiste1_.id
where artiste1_.nom='Eastwood'
```

- La version SQL comprend une contrainte d'égalité entre la clé primaire de "Artiste" et la clé étrangère dans "Film", contrainte qui n'apparaît pas en HQL puisqu'elle est représentée par une association entre objets dans le graphe virtuel.
- La définition du **mapping** permet à Hibernate d'engendrer le code SQL correct.

Jointures (suite)

- On peut adopter en HQL une expression plus proche de la logique SQL, mais moins naturelle du point de vue d'une logique "graphe d'objet".

```
select film
from Film as film,  Artiste as a
where film.realisateur = a
and a.nom='Eastwood'
```

- **Le code SQL engendré est le même**.
- Il n'y a donc aucune différence en termes de performance ou d'expressivité.

Jointures (suite)

- C'est une autre spécificité "objet" de HQL : on compare directement **l'identité** de deux objets (un artiste, un réalisateur) avec la clause :

```
where film.realisateur = a
```

- et pas, comme en SQL, **l'égalité** des clés primaires de ces objets.
- Mais il est possible d'effectuer directement une jointure sur les clés en HQL :

```
select film
from Film as film,  Artiste as a
where film.realisateur.id = a.id
and a.nom='Eastwood'
```

- Ici, "id" est un mot-clé représentant la clé primaire, quel que soit son nom réel.

La clause “select“

- Autre exemple de jointure, avec une association un à plusieurs. On sélectionne tous les films dont un des rôles est **McClane** :

```
select distinct film
from Film as film
join film.roles as role
where role.nom= 'McClane'
```

- La clause “select“ prend ici une importance particulière : dans la mesure où nous accédons aux extensions de deux classes (“Film“ et “Role“), il est préférable de préciser quels sont les objets que nous plaçons dans le résultat.
- Sans clause “select“, on obtient une liste de paires d’objets (un film, un rôle), soit une instance de “List<Object[]>“.
- Pour chaque élément de la liste, le premier composant du tableau est un “Film“, le second un “Rôle“.
- Le film est dupliqué autant de fois qu’il a de rôles associés, ce qui rend indispensable l’utilisation du mot-clé “distinct“.

La clause “select“

- Executez par exemple la requête suivante.

```
select film.titre, role.nom
from Film as film
join film.roles as role
```

- On obtient autant de fois le titre du film qu'il y a de rôles.
- Un tel résultat est assez laborieux à traiter, et on perd en grande partie les avantages de HQL (à savoir instancier un résultat directement exploitable).
- Dernier exemple, tous les films où Clint Eastwood joue un rôle :

```
select distinct film
from Film as film
join film.roles as role
join role.pk.acteur as acteur
where acteur.nom= 'Eastwood'
```

- Notez le chemin “role.pk.acteur“, exprimant un **chemin** dans le graphe des objets et composants. Je vous invite à consulter la requête SQL engendrée par Hibernate pour cette jointure.

La clause “join” de HQL

- Les jointures de HQL sont les mêmes que celles du SQL ANSI.
  - “[inner] join“ : exprime une jointure standard pour impliquer (pas forcément matérialiser) les objets associés;
  - “left [outer] join“ : exprime une **jointure externe à gauche** : si par exemple un film n’a pas de metteur en scène, il (le film) fera quand même partie du résultat, avec un objet “realisateur“ associé à “null“;
  - “right [outer] join“ est la complémentaire de “left join“ (et a donc peu d’intérêt en pratique).
- Pour bien voir la différence, exécutez les deux requêtes suivantes :

```
from Film as film
inner join film.roles as role
```

- et

```
from Film as film
left outer join film.roles as role
```

- La seconde devrait vous ramener **aussi** les films pour lesquels aucun rôle n’est connu.
- Cette requête renvoie une instance de “List<Object[]>“, qu’il faut ensuite décoder, pour obtenir, respectivement, des films et des rôles.

La clause “join” de HQL

- si, à partir de l’un des films sélectionnés, on veut accéder aux rôles par la collection “film.roles“, Hibernate a-t-il déjà instancié cette collection, ou va-t-il devoir à nouveau accéder à ces rôles avec une requête SQL ?

La clause “join” de HQL

- si, à partir de l’un des films sélectionnés, on veut accéder aux rôles par la collection “film.roles“, Hibernate a-t-il déjà instancié cette collection, ou va-t-il devoir à nouveau accéder à ces rôles avec une requête SQL ?
- Réponse : une nouvelle requête sera effectuée pour trouver les rôles du film, ce qui est dommage car la jointure SQL sera effectuée deux fois.
- **Les requêtes HQL que nous présentons de créent pas de graphe, mais instancient simplement les objets sélectionnés, sans lien entre eux.**
- Dit autrement : le fait d’effectuer une jointure entre un film et des rôles **n’implique pas** que le graphe **film-rôles** est matérialisé dans le cache de premier niveau.

La clause “join” de HQL

- Ce n’est pas si illogique que cela en a l’air, et cela rend possible d’appliquer des **restrictions** sur les objets sélectionnés, comme dans l’exemple ci-dessous.

```
from Film as film
left outer join film.roles as role
where role.nom='McClane'
```

- Si on matérialisait le graphe à partir des objets sélectionnés, la collection “roles“ de chaque film serait incomplète puisqu’elle ne comprendrait que ceux dont le nom est “McClane“.
- En résumé, il faut bien distinguer deux motivations différentes, et partiellement incompatibles :
  - la **sélection** d’objets satisfaisant des critères de restriction;
  - la **matérialisation** d’une partie du sous-graphe de données.
- Les requêtes HQL que nous présentons ici correspondent à la première motivation.
- Il est possible d’utiliser HQL **aussi** pour la matérialisation du graphe de données, avec une option “fetch“ que nous étudierons dans le prochain chapitre.

La clause "join" de HQL

- Ces deux motivations peuvent être **incompatibles** : si je **restreins** avec la clause **where** les objets sélectionnés (les films avec un rôle McClane), je ne peux pas obtenir en même temps un graphe complet (tous les rôles des films, dont un des rôles est McClane)

- Pour l'instant, on se concentre sur l'utilisation de HQL pour sélectionner et matérialiser les objets qui vous intéressent, sans chercher à optimiser les performances ou éliminer les redondances de requêtes SQL engendrées.
- Cela revient à **toujours utiliser** la clause "select" pour garder les objets qui nous intéressent, et à utiliser les jointures pour exprimer des restrictions.
- Par exemple :

```
select distinct film
from Film as film
left outer join film.roles as role
where role.nom='McClane'
```

- Comme en SQL, une autre manière d'exprimer une jointure est d'utiliser des sous-requêtes, qui sont présentées un peu plus loin.