

Leçon 2 – Java Persistance Query Langage

*Cette API permet d'interroger les objets avec une sorte de SQL orienté objet.
Cela permet de simplifier de nombreux traitements.*

1 UN QUERY LANGUAGE ORIENTÉ OBJET

Grâce aux entités, il est possible de manipuler des objets sans se soucier de la base de données relationnelles qui se trouve derrière et qui assure leur persistance. On peut ainsi se passer d'accès direct à la base via JDBC. Toutefois, le gestionnaire d'entité est limité par rapport à ce qu'offre SQL. Reprenons l'exemple de la bibliothèque. Comment obtenir la liste des lecteurs qui ont un emprunt pour le moment ? Plutôt difficile à coder sans SQL.

Pour pallier à ces manques, Java EE introduit JPQL (anciennement appelé EJB-QL). JPQL est proche de SQL3 (SQL OO) mais avec ses propres contraintes. On retrouve ainsi la puissance de SQL tout en restant en OO. Voici un exemple pour consulter les lecteurs ayant droits à 5 emprunts simultanés ou plus.

```
SELECT l FROM Lecteur l WHERE l.lcNbMaxEmp1 >= 5
```

Après exécution de cette requête (nous verrons comment), on obtient une `List<Lecteur>` satisfaisant à la requête. Comparez cela à ce que vous devez faire en JDBC (exécuter la requête; parcourir le « resultset » et pour chaque occurrence, créer un objet à partir des valeurs des colonnes)

Tout comme avec JDBC, il est possible d'écrire des requêtes avec paramètre. Voici un exemple complet

```
Query query = em.createQuery(
    "SELECT l FROM Lecteur l WHERE l.lcNbMaxEmp >= :max");
query.setParameter("max", 5);
List<Lecteur> emprunts = query.getResultList();
```

En pratique, il faudra aussi veiller à la performance. Ainsi, on pourra faire réaliser un maximum de choses par le serveur de données (son efficacité est plus grande et on réduit les transferts sur le réseau).

¹Ou « l.nbMaxEmprunts » si vous avez eu la bonne idée de renommer cet attribut.

2 LES NAMED-QUERIES

Si vous avez lu attentivement (mais bien sûr que vous l'avez fait !) les entités générées par Netbeans, vous aurez déjà remarqué des requêtes JPQL sous l'annotation « @NamedQueries ».

```
@NamedQueries( {  
    @NamedQuery(name = "Lecteur.findByLecid",  
        query = "SELECT l FROM Lecteur l WHERE l.lecid = :lecid"),  
    @NamedQuery(name = "Lecteur.findByLecnom",  
        query = "SELECT l FROM Lecteur l WHERE l.lecnom = :lecnom"),  
    @NamedQuery(name = "Lecteur.findByLecprenom",  
        query = "SELECT l FROM Lecteur l WHERE l.lecprenom = :lecprenom")  
})
```

Cette possibilité de nommer des requêtes permet d'accroître la lisibilité du code, de ne pas dupliquer une requête utilisée à plusieurs endroits, de simplifier d'éventuelles modifications ainsi que de permettre une pré-compilation des requêtes. Pour les utiliser :

```
Query query = em.createNamedQuery("Lecteur.findByLecprenom");
```

3 EXERCICES

Reprenez le projet « bibliothèque ». Écrivez des requêtes nommées pour résoudre facilement les problèmes suivants.

Assez facile

- Affichez la liste de tous les lecteurs.
- Affichez la liste des lecteurs triée (en ascendant) sur leur prénom.
- Affichez la liste des lecteurs qui ont un emprunt pour l'instant.
- Affichez la liste des livres empruntés depuis plus d'un mois.

Un peu plus difficile

- Affichez la liste des ouvrages disponibles pour l'instant.
- Affichez les éditeurs ayant un livre en emprunt pour le moment