Fetch

adriencaubel.fr

Table des matières

Introduction

Rappels

- Nous avons étudié les concepts suivants
  - La notion `@Entity`
  - Les associations `@OneToMany` ... et `mappedBy` (bidirectionnelle)
  - L'héritage `@Inheritance`

Nous allons maintenant nous intéresser à **l'optimisation des lectures**

Pourquoi ?

Lorsqu'on travaille avec JPA, il est essentiel de comprendre comment sont chargées les relations entre entités pour optimiser les performances de l'application.

Exemple

- Un `Etudiant` a une liste de `Note`
- Lorsqu'on charge un `etudiant` doit-on également charger les `notes` ?

Stratégies Fetching

Les deux stratégies de fetching

Deux stratégies principales existent :

- **Lazy Fetching** (chargement paresseux)
- **Eager Fetching** (chargement immédiat).

Fetching par défaut

- Par défaut, `@OneToMany` et `@ManyToMany` adopte une approche **Lazy**
- Par défaut, `@OneToOne` et `@ManyToOne` adapte une approche **Eager**

```
class Etudiant {
    ...
    @OneToMany(mappedBy = "etudiant") // Lazy par défaut
    private List<Note> notes
}
```

Par défaut la liste de `notes` ne sera pas chargée en même temps de l'étudiant

Fetch LAZY

```java
@Entity
public class Etudiant {
    @OneToMany(mappedBy = "etudiant") private List<Livre> livresLus;
}
```

```java
Etudiant etudiant = entityManager.find(Etudiant.class, etudiantId); // 1
List<Livre> livres = etudiant.getLivresLus(); // 2
```

Comme LAZY deux requêtes SQL vont être nécessaires

1. `SELECT * FROM etudiant WHERE id = ?;`
2. puis `SELECT * FROM livre WHERE etudiant_id = ?;`

Fetch EAGER

```
    @OneToMany(mappedBy = "etudiant", fetch = FetchType.EAGER)
    private List<Livre> livresLus;
```

```
Etudiant etudiant = entityManager.find(Etudiant.class, etudiantId);
List<Livre> livres = etudiant.getLivresLus();
```

Une seule requête, tout est récupéré en même temps

```
SELECT e.*, l.*
FROM etudiant e
LEFT JOIN livre l ON e.id = l.etudiant_id
WHERE e.id = ?;
```

Toujours choisir EAGER ?

Et bien ça dépend.

- Dans le cas où l'on sait que la relation *livres* sera explorée systématiquement après la lecture d'un étudiant, il serait plus malin de n'émettre qu'un seul SELECT avec une jointure, de manière à peupler la relation *livres* à l'avance.
  - Cela ne ferait qu'un seul aller-retour avec la base de données.
- En revanche, dans le cas d'une relation qui, pour des raisons applicatives, ne serait pas explorée, ou rarement, alors l'exécution de la jointure lors du SELECT serait un surcoût inutile.