

L'héritage avec Jakarta Persistence

Contrôle de l'héritage avec JPA

Emmanuel BRUNO

Université de Toulon, LIS UMR CNRS 7020

emmanuel.bruno@univ-tln.fr

2026-10-01

Table des matières

- 1 Une seule relation par hiérarchie 1
- 2 Une relation par classe (strategies JOINED ou TABLE_PER_CLASS) 2
- 3 Conclusion 2

JPA supporte plusieurs stratégies d'héritage, chacune adaptée à des besoins spécifiques en termes de structure de base de données.

En utilisant l'annotation `@Inheritance` il est possible de spécifier comment les entités héritées seront stockées dans la base de données, que ce soit dans une table unique regroupant toutes les entités, des tables distinctes pour chaque classe, ou une approche intermédiaire.

```
Dot Executable: /opt/local/bin/dot
Dot executable does not exist
Cannot find Graphviz. You should try
```

```
@startuml
testdot
@enduml
```

or

```
java -jar plantuml.jar -testdot
```

Figure 1. – La hiérarchie des personnes

1 Une seule relation par hiérarchie

Par défaut JPA définit une seule relation pour une hiérarchie d'héritage. Toutes les classes de la hiérarchie sont mappées vers une seule table dans la base de données. La table contient des colonnes pour toutes les propriétés de toutes les classes de la hiérarchie. Un discriminant (généralement une colonne spécifiant le type) est utilisé pour différencier les enregistrements appartenant à différentes classes.

```
No JDBC connection available.
Set system properties
'jdbc.url' (and optionally
'jdbc.user'/'jdbc.password'), or
provide a Connection in the kernel
environment.
```

Figure 2

```
No JDBC connection available.
Set system properties
'jdbc.url' (and optionally
'jdbc.user'/'jdbc.password'), or
provide a Connection in the kernel
environment.
```

2 Une relation par classe (stratégies JOINED ou TABLE_PER_CLASS)

Mais il est aussi possible de demander une relation par classe.

Avec `@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)` chaque classe de la hiérarchie est mappée vers sa propre table dans la base de données. Les tables ne contiennent que les colonnes nécessaires pour stocker les propriétés spécifiques à chaque classe.

Avec `@Inheritance(strategy = InheritanceType.JOINED)` chaque classe de la hiérarchie est mappée vers une table distincte, et il y a également une table commune (table de la superclasse) contenant les propriétés partagées par toutes les sous-classes. Les tables des sous-classes contiennent uniquement les colonnes spécifiques à chaque classe, et une clé étrangère relie ces tables à la table commune.

table commune pour les propriétés partagées, tandis que la stratégie `TABLE_PER_CLASS` favorise l'indépendance des tables pour chaque classe.

```
@SuperBuilder
@NoArgsConstructor(access =
    AccessLevel.PROTECTED)
@Getter
@Setter
@ToString(callSuper = true)

@Entity(name="TeacherHeritage2")
@Table(name="TEACHER", schema =
    "EX_HERITAGE2")
public class Teacher extends Person {

    @Column(nullable = false)
    private double salary;
}
```

3 Conclusion

En résumé, la stratégie JOINED privilégie la normalisation des données en utilisant une