

L'HÉRITAGE AVEC *JPA*



FMS ACADEMY
MEHDIQUI Ayyoub

2023



PLAN

1. Héritage avec une *seule table*
2. Héritage avec *jointure* de tables
3. Héritage avec une *table par classe*

Introduction

JPA offre plusieurs stratégies pour associer une relation d'héritage dans le **modèle objet** avec une représentation dans le **modèle relationnel** de données.

Pour décrire la relation d'héritage avec JPA, on utilise l'annotation [@Inheritance](#) ou [@MappedSuperclass](#)



L'annotation @Inheritance

L'annotation `@Inheritance` permet d'indiquer qu'il existe une relation d'héritage dans le modèle objet et dans le modèle relationnel. Comme il existe plusieurs façon de représenter un héritage de données dans le modèle relationnel, l'annotation `@Inheritance` dispose de l'attribut `strategy` pour préciser la stratégie utilisée dans le modèle de données. Cette stratégie est une énumération du type `InheritanceType` et accepte les valeurs :

`SINGLE_TABLE`

l'héritage est représenté par une seule table en base de données

`JOINED`

l'héritage est représenté par une jointure entre la table de l'entité parente et la table de l'entité enfant

`TABLE_PER_CLASS`

l'héritage est représenté par une table par entité

Single_Table

```
package fr.epsi.b3.vehicule;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="vehicule_type")
public abstract class Vehicule {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    private String marque;

    // getters/setters omis

}
```

```
package fr.epsi.b3.vehicule;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue("V")
public class Voiture extends Vehicule {

    private int nbPlaces;

    // getters/setters omis

}
```

```
package fr.epsi.b3.vehicule;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue("M")
public class Moto extends Vehicule {

    private int cylindree;

    // getters/setters omis

}
```

Single_Table

L'annotation `@DiscriminatorValue` permet de préciser la valeur dans la colonne discriminante qui permet d'identifier un objet du type de cette classe. Cette valeur doit être unique pour l'ensemble des classes de l'héritage. Dans l'exemple ci-dessus, lors de la persistance d'un objet de la classe `Voiture`, JPA positionnera automatiquement la valeur `"V"` dans la colonne `vehicule_type`.

Pour l'exemple précédent, le schéma de base de données contiendra la table `Vehicule` qui peut être créée comme suit :

```
create table Vehicule (  
    id int(11) AUTO_INCREMENT,  
    vehicule_type varchar(1) NOT NULL,  
    marque varchar(255),  
    cylindree int,  
    nbPlaces int,  
    primary key (id)  
) engine = InnoDB;
```

Note

L'inconvénient de la stratégie `SINGLE_TABLE` est qu'il n'est pas possible d'ajouter des contraintes de type `NOT NULL` sur les colonnes représentant les propriétés des classes filles.

Conclusion

Si vous recherchez les meilleures performances et que vous avez besoin d'utiliser des requêtes et des relations **polymorphes**, choisissez la stratégie de **table unique**.

Mais sachez que vous ne pouvez pas utiliser de contraintes not nul sur les attributs de sous-classe, ce qui augmente le risque d'incohérence des données.



RESSOURCES

https://gayerie.dev/epsi-b3-orm/javaee_orm/jpa_inheritance.html

<https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/>