

TRAVAUX PRATIQUE : JPA N°2 : LES STRATÉGIES D' HÉRITAGE

INTRODUCTION

CE TP CONSISTE À APPLIQUER LES STRATÉGIES D'HÉRITAGE OFFERTE PAR LES ANNOTATION JPA.

□ TP N°1 : HÉRITAGE (STRATÉGIE : JOINED)

MAPPER UNE RELATION D'HÉRITAGE EN UTILISANT LA STRATÉGIE JOINED

□ TP N°2 : HÉRITAGE (STRATÉGIE : SINGLE_TABLE)

MAPPER UNE RELATION D'HÉRITAGE EN UTILISANT LA STRATÉGIE SINGLE_TABLE

□ TP N°3 : HÉRITAGE (STRATÉGIE : TABLE_PER_CLASS)

MAPPER UNE RELATION D'HÉRITAGE EN UTILISANT LA STRATÉGIE TABLE_PER_CLASS

LES ANNOTATIONS À MAÎTRISER.

`@Inheritance(strategy=InheritanceType.JOINED)`

`@PrimaryKeyJoinColumn(name="ID_BOOK")`

`@Inheritance(strategy=InheritanceType.SINGLE_TABLE)`

`@DiscriminatorColumn(name="COLUMN")`

`@DiscriminatorValue("VALUE")`

`@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)`

PRÉ-REQUIS

IL EST OBLIGATOIRE D'AVOIR REALISER LES TRAVAUX PRATIQUE : JPA N°1

LA STRATÉGIE JOINED

1. MODIFIER LE FICHIER META-INF/PERSISTENCE.XML POUR CRÉER LES TABLES À PARTIR DES ANNOTATIONS JPA UTILISÉES DANS LES ENTITÉS JAVA. LA PROPRIÉTÉ À AJOUTER EST LA SUIVANTE.

```
<property name="hibernate.hbm2ddl.auto" value="create" />
```

2. VÉRIFIER QUE LA CLASSE **BOOK** EST CRÉÉ DANS LE PACKAGE "MODELS" COMME DEMANDÉ DANS LE TP1

```
package model;
import javax.persistence.Column;
import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

@Entity
@Table(name = "tbooks")
public class Book {
    @Id // simple primary key
    @GeneratedValue(strategy = GenerationType.TABLE)
    private int id;
    @Column(name = "title")
    private String title;
    public Book(int id, String title) {
        super();
        this.id = id;
        this.title = title;
    }
    public Book() {
        super();
    }
    public Book(String title) {
        super();
        this.title = title;
    }
    public int getId() {
        return id;
    }
}
```

```

    }
    public void setId(int id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}

```

3. CRÉER UNE CLASSE **ROMAN** AVEC **THEME_LITTERAIRE** COMME ATTRIBUT. GÉNÉRER LES GETTERS ET LES SETTERS. CRÉER UN LIEN D'HÉRITAGE AVEC LA CLASSE MERE **BOOK**. GÉNÉRER LE CONSTRUCTEUR.

```

package model;
import javax.persistence.Entity;

@Entity
public class Roman extends Book{
    private String themeLitteraire;
    public String getThemeLitteraire() {
        return themeLitteraire;
    }
    public void setThemeLitteraire(String themeLitteraire) {
        this.themeLitteraire = themeLitteraire;
    }
    public Roman(String title, String themeLitteraire) {
        super(title);
        this.themeLitteraire = themeLitteraire;
    }
}

```

4. PARCE QUE UN LIEN D'HÉRITAGE EST CRÉÉ ENTRE **ROMAN** ET **BOOK**, IL FAUT DÉCLARER LA STRATÉGIE DE MAPPING À UTILISER. COMMENÇANT PAR LA STRATÉGIE **JOINED**. DANS LA CLASSE **BOOK** UTILISER L'ANNOTATION **@INHERITANCE(STRATEGY=INHERITANCE_TYPE.JOINED)** ET DANS LA CLASSE **ROMAN** UTILISER L'ANNOTATION **@PRIMARYKEYJOINCOLUMN(NAME="ID_BOOK")**

```

@Entity
@Table(name = "tbooks")
@Inheritance(strategy=InheritanceType.JOINED)
public class Book {
    ...
}

@Entity
@PrimaryKeyJoinColumn(name="id_book")
public class Roman extends Book{

```

```
...  
}
```

5. ESSAYER D'INSÉRER UN ROMAN ET VÉRIFIER ATTENTIVEMENT LA STRUCTURE DES TABLES CRÉÉES.
EXÉCUTER LA CLASSE DE TEST DÉJÀ CRÉÉ DANS LE Tp1.

```
package service;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import model.Roman;  
public class Test {  
    public static void main(String[] args) {  
        ApplicationContext ap= new ClassPathXmlApplicationContext("spring.xml");  
        IBookService service= (IBookService)ap.getBean("ser");  
        service.add(new Roman("titré Roman","Thème Littéraire"));  
    }  
}
```

VÉRIFIEZ LA CONSOLE ET LE SCHÉMA DE LA BASE

CONSOLE: CRÉATION D'UNE CLÉ ÉTRANGÈRE DANS LA TABLE ROMAN AVEC UNE RÉFÉRENCE SUR L'ID DE LA TABLE BOOKS

```
Hibernate:  
    create table hibernate_sequences (  
        sequence_name varchar(255) not null,  
        next_val bigint,  
        primary key (sequence_name)  
    )  
Hibernate:  
    create table Roman (  
        themeLitteraire varchar(255),  
        id_book integer not null,  
        primary key (id_book)  
    )  
Hibernate:  
    create table tbooks (  
        id integer not null,  
        title varchar(255),  
        primary key (id)  
    )  
Hibernate:  
    alter table Roman  
        add constraint FK6nq5bo2l2qmermpplocdjxn2p  
        foreign key (id_book)  
        references tbooks (id)
```

SCHÉMA:

SQL Query Area	
1	SELECT * FROM roman r;
themeLitteraire	id_book
ThemeLitteraire	1

SQL Query Area	
1	SELECT * FROM tbooks t;
id	title
1	titreRoman

CONCLUSION JOINED

LA MEILLEURE STRATÉGIE A PRIORI EST **JOINED** : C'EST LA SEULE IMPLÉMENTATION DE L'HÉRITAGE AU SENS STRICT, TANT EN JAVA QUE DANS LE MODÈLE DE DONNÉES. ELLE IMPLIQUE NÉANMOINS SYSTÉMATIQUEMENT DES JOINTURES EN LECTURE, ET POTENTIELLEMENT PLUSIEURS REQUÊTES EN ÉCRITURES. LES REQUÊTES DE RECHERCHE PEUVENT VITE DEVENIR LOURDES.

LA STRATÉGIE **SINGLE_TABLE**

1. MAINTENANT ON CHANGE LA STRATÉGIE **JOINT** PAR **SINGLE_TABLE** AU NIVEAU DE LA CLASSE **BOOK**. CELA SIGNIFIE QU'UNE SEULE TABLE SERA UTILISÉE À LA FOIS POUR LES BOOKS ET POUR LES ROMANS. CETTE SEULE TABLE DOIT CONTENIR UNE COLONNE SUPPLÉMENTAIRE POUR DISTINGUER ENTRE LES BOOKS ET LES ROMANS. CETTE COLONNE EST APPELÉE **DISCRIMINATOR**. LES CLASSES **BOOK** ET **ROMAN** DEVIENNENT ALORS:

```
@Entity
@Table(name = "tbooks")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="Column_distinguish")
@DiscriminatorValue("value_for_books")
public class Book {
    ...
}

@Entity
@DiscriminatorValue("value_for_romans")
public class Roman extends Book{
    ...
}
```

2. ESSAYER D'INSÉRER UN ROMAN ET VÉRIFIER ATTENTIVEMENT LA STRUCTURE DES TABLES CRÉÉES. EXÉCUTER LA CLASSE DE TEST DÉJÀ CRÉÉ DANS LE **Tp1**.

VÉRIFIEZ LA CONSOLE ET LE SCHÉMA DE LA BASE

CONSOLE: CRÉATION D'UNE SEULE TABLE AVEC LA COLONNE **DISCRIMINATOR**

```
Hibernate:
    create table hibernate_sequences (
        sequence_name varchar(255) not null,
        next_val bigint,
        primary key (sequence_name)
    )
Hibernate:
    create table tbooks (
        Column_distinguish varchar(31) not null,
```

```

        id integer not null,
        title varchar(255),
        themeLitteraire varchar(255),
        primary key (id)
    )
Hibernate:
    select
        tbl.next_val
    from
        hibernate_sequences tbl
    where
        tbl.sequence_name=? for update

Hibernate:
    insert
    into
        hibernate_sequences
        (sequence_name, next_val)
    values
        (?,?)
Hibernate:
    update
        hibernate_sequences
    set
        next_val=?
    where
        next_val=?
        and sequence_name=?
Hibernate:
    insert
    into
        tbooks
        (title, themeLitteraire, Column_distinguish, id)
    values
        (?, ?, 'value_for_romans', ?)

```

SCHÉMA:

1	SELECT * FROM tbooks t;			
Column_distinguish	id	title	themeLitteraire	
value_for_romans	1	titreRoman	ThemeLitteraire	

CONCLUSION SINGLE_TABLE

LA STRATÉGIE **SINGLE_TABLE** EST UNE ALTERNATIVE TRÈS HONNÊTE : LES PERFORMANCES EN LECTURE ET ÉCRITURE SONT TRÈS BONNES. EN REVANCHE, ON A POTENTIELLEMENT UN GRAND NOMBRE DE COLONNES ET BEAUCOUP DE VALEURS **NULL**.

LA STRATÉGIE **TABLE_PER_CLASS**

1. MAINTENANT ON CHANGE LA STRATÉGIE **SINGLE_TABLE** PAR **TABLE_PER_CLASS** AU NIVEAU DE LA CLASSE **BOOK**. CELA SIGNIFIE QU'UNE TABLE SERA CRÉÉE PAR CLASSE. DANS NOTRE CAS, ON AURA UNE TABLE POUR LES **BOOKS** ET UNE AUTRE POUR LES **ROMANS**. LA CLASSE **BOOK** ET LA CLASSE **ROMAN** DEVIENNENT:

```
@Entity
@Table(name = "tbooks")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Book {
    ...
}

@Entity
public class Roman extends Book{
    ...
}
```

2. ESSAYER D'INSÉRER UN **ROMAN** ET VÉRIFIER ATTENTIVEMENT LA STRUCTURE DES TABLES CRÉÉES. EXÉCUTER LA CLASSE DE TEST DÉJÀ CRÉÉ DANS LE **Tp1**.

VÉRIFIEZ LA CONSOLE ET LE SCHÉMA DE LA BASE

CONSOLE: CRÉATION D'UNE TABLE PAR CLASSE

```
Hibernate:
    create table hibernate_sequences (
        sequence_name varchar(255) not null,
        next_val bigint,
        primary key (sequence_name)
    )
Hibernate:
    create table Roman (
        id integer not null,
        title varchar(255),
        themeLitteraire varchar(255),
        primary key (id)
    )
Hibernate:
    create table tbooks (
        id integer not null,
        title varchar(255),
        primary key (id)
    )
Hibernate:
    select
        tbl.next_val
    from
        hibernate_sequences tbl
    where
```

```
tbl.sequence_name=? for update
```

Hibernate:

```
insert
into
    hibernate_sequences
    (sequence_name, next_val)
values
    (?,?)
```



Hibernate:


```
update
    hibernate_sequences
set
    next_val=?
where
    next_val=?
    and sequence_name=?
```

Hibernate:

```
insert
into
    Roman
    (title, themeLitteraire, id)
values
    (?, ?, ?)
```

SCHÉMA:

1	SELECT * FROM roman r;		
		id	title
		1	titreRoman
			themeLitteraire
			ThemeLitteraire

1	SELECT * FROM tbooks t;	
		id
		title

CONCLUSION SINGLE_TABLE

LE MODE **TABLE_PER_CLASS** EST UN PEU LE PARENT PAUVRE DE CE COMPARATIF AVEC SES ALLURES D'USINE À GAZ. IL EST VRAI QUE L'ON PERD TOUT-À-FAIT L'INTÉRÊT DE L'HÉRITAGE DANS L'APPROCHE BASE DE DONNÉES (CE QUI N'EST PAS LE CAS DU POINT DE VUE **JAVA**). ON A POTENTIELLEMENT BEAUCOUP DE DONNÉES DUPLIQUÉES, CE QUI SIGNIFIE QU'UNE MODIFICATION SUR UNE DES TABLES IMPLIQUÉES RISQUE DE DEVOIR ÊTRE RÉPERCUTÉE SUR UNE OU PLUSIEURS AUTRE(S) TABLE(S). IL PROPOSE TOUTEFOIS DES PERFORMANCES CORRECTES EN LECTURE ET EN ÉCRITURE (MÊME S'IL FAUT GÉRER « À LA MAIN » LA COHÉRENCE DES DONNÉES ENTRE LA SUPER-CLASSE ET LES SOUS-CLASSES).