Héritage

Démonstration 9 du module 5

Les objectifs de cette démonstration sont

- Tester et mettre en place les 3 stratégies d'héritage

Contexte

- Continuer dans le projet précédent
- Nouveau sujet : Gestion de véhicule.
  - Avec 2 types : Berline et les VoitureDeCourse
- Renommer le package des entités précédentes en com.
- Mettre l'annotation @Profil(«Demo») sur le bean dans la classe d'exécution

Déroulement

1. SINGLE-TABLE

- Cette stratégie permet de représenter en base de données un héritage avec une seule table.
  - Une colonne contiendra un identifiant pour déterminer le type réel de la classe : on parle de la colonne discriminante.
  - La table doit contenir toutes les colonnes nécessaires pour stocker les informations de la super classe et de l'ensemble des classes filles.
- Créer un package : fr.eni.demo.heritage.singletable
- Créer les 3 classes Voiture, Berline et VoitureDeCourse

```
package fr.eni.demo.heritage.singletable;

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;

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "DISCR")
@DiscriminatorValue(value = "V")
public class Voiture {
```

```
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Integer id;

	private String marque;

	public Voiture() {
	}

	public Voiture(String marque) {
		this.marque = marque;
	}

	public Integer getId() {
		return id;
	}

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

	public String getMarque() {
		return marque;
	}

	public void setMarque(String marque) {
		this.marque = marque;
	}

	@Override
	public String toString() {
		return "Voiture [id=" + id + ", marque=" + marque + "]";
	}

}
```

- La classe mère est la classe Voiture, elle déclare donc l'annotation d'héritage
  - @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
  - Elle précise la colonne discriminante : @DiscriminatorColumn(name = "DISCR")
  - Elle positionne une valeur dans cette colonne pour ses instance « V »
- Les 2 autres classe, hérite de celle-ci,
  - redéfinisse les constructeurs
  - ajoute l'annotation pour préciser la valeur discriminante : `@DiscriminatorValue`

```
package fr.eni.demo.heritage.singletable;

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

@Entity
@DiscriminatorValue(value = "B")
public class Berline extends Voiture {

	private String couleurCuir;

	public Berline() {
	}

	public Berline(String marque, String couleurCuir) {
		super(marque);
		this.couleurCuir = couleurCuir;
```

```
	}

	public String getCouleurCuir() {
		return couleurCuir;
	}

	public void setCouleurCuir(String couleurCuir) {
		this.couleurCuir = couleurCuir;
	}

	@Override
	public String toString() {
		return "Berline [couleurCuir=" + couleurCuir + "]";
	}

}
```

```
package fr.eni.demo.heritage.singletable;

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

@Entity
@DiscriminatorValue(value = "C")
public class VoitureDeCourse extends Voiture {
	public String ecurie;

	public VoitureDeCourse() {
	}

	public VoitureDeCourse(String marque, String ecurie) {
		super(marque);
		this.ecurie = ecurie;
	}

	public String getEcurie() {
		return ecurie;
	}

	public void setEcurie(String ecurie) {
		this.ecurie = ecurie;
	}

	@Override
	public String toString() {
		return "VoitureDeCourse [ecurie=" + ecurie + "]";
	}

}
```

- Créer un Repository sur la classe mère

```
package fr.eni.demo.heritage.singletable;

import org.springframework.data.jpa.repository.JpaRepository;

public interface VoitureHSTRepository extends JpaRepository<Voiture, Integer> {

}
```

- Dans la classe d'exécution de l'application
  - Copier le code du nouveau bean :

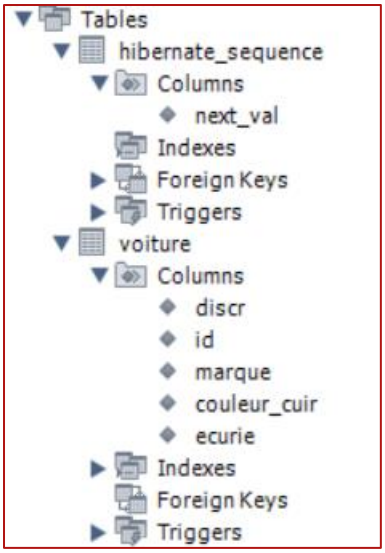
```
@Bean
	public CommandLineRunner demoHeritageSingleTable(VoitureHSTRepository voitureRepo) {
		return (args) -> {
			fr.eni.demo.heritage.singletable.Voiture clio = new
fr.eni.demo.heritage.singletable.Voiture("RenaultClio");
			fr.eni.demo.heritage.singletable.Berline bmw = new
fr.eni.demo.heritage.singletable.Berline("BMW", "Rouge");
			fr.eni.demo.heritage.singletable.VoitureDeCourse ferrari = new
fr.eni.demo.heritage.singletable.VoitureDeCourse(
					"Ferrari", "Scuderia Ferrari");

			voitureRepo.save(clio);
			voitureRepo.save(bmw);
			voitureRepo.save(ferrari);

			System.out.println("Liste des voitures : ");
			System.out.println("-------------------------------");
			for (fr.eni.demo.heritage.singletable.Voiture v : voitureRepo.findAll()) {
				System.out.println(v.toString());
			}
		};
	}
```

- Exécution, Traces attendues :

```
…
Hibernate:

    create table hibernate_sequence (
       next_val bigint
    ) engine=InnoDB
Hibernate:

    insert into hibernate_sequence values ( 1 )
Hibernate:

    create table voiture (
       discr varchar(31) not null,
        id integer not null,
        marque varchar(255),
        couleur_cuir varchar(255),
        ecurie varchar(255),
        primary key (id)
    ) engine=InnoDB
…
Hibernate:
    select
        next_val as id_val
    from
        hibernate_sequence for update

Hibernate:
    update
        hibernate_sequence
    set
        next_val= ?
    where
        next_val=?
Hibernate:
    insert
    into
        voiture
```

```
        (marque, discr, id)
    values
        (?, 'V', ?)
Hibernate:
    select
        next_val as id_val
    from
        hibernate_sequence for update

Hibernate:
    update
        hibernate_sequence
    set
        next_val= ?
    where
        next_val=?
Hibernate:
    insert
    into
        voiture
        (marque, couleur_cuir, discr, id)
    values
        (?, ?, 'B', ?)
Hibernate:
    select
        next_val as id_val
    from
        hibernate_sequence for update

Hibernate:
    update
        hibernate_sequence
    set
        next_val= ?
    where
        next_val=?
Hibernate:
    insert
    into
        voiture
        (marque, ecurie, discr, id)
    values
        (?, ?, 'C', ?)
Liste des voitures :
------------------------------
Hibernate:
    select
        voiture0_.id as id2_0_,
        voiture0_.marque as marque3_0_,
        voiture0_.couleur_cuir as couleur_4_0_,
        voiture0_.ecurie as ecurie5_0_,
        voiture0_.discr as discr1_0_
    from
        voiture voiture0_
Voiture [id=1, marque=RenaultClio]
Berline [couleurCuir=Rouge]
VoitureDeCourse [ecurie=Scuderia Ferrari]
```

- Hibernate crée
    - une table « hibernate_sequence » pour gérer les clefs.
    - Et la table voiture :



- Voici les données en base :

discr	id	marque	couleur_cuir	ecurie
V	1	RenaultClio	NULL	NULL
B	2	BMW	Rouge	NULL
C	3	Ferrari	NULL	Scuderia Ferrari

  - Il faut remarquer qu'il y a à chaque fois des colonnes à null.
  - Cela fait beaucoup d'enregistrement pour peu de données utiles

2. *TABLE_PER_CLASS :*

- Cette stratégie permet de représenter en base de données un héritage avec une table par entité.
  - Les attributs hérités sont répétés dans chaque table.
  - Ainsi, la notion d'héritage n'est pas exprimée dans le modèle relationnel de données.
- La stratégie TABLE_PER_CLASS est plus complexe à mettre en place pour la gestion des clés primaires.
  - Si on prend un héritage entre la classe abstraite Vehicule, les classes filles Voiture et Camion.
  - Pour JPA, les objets de type Voiture et Camion sont également des objets de type Vehicule.
  - À ce titre, il ne peut pas exister deux objets avec la même clé primaire. Mais, comme ces objets sont représentés en base de données par deux tables différentes, une colonne de type AUTO_INCREMENT en MySQL ne suffit pas à garantir qu'il n'existe pas une voiture ayant la même clé qu'un camion.
  - Avec, cette stratégie, il n'est pas possible d'utiliser l'annotation @GeneratedValue avec la valeur IDENTITY,
  - On peut, par exemple, utiliser une table servant à générer une séquence de clés.

- Modifier le package des entités précédentes en com
  - Et poser @Profile(«demo») sur le bean
- Créer un package : fr.eni.demo.heritage.tableperclass
- Dupliquer les 3 classes Voiture, Berline et VoitureDeCourse
- Classe Voiture
  - Modifié l'annotation d'héritage
  - Supprimer les annotations : `@DiscriminatorColumn` et `@DiscriminatorValue`
  - Ajouter un nom à l'entité et un nouveau nom à la table

```
package fr.eni.demo.heritage.tableperclass;

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(name="voiture_tpc")
@Table(name="voiture_tpc")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Voiture {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Integer id;
	private String marque;
…
```

- Dans les classes Berline et VoitureDeCourse
  - Retirer l'annotation : `@DiscriminatorValue`
- Créer un Repository pour la classe mère

```
package fr.eni.demo.heritage.tableperclass;

import org.springframework.data.jpa.repository.JpaRepository;

public interface VoitureHTPCRepository extends JpaRepository<Voiture, Integer> {

}
```

- Dans la classe d'exécution de l'application
  - Copier le code du nouveau bean :

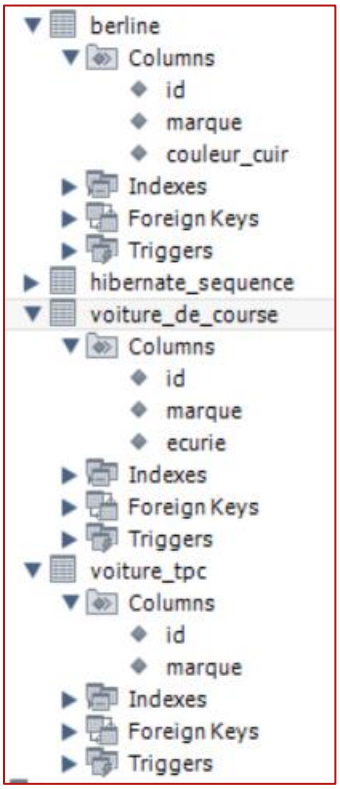
```
@Bean
public CommandLineRunner demoHeritageTablePerClass(VoitureHTPCRepository voitureRepo) {
	return (args) -> {
		fr.eni.demo.heritage.tableperclass.Voiture clio = new fr.eni.demo.heritage.tableperclass.Voiture("RenaultClio");
		fr.eni.demo.heritage.tableperclass.Berline bmw = new fr.eni.demo.heritage.tableperclass.Berline("BMW", "Rouge");
		fr.eni.demo.heritage.tableperclass.VoitureDeCourse ferrari = new fr.eni.demo.heritage.tableperclass.VoitureDeCourse(
				"Ferrari", "Scuderia Ferrari");

			voitureRepo.save(clio);
			voitureRepo.save(bmw);
			voitureRepo.save(ferrari);

			System.out.println("Liste des voitures : ");
			System.out.println("-------------------------------");
			for (fr.eni.demo.heritage.tableperclass.Voiture v : voitureRepo.findAll()) {
				System.out.println(v.toString());
			}
		};
	}
```

- Exécution, Traces attendues :

```
...
Hibernate:

    create table berline (
       id integer not null,
        marque varchar(255),
        couleur_cuir varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate:

    create table hibernate_sequence (
       next_val bigint
    ) engine=InnoDB
Hibernate:

    insert into hibernate_sequence values ( 1 )
Hibernate:

    create table voiture_tpc (
       id integer not null,
        marque varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate:

    create table voiture_de_course (
```

```
    id integer not null,
     marque varchar(255),
     ecurie varchar(255),
     primary key (id)
) engine=InnoDB
```

- La grosse différence est la création de 4 tables :
  - Celle de séquence pour gérer les clefs
  - Et une par classe



- Voici les données en base :

Voiture_tpc	Voiture_de_course	berline
id: 1, marque: RenaultClio	id: 3, marque: Ferrari, ecurie: Scuderia Ferrari	id: 2, marque: BMW, couleur_cuir: Rouge

  - Il faut remarquer qu'il n'y pas de colonne à null. Pas de perte de place dans les enregistrements
  - Par contre, 3 tables créées et les id des 3 tables sont liées
  - Donc un coup supplémentaire sur la génération des clefs primaires

3. JOINED

- Cette stratégie permet de représenter en base de données un héritage avec une table par entité.
    - Pour les classes filles, JPA réalisera une jointure entre la table représentant l'entité et la table représentant l'entité de la super classe.
    - L'implémentation est très proche de celle d'un héritage avec une seule table (seule la stratégie change) mais le schéma de la base de données est très différent.

- Modifier le package des entités précédentes en com
    - Et poser @Profile(«demo») sur le bean
- Créer un package : fr.eni.demo.heritage.joined
- Dupliquer les 3 classes de la version Single Table : Voiture, Berline et VoitureDeCourse
- Classe Voiture
    - Modifié l'annotation d'héritage
    - Ajouter un nom à l'entité et un nouveau nom à la table

```
package fr.eni.demo.heritage.joined;

import javax.persistence.*;

@Entity(name="voiture_hj")
@Table(name="voiture_hj")
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name = "DISCR")
@DiscriminatorValue(value = "V")
public class Voiture {
```

- Dans les classes Berline et VoitureDeCourse
    - Ajouter un nom à l'entité et un nouveau nom à la table

```
package fr.eni.demo.heritage.joined;

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

@Entity(name="berline_hj")
@Table(name="berline_hj")
@DiscriminatorValue(value = "B")
public class Berline extends Voiture {
```

```
package fr.eni.demo.heritage.joined;

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

@Entity(name = "voituredecourse_hj")
@Table(name = "voituredecourse_hj")
@DiscriminatorValue(value = "C")
public class VoitureDeCourse extends Voiture {
```

- Créer un Repository pour la classe mère

```
package fr.eni.demo.heritage.joined;

import org.springframework.data.jpa.repository.JpaRepository;

public interface VoitureHJRepository extends JpaRepository<Voiture, Integer> {

}
```

- Dans la classe d'exécution de l'application
  - Copier le code du nouveau bean :

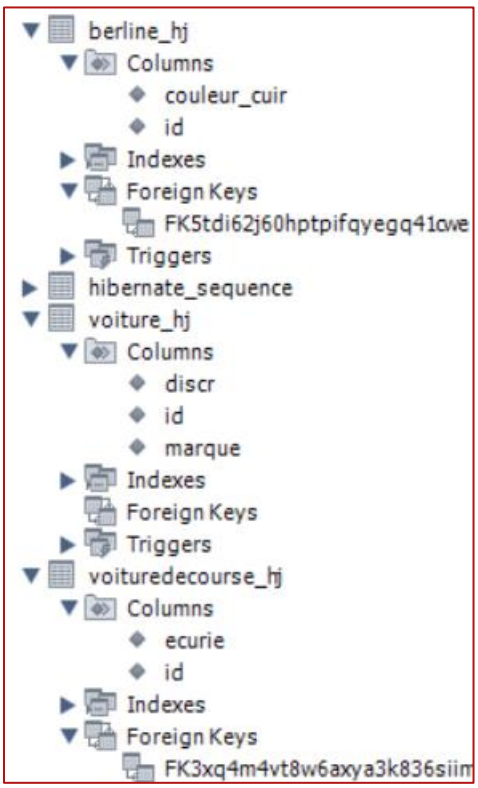
```
@Bean
	public CommandLineRunner demoHeritageJoined(VoitureHJRepository voitureRepo) {
		return (args) -> {
			fr.eni.demo.heritage.joined.Voiture clio = new fr.eni.demo.heritage.joined.Voiture("RenaultClio");
			fr.eni.demo.heritage.joined.Berline bmw = new fr.eni.demo.heritage.joined.Berline("BMW", "Rouge");
			fr.eni.demo.heritage.joined.VoitureDeCourse ferrari = new fr.eni.demo.heritage.joined.VoitureDeCourse(
					"Ferrari", "Scuderia Ferrari");

			voitureRepo.save(clio);
			voitureRepo.save(bmw);
			voitureRepo.save(ferrari);

			System.out.println("Liste des voitures : ");
			System.out.println("-------------------------------");
			for (fr.eni.demo.heritage.joined.Voiture v : voitureRepo.findAll()) {
				System.out.println(v.toString());
			}
		};
	}
```

- Exécution, Traces attendues :

```
...
Hibernate:

    create table berline_hj (
       couleur_cuir varchar(255),
        id integer not null,
        primary key (id)
    ) engine=InnoDB
Hibernate:

    create table hibernate_sequence (
       next_val bigint
    ) engine=InnoDB
Hibernate:

    insert into hibernate_sequence values ( 1 )
Hibernate:

    create table voiture_hj (
       discr varchar(31) not null,
        id integer not null,
        marque varchar(255),
        primary key (id)
    ) engine=InnoDB
Hibernate:

    create table voituredecourse_hj (
       ecurie varchar(255),
        id integer not null,
        primary key (id)
```

```
    ) engine=InnoDB
Hibernate:

    alter table berline_hj
       add constraint FK5tdi62j60hptpifqyegq41cwe
       foreign key (id)
       references voiture_hj (id)
Hibernate:

    alter table voituredecourse_hj
       add constraint FK3xq4m4vt8w6axya3k836siimp
       foreign key (id)
       references voiture_hj (id)
```

- La grosse différence est la création de clefs de jointure vers la table voiture_hj dans les 2 tables filles



- Voici les données en base :

Voiture_hj	Voituredecourse_hj	Berline_hj
discr id marque V 1 RenaultClio B 2 BMW C 3 Ferrari	ecurie id Scuderia Ferrari 3	couleur_cuir id Rouge 2

  - Il faut remarquer qu'il n'y pas de colonne à null. Pas de perte de place dans les enregistrements
  - Par contre, 3 tables créées et obligation de faire des jointures pour récupérer les données complètes des classes filles

- C'est celle la plus communément utilisée, elle fonctionne aussi bien sur une base de données existante que sur une à générer.