Persistance des données

JPA/Hibernate

Spring Data

1 Présentation
Jakarta Persistence API

- JPA est une abstraction de la couche JDBC, les classes et les annotations JPA sont dans le package javax.persistance (jakarta.persistence).

- JPA 1.0 a été introduite dans la spécification JAVA EE 5, la spécification Jakarta EE 9 supporte la version JPA 3.0

- Composants:
  - ORM: mécanisme de mapping relationnel objet
  - L'API Entity Manager qui gère les opérations CRUD
  - JPQL : langage de requête orienté objet.
  - JTA (Jakarta Transaction API) : Mécanisme de gestion des verrouillages et des transactions dans un environnement concurrent

- Frameworks de persistance JPA
  - TopLink
  - EclipseLink (framework de référence JPA, basé sur TopLink).
  - Hibernate.

2- Entity

- Une classe Entité est un Javabean annoté par @Entity
- Annotations
  - L'annotation @Table permet de modifier le nom par défaut de la table associée à l'entité @Table(name=" Tlivre").
  - @Column: pour personnaliser les attributs (par défaut le nom d'un champ dans la table est identique au nom de l'attribut)

4- Exemple

```java
package spring.cours.jpa.model;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Utilisateur {
@Id
private int id;
private String nom;

public int getId() {
return id;
}

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

public String getNom() {
return nom;
}

public void setNom(String nom) {
this.nom = nom;
}

}
```

5- Propriétés des colonnes

- @Column permet de déterminer les propriétés d'une colonne dans la table associée.
- Principales propriétés de l'annotation column :

Propriété	Valeur par défaut
name	Nom de l'attribut
nullable	True
Length (String)	255
unique	False

```
@Entity
@Table(name="users")
public class Utilisateur {
@Id
private int id;
@Column(nullable = false, name = "username", length = 60)
private String nom;
@Column(unique = true)
private String email;
// get, set
}
```

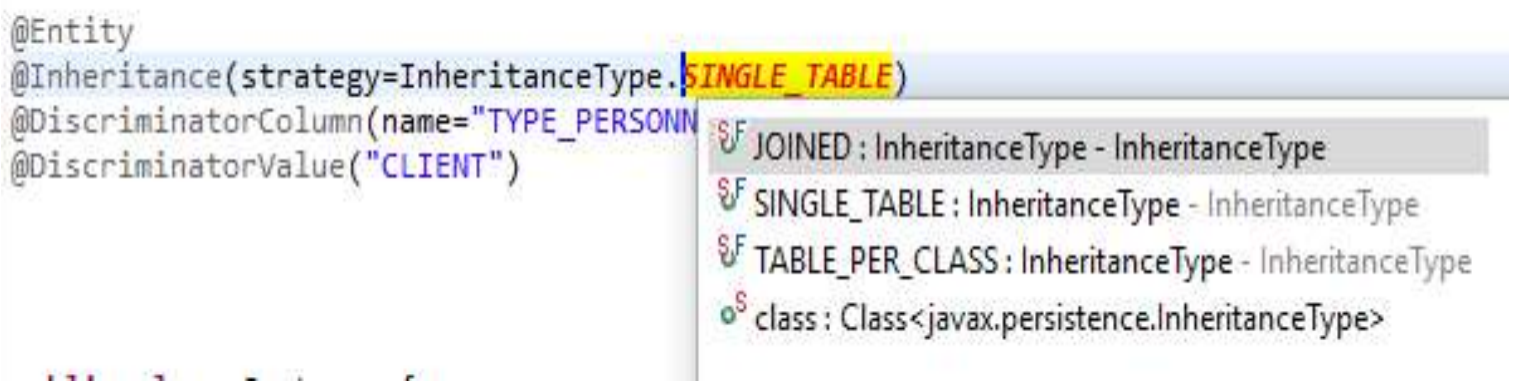
6- Auto incrément

L'annotation GeneratedValue permet de créer une clé primaire auto incrément.

- `@GeneratedValue`
- `@GeneratedValue(strategy=GenerationType.`***`AUTO`***`)`: ***le framework choisit la strégie adaptée selon le gestionnaire de base de données utilisé.***
- `@GeneratedValue(strategy = GenerationType.`***`IDENTITY`***`)`: ***type identity.***
- `@GeneratedValue(strategy = GenerationType.`***`SEQUENCE`***`)`: ***crée une séquence.***
- `@GeneratedValue(strategy = GenerationType.`***`TABLE`***`)`: ***crée une table pour conserver la dernière valeur utilisée.***

7- Héritage

- La JPA supporte trois stratégies de génération d'une relation d'héritage:
  - Une seule table (SINGLE_TABLE:valeur par défaut).
  - Une table par classe (JOINED).
  - Une table par sous classe (TABLE_PER_CLASS).
- L'annotation @Inheritance est appliquée sur la classe de base.

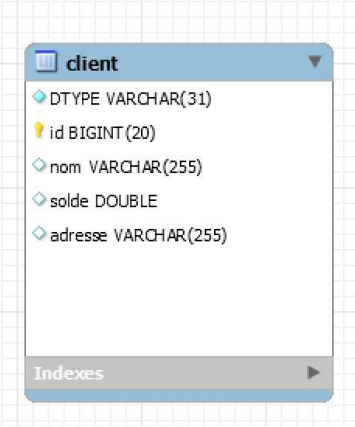


Exemple 1: une seule table

```java
@Entity
@Inheritance
public class Client {
	@Id
	@GeneratedValue
	private long id;
	private String nom;
…
}
```

```java
@Entity
public class ClientImportant
extends Client {

	private double solde;
…
}
```

```java
@Entity
public class ClientRegulier extends
Client {

	private String adresse;
…
}
```



DTYPE	id	nom	solde	adresse
Client	1	un client	NULL	NULL
ClientImportant	2	Client important 1	3999	NULL
ClientRegulier	3	Client régulier 1	NULL	1 10 rue 2

Le champ DTYPE contient le type de client (la valeur de ce champ contient par défaut le nom de la classe).

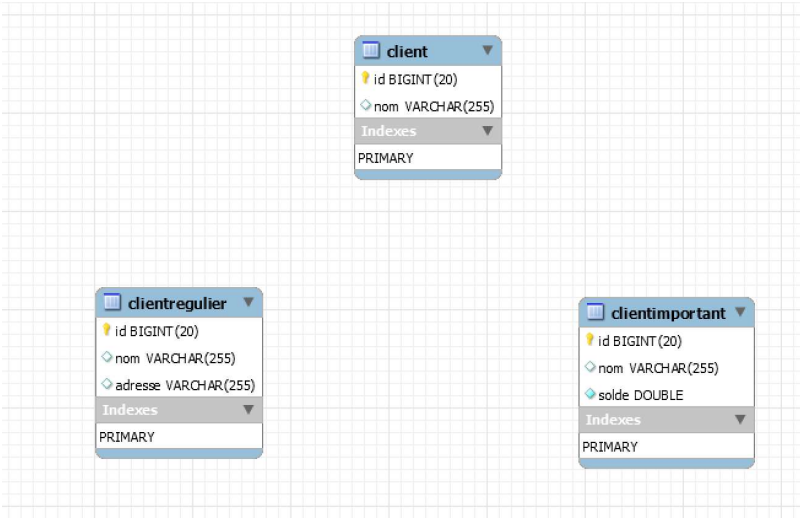
```java
@Entity

@Inheritance(strategy =
InheritanceType.SINGLE_TABLE)

@DiscriminatorColumn(name = "DTYPE",
discriminatorType = DiscriminatorType.STRING)

@DiscriminatorValue(value = "Client")

public class Client {

…

}
```

```java
@Entity

@DiscriminatorValue(value="ClientImportant")

public class ClientImportant

extends Client {

…

}
```

```java
@Entity

@DiscriminatorValue(value
= "ClientRegulier")

public class
ClientRegulier extends
Client {

…

}
```

Exemple 2: Une table par classe

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Client { …
```



Client

id	nom
1	un client

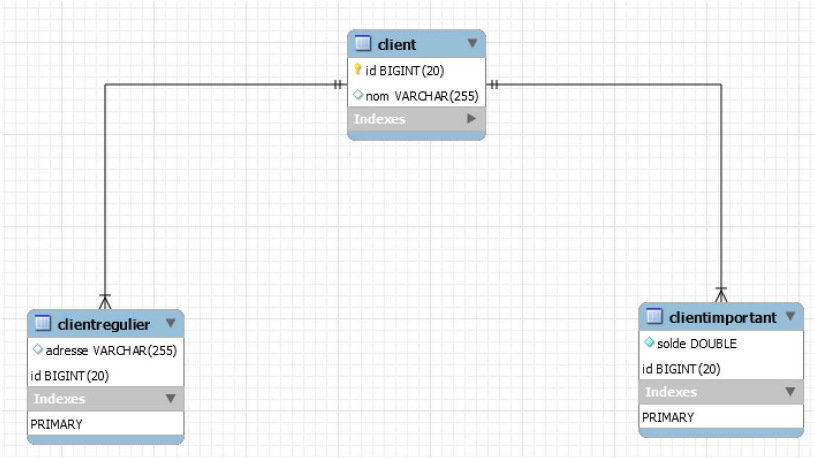
Client Important

id	nom	solde
2	Client important 1	3999

Client régulier

id	nom	adresse
3	Client régulier 1	1 10 rue 2

Exemple 3: Joined



Client

id	nom
1	un client
2	Client important 1
3	Client régulier 1

Client Important

solde	id
3999	2

Client régulier

adresse	id
1 10 rue ?	3

Hibernate DDL

La propriété spring.jpa.hibernate.ddl-auto définit le mode de génération de la base de données par Hibernate, valeurs:

- create
- create-drop: à chaque démarrage la base de données est créée puis supprimée.
- update : mise du schéma de la base de données (si c'est possibe)
- none : valeur par défaut (à utiliser en production).

Associations

La spécification JPA supporte trois types d'association :

- One to one
- Many to one / One to Many
- Many to Many

Les trois types d'association peuvent être unidirectionnelles (navigables dans un sens) ou bidirectionnelles (navigables dans les deux sens), la navigabilité impacte l'utilisation des classes dans le programme et non pas les tables générées dans la base de données.

@ManyToOne

L'association ManyToOne est la plus utilisée dans la pratique, dans la classe Produit on ajoutera un attribut de type Categorie.

Categorie	Produit

@Entity
public class Categorie {
@Id
@GeneratedValue(strategy = GenerationType.*IDENTITY*)
private long catId;
private String libelle;
…
}
	@Entity
	public class Produit {
	@Id
	@GeneratedValue(strategy=GenerationType.*IDENTITY*)
	private long reference;
	private String designation;
	private double prix;
	@ManyToOne
	private Categorie categorie;
	…
	}

@OneToMany

Navigabilité : Catalogue → Produit

Catalogue	

`@Entity`
`public class Catalogue {`
`@Id`
`@GeneratedValue(strategy = GenerationType.IDENTITY)`
`private long id;`
`private String titre;`
`@OneToMany`
`private List<Produit> produits=new ArrayList<>();`
`…}`
	Par défaut l'annotation ManyToOne génère une table association (ce qui est souhaité dans notre exemple) , pour généré une clé étrangère dans la table Produit, il faut ajouter l'annotation JoinColumn pour définir le nom de la clé étrangère dans la table produit :
	`@OneToMany`
	`@JoinColumn(name="catalogue_id")`
	`private List<Produit> produits=new ArrayList<>();`

ManyToMany

•la navigabilité Produit → Magasin

```
@ManyToMany(mappedBy = "produits")
private List<Magasin> magasins=new ArrayList<>();
```

Ce qui permettra d'accéder aux magasins à partir d'un produit p.getMagasins()

JPQL

JPQL (Java Persistence Query Language) est un langage d'interrogation orienté objet similaire au langage SQL,
Structure d'une requête

Exemple de requêtes:

- SELECT li From Livre li
- SELECT li From Livre li Where li.titre='UML'
- SELECT li.titre, li.Editeur from livre li
- SELECT c.Adresse.pays from client c

Syntaxe d'une requête:

- SELECT
- FROM
- [WHERE]
- [ORDER BY]
- [GROUP BY]
- [HAVING]

Sélection selon une condition (JPA 2.0)

- SELECT CASE l.editeur WHEN 'Eyrolles"
- THEN l.prix * 0.5
- ELSE l.prix * 0.8
- END
- FROM Livre l

SQL	JPQL
SELECT * FROM produit WHERE prix > 100 ORDER BY designation;	**SELECT p** **FROM** Produit p **WHERE p.prix** > 100 **ORDER BY** p.designation