D’UN POJO À UNE ENTITY BEAN

I. UN PEU DE VOCABULAIRE

POJO

Plain Old Java Object (classe « classique ») - détaillés en partie II.

ENTITY BEAN

POJO ordinaire auquel on ajoute des annotations afin de le lier à une table spécifique.

HIBERNATE

ORM (outil) qui permet de lier une classe JAVA à une table contenue dans une base de données spécifiée

ANNOTATION

Elément précédé du symbole @ qui permet de définir l’élément qui suit l’annotation. Par exemple, @Column permet de lier l’attribut qui suit à une colonne de la base de données.

II. LES POJO

Un POJO est une classe qui normalement n'hérite d'aucune autre classe, n'implémente aucune interface et ne contient aucune annotation liée à l'utilisation d'un framework.

Voici un exemple concret :

```java
public class Question {
    private Long id;
    private String libelle;
    private Niveau niveau;
    private Theme theme;
    private Proposition bonneReponse;
    private List<Proposition> propositions;

    public Question(Long pId, String pLibQue, Niveau pNivQue, Theme pTheme, Proposition pBonRep) {
        this.id = pId;
        this.libelle = pLibQue;
        this.niveau = pNivQue;
        this.theme = pTheme;
        this.bonneReponse = pBonRep;
    }

    public Long getId() { return this.id; }

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

    public String getLibelle() { return this.libelle; }

    public void setLibelle(String pLibQue) { this.libelle = pLibQue; }

    public Niveau getNiveau() { return this.niveau; }

    public void setNiveau(Niveau pNivQue) { this.niveau = pNivQue; }

    public Theme getTheme() { return this.theme; }

    public void setTheme(Theme pTheme) { this.theme = pTheme; }

    public Proposition getBonneReponse() { return this.bonneReponse; }

    public void setBonneReponse(Proposition pBonRep) { this.bonneReponse = pBonRep; }
}
```

Le POJO ci-dessus Question est composé de 3 attributs, un constructeur et des accesseurs (getters et setters).
Nous ne reviendrons pas sur ces concepts puisqu'ils sont acquis.

III. VERS LES ENTITIES

Ce qui change concrètement : les **annotations** issues de la classe `javax.persistence.*`.

En effet, chaque élément créé (une classe, un attribut) doit être annoté afin qu'Hibernante puisse le relier à la base de données définie dans le fichier de configuration de l'application.

A. DÉCLARATION DE L'ENTITY

Ainsi, la déclaration de la classe se fait à présent comme ceci :

`@Entity` pour déclarer que ce POJO devient une entité

`@Table(name="nomDeLaTableEnBdd")` pour indiquer le nom de la table qui correspond à ce POJO en base de données.

```
@Entity
@Table(name="question")
public class Question {
```

B. DÉCLARATION DES ATTRIBUTS SIMPLES

Est appelé attribut simple tout attribut contenant une donnée autonome : pas de clefs étrangères, pas de table de jointure.

1. CAS DE L'IDENTIFIANT

Dans une base de données relationnelles, l'ID est très souvent généré. Pour cela nous avons des annotations spécifiques :

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

2. AUTRES COLONNES

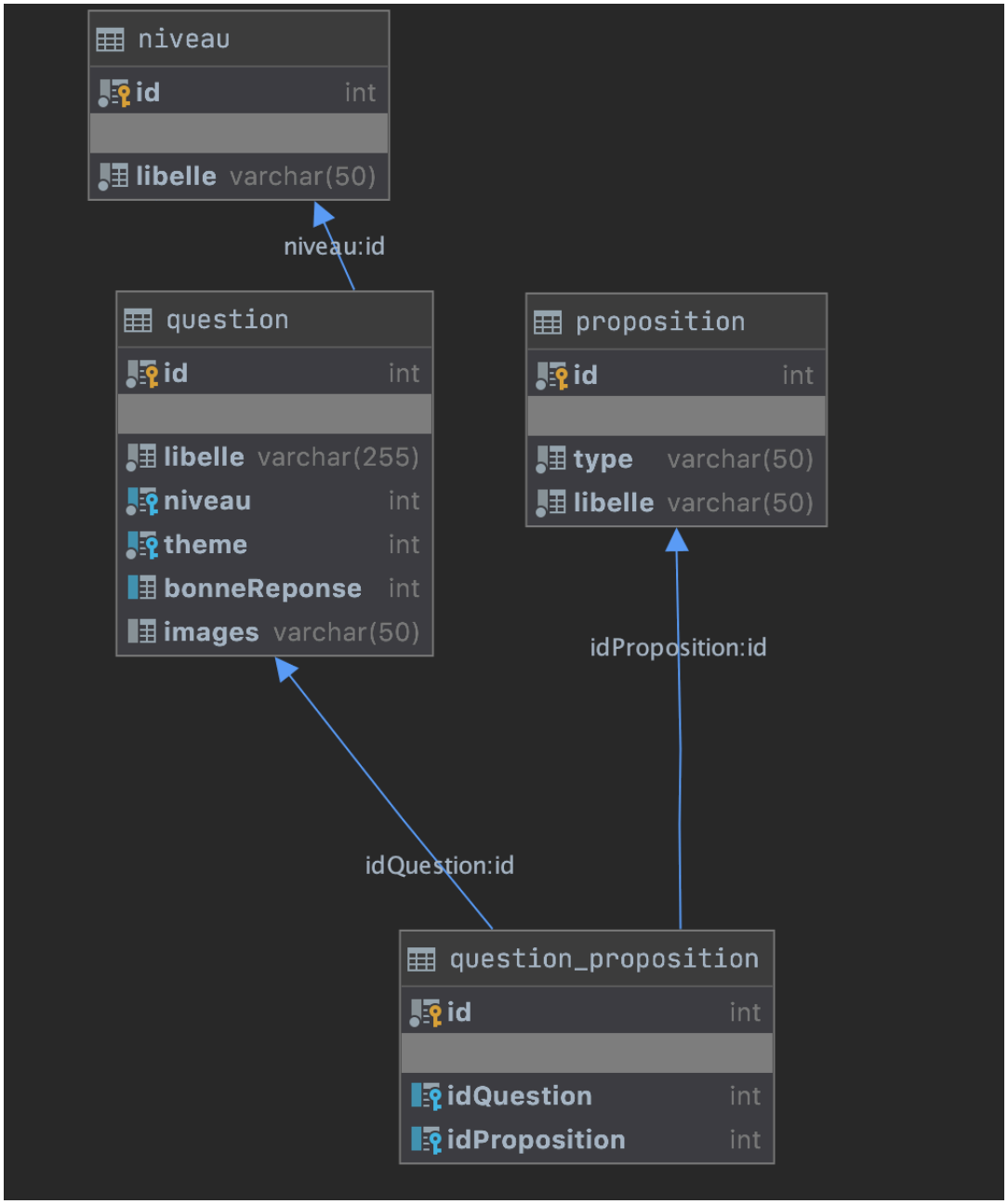
Pour cela, nous avons besoin de l'annotation :
`@Column(name="nomDeLaColonneEnBdd")`

```
@Column(name="libelle")
private String libelle;
```

C. DÉCLARATION DES ATTRIBUTS COMPLEXES

Est appelé attribut complexe tout attribut relatif à une clef étrangère ou à une table de jointure.

Prenons l'exemple ci-dessous :



1. CAS DES CLEFS ETRANGERES

Chaque attribut doit être précédé de plusieurs annotations.
Une pour indiquer le type de liaison :

`@OneToOne` (un objet question est relié à un niveau)

Une pour indiquer la colonne concernée par la clef étrangère :

`@JoinColumn(name="nomDeLaColonneEnBdd")` colonne de la base de données contenant la clef étrangère

```
@OneToOne(fetch = FetchType.LAZY)
@JoinColumn(name="niveau")
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
private Niveau niveau;
```

2. CAS DES TABLES DE JOINTURE

Initialement, **les tables de jointure doivent porter le nom des deux tables concernées** séparées par un underscore. La première table correspond à celle qui récupère les données (ici question récupère plusieurs propositions).

Côté attributs, les propositions vont être rapportées vers la question. Côté Java, cela correspond à une objet de type `List<Proposition>`. Du point de vue Entity Bean, cela se traduit par l'annotation suivante :

`@OneToMany` une question contient plusieurs propositions

```
@OneToMany
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
private List<Proposition> propositions;
```

L'annotation @JsonIgnoreProperties est utile pour charger les objets dynamiquement.