Difference between @JoinColumn and mappedBy in springboot-2024

* The **@JoinColumn** annotation specifies the column to join. **@JoinColumn indicates that this entity is the owner** of the relationship (that is: the corresponding table has a column with a foreign key to the referenced table)
* **mappedBy** specifies the referencing side (non-owning side) of the relationship.
* **mappedBy** is required in case of **Bidirectional mapping.**
* When you use the mappedBy attribute in the @OneToMany annotation, you are essentially telling Hibernate that the relationship is bidirectional and that the child entity (the "many" side of the relationship) is responsible for managing the association. The mappedBy attribute specifies the field in the child entity that owns the relationship.
* The **entity not using MappedBy** is said to be the owner of the relationship
* If the relationship is bidirectional, the non-owning side must use the mappedBy element of the OneToOne annotation to specify the relationship field or property of the owning side.
* The owning side of the relation tracked by Hibernate is the side of the relation that owns the foreign key in the database.
* owning side is the entity that has the reference to the other one
* Using mappedBy is important in bidirectional relationships because it helps Hibernate to understand how the entities are related to each other and how changes in one entity should be reflected in the other.
* The mappedBy attribute instructs Hibernate that the other end of the association, in this case the @ManyToOne side is responsible for managing this relationship, meaning that Hibernate will propagate the association state transition from the child side only while ignoring the parent side.

Bidirectional relationship provides navigational access in both directions so that you can access the other side without explicit queries.

@Entity

public class Author {

**@OneToMany(mappedBy = "author") 🡸 means owned by, author is the owner of the relationship**

private List<Book> books;

}

@Entity

public class Book {

@ManyToOne

**@JoinColumn(name = "author\_id")**

private Author author;

}

The mappedBy attribute on the OneToMany annotation specifies that the author property on the Book entity is the owner of the relationship. This means that you can navigate the relationship from the Author to associated Book entities, and from the Book entity to its associated Author entity.

Unidirectional One-to-Many: In a unidirectional one-to-many relationship, only one entity has a reference to the other entity. This means that you can only navigate the relationship in one direction, from parent to child.

@Entity

public class Author {

@OneToMany

@JoinColumn(name = "author\_id")

private List<Book> books;

}

@Entity

public class Book {

// No reference to the Author entity

}

In this example, the Author entity has a collection of Book entities, but the Book entity does not have a reference to the Author entity. This means that you can only navigate the relationship from the Author entity to its associated Book entities, but not from the Book entity to its associated Author entity.

**@JoinColumn**

The purpose of @JoinColumn is to create a join column if one does not already exist. If it does, then this annotation can be used to name the join column.

**MappedBy**

The purpose of the MappedBy parameter is to instruct JPA: Do NOT create another join table as the relationship is already being mapped by the opposite entity of this relationship.

* Uni-directional mapping from Branch to Company.
* Bi-directional mapping from Company to Branch.
* Only Uni-directional mapping from Company to Branch.

So any use-case will fall under this three categories.

Uni-directional mapping from Branch to Company 🡸 Use **JoinColumn** in Branch table.

Bi-directional mapping from Company to Branch 🡸 Use **mappedBy** in Company table

Uni-directional mapping from Company to Branch 🡸 Just use @JoinColumn in Company table.

In case of **One To One Bidirectional Mapping**

@Entity(name = "Parent")

@Table(name = "parent")

**public** **class** Parent {

@OneToOne(cascade = CascadeType.***ALL***)

**private** Child child;

}

@Entity(name = "Child")

@Table(name = "child")

**public** **class** Child {

/\*

\* By writting mappedBy = child,

\* Here Parent is the owner

\* and parent table will have

\* child id as reference

\*/

@OneToOne(mappedBy = "child")

**private** Parent parent;

}

**Database Table Design**

Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generated

\*\*\* **Here Parent is the owner**.

@Entity(name = "Branch")

@Table(name = "branch")

**public** **class** Branch {

@OneToOne

**private** Company company;

}

@Entity(name = "Company")

@Table(name = "company")

**public** **class** Company {

/\*

\* In this case Branch class

\* is the owning side.

\* Whenever you use mappedBy,

\* the opposite side class

\* will be the owner.

\*/

@OneToOne(cascade = CascadeType.***ALL***,

mappedBy = "company")

**private** Branch branch;

}

**\*\*\* Here Branch is the owner**

**Database Table Design**

Graphical user interface, table

Description automatically generated Table

Description automatically generated