Hibernate/JPA notes

# What is Hibernate about

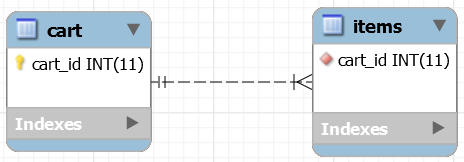
# Tutorial One-Many Relationship.

This quick Hibernate tutorial will take us through an example of a *one-to-many* mapping using JPA annotations, an alternative to XML.

We’ll also learn what bidirectional relationships are, how they can create inconsistencies, and how the idea of ownership can help.

Simply put,***one-to-many* mapping means that one row in a table is mapped to multiple rows in another table.**

Let’s look at the following [entity-relationship diagram](https://www.baeldung.com/cs/erd) to see a*one-to-many* association:



For this example, we’ll implement a cart system where we have a table for each cart and another table for each item. **One cart can have many items, so here we have a one-to-many mapping.**

The way this works at the database level is we have a cart\_id as a primary key in the cart table and also a cart\_id as a foreign key in items.

The way we do it in code is with @OneToMany.

Let’s map the Cart class to the collection of Item objects in a way that reflects the relationship in the database:

**public** **class** **Cart** {

@OneToMany(mappedBy="cart")

**private** Set<Item> items;

}

We can also add a reference to *Cart* in each *Item* using *@ManyToOne*, making this a [bidirectional](https://docs.jboss.org/hibernate/orm/4.1/manual/en-US/html/ch07.html#collections-bidirectional) relationship. Bidirectional means that **we are able to access *items* from *carts*, and also *carts* from *items***.

The *mappedBy* property is what we use to tell Hibernate which variable we are using to represent the parent class in our child class.

## Maven Setup

Let’s start by adding the Hibernate and H2 driver dependencies to our *pom.xml* file. The Hibernate dependency uses JBoss logging, and it automatically gets added as transitive dependencies:

* Hibernate version 6*.1.7.Final*
* H2 driver version *2.1.214*

Please visit the Maven central repository for the latest versions of [Hibernate](https://mvnrepository.com/artifact/org.hibernate/hibernate-core) and the [H2](https://mvnrepository.com/artifact/com.h2database/h2) dependencies.

## Hibernate Session Factory

Next, let’s create the Hibernate SessionFactory for our database interactions:

**public** **static** SessionFactory **getSessionFactory**() {

**ServiceRegistry** serviceRegistry = **new** **StandardServiceRegistryBuilder**()

.applySettings(dbSettings())

.build();

**Metadata** metadata = **new** **MetadataSources**(serviceRegistry)

.addAnnotatedClass(Cart.class) // other domain classes

.buildMetadata();

**return** metadata.buildSessionFactory();

}

**private** **static** Map<String, Object> **dbSettings**() {

// return Hibernate settings

}

## The Models.

The mapping-related configurations will be done using JPA annotations in the model classes:

@Entity

@Table(name="CART")

**public** **class** **Cart** {

//...

@OneToMany(mappedBy="cart")

**private** Set<Item> items;

// getters and setters

}

Please note that the @OneToMany annotation is used to define the property in Item class that will be used to map the mappedBy variable. That is why we have a property named “cart” in the Item class:

@Entity

@Table(name="ITEMS")

**public** **class** **Item** {

//...

@ManyToOne

@JoinColumn(name="cart\_id", nullable=false)

**private** Cart cart;

**public** **Item**() {}

// getters and setters

}

It’s also important to note that the @ManyToOne annotation is associated with the Cart class variable. @JoinColumn annotation references the mapped column.

## **5. In Action**

In the test program, we are creating a class with a main() method for getting the Hibernate Session, and saving the model objects into the database implementing the one-to-many association:

sessionFactory = HibernateAnnotationUtil.getSessionFactory();

session = sessionFactory.getCurrentSession();

LOGGER.info("Session created");

tx = session.beginTransaction();

session.save(cart);

session.save(item1);

session.save(item2);

tx.commit();

LOGGER.info("Cart ID={}", cart.getId());

LOGGER.info("item1 ID={}, Foreign Key Cart ID={}", item1.getId(), item1.getCart().getId());

LOGGER.info("item2 ID={}, Foreign Key Cart ID={}", item2.getId(), item2.getCart().getId());Copy

This is the output of our test program:

Session created

Hibernate: insert into CART values ()

Hibernate: insert into ITEMS (cart\_id)

values (?)

Hibernate: insert into ITEMS (cart\_id)

values (?)

Cart ID=7

item1 ID=11, Foreign Key Cart ID=7

item2 ID=12, Foreign Key Cart ID=7

Closing SessionFactoryCopy

## **6. The**@ManyToOne**Annotation**

As we have seen in section 2, we can specify a many-to-one relationship by using the @ManyToOne annotation. A many-to-one mapping means that many instances of this entity are mapped to one instance of another entity – **many items in one cart**.

**The @ManyToOne annotation lets us create bidirectional relationships too.** We’ll cover this in detail in the next few subsections.

## **6.1. Inconsistencies and Ownership**

Now, if Cart referenced Item, but Item didn’t in turn reference Cart, **our relationship would be unidirectional**. **The objects would also have a natural consistency.**

In our case though, the relationship is bidirectional, **bringing in the possibility of inconsistency.**

Let’s imagine a situation where a developer wants to add an item1 to the cart1 instance and an item2 to the cart2 instance, but makes a mistake so that the references between cart2 and item2 become inconsistent:

**Cart** cart1 = **new** **Cart**();

**Cart** cart2 = **new** **Cart**();

**Item** item1 = **new** **Item**(cart1);

**Item** item2 = **new** **Item**(cart2);

Set<Item> itemsSet = **new** **HashSet**<Item>();

itemsSet.add(item1);

itemsSet.add(item2);

cart1.setItems(itemsSet); // wrong!

As shown above, item2 references cart2, whereas *cart2*doesn’t reference item2, and that’s bad.

**How should Hibernate save item2 to the database?** Will item2 foreign key reference cart1 or cart2?

We resolve this ambiguity using the idea of an owning side of the relationship; references belonging to the owning side take precedence and are saved to the database.

### **6.2.**Item**as the Owning Side**

As stated in the [JPA specification](http://download.oracle.com/otndocs/jcp/persistence-2.0-fr-eval-oth-JSpec/) under section 2.9, **it’s a good practice to mark the many-to-one side as the owning side.**

In other words, Item would be the owning side and Cart the inverse side, which is exactly what we did earlier.

So how did we achieve this?

**By including the mappedBy attribute in the Cart class, we mark it as the inverse side.**

**At the same time, we also annotate the Item.cart field with @ManyToOne, making Item the owning side.**

Going back to our “inconsistency” example, now Hibernate knows that **item2‘s reference is more important and will save item2‘s reference to the database.**

Let’s check the result:

item1 ID=1, Foreign Key Cart ID=1

item2 ID=2, Foreign Key Cart ID=2Copy

Although cart references item2 in our snippet, item2‘s reference to cart2 is saved in the database.

### **6.3.**Cart**as the Owning Side**

It’s also possible to mark the one-to-many side as the owning side, and the many-to-one side as the inverse side.

Although this is not a recommended practice, let’s go ahead and give it a try.

The code snippet below shows the implementation of the one-to-many side as the owning side:

**public** **class** **ItemOIO** {

// ...

@ManyToOne

@JoinColumn(name = "cart\_id", insertable = false, updatable = false)

**private** CartOIO cart;

//..

}

**public** **class** **CartOIO** {

//..

@OneToMany

@JoinColumn(name = "cart\_id") // we need to duplicate the physical information

**private** Set<ItemOIO> items;

//..

}

Notice how we removed the mappedBy element and set the many-to-one @JoinColumn as insertable and updatable to false.

If we run the same code, the result will be the opposite:

item1 ID=1, Foreign Key Cart ID=1

item2 ID=2, Foreign Key Cart ID=1Copy

As shown above, now item2 belongs to the cart.

## **7. Conclusion**

We have seen how easy it is to implement the one-to-many relationship with the Hibernate ORM and H2 database using JPA annotations.

Additionally, we learned about bidirectional relationships and how to implement the notion of an owning side.

The source code in this article can be found [over on GitHub](https://github.com/eugenp/tutorials/tree/master/persistence-modules/hibernate-annotations).