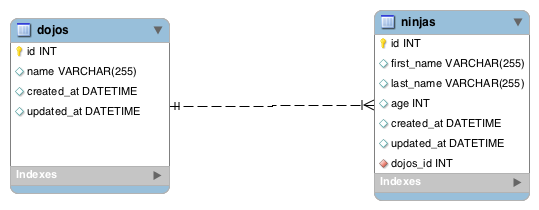
Relationships

**Relationships Continued**

In this tab, we will learn how to create One-to-Many relationships

**1:n**

We are going to create a 1:n relationship between a dojo and many ninjas



**models/Dojo.java**

// ...

@Entity

@Table(name="dojos")

public class Dojo {

@Id

@GeneratedValue

private Long id;

private String name;

@Column(updatable=false)

private Date createdAt;

private Date updatedAt;

@OneToMany(mappedBy="dojo", fetch = FetchType.LAZY)

private List<Ninja> ninjas;

public Dojo() {

}

// ...

// getters and setters removed for brevity

// ...

}

**models/Ninja.java**

// ..

@Entity

@Table(name="ninjas")

public class Ninja {

@Id

@GeneratedValue

private Long id;

private String firstName;

private String lastName;

private int age;

@Column(updatable=false)

private Date createdAt;

private Date updatedAt;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name="dojo\_id")

private Dojo dojo;

public Ninja() {

}

// ...

// getters and setters removed for brevity

// ...

}

**New Annotations Used**

* @OneToMany: Defines a many-valued association with one-to-many multiplicity. This may be used within an embeddable class contained within an entity class to specify a relationship to a collection of entities. Notice that in this case, our ninjas attribute is of type List<Ninja>.
  + @OneToMany(mappedBy="dojo"): This will map the ninjas attribute in the Dojo class to the dojo attribute in the Ninja class.
  + There are a few other options that you can have. We encourage you to visit the useful link for the One-to-Many relationship.
* @ManyToOne: Defines a single-valued association to another entity class that has many-to-one multiplicity. This may be used within an embeddable class to specify a relationship from the embeddable class to an entity class. Notice that our dojo attribute is referring to the dojo\_id. Therefore, this attribute gives the the dojo that a specific ninja belongs to.
* @JoinColumn(name="dojo\_id"): Defines mapping for composite foreign keys. It indicates that the corresponding table to this entity has a foreign\_key to the referenced table.

**Useful Links**

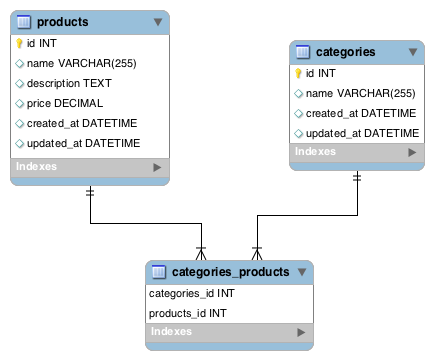
* [One-to-Many](http://www.objectdb.com/api/java/jpa/OneToMany)
* [Many-to-One](http://www.objectdb.com/api/java/jpa/ManyToOne)

# Relationships Continued

In this tab, we will learn how to create Many-to-Many relationships

## n:n

We are going to create a n:n relationship between products and categories. A product can be in many categories and a category can have many products.



##### models/Product.java

// ...

@Entity

@Table(name="products")

public class Product {

@Id

@GeneratedValue

private Long id;

private String name;

private String description;

private float price;

@Column(updatable=false)

private Date createdAt;

private Date updatedAt;

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(

name = "categories\_products",

joinColumns = @JoinColumn(name = "product\_id"),

inverseJoinColumns = @JoinColumn(name = "category\_id")

)

private List<Category> categories;

public Product() {

}

// ...

// getters and setters removed for brevity

// ...

}

##### models/Category.java

// ..

@Entity

@Table(name="categories")

public class Category {

@Id

@GeneratedValue

private Long id;

private String name;

@Column(updatable=false)

private Date createdAt;

private Date updatedAt;

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(

name = "categories\_products",

joinColumns = @JoinColumn(name = "category\_id"),

inverseJoinColumns = @JoinColumn(name = "product\_id")

)

private List<Product> products;

public Category() {

}

// ...

// getters and setters removed for brevity

// ...

}

Setting up your models like the above, will automatically create the middle joining table for us.

### New Annotations Used

* @ManyToMany: Defines a many-valued association with many-to-many multiplicity. You will have to use this annotation on both entities.
* @JoinTable: Defines the middle table the our entities will be mapped to.
  + @JoinTable(name="categories\_products"): The name of the middle table.
  + joinColumns: The foreign key that matches the primary key of the embedded class when the tables is joined.
  + inverseJoinColumns: The foreign key that that matched the foreign key of the opposite class when the tables are joined.

### Useful Links

* [Many-to-Many](http://www.objectdb.com/api/java/jpa/ManyToMany)