Hibernate Tutorial

SessionFactory is a factory class for Session objects. It is available for the whole application while a Session is only available for particular transaction.

12

**SessionFactory** is an interface. SessionFactory can be created by providing Configuration object, which will contain all DB related property details pulled from either hibernate.cfg.xml file or hibernate.properties file. SessionFactory is a factory for Session objects.

We can create one SessionFactory implementation per database in any application. If your application is referring to multiple databases, then you need to create one SessionFactory per database.

The SessionFactory is a heavyweight object; it is usually created during application start up and kept for later use. The SessionFactory is a thread safe object and used by all the threads of an application.

**A Session** is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed. The main function of the Session is to offer, create, read, and delete operations for instances of mapped entity classes.

The main difference between the first level and second level cache in Hibernate is that **the first level is maintained at the Session level and accessible only to the Session, while the second level cache is maintained at the SessionFactory level and available to all Sessions**.

<https://www.baeldung.com/hibernate-5-bootstrapping-api>

<https://www.javaguides.net/2019/03/jsp-servlet-hibernate-crud-example.html>

JoinColumn vs mappedBy

https://www.baeldung.com/jpa-joincolumn-vs-mappedby

Overview of JPA/Hibernate Cascade Types

https://www.viralpatel.net/hibernate-many-to-many-annotation-mapping-tutorial/

Many to Many

**@ManyToMany** – Is used to create many-to-many relationship between Employee and Meeting entities. If the Collection is defined using generics to specify the element type, the associated target entity class does not need to be specified; otherwise it must be specified. Every many-to-many association has two sides, the owning side and the non-owning, or inverse, side. The join table is specified on the owning side. If the association is bidirectional, either side may be designated as the owning side.

Note that in above entity classes, Employee is defined as relationship owner as @JoinColumn is define in Employee class and mappedBy is specified in Meeting class.

**@JoinTable** – Is used to define the join table (link table) for many-to-many relationship. It is specified on the owning side of a many-to-many association, or in a unidirectional one-to-many association. In this case the join table is EMPLOYEE\_MEETING.

If the JoinTable annotation is missing, the default values of the annotation elements apply. The name of the join table is assumed to be the table names of the associated primary tables concatenated together (owning side first) using an underscore.

**@JoinColumn** – Is used to define the join column (linking column) in both main tables.

Note that we have used SET to map meetings with employee and vice versa. A <set> is similar to except that it can only store unique objects. That means no duplicate elements can be contained in a set. When you add the same element to a set for second time, it will replace the old one. A set is unordered by default but we can ask it to be sorted. The corresponding type of a <set> in Java is java.util.Set.