Contents

## Introduction to Hibernate

While talking about the best Java web frameworks, Hibernate ORM can't be ignored. Hibernate is an essential object-relation mapping tool for Java programming language. It offers a mapping framework for a domain model (object-oriented) to one relational database. Hibernate can solve the object-relational impedance incongruity problems by substituting persistent and direct database with high-level object controlling functions. It is free software distributed under public 2.1 License of GNU Lesser General.

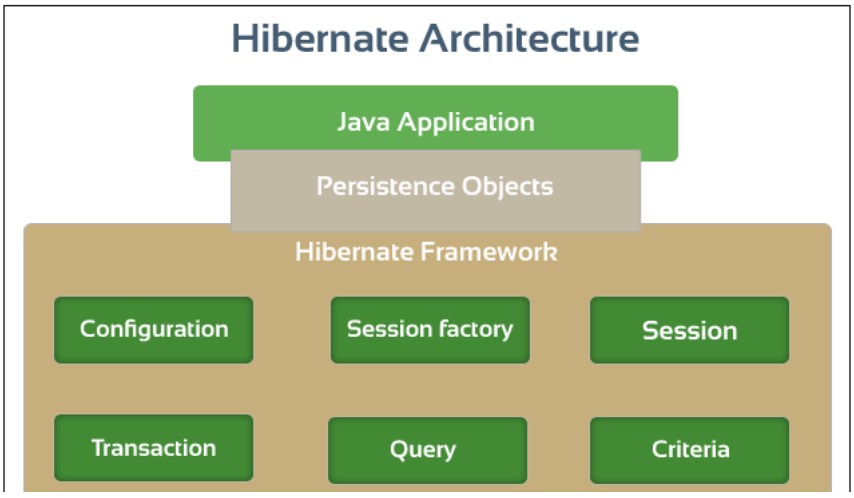
Pros:

- Hibernate enables you to communicate with any database by making tiny alternations in code
- MySQL, Db2 or Oracle, Hibernate is DB independent
- Caching instrument to bug catalog with same queries
- N+1 or Sluggish loading support
- Low risk of data loss and it requires less power

Cons:

- If power goes off, you can lose all your data
- Restarting can be extremely slow

Hibernate framework is an advanced ORM framework that lets you perform the database operation on Java objects. This framework carries the best features that help backend developers fine-tune data access layer.



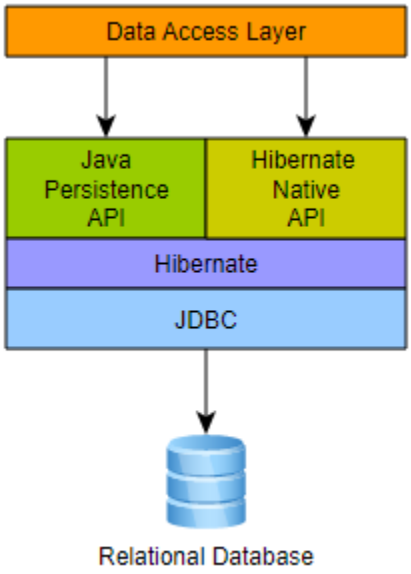

**Productivity-boosting features**

- N+1 or Lazy loading support
- Built in Caching mechanism
- High-level object handling functions.

- It allows you to communicate with any database using tiny alterations.

**Architecture**

Hibernate, as an ORM solution, effectively "sits between" the Java application data access layer and the Relational Database, as can be seen in the diagram above. The Java application makes use of the Hibernate APIs to load, store, query, etc its domain data.

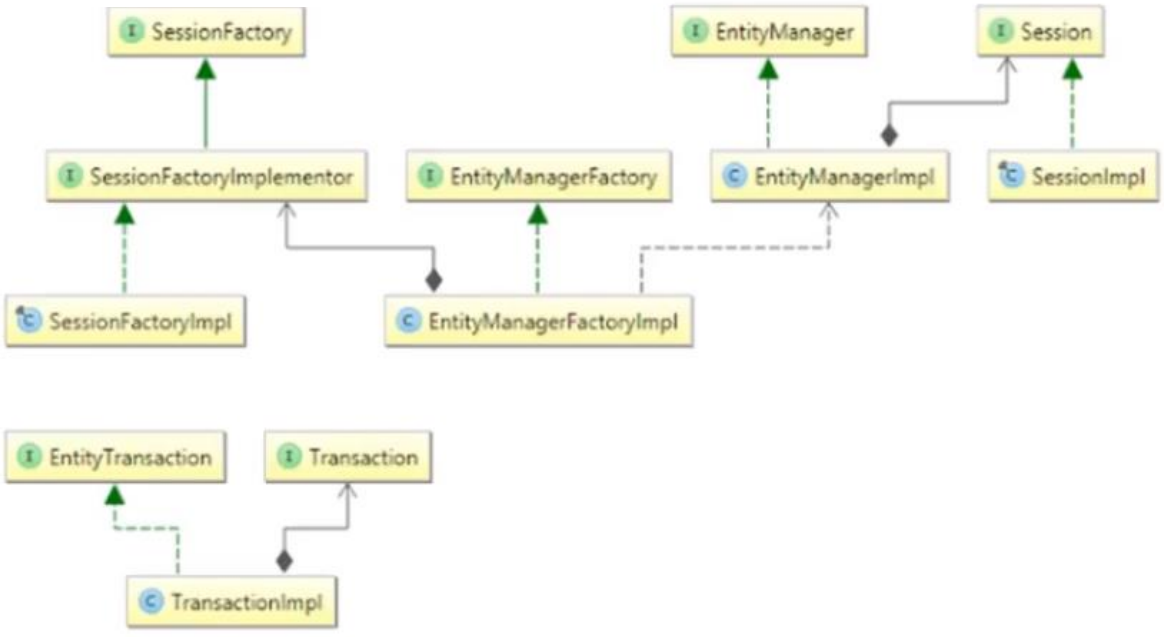



As a JPA provider, Hibernate implements the Java Persistence API specifications and the association between JPA interfaces and Hibernate specific implementations can be visualized in the following diagram:

**Hibernate Components**

**SessionFactory (org.hibernate.SessionFactory)**

A thread-safe (and immutable) representation of the mapping of the application domain model to a database. Acts as a factory for ***org.hibernate.*Session** instances. The




**EntityManagerFactory** is the JPA equivalent of a **SessionFactory** and basically those two converge into the same **SessionFactory** implementation.

A **SessionFactory** is very expensive to create, so, for any given database, the application should have only one associated **SessionFactory**. The **SessionFactory** maintains services that Hibernate uses across all Session(s) such as second level caches, connection pools, transaction system integrations, etc.

**Session (org.hibernate.Session)**

A single-threaded, short-lived object In JPA nomenclature, the Session is represented by an EntityManager.Behind the scenes, the Hibernate Session wraps a JDBC java.sql.Connection and acts as a factory for org.hibernate.Transaction instances. It maintains a generally "repeatable read" persistence context (first level cache) of the application domain model.

**Transaction (org.hibernate.Transaction)**

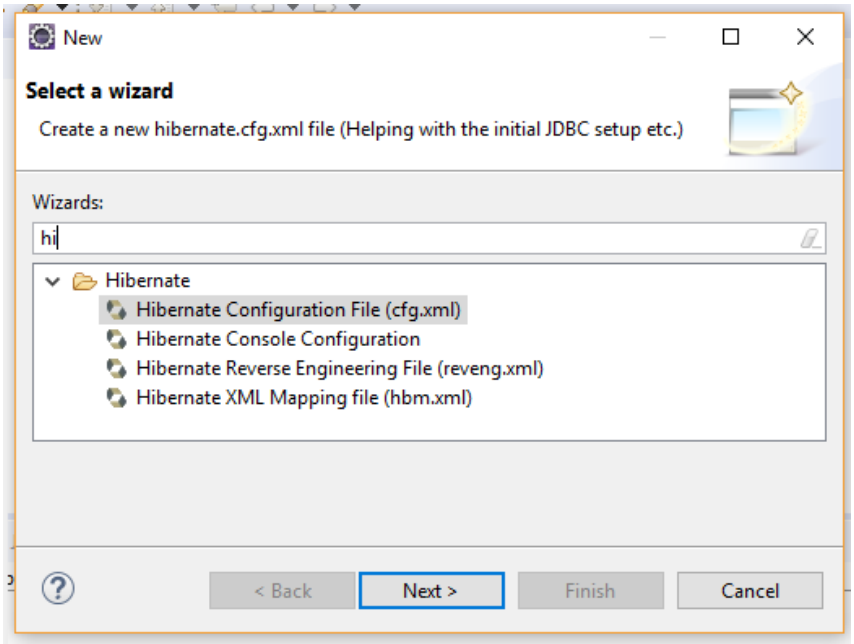
A single-threaded, short-lived object used by the application to demarcate individual physical transaction boundaries.EntityTransaction is the JPA equivalent and both act as an abstraction API to isolate the application from the underlying transaction system in use (JDBC or JTA).

**Installation JBoss Tools**

Step 1: Goto Help on any version of eclipse and click Eclipse Marketplace and type **hibernate** to the search box

Step 2: Select JBoss Tols and install, it will ask you are you want to install plugin after some time than click on. Once you click ok, when installation complete it will prompt you to restart eclipse then click ok

Step 3: Once your eclipse restarts and plugin installed successfully you will see Hibernate as below from new project and other option and type **hibernate** as shown in the figure



**Hibernate Annotations**

| Annotation | Package Detail/Import statement |
|---|---|
| @Entity | import javax.persistence.Entity; |
| @Table | import javax.persistence.Table; |
| @Column | import javax.persistence.Column; |
| @Id | import javax.persistence.Id; |
| @GeneratedValue | import javax.persistence.GeneratedValue; |
| @OrderBy | import javax.persistence.OrderBy; |
| @Transient | import javax.persistence.Transient; |
| @Lob | import javax.persistence.Lob; |
| Hibernate Association Mapping Annotations | |
| @OneToOne | import javax.persistence.OneToOne; |
| @ManyToOne | import javax.persistence.ManyToOne; |
| @OneToMany | import javax.persistence.OneToMany; |
| @ManyToMany | import javax.persistence.ManyToMany; |
| @PrimaryKeyJoinColumn | import javax.persistence.PrimaryKeyJoinColumn; |
| @JoinColumn | import javax.persistence.JoinColumn; |
| @JoinTable | import javax.persistence.JoinTable; |
| @MapsId | import javax.persistence.MapsId; |

**@Table**

Specify the database table this Entity maps to using the name attribute of @Table annotation. In the example below, the data will be stored in 'company' table in the database.

```
@Entity
```

```
@Table(name = "company")
public class Company implements Serializable {
	...
}
```

**@Column**

Specify the column mapping using @Column annotation to change the column name.

```
@Entity
@Table(name = "company")
public class Company implements Serializable {

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

	...
}
```

**@Id**

Annotate the id column using @Id for primary key field

```
@Entity
@Table(name = "company")
public class Company implements Serializable {

	@Id
	@Column(name = "id")
	private int id;

	...
}
```

**@GeneratedValue**

Let database generate (auto-increment) the id column.

```
@Entity
```

```
@Table(name = "company")
public class Company implements Serializable {

	@Id
	@Column(name = "id")
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;

	...
}
```

**@OrderBy**

Sort your data using @OrderBy annotation. In example below, it will sort all contacts in a company by their firstname in ascending order.

```
@Entity
@Table(name = "company")
public class Company implements Serializable {

	@Id
	@Column(name = "id")
	@GeneratedValue
	private int id;

	@OrderBy("firstName asc")
	private Set contacts;

	...........

}
```

**@Transient**

Annotate your transient properties with @Transient.

**@Lob**

Annotate large objects with @Lob.

## Maven and Gradle

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

If you think that Maven could help your project, you can find out more information in the "About Maven" section of the https://maven.apache.org/what-is-maven.html website

You can create maven project for desktop and web applications.

**pom.xml file for desktop app**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>

      <groupId>np.com.coretechnology</groupId>
      <artifactId>desktop</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <packaging>jar</packaging>

      <name>desktop</name>
      <url>http://maven.apache.org</url>

      <properties>
            <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
      </properties>

      <dependencies>

            <dependency>
                <groupId>org.hibernate.orm</groupId>
                <artifactId>hibernate-core</artifactId>
                <version>6.2.6.Final</version>
            </dependency>
            <dependency>
                  <groupId>mysql</groupId>
                  <artifactId>mysql-connector-java</artifactId>
                  <version>8.0.28</version>
            </dependency>

            <dependency>
```

```
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.6.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

**pom.xml file for web application**

```
<project                        xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>np.com.coretechnology</groupId>
    <artifactId>web</artifactId>
    <packaging>jar</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>web Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <dependencies>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>org.hibernate.orm</groupId>
```

```
            <artifactId>hibernate-core</artifactId>
            <version>6.2.6.Final</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.28</version>
        </dependency>


        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <finalName>web</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.6.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>

    </build>
</project>
```

**Questions**

Define Hibernate. Explain with Architecture Diagram
Define Annotations. Explain Different types Annotations with examples
Define Maven. Explain the working of Maven and also its advantages
Define pom.xml file. Illustrate with example for desktop and web application

**Hibernate Configuration file**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
            "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
            "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
      <session-factory>
            <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
            <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/ems_desktop</prop
erty>
            <property name="hibernate.connection.username">root</property>
            <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
            <property name="hibernate.hbm2ddl.auto">update</property>
            <property name="hibernate.show_sql">true</property>
            <property name="hibernate.format_sql">true</property>
            <property name="hibernate.connection.pool_size">2</property>
            <mapping class="np.com.westernit.ems_desktop.entit.Student" />
      </session-factory>
</hibernate-configuration>
```

**Hibernate SessionFactory Class**

```
package np.com.westernit.ems_desktop.utils;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtils {
      private static SessionFactory sessionFactory;
      public static SessionFactory getSessionFactory(){
            if(sessionFactory!=null) {
                  return sessionFactory;
            }
            Configuration cfg = new Configuration();
            sessionFactory = cfg.configure().buildSessionFactory();
            return sessionFactory;
      }
}
```

**OR**

**Java Configuration class for Configuration class**

```
package np.westernit.app.utils;
import java.util.Properties;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
```

```
import org.hibernate.cfg.Environment;
import org.hibernate.service.ServiceRegistry;
import np.westernit.app.entities.Student;

public class HibernateUtils {
private static SessionFactory sessionFactory;
public static SessionFactory getSessionFactory() {
if(sessionFactory!=null) {
      return sessionFactory;
}
Configuration cfg = new Configuration();
Properties settings = new Properties();
settings.put(Environment.URL, "jdbc:mysql://localhost:3306/hibernatedemo");
settings.put(Environment.USER, "root");
settings.put(Environment.PASS, "");
settings.put(Environment.DRIVER, "com.mysql.cj.jdbc.Driver");
settings.put(Environment.DIALECT, "org.hibernate.dialect.MySQLDialect");
settings.put(Environment.HBM2DDL_AUTO, "create");
settings.put(Environment.SHOW_SQL, "true");
settings.put(Environment.FORMAT_SQL, "true");

cfg.setProperties(settings);
cfg.addAnnotatedClass(Student.class);

ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
            .applySettings(cfg.getProperties()).build();

sessionFactory = cfg.buildSessionFactory(serviceRegistry);
return sessionFactory;

}
}
```

**Entity class for CRUD operations**

```
package np.com.westernit.ems_desktop.entit;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="tbl_students")
public class Student {

      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
	private Long id;

	@Column(nullable = false, unique = true,length = 30, name="first_name")
	private String firstName;

	private String lastName;

	public Student() {
		// TODO Auto-generated constructor stub
	}
	public Long getId() {
		return id;
	}

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

	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	public Student(String firstName, String lastName) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
	}

}
```

**Create record or save record to database**

```
Student s = new Student("Mr Bahadur", "Bist");
try(Session session = HibernateUtils.getSessionFactory().openSession()){
	session.getTransaction().begin();
	//save to database
	session.persist(s);
	session.getTransaction().commit();
}catch(Exception e) {
	System.out.println(e);
```

```
}
```

**Update Record from a database**

```
try(Session session = HibernateUtils.getSessionFactory().openSession()){
	session.getTransaction().begin();
	Student s = session.get(Student.class, 1);
	s.setFirstName("Mr. Sunil");
	session.persist(s);
	session.getTransaction().commit();
}catch(Exception e) {
	System.out.println(e);
}
```

**Delete Record from a database**

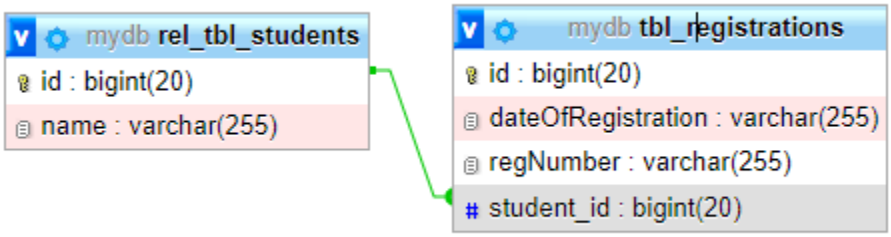
```
try(Session session = HibernateUtils.getSessionFactory().openSession()){
	session.getTransaction().begin();
	Student s = session.get(Student.class, 1);
	session.remove(s);
	session.getTransaction().commit();
}catch(Exception e) {
	System.out.println(e);
}
```

**Select all records**

```
try(Session session = HibernateUtils.getSessionFactory().openSession()){
	session.getTransaction().begin();
	List<Student> students = session.createQuery("SELECT s FROM Student s",
Student.class).getResultList();
	for(Student std: students) {
		System.out.println(std.getFirstName());
	}
	session.getTransaction().commit();

}catch(Exception e) {
	System.out.println(e);
}

//OR
try(Session session = HibernateUtils.getSessionFactory().openSession()){
	//List<Student> students =
session.createCriteria(Student.class).list();
	//List<Student> students =(List<Student>) session.createQuery("FROM
Student").list();
	CriteriaBuilder builder = session.getCriteriaBuilder();
	CriteriaQuery<Student> criteria = builder.createQuery(Student.class);
	criteria.from(Student.class);
	List<Student> students= session.createQuery(criteria).getResultList();
	for(Student s: students){
```

```
            System.out.println(s.getFirstName());
      }

      System.out.println("Successful...");
}catch(Exception e){
      System.out.println("Error On Reading All: "+e.getMessage());
            }


      }
}
```

## One to One Relation



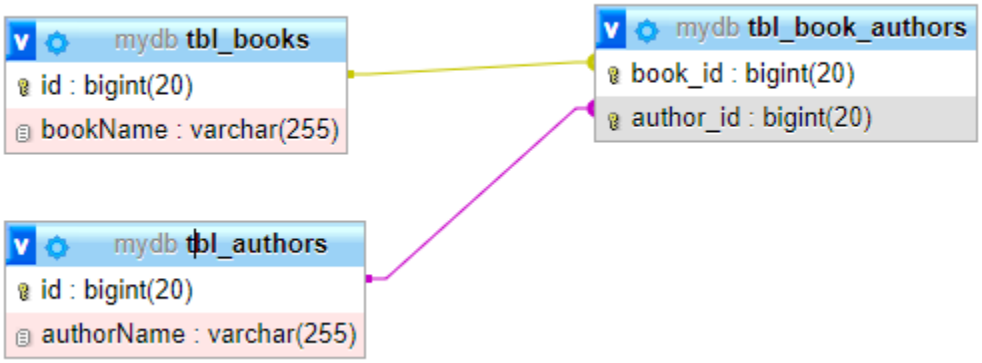

```
@Entity
@Table(name="tbl_registrations")
public class Registration {
      @Id
      @GeneratedValue(strategy = GenerationType.AUTO)
      private Long id;
      private String regNumber;
      private String dateOfRegistration;

      @OneToOne(cascade = CascadeType.ALL)
      @JoinColumn(name="student_id", nullable = false,referencedColumnName =
"id")
      private Student student;

}

@Entity
@Table(name="rel_tbl_students")
public class Student implements Serializable {
      @Id
      @GeneratedValue(strategy = GenerationType.AUTO)
      private Long id;
      @Column(nullable = false)
      private String name;
```

```
	@OneToOne(mappedBy = "student")
	private Registration registration;

}
```

**Entity for One to Many Relationship mapping**


```
@Entity
@Table(name="tbl_students")
public class Student implements Serializable {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;
	@Column(nullable = false)
	private String name;

	@OneToMany(mappedBy = "student", cascade = CascadeType.ALL)
	private List<Address> address;
}
@Entity
@Table(name="tbl_address")
public class Address implements Serializable {

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

	private String country;

	@ManyToOne
	@JoinColumn(name="student_id", nullable = false, referencedColumnName =
"id")
	private Student student;
}
```

## Entity for Many to Many Relationships mapping




```
@Entity
@Table(name="tbl_books")
public class Book implements Serializable {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;
	private String bookName;

	@ManyToMany(cascade = { CascadeType.ALL })
	@JoinTable(
			name = "tbl_book_authors",
			joinColumns = { @JoinColumn(name = "book_id") },
			inverseJoinColumns = { @JoinColumn(name = "author_id") }
			)
	private Set<Author> authors = new HashSet<>();
}

@Entity
@Table(name="tbl_authors")
public class Author implements Serializable {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;
	private String authorName;

	@ManyToMany(mappedBy = "authors")
	private Set<Book> books = new HashSet<>();
}
```