(/)

Hibernate Many to Many Annotation Tutorial

Last updated: January 8, 2024

Written by: Zeger Hendrikse (https://www.baeldung.com/author/zeger-author)

Reviewed by: Zeger Hendrikse (https://www.baeldung.com/editor/zeger-author)

Persistence (https://www.baeldung.com/category/persistence)

Entity Relationships (https://www.baeldung.com/tag/entity-relationships)

Hibernate (https://www.baeldung.com/tag/hibernate)

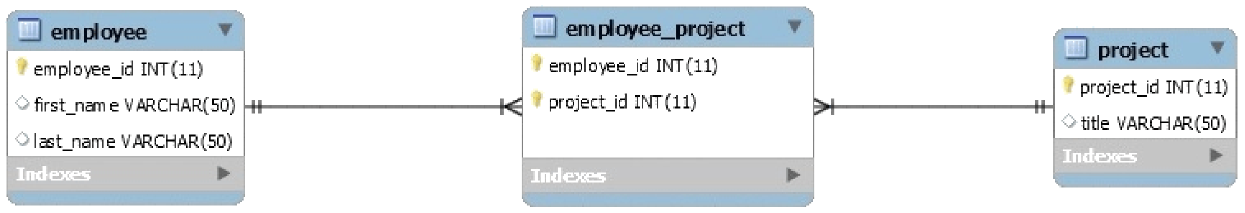
1. Introduction

(/)

In this quick tutorial, we'll have a quick look at how the *@ManyToMany* annotation can be used for specifying this type of relationships in Hibernate.

2. A Typical Example

Let's start with a simple Entity Relationship Diagram – which shows the many-to-many association between two entities *employee* and *project:*



(/wp-content/uploads/2017/09/New.png)

In this scenario, any given *employee* can be assigned to multiple projects and a *project* may have multiple employees working for it, leading to a many-to-many association between the two.

We have an *employee* table with *employee_id* as its primary key and a *project* table with *project_id* as its primary key. A join table *employee_project* is required here to connect both sides.

3. Database Setup

Let's assume we have an already created database with the name *spring_hibernate_many_to_many.*

We also need to create the *employee* and *project* tables along with the *employee_project* join table with *employee_id* and *project_id* as foreign keys:

```
CREATE TABLE `employee` (
  `employee_id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(50) DEFAULT NULL,
  `last_name` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`employee_id`)
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8;

CREATE TABLE `project` (
  `project_id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`project_id`)
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=utf8;

CREATE TABLE `employee_project` (
  `employee_id` int(11) NOT NULL,
  `project_id` int(11) NOT NULL,
  PRIMARY KEY (`employee_id`,`project_id`),
  KEY `project_id` (`project_id`),
  CONSTRAINT `employee_project_ibfk_1`
   FOREIGN KEY (`employee_id`) REFERENCES `employee` (`employee_id`),
  CONSTRAINT `employee_project_ibfk_2`
   FOREIGN KEY (`project_id`) REFERENCES `project` (`project_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

With the database set up, next step would be the preparation of the Maven dependencies and Hibernate configuration. For information on this, please refer to the article on Guide to Hibernate4 with Spring (/hibernate-4-spring)

4. The Model Classes

The model classes *Employee* and *Project* need to be created with JPA annotations:

```
@Entity                    (/)
@Table(name = "Employee")
public class Employee {
    // ...

    @ManyToMany(cascade = { CascadeType.ALL })
    @JoinTable(
        name = "Employee_Project",
        joinColumns = { @JoinColumn(name = "employee_id") },
        inverseJoinColumns = { @JoinColumn(name = "project_id") }
    )
    Set<Project> projects = new HashSet<>();

    // standard constructor/getters/setters
}
```

```
@Entity
@Table(name = "Project")
public class Project {
    // ...

    @ManyToMany(mappedBy = "projects")
    private Set<Employee> employees = new HashSet<>();

    // standard constructors/getters/setters
}
```

As we can see, **both the *Employee* class and *Project* classes refer to one another, which means that the association between them is bidirectional.**

In order to map a many-to-many association, we use the *@ManyToMany*, *@JoinTable* and *@JoinColumn* annotations. Let's have a closer look at them.

The *@ManyToMany* annotation is used in both classes to create the many-to-many relationship between the entities.

This association has two sides i.e. the owning side and the inverse side. In our example, the owning side is *Employee* so the join table is specified on the owning side by using the *@JoinTable* annotation in *Employee* class. The *@JoinTable* is used to define the join/link table. In this case, it is *Employee_Project.*

The *@JoinColumn* annotation is used to specify the join/linking column with the main table. Here, the join column is *employee_id* and *project_id* is the inverse join column since *Project* is on the inverse side of the relationship.

In the *Project* class, the *mappedBy* attribute is used in the *@ManyToMany* annotation to indicate that the *employees* collection is mapped by the *projects* collection of the owner side.

5. Execution

In order to see the many-to-many annotation in action, we can write the following JUnit test:

```
public class HibernateManyToManyAnnotationMainIntegrationTest {
        private static SessionFactory sessionFactory;
        private Session session;

        //...

        @Test
        public void givenSession_whenRead_thenReturnsMtoMdata() {
            prepareData();
            @SuppressWarnings("unchecked")
            List<Employee> employeeList = session.createQuery("FROM
Employee").list();
            @SuppressWarnings("unchecked")
            List<Project> projectList = session.createQuery("FROM
Project").list();
            assertNotNull(employeeList);
            assertNotNull(projectList);
            assertEquals(2, employeeList.size());
            assertEquals(2, projectList.size());

            for(Employee employee : employeeList) {
               assertNotNull(employee.getProjects());
               assertEquals(2, employee.getProjects().size());
            }
            for(Project project : projectList) {
               assertNotNull(project.getEmployees());
               assertEquals(2, project.getEmployees().size());
            }
        }

        private void prepareData() {
            String[] employeeData = { "Peter Oven", "Allan Norman" };
            String[] projectData = { "IT Project", "Networking
Project" };
            Set<Project> projects = new HashSet<Project>();

            for (String proj : projectData) {
                projects.add(new Project(proj));
            }

            for (String emp : employeeData) {
                Employee employee = new Employee(emp.split(" ")[0],
emp.split(" ")[1]);
                employee.setProjects(projects);

                for (Project proj : projects) {
                    proj.getEmployees().add(employee);
                }
                session.persist(employee);
            }
        }

        //...
}
```

We can see the many-to-many relationship between the two entities created in the database: the *employee*, *project*, and *employee_project* tables with sample data representing the relationship.

6. Conclusion

In this tutorial, we saw how to create mappings using Hibernate's many-to-many annotations, which is a more convenient counterpart compared to creating XML mapping files.

The source code of this tutorial can be found over on GitHub (https://github.com/eugenp/tutorials/tree/master/persistence-modules/hibernate-mapping-2).

Comments are closed on this article!

COURSES

ALL COURSES (/ALL-COURSES)
ALL BULK COURSES (/ALL-BULK-COURSES)
ALL BULK TEAM COURSES (/ALL-BULK-TEAM-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)

APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
(/)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)
JOBS (/TAG/ACTIVE-JOB/)
OUR PARTNERS (/PARTNERS)
PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)