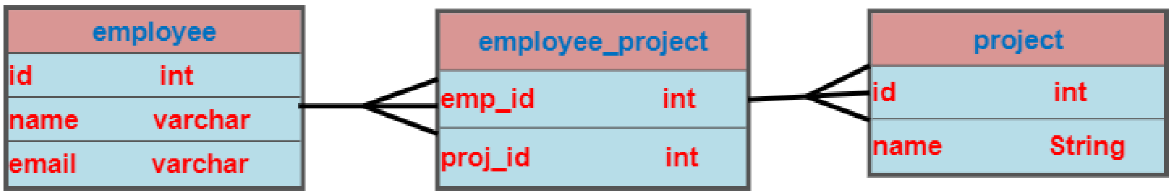
DAY 6
Implementation Of Many To Many Relationship In Hibernate

Now Let's move ahead and understand crud operation on ManyToMany relationship

A many-to-many relationship occurs when multiple records in a table are associated with multiple records in another table. For example, a many-to-many relationship exists between employees and projects:one employee will be working on the multiple projects and on one project multiple employees will be working

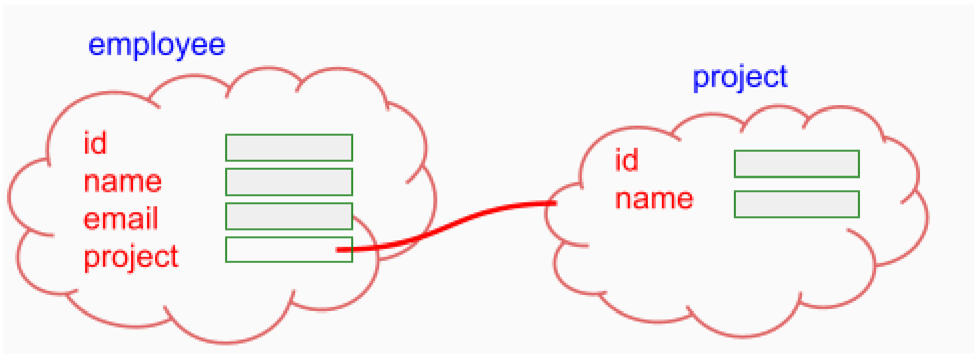
Scenario 1:

Let's take the employee and project table and there exist many-to-many relationships between these two tables then there will be one more temporary table created with the primary key of both the tables where the mapping will occur as shown below.



Steps to many-to-many relationship

- Create module class for employee table with id, name, email
- Add Project attribute which is the reference of the project object in the employee class



- Add **@ManyToMany** annotations and mention cascade type except CascadeType.Remove

```
@ManyToMany(cascade = {CascadeType.DETACH,
CascadeType.MERGE,CascadeType.PERSIST,
CascadeType.REFRESH})
private List<Project> projects;
```

- Next you need to use @JoinTable annotation to join two tables using a third table. This third table associates two tables by their Ids. To define @JoinTable we can use the below attributes.
  name: This is the name of the third table.
  joinColumns: Assign the column of the third table related to the entity itself.
  inverseJoinColumns: Assign the column of the third table related to the associated entity.

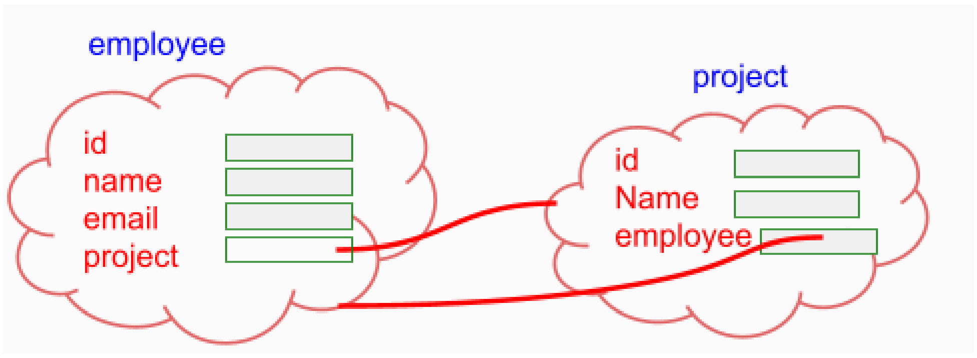
```
@ManyToMany(cascade = {CascadeType.DETACH,
CascadeType.MERGE,CascadeType.PERSIST,
CascadeType.REFRESH})

@JoinTable(name = "employee_project",
joinColumns = @JoinColumn(name = "emp_id"),
inverseJoinColumns = @JoinColumn(name = "proj_id"))
private List<Project> projects;
```

Employee.class

```java
package com.tap.model;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "employee")
public class Employee {

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

        @ManyToMany(cascade = {CascadeType.DETACH,
CascadeType.MERGE,
                        CascadeType.PERSIST, CascadeType.REFRESH})
        @JoinTable(name = "employee_project",
                        joinColumns = @JoinColumn(name = "emp_id"),
                        inverseJoinColumns = @JoinColumn(name = "proj_id"))
        private List<Project> projects;

        public Employee() {
        }
```

```
public Employee(int id, String name, String email) {
        super();
        this.id = id;
        this.name = name;
        this.email = email;
}

public int getId() {
        return id;
}

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

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public String getEmail() {
        return email;
}

public void setEmail(String email) {
        this.email = email;
}

public List<Project> getProjects() {
        return projects;
}

public void setProjects(List<Project> projects) {
		this.projects = projects;
	}

	@Override
	public String toString() {
		return "Employee [" + id + ", " + name + ", " + email + "]";
	}



}
```

- Create module class for project table
- Add employee attribute which is the reference of the employee object in the project class



- Add @ManyToMany annotations and mention cascade type except CascadeType.Remove

```
@ManyToMany(cascade = {CascadeType.DETACH,
CascadeType.MERGE,CascadeType.PERSIST,
CascadeType.REFRESH})
private List<Employee> employees;
```

- Next you need to use @JoinTable annotation to join two tables using a third table. This third table associates two tables by their Ids. To define @JoinTable we can use the below attributes.
  name: This is the name of the third table.
  joinColumns: Assign the column of the third table related to the entity itself.
  inverseJoinColumns: Assign the column of the third table related to the associated entity.

```
@ManyToMany(cascade = {CascadeType.DETACH,
CascadeType.MERGE,CascadeType.PERSIST,
CascadeType.REFRESH})

@JoinTable(name = "employee_project",
joinColumns = @JoinColumn(name = "proj_id"),
inverseJoinColumns = @JoinColumn(name = "emp_id"))
private List<Employee> employees;
```

Project.java

```
package com.tap.model;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "project")
public class Project {

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

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

	@ManyToMany(cascade = {CascadeType.DETACH,
CascadeType.MERGE,
				CascadeType.PERSIST, CascadeType.REFRESH})
	@JoinTable(name = "employee_project",
				joinColumns = @JoinColumn(name = "proj_id"),
				inverseJoinColumns = @JoinColumn(name = "emp_id"))
	private List<Employee> employees;

	public Project() {
	}
```

```java
public Project(int id, String name) {
        super();
        this.id = id;
        this.name = name;
}

public int getId() {
        return id;
}

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

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public List<Employee> getEmployees() {
        return employees;
}

public void setEmployees(List<Employee> employees) {
        this.employees = employees;
}

@Override
public String toString() {
        return "Project [" + id + ", " + name + "]";
}
```

```
}
```

- Now let's try to perform a crud operation on the ManyToMany relationship. Right now there is no table of employee and project in the database
- First we will create three employee object and two project object
- Next we will assign the projects to the employee object

Logic:

```java
// create employee object
Employee alex = new Employee(1, "Alex", "alex@gmail.com");
Employee bob = new Employee(2 , "Bob", "bob@gmail.com");
Employee charli = new Employee(3, "charli",
"charli@gmail.com");

//Create projects
Project pr = new Project(1, "Premierpro");
Project af = new Project(2, "Aftereffects");

// creating a list to store employees working on the projects
ArrayList<Employee> premployees = new ArrayList<Employee>();
ArrayList<Employee> afemployees = new ArrayList<Employee>();

// adding employees to premiere pro project list
premployees.add(alex);
premployees.add(bob);
premployees.add(charli);

//Set employee object to premierpro project object using
setter method
pr.setEmployees(premployees);
```

```
// adding employees to after effects project list
afemployees.add(bob);
afemployees.add(charli);

//Set employee object to after effects project object using
setter method
af.setEmployees(afemployees);

//save the employees to the database
session.save(alex);
session.save(bob);
session.save(charli);

//save the projects to database
session.save(pr);
session.save(af);
```

ManyToMany.java

```
import java.util.ArrayList;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToMany {

        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Session session = null;
                try {
//                      Create session Factory

                        sessionFactory = new Configuration()
                                .configure()
                                .addAnnotatedClass(Employee.class)
                                .addAnnotatedClass(Project.class)
                                .buildSessionFactory();
//                      Create session
                        session = sessionFactory.openSession();

//                      Create Transaction
                        Transaction transaction = session.beginTransaction();

//                      CRUD Operation
                        Employee alex = new Employee(1, "Alex",
"alex@gmail.com");
                        Employee bob = new Employee(2      , "Bob",
"bob@gmail.com");
                        Employee charli = new Employee(3, "charli",
"charli@gmail.com");

//                      Creating Projects
                        Project pr = new Project(1, "Premierpro");
                        Project af = new Project(2, "Aftereffects");

                        ArrayList<Employee> premployees = new
ArrayList<Employee>();
                        ArrayList<Employee> afemployees = new
ArrayList<Employee>();

//                      Assigning employees to premier pro project
                        premployees.add(alex);
                        premployees.add(bob);
                        premployees.add(charli);
                        pr.setEmployees(premployees);
//                              Assigning employees to After effects project
                        afemployees.add(bob);
                        afemployees.add(charli);
                        af.setEmployees(afemployees);

                        session.save(alex);
                        session.save(bob);
                        session.save(charli);

                        session.save(pr);
                        session.save(af);

                        transaction.commit();
                } finally {
//                              Closing Resources
                        session.close();
                        sessionFactory.close();

                }
        }

}
```

Output:

Project Table

id	name
1	Premierpro
2	Aftereffects

Employee Table

id	email	name
1	alex@gmail.com	Alex
2	bob@gmail.com	Bob
3	charli@gmail.com	charli

Employee_project Table

proj_id	emp_id
1	1
1	2
1	3
2	2
2	3

Scenario 2:
Now let’s see how to fetch the data of the employees and the projects they are working in
- First get the employee object
- Next using employee object get the project object
- Print employee and list of projects they are working on

Logic:

```
Employee employee = session.get(Employee.class, 1); //get the employee object using session.get()

List<Project> projects = employee.getProjects(); //get the projects that employee is working on using getter method of projects

System.out.println(employee ); //print employee object

for (Project project : projects) {
     System.out.println(project);
} // print all the projects that employee is working
```

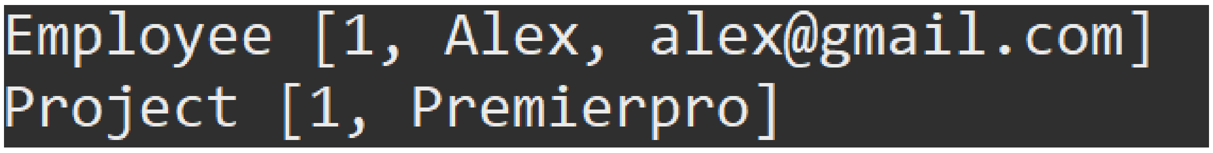
ManyToManyFetchData.java

```
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToManyFetchData {
```

```java
public static void main(String[] args) {
        SessionFactory sessionFactory = null;
        Session session = null;

        try {
                // Create session Factory

                sessionFactory = new Configuration()
                                .configure()
                                .addAnnotatedClass(Employee.class)
                                .addAnnotatedClass(Project.class)
                                .buildSessionFactory();
                // Create session
                session = sessionFactory.openSession();

                // Create Transaction
                Transaction transaction = session.beginTransaction();

                // CRUD Operation
                Employee employee = session.get(Employee.class, 1);
                List<Project> projects = employee.getProjects();

                System.out.println(employee );

                for (Project project : projects) {
                        System.out.println(project);
                }

                transaction.commit();
        } finally {
                // Closing Resources
                session.close();
                sessionFactory.close();

        }
}
```

```
}
```

Output:



```
Employee [1, Alex, alex@gmail.com]
Project [1, Premierpro]
```

Now let’s try to fetch project object and next employees working on that projects

Logic:

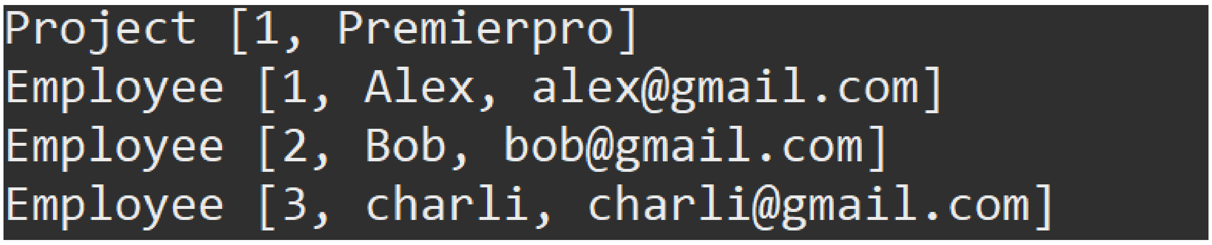
```
Project project = session.get(Project.class, 1); // get the project object using session.get()

List<Employee> employees = project.getEmployees(); //get employee object using getter method of employee object present in project class

System.out.println(project);

for (Employee employee : employees) {
     System.out.println(employee);
}
```

ManyToManyFetchData.java

```
import java.util.List;

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

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToManyFetchData {

	public static void main(String[] args) {
		SessionFactory sessionFactory = null;
		Session session = null;
```

```
        try {
            // Create session Factory

            sessionFactory = new Configuration()
                        .configure()
                        .addAnnotatedClass(Employee.class)
                        .addAnnotatedClass(Project.class)
                        .buildSessionFactory();
            // Create session
            session = sessionFactory.openSession();

            // Create Transaction
            Transaction transaction = session.beginTransaction();

            // CRUD Operation
            Project project = session.get(Project.class, 1);
            List<Employee> employees = project.getEmployees();

            System.out.println(project);

            for (Employee employee : employees) {
                System.out.println(employee);
            }

            transaction.commit();
        } finally {
            // Closing Resources
            session.close();
            sessionFactory.close();

        }
    }

}
```

Output:



```
Project [1, Premierpro]
Employee [1, Alex, alex@gmail.com]
Employee [2, Bob, bob@gmail.com]
Employee [3, charli, charli@gmail.com]
```

Scenario 4:
Now let’s try to delete the project from the project table

Logic:

```
Project project = session.get(Project.class, 1); // get the project that needs to be deleted using session.get()
session.delete(project); // using delete() delete the project
```

ManyToManyDelete.java

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.tap.model.Employee;
import com.tap.model.Project;

public class ManyToManyDelete {

	public static void main(String[] args) {
		SessionFactory sessionFactory = null;
		Session session = null;

		try {
			// Create session Factory

			sessionFactory = new Configuration()
					.configure()
					.addAnnotatedClass(Employee.class)
					.addAnnotatedClass(Project.class)
					.buildSessionFactory();
			// Create session
			session = sessionFactory.openSession();
```

```
                                // Create Transaction
                                Transaction transaction = session.beginTransaction();

                                // CRUD Operation
                                Project project = session.get(Project.class, 1);
                                session.delete(project);


                                transaction.commit();
                        } finally {
                                // Closing Resources
                                session.close();
                                sessionFactory.close();

                        }
                }

}
```

Output:
Project Table

id	**name**
2	Aftereffects

Employee Table

id	**email**	**name**
1	alex@gmail.com	Alex
2	bob@gmail.com	Bob
3	charli@gmail.com	charli

Employee_project Table

proj_id	emp_id
2	2
2	3