TP 6 des architectures logicielles
Séance 6 : Architecture n-tiers avec du JPA avec plusieurs entités

1 Préparation de l'environnement Eclipse

1. Environment Used

- JDK 7 (Java SE 7)
- JPA 2.0
- Eclipse
- MySQL server 5.6
- EclipseLink

2. Description du projet

- Nous allons voir comment manipuler les relations entre plusieurs entités ainsi que le mapping et la persistance sur une base de données MySQL.

1 La persistance des entités relationnelles

1. Créer un nouveau projet JPA : « People »
2. Créer un nouveau package `ma.ac.model`
3. Ajouter les trois classes entity suivantes :

```
package ma.ac.model;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

@Entity
public class Family {
	@Id
	@GeneratedValue(strategy = GenerationType.TABLE)
	private int id;
	private String description;

	@OneToMany(mappedBy = "family")
	private final List<Person> members = new ArrayList<Person>();

	public int getId() {
		return id;
	}
```

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

	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public List<Person> getMembers() {
		return members;
	}

}

package ma.ac.model;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Transient;

@Entity
public class Person {
	@Id
	@GeneratedValue(strategy = GenerationType.TABLE)
	private String id;
	private String firstName;
	private String lastName;

	private Family family;

	private String nonsenseField = "";

	private List<Job> jobList = new ArrayList<Job>();

	public String getId() {
		return id;
	}

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

	public String getFirstName() {
		return firstName;
	}

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

```
	// Leave the standard column name of the table
	public String getLastName() {
		return lastName;
	}

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

	@ManyToOne
	public Family getFamily() {
		return family;
	}

	public void setFamily(Family family) {
		this.family = family;
	}

	@Transient
	public String getNonsenseField() {
		return nonsenseField;
	}

	public void setNonsenseField(String nonsenseField) {
		this.nonsenseField = nonsenseField;
	}

	@OneToMany
	public List<Job> getJobList() {
		return this.jobList;
	}

	public void setJobList(List<Job> nickName) {
		this.jobList = nickName;
	}

}

package ma.ac.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Job {
	@Id
	@GeneratedValue(strategy = GenerationType.TABLE)
	private int id;
	private double salery;
	private String jobDescr;

	public int getId() {
		return id;
	}

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

	public double getSalery() {
```

```
        return salery;
    }

    public void setSalery(double salery) {
        this.salery = salery;
    }

    public String getJobDescr() {
        return jobDescr;
    }

    public void setJobDescr(String jobDescr) {
        this.jobDescr = jobDescr;
    }

}
```

4. Customise your file "persistence.xml" in "src/META-INF". Remember to change the path to the database.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
        <persistence-unit name="testjpa">
                <class>de.vogella.jpa.eclipselink.model.Family</class>
                <class>de.vogella.jpa.eclipselink.model.Person</class>
                <class>de.vogella.jpa.eclipselink.model.Job</class>
                <properties>
 <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
 <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/people" />
 <property name="javax.persistence.jdbc.user" value="root" />
 <property name="javax.persistence.jdbc.password" value="" />
 <property name="eclipselink.ddl-generation" value="create-tables" />
 <property name="eclipselink.ddl-generation.output-mode" value="database" />
 </properties>
        </persistence-unit>
</persistence>
```

5. Ajouter la classe de tests suivante :

```
package ma.ac.main;


import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;


import de.vogella.jpa.eclipselink.model.Family;
import de.vogella.jpa.eclipselink.model.Person;


public class JpaTest {

        private static final String PERSISTENCE_UNIT_NAME = "people";
        private static EntityManagerFactory factory;


        public static void main(String[] args) throws Exception {
                JpaTest.setUp();
```

```
	}

	public static void setUp() throws Exception {
		factory = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);
		EntityManager em = factory.createEntityManager();

		// Begin a new local transaction so that we can persist a new entity
		em.getTransaction().begin();

		// read the existing entries
		Query q = em.createQuery("select m from Person m");
		// Persons should be empty

		// do we have entries?
		boolean createNewEntries = (q.getResultList().size() == 0);

		// No, so lets create new entries
		// if (createNewEntries) {

			Family family = new Family();
			family.setDescription("Family for the Knopfs");
			em.persist(family);
			for (int i = 0; i < 40; i++) {
				Person person = new Person();
				person.setFirstName("Jim_" + i);
				person.setLastName("Knopf_" + i);
				em.persist(person);
				// now persists the family person relationship
				family.getMembers().add(person);
				em.persist(person);
				em.persist(family);
			}
		// }

		// Commit the transaction, which will cause the entity to
		// be stored in the database
		em.getTransaction().commit();

		// It is always good practice to close the EntityManager so that
		// resources are conserved.
		em.close();

	}
}


public void checkAvailablePeople() {

		// now lets check the database and see if the created entries are there
		// create a fresh, new EntityManager
		EntityManager em = factory.createEntityManager();

		// Perform a simple query for all the Message entities
		Query q = em.createQuery("select m from Person m");

		// We should have 40 Persons in the database
		List<Person> todoList = q.getResultList();
			for (Person todo : todoList) {
				System.out.println(todo);
			}

		em.close();
	}


	public void checkFamily() {
		EntityManager em = factory.createEntityManager();
		// Go through each of the entities and print out each of their
		// messages, as well as the date on which it was created
		Query q = em.createQuery("select f from Family f");
```

```
		// We should have one
		List<Family> todoList = q.getResultList();
			for (Family todo : todoList) {
				System.out.println(todo);
			}
		em.close();
	}

	public void deletePerson() {
		EntityManager em = factory.createEntityManager();
		// Begin a new local transaction so that we can persist a new entity
		em.getTransaction().begin();
		Query q = em
				.createQuery("SELECT p FROM Person p WHERE p.firstName = :firstName
AND p.lastName = :lastName");
		q.setParameter("firstName", "Jim_1");
		q.setParameter("lastName", "Knopf_!");
		Person user = (Person) q.getSingleResult();
		em.remove(user);
		em.getTransaction().commit();
		Person person = (Person) q.getSingleResult();
		// Begin a new local transaction so that we can persist a new entity

		em.close();

	}
```

6. Vérifier la pertinence des résultats :