Java Persistence API (la suite)

Quelques liens :

- API JPA 3.0 (JEE 10) [1]
- Tutorial JPA (JEE 9) [2]
- Une documentation sur JPQL [3]
- La page de Wikipedia [4]
- De très bons exemples [5]

1 L'héritage dans JPA

1.1 Héritage avec table unique

Nous allons étudier la représentation d'un arbre d'héritage dans une structure relationnelle. Définissons trois classes : une pour les UE, une autre (qui hérite de la première) pour les UE de master et une troisième (qui hérite également de la première) pour les UE de Licence :

```java
package myapp.jpa.model;

import jakarta.persistence.Basic;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class UE {

    @Id
    private String code;

    @Basic
    private int ects;

}
```

Les UE de Master :

1. https://jakarta.ee/specifications/platform/10/apidocs//jakarta/persistence/package-summary.html
2. https://eclipse-ee4j.github.io/jakartaee-tutorial/#persistence
3. http://download.oracle.com/docs/cd/E13189_01/kodo/docs40/full/html/ejb3_overview_query.html
4. http://en.wikipedia.org/wiki/Java_Persistence_API
5. http://www.java2s.com/Tutorial/Java/0355__JPA/Catalog0355__JPA.htm

```java
package myapp.jpa.model;

import jakarta.persistence.Basic;
import jakarta.persistence.Entity;

import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;

@Entity
@Data
@NoArgsConstructor
@EqualsAndHashCode(callSuper = true)
public class MasterUE extends UE {

    @Basic
    private String masterName;

    public MasterUE(String code, int ects, String masterName) {
        super(code, ects);
        this.masterName = masterName;
    }
}
```

Les UE de Licence :

```java
package myapp.jpa.model;

import jakarta.persistence.Basic;
import jakarta.persistence.Entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(callSuper = true)
public class LicenceUE extends UE {

    @Basic
    private String description;

    public LicenceUE(String code, int ects, String description) {
        super(code, ects);
        this.description = description;
    }

}
```

Travail à faire : Faites un test unitaire en créant (en base) une instance de chaque classe. Analysez la table unique créée par JPA. Utilisez ensuite la méthode `findAll` pour chaque classe et vérifiez que vous obtenez 3 UE, 1 UE de Master et 1 UE de Licence.

1.2 Héritage avec table de jointure

Dans cette deuxième stratégie, les classes sont représentées par plusieurs tables mais les propriétés communes sont représentées une seule fois :

```
...

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class UE {

    ...

}
```

Travail à faire : Analysez les tables créées par JPA.

1.3 Héritage avec tables séparées

Dans cette troisième stratégie, les classes sont représentées par plusieurs tables spéparées :

```
...

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class UE {

    ...

}
```

Travail à faire : Analysez les tables créées par JPA.

2 Requêtes construites par programmation

Jusqu'à maintenant nous avons directement utilisé des requêtes JPQL sous la forme de chaîne de caractères éventuellement paramétrées. A partir de JPA 2 il est possible de construire dynamiquement une requête bien typée à partir d'une API.

Travail à faire : En vous aidant de ce chapitre du tutoriel JEE 9 [6], améliorez la méthode `findAll` en supprimant le paramètre `query` que nous utilisions avant.

```
public <T> Collection<T> findAll(Class<T> clazz) {
    ...
}
```

Tentez ensuite de construire une méthode qui renvoie les UE de Master à 6 crédits. Pour ce faire, vous devez ajouter un critère à votre requête (méthode `where` appliquée à l'instance de la classe `CriteriaQuery<MasterUE>`.

3 Utiliser Spring Data

Maintenant que vous connaissez un peu mieux le fonctionnement de JPA, nous allons utiliser Spring-data pour supprimer le code des classes Dao.

6. https://eclipse-ee4j.github.io/jakartaee-tutorial/#using-the-criteria-api-to-create-queries

1) Commencez par créer l'interface ci-dessous :

```
package myapp.jpa.dao;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import myapp.jpa.model.Person;

public interface PersonRepository extends JpaRepository<Person, Long> {

    List<Person> findByFirstName(String name);

    List<Person> findByFirstNameLike(String name);

}
```

2) Ajoutez ensuite une clause à votre classe de configuration pour activer le traitement des `JpaRepository` :

```
package myapp.jpa.dao;

import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import myapp.jpa.model.Person;

@Configuration
@EntityScan(basePackageClasses = Person.class)
// NOUVEAU
@EnableJpaRepositories(basePackageClasses = SpringDaoConfig.class)
public class SpringDaoConfig {

}
```

3) Terminez par une classe de test :

```java
package myapp.jpa.dao;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.Date;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import myapp.jpa.dao.PersonRepository;
import myapp.jpa.model.Person;

@SpringBootTest
public class TestPersonRepository {

    @Autowired
    PersonRepository dao;

    @Test
    public void testRepository() {
        // détruire les instances
        dao.deleteAll();
        assertFalse(dao.findAll().iterator().hasNext());
        // créer une instance
        var p = new Person("AAA", new Date());
        dao.save(p);
        // tester une instance
        var op = dao.findById(p.getId());
        assertTrue(op.isPresent());
        p = op.get();
        assertEquals("AAA", p.getFirstName());
    }

}
```

Remarque : Vous noterez que `PersonRepository` est une interface mais que nous n'avons pas d'implémentation. C'est Spring qui se charge de construire l'implémentation directement à partir des noms de méthodes déclarées dans l'interface. Vous remarquerez le `Like` dans le nom d'une méthode. C'est la base de la technologie **Spring-data** (plus d'information [7]). Vous pouvez lire la documentation sur les méthodes de requêtes [8].

Travail à faire :

- Enrichissez votre classe de test pour tester les méthodes `findByFirstName` et `findByFirstNameLike`.
- Avec cette documentation [9], ajoutez une méthode `deleteLikeFirstName(String pattern)` à la classe `PersonRepository` (utilisez `@Query` et `@Transactional`).

7. https ://docs.spring.io/spring-data/jpa/docs/current/reference/html/
8. https ://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.query-creation
9. https ://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.modifying-queries