Internet Applications Design and Implementation

(Lecture 5 - JPA Associations)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt),
contributions from Beatriz Moreira, APDC-INV 2017/2018)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Outline

- JPA Associations
- Optimization of data fetching

Internet Applications Design and Implementation 2020 - 2021

(Lecture 5 - Part 1 - JPA Associations)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
       @ManyToOne
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @OneToMany
        val books:Set<Book>
)
```

```
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val fantasy = Category(0, “Fantasy", emptySet<Book>())
categories.save(fantasy)
val lor = Book(0,"Lord of the Rings", fantasy)
books.save(lor)
```

```
call next value for hibernate_sequence
insert into category (name, id) values (?, ?)
binding parameter [1] as [VARCHAR] - [Fantasy]
binding parameter [2] as [BIGINT] - [1]
call next value for hibernate_sequence
insert into book (kind_id, name, id) values (?, ?, ?)
binding parameter [1] as [BIGINT] - [1]
binding parameter [2] as [VARCHAR] - [Lord of the Rings]
binding parameter [3] as [BIGINT] - [2]
```

JPA Associations

```java
@Entity
@Table(name = "book_category")
public class BookCategory {
    private int id;
    private String name;
    private Set<Book> books;

    public BookCategory(){ … }

    public BookCategory(String name) {…}

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() {…}

    public void setName(String name) {…}

    @OneToMany(mappedBy = "bookCategory",
               cascade = CascadeType.ALL)
    public Set<Book> getBooks() { … }

    public void setBooks(Set<Book> books) { … }
}
```

```java
@Entity
public class Book{
    private int id;
    private String name;
    private BookCategory bookCategory;

    public Book() {}

    public Book(String name) { this.name = name;}

    public Book(String name, BookCategory bookCategory) { … }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() { … }

    public void setName(String name) { … }

    @ManyToOne
    @JoinColumn(name = "book_category_id")
    public BookCategory getBookCategory() { … }

    public void setBookCategory(BookCategory bookCategory) {…}
}
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @ManyToOne
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @OneToMany
        val books:Set<Book>
)
```

```
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val book = books.findById(lor.id)
logger.info(book.get().kind.toString())
```

```
select book0_.id as id1_0_0_,
      book0_.kind_id as kind_id3_0_0_,
      book0_.name as name2_0_0_,
      category1_.id as id1_1_1_,
      category1_.name as name2_1_1_
from book book0_
left outer join category category1_ on
   book0_.kind_id=category1_.id where book0_.id=?
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @ManyToOne
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @OneToMany
        val books:Set<Book>
)
```

```
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val kind = categories.findById(fantasy.id)
```

```
select
   category0_.id as id1_1_0_,
   category0_.name as name2_1_0_
from category category0_
where category0_.id=?
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,

        @OneToMany
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @ManyToOne
        val books:Set<Book>
)
```

```
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val kind = categories.findById(fantasy.id)
logger.info(kind.toString())
```

```
failed to lazily initialize a collection of role:
com.demo.Category.books
```

JPA Associations

```kotlin
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @ManyToOne
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @OneToMany(fetch = FetchType.EAGER)
        val books:Set<Book>
)
```

```kotlin
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val kind = categories.findById(fantasy.id)
logger.info(kind.toString())
```

```
Optional[Category(id=1, name=Fantastic, books=[])]
```

```
select
category0_.id as id1_1_0_,
category0_.name as name2_1_0_,
books1_.category_id as category1_2_1_,
book2_.id as books_id2_2_1_,
book2_.id as id1_0_2_,
book2_.kind_id as kind_id3_0_2_,
book2_.name as name2_0_2_,
category3_.id as id1_1_3_,
category3_.name as name2_1_3_
from category category0_
left outer join category_books books1_ on category0_.id=books1_.category_
left outer join book book2_ on books1_.books_id=book2_.id
left outer join category category3_ on book2_.kind_id=category3_.id
where category0_.id=?
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        @Column(nullable = false)
        val name:String,
        @ManyToOne
        val kind:Category
)
@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        @Column(nullable = false)
        val name:String,
        @OneToMany(cascade = arrayOf(CascadeType.ALL), mappedBy = "kind", fetch = FetchType.EAGER)
        var books:List<Book>
)
```

```
val fantasy = Category(0, "Fantasy", emptyList<Book>())
val lor = Book(0,"Lord of the Rings", fantasy)
val silm = Book(0,"Silmarillion", fantasy)
fantasy.books = listOf(lor,silm)
categories.save(fantasy)
```

```
insert into category (name, id) values (?, ?)
binding parameter [1] as [VARCHAR] - [Fantasy]
binding parameter [2] as [BIGINT] - [1]
insert into book (kind_id, name, id) values (?, ?, ?)
binding parameter [1] as [BIGINT] - [1]
binding parameter [2] as [VARCHAR] - [Lord of the Rings]
binding parameter [3] as [BIGINT] - [2]
insert into book (kind_id, name, id) values (?, ?, ?)
binding parameter [1] as [BIGINT] - [1]
binding parameter [2] as [VARCHAR] - [Silmarillion]
binding parameter [3] as [BIGINT] - [3]
```

```
Category(id=1, name=Fantasy, books=[{ Lord of the Rings, Fantasy }, { Silmarillion, Fantasy }])
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        @Column(nullable = false)
        val name:String,
        @ManyToOne
        val kind:Category
)
@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        @Column(nullable = false)
        val name:String,
        @OneToMany(cascade = arrayOf(CascadeType.ALL), mappedBy = "kind", fetch = FetchType.EAGER)
        var books:List<Book>
)
```

```
val kind = categories.findById(fantasy.id)
logger.info(kind.toString())
logger.info(kind.get().books.size.toString())
for (b in kind.get().books) {
    logger.info(b.toString())
    logger.info(b.kind.toString())
}
```

```
select
    category0_.id as id1_1_0_,
    category0_.name as name2_1_0_,
    books1_.kind_id as kind_id3_0_1_,
    books1_.id as id1_0_1_,
    books1_.id as id1_0_2_,
    books1_.kind_id as kind_id3_0_2_,
    books1_.name as name2_0_2_
from category category0_
left outer join book books1_ on
    category0_.id=books1_.kind_id where category0_.id=?
```

```
Category(id=1, name=Fantasy, books=[{ Lord of the Rings, Fantasy }, { Silmarillion, Fantasy }])
```

Eager and Lazy

- Evaluation modes - transferring objects to memory

```
@OneToMany(mappedBy = "course", cascade = CascadeType.ALL, fetch = FetchType.EAGER)
private Set<Enrollment> enrollments;
```

- Eager: transfers all objects related to the root

```
@OneToMany(mappedBy = "course", cascade = CascadeType.ALL, fetch = FetchType.Lazy)
private Set<Enrollment> enrollments;
```

- Lazy: transfers object when needed
  - Needs a transactional environment

```
@Component
public class ProfessorService {

    @Autowired
    CourseRepository courseRepository;

    @Autowired
    ProfessorRepository professors;

    @Transactional
    public void addCourses(String name, String... courses) {
        Professor p = professors.findByName(name);

        Set<Course> cs = p.getCourses();
        for(String c: courses)
            cs.add(courseRepository.findByName(c).get(0));
        professors.save(p);
    }
```

Many To Many

```java
@Entity
public class Course {

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

    private String name;

    private int credits;

    @OneToMany(mappedBy = "course",
               cascade = CascadeType.ALL,
               fetch = FetchType.EAGER)
    private Set<Enrollment> enrollments;

    @ManyToMany(mappedBy = "courses")
    private Set<Professor> professors;
}
```

```java
@Entity
public class Professor {

    public Professor(String name) {
        this.setName(name);
    }

    public Professor() {}

    @javax.persistence.Id
     @GeneratedValue(strategy=GenerationType.AUTO)
    private long Id;

    private String name;

    @ManyToMany
    private Set<Course> courses;
}
```

creates *professor_courses* table in a bidirectional relation

https://en.wikibooks.org/wiki/Java_Persistence/ManyToMany

http://www.objectdb.com/java/jpa/getting/started

https://hellokoding.com/jpa-many-to-many-extra-columns-relationship-mapping-example-with-spring-boot-maven-and-mysql/

Internet Applications Design and Implementation 2020 - 2021

(Lecture 5 - Part 2 -Optimization of data fetching)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

N+1 query problem

```
@Entity
@Table(name = "book_category")
public class BookCategory {
    private int id;
    private String name;
    private Set<Book> books;

    public BookCategory(){ … }

    public BookCategory(String name) {…}

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() {…}

    public void setName

    @OneToMany(mappedBy
               cascade
    public Set<Book> getBooks() { … }

    public void setBoo
}
```

```
@Entity
public class Book{
    private int id;
    private String name;
    private BookCategory bookCategory;

    public Book() {}

    public Book(String name) { this.name = name;}

    public Book(String name, BookCategory bookCategory) { … }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() { … }

    @JoinColumn(name = "book_category_id")
    public BookCategory getBookCategory() { … }

    kCateg
```

```
@Repository
public interface BookCategoryRepository extends JpaRepository<BookCategory, Integer>{}
```

```
for( BookCategory c: categoryRepo.findAll() )
 for( Book b: c.books )
  …
```

```
select * from book_category;
select * from book where bookCategory = ?
…
select * from book where bookCategory = ?
```

N+1 query problem

```
@Entity
@Table(name = "book_category")
public class BookCategory {
    private int id;
    private String name;
    private Set<Book> books;

    public BookCategory(){ … }

    public BookCategory(String name) {…}

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() {…}

    public void setName(String name) { }

    public Set<Book> getBooks() { … }

}
```

```
@Repository
public interface BookCategoryRepository extends JpaRepository<BookCategory, Integer>{}
```

```
for( Book b : BookRepo.findAll() )
 System.out.println( b.toString() + b.bookCategory.toString())
```

```
@Entity
public class Book{
    private int id;
    private String name;
    private BookCategory bookCategory;

    public Book() {}

    public Book(String name) { this.name = name;}

    public Book(String name, BookCategory bookCategory) { … }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() { … }

    @JoinColumn(name = "book_category_id")
    public BookCategory getBookCategory() { … }

    setBookCategory(BookCategory bookCategory) {…}
```

```
select * from book;
select * from bookCategory where bookCategory = ?
…
select * from bookCategory where bookCategory = ?
```

Prefetching

- Instead of using global fetching strategies, one can define how objects/ collections related to one particular entity are loaded to memory

```
fun appointmentsOfPet(id: Long): List<AppointmentDAO> {
    val pet = pets.findById(id)
                  .orElseThrow { NotFoundException("There is no Pet with Id $id") }
    return pet.appointments
}
```

Two queries

```
interface PetRepository : JpaRepository<PetDAO, Long> {
    @Query("select p from PetDAO p inner join fetch p.appointments where p.id = :id")
    fun findByIdWithAppointment(id:Long) : Optional<PetDAO>
}
```

```
fun appointmentsOfPet(id: Long): List<AppointmentDAO> {
    val pet = pets.findByIdWithAppointment(id)
                  .orElseThrow { NotFoundException("There is no Pet with Id $id") }
    return pet.appointments
}
```

Appointment's collections is fetched and loaded in one single query

Custom queries for efficient execution

- ORM relations can produce a large number of queries… which can be optimized by means of a single query being dispatched to the DB

```
@Query(" select s from Student s "+
     "   inner join fetch s.courses en "+
     "   inner join en.course c "+
     " where c.name = :name")
List<Student> searchByCourse(@Param("name") String name);
```

- Optimization may work by summarising data

```
@Query("select new ciai.model.StudentSummary(s.name,s.age) from Student s")
List<StudentSummary> summaryStudents();
```

Java Persistence Query Language (JPQL ⊆ HQL)

- Simple custom queries

```
@Query("select s from Student s where s.name like CONCAT(?,'%')")
List<Student> search(String name);
```

- Complex custom queries

```
@Query(" select s from Student s "+
     "   inner join s.courses en "+
     "   inner join en.course c "+
     " where c.name = :name")
List<Student> searchByCourse(@Param("name") String name);
```

http://www.objectdb.com/java/jpa/getting/started

Now it’s time for you to research and experiment…