Laborator 9: JPA, Hibernate

FETCH TYPES

Eager se vor încărca toate entitățile dependente.
Poate avea impact asupra performanței aplicției. Spre exemplu, într-o relație de tip @OneToMany, entitatea părinte poate avea un număr foarte mare de entități fiu. Dacă sunt necesare doar date referitoare la entitatea părinte eager loading nu este necesar.

Lazy se va încărca entitatea părinte. Entitățile dependente vor fi încărcate la cerere.

Este recomandat ca în view-urile de tip master să fie utilizată lazy loading iar în view-urile de tip detail să fie utilizată eager loading.

Valori default:

@OneToOne	FetchType.EAGER
@OneToMany	FetcheType.LAZY
@ManyToOne	FetchType.EAGER
@ManyToMany	FetcheType.LAZY

Pentru a folosi alte valori se poate specifica în adnotări atributul fetch. Spre exemplu:
@ManyToOne(**fetch=FetchType.LAZY**)

Observații Lazy Loading:

Atunci când se solicită încărcarea datelor (lazy loading) și sesiunea Hibernate este închisă, Hibernate va arunca o excepție.

Datele se pot încărca utilizând:
- session.get
- metode de getter
- Hibernate query HQL

1. Importați proiectul LAB9_START. Creați în clasa apbdoo.laboratorul9. Completați metoda testLazyLoading a clasei apbdoo.laboratorul9.repositories.CategorieRepository

```
Categorie categorie =
categorieRepository.findByDenumire("comedie");
assertFalse(Hibernate.isInitialized(categorie.getFilme()));
```

2. Modificați relația Categorie – Filme astfel încât fetch Type se fie Eager:

```
fetch = FetchType.EAGER
```

3. Dezactivați testul TestEagerLoading. Adnotați meotda @Ignore.

4. Adăugați testul:

```
Categorie categorie =
categorieRepository.findByDenumire("comedie");
assertTrue(Hibernate.isInitialized(categorie.getFilme()));
```

Query methods

Finder Methods:

Interfețele care extind CrudRepository pot include metode care folosesc următoarele convenții de notație fiindByAttribut**Keyword**Attribute

Keyword poate fi And, Or, Like, OrderBy, GraterThan etc.

Spre exemplu pentru o entitate cu atribute name, description, value se pot apela metodele:

findByName(String name)
findByName**And**Description(String name, String description)
findByNameLike(String name)
findByValueGraterThan(Double value)

Spring Data va genera automat implementari pentru aceste metode.

Mai multe exemple pot fi consultate la:
https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods
https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repository-query-keywords

5. Adaugți în interfața FilmRepository metodele:

```
List<Film> findByTitluLike(String titlu);
List<Film> findByTitluAndMinuteGreaterThan(String titlu,Integer
id);
List<Film> findByTitluContains(String titlu);
List<Film> findByMinuteBetween(Integer minute1, Integer minute2);
List<Film> findByIdIn(List<Long> ids);
```

6. Adăugați o clasă test pentru aceste metode.

```
filme = filmRepository.findByIdIn(Arrays.asList(1L,2L));
log.info("findByIdIn ...");
filme.forEach(film -> log.info(film.getId() + " " +
film.getTitlu()));
```

PagingAndSorting

PagingAndSortingRepository extinde interfața CRUDRepository și adăugă metode care permit paginarea, metode de tip fiind care primesc ca argument obiecte de tipul Pageable sau Sort.

Pageable este o interfață care transmite informații despre paginare metodelor de tip find și are implementările:

PageRequest, AbstractPageRequest, QPageRequest.

PageRequest Un obiect de tip PageRequest poate fi instanțiat prin utilizarea metodei **of** care argumente pentru specificarea numărului paginii solicitate precum și a dimensiunii acesteia. Paginile sunt indexate începând cu valoarea 0.

Sort are metodele by(String proprietate) si ascending() descending() pentru sortare crescătoare sau descrescătoare**.** Default sortarea se face crescător.

7. Modificați FilmRepository astfel încât să extindă interfața PagingAndSortingRepository

8. Creați un test care va obține o pagină care să conțină 2 filme.

```
Pageable pageable = PageRequest.of(0, 2);
Page<Film> filmPage= filmRepository.findAll(pageable);
filmPage.forEach(film ->log.info(film.getTitlu()));
```

9. Creați un test pentru a obține lista filmelor sortate descrescător după titlu.

```
filmRepository.findAll(Sort.by("titlu").descending()).
               forEach(film -> log.info(film.getTitlu()));

filmRepository.findAll(Sort.by("minute").ascending().
               by("titlu").descending()).
               forEach(film -> log.info(film.getTitlu()));
```

10. Utilizați metoda of a clasei PageRequest cu trei argumente: pageId, pageSize si Sort pentru a afișa prima pagină din lista filmelor sortate descrescător după titlu, cu 2 filme.

```
Pageable pageable = PageRequest.of(0, 2, sort);
```

11. Adăugați metodei findByIdIn un argument de tipul Pageable. Testați.

```
Pageable pageable = PageRequest.of(0, 2, sort);
```

12. Adăugați în FilmRepository o metodă pentru a lista doar titlurile și durata filmelor. Testați.

```
@Query("select titlu, minute from Film")
List<Object[]> findAllpartial();
```

13. Creați o metoda care va căuta filmele după titlu. Testați.

```
@Query("from Film where titlu = :titlu")
List<Film> findByTitlu(@Param("titlu") String titlu);
```

14. Adăugați în FilmService metoda:

```
Page<Film> getPage(Pageable pageable);
```

15. Adăugați un Controller care să servească view-ul filmpage.html.