Spring Data

Yalın hibernate’in problemleri

- Hala fazla kod var
- Çok low level detay gerekiyor
- Configuration işlemleri çok zor

Spring Data Repositories

Spring Data Repositories

- Bütün veri tabanı ile ilgili detayları abstract etmenizi sağlar, ve session management, transaction, hibernate gibi detaylarlardan sizi kurtarır
- Sadece interface ve template function'lar ile veri tabanına erişmenizi sağlar
- JPA annotation'ları ile işaretlenmiş class'lar ile sorunsuz şekilde çalışır.

```
public interface UserRepository extends JpaRepository<User, Long> {
}
```

- Interface üzerine @Component veya @Repository yazmanıza gerek yok
- Interface'in implementation'ını yapmanıza gerek yok

JpaRepository'deki mevcut fonksiyonlar

Function Name	Description	Return Value
findAll()	Gets all the entity	List of given class
findById(ID)	Finds one item by id	Single item of given class
save(EntityType)	Saves the given entity to database	Saved entity item
saveAll(Iterable<EntityType>)	Saves all of the given entities	List of saved entities
existsById(ID)	Checks if an entity with given ID exists	boolean
count()	Gets the number of items in that table	long
deleteById(ID)	Deletes the entity with given ID	void
delete(EntityType)	Deletes the entity	void
deleteAll(Iterable<EntityType>)	Deletes all given entities	void
deleteAll()	Empties the table	void

```java
@Service
public class UserService {

    private final UserRepository userRepository;

    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public void saveUser(final User user) {
        userRepository.save(user);
    }

}
```

Derived Query'ler

- Repository'lere belli formatlarda function tanımları yazarak hemen hemen her query'nin çalıştırılmasını sağlayabiliriz. Bunu yapmak için de findBy, getBy gibi prefix'ler ile başlayan function tanımları yazmamız gerekmekte. Daha sonrasında ise ne ile query etmek istiyorsak onu yazmamız gerekiyor.

```
public interface UserRepository extends JpaRepository<User, Long> {
    List<User> findByName(String name);
}
```

- Daha detaylı bilgi için buraya ve buraya bakınız

Derived Query'ler

- find..By, get..By, query..By, read..By ile başlayan fonskyionlar ile yandaki expression'lar birleştirelerek karmaşık query'lerin yazılması sağlanabilir. 4 fonksiyon da aynıdır. Genellikle find kullanılıyor.

Logical keyword	Keyword expressions
AND	And
OR	Or
AFTER	After, IsAfter
BEFORE	Before, IsBefore
CONTAINING	Containing, IsContaining, Contains
BETWEEN	Between, IsBetween
ENDING_WITH	EndingWith, IsEndingWith, EndsWith
EXISTS	Exists
FALSE	False, IsFalse
GREATER_THAN	GreaterThan, IsGreaterThan
GREATER_THAN_EQUALS	GreaterThanEqual, IsGreaterThanEqual
IN	In, IsIn
IS	Is, Equals, (or no keyword)
IS_EMPTY	IsEmpty, Empty
IS_NOT_EMPTY	IsNotEmpty, NotEmpty
IS_NOT_NULL	NotNull, IsNotNull
IS_NULL	Null, IsNull
LESS_THAN	LessThan, IsLessThan
LESS_THAN_EQUAL	LessThanEqual, IsLessThanEqual
LIKE	Like, IsLike
NEAR	Near, IsNear
NOT	Not, IsNot
NOT_IN	NotIn, IsNotIn
NOT_LIKE	NotLike, IsNotLike
REGEX	Regex, MatchesRegex, Matches
STARTING_WITH	StartingWith, IsStartingWith, StartsWith
TRUE	True, IsTrue
WITHIN	Within, IsWithin

Gelen query sonucunu sıralama

- Her derived query sonuna OrderBy...[ASC,DESC] ifaredelerini ekleyerek belli bir alana göre sıralanmasınsını sağlayabiliriz.
- Aynı zamanda tüm derived query'lere bir Sort objesi vererek de sorting işlemi yapılabilir

```java
public interface BookRepository extends JpaRepository<Book, Long> {
    List<Book> findByTitleOrderByAge(String title);
}
```

```java
public interface BookRepository extends JpaRepository<Book, Long> {
    List<Book> findByTitle(String title, Sort sort);
}
```

```java
bookRepository.findByTitle( title: "Clean Code", Sort.by("age"));
```

Gelen sonuçların sayıca kısıtlanması

- find keyword'ünden sonra FirstX veya TopX vererek ilk X entity'yi alabiliriz.
- Daha iyi bir yöntem olarak, tıpkı sort gibi tüm derived query'ler Pageable parametresi alabilir. Bu pageble ile çok kolay bir şekilde hangi sayfadan kaç entity almak istediğimizi söyleyebiliriz.

```
Page<Book> books = bookRepository.findAll(PageRequest.of( page: 5,  size: 10));

Page<Book> books = bookRepository.findAll(PageRequest.of( page: 5,  size: 10, Sort.by("title")));
```

Diğer derived query'ler

- delete: Silmek için kullanılır. Sorgu atmak için kullanılan çoğu özellik burda da kullanılabilir.
- count: Belli kriterlere uyan entity'leri saymak için kullanılır
- exists: Belli kriterlere uyan entity olup olmadığına bakar
- Not: Kaydetme ve güncelleme işlemleri doğrudan save() methodu üzerinden yapılır, ekstra query'e ihtiyaç yoktur

PRACTICE

Uygulama

- Yanda görüldüğü gibi bir entity'yi entity klasörü altına oluşturun.
- Bu entity'ye ait BookRepository'i repository klasörü altına oluşturun.
- Yollayacağım entity'leri veri tabanına kaydedin.
- Aşağıdaki sorguları çalıştırmak için BookRepository interface'inde fonksiyonlar tanımlayın ve cevaplarını konsola yazın:
  - Title'ı «Domain Driven Design» olan kitabı bulun
  - Yaşı 15 ve üstü kitapları bulun ve yaşlarına göre küçükten büyüğe doğru sıralayın.
  - 2000 yılından sonra yayınlanan kitapları bulun. Bunları 5'lik sayfalara bölün ve 2. sayfayı getirin
  - İçerisinde «Clean» sözcüğü geçen kitapları bulun
  - Robert C. Martin tarafından yazılan 10 yaşından büyük kitapları bulun
  - Kent Beck'e ait kitapların sayısını bulun
  - Martin Fowler'a ait kitap olup olmadığı

```java
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Book {

    @Id
    @GeneratedValue
    private Long id;
    @Column(unique = true)
    private String title;
    private String author;
    private Long age;
    private LocalDateTime publishDate;
}
```

@Query ile SQL sorgusu

Derived query’lerin sıkıntıları

- Method isimleri çok uzun olabiliyor
- Join, like, veya inner query gibi karmaşık query’ler için yetersiz kalıyor
- (name = ‘hebele’ and age > 15) or student=false

Karmaşık durumlar için @Query

```
public interface UserRepository extends JpaRepository<Users, Long> {

    @Query("select u from Users u")
    Users findAllUsers();

}

 @Query("select u from Users u where u.username = 'admin'")
 Users findAdminUser();
```

Sort ve pagination

```
@Query("select u from Users u where u.username like '%alice%'")
List<Users> findAliceUsers(Sort page);


@Query("select u from Users u where u.username like '%alice%'")
List<Users> findAliceUsers(Pageable page);
```

Indexed query parametreleri

```
@Query("select s from Student s where s.name = ?1 and s.surname = ?2")
List<Student> findStudentByName(String name, String surname);
```

Named query parameters

```
@Query("select s from Student s where s.name = :name and s.surname = :surname")
List<Student> findStudentByName(@Param("name") String name,
                                @Param("surname") String surname);
```

Native query

```
@Query(value = "select * from student where name = :name and surname = :surname", nativeQuery = true)
List<Student> findStudentByName(@Param("name") String name,
                                @Param("surname") String surname);
```

PRACTICE

Uygulama

- Yanda görüldüğü gibi bir entity'yi entity klasörü altına oluşturun.
- Bu entity'ye ait BookRepository'i repository klasörü altına oluşturun.
- Yollayacağım entity'leri veri tabanına kaydedin.
- Aşağıdaki sorguları çalıştırmak için BookRepository interface'inde fonksiyonlar tanımlayın ve cevaplarını konsola yazın:
  - Title'ı «Domain Driven Design» olan kitabı bulun
  - Yaşı 15 ve üstü kitapları bulun ve yaşlarına göre küçükten büyüğe doğru sıralayın.
  - 2000 yılından sonra yayınlanan kitapları bulun. Bunları 5'lik sayfalara bölün ve 2. sayfayı getirin
  - İçerisinde «Clean» sözcüğü geçen kitapları bulun
  - Robert C. Martin tarafından yazılan 10 yaşından büyük kitapları bulun
  - Kent Beck'e ait kitapların sayısını bulun
  - Martin Fowler'a ait kitap olup olmadığı

```java
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Book {

    @Id
    @GeneratedValue
    private Long id;
    @Column(unique = true)
    private String title;
    private String author;
    private Long age;
    private LocalDateTime publishDate;
}
```

Spring transaction management

Spring transaction management

- Transaction yönetimi, hibernate tarafında çok zor
- Spring, bize çok daha kolay bir alternatif sunuyor

```java
@Service
public class SpringTransactionManagement {

    private final UserRepository userRepository;

    @Autowired
    public SpringTransactionManagement(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Transactional
    public void replaceUser(final User user) {
        userRepository.deleteById(user.getId());
        userRepository.save(user);
    }
}
```

@Transactional ile ilgili uyarı

- @Transactional, proxy kullandığı için, aynı class içerisinde @Transactional bir fonksiyon çağrılamaz.

```java
@Service
public class SpringTransactionManagement {

    private final UserRepository userRepository;

    @Autowired
    public SpringTransactionManagement(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public void doStuff(final User user) {
        replaceUser(user);
    }

    @Transactional
    public void replaceUser(final User user) {
        userRepository.deleteById(user.getId());
        userRepository.save(user);
    }
}
```

- Bu yüzden @Transactional ile işaretlenen methodlar:
  - En az protected olmalı
  - Class dışından bir yerden çağrılmalı

Jpa repository'lerinde transaction

- JpaRepository ile gelen save delete gibi fonksiyonlar, otomatik olarak transactional oluyor.
- Kendi yazdığımız modifiye eden fonksiyonların tepesine @Transactional yazmamız, veya kullanıldığı yerde @Transactional kullanmamız gerekiyor

```java
public interface UserRepository extends JpaRepository<User, Long> {
    List<User> findByName(String name);

    @Transactional
    void deleteByName(String name);

    @Transactional
    @Query("delete from User u where u.name = :name")
    void deleteByUserName(String name);
}
```