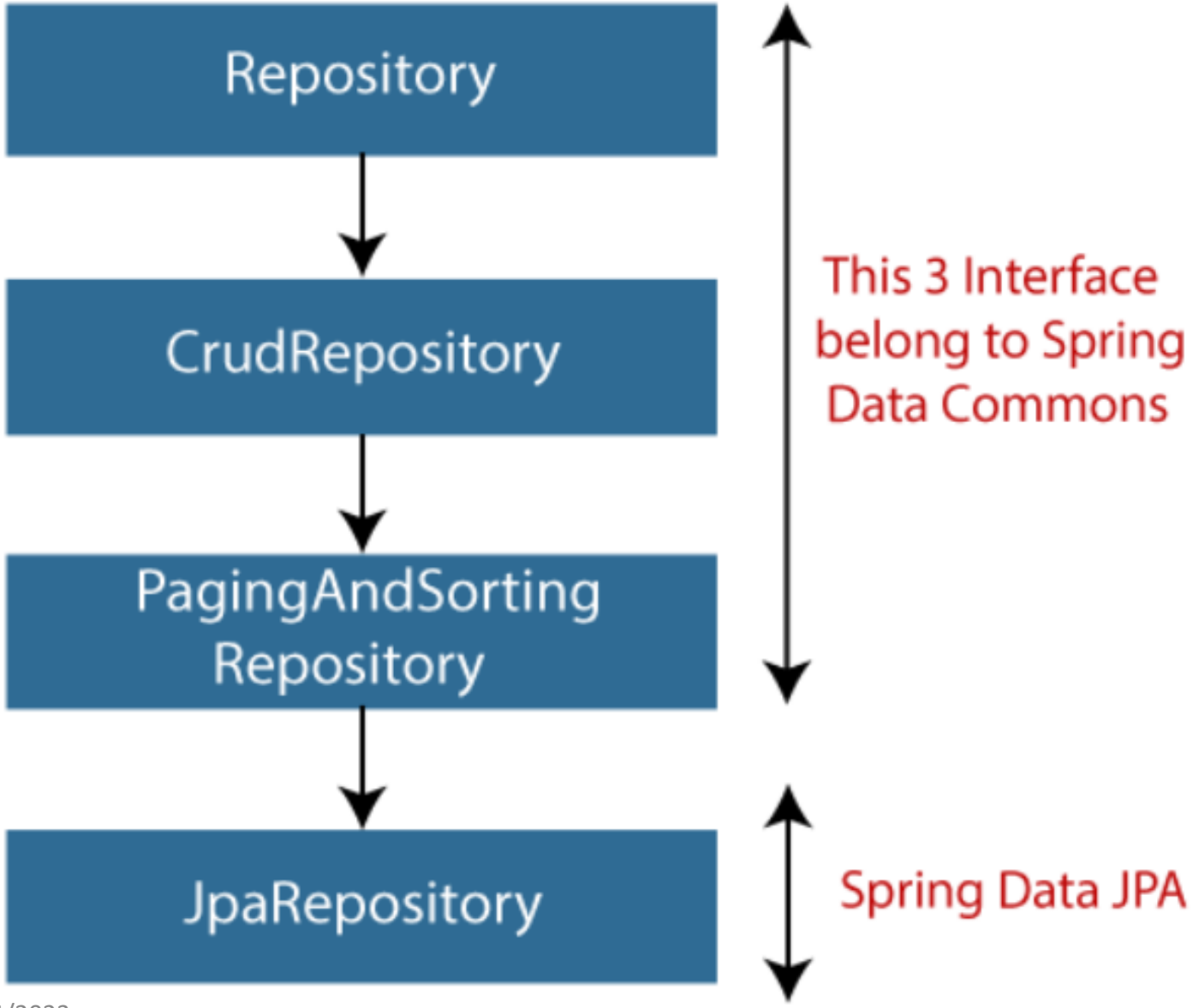
Spring Boot Derived Query Methods in

Spring Data JPA repositories

springboot02

Spring Boot DATA JPA Repository Interface



Spring Boot DATA JPA CRUD Repository

```java
public interface CrudRepository<T, ID> extends Repository<T, ID> {

  <S extends T> S save(S entity);       ❶

  Optional<T> findById(ID primaryKey); ❷

  Iterable<T> findAll();               ❸

  long count();                        ❹

  void delete(T entity);               ❺

  boolean existsById(ID primaryKey);   ❻

  // … more functionality omitted.
}
```

❶ Saves the given entity.

❷ Returns the entity identified by the given ID.

❸ Returns all entities.

❹ Returns the number of entities.

❺ Deletes the given entity.

❻ Indicates whether an entity with the given ID exists.

Spring Boot Derived Query Method

- Spring Boot aggiunge un livello di **astrazione** nella definizione delle query di accesso ai dati, **basato sugli oggetti** (Entity) e non su statement SQL
- Le query vengono generate **automaticamente** a partire dal nome del **metodo**
- L'applicazione **non ha** consapevolezza del sistema di persistenza sottostante
- Il cuore delle Derived Query sono i repository, **interfacce** basate su un **dominio** (Entity oggetto della query) e un **ID** (Primary Key dell'Entity)
- L'interfaccia radice è **Repository** e, in ordine di ereditarietà, **CrudRepository** e **PagingAndSortingRepository** e **JpaRepository**
- Non si istanziano classi, nessuna operazione di configurazione, si eseguono direttamente i metodi di persistenza delle interface dei repository

Spring Boot Derived Query Method

- I metodi derivati hanno due parti principali separate dalla prima key **By**

- La prima parte, come **find**, è detta **introducer** e il resto, come **ByTitle**, è detta **criteria**

```
1  package com.demo.repository;
2
3  import java.util.List;
6
7  public interface ArticleRepository extends CrudRepository<Article, Long>  {
8      List<Article> findByTitle(String title);
9      List<Article> findDistinctByCategory(String category);
10     List<Article> findByTitleAndCategory(String title, String category);
11 }
```

Spring Boot Derived Query Method

```java
@Table(name = "users")
@Entity
class User {
    @Id
    @GeneratedValue
    private Integer id;

    private String name;
    private Integer age;
    private ZonedDateTime birthDate;
    private Boolean active;

    // standard getters and setters
}
```

- Definiamo una Entity Class e un repository per accedere all'Entity. Notare il dominio (User) e l'ID (Integer)
- Nell'interface definiamo i metodi per derivare le query

```java
interface UserRepository extends JpaRepository<User, Integer> {}
```

Spring Boot Derived Query Method

❑ Equality Condition Keywords

- L'esatta uguaglianza è una delle condizioni più usate, ci sono molte opzioni per esprimere gli operatori **=** oppure **IS** nella query
- Si possono giusto accodare i nomi delle proprietà senza altre keyword per avere una esatta condizione di match

```
List<User> findByName(String name);
```

```
List<User> findByNameIs(String name);
List<User> findByNameEquals(String name);
```

Spring Boot Derived Query Method

❑ Equality Condition Keywords

- ➢ La disuguaglianza è espressa con l'operatore **IsNot**
- ➢ La condizione di null si codifica indicando null nell'argomento del metodo o più chiaramente con gli operatori **IsNull** e **IsNotNull**

```
List<User> findByNameIsNot(String name);
```

```
List<User> findByNameIsNull();
List<User> findByNameIsNotNull();
```

Spring Boot Derived Query Method

- **Equality Condition Keywords**
  - Si può aggiungere anche **True** o **False** per dare più chiarezza alle espressioni booleane

```
List<User> findByActiveTrue();
List<User> findByActiveFalse();
```

Spring Boot Derived Query Method

❑ Similarity Condition Keywords

Se si ha necessita di utilizzare per la query un **pattern** su una proprietà ci sono alcune opzioni e **NON** è necessario specificare alcun **%** in prefix, suffix e infix.:

- **StartingWith**

```
List<User> findByNameStartingWith(String prefix);
```

- **EndingWith**

```
List<User> findByNameEndingWith(String suffix);
```

- **Containing**

```
List<User> findByNameContaining(String infix);
```

Spring Boot Derived Query Method

- **Similarity Condition Keywords**

  Si può anche specificare la propria condizione di **Like** e in questo caso si inserisce l'operator **sql like %** nel parametro del metodo

```
List<User> findByNameLike(String likePattern);
```

```
String likePattern = "a%b%c";
userRepository.findByNameLike(likePattern);
```

Spring Boot Derived Query Method

❑ Comparison Condition Keywords

- Si possono usare le keyword ***LessThen*** e ***LessThenEqual*** per comparare i risultati con i valori dati dagli operatori **<** e **<=**

```
List<User> findByAgeLessThan(Integer age);
List<User> findByAgeLessThanEqual(Integer age);
```

- Oppure le keyword corrispondenti ***GreaterThen*** e ***GreaterThenEqual***

```
List<User> findByAgeGreaterThan(Integer age);
List<User> findByAgeGreaterThanEqual(Integer age);
```

Spring Boot Derived Query Method

❑ Comparison Condition Keywords

- Oppure si può usare la keyword ***Between*** per indicare un intervallo

```
List<User> findByAgeBetween(Integer startAge, Integer endAge);
```

- Oppure la keyword **In** per fornire una **Collection** di valori da comparare

```
List<User> findByAgeIn(Collection<Integer> ages);
```

- Oppure le keyword ***After*** e ***Before*** per il matching di date

```
List<User> findByNameOrBirthDate(String name, ZonedDateTime birthDate);
List<User> findByNameOrBirthDateAndActive(String name, ZonedDateTime birthDate, Boolean active);
```

Spring Boot Derived Query Method

❑ Multiple Condition Expressions

- Attraverso gli operatori ***And*** e ***Or*** si possono realizzare tutte le espressioni volute
- Come in Java l'ordine di precedenza è ***And*** e ***Or***
- Spring Data JPA non pone limiti a queste espressioni ma, per query troppo complesse, conviene usare l'annotation **@Query** per scrivere direttamente la query

```
List<User> findByNameOrBirthDate(String name, ZonedDateTime birthDate);
List<User> findByNameOrBirthDateAndActive(String name, ZonedDateTime birthDate, Boolean active);
```

Spring Boot Derived Query Method

❑ Sorting the Results

- Per ordinane in ordine alfabetico si utilizza l'operatore ***OrderBy***

```
List<User> findByNameOrderByName(String name);
List<User> findByNameOrderByNameAsc(String name);
```

- L'ordine ascendente è il default, per l'odine discendente di usa ***Desc***

```
List<User> findByNameOrderByNameDesc(String name);
```

Spring Boot Derived Query Method Keywords

Keyword	Sample	JPQL snippet
Distinct	findDistinctByLastnameAndFirstname	select distinct … where x.lastname = ?1 and x.firstname = ?2
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals	… where x.firstname = ?1
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1

Spring Boot Derived Query Method Keywords

Keyword	Sample	JPQL snippet
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull, Null	findByAge(Is)Null	… where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1

Spring Boot Derived Query Method Keywords

Keyword	Sample	JPQL snippet
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstname) = UPPER(?1)

Spring Boot Derived Query Method

- **Query custom esplicite**
  - Si utilizza l'annotation **@Query**

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

  @Query("select u from User u where u.emailAddress = ?1")
  User findByEmailAddress(String emailAddress);
}
```

  - Si può usare una forma avanzata di **Like**

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

  @Query("select u from User u where u.firstname like %?1")
  List<User> findByFirstnameEndsWith(String firstname);
}
```

Spring Boot Derived Query Method

- **Query custom esplicite**
  - Le query possono avere i parametri **named** oltre che posizionali

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

  @Query("select u from User u where u.firstname = :firstname or u.lastname = :lastname")
  User findByLastnameOrFirstname(@Param("lastname") String lastname,
                                 @Param("firstname") String firstname);
}
```

  - Ed essere scritte in **sql nativo**

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

  @Query(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1", nativeQuery = true)
  User findByEmailAddress(String emailAddress);
}
```