finder methods / findBy Methods   in spring data jpa
===

interface
=>These are abstract method declarations done in our repository by following certain naming conventions
=>Using these we can perform only Select Operations and Non-Select Operations are not possible
=> These finder methods will be implemented in the InMemory Proxy class of Repository Interface generating Select SQL queries internally ...
=> the select operations which are not possible with findxxx() methods of different predefined Repositories can be developed using using these custom finder methods..

syntax::
                    fixed
    <public> <RT> findBy<propertyName(s)><Condition(s)>(<params>.....)
                   prefix

=>These finder methods can be used for
  a) Entity operations (selecting all the col values of db table)
        eg: select * from Movie
                                                                all cols of the db table
                                          (or) select mid,name ,ratings,year from Movie where year in(?,?,?)

  b) Scalar operations/Projections (selecting specific single col or multiple col values of Db table)
        eg: select mid,name from Movie where mid>= ? and mid<= ?
        eg: select mid,year from Movie where mid>=? and mid<=?
        eg: select name,year from Movie where mid>=? and mid<=?

Different ways of writing persistence logics
in spring data jpa
  a) using pre-defined Repository methods
        i)CrudRepository ii) PagingAndSortingRepository
              iii) JpaRepository
  b) finder methods in our repository (select operations)
  c) @Query methods (To use JPQL, SQL Queries for select operations)
  d) @Query + @Modifying methods (for Non-select operations)

=>The finder methods of pre-defined Repositories are useful to perform either bulk select operations or single record select operation only by taking id value as the criteria value... In order to perform similar select operations by taking other than id values as the criteria values we need to use these finder methods

Example on Entity Queries (select queries giving 0 or more records by selecting all col values)

In Repository interface

```
public interface IMovieRepo extends JpaRepository<Movie, Integer> {

        //select mid,mname,year,ratings from Movie where mname=?
        public List<Movie> findByMnameEquals(String name);
        //select mid,mname,year,ratings from Movie where mname=?
        public List<Movie> findByMnameIs(String name);
        //select mid,mname,year,ratings from Movie where mname=?
        public List<Movie> findByMname (String name);
```

All the 3 methods are same

note: If condition is not specified in findBy methods
      then =(equal to) condition on the given property name will be applied automatically

```
}
```

Why do we need findByXxx() methods as the custom methods in Repository interfaces?

Ans) The pre-defined findxxx() methods of pre-defined Repository interfaces like findById(-) inCrudRepository and getReferenceById(-) in JPARepository are capable performing select operations only by taking @Id property value as the criteria value. But we want to select the records /objects based on other property/properties values as the criteria values.. for that we need these custom finder /findByXxx() methods

In Runner class

```
@Component
public class FinderMethodsTestRunner implements CommandLineRunner {
        @Autowired
        private IMovieRepo repo; // InMemory proxy class obj of our Repository(I) will be injected

        @Override
        public void run(String... args) throws Exception {
                //=====finder methods ==========
                repo.findByMnameEquals("Rathan").forEach(System.out::println);
                System.out.println("-------------------------------");
                repo.findByMnameIs("Don").forEach(System.out::println);
                System.out.println("-------------------------------");
                repo.findByMname("RRR").forEach(System.out::println);

        }

}
```

In MySQL, the finder method generated SQL Query does not apply case sensitivity by default.. whereas the Oracle generated SQL Query applies case sensitivity by default. So in oracle if want ignore case Sensitivity add the work "EqualsIgnoreCase"

public List<Artist> findByCategoryEqualsIgnoreCase(String category);

Entity class name

note: In our repository Interface if the findBy____() method return type List<T> or Iterable<T> then internally generated SELECT SQL Query will perform entity Operation (select all col values of db table for given condition on given property/col)

```
public List<Movie> findByMnameEquals(String name);
```

generates the where clause condition
(where mname=?)

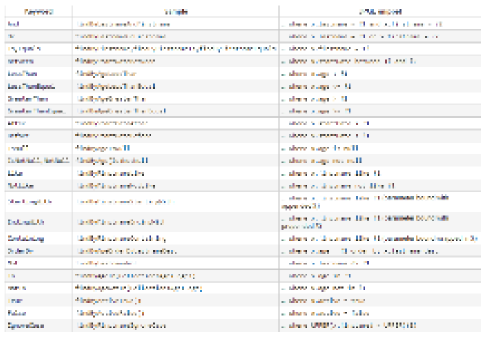
generates entity
select query
(select * from <Table> or
select col1,col2...(all cols) from <Table>)

=>If List<> or Iterable<> generic type having class is other than entity like String or user-defined interface/class then it internally generates scalar select SQL query (selecting specific column values)

note: the db table names, col names are not case sensitive .. but col data/values are case sensitive.

note: from spring 3.x level we can develop findByXxx() methods also as the getByXxx() or readByXxx() methods. but other words like showByXxx(), fetchByXxx() are not allowed

Reference Image for finder methods
==============================



Keyword	Sample	JPQL snippet
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
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… 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
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.firstame) = UPPER(?1)

More examples

Repository interface

```
public interface IMovieRepo extends JpaRepository<Movie, Integer> {

        //select mid,mname,year,ratings from Movie where mname=?
        public List<Movie> findByMnameEquals(String name);
        //select mid,mname,year,ratings from Movie where mname=?
        public List<Movie> findByMnameIs(String name);
        //select mid,mname,year,ratings from Movie where mname=?
        public Iterable<Movie> findByMname (String name);

        //select mid,mname,year,ratings from Movie where mname like 'R%'
        public Iterable<Movie> findByMnameStartingWith(String initChars);

        //select mid,mname,year,ratings from Movie where mname like '%n'
        public Iterable<Movie> findByMnameEndingWith(String lastChars);

        //select mid,mname,year,ratings from Movie where mname like '%chars%'
        public Iterable<Movie> findByMnameContaining(String chars);

        //select mid,mname,year,ratings from Movie where mname like '%chars%'
                public Iterable<Movie> findByMnameEqualsIgnoreCase(String name);

                //select mid,mname,year,ratings from Movie where mname like '%chars%'
                public Iterable<Movie> findByMnameContainingIgnoreCase(String chars);

                //select mid,mname,year,ratings from Movie where mname like 'R%'  // movies starting with R
                //select mid,mname,year,ratings from Movie where mname like '___'  // 3 letter movies
                //select mid,mname,year,ratings from Movie where mname like '%e%'  //containing letter e
                //select mid,mname,year,ratings from Movie where mname like '%R'  //ending letter R
                        public Iterable<Movie> findByMnameLike(String chars); // pass wild chars while calling method
}
```

In runner class

```
//=====finder methods ==========
                repo.findByMnameEquals("Rathan").forEach(System.out::println);
                System.out.println("-------------------------------");
                repo.findByMnameIs("Don").forEach(System.out::println);
                System.out.println("-------------------------------");
```

finder methods / findBy Methods

in spring data jpa interface =>These are abstract method declarations done in our repository by following certain naming convetions

==

=>Using these we can perform only Select Opeations and Non-Select Operations are not possible

=> These finder methods will be implemented in the InMemory Proxy class of Repository Interface generating Select SQL queries internally ...

=> The select operations which are not possible with findXxx() methods of differnt prefined Repositories can be developed using using these custom finder methods..

syntax::

fixed

<public> <RT> findBy<propertyName(s)><Condition(s)>(<params> ....)

getBy readBy

=>These finder methods can be used for

a) Entity operations (selecting all the col values of db table) eg: select * from Movie

Different ways of writing persisstence logics

in spring data jpa

a) using pre-defined Repository methods

i) CrudRepository ii) PagingAndSortingRepository iii) JpaRepository

b) finder methods in our repository (select operations)

c) @Query methods (To use JPQL, SQLQueries for select operations)

d) @Query + @Modifying methods (for Non-select operations)

=>The finder methods of pre-defined Repositories are useful to perform either bulk select operations or single record select operation only by taking id value as the criteria value.. In order to perform similar select operations by taking other id values as the criteria values we need to use these finder methods

all cols of the db table

(or) select mid,name,ratings,year from Movie where year in(?,?,?)

b) Scalar opeations/Projections (selecting specific single col or multiple col values of Db table)

eg:

select mid,name from Movie where mid>=? and mid<=?

eg:

eg:

select mid,year from Movie where mid>=? and mid<=? select name,year from Movie where mid>=? and mid<=?

Example on Entity Queries (select queries giving 0 or more records by selecting all col values) In Repsoitory interface

public interface IMovieRepo extends JpaRepository<Movie, Integer> {

//select mid,mname,year,ratings from Movie where mname=? public List<Movie> findByMnameEquals(String name); //select mid,mname, year, ratings from Movie where mname=? public List<Movie>

findByMnameIs(String name); //select mid,mname,year,ratings from Movie where mname=? public List<Movie> findBymname (String name);

}

In Runner calss

All the 3 methods are same

note:: if condition is not specified in findBy_____ methods

Why do we need findByXxx() methods as the custom methods in Repository interfaces? Ans) The pre-defined finder() methods of pre-defined Repository interfaces like findById(-) inCrud Repository and getReferenceById(-) inJPARepository are captable performing select operations only by taking @Id property value the criteria value But we want to select the records /objects based on other prpperty/properties values as the criteria values.. for that we need these custom finder /findByXxx() methods

then =(equal to) condition on the given property name will be applied automatically

@Component

public class FinderMethodsTestRunner implements CommandLineRunner { @Autowired

private IMovieRepo repo; // InMememory proxy class obj of our Repository(I) will be injected

@Override

public void run(String... args) throws Exception {

//=====finder methods ============

repo.findByMnameEquals("Anthim").forEach(System.out::println);

System.out.println("

repo.findByMnameIs("Don").forEach(System.out::println);

-");

System.out.println(".

--");

repo.findBymname("RRR").forEach(System.out::println);

In MySQL, the finder method generated SQL Query does not apply case sensitivity by default.. where as the Oracle generated SQL Query applies case sensitivity by default. So in oracle if want ignore case Sensitivity add the work "EqualsIgnoreCase"

public List<Artist>

findByCategoryEqualsIgnoreCase(String category);

}

}

Entity class name

note:: In our repository interface if the findBy______() method return type List<T> or Iterable<T> then internally generated SELECT SQL Query will perform db table for given condition on given property/col)

entity Operation (select all col values of

public List<Movie> findByMnameEquals(String name);

गु generate entity

select query

generates the where clause condtion (where mname=?)

(select * from <Table> or

select col1,col2,..(all cols) from <Table>)

=>n List<-> or Iterrable<-> generic type having class

is other than entity like String or user-defined interface/class then it internally generates scalar select SQL query (selecting specific colum values)

note:: The db table names, col names are not case-sensitive.. but col data/values are case-sensitive.

Reference image for finder methods

boot

note:: from spring 3.x we can develop findByXxx() methods also as the

getByXxx() or readByXxx() methods. but other words like showByXxx(), fetchByXxx() are not allowed

Keyword

And

Or

`Is, Equals`

Between

LessThan

LessThanEqual

Greater Than

Greater ThanEqual

`After`

Before

`IsNull`

findByAgeGreater ThanEqual

`findByStartDateAfter`

`findByStartDateBefore`

`findByAgeIsNull`

`IsNotNull, NotNull findByAge (Is) NotNull`

`Like`

`NotLike`

`findByFirstnameLike`

`findByFirstnameNotLike`

Sample

`findByLastnameAndFirstname`

`findByLastnameOrFirstname`

JPQL snippet

```
...where x.lastname = ?1 and x.firstname = ?2 ...where x.lastname = ?1 or x.firstname
= ?2
```

findByFirstname, findByFirstnameIs, findByFirstname Equals ... where x.firstname = 1?

findByStartDateBetween

findByAgeLessThan

findByAgeLess Than Equal

findByAgeGreaterThan

... where x.startDate between 1? and ?2 ...where x.age < ?1

...where x.age <= ?1

...where x.age > ?1

...where x.age >= ?1

StartingWith

findByFirstnameStartingWith

Endingwith

findByFirstnameEndingWith

Containing

findByFirstnameContaining

OrderBy

findByAgeOrderByLastnameDesc

Not

In

findByLastnameNot

findByAgeIn(Collection<Age> ages)

...where x.startDate > ?1

...where x.startDate < ?1 ...where x.age is null where x.age not null

...where x.firstname like ?1

where x.firstname not like ?1

... where x.firstname like ?1 (parameter bound with appended %)

where x.firstname like ?1 (parameter bound with prepended %)

... where x.firstname like ?1 (parameter bound wrapped in %)

where x.age = ?1 order by x.lastname desc

...where x.lastname <> ?1

...where x.age in ?1

NotIn

findByAgeNot In (Collection<Age> age)

...where x.age not in ?1

True

False

IgnoreCase

```
findByActiveTrue()

findByActiveFalse()

findByFirstnameIgnoreCase

...where x.active= true

...where x.active= false

...where UPPER (x.firstame) = UPPER(?1)
```

More examples

Repository Interface

public interface IMovieRepo extends JpaRepository<Movie, Integer> {

}

//select mid,mname,year,ratings from Movie where mname=? public List<Movie> findByMnameEquals(String name); //select mid,mname,year, ratings from Movie where mname=? public List<Movie> findByMnameIs(String name); //select mid,mname, year, ratings from Movie where mname=? public Iterable<Movie> findBymname (String name); //select mid,mname,year, ratings from Movie where mname like 'R%' public Iterable<Movie> findByMnameStartingWith(String initChars); //select mid,mname,year, ratings from Movie where mname like '%n' public Iterable<Movie> findByMnameEndingWith(String lastChars); //select mid,mname,year, ratings from Movie where mname like '%dhe%' public Iterable<Movie> findByMnameContaining(String chars); //select mid,mname,year, ratings from Movie where mname like '%dhe%' public Iterable<Movie> findByMnameEqualsIgnoreCase(String name); //select mid,mname,year,ratings from Movie where mname like '%dhe%' public Iterable<Movie> findByMnameContainingIgnoreCase(String chars); //select mid,mname, year, ratings from Movie where mname like 'R%' // movies starting with R //select mid,mname, year, ratings from Movie where mname like '______' // 3 letter movies //select mid,mname, year, ratings from Movie where mname like '%R%' //Containing letter R //select mid,mname, year, ratings from Movie where mname like '%R' //ending letter R public Iterable<Movie> findByMnameLike(String chars); // pass wild chars while calling method

In runner class

//=====finder methods =======

repo.findByMnameEquals("Anthim").forEach(System.out::println);

System.out.println("

-");

repo.findByMnameIs("Don").forEach(System.out::println);

System.out.println("

-");

repo.findBymname("RRR").forEach(System.out::println);

System.out.println("

-");

repo.findByMnameStartingWith("Ra").forEach(System.out::println);

System.out.println("--

----");

repo.findByMnameEndingWith("n").forEach(System.out::println);

System.out.println("

```
-");
repo.findByMnameContaining("m").forEach(System.out::println);
System.out.println("
-");
repo.findByMnameEqualsIgnoreCase("rrR").forEach(System.out::println);
System.out.println("
-");
repo.findByMnameContainingIgnoreCase("r").forEach(System.out::println);
System.out.println(".
-");
//repo.findByMnameLike("R%").forEach(System.out::println); //or
//repo.findByMnameLike("______").forEach(System.out::println); //or
//repo.findByMnameLike("%R").forEach(System.out::println);//or
repo.findByMnameLike("%R%").forEach(System.out::println); //or
```

BootJpaProj04-JpaRepository-findByMethods [boot]

src/main/java

#com.nt

BootJpaProj04_JpaRepositoryApplication.java

>

com.nt.entity

> JobSeeker.java

#com.nt.repository

> JobSeekerRepository.java

com.nt.runner

> FinderMethods TestRunner.java

src/main/resources

src/test/java

```
//Entity class package com.nt.entity;
import jakarta.persistence.Column;
import jakarta.persistence.Entity; import jakarta.persistence.GeneratedValue; import
jakarta.persistence.GenerationType; import jakarta.persistence.Id;
import jakarta.persistence.SequenceGenerator;
import jakarta.persistence.Table;
import jakarta.persistence.Transient;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor; import lombok.NonNull;
import lombok.RequiredArgsConstructor;
```

@Entity

@Table(name="JOB_SEEKER_INFO")

@Data

@AllArgsConstructor

>

>

JRE System Library [JavaSE-17]

> Maven Dependencies

//@NoArgsConstructor

@RequiredArgsConstructor

>

target/generated-sources/annotations

public class JobSeeker {

>

target/generated-test-sources/test-annotations

@Id

> src

>

target

@Column(name="JS_ID")

HELP.md

mvnw

mvnw.cmd

M pom.xml

//Repository Interface

package com.nt.repository;

import java.util.List;

}

@SequenceGenerator(name="gen1",sequenceName = "jsId_seq",initialValue = 1000, allocationSize = 1)
@GeneratedValue(generator="gen1",strategy = GenerationType.SEQUENCE)

//@GeneratedValue(strategy = GenerationType.AUTO)

private Integer jsId;

@Column(name="JS_NAME",length =20)

@NonNull

private String jsName;

@Column(name="JS_QLFY",length =20)

@NonNull

```
private String qlfy;
@Column(name="JS_PERCENTAGE")
@NonNull
private Double percentage;
//@Transient
@Column(name="JS_CONTACT_INFO")
@NonNull
private Long mobileNo;
public JobSeeker() {
}
System.out.println("JobSeeker:: 0-param
constructor::"+this.getClass()+"....."+this.getClass().getSuperclass());
Runner class
===========
@Component
public class Finder Methods TestRunner implements CommandLineRunner {
@Autowired
private JobSeekerRepository jsRepo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.nt.entity.JobSeeker;
public interface JobSeekerRepository extends JpaRepository<JobSeeker, Integer> {
}
public List<JobSeeker> findByJsNameEquals(String name);
public List<JobSeeker> getByJsNameIs(String name);
public List<JobSeeker> readByJsName(String name);
public List<JobSeeker> findByPercentageBetween (double start,double end); public List<JobSeeker>
findByJsNameStarting With(String nameInitialChars);
public List<JobSeeker> findByJsNameEnding WithIgnoreCase(String nameLastChars);
public List<JobSeeker> findByJsNameContainingIgnoreCase(String chars);
public List<JobSeeker> findByJsNameLikeIgnoreCase(String pattern);
public List<JobSeeker> findByQlfyIn(List<String> qualifications);
public List<JobSeeker> readByMobileNoIsNull();
public List<JobSeeker> findByQlfyInOrderByQlfyAsc(List<String> qualifications);
@Override
public void run(String... args) throws Exception {
/*List<JobSeeker> list=jsRepo.findByJsNameEquals("mahesh");
list.forEach(System.out::println);
*/
```

```
/* jsRepo.readByJsName("mahesh").forEach(System.out::println);

System.out.println("-----

jsRepo.getByJsNameIs("mahesh").forEach(System.out::println);*/

//jsRepo.findByPercentage Between (45.0, 89.0).forEach(System.out::println);

//jsRepo.findByJsNameStarting With("M").forEach(System.out::println);

//jsRepo.findByJsNameEndingWithIgnoreCase("H").forEach(System.out::println);
//jsRepo.findByJsNameContainingIgnoreCase("ah").forEach(System.out::println);

/* jsRepo.findByJsNameLikeIgnoreCase("m%").forEach(System.out::println);

System.out.println(".

jsRepo.findByJsNameLikeIgnoreCase("%h").forEach(System.out::println);

System.out.println("_

_"); ");

jsRepo.findByJsNameLikeIgnoreCase("%sh").forEach (System.out::println);*/

// isRepo.findByQlfyIn(List.of("B.E","B.Sc")).forEach(System.out::println);

//jsRepo.readByMobile NoIsNull().forEach(System.out::println);

jsRepo.findByQlfyInOrderByQlfyAsc(List.of("B.E","B.sc", "B.Tech")).forEach (System.out::println);

}
```

note:: finder methods or findBy methods are good as custom methods in JpaRepository as along with we are designing the methods using single condition on single property data otherwise its better to use @Query methods basele custom methods