Select operations using CrudRepository of spring data JPA

Optional<T> findById(ID id); --> to get single record

Iterable<T> findAll(); ->to get all records

Iterable<T> findAllById(Iterable<ID> ids); -->to get multiple records based on given ids boolean existsById(ID id);

-> To check record is avaiable or not long count(); --> to get count of records.

Need of Java 8 Optional API

problem public Student getStudentById(int id){

if(id>0)

return new Student(101,"raja","hyd",50.77f); else

}

return null;

<U> Optional<U> flatMap (Function<? super T,? extends

T

int

void

void

boolean

boolean

<U> Optional<U>

Optional<? extends U>> mapper)

get()

hashCode()

ifPresent (Consumer<? super T> action)

ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction)

isEmpty()

isPresent()

map (Function<? super T,? extends U> mapper)

<T> Optional<T>

method call

static

Student st=getStudentById(-1); if(st.getAvg()>75) -->Throws NullPointerException S.o.p("Dist");

static

else

<T> Optional<T> Optional<T>

S.o.p("pass");

T

T

T

<X extends

Throwable>

T

Stream<T>

optional obj(empty)

Optional.of(new Student());

(opt)

Optional opt=Optional.empty();

String

opt.isPresent() gives the false

opt.isEmpty() gives true

object

=> if method is returing null or certain object directly.. then there is possiblity of getting NullPointerException after invoking the method. So do not get required from method call directly.. get that object into another container object called Optional object given by Java8.. Optional API provides lots of methods to check wheather the expected object has come or not. Usig them we can avoid NullPointerException maxumum

Optional obj (opt) tudent obj

Optional opt=

opt.isPresent() -->gives true

Student st=opt.get(); gives object

Improvised code

public Optional<Student> getStudentById(int id){ if(id>0)

obj

other<T>

One Optional can hold only one object at a time or can remain empty

return Optional.of(new Student(101,"raja","hyd",78.67f); else

return Optional.empty();

}

mehtod call

of (T value)

of Nullable(T value)

or (Supplier<? extends Optional<? extends T>> supplier)

orElse(T other)

orElseGet (Supplier<? extends T> supplier)

orElseThrow()

orElseThrow(Supplier<? extends

X> exceptionSupplier)

stream()

toString()

To place given object inside the Optional object useOptional.of(-) method eg:: Student st=new Student(101,"raja","hyd"); Optional opt=Optional.of(st);

Optional<Student> opt=getStudentById(101); if(opt.present())

else

System.out.println(opt.get()); //gives the Student object System.out.println("student not found");

OptionalAPITest.java package com.nt.basics;

import java.util.Date; import java.util.Optional;

public class OptionalAPITest {

public static Optional<Date> getDateByMonth(int no){ if(no>=1 && no<=12)

return Optional.of(new Date());

findById

Optional <T> findById(ID id)

Retrieves an entity by its id.

Parameters:

id - must not be null.

Returns:

the entity with the given id or Optional #empty() if none found. Throws:

IllegalArgumentException if id is null.

example App

=>This method performs early or eager loading of the object/record i.e the moment this method called the SQL Query will be generated to fetch reocd from db table irrespective of wheather that record/obj will be used or not

service Interface

public Doctor showDoctorById(Integer id);

service impl class

@Override

public Doctor showDoctorById(Integer id) {

Doctor dutyDoctor=new Doctor();

dutyDoctor.setSpecialization("duty doctor");

Doctor doctor= doctorRepo.findById(id).orElse(dutyDoctor); return doctor;

}

(or)

@Override

(best version)

}

@Override

}

else

return Optional.empty();

}

public static void main(String[] args) {

Optional<Date> opt=getDateByMonth(-12); if(opt.isPresent()) {

}

else {

Date d=opt.get();

System.out.println("Recived obj::"+d);

System.out.println("Invalid month");

}

In service Interface

======

public Optional<JobSeeker> getJobSeekerById(int id);

In service Impl class

@Override

public Optional<JobSeeker> getJobSeekerById(int id) {

return jsRepo.findById(id);

public Doctor showDoctorById(Integer id) {

Doctor doctor= doctorRepo.findById(id).orElseThrow(()-> new IllegalArgumentException("invalid Doctor Id")); return doctor;

(or)

}

In Client App

==========

try {

else

public Doctor showDoctorById(Integer id) {

Optional<Doctor> opt=doctorRepo.findById(id);

if(opt.isPresent())

return opt.get();

throw new IllegalArgumentException("invalid doctor Id");

Optional<JobSeeker> opt=jsService.getJobSeekerById(121); if(opt.isPresent())

System.out.println("Job Seeker found::"+opt.get()); else

Optional obj(opt)

System.out.println("Job Seeker not found");

}

}

Db s/w

catch(Exception e) {

Doctor obj

jpa doctor_info(db table)

BFR

rs(ResultSet ob) if foind

opt.get()

e.printStackTrace();

Doctor obj

}

101

findById(-)

101....

if not found Optional (empty obj)

ALR

Client App

try {

Doctor doctor-service.showDoctorById(1010);

System.out.println(doctor);

}

catch(Exception e) {

}

//e.printStackTrace();

System.out.println(e.getMessage());

Updating the object

=> we donot have seperate update(-) method in any Repository.. we have only save(-) method which can be used for both save object or update object operation becoz it internally uses em.persist (-) for save object and em.merge(-) method for update object operation.

Two types of update object operations

(a) Full object modification (Except id value) or insertion

|----> use repo.save(-) method directly

(b) Partial Object modification (Except id value)

|---> use findById(-) and save(-) method gogather

In O-R Mapping,

=> Saving Object means saving the record in to Db table by collecting it from the Entity object

=> Updating object means updating the record of DB table represented by the Entity Object

=> Loading object means selecting/loading the record into the Entity Object

=> Deleting object means deleting the record represented by the Entity Object

usecase

update_doctor.html

doctor id::

0000

doctor namer

specialization income::

register

docotor_update.jsp

id:

Doctor Report Generation

name:

html table

1

raja CAR 90¢r edit

2 ravi Gynic 30d edit

rajesh babu

(read only) raja specialization: CAR income

900000000-

Cupdate dostor

100000000

or update doctor

↓

(this needs full object saving if

the given doctor id is not found or

full object update if the given doctor id is found)

Example App on full object modification or insertion

==========

service Interface

==========

public String registerOrUpateArtist(Artist artist);

service Impl class

```
@Override
public String registerOrUpateArtist (Artist artist) {
//save or update object
artistRepo.save(artist);
return "Artist is saved/upated";
(This needs partial object updation)
}
note:: while exeuting this code make sure that
no @GenertedValue is placed on the top of @Id Property in the Entity class
Client App
==========
try {
Doctor doc=new Doctor();
Example app partial Object modification
In service Interface
public String updateCustomerAddrs(int cno,String newAddrs);
In service Impl class
@Override
public String updateCustomerAddrs(int cno, String newAddrs) {
//Load the customer
Optional<Customer> opt-custRepo.findById(cno); if(opt.isPresent()) {
//get Customer object from the Optional object Customer cust=opt.get();
cust.setCadd(newAddrs);
custRepo.save(cust);
return cno+" customer address is updated";
}
return cno+" Customer is not found for updation";
doc.setDocId(1015); doc.setDocName("karan"); doc.setIncome(9000.0); doc.setSpecialization("Cardio");
System.out.println(service.registerOrUpdateDoctor(doc));
}
catch(Exception e) {
e.printStackTrace();
}
How can u make certain property of Entity class not participating in Persistence Operations?
Ans) We can make that certain property as the @Transient property as shown below in the Entity class
//Doctor.java
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.Data;
@Entity
@Table(name="JPA_DOCTOR_INFO")
@Data
public class Doctor {
@Column(name="DOC_ID")
@Id
usecase:: if do not want insert or update or retrieve certain property data temporarly then take the support of this @Transient
@SequenceGenerator(name="gen1",sequenceName = "DOCID_SEQ",initialValue = 1,allocationSize = 1)
@GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
//@GeneratedValue(strategy = GenerationType.AUTO)
private Integer docId;
@Column(name="DOC_NAME",length = 25)
private String docName;
@Column(name="SPECIALIZATION",length = 20)
private String specialization;
@Transient
@Column(name="INCOME")
private Double income;
}
note:: Instead of using @Transient, we can set null value or no value to certain property to make that property not participating in Persistence operation (But it works only in insert, update object operations)
Delete Object Operations in CrudRepository
Prefer @Transient always
In Runner class
try {
String resultMsg=custService.updateCustomerAddrs(101, "new york");
System.out.println(resultMsg);
}
catch(Exception e) {
```

e.printStackTrace();

}

Deletes a given entity.

Deletes all entities managed by the repository.

`deleteAll(Iterable <? extends T> entities) Deletes the given entities.`

`void`

`delete(T entity)`

`void`

`deleteAll()`

`void`

`void`

`void`

`deleteById(ID id)`

`deleteAllById(Iterable <? extends ID> ids) Deletes all instances of the type T with the given IDs. Deletes the entity with the given id.`

update object, load obj by id

delete object by id methods perform their persistence operation by taking

the id value as the criteria value.

To perform persistence operations in db table using spring data jpa by taking other than id value as the criteria value, we need to place custom methods in the respository interface

`void`

`deleteById(ID id)`

Deletes the entity with the given id.

`void deleteById(ID id)`

Deletes the entity with the given id.

If the entity is not found in the persistence store it is silently ignored. (no exception will be raised)

Parameters:

id - must not be null.

Throws:

`IllegalArgumentException - in case the given id is null`

Example App

==========

In service Interface

public String deleteDoctorById(Integer id);

In service Impl class

Model class ==== just java bean class

Entity class =====> Java Bean with JPA annotations of O-R mapping (@Entity,@Table, @Column and etc..)

To transfer data b/w the layers of the same project or different projects we take the support of Model class..

Some times the Entity class itself acts the model class

```
@Override
public String deleteDoctorById(Integer id) {
//Load object
Optional<Doctor> opt=doctorRepo.findById(id);
if(opt.isPresent()) {
doctorRepo.deleteById(id);
return id+" doctor is deleted";
}
else {
}
}//method
```

In client App

```
=========
return id+" doctor not found for deletion";
try {
System.out.println(service.deleteDoctorById(111));
}
catch(Exception e) {
e.printStackTrace();
}
```

`void`

`delete(T entity)`

`void delete(T entity)`

Deletes a given entity.

Parameters:

Deletes a given entity.

Though we pass compleete entity object as the arg value

it takes only @Id property value as the criteria value to

entity - must not be null. to select and delete the object/r

Throws:

`IllegalArgumentException in case the given entity is null.`

OptimisticLocking FailureException - when the entity uses optimistic locking and has a version attribute with a different value from that found in the persistence store. Also thrown if the entity is

assumed to be present but does not exist in the database.

Example app

```
==============
```

In service Interface

public String delete Doctor(Doctor doctor);

Service Impl class

```
@Override
public String delete Doctor(Doctor doctor) {
//Load object
Optional<Doctor> opt=doctorRepo.findById(doctor.getDocId());
if(opt.isEmpty()) {
return doctor.getDocId()+" doctor is not found";
}
else
{
doctorRepo.delete(opt.get());
}
}
```

In client app

```
return doctor.getDocId()+" doctor found and deleted";
try {
}
Doctor doc=new Doctor();
doc.setDocId(12); doc.setDocName("karan");
System.out.println(service.deleteDoctor(doc));
catch (Exception e) {
}
e.printStackTrace();
```

Explain the usecases to use deleteById(-) method and delete(T) method?

confirmation box

Are sure that u want to delete?

yes

no

usecase1::

report page

raja 1000 CRD delete

?id=1

2

rajesh 20000 CRD

delete

?id=2

use deleteById(-)

usecase2:

delete_doctor.html (form page)

report page

doctorId:

(non editable)

raja

1000 CRD view and delete

doctor Name: raia

?id=1

2

rajesh 20000

CRD view and delete

secialization

?id=2

income

CRD 1000

Delete

recived the form data into

entity class object and

use delete (T) method having entity object as the arg value

note:: To perform update operations, delete operations and find operations with our choice conditions and property/col values as the criteria value .. use custom methods in the Custom Repository interface

note:: partial insertion of record/object, partial update of record /object and parital loading of record is possible

partial record deletion is not there.. deleting one or two col of values db table records comes undePartial/object updation.