Associations in spring data jpa

(Relationships or Association Mapping or Multiplicity in spring data jpa or Hibernate)

(Spring boot Data JPA)

Need of keeping db tables in assocation

=>keeping multiple entities/items data in single db table as records is having the following problems a) Data Redundency (Duplication) Problem

b) Data Management problem (Data Inconsistency Problem)

note:: when the DB tables are kept in association the relavent Entity classes should also be kept in the Association

Problem::

Customer_Phone_details (db tables)

cno (n)

cname(vc2) caddrs(vc2) mobileNo(n)

provider(vc2)

type(vc2)

101

raja

hyd

9999999999

airtel

101

raja

hyd

9999999888

vodfone

residence office

102

mahesh

vizag

9499999888

jio

102

mahesh

vizag

9499999888

vodfone

office residence

Keeping huge amount

of data in single db table is not recomanded process.

cno (n) (pk)

Data Redundency Problem

Solution1: Keep multiple entities info in multiple db tables like user information in seperate db table and phone number details in seperate db table

(Bad)

customer(db table1)

cname(vc2) caddrs(vc2)

phone_details (db table2)

regNo(pk) mobileNo(n)

provider(vc2)

type(vc2)

101 102

raja

hyd

1

mahesh

vizag

2

9999999999 9999999888

3

9499999888

airtel vodfone jio

residence

office

4

9499999888

vodfone

Solution2:

(Good)

from

office residence

=> Now data redundency problem is gone.. but data management /navigation problem is created i.e we can not access phone number details cusotmer table records and vice-versa.

(Becoz the db tables not in link)

take two db tables to maintain multiple entities info and keep them in relationship using FK (Foriegn key column)

customer(db table1) (parent db table)

cno (n) cname(vc2) caddrs(vc2)

P

phone_details (db table2) (child db table)

regNo(pk) mobileNo(n)

provider(vc2)

type(vc2)

customer_no(FK with cno)

101 102

raja mahesh

hyd

1

vizag

2

9999999999 9999999888

airtel vodfone

residence

101

office

101

3

9499999888

4

9499999888

jio vodfone

office

102

residence

102

Advantages::

(a) No Data Redundency Problem

(b) No Data Navigation problem ..using FK column we can acess child table records from parent db table records and vice-versa..

as

be

if db tables are in relationship then the associated entity classes should also taken having relationship.. but we keep db tables in relationship using FK column where we keep Entity classes in relationship using Composition and Collections i.e the way db tables are in relationship is different from the way entity classes in relatioship ..To avoid this mismatch use the advannced o-r mapping called Association Mapping..

Spring data JPA /ORM'supports 4 types associations

a) One to One →

b) One to Many

c) Many To One →

d) Many to Many

=>student and RankInfo (1 to 1) =>User and Phone Number (1 to M) =>Dept and Employee (1 to M) =>Student and Course (M to M) => Facutly and Student (M to M) => DrivingLicense and Citizen (1 to 1)

=> Passport and Citizen (1 to 1)

Other mismatches

========================

=> Db tables can not be there in the inheritence ..but entity classes can be there in the inheritence,So to map the entity classes

of inheritence with Db tables go for "Inheritence mapping"

=> Db tables can not be there in the Composition ..but entity classes can be there in the Composition,So to map the entity classes of composition with Db tables go for "Component mapping"

=> Db tables do not have collection type cols.. but entity classes

can have array /collection type properties,So to map Entity classes with collection to db tables use "collection Mapping"

note: It is always good practice to keep FK column in the child db table for better association b/w parent and child db tables, this also avoid data redundancy problem

note: Using single FK column that is placed generally in child table we can access parent table records from child table and child table records from parent table (i.e Bi-Directional access is possible)

As part of Association mapping cfgs we need to encounter various cfgs related inheritance mapping, component mapping and collection mapping

=> Employee and Dept (M to 1)

=> Student and College (M to 1)

=> Doctor and Patient (M to M)

=> CoronnaVaccine and Citizen (1 to 1)

=> Person and AadharCard (1 to 1)

=> Customer and PhoneNumber (1 to M)

Associations in ORM f/ws or spring data JPA can be implemented in two modes

a) Uni-Directional Assoicaiton

(Either parent to child or child to parent access possible)

b) Bi-Directional Assoicaiton

(parent to child and child to parent access possible)

Annotations required for the Association mapping @OneToOne

@OneToMany

@ManyToOne

@ManyToMany

@JoinColumn ---> To specify the FK column @JoinTable ----> To specify third table in many to many Association

=> To keep db tables either in uni or bi-directional association we just need to take FK column only in child db table. But to keep Entity classes in uni-directional association take special property only in one entity class and for bi-directional association take special properties in both parent and child entity classes

=>Asscociations in ORM f/ws or spring data JPA can be build in Entity classes in two ways (Either uni-directional one Bi-Directional one)

HAS-A

a) Using Reference type properties in composition (Non-Collection property)

(1 to 1, M to 1)

b) Using Collection type propeties in Composition (Collection type HAS-A property) (1 to M, M to M)

in

=>We can keep db tables either uni-directional or in bi-directionla association using signle FK column,

But in Java no FK colum is available..So to keep Entity classes in association we need to use

with.

composition either collection or non-collection type properties. (For uni-directional one side and for bi-directional both sides)

One to Many Bi-Directional Association / Many to One Bi-Directional Association

===

=====

=====

=>Bi-directional association means we can access parent objs from child objs and vice-versa. => To build one to many Bi-Directional Association the parent class should have special property of type Collection (set/list/map) to hold bunch of child class objs, similarly child class should have special property of type Parent class (parent class reference varable) to hold parent class obj.

one

=>One to many association is from parent to child where as

from child to parent it is many to one association.

Person and Phone Number relationship is One to many relationship from person Prospective becoz one person can have multiple phonNumbers.. The same is many to one relationship from Phone Number prospective becoz multiple phones can belong to a single person...

OTOM_Person (parent db tablet

PID(PK) PNAME(vc2) PADDRS(vc2)

OTOM_PhoneNumber (child table)

REGNO (pk) PHONENo(n) PROVIDER(vc2) TYPE(vc2) PERSON_ID(FK)

1

raja

hyd

1000

2

suresh

vizag

1001

99999999 88888888

airtel jio

1002 1003

7777777777 jio 7777777999

airtel

personal 1 office

1

personal

2

hone

2

=>While designing db tables .. do not take the columns that holds outside business values as PK column becoz the values may change based on outside world business polocies (Do not take natural key column as the PK column)

eg: taking mobileNo, voterId,aadharNo, passportNo and etc.. cols as PK column is bad pratice becoz these values may change becoz of GOVT polocies (business policies) (take surrogate key column as the PK column)

=>Always take that column as PK column which gets values from underlying App or Db s/w dynamically

eg:: Column that holds sequence generated values (oracle)

Column that hold automincrement values (mysql)

=>In associating cascading means.. the nons-select persistence operations perfomed on the main obj will be propagated/cascadded to the associated objs.. possible values are

cascade CascadeType.ALL, (Best)

cascade CascadeType.PERSIST,

cascade CascadeType.REMOVE,

cascade CascadeType.MERGE,

cascade CascadeType.DETACH, cascade CascadeType.REFRESH

Enity classes

===========

```
package com.nt.entity;
@Entity
@Table(name="JPA_OTM_PERSON")
@Setter
@Getter
//@NoArgsConstructor
```

@RequiredArgsConstructor

public class Person implements Serializable { @Id

@GeneratedValue

private Integer pid;

@Column(length = 20)

@NonNull

private String pname;

@Column(length = 20)

@NonNull

=>While designing db tables for relationship it is recomanded to take FK column in the child table becoz it avoids the redundancy problem in the parent db table.

=>While designing Entity classes for Association mapping

do not use @Data for generating common code becoz

the generated hashCode(), toString(), equals() are not suitable for Association mapping, generate them manually or use Object class methods.. So use only @Setter and @Getter annotations

private String paddrs;

@OneToMany(targetEntity = PhoneNumber.class, cascade = CascadeType.ALL,

fetch =FetchType.LAZY, mappedBy = "personInfo")

//@JoinColumn(name="PERSON_ID",referencedColumnName = "PIP")

private Set<PhoneNumber> contactDetails; //builds One to Many assocation from parent

@Override

FK col to be created

public String toString() { in child table

PK column of Parent table

return "Person [pid=" + pid + ", pname=" + pname + ", paddrs=" + paddrs + "]";

}

public Person() {

System.out.println("Person:: 0-param constructor");

}

PhoneNumber.java

@Entity

@Table(name="JPA_OTM_PHONE_NUMBER")

@Setter

@Getter

//@NoArgsConstructor

@AllArgsConstructor

@RequiredArgsConstructor

public class PhoneNumber implements Serializable {

@Id

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

private Integer regNo;

@NonNull

//toString()

private Long mobileNo;

@Override

@Column(length = 20)

public String toString() {

@NonNull

private String provider;

@Column(length = 20)

}

@NonNull

private String numberType;

}

@ManyToOne(targetEntity = Person.class,cascade = CascadeType.ALL, fetch = FetchType.LAZY)
@JoinColumn(name="PERSON_ID",referencedColumnName = "PID")

private Person personInfo;

public PhoneNumber() {

System.out.println("PhoneNumber: 0-param constructor");

return "PhoneNumber [regNo=" + regNo + ", mobileNo=" + mobileNo + ", provider=" + provider + ", numberType=" + numberType + "]";

=>In bi-directional Assoications.. instread of specifying foreign key colum using @JoinColum in both parent and child

classes we can specify only at one side with the support of "mappedBy" param.

@OneToMany(targetEntity = PhoneNumber.class, cascade = CascadeType.ALL,mappedBy = "person")

to

=>In One to Many association we need specify

mappedBy at One Side (parent class)

=>In Many to Many and One To One association we can specify

mappedBy any Side (parent class or child class)

=> In One To Many Assocation the default fetch type is Lazy

FectType.LAZY ---- the main objects will be loaded normally but the associated

(default in

One To many)

FectType.EAGER

Repository Interfaces

object will be loaded on demanad i.e only when need of

utliazation is initiated

(Best)

---- Here the Associated objects will be loaded along with the main objects irrespective of wheather u haved initiated the utilization or not

The property of Associated class

on which @JoinColum is used to specify FK column.

=> CasecadeType is related to non-select persistence operations in Association Mapping

=> FetchType is related to Select Persistence operations in the Association Mapping

```
package com.nt.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import com.nt.entity.Person;
public interface IPersonRepostitory extends JpaRepository<Person, Integer> {
}
```

```
package com.nt.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import com.nt.entity.PhoneNumber;
public interface IPhoneNumberRepository extends JpaRepository<PhoneNumber, Integer> {
}
```

service Interface

```
package com.nt.service;
public interface IOTOMAssociationMgmtService {
public void save Data UsingParent();
public void save Data UsingChild();
public void loadDataUsingParent();
}
```

Service Impl class

```
@Service("OTOMService")
public class OTOMAssociation MappingServiceImpl implements IOTOMAssociationMgmtService {
@Autowired
private IPersonRepostitory personRepo;
@Autowired
private IPhoneNumberRepository phoneRepo;
@Override
public void saveDataUsingParent() {
//create Parent object
```

```
Person per=new Person("raja", "hyd");
//create child objects
PhoneNumber ph1-new PhoneNumber(9999999L, "airtel", "personal");
PhoneNumber ph2=new PhoneNumber(88888888L, "vi", "office");
//add parent to child
ph1.setPersonInfo(per); ph2.setPersonInfo(per);
//add childs to parent
Set<PhoneNumber> phonesSet=new HashSet();
phonesSet.add(ph1); phonesSet.add(ph2);
per.setContactDetails(phonesSet);
//save the parent object
personRepo.save(per);
System.out.println("Person and his associated phoneNumbers are saved (parent to child)");
}//method
@Override
public void saveDataUsingChild() {
//create Parent object
//create child objects
Person per=new Person("ramesh", "vizag");
PhoneNumber ph1-new PhoneNumber(88888888L, "airtel", "personal");
PhoneNumber ph2=new PhoneNumber(77777777L, "vi", "office");
//add parent to child
ph1.setPersonInfo(per); ph2.setPersonInfo(per);
//add childs to parent
Set<PhoneNumber> phonesSet-new HashSet();
phonesSet.add(ph1); phonesSet.add(ph2);
per.setContactDetails(phonesSet);
//save the parent object
phoneRepo.save(ph1); phoneRepo.save(ph2);
System.out.println("Person and his associated phoneNumbers are saved (child to parent)");
}
@Override
public void load Data UsingParent() {
```

application.properties

=======

Boot-DataJPA-Proj12-Association Mapping-OTOM-MTOO [boot]

> Spring Elements

#src/main/java

✓com.nt

BootDataJpaProj12Association Mapping OTOMApplication.java

com.nt.entity

> Person.java

> PhoneNumber.java

com.nt.repository

>IPersonRepostitory.java

> IPhoneNumberRepository.java

com.nt.runner

> OTOMAssociationMappingRunner.java

com.nt.service

>IOTOMAssociationMgmtService.java

> OTOMAssociation MappingServiceImpl.java

#src/main/resources

application.properties

src/test/java

> JRE System Library [JavaSE-17]

>

Project and External Dependencies

>bin >

gradle

> src

build.gradle

gradlew

gradlew.bat

WHELP.md

settings.gradle

```
Iterable<Person> it-personRepo.findAll();
it.forEach(per->{
System.out.println("parent::"+per);
//get childs of each parent
/* Set<PhoneNumber> childs=per.getContactDetails();
System.out.println("childs count ::"+childs.size());*/
/* childs.forEach(ph->{
System.out.println("child ::"+ph.getMobileNo());
```

#jdbc properties (for oracle)

spring.datasource.driver-class-name-oracle.jdbc.driver.Oracle Driver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=manager

spring.datasource.hikari.maximum-pool-size=100

spring.datasource.hikari.minimum-idle=10

spring.datasource.hikari.keepalive-time=100000

});*/

After commenting child objs processing as shown

});

above, u run the code using both

spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true

FetchType.EAGER and FetchType.LAZY

}

}//class

Runner class

===================

package com.nt.runner;

For FetchType.LAZY ---> the App generates single select query selecting all the records of the parent i.e childs are not

selected and loaded

For FetchType.EAGER --> The App generates 1+n select Queries

spring.jpa.datasource-platform=org.hibernate.dialect.Oracle 10gDialect

spring.jpa.show-sql=true

to load parent and child records .. here 1 select query to get all parent records

and "n" select queries to get all the childs records of the "n" parents

In One To Many Association the default

FetchType is FetchType.LAZY

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import com.nt.service.IOTOMAssociation MgmtService;

@Component

public class OTOMAssociation Mapping Runner implements CommandLineRunner {

@Autowired

private IOTOMAssociation MgmtService service;

@Override

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

// service.saveDataUsing Parent();

}

}

//service.saveDataUsingChild();

service.loadDataUsingParent();

In Many To One Assocation Mapping the default fetch type is EAGER

FetchType.EAGER :: The associated parent objs will be loaded along with the child objs normally
FetchType.LAZY :: the child objs will be loaded normally and the associated parent objs will be loaded lazily on demanded basis.. (Till that time Proxy parent objs will be linked with child objs)

(good)

In service interface

public void loadDataUsingChild();

In Service Impl class

@Override

public void loadDataUsingChild() {

// get All child objects and the associated parent objs

childs.forEach(ph->{

List<PhoneNumber> childs=phoneRepo.findAll(); System.out.println("child --->"+ph);

});

}//method

In runner class

//get the associated parent objs

/* Person per=ph.getPerson();

System.out.println("Parent--->"+per);*/

otmService.loadDataUsingChild();

By commenting the code related parent objs accssing and processing

we can feel the Eager loading and Lazy loading

FetchType.EAGER ---> we get 1+n select queries to get all the childs and associated parent records 1 query to get

all childs, n queries to "n" parent of the childs FetchType.LAZY ---> gives only 1 select query loading only childs records i.e associated parent records will not be loaded becoz Parent objs processing is commented

What is the Lazy loading and eager Loading w.r.t Association Mapping?

Ans) In Association mapping, lazy loading means the main objs will be loaded normally but associated objs will be loaded lazily on demand basis

In Association mapping, eager loading means the Associated objs will be loaded along with

main objs.

In How many ways we can achieve Lazy Loading in spring data jpa programming?

Ans) =>While dealing with simple o-r mapping (single db table operations) we can use

repo.getReference ById(-) method for lazy loading

=>While dealing with advanced o-r mapping like Association mapping we can use FetchType.LAZY for achieving layz loading (two db tables of relationship)