spring.datasource.url=jdbc:mysql://localhost:3306/resturant

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

#spring.jpa.hibernate.ddl-auto=update

* *validate*: validate the schema, makes no changes to the database.
* *update*: update the schema.
* *create*: creates the schema, destroying previous data.
* *create-drop*: drop the schema when the SessionFactory is closed explicitly, typically when the application is stopped.

Enabling h2 console

#spring.h2.console.enabled=true

# to check time statistic like how much time taken by query to prepere and execute

spring.jpa.properties.hibernate.generate\_statistics=true

logging.level.org.hibernate.stat=debug

#checking value in query

logging.level.org.hibernate.type=trace

stoping to change bootstrap console

spring.datasource.url=jdbc:h2:mem:testdb

spring.data.jpa.repositories.bootstrap-mode=default

Sequence

@Id

@GeneratedValue(strategy=GenerationType.***SEQUENCE***,generator="prod\_trx\_seq")

@SequenceGenerator(name="prod\_trx\_seq",sequenceName="prod\_trx\_seq")

Long product\_bill;

Working with JDBC template

1. How to enable h2 console.
   1. spring.h2.console.enabled=true ---🡪 Properties file
   2. <http://localhost:8080/h2-console/>

2021-03-09 08:07:17.030 INFO 16332 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:d64802f3-0f6f-4c05-ae78-aaf700c0f99c'

Crud example using JMS template

----🡪 @Autowired

JdbcTemplate jdbc;

System.***out***.println("Before Update");

**for**(User u:jdbc.query("select \* from user",**new** BeanPropertyRowMapper<User>(User.**class**))) {

System.***out***.println(u);

}

jdbc.update("update user set name=?1 where id=?2",**new** Object[] {"Vicky Arya",3});

System.***out***.println("After Update and before insert");

**for**(User u:jdbc.query("select \* from user",**new** BeanPropertyRowMapper<User>(User.**class**))) {

System.***out***.println(u);

}

jdbc.update("insert into user (id,name,location) values (?1,?2,?3)",**new** Object[] {4,"ANUJ","Fazilka"});

System.***out***.println("after insert before delete");

**for**(User u:jdbc.query("select \* from user",**new** BeanPropertyRowMapper<User>(User.**class**))) {

System.***out***.println(u);

}

jdbc.update("delete from user where id =?1",**new** Object[] {1});

System.***out***.println("After delete ");

**for**(User u:jdbc.query("select \* from user",**new** BeanPropertyRowMapper<User>(User.**class**))) {

System.***out***.println(u);

}

Output :

Before Update

User [id=1, name=Deepak, location=Abohar]

User [id=2, name=Tarun, location=Bathinda]

User [id=3, name=Vicky, location=Canada]

After Update and before insert

User [id=1, name=Deepak, location=Abohar]

User [id=2, name=Tarun, location=Bathinda]

User [id=3, name=Vicky Arya, location=Canada]

after insert before delete

User [id=1, name=Deepak, location=Abohar]

User [id=2, name=Tarun, location=Bathinda]

User [id=3, name=Vicky Arya, location=Canada]

User [id=4, name=ANUJ, location=Fazilka]

After delete

User [id=2, name=Tarun, location=Bathinda]

User [id=3, name=Vicky Arya, location=Canada]

User [id=4, name=ANUJ, location=Fazilka]

Custome Row Mapper

@Component

**public** **class** JDBCTemplateCustomeRowMapperExample {

@Autowired

JdbcTemplate jdbc;

**class** PersonRowMapper **implements** RowMapper<Person>{

@Override

**public** Person mapRow(ResultSet rs, **int** rowNum) **throws** SQLException {

Person person=**new** Person();

person.setId(rs.getLong(1));

person.setPersonName(rs.getString(2));

person.setCity(rs.getString(3));

**return** person;

}

}

**public** **void** f1() {

**for**(Person p:jdbc.query("select \* from user",**new** PersonRowMapper())) {

System.***out***.println(p);

}

}

}

Output:

Person [id=1, personName=Abohar, city=Deepak]

Person [id=2, personName=Bathinda, city=Tarun]

Person [id=3, personName=Canada, city=Vicky]

Entity Manager Example

1. Find by Id

@Repository

@Transactional

**public** **class** JPAEntityExample {

@PersistenceContext

EntityManager entManager;

**public** User findById(Integer id) {

**return** entManager.find(User.**class**, id);

}

**public** User insertOrUpdate(User obj) {

**return** entManager.merge(obj);

}

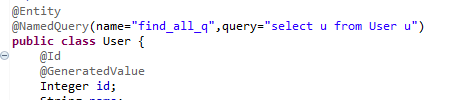
**public** **void** deleteById(Integer Id) {

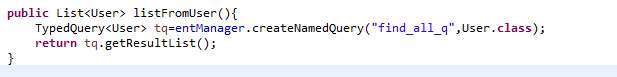
entManager.remove(**this**.findById(Id));

}

}

NameQuery





**Tracking Transaction using Entity Manager**

This will insert Record to DB

Not commit until EM complete transaction

@Autowired

EntityManager em;

User u=**new** User();

This statement update user location in DB because “Entity Manager” keep track of all entity

u.setName("Anuj");

u.setLocation("chandigarh");

em.persist(u);

u.setLocation("###########");

em.flush(); --- > commit changes to DB.

em.detach(u);--🡪 to remove tracking of element u

em.clear()--🡪 remove tracking of all elment

em.refresh(entity) 🡪 entity will know contain a fresh copy simpiler commited in DB

JPQL basic:

**Code :**

**public** **void** basicJPQL() {

List result=em.createQuery("select u from User u").getResultList();

System.***out***.println(result);

}

Output :

[User [id=1001, name=Deepak, location=Abohar], User [id=1002, name=Tarun, location=Bathinda], User [id=1003, name=Vicky, location=Canada]]

**Code :**

**public** **void** basicJPQL() {

TypedQuery<User> ls=em.createQuery("select u from User u",User.**class**);

System.***out***.println(ls.getResultList());

}

Output :

**[User [id=1001, name=Deepak, location=Abohar], User [id=1002, name=Tarun, location=Bathinda], User [id=1003, name=Vicky, location=Canada]]**

**Code :**

**public** **void** basicJPQL() {

TypedQuery<User> ls=em.createQuery("select u from User u where name like '%D%'",User.**class**);

System.***out***.println(ls.getResultList());

}

Output :

**[User [id=1001, name=Deepak, location=Abohar]]**

Working with column annotation

@Column(name="userName",nullable=**false**)

String name;

LocalDateTime java 8 class

**Timestamp for update and creation**

@CreationTimestamp

**private** LocalDateTime sysCreationDate;

@UpdateTimestamp

**private** LocalDateTime sysUpdateDate;

**Native query:**

**1.select with :param**

@Override

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

Query query=em.createNativeQuery("select \* from user where id=:id and location like :location",User.**class**);

query.setParameter("id", 1001);

query.setParameter("location", "%A%");

List<User> ls=query.getResultList();

System.***out***.println(ls);

}

**2.select with ?**

@Override

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

Query query=em.createNativeQuery("select \* from user where id=? and location like ?",User.**class**);

query.setParameter(1, 1001);

query.setParameter(2, "%A%");

List<User> ls=query.getResultList();

System.***out***.println(ls);

}

**3.update**

@Override

@Transactional

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

Query query=em.createNativeQuery("update user set user\_name='Deepu' where id=?");

query.setParameter(1, 1001);

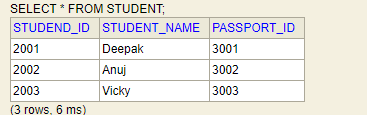
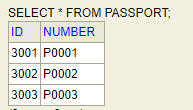
Integer noOfRowUpdate=query.executeUpdate();

System.***out***.println(noOfRowUpdate);

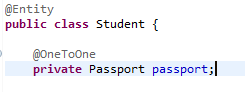
}

Note : if you are executing any query like update delete insert these query come under transaction management @Transcationl --------

**@OneToOne**

** **

This statement create a column in student table “Passport\_Id”

****

@Repository

@Transactional

**public** **class** StudentRepo {

Select

student0\_.studend\_id as studend\_1\_3\_0\_,

student0\_.passport\_id as passport3\_3\_0\_,

student0\_.student\_name as student\_2\_3\_0\_,

passport1\_.id as id1\_1\_1\_,

passport1\_.number as number2\_1\_1\_

from

student student0\_

left outer join

passport passport1\_

on student0\_.passport\_id=passport1\_.id

where

student0\_.studend\_id=?

This join statement will

Excute on by find method

@Autowired

EntityManager em;

**public** **void** findById() {

Student st= em.find(Student.**class**, 2001l);

System.***out***.println(st.getStudentName());

}

**LasyFetch**

@Entity

**public** **class** Student {

When we use lazy fetch then join query will not run Only student query will run

Hibernate: select student0\_.studend\_id as studend\_1\_3\_0\_,

student0\_.passport\_id as passport3\_3\_0\_,

student0\_.student\_name as student\_2\_3\_0\_

from

student student0\_

where

student0\_.studend\_id=?

@OneToOne(fetch=FetchType.***LAZY***)

**private** Passport passport;

--------

@Override

@Transactional

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

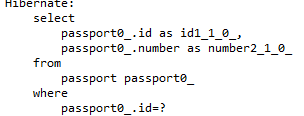
Student st=stcurdrepo.findById(2001l).get();

System.***out***.println(st.getStudentName());

System.***out***.println(st.getPassport().getNumber());

}

But when we encounter this statement this will run : query for passport table



@Transactional

Note: this annotation is nessary for lazy fatch -🡪 this annotation keep session alive until this method return and all change in entity will save in db if this method run successfully without

Exception.

Without tranacation method :

@Override

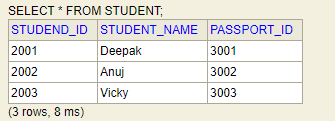
**public** **void** run(String... args) **throws** Exception {

Student st=stcurdrepo.findById(2001l).get();

st.setStudentName(st.getStudentName()+" -Updated");

}

Output



With tranacation method :

@Override

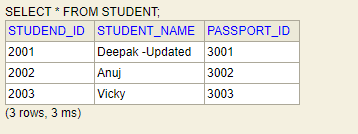
@Transactional

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

Student st=stcurdrepo.findById(2001l).get();

st.setStudentName(st.getStudentName()+" -Updated");

}



Bidirectional OneToOne

Student class :

@Entity

**public** **class** Student {

@OneToOne(fetch=FetchType.***LAZY***)

**private** Passport passport;

**Passport Class :**

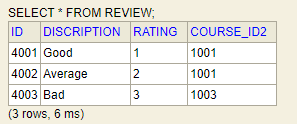
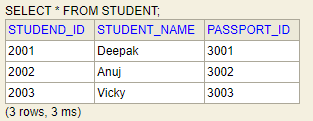
@Entity

**public** **class** Passport {

@OneToOne(fetch=FetchType.***LAZY***,mappedBy="passport")

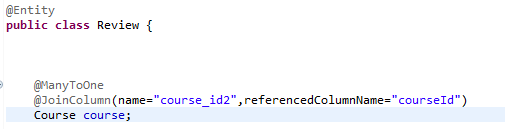
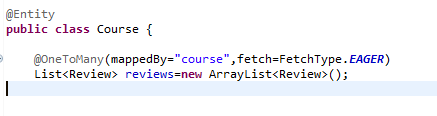
**private** Student student;

**@OneToMany AND @ManyToOne**

** **

Review can have only one Course

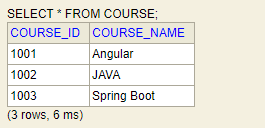
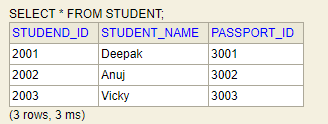
Course can have multiple reviews

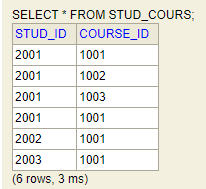
** **

@JoinColumn -🡪 name refer will be new created column

referencedColumnName name of the column primary key in course class

**@ManyToMany**

** **

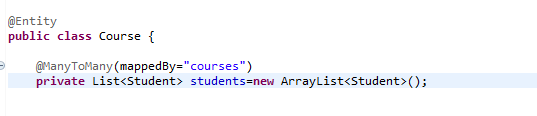
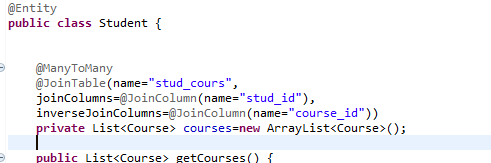
****

Column name in

Course table

Column name for student table

Mapping table name

** **

­=========

1. CURD operation using jdbc template .
2. CustomeBean mapping
3. JPA entity manager
4. Named Query find all result
5. Play with Entity manager Flush,clear,refresh ,persitance
6. JPQL query –
7. @CreationTimestemp and @UpdateTimeStamp
8. Native query
9. OneToOne ,OneToMany,ManyToOne