**Hibernate**

1)Lazy Loading vs Eagar loading

2)Relational Mapping

2.1--)one to One :Customer-->Item

-In one to one mapping there are two way unidirectional and bidirectional

-in unidirectinal mapping the customer table has the column for item table.

-In bidirectinal mapping the table who has the foreign key that is the owner side in this mapping and other side has mappedBy used.Here customer has @JoinColumn with foreign key

and item has mappedby annotation.

--one to one joining table where new table created which is third table

@JoinTable(name="cutomer\_item,

joinColumns={@joinColumn(name="customer\_id")}

reverseJoinColumns={@joinColumn(name="item\_id")});

private Item item;

when we are doing joining table item dont need to provide anything as third table is careated

2.2--)one to many:Cutomer --> Item

-in one to many mapping in unidirectional mapping if i dont provide any @join column annotation and in customer table if i annotate with only

@OneTOMany(cascade=cascadeType.all)

private List<Item>items;

-it will create third column with both primary key from cutomer and id table.As i am not providing any @join column

-If i am providing @joinColumn over customer table then customer\_id column created in many side that is Item side .so in one to many many side is always owneer side.

@OneTOMany()

@joinColumn(name="customer\_id)

private List<Item>items;

-to add element in of item in customer by

c.getItems().add(i1);

c.getItems().add(i2);

repo.save(c);

2.3--)many to one

-in many to one mapping here we always do two-way mapping so to do that what we can do on the customer side we have one too many and in the item, we have many-to-one with the item on is owner side

so the .item never has mappedby annotation this annotation is always at customer side.

in customer side

@OneToMany(cascade=cascadeType.All,mappedBy="customer")

private List<Item>items;

in Item side

@ManyToOne(cascade=CascadeType.all)

@joinColumn(name="customer\_id)

private Customer customer;

-in such a case only two columns created and customer table won't create any column cutomer table item will not be created only in the item table one extra column be created which is the customer id and in cutomer side we use mapped by customer and in item side we use @joinColumn with name of customerId varible

2.4--) many to many:Set<customer>--->Set<Item>

-in many to many we use Set instead of List

-if we are doign unidirectional then we jsut have to use @many to many at customer side then it automatically create third table

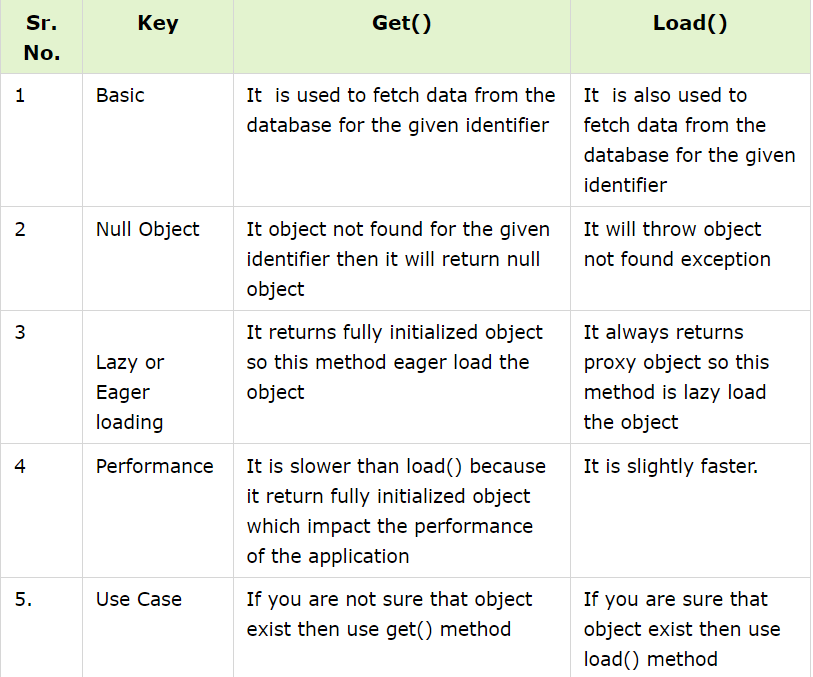
-if we are not satisfied with default implementation we can use @joinTable.

-if we have to do bidirectinal mapping then in item side create object of customer set and give mapped by

3)Scope of hibernate Entity

-

4) **Get vs Load in hibernate**



5)The N+1 query problem occurs when an ORM, like Hibernate, executes one query to retrieve the parent entity

and N queries to retrieve the child entities. This can happen when you use FetchType.LAZY for your entity associations.

We can solve this by uing left join fetch using @ query annotationwith single query

-and we can use @EntityGraph(attributePath=”listOfEmployee”)

Ex

@Query(select d from department d left join fetch p.listofEmployee

Private List<Department>findWithNPLusONe();

**6)State or Life cycle of Hibernate State**

-Transient State-when we create new object and value set to this object our object is in transient state

-Persistent-when you want to save your object in db and then your object is associated with session and db so that is persisitent state

-Detached-if you perform any update on that object it wont change in db that is detached state

-Remove-It will remove object

7)What is store procedure how we can call them through data jpa

8)Rollback in Hibernate:

In Hibernate, a rollback is a mechanism to undo the changes made during the current transaction and restore the state of the database to what it was before the transaction started. Transactions in Hibernate are managed through the Session interface, and you can use the rollback method to perform a rollback.

Session session=getSessionFactory().openSession();

Transaction tx=null;

Try{

Tx=session.beginTransaction();

}

Catch(Exception e){

If(tx!=null){

Tx.rollBack();

}