## Hibernate 4,5,6

|  |  |
| --- | --- |
| In hibernate 4 | In hibernate 5 onwards |
| Old way to create session factory | New way to create session facotry |
| session.save()is deprecated | Instead use ses.persist() |
| Old is from javax.persistence | For hib 6, java 11 Is mandatory  From hib 6+ All import statements from Jakarta.persistence |
| For primary key auto generation use  @Id   @GeneratedValue(strategy = GenerationType.AUTO) | @Id  @GeneratedValue(strategy = GenerationType.IDENTITY) |
|  |  |
|  |  |
|  |  |

**Old way**

Configuration cfg = new Configuration**()**.addResource**(**"Employee.hbm.xml"**)**.addResource**(**"Address.hbm.xml"**)**;

SessionFactory sessionFactory = cfg.buildSessionFactory**()**;

Session session = sessionFactory.openSession**()**;

**New way to create session factory object**

private static StandardServiceRegistry registry;

private static SessionFactory sessionFactory;

public static SessionFactory getSessionFactory() {

if (sessionFactory == null) {

try {

// Create registry

registry = new StandardServiceRegistryBuilder().configure().build();

// Create MetadataSources

MetadataSources sources = new MetadataSources(registry);

// Create Metadata

Metadata metadata = sources.getMetadataBuilder().build();

// Create SessionFactory

sessionFactory = metadata.getSessionFactoryBuilder().build();

# Session operations

Session object = connection Object ++

L1 cache= session level cache

L2 cache= session factory level cache

Composition vs aggregation

Aggregation- means a group of houses in colony- there is no tight relationship among themselves,

Composition:-means there is a tight relationship- if house is destroyed all rooms are destroyed

### Ses.persist() example

Session s = sf.openSession();  
Transaction transaction = null;  
if(!s.getTransaction().isActive()){  
 System.*out*.println("current transaction is not active so creating new ");  
 transaction = s.beginTransaction();  
}  
s.persist(p);  
transaction.commit();

Mapping Strategies

1. Write as 2 pojos(only 1 can be entity) but store in single table using @Embeddable, @Embedded annotations

Like , Address(normal pojo), Employee(Entity) both these classes are separate but store them in single table

Refer 81st class notes



1. When to go for inheritance:- IS-A relationship- if one class wants to use the entire properties of another class then go for inheritance
2. When to go for has-A relationship- if one class wants to use only some properties of another class then go for

HAS-A relationship

1. Always ensure to keep common data in another table and refer using relationships
2. If entities are in IS-A relationship parent-child relationship then go for 1 separate table for each child, one separate table for parent (Best is Table per sub class -TPSC)

Ex:-

Here use annotations like @nheritance (strategy=InheritenceType.joined)



@PrimarykeyJoinColumn(name=”payment\_id”,referencedColumnName=”pid”)

OneToMany

@Entity  
public class Employee {  
 @OneToMany  
 List<Laptop> laptops;  
   
 @Id  
 @GeneratedValue  
 private Long id;  
  
 String name;  
  
 Double marks;  
  
}

One student have many employees, since in student tables as we cant have a list type,

Internally hibernate will create a separate table,

|  |  |
| --- | --- |
| Student\_id | Laptop\_id |
| S1 | L1 |
| S1 | L2 |

The above states, each student will have many laptops (s1 having 2 laps called l1,l2)

To stop this additional tables, we have to tell hibernate saying, BRO don’t create another table

Instead in laptop column itself we will create student id, use that , then we have to tell mapped by field

@OneToMany(mappedBy = "employee")  
List<Laptop> laptops;

## Mapping statements

1. One category will have many products & Many products belongs same category (here think in database row level)
2. One dept will have many student, each student belongs to one dept

## Cascade types

CascadeType.aLL- means whern we save Product , then save dependent Category object also to db

public class Product {  
 @ManyToOne(cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "category\_id")  
 private Category category;

@JoinColumn

**in employee class , due to @OneToOne, it will create a foreign key column named (default is dependent class entityname+dep class pk col name- address\_add\_id),**

**if u dont want default name and if u want only custome fk col name, then give that name in @joinColumn("fk\_add\_id")**

**now that new fk in emp table as fk\_add\_id**

