**Hibernate**

Configuration con = new Configuration().configure(“hibernate.cfg.xml”).addAnnotatedClass(Alien.class);

1. Configuration is a class which is use to configure the database url , username and password etc.
2. .configure(“hibernate.cfg.xml”) method gives the path to the “hibernate.cfg.xml” file where all the configuration have done.
3. .addAnnotatedClass(Alien.class) method gives the class which object hibernate push to the database i.e. it is a entity class.

SessionFactory is an interface , used to perform crud operation on entity layer.

SessionFactory sf=con.buildSessionFactory();

Session session=sf.openSession();//we need session object to do the operations live save, fetch etc data from database.

Transaction tx=session.beginTransaction();

alien=(Alien)session.get(Alien.class, 101);

tx.commit();

@Transient annotation is used to not push the variable/column to database.

@Emmbedable annotation adds the class variables to the entity class .The class reference is a non-static variable in the entity class and also creates the fields in the database table.

@OneToOne is use to map a entity which has associated with single instance of another entity.

@OneToMany is use to map a entity which has associated with multiple instance of another entity.

1. If I use only @OneToMany annotation , suppose we have two entities as Laptop and Student .So two tables have created and also another mapping table student\_laptop created.

@ManyToOne is use to map a entity which has multiple instances associated with single instance of another entity.

1. If I use @OneToMany annotation on one entity and @ManyToOne on another entity then also extra table created.
2. But, if I use @OneToMany(mappedBy=”student”), then in the laptop table extra column is created, which is a primary key on student table. So extra table doesn’t created.

@ManyToMany is use to map a entity which has multiple instances associated with multiple instance of another entity.

1. Similarly we have to use mappedBy to create the mapping from which table we want to.
2. mappedBy is applicable to @OneToMany and @ManyToMany

fetch=FetchType.LAZY

1. Here the fetching of data is lazy.
2. If you are fetching a student that has multiple laptops form two tables student and laptop , it gives the data of student only.
3. If you want to fetch the laptops you have to iterate over the laptops.
4. Student student = session.get(Student.class, 101);
5. System.***out***.println(student.getName());
6. List<Laptop> laptop = student.getLaptop();
8. for (Laptop lap : laptop) {
9. System.***out***.println(lap);
10. }
11. Here it fires two queries one for student and another for laptop.
12. Hibernate: select s1\_0.rollNumber,s1\_0.name from Student s1\_0 where s1\_0.rollNumber=?
13. ashu
14. Hibernate: select l1\_0.student\_rollNumber,l1\_0.lId,l1\_0.lName from Laptop l1\_0 where l1\_0.student\_rollNumber=?
15. Laptop [lId=4, lName=hp1]

Fetch= FetchType.EAGER

1. Here fetching of data is eager.
2. Here it fires one left join query and give both laptops and students.
3. Student student = session.get(Student.class, 101);
4. System.***out***.println(student);

Hibernate: select s1\_0.rollNumber,s1\_0.name,l1\_0.student\_rollNumber,l1\_0.lId,l1\_0.lName from Student s1\_0 left join Laptop l1\_0 on s1\_0.rollNumber=l1\_0.student\_rollNumber where s1\_0.rollNumber=?

Student [rollNumber=101, name=ashu, laptop=[Laptop [lId=4, lName=hp1]]]

Notes:-

1. By default fetch type is LAZY.
2. If want to change the fetch type then you have to make changes in @OneToMany(fetch=FetchType.EAGER) .

**Caching In Hibernate**

1. When we are fetching data from database then it hits the database and gives the data to application server and so on to client.
2. But, again the user or client want to fetch the data in the same session. So, it again goes to the database and hits the database and gives the data.
3. Hence, we can use cache memory to save the data in the memory for a session. So, when the client again want the data then from the cache memory we can fetch the data.
4. So, it gives good performance.
5. Caching in hibernate can done in 2 ways.
6. First Level Cache
7. Second Level Cache
8. The above example uses first level cache.
9. Suppose the client uses another session fetch the same data. But, the first level cache is only useful for the current session. Then the second level cache comes into use.
10. All the sessions in the application can share second level cache.
11. We use third party providers to use second level cache in hibernate. Ex : ehcache , os, swarm.
12. In pom.xml we have to add the dependency for ehcache.
13. By default the first level cache is enabled.
14. In hibernate.cfg.xml we have to add some properties to enable second level cache and mention the provider.
15. In entity class we have to apply some annotations like @Cacheble and @Cache to access the second level cache.

Get Vs Load

1. Get fires the query as you call it using session.
2. But load fires the query when you are using the object of Entity.
3. Before using the object load gives a proxy object or fake object.
4. But get gives the object weather you are using it or not.
5. We normally use get. But sometime we have to use load.
6. If during fetching data the object is not found in the database then get gives null.
7. But load gives object not found exception.

Note:- After using the data read in both get and load, if the object is not found then it gives null and object not found exception respectively.