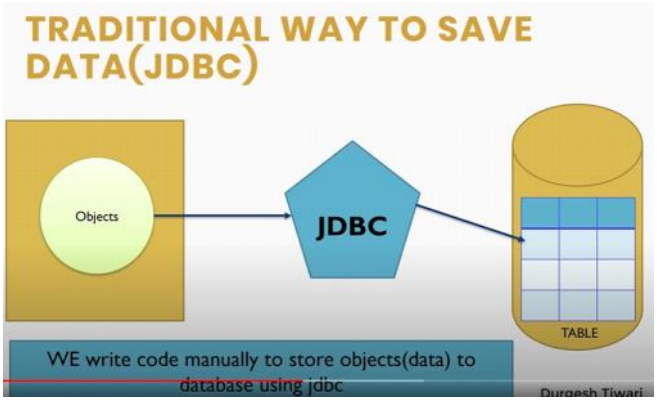
Saturday, December 26, 2020 3:50 AM

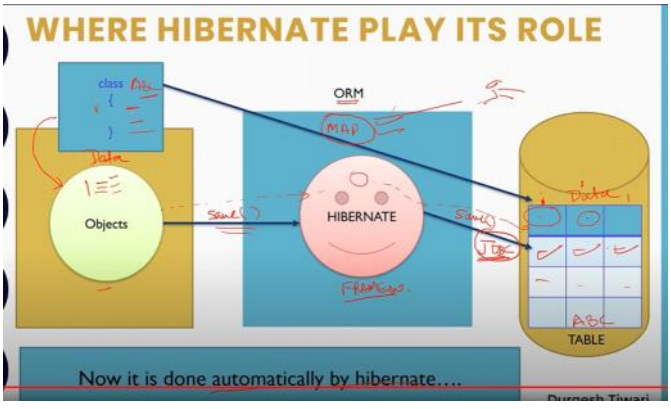
Hibernate

#0. Hibernate Tutorial | Course Overview | Prerequisite of hibernate course | hindi

We don't need to worry about sql query it will be performed by hibernate

- Hibernate is a Java framework that simplifies the development of Java application to interact with the database.
- Hibernate **is ORM (Object Relational Mapping)** tool.
- Hibernate is an Open source, lightweight.
- Hibernate is **a non-invasive** framework, means it won't forces the programmers to extend/implement any class/interface.
- It is invented by **Gavin King in 2001.**
- **Any type of application can build with Hibernate Framework.**





There are 3 objects in hibernate
1.Transient-> Normal object
2.Persistent- object to hibernate object
3.Detached-> hibernate to normal object

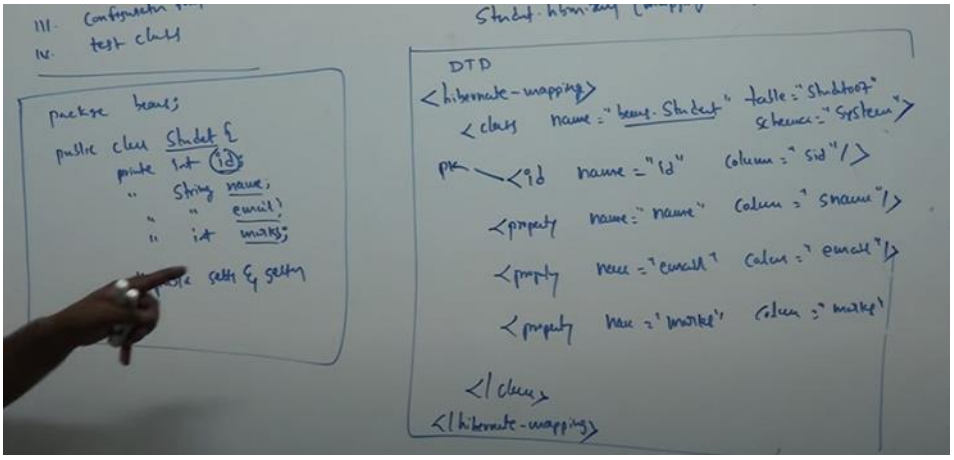
Here we have to make crud operation using object

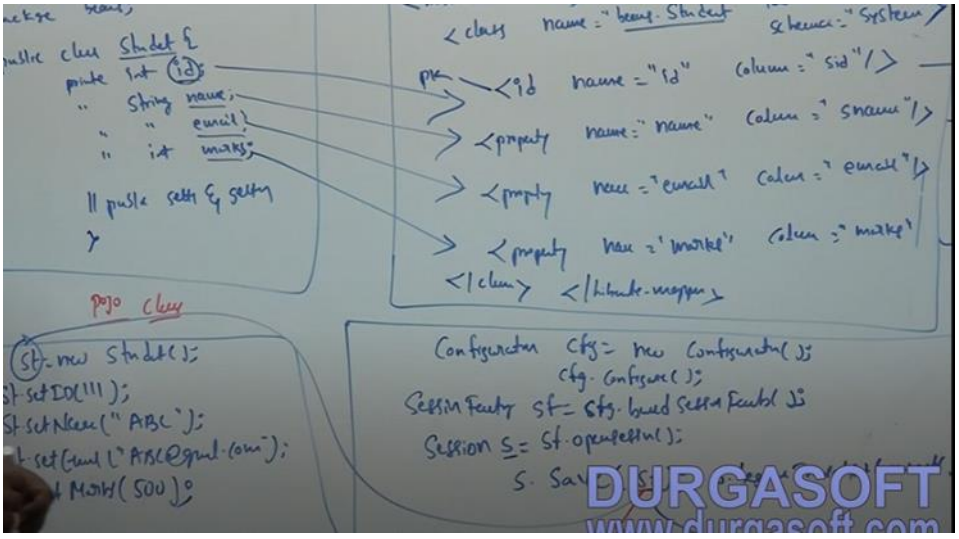
Transient object is normal object

Once we attached to hibernate.. This attached is called persistent object

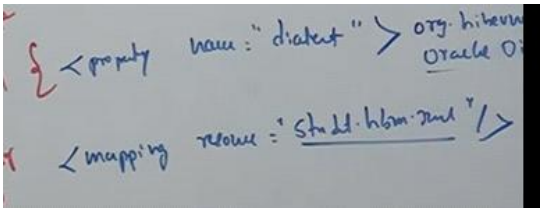
If we remove object from hibernate then it is called detached hiberante

If we do transaction.commit() then it will move database state or permanent state

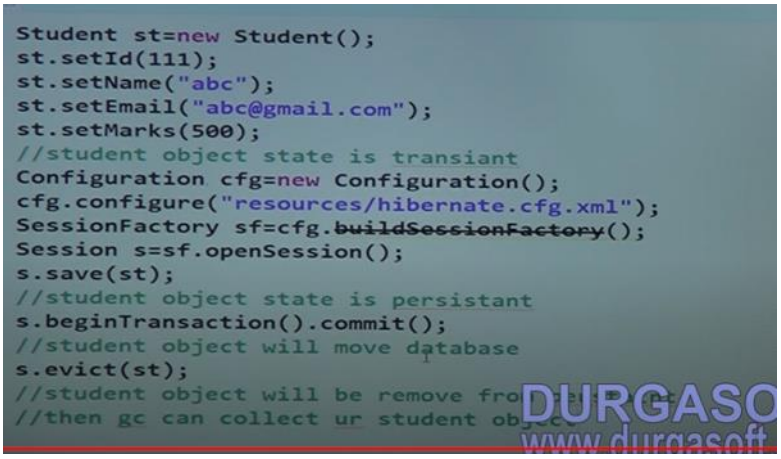




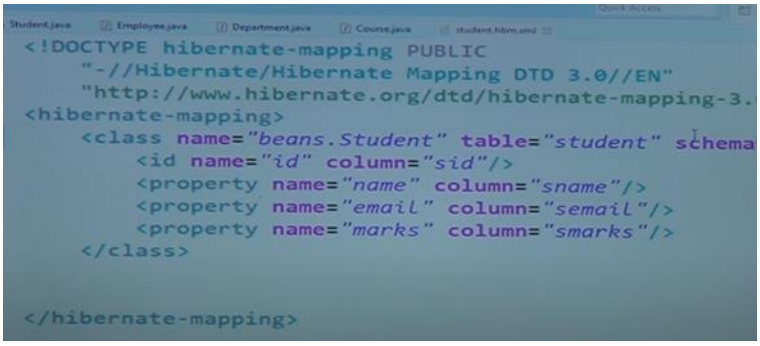
This pojo class is temporary state it can anytime collected by garbage collector
So to avoid that we need to add that in s.save

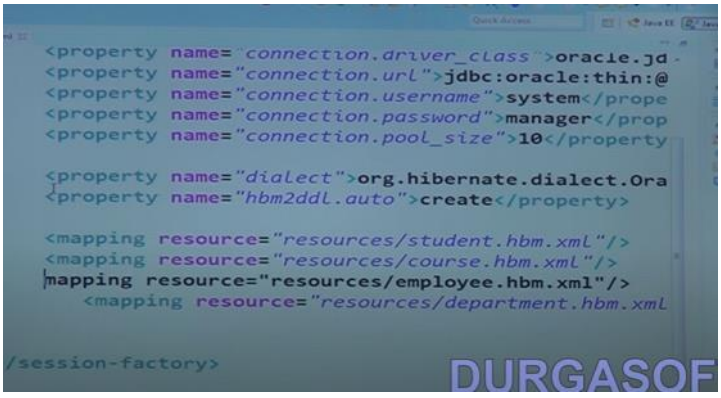


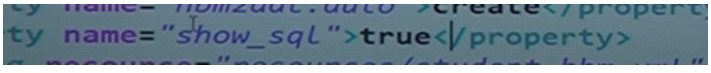
We need to attach mapping file in hibernate.config.file



Hbm2ddl.auot> create<
Then it will automatically create table with dropping earlier schema



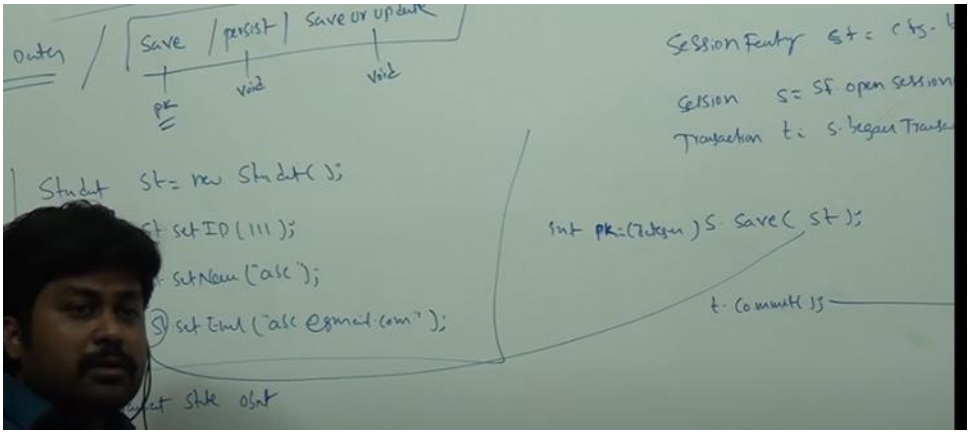




Internal table creation

Hbm2ddl.auot> Update<
Then it will automatically create table with dropping earlier schema

Validate
Must have that 5 coloumns



s.save(st)will return primary key
s.persist(st) will return void

For update records we have update and merge

Update will not work in cache method so then we need to do with merge method

Serialzible

```
public static void main(String[] args) throws IOException, Exception {
    Save s= new Save();
    s.id=10;
    File f= new File("C:\\Users\\kumarmur\\Desktop\\output.txt");
    FileOutputStream fl= new FileOutputStream(f);
    ObjectOutputStream dl= new ObjectOutputStream(fl);
    dl.writeObject(s); //state of object means s.id=10 will be stored


    FileInputStream fl1= new FileInputStream(f);
    ObjectInputStream dl1= new ObjectInputStream(fl1);
    Save s1=(Save) dl1.readObject();
    System.out.println(s1.id);

}

static class Save implements Serializable { // we cannot allow to store object bydefault
    // it can be used for malicious purpsoe so we use serialize interface
    int id;
```

If we excute saveorupdate(st) opeartion

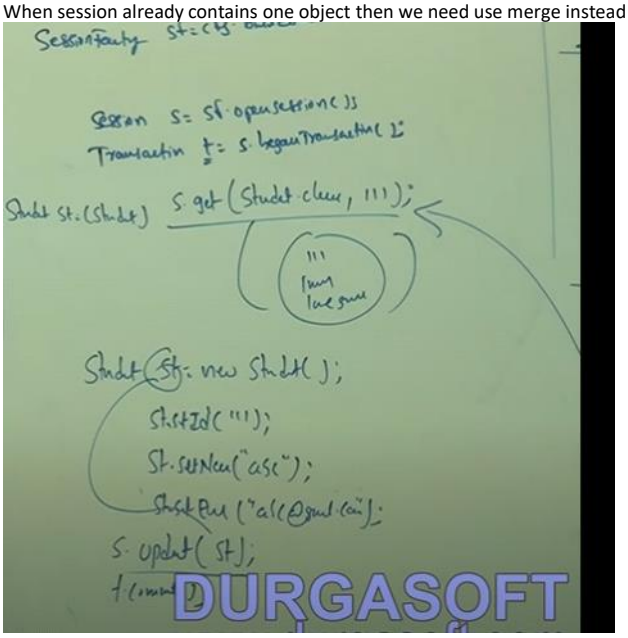
First it will run select query on that primary id;
Then it will compare

Using s.update(St);
s.merge(st);

Limitation
We can only update non primary keys
It is not possible to update one coloumn

When to use merge

When session already contains one object then we need use merge instead update otherwise it will throw exception of duplicate



```
Transaction t = session.beginTransaction();

session.get(Student.class, 111);

Student st=new Student();
st.setId(111);
st.setName("ABC");
st.setEmail("ABC@GMAIL.COM");
st.setAddress("HYD");

session.update(st);
t.commit();
session.close();
sf.close();
System.out.println("update success");
```

```
public static void main(String[] args) {

    Configuration cfg = new Configuration();
    cfg.configure("resources/oracle.cfg.xml");
    SessionFactory sf = cfg.buildSessionFactory();
    Session session = sf.openSession();
    Transaction t = session.beginTransaction();

    Student st=new Student();
    Student st=new Student();
```

```
Student st=new Student();

st.setId(222);

session.delete(st);
t.commit();
session.close();
sf.close();
System.out.println("");
```

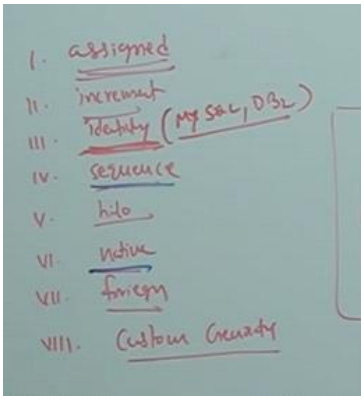
```
Object o=session.get(Student.class, 111);

Student st=(Student)o;
System.out.println(st.getId());
System.out.println(st.getName());
System.out.println(st.getEmail());
System.out.println(st.getAddress());


session.close();
sf.close();
System.out.println("select success");
```

It will do select operation only if when we do get of non primary key in session.load(Student.class, 11);

Primary key auto genrated
1.
Assigned (default)-> It is handle by user itself

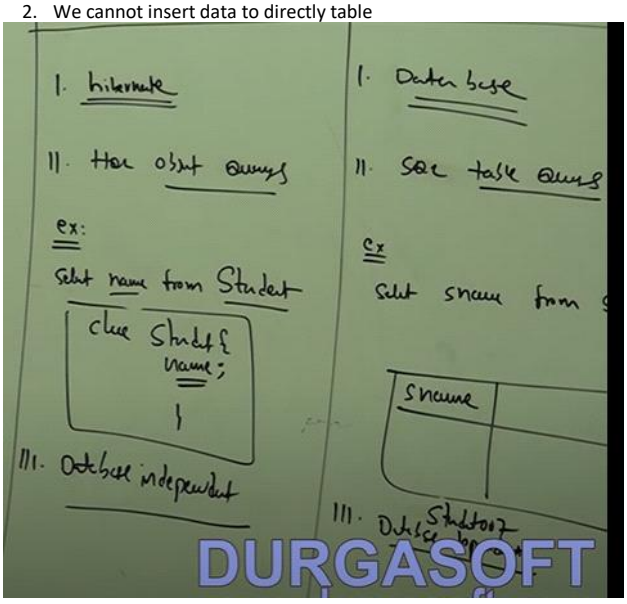
2.Increment-> It will do select max(id) from record and then update by incrmeneting by 1

```
<hibernate-mapping>
    <class name="beans.BookMovie" table="tbc
    <id name="id">
        <generator class="assigned"/>
    </id>
    <property name=""></property>

    </class>
```

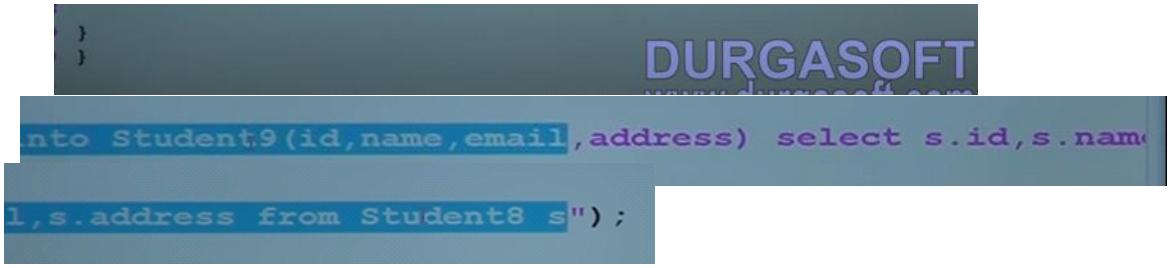


HQL-> Hiberante Query Language

1. We can shift one table data to other table
2. We cannot insert data to directly table




```
    Transaction t=s.beginTransaction();

    //insert

    //one table data we have to insert into another tab

Query q=s.createQuery("insert into Students(id,name,ema
int i=q.executeUpdate();
t.commit();
sf.close();
System.out.println("insert success");


}
}
```



Int i=q.excuteUpdate();
I is how many rows affected by this query

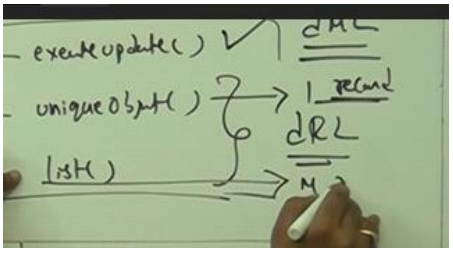
Update Hql -> Resolve merge and update problem-> That earlier we have to update all value of coloumn and cannot upadate primary key

```
Session session=sf.openSession();
Transaction t=session.beginTransaction();
String hql="update Student set name='ABC',email='ABC@y
Query query=session.createQuery(hql);
int i=query.executeUpdate();
System.out.println(i);
session.close();
```

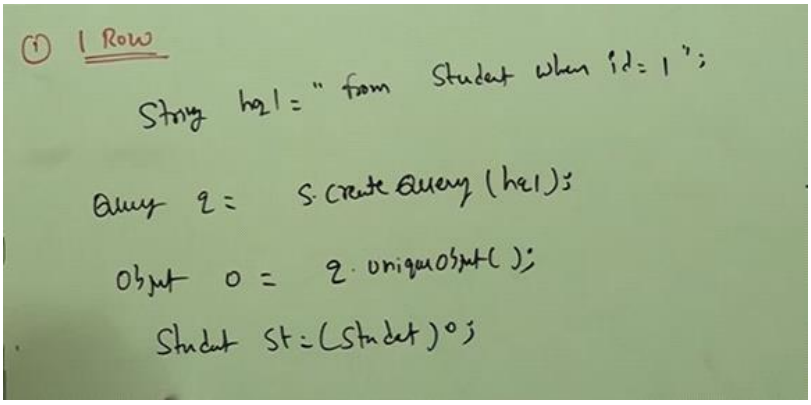


```
public static void main(String[] args) {
    Configuration cfg=new Configuration();
    cfg.configure("resources/oracle.cfg.xml");
    SessionFactory sf=cfg.buildSessionFactory();

    Session session=sf.openSession();
    Transaction t=session.beginTransaction();
    String hql="delete Student where id=555";

    Query query=session.createQuery(hql);
    int i=       query.executeUpdate();
    t.commit();
    System.out.println(i);
    session.close();
```


Select single row operation->

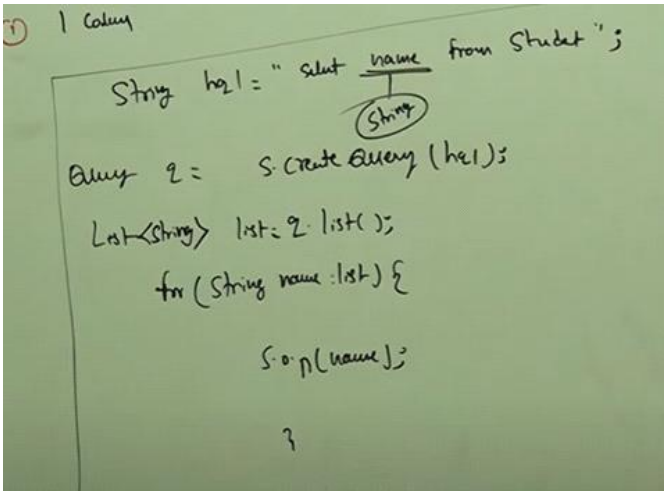


Multiple record= using list method
If only one record then we can go with unqiue record




```
Configuration cfg = new Configuration();
cfg.configure("resources/oracle.cfg.xml");

SessionFactory sf = cfg.buildSessionFactory();
Session session = sf.openSession();

String hql="from Employee where id=111";

Query q=session.createQuery(hql);
Employee emp=(Employee) q.uniqueResult();
        System.out.println(emp.getId());
        System.out.println(emp.getName());
        System.out.println(emp.getEmail());
        System.out.println(emp.getSalary());
```

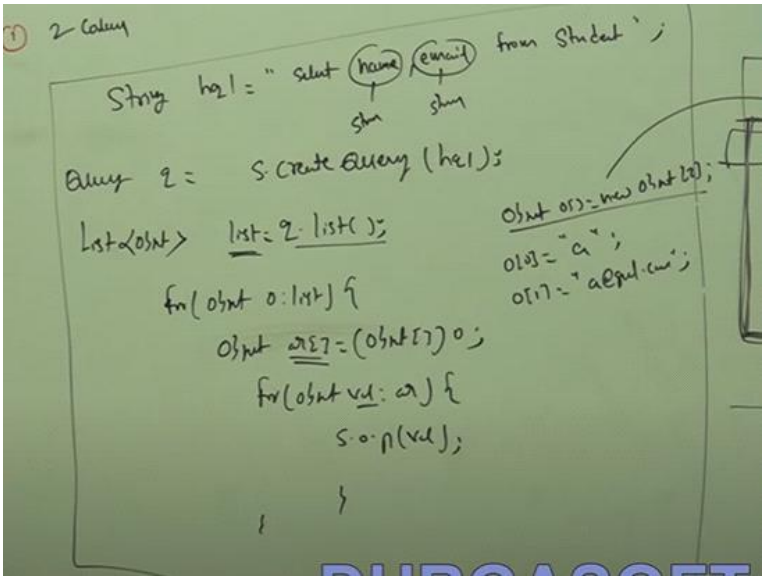
One single object of student then we will use unique object




```
Configuration cfg = new Configuration();
cfg.configure("resources/oracle.cfg.xml");

SessionFactory sf = cfg.buildSessionFactory();
Session session = sf.openSession();

String hql="select name from Employee";
Query q=session.createQuery(hql);
List<String> list=  q.list();
for(String name:list)
{
    System.out.println(name);
}
session.close();
sf.close();
```

If we want to get 1 whole coloumn above one

If we want to get two column then it will convert to object of arrays




```
SessionFactory sf = cfg.buildSessionFactory();
Session session = sf.openSession();

String hql="select name,email from Employee";
Query q=session.createQuery(hql);
List<Object> list=  q.list();
for(Object o:list)
{
    Object ar[]=(Object[])o;
    System.out.println(ar[0]);
    System.out.println(ar[1]);

}
session.close();
sf.close();
```

Whole object

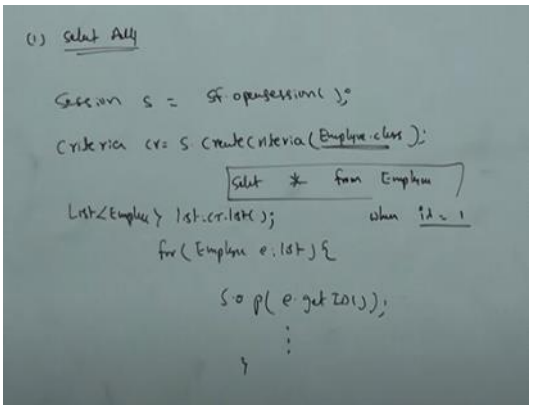
```
SessionFactory sf = cfg.buildSessionFactory();
Session session = sf.openSession();

String hql="from Employee";
Query q=session.createQuery(hql);
List<Employee> list=    q.list();
for(Employee emp:list)
{
        System.out.println("ID:"+emp.getId());
        System.out.println("NAME:"+emp.getName());
        System.out.println("EMAIL:"+emp.getEmail());
        System.out.println("SALARY:"+
}
session.close();
```

Aggregate or sum

```
String hql="select avg(salary) from Employee";
Query q=session.createQuery(hql);
    double avg=(Double)q.uniqueResult();
System.out.println("salary avg="+avg);
session.close();
```

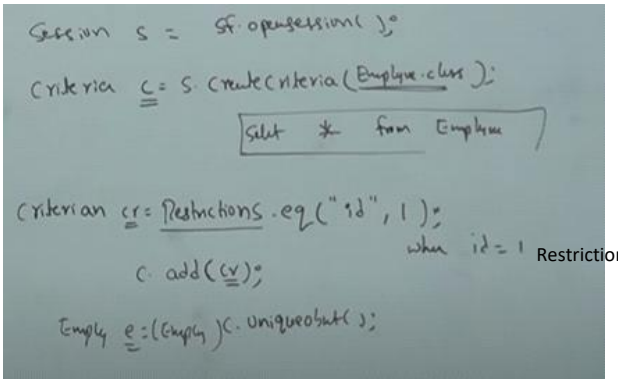
HQL we can use any CRUD operation

Criterias->**we can use only select operation with condition**

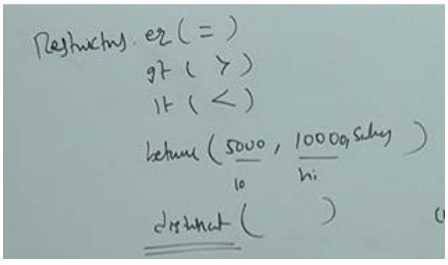


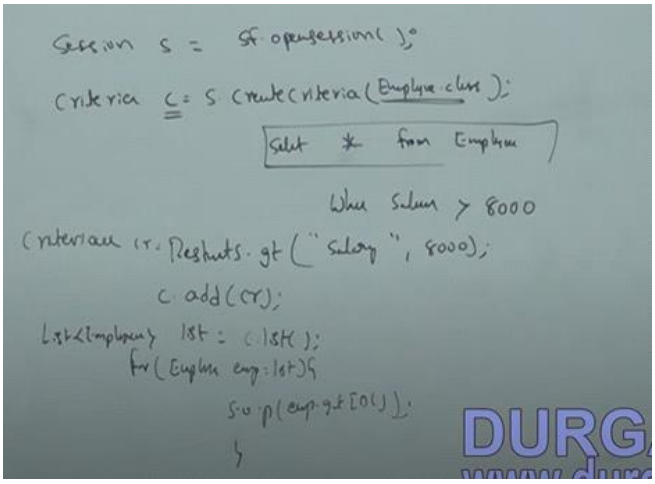
When we want to apply select * from employee where id=1; where condition then we will apply restriction

Crieterian means condition



Restriction means where Id=1



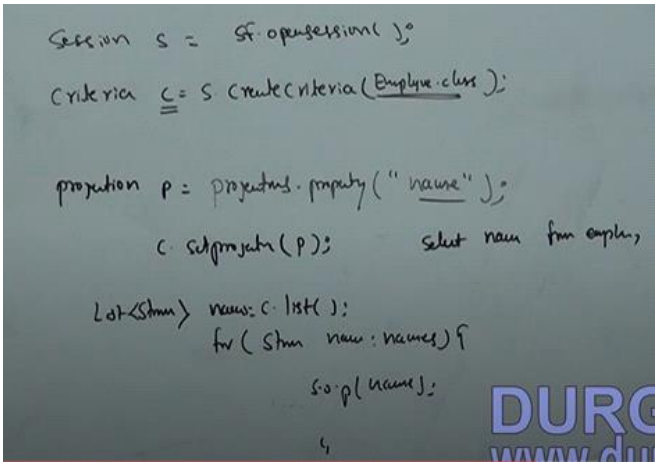


```
SessionFactory sf=cfg.buildSessionFactory();

Session session=sf.openSession();

Criteria c= session.createCriteria(Employee.class);

        //where =
Criterion cr=Restrictions.eq("id", 111);
c.add(cr);
Employee emp=(Employee) c.uniqueResult();
System.out.println(emp.getName());
```

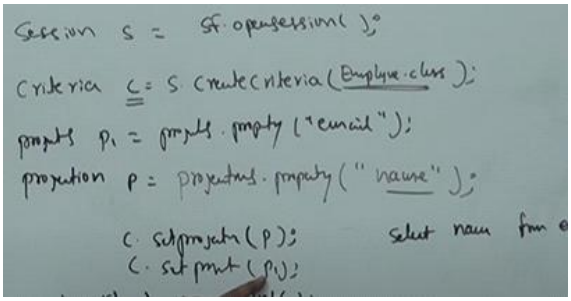
```
        //where =
//Criterion cr=Re
//where >
Criterion cr=Restrictions.gt("salary",70000);
c.add(cr);
    List<Employee>  emplist=c.list();
    for(Employee emp:emplist)
System.out.println(emp.getName()+";
```

Projections->
If we want only one column or max salary or aggregate one value we use projections api

Select name from employee only one coloumn ->Projection.porperty("name");
Partial record->

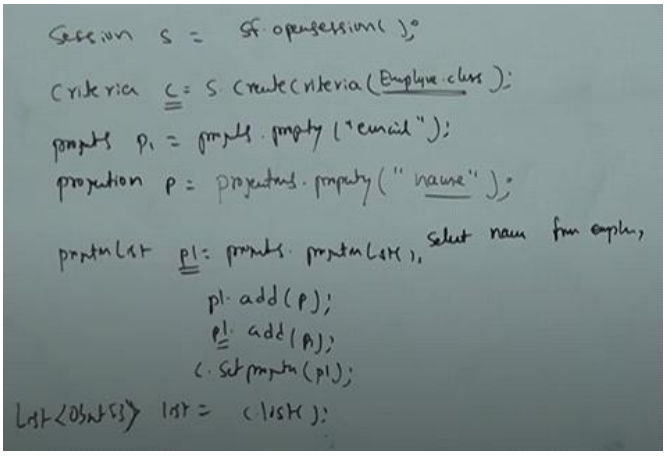


For one crietria we can apply multple criteria
But at a time only one projection

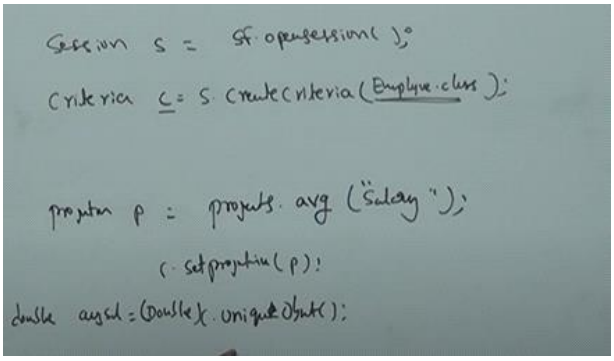


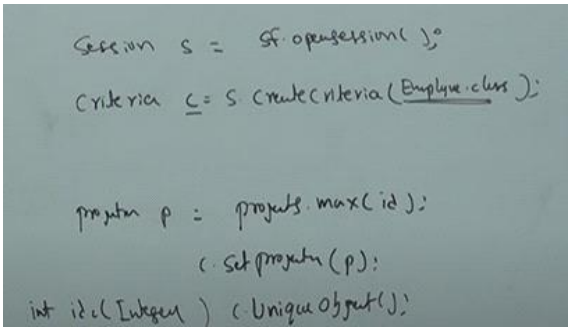
Above image is wrong as first projection will get overide by second one

If we want multiple column then we need to use projection list class



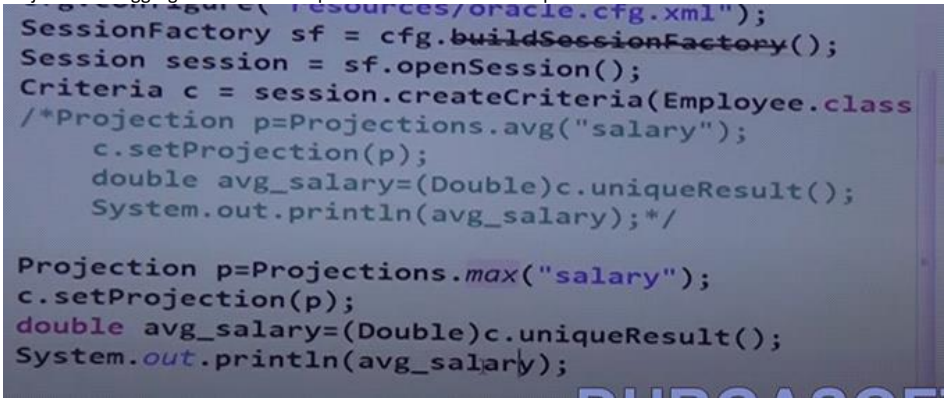
If we want to aggregate or do sum





Restriction are for condtion (=,>,<, between, like)

Projection for aggregate function and particular coloumn select operation



Cache support

One user
All user
One instance

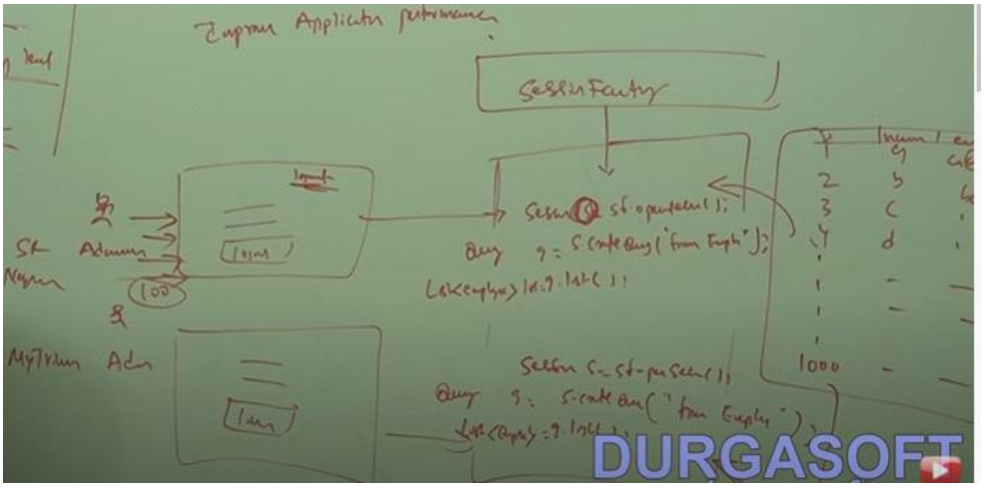
If Constant table if we want playerlist everytime

If constant it will everytime hit database..

View-> Application-> Database
If we want to see
Playerlist
It will hit everytime
To database
10000 hit call from application to database. Then we require I lakh connection and closed.. How much processing will waste

Cache will take temporary storage and it will return 1lakh time..
Cache reduce databse call. Improve application performance

Session cache->
It is useful for one user login to application.
Fetch information for data



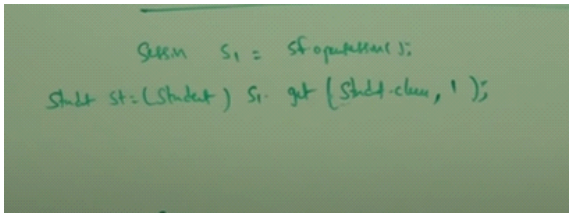
It will fetch only one time per session for one user
For remain 99 call they call from cache only from same session object

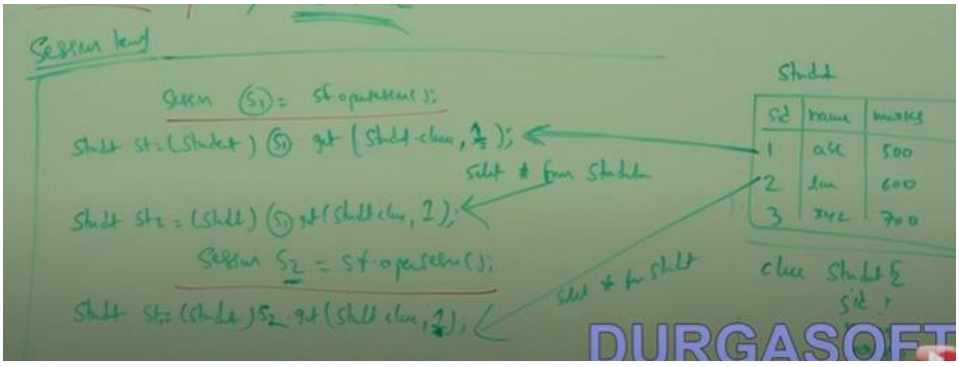
Different user has different session that it will cache
Example- If we login to gmail

1. First time it will retrive from database
2. But after that when we refresh it will return from session cache
3. Until logout only single call to database.
4. In session one time call datatabse

Session factory->
It is also called second level caching
If there are two user then one will hit database then it will avaible for second user also
As it is for session factory level

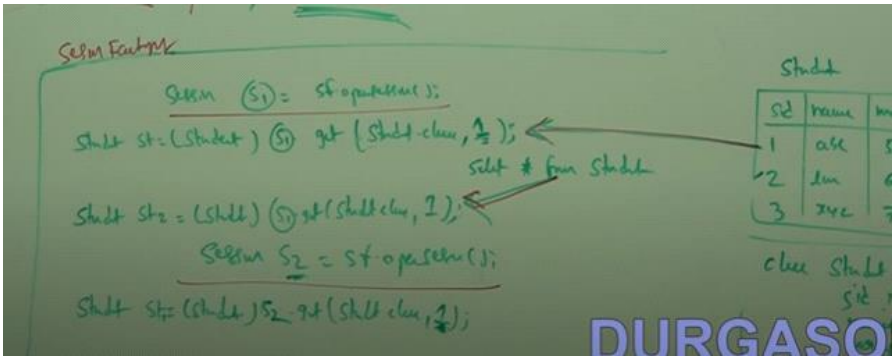
Query level-> Same type query same query like maximum salary.

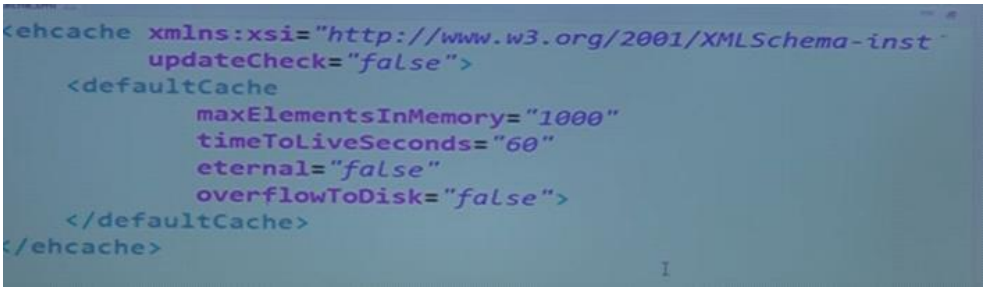




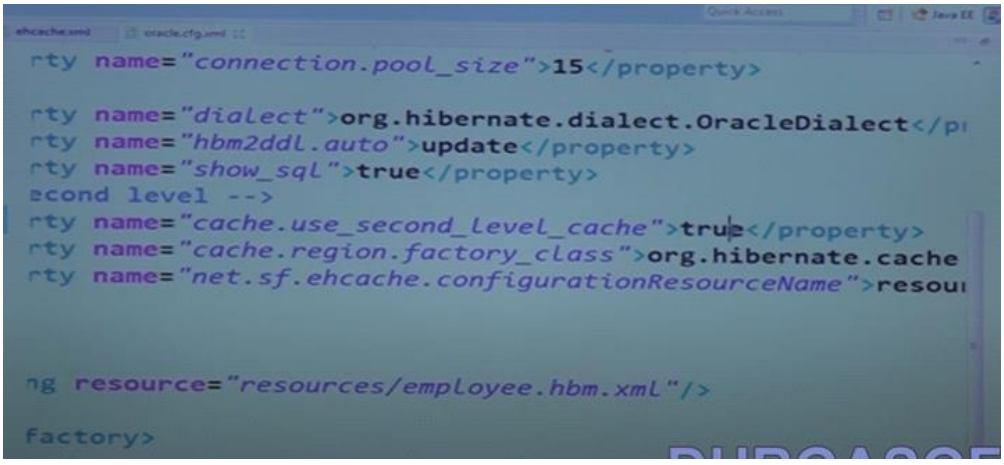
Two call for session cache for two session factory

But sesion factory cahce then same data availbale for both session factory



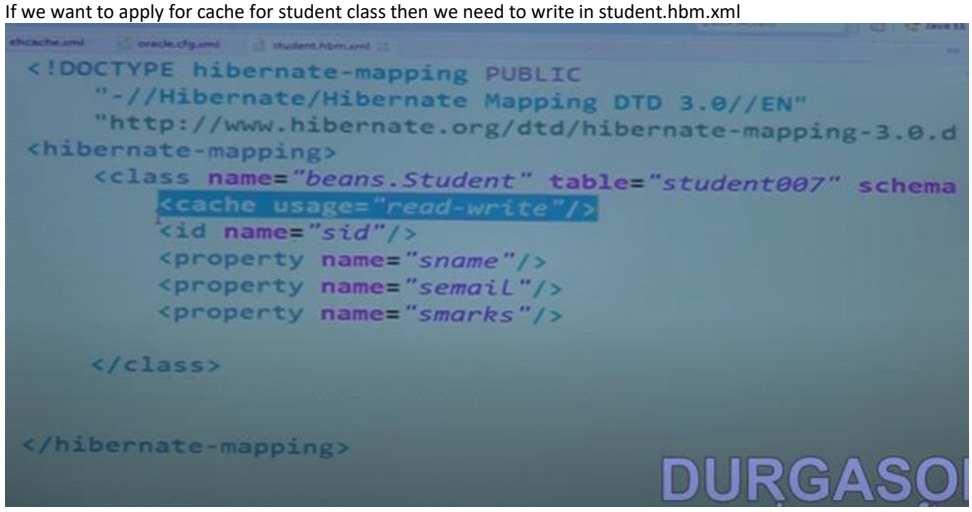


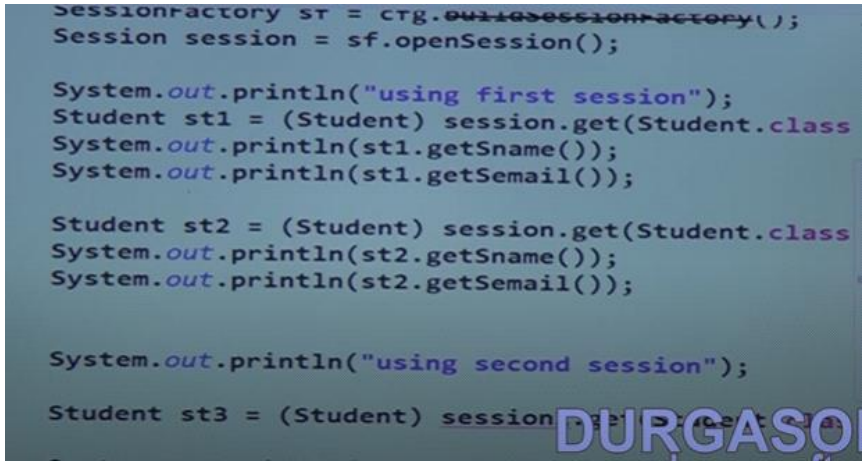
To add session level factory we need to add this
Maxelementinmemory-> Number of objects



We need to add this in configuration file

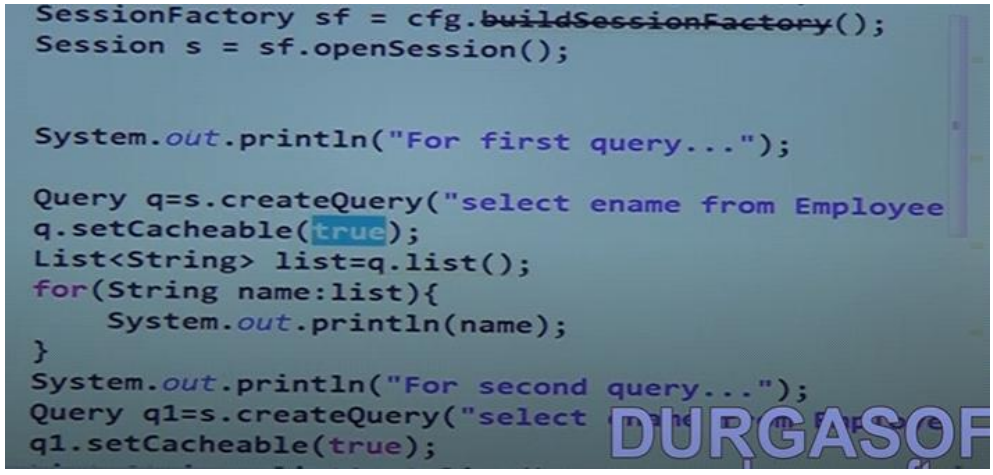
If we want to apply for cache for student class then we need to write in student.hbm.xml





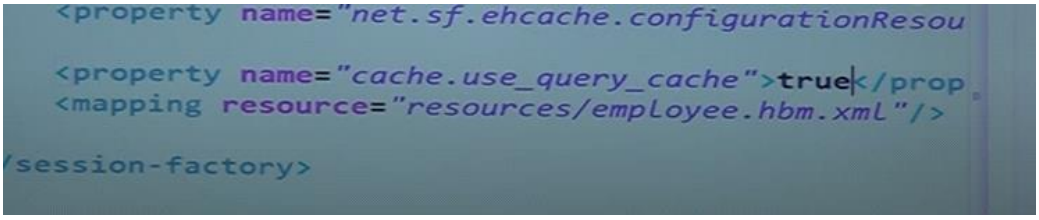
For this st1 and st2 it will fire single query as same session in session cache

But in session factory only one sleect operation



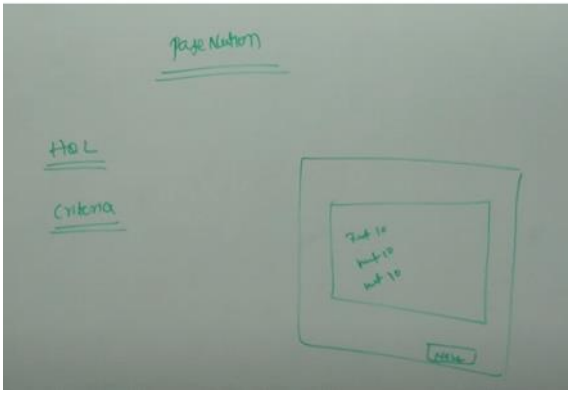
Query cache-> q.setcacheable(True)
It will store this query in cache

```
List<String> list=q.list();
for(String name:list){
    System.out.println(name);
}

System.out.println("For second query...");
Query q1=s.createQuery("select ename from Employee");
q1.setCacheable(true);

List<String> list1=q1.list();
for(String name:list1){
    System.out.println(name);
}
System.out.println("for third querys..");

Query q2=s.createQuery("select ename from Employee");
q2.setCacheable(true);
List<String> list2=q2.list();
```



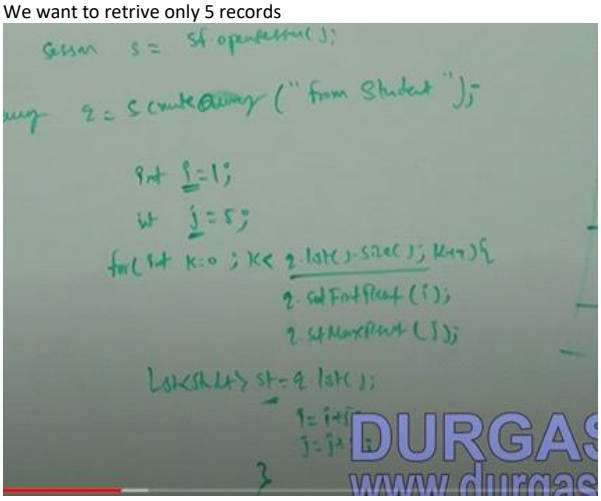

```
e1.setEid(111);
e1.setEname("abc");
e1.setEmail("abc@gmail.com");
e1.setSalary(500000);

Employee e2=new Employee();
e2.setEid(222);
e2.setEname("lmn");
e2.setEmail("lmn@gmail.com");
e2.setSalary(600000);

s.save(e1);
s.save(e2);
s.beginTransaction().commit();
System.out.println("success");
```

Pagination

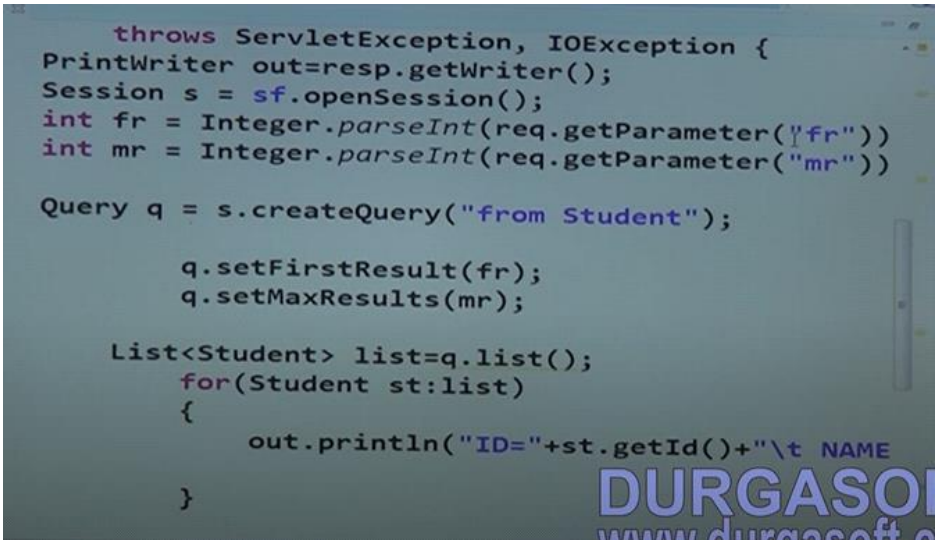


If we want to retrive limited value

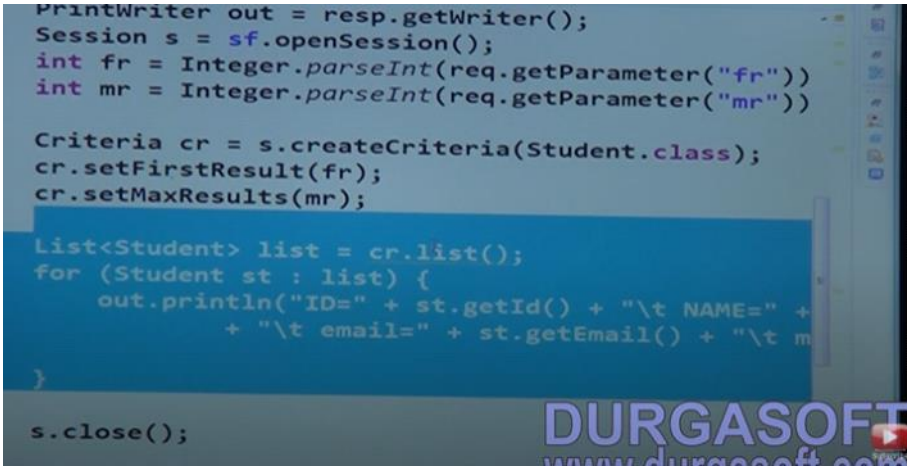
We want to retrive only 5 records



q.setfirstsize(0)
q.setmaxresult(5)



Criteria pagination



@Transaction is session management

```
timeToIdleSeconds="300"
timeToLiveSeconds="600">
```

If it has not been used for 300 seconds
Or after stays in cache for 600 seconds
When echache exceed from its configured size ehcache removed expired elements
If that doesn't clear enough space it cleasr object from ehcache
By default it remove LRU elements

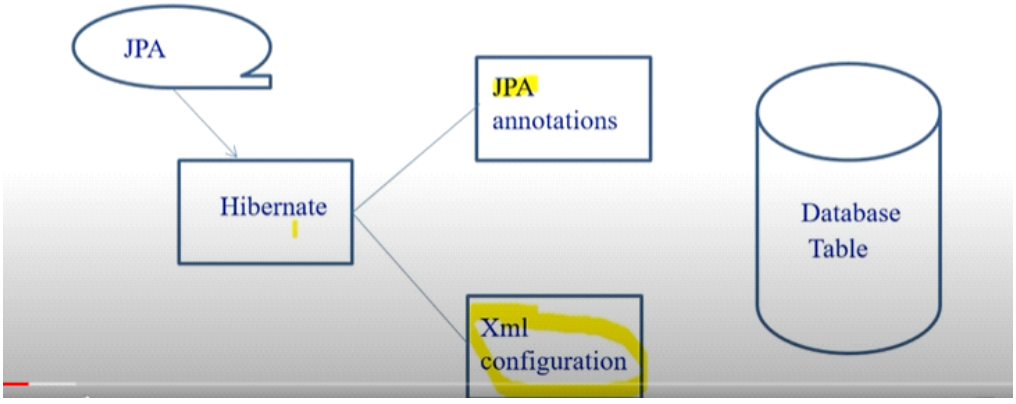
Second level caching is we put on entity or whole object

Default cache is made

```
@Configuration
@EnableCaching
public class CacheConfig {

    @Bean
    public CacheManager cacheManager() {
        SimpleCacheManager cacheManager = new SimpleCacheManager();
        List<Cache> cacheList = new ArrayList<Cache>();
        cacheList.add(new ConcurrentMapCache("userCache"));
        cacheList.add(new ConcurrentMapCache("addressCache"));
        cacheManager.setCaches(cacheList);
        return cacheManager;
    }
}
```

Nosql->
We call collection to table

- **How** ?
- It provides JPA implementation hence we can use JPA annotations as well as xml configurations to achieve this mapping.



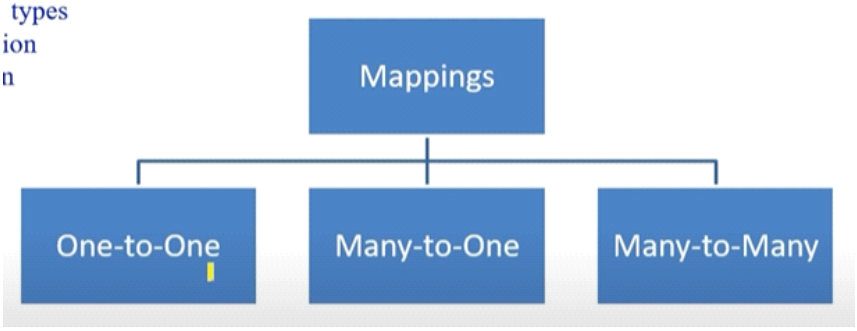
- **Why Hibernate ?**
- Hibernate eliminates all the boiler-plate code that comes with JDBC.
- It supports HQL with is more Object oriented.
- It provides transaction management implicitly.
- Hibernate throws JDBCException or HibernateException which are the unchecked exceptions, so we don't need to worry about handling using try and catch.
- Hibernate supports caching for better performance.

Q) Important Interfaces used in Hibernate

- **SessionFactory (org.hibernate.SessionFactory)** – Instance of this is used to retrieve Session objects for database operations. We need to initialize that once and can cache it to reuse it again and again. Its like one SessionFactory object per database connection. Like 1 for mysql, 1 for oracle.
- **Session (org.hibernate.Session)** – Its factory for transaction, it's used for connecting application with persistant store like hibernate framework / DB. It is used to get a physical connection with the database. It also provides methods for CRUD operations.
- **Transaction (org.hibernate.Transaction).** – This specifies single / atomic units of work

```
SessionFactory factory = metadata.getSessionFactoryBuilder().build();
Session session = factory.openSession();
Transaction t = session.beginTransaction();

session.save(persistantObj);
    t.commit();
factory.close();
    session.close();
```

types
ion
n



Many to Many

```
class Student{

@ManyToMany(targetEntity = Degree.class, cascade = { CascadeType.ALL })
@JoinTable(name = "DegreeStudentThirdTable",
           joinColumns = { @JoinColumn(name = "StudentId") },
           inverseJoinColumns = { @JoinColumn(name = "CertificateId") })
private List<Degree> degrees;

}
```

Q) What are hibernate configuration file

contains database specific configurations and used to initialize SessionFactory.

Conventionally, its name should be hibernate.cfg.xml

If u need to connect to SQL then create another one here.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">codedecode</property>
        <property name="connection.password">codedecode</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="employee.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

Q) What are hibernate mapping file

The mapping file name conventionally, should be class_name.hbm.xml.

hibernate-mapping : root element
class : specifies the Persistent class.
id : specifies the primary key attribute in the class.
generator : used to generate the primary key.
property : specifies the property name of the Persistent class.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd

 <hibernate-mapping>
  <class name="com.codedecode.Employee" table="empTable">
    <id name="id">
     <generator class="assigned"></generator>
    </id>

    <property name="firstName"></property>
    <property name="lastName"></property>

  </class>

</hibernate-mapping>
```

Q) difference between openSession and getCurrentSession

getCurrentSession() method returns the session bound to the context.
Since this session object belongs to the context of Hibernate, it is okay if you don't close it. Once the SessionFactory is closed, this session object gets closed.

While

openSession() method helps in opening a new session.
`You should close this session object once you are done with all the database operations. And also, you should open a new session for each request in a multi-threaded environment.

- get() loads the data as soon as it's called whereas load() returns a proxy object and loads data only when it's actually required, **so load() is better because it support lazy loading.**
- **Since load() throws exception** when data is not found, we should use it only when we know data exists.
- We should use **get() when we want to make sure data exists** in the database.

Q) hibernate caching – Second level cache

- Hibernate Second Level cache is disabled by default but we can enable it through configuration.
- Currently EHCache and Infinispan provides implementation for Hibernate Second level cache and we can use them.