**Copy Hibernate jars to class path**

**Create hibernate.cfg.xml**

**<hibernate-configuration>**

**<session-factory>**

**<property** name="hibernate.bytecode.use\_reflection\_optimizer"**>**false**</property>**

**<property** name="hibernate.connection.driver\_class"**>**com.mysql.jdbc.Driver**</property>**

**<property** name="hibernate.connection.password"**>**password**</property>**

**<property** name="hibernate.connection.url"**>**jdbc:mysql://localhost:3306/mkyong**</property>**

**<property** name="hibernate.connection.username"**>**root**</property>**

**<property**

//dialect is something specific to database provider

name="hibernate.dialect"**>**org.hibernate.dialect.MySQLDialect**</property>**

**<property** name="show\_sql"**>**true**</property>**

**<mapping** resource="com/mkyong/common/Stock.hbm.xml"**></mapping>**

**</session-factory>**

**</hibernate-configuration>**

**Create model object using mandatory annotations @Entity and @Id**

Create TestApp

Create model object using new and initialize fields.

Create SessionFactory Object

SessionFactory sessionFactory = **new** Configuration().configure().buildSessionFactory(“hibernate.cfg.xml”);

Create Session Obeject

Session session= sessionFactory.openSession();

Begin Transaction

Session.beginTransaction();

Save the object to DB

Session.save(pass the model object);

Commit the taransaction

Session.getTransaction().commit();

Session. Close();// check ur Database now ☺ ☺

**Look the hibernate.cfg.xml**

Look for property name hhd2ml.auto

Default is create//ever time session factory create its drop the previous created schema/table and create entirely new schema by checking mapping class configuration.

Change is to update//only update table in dB when any structural modification done.

**Difference @Entity (name=”USER”) with @Entity @Table (name=”USER”)**

For Hibernate query (HQL) with pic with entity name not with table name

And in both cases table name is same.

**Difference with placing @Column annotation over field and placing over Getter**

Pic the value with field in first and from getter in second

**Some annotation**

@basic vs @Transient//don’t want to save the obeject

By defaut is basic

@Temporal(TemporalType.Date) // only save Date not time

@Temporal(TemporalType.Time) // only save time

By default is TemporalType.TimeStamp ☺ ☺

@Lob telling Hibernate to create large field for this in table. (Large Object)

**Getting record from Table:**

SessionFactory sessionFactory = **new** Configuration().configure().buildSessionFactory(“hibernate.cfg.xml”);

Session session= sessionFactory.openSession();

Session.beginTransaction();

Session.get(User.class,1)//first parameter is class and second is primary Id value

Is extract Obejct

So: User user= (User) session.get(User.class,1)

Sysout(user fields);// always get value from getter.

**Natural key vs Surrogate key**

Natural key like user can register with using login id and login id have business significance.

Surrogate key is something added as extra field just to maintain order not having business significance.

**@GeneratedValue**

Generate unique ids as new row added to table; it can be used for both natural and surrogate key.

@GeneratedValue(strategy=GenerationType.Sequence)

Its picks nextval (‘hibernate.sequence’) and assing to row

**Address in user table//Address in not a entity**

**Separate column for each address column in User table**

**Embedded Objects:**

Address as @Embedded in User table

And Address class should be @Embeddable

Be default hibernate create column name to embedded address.

We can make @column name in Address table to modify column name

Another we also have @AttributeOverrides which override column name in user table as in case we need two or more address.

**EmbeddedId:**

We can use other Embeddable object as our primary ID(group of column contribute to make user primary key)

**Separate table for each address column in User table**

**@EmbeddedCollection:**

Suppose we want to have list of address for a particular User Id

@ EmbeddedCollection //work also for single address.

List<Address> listOfAddress=new ArrayList<Address>();

It creates a new Address table with default table name and column names.

And add user id as foreign key in address table.

We can modify names using

**@JoinTable**(name=”User\_Address”, joinColumns=@JoinColumn(name=”UserId”));

Now if we want to add additional id to Address table as primary id.

Syntax:(I pick below form hibernate specific annotations)

@ EmbeddedCollection

@JoinTable(name=”User\_Address”, joinColumns=@JoinColumn(name=”UserId”));

**@GenericGenerator**(name="hilo-gen", strategy="hilo") // generate Id for collection ID

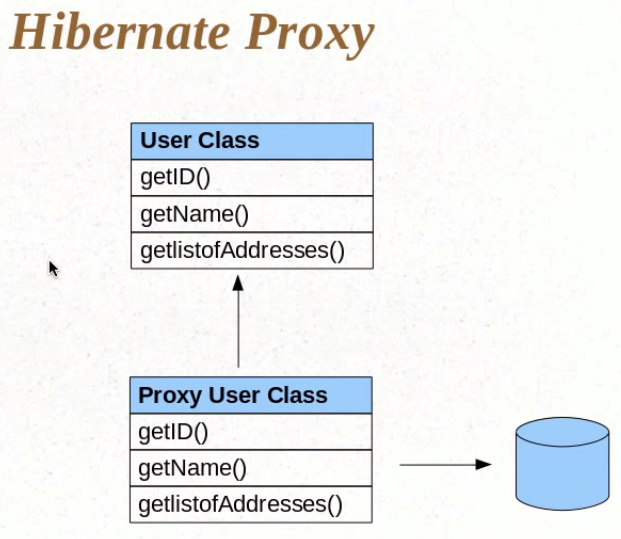
@CollectionId(columns = @Column(name="Address\_id"), generator = "hilo-gen", type = @Type(type="long"))

**Lazy Vs Eager Loading**

Fetch strategy for collection// by default is Lazy

While accessing object from table Hibernate create proxy with first level user field initialized.

Lazy: only initialize first member variable not collection. Initialize the list when access call to these collections.



@Elementcollection(fetch = Fetchtype.Eager) Eager: All collections initialize in advance.

Does hibernate also create proxy in case of Eager Fetching??? Yes as other collection may have Lazy approach.

**One to One Mapping**

User Entity and Vehicle Entity

In User class

**@OneToOne**

Vehicle vehicle;

Its add user ID of User as foreign key in Vehicle table.

**@OneTo Many**

@Jointable(name=”” join column=”” and inversejoin column=” ”)

Set<Vehicle> listOfVehicle =new ArrayList<Vehicle>();

Its create a new mapping table named User\_Vehicle(userId,vehicleId)

**ManyToMany**

User

@ ManyToMany

Set<Vehicle> listOfVehicle =new ArrayList<Vehicle>();

And

Vehicle

@ ManyToMany (mapedBy(name=” listOfVehicle)

Set<User> listOfUser =new ArrayList<User>();

In this case with mapedBy one extra table is created have (userId ad vechicle Id)

**@NotFound**

In case of these relationship if and reference data is not fount hibernate will throw exception and to avoid that we can use @NotFound annotation.

**@Cascade:**

User

@ OneToMany (cascade= CascadeType.SAVE\_UPDATE)

Set<User> listOfUser =new ArrayList<User>();

In this case if vehicle is automatically persisted with the persistence of user.

But with one modification in replace of session.save() we need to use session.persist();

**Inheritance☹happens between entities**

Single table strategy

Table per class

Joined strategy

Vehicle

Filed 1,field2,field 3

Two Weeheler extends Vechile

Field 1, field 2

Fous Weeler extend Vechile

Field 1,Field2

**Single table strategy**

Now session.save all Entities hibernate create single table by default

Dtype, V.1, V.2, V.3, TW.1, TW.2, FW.1, FW.2

Dtype is discriminator type which is by default class name (which can be configurable)

//problem is Empty columns in table

**Table per class**

@Inheritance(strategy=InheritanceType.TableperClass

User

When save all entities hibernate create three class

V.1, V.2, V.3

V.1, V.2, V.3, TW.1, TW.2

V.1, V.2, V.3, FW.1, FW.2

//No empty column but still have redundant data

**Table per class**

@Inheritance(strategy=InheritanceType.Joined

User

When save all entities hibernate create three class

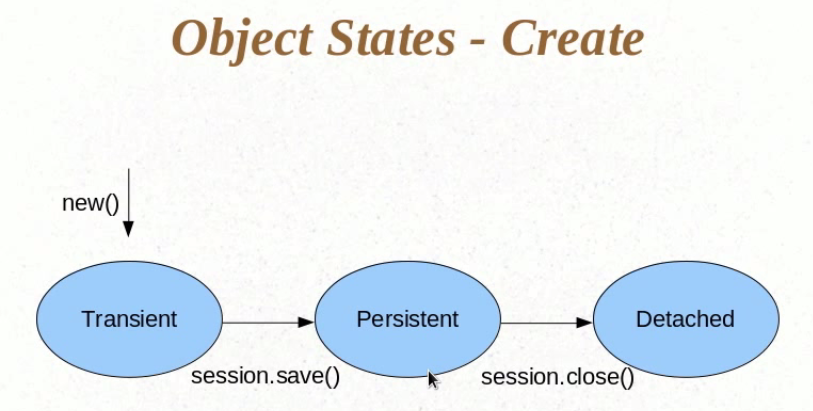
V.1, V.2, V.3

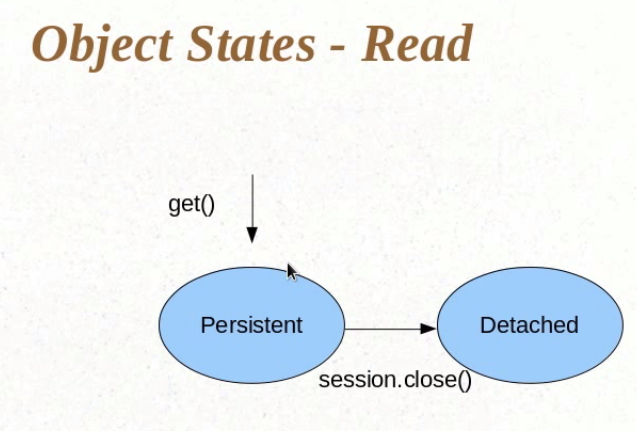
V.1, TW.1, TW.2

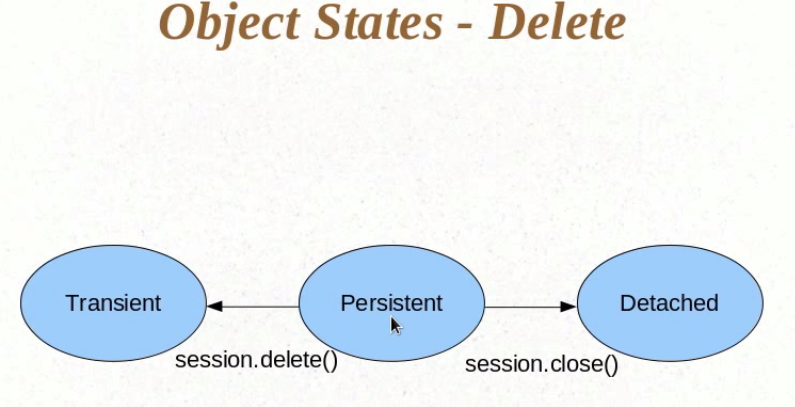
V.1, FW.1, FW.2

//Data redundancy is decreasing and no empty column(BEST APPROACH)

While retrieving data need to use join







Dirty Checking

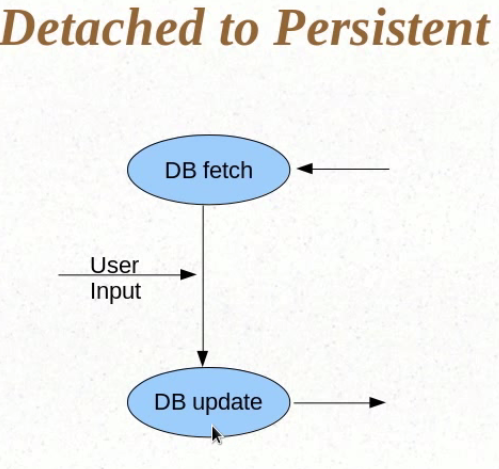
Hibernate Intelligently check the update of user object in persistence state

@Org.hibernate.Entity(Selectbefore=true)

Detached to persistence:

What happen when session.update(object) call hibernate is not aware the changes done to object so fire a update query even object state change not done.

To avoid this is select before = true its call a select query to make object in persistence state then put update query if object state changes happen.



**Query Object:**

Query query=session.createQuery(“from userdetails”);

List users= query.list();//List<User> list=(List<User>) query.list();

Query.setFirstResult(5);

Query.setmaxResult(7);

**Criteria Object:**

**First level cache: Session level or Query level cahche**

**Second level cache**

**@Cacheable**

**@Cache()**

**Query.set(true);**