Link: https://www.tutorialspoint.com/hibernate/hibernate\_interview\_questions.htm

https://www.javainterviewpoint.com/

-------------------------------------------------------------------------------------

1.>>What is JDBC?

JDBC stands for Java Database Connectivity and provides a set of Java API for accessing the relational databases from Java program. These Java APIs enables Java programs to execute SQL statements and interact with any SQL compliant database.

------------------------------------------------------------------------------------------

2.>What is ORM?

ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C# etc.

------------------------------------------------------------------------------------------

3.>What are the advantages of ORM over JDBC?

An ORM system has following advantages over plain JDBC

Sr.No. Advantages

1 Lets business code access objects rather than DB tables.

2 Hides details of SQL queries from OO logic.

3 Based on JDBC 'under the hood'

4 No need to deal with the database implementation.

5 Entities based on business concepts rather than database structure.

6 Transaction management and automatic key generation.

7 Fast development of application.

------------------------------------------------------------------------------------

4.>Name some of the ORM frameworks based on JAVA.

There are several persistent frameworks and ORM options in Java.

->Enterprise JavaBeans Entity Beans

->Java Data Objects

->Castor

->TopLink

->Spring DAO

->Hibernate

------------------------------------------------------------------------------------

5.>What is Hibernate?

Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as an open source persistent framework created by Gavin King in 2001.

It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

----------------------------------------------------------------------------------

6.>What are the advantages of using Hibernate?

Following are the advantages of using Hibernate:

Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.

<!-- Mapping class -->

<mapping class="com.java.sanjeev.Transaction" />

->Provides simple APIs for storing and retrieving Java objects directly to and from the database.

->If there is change in Database or in any table then the only need to change XML file properties.

->Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.

->Hibernate does not require an application server to operate.

->Manipulates Complex associations of objects of your database.

->Minimize database access with smart fetching strategies.

->Provides Simple querying of data.

<----------------------------------------------------------------------------->

</\* SQL queries are also known as native query\*\>

Session session = sessionFactory.openSession();

String deviceselectQuery = "SELECT \* FROM DEVICE WHERE (Host = ?)";

query = session.createSQLQuery(deviceselectQuery).

addEntity(Device.class).setParameter(0, hostName);

List<Device> deviceObjlist = query.list();

<---OR--->

String hql = "from " + className + " where deviceid = :deviceid";

Query query = session.createQuery(hql).setParameter("deviceid", deviceId + "");

List queryResult = query.list();

<---OR--->

String serviceselectQuery = "SELECT \* FROM OPERATIONTEMPLATES WHERE (SERVICE\_TYPE = ? AND ORDER\_TYPE = ? AND DEVICE\_TYPE = ? AND DEVICE\_VERSION =? )";

session.createSQLQuery(serviceselectQuery).addEntity(OperationTemplates.class)

.setParameter(0, serviceType).

setParameter(1, orderType)

.setParameter(2, deviceContext.getDevicetype()).

setParameter(3, deviceContext.getOsversion())

List queryResult = query.list()

-------------------------------------------------------------------------------------------------------

7.>What are the key components/objects of hibernate?

Following are the key components/objects of Hibernate −

**Configuration** − Represents a configuration or properties file required by the Hibernate.

**SessionFactory** − Configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated.

**Session** − Used to get a physical connection with a database.

**Transaction** − Represents a unit of work with the database and most of the RDBMS supports transaction functionality.

**Query** − Uses SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.

**Criteria** − Used to create and execute object oriented criteria queries to retrieve objects.

-------------------------------------------------------------------------------------------------------------------

8.>What are the two key components of a hibernate configuration object?

The Configuration object provides two keys components −

**Database Connection** − This is handled through one or more configuration files supported by Hibernate.

These files are hibernate.properties and hibernate.cfg.xml.

**Class Mapping Setup** -This component creates the connection between the Java classes and database tables.

example:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

**<hibernate-configuration>**

**<session-factory>**

<property name="hibernate.connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>

<property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</property>

<property name="hibernate.connection.username">gsngae</property>

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

<property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>

<property name="org.hibernate.envers.track\_entities\_changed\_in\_revision">true</property>

<property name="hibernate.hbm2ddl.auto">create</property>

<property name="connection.pool\_size">5</property>

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

<!-- Mapping class -->

<mapping class="com.java.sanjeev.Transaction" />

**</session-factory>**

**</hibernate-configuration>**

--------------------------------------------------------------------------------------------------------------------

9.>What is a configuration object in hibernate?

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization.

It represents a configuration or properties file required by the Hibernate.

Configuration cfg =new Configuration().configure();

sessionFactory sessionFactory = cfg.buildSessionFactory();

Session session = sessionFactory.openSession();

session.beginTransaction();

session.save(OBJECT);

session.getTransaction().commit();

-----

OR

-----

Session session = HibernateUtil.getSession();

Transaction tx = session.getTransaction();

tx.begin();

session.save(OBJECT);

tx.commit();

CURD

session.update(employee);

session.delete(employee);

----------------------------------------------------------------------------------------------------------------------

10.>What is a SessionFactory in hibernate?

**SessionFactory** − Configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated.

The SessionFactory is a thread safe object and used by all the threads of an application.

The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use.

**You would need one SessionFactory object per database using a separate configuration file.**

So if you are using multiple databases then you would have to create multiple SessionFactory objects.

-----------------------------------------------------------------------------------------------------------------------

11.>What is Session in hibernate?

A Session is used to get a physical connection with a database.

The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database.

Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

=====================================================================================

11.1>What is the purpose of Session.beginTransaction() method?

**Session.beginTransaction()** method begins a unit of work and returns the associated Transaction object.

=====================================================================================12.>What is Transaction in hibernate?

A **Transaction** represents a unit of work with the database and most of the RDBMS supports transaction functionality.

Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

-------------------------------------------------------------------------------------------------------------------------------

13.>What is Query in hibernate?

Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

/\*

query.list()

query.executeUpdate()

\*/

example-1:

Query query=session.createQuery("from Emp");//here persistent class name is Emp

List list=query.list();

example-2: update query

Transaction tx=session.beginTransaction();

Query q=session.createQuery("update User set name=:n where id=:i");

q.setParameter("n","Udit Kumar");

q.setParameter("i",111);

int status=q.executeUpdate();

System.out.println(status);

tx.commit();

example-3: HQL delete query

Query query=session.createQuery("delete from Emp where id=100");

//specifying class name (Emp) not tablename

query.executeUpdate();

--------------------------------------------------------------------------------------------------------------------------------------

14.>What is Criteria in hibernate?

Criteria object are used to create and execute object oriented criteria queries to retrieve objects.

Example of HCQL to get the 10th to 20th record

Crietria c=session.createCriteria(Emp.class);

c.setFirstResult(10);

c.setMaxResult(20);

List list=c.list();

Example of HCQL to get the records whose salary is greater than 10000

Crietria crietria=session.createCriteria(Emp.class);

crietria.add(Restrictions.gt("salary",10000));//salary is the propertyname

List list=c.list();

Example of HCQL to get the records in ascending order on the basis of salary

Example:

Crietria c=session.createCriteria(Emp.class);

c.addOrder(Order.asc("salary"));

List list=c.list();

Example:

persistenceService.beginTransaction();

DetachedCriteria criteria = DetachedCriteria.*forClass*(OpticalOTDRScanResult.**class**);

OpticalOTDREquipment otdrEquipment = OpticalOTDRRetriever.*getOTDREquipment*(otdrID, persistenceService);

criteria.add(Restrictions.*eq*(OpticalOTDRScanResult.***OPTICAL\_OTDREQUIPMENT***,otdrEquipment));

criteria.add(Restrictions.*eq*(OpticalOTDRScanResult.***SCAN\_TYPE***,scanTypeEnum));

criteria.add(Restrictions.*eq*(OpticalOTDRScanResult.***DIRECTION***,directionEnum));

criteria.add(Restrictions.*in*(OpticalOTDRScanResult.***STATUS***,listStatusEnum));

criteria.add(Restrictions.*ne*(OpticalOTDRScanResult.***TIMESTAMP***,OTDRConstants.***DUMMY\_TS***));

criteria.add(Restrictions.*eq*(OpticalOTDRScanResult.***IMPORT\_TYPE***,importType));

resultList = persistenceService.getInstances(criteria);

------------------------------------------------------------------------------------------------------------------------------------------

15.>Name some of the properties you would require to configure for a databases in a standalone situation.

Sr.No. Properties & Description

1.> hibernate.dialect - This property makes Hibernate generate the appropriate SQL for the chosen database.

2.> hibernate.connection.driver\_class - The JDBC driver class.

3.>hibernate.connection.url - The JDBC URL to the database instance.

4.>hibernate.connection.username - The database username.

5.>hibernate.connection.password - The database password.

6.>hibernate.connection.pool\_size - Limits the number of connections waiting in the Hibernate database connection pool.

7.>hibernate.connection.autocommit - Allows autocommit mode to be used for the JDBC connection.

8.> hibernate.hbm2ddl.auto ---it can have 4 values --create,update,create-drop,validate

create-->will drop existing schema along with dropping previous record and create new one and loads new data.

create-drop-->everything will be same like create but in this if we will close sessionFactory object explicitly then it will drop the tablefrom DB.

update-->every time we run our application hibernate will just update the schema.For example,if we change column name then it will update.But our data will be safe.

It should not be used in production because it can give weired result.

validate-->it will validate the existing database schema with the one which can be generated using annotated model classes.And would not update or make any changes to the database.

If there is no difference then it will execute normally and if there is some difference then throw exception at runtime.

------------------------------------------------------------------------------------------------------------------------------------

16.>Which method is used to create a HQL query?

Query query=Session.createQuery creates a new instance of Query for the given HQL query string.

-----------------------------------------------------------------------------------------------------------------------------------------

17.>Which method is used to create a SQL query?

Session.createSQLQuery creates a new instance of SQLQuery for the given SQL query string.

example:

String sql = "SELECT first\_name, salary FROM EMPLOYEE";

SQLQuery query = session.createSQLQuery(sql);

------------------------------------------------------------------------------------------------------------------------------------------

18.>Which method is used to remove a persistent instance from the datastore?

Session.delete removes a persistent instance from the datastore.

------------------------------------------------------------------------------------------------------------------------------------------

19.>Which method is used to save the state of the given instance from the underlying database?

>>Session.save saves the state of the given instance from the underlying database.

Which method is used to update the state of the given instance from the underlying database?

>>Session.update updates the state of the given instance from the underlying database.

Which method is used to save or update the state of the given instance from the underlying database?

>>Session.saveOrUpdate either saves(Object) or updates(Object) the given instance.

------------------------------------------------------------------------------------------------------------------------------------------------

20.>What are the three states of a persistent entity at a given point in time?

Instances may exist in one of the following three states at a given point in time :::

transient − A new instance of a persistent class which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.

persistent − You can make a transient instance persistent by associating it with a Session. A persistent instance has a representation in the database, an identifier value and is associated with a Session.

detached − Once we close the Hibernate Session, the persistent instance will become a detached instance.

--------------------------------------------------------------------------------------------------------------------------------------

21.>What are persistent classes in hibernate?

Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate.

============================================================

\*\*\*\*\*Transient, Persistent and Detached Objects\*\*\*\*\*

Transient:Before an object handed to hibernate ,it is transient

Persistent: Once we pass object to session.save(),after that any change we make to the object will go as an update statement to database before we commit the transaction.

Detached:after session.close() ,it will become detached object.

Object States in Hibernate – Transient,Persistent and Detached:

1. Transient Object State:

An object which is not associated with hibernate session and does not represent a row in the database is considered as transient. It will be garbage collected if no other object refers to it.

An object that is created for the first time using the new() operator is in transient state.

When the object is in transient sate then it will not contain any identifier (primary key value).

You have to use save, persist or saveOrUpdate methods to persist the transient object.

Employee emp = new Employee();

emp.setName("Ravi Raj");

// here emp object is in a transient state

2.Persistent Object State:

An object that is associated with the hibernate session is called as Persistent object. When the object is in persistent state, then it represent one row of the database and consists of an identifier value.

You can make a transient instance persistent by associating it with a Session.

Long id = (Long) session.save(emp);

// emp object is now in a persistent state

3. Detached Object State:

Object which is just removed from hibernate session is called as detached object.The state of the detached object is called as detached state.

Any changes made to the detached objects are not saved to the database. The detached object can be reattached to the new session and save to the database using update, saveOrUpdate and merge methods.

session.close();

//object in detached state

--------------------------------------------------------------------------------------------------

//object is in transient state

UserDetails userdetails=new UserDetails();

userdetails.setUserName("SANJEEV");

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

////object is in persistent state

session.save(userdetails);

userdetails.setUserName("changed use name");

userdetails.setUserName("changed use name again");

session.getTransaction().commit();

session.close();

//object is in detatched state

userdetails.setUserName("changed use name after detached");

--------------------------------------------------------------------------------------------------

output:

Hibernate: select hibernate\_sequence.nextval from dual

Hibernate: insert into User\_Details (userName, userId) values (?, ?)

Hibernate: update User\_Details set userName=? where userId=?

UserId Username

1 changed use name again

------------------------------------------------------------------------------------------------------

<---Understanding state changes---->

when we do session.get(UserDetails.class,7) ,hibernate provides us persistent object

so basically we can update our object and will persist in the db.

for delete:

transient <------------------ Persistent -----------------> Detached

session.delete() session.close()

------------------------------------------------------------------------------------------------------------

--->Table Creation Steps and most common annotations

Note(@Table annotation is only used if we want to give name other than Class Name)

@Entity

@Table(name="Student\_Information")

public class Student\_Info**{**

@Id

private int rollNo;

@Transient //if we want a column to ignore while persisting into DB then we can use transient.

@Column(name="Full\_Name", nullable=false) //This is used if we want to give column name other than default. and nullable=false if don't want it to be null.

private String name;

//getters and Setters

**}**

similarly we can use @Temporal(TemporalType.DATE) for storing the date with timestamp or without timestamp.

Primary Key Auto generation::If we just want an serial number not the business details in it then we can go with it.

@GeneratedValue or @GeneratedValue(strategy=GenerationType.SEQUENCE)

-->

public static void main(String args[]){

Student\_Info student\_Info=new Student\_Info();

student\_Info.setRolNo(1);

Configuration cfg =new Configuration().configure();

sessionFactory sessionFactory = cfg.buildSessionFactory();

Session session = sessionFactory.openSession();

session.beginTransaction();

session.save(student\_Info);

session.getTransaction().commit();

session.close();

sessionFactory.close();

}

=========================================================================

One to One Mapping::

If each row in Table A is linked to each row in Table B

No. of rows in Table A= No. of rows in Table B

Student Table StudentDetail Table

Student\_Id Student\_Name Student\_Id Student\_Mobile\_no

1 Akash 1 444444444444

2 Ramesh 2 555555555555

here Student\_Id in StudentDetail Table is having a foreign key reference

with student\_id in student table

public class Student

{

@Id @GeneratedValue

private int student\_id;

}

public class StudentDetail

{

@Id @GeneratedValue(generator="newGenerator") //name of the primary key generator

@GenericGenerator(name="newGenerator", strategy="foreign",parameters={@Parameter(value="student",name="property")})

private int student\_id;

@OneToOne(cascade=CascadeType.ALL)

@JoinColumn(name="student\_id")

private Student student;

}

MAIN CLASS--->

Student student=new Student();

student.setName("HK");

StudentDetail studentDetail=new StudentDetail();

studentDetail.setStudent\_mobile\_number("555555");

//IMP step

studentDetail.setStudent(student);

//Now we are saving only studentDetail object but hibernate will save student object as well.

//So this is magic of one to one mapping

session.save(studentDetail);

=========================================================================

Association Mappings with JPA and Hibernate

@OneToMany(cascade=CascadeType.ALL)

@JoinColumn(name="EMPLOYEE\_ID")

https://howtodoinjava.com/hibernate/hibernate-one-to-many-mapping-using-annotations/

============================

-----------------------------------------------------------------------------------

-->@GeneratedValue(strategy=GenerationType.)

1-->**AUTO** - is the default generation type and lets the persistence provider choose the generation strategy. it selects a generation strategy based on the database specific dialect. For most popular databases, it selects GenerationType.SEQUENCE

2-->**IDENTITY** -

3-->**SEQUENCE**--basically hibernate is using sequence object for generating PK

4-->**TABLE**

------------------------------------------------------------------------------------

// @Embeddable --tells hibernate not to create separate table for this class.

//this type of class is being used when we have user\_details table and in that if we want to save address (which internally having details)

//so to handle that in UserDetails class will be cumbersome.

//Good thing is that all the address column will come as part of User\_Details table only because we are using @Embeddable annotations.

//Value Object: objects which doesn't have its meaning at their own.

-->Example

@Embeddable

public class Address{

@Column(name="CITY\_NAME")

private String city;

@Column(name="STATE\_NAME")

private String state;

@Column(name="STREET\_NAME")

private String street;

//getters and setters

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

public String getStreet() {

return street;

}

public void setStreet(String street) {

this.street = street;

}

}

---------------------------------------------------------------------

@Entity

@Table (name="User\_Details")

public class UserDetails

{

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int userId;

private String userName;

@Embedded

private Address address;

//getters and setters

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public Address getAddress() {

return address;

}

public void setAddress(Address address) {

this.address = address;

}

}

----------------------------------------------------------------------

IMP--lets if we have two address in our UserDetails class.then we can override our attributes as well so that we can give different name

example:::

@Embedded

@AttributeOverrides(

{

@AttributeOverride(name="street" ,column=@Column(name="Home\_Street\_Name")),

@AttributeOverride(name="city" ,column=@Column(name="Home\_CITY\_Name")),

@AttributeOverride(name="state" ,column=@Column(name="Home\_State\_Name"))

}

)

private Address homeAddress;

@Embedded

private Addtess officeAddress;

------------------------------------------------------------------------

Note::- If we want to make Embedded as PK the we can do as -----@EmbeddedId

------------------------------------------------------------------------

How to Savings Collections---- Below example is case of value object inside entity object.

In case of saving collection,hibernate will create a subtable to store those collection elements

Primary key for that table will be from main entity.

--------------------------------------------------------------------------

@Embeddable

public class Address {

@Column(name="STREET\_NAME")

private String street;

@Column(name="CITY\_NAME")

private String city;

@Column(name="STATE\_NAME")

private String state;

@Column(name="PIN\_CODE")

private String pincode;

public String getStreet() {

return street;

}

public void setStreet(String street) {

this.street = street;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

public String getPincode() {

return pincode;

}

public void setPincode(String pincode) {

this.pincode = pincode;

}

}

-------------------------------------------------------------------

@Entity

@Table(name="User\_Details")

public class UserDetails {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int userId;

private String userName;

//using this ,hibernate will create sub table with User\_Details\_listofAddress

it will have column as userId (Foreign Key),STREET\_NAME,CITY\_NAME,STATE\_NAME,PIN\_CODE.

@ElementCollection

private Set<Address> listofAddress=new HashSet<Address>();

public int getUserId() {

return userId;}

public void setUserId(int userId) {

this.userId = userId;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public Set<Address> getListofAddress(){

return listofAddress;

}

public void setListofAddress(Set<Address> listofAddress){

this.listofAddress=listofAddress;

}

}

---------------------------------------------------------------------

public class HibernateTest {

public static void main(String[] args) throws Exception{

UserDetails userdetails=new UserDetails();

userdetails.setUserName("SANJEEV");

Address add1=new Address();

add1.setCity("BNGLORE");

add1.setState("KARNATAKA");

Address add2=new Address();

add2.setCity("PUNE");

add2.setState("Maharastra");

userdetails.getListofAddress().add(add1);

userdetails.getListofAddress().add(add2);

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

session.save(userdetails);

session.getTransaction().commit();

session.close();

}

}

-----------------------------------------------------------------------------------------------

result:it will create 2 tables: User\_Details,UserDetails\_listofAddress with userId as F.K.

-------------------------------------

we can provide the name of the sub table using annotation as

@ElementCollection

@JoinTable(name="User\_Address")

private Set<Address> listofAddress=new HashSet<Address>();

-----------------------------------------------------------------------------------------------

Note we can change Set<Address> to -->Collection<Address>

@CollectionId

private Collection<Address> listofAddress=new HashSet<Address>();

-----------------------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*Proxy Objects and Eager and Lazy Fetch Types\*\*\*\*\*\*\*\*\*\*\*

-->Lazy Initialization: When a user has many addresses it is not efficient to load all of its addresses with it when they are not needed.

So in suchlike cases, we can declare that we want addresses to be loaded when they are actually needed.

This is called lazy loading.

**FetchType LAZY - fetch when needed**.

NOTE:LAZY is default implementation.

In that hibernate will do a another select query in db get the list of address when getLisOfAddresses() is being called using session() object.

-->Eager Initialization: The purpose of Join Fetch strategy is optimization in terms of time.

So in this case we don’t make database call again and again. So this will be much faster. Agreed that this will bad if we are fetching too many objects in a session because we can get java heap error.

**FetchType EAGER - fetch immediately**.

in Lazy Initialization we do not initialize entire object, we only initialize the first level variables (i.e. only first level of member variable of the object) . Then we only initialize the list only when we actually access it.

**Note:\*Hibernate provides proxy user class at back end to fetch the data from database.**

---->syntax for declaring fetch type

@Entity

@Table(name="User\_Details")

public class UserDetails {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int userId;

private String userName;

@ElementCollection(fetch=FetchType.EAGER)

//@ElementCollection(fetch=FetchType.LAZY)

@JoinTable(name="User\_Address" , joinColumns=@JoinColumn(name="USER\_ID"))

private Collection<Address> listofAddress=new ArrayList<Address>();

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public Collection<Address> getListofAddress() {

return listofAddress;

}

public void setListofAddress(Collection<Address> listofAddress) {

this.listofAddress = listofAddress;

}

}

---------------------------------------------------------------------------------------------

package com.javahibernate;

import javax.persistence.Column;

import javax.persistence.Embeddable;

@Embeddable

public class Address {

@Column(name="STREET\_NAME")

private String street;

@Column(name="CITY\_NAME")

private String city;

@Column(name="STATE\_NAME")

private String state;

@Column(name="PIN\_CODE")

private String pincode;

public String getStreet() { return street;}

public void setStreet(String street) {this.street = street;}

public String getCity() {return city;}

public void setCity(String city) {this.city = city;}

public String getState() {return state;}

public void setState(String state) {this.state = state;}

public String getPincode() {return pincode;}

public void setPincode(String pincode) {this.pincode = pincode;}

}

---------------------------------------------------------------------------------------------

public class HibernateTest {

public static void main(String[] args) throws Exception

{

UserDetails userdetails=new UserDetails();

userdetails.setUserName("SANJEEV");

Address add1=new Address();

add1.setCity("BNGLORE");

add1.setState("KARNATAKA");

Address add2=new Address();

add2.setCity("PUNE");

add2.setState("Maharastra");

userdetails.getListofAddress().add(add1);

userdetails.getListofAddress().add(add2);

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

session.save(userdetails);

session.getTransaction().commit();

session.close();

userdetails=null;

session=sessionFactory.openSession();

userdetails=(UserDetails) session.get(UserDetails.class,1);

session.close();

//Note :it will not return any error because ListofAddress is already loaded because of EAGER initialization.

//in case of LAZY -if we close session before getting ListofAddress ,it will throw error because in lazy,hibernate works on proxy object.

System.out.println(userdetails.getListofAddress().size());

}

}

-------------------------------------------------------------------------------------------------

Entity object inside another entity object

\*\*\*\*\*\*\*\*\*ONE TO ONE mapping\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

in this for each record in one table let’s say User\_Details ,there will be exactly one record in the Vehicle table.That is called One-to-One mapping

in below example by default name of FK will be vehicle\_vehicleId.

Note: If we want to change the name , then we have to use @JoinColumn(name="some name of column") annotation. (it is the optional)

//@OneToOne

//@JoinColumn(name="VEHICLE\_ID")

//private Vehicle vehicle;

-------------------------------------

@Entity

@Table(name="User\_Details")

public class UserDetails {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int userId;

private String userName;

@OneToOne

private Vehicle vehicle;

public int getUserId() {return userId;}

public void setUserId(int userId) {this.userId = userId;}

public String getUserName() {return userName;}

public void setUserName(String userName) {this.userName = userName;}

public Vehicle getVehicle() {return vehicle;}

public void setVehicle(Vehicle vehicle) {this.vehicle = vehicle;}

}

---------------------------------------------

@Entity

public class Vehicle {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int vehicleId;

private String vehicleName;

public int getVehicleId() {return vehicleId;}

public void setVehicleId(int vehicleId) {this.vehicleId = vehicleId;}

public String getVehicleName() {return vehicleName;}

public void setVehicleName(String vehicleName) {this.vehicleName = vehicleName;}

}

------------------------------------------------

public class HibernateTest {

public static void main(String[] args) throws Exception{

UserDetails userdetails=new UserDetails();

userdetails.setUserName("SANJEEV");

Vehicle vehicle=new Vehicle();

vehicle.setVehicleName("HONDA CITY");

userdetails.setVehicle(vehicle);

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

session.save(userdetails);

session.save(vehicle);

session.getTransaction().commit();

session.close();

}

}

------------------------------------------------------------------

output:

Hibernate: insert into User\_Details (userName, vehicle\_vehicleId, userId) values (?, ?, ?)

Hibernate: insert into Vehicle (vehicleName, DTYPE, vehicleId) values (?, 'Vehicle', ?)

Hibernate: update User\_Details set userName=?, vehicle\_vehicleId=? where userId=?

It will add vehicle id in User\_Details table as F.K.

------------------------------------------------------------------

\*\*\*\*\*\*\***OneToMany along with bi-directional relationship** \*\*\*\*\*\*\*\*\*\*

The term “bidirectional” literally means “functioning in two directions”, which is the concept that we will apply in our relationships between two Java objects.

When we have a bidirectional relationship between objects,

it means that we are able to access Object A from Object B, and Object B from Object A.

Bidirectional Relationships:

In a bidirectional relationship, each entity has a relationship field or property that refers to the other entity.

Through the relationship field or property, an entity class’s code can access its related object.

If an entity has a related field, the entity is said to “know” about its related object. For example,

if User knows what Vehicle instances it has and if Vehicle knows what User it belongs to, they have a bidirectional relationship.

Bidirectional relationships must follow these rules:

1.>The inverse side of a bidirectional relationship must refer to its owning side by using the mappedBy element of the::

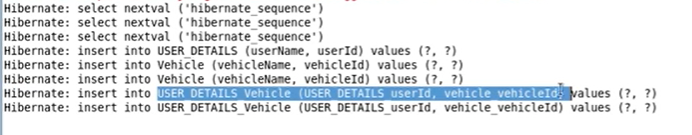
@OneToOne, @OneToMany, or @ManyToMany annotation. The mappedBy element designates the property or field in the entity that is the owner of the relationship.

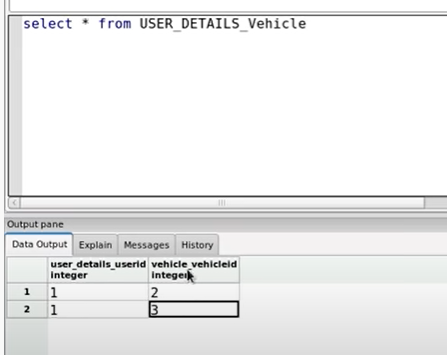
2.>The many side of many-to-one bidirectional relationships must not define the mappedBy element. The many side is always the owning side of the relationship.

3.>For one-to-one bidirectional relationships, the owning side corresponds to the side that contains the corresponding foreign key.

For many-to-many bidirectional relationships, either side may be the owning side.

**IMP: whenever we are saving collection, hibernate will create a separate table. In below example ,hibernate will create a separate table USER\_VEHICLE which is having 2 columns USER\_ID and VEHICLE\_ID.**



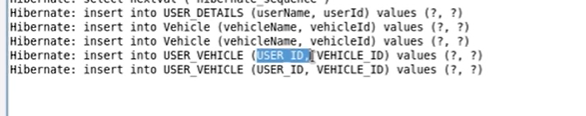


**NOTE: @JoinTable(name="USER\_VEHICLE",joinColumns=@JoinColumn(name="USER\_ID"),inverseJoinColumns=@JoinColumn(name="VEHICLE\_ID"))**

**->above syntax is being used for configuring names which we want in the new table. ()**

**->If we will not use the above command, Hibernate will create table with some default name. Same applicable for columns as well. It is just used to override default names.**

**Please refer below screenshots:**



@Entity

@Table(name="User\_Details")

public class UserDetails {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int userId;

private String userName;

@OneToMany

**@JoinTable(name="USER\_VEHICLE",joinColumns=@JoinColumn(name="USER\_ID"),**

**inverseJoinColumns=@JoinColumn(name="VEHICLE\_ID"))**

private Collection<Vehicle> vehicleList=new ArrayList<Vehicle>();

public int getUserId() {return userId;}

public void setUserId(int userId) {this.userId = userId;}

public String getUserName() {return userName;}

public void setUserName(String userName) {this.userName = userName;}

public Collection<Vehicle> getVehicleList() {return vehicleList;}

public void setVehicleList(Collection<Vehicle> vehicleList) {this.vehicleList = vehicleList;}

}

-----------------------------------------------------------------

@Entity

public class Vehicle {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int vehicleId;

private String vehicleName;

/\* this is a reverse relation ship.or Bi-directional relationship .It is optional

**Now we can get vehicle from user and user from vehicle.in Vehicle class** \*/

@ManyToOne

private UserDetails user\_Details;

public int getVehicleId() {return vehicleId;}

public void setVehicleId(int vehicleId) {this.vehicleId = vehicleId;}

public String getVehicleName() {return vehicleName;}

public void setVehicleName(String vehicleName) {this.vehicleName = vehicleName;}

public UserDetails getUser\_Details() {return user\_Details;}

public void setUser\_Details(UserDetails user\_Details) { this.user\_Details = user\_Details;}

}

---------------------------------------------------------------

public class HibernateTest {

public static void main(String[] args) throws Exception{

UserDetails userdetails=new UserDetails();

userdetails.setUserName("SANJEEV");

Vehicle vehicle1=new Vehicle();

vehicle1.setVehicleName("HONDA CITY");

Vehicle vehicle2=new Vehicle();

vehicle2.setVehicleName("Baleno");

userdetails.getVehicleList().add(vehicle1);

userdetails.getVehicleList().add(vehicle2);

vehicle1.setUser\_Details(userdetails);

vehicle2.setUser\_Details(userdetails);

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

session.save(userdetails);

session.save(vehicle1);

session.save(vehicle2);

session.getTransaction().commit();

session.close();

}

}

This example will create 3 tables.

User\_Details - userName,userId

Vehicle - vehicleName , vehicleId

User\_Vehicle - USER\_ID , VEHICLE\_ID

* There is 1 other way to represent **one to many OR many to one.**

That is object on the many side of the relationship has a reference on the other side.

Towards one to many side , User can’t have column for vehicle because 1 user can have multiple vehicle **but** vehicle table can have column for userid because any vehicle can have single user.

So we can have mapping in vehicle table for user id without having separate table for mapping between userId and vehicleId.

**in UserDetails class**

@Entity

@Table(name="User\_Details")

public class UserDetails {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int userId;

private String userName;

/\* Here mapped by signifies where we want the mapping. So mapping will be done in the class where user\_Details is present which means mapping is present on the other side of the relationship. mappedBy will make sure that separate table should not get create for mapping between userId and vehicleId\*/

@OneToMany(mappedBy=”user\_Details”)

private Collection<Vehicle> vehicleList=new ArrayList<Vehicle>();

**in Vehicle class**

**@Entity**

public class Vehicle {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int vehicleId;

private String vehicleName;

/\* \*/

@ManyToOne

@JoinColumn(name=”USER\_ID”)

private UserDetails user\_Details;

public int getVehicleId() {return vehicleId;}

public void setVehicleId(int vehicleId) {this.vehicleId = vehicleId;}

public String getVehicleName() {return vehicleName;}

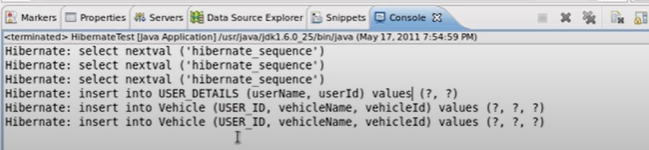
public void setVehicleName(String vehicleName) {this.vehicleName = vehicleName;}

public UserDetails getUser\_Details() {return user\_Details;}

public void setUser\_Details(UserDetails user\_Details) { this.user\_Details = user\_Details;}

}

**In below output we can see that , hibernate is not creating separate table for mapping between userId and vehicleId. Instead doing in vehicle table itself.**



**in HibernateTest class**

public class HibernateTest {

public static void main(String[] args) throws Exception{

UserDetails user=new UserDetails();

user.setUserName("SANJEEV");

Vehicle vehicle1=new Vehicle();

vehicle1.setVehicleName("HONDA CITY");

Vehicle vehicle2=new Vehicle();

vehicle2.setVehicleName("Baleno");

user.getVehicleList().add(vehicle1);

user.getVehicleList().add(vehicle2);

vehicle1.setUser(user);

vehicle2.setUser(user);

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

session.save(user);

session.save(vehicle1);

session.save(vehicle2);

session.getTransaction().commit();

session.close();

}

}

=====================================================================================

//mappedBy informing hibernate that where we want the mapping. so we defined mappedBy="user".So it means we want mapping where user object is present.

So hibernate will look on the other side where user object is present.so instead of mapping vehicleId and userId in separate table.it will map them in vehicle table.

//mappedBy helps us to have to make sure that no separate table get create for mapping.

//MappedBy signals hibernate that the key for the relationship is on the other side. This means that although you link 2 tables together,

-----------------------------------------------------------------------------------------------------========

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*One more Example\*\*\*\*\*\*\*\*\*\*\*\*

@Entity

@Table(name = "UNIVERSITY")

public class University {

@Id

@GeneratedValue

@Column(name = "UNIVERSITY\_ID")

private long id;

@Column(name = "NAME")

private String name;

@Column(name = "COUNTRY")

private String country;

//mappedBy informing hibernate that where we want the mapping. so we defined mappedBy="university".So it means we want mapping where university object is present.

//so hibernate will look on the other side where university object is present.

@OneToMany(mappedBy = "university", cascade = CascadeType.ALL)

private List<Student> students;

public University() {}

public University(String name, String country) {

this.name = name;

this.country = country;

}

public long getId() {return id;}

public void setId(long id) { this.id = id;}

public String getName() { return name; }

public void setName(String name) { this.name = name;}

public String getCountry() {return country;}

public void setCountry(String country) {this.country = country;}

public List<Student> getStudents() {return students; }

public void setStudents(List<Student> students) { this.students = students;}

@Override

public String toString() {return "University [id=" + id + ", name=" + name + ", country="+ country + "]"}

}

@Entity

@Table(name = "STUDENT")

public class Student {

@Id

@GeneratedValue

@Column(name = "STUDENT\_ID")

private long id;

@Column(name = "FIRST\_NAME")

private String firstName;

@Column(name = "LAST\_NAME")

private String lastName;

@Column(name = "SECTION")

private String section;

@ManyToOne(optional = false)

@JoinColumn(name = "UNIVERSITY\_ID")

private University university;

public Student() {

}

public Student(String firstName, String lastName, String section) {

this.firstName = firstName;

this.lastName = lastName;

this.section = section;

}

public long getId() { return id;}

public void setId(long id) {this.id = id; }

public String getFirstName() {return firstName; }

public void setFirstName(String firstName) {this.firstName = firstName; }

public String getLastName() {return lastName; }

public void setLastName(String lastName) {lastName = lastName; }

public String getSection() {return section; }

public void setSection(String section) {this.section = section; }

public University getUniversity() {return university; }

public void setUniversity(University university) {this.university = university; }

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + (int) (id ^ (id >>> 32));

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (!(obj instanceof Student))

return false;

Student other = (Student) obj;

if (id != other.id)

return false;

return true;

}

@Override

public String toString() {

return "Student [id=" + id + ", firstName=" + firstName + ", lastName="

+ lastName + ", section=" + section + "]";

}

}

-----------------------------------------------------------------------------------------------------

Difference between Unidirectional and bi-directional

This is a Unidirectional association

public class User {

private int id;

private String name;

@ManyToOne

@JoinColumn(

name = "groupId")

private Group group;

}

public class Group {

private int id;

private String name;

}

The Bidirectional association

public class User {

private int id;

private String name;

@ManyToOne

@JoinColumn(

name = "groupId")

private Group group;

}

public class Group {

private int id;

private String name;

@OneToMany(mappedBy="group")

private List<User> users;

}

------------------------------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*Bi-Directional relationship using\*\*\*\*\*\*\*Many To Many Mapping\*\*\*\*\*\*\*\*\*\*\*\*\*

->There is only 1 way to represent many to many relationships. There has to be a separate mapping table.

Assume a scenario,rental vehicle where user can rent many vehicle and vehicle can be rented my multiple user.

In that case 1 user might be having multiple vehicles allocated and vehicle can be allocated to multiple user.

So we can't having in either user table or vehicle table. It has to be a separate table.

------------------------------------------------------------------------------------------------------

@Entity

@Table(name="User\_Details")

public class UserDetails {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int userId;

private String userName;

@ManyToMany

@JoinTable(name="USER\_VEHICLE",joinColumns=@JoinColumn(name="USER\_ID"),

inverseJoinColumns=@JoinColumn(name="VEHICLE\_ID"))

/\*Collection of vehicles in UserDetails

vechicle collection inside user table

\*/

private Collection<Vehicle> vehicleList=new ArrayList<Vehicle>();

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public Collection<Vehicle> getVehicleList() {

return vehicleList;

}

public void setVehicleList(Collection<Vehicle> vehicleList) {

this.vehicleList = vehicleList;

}

}

--------------------------------------------------------------------------------------------

@Entity

public class Vehicle {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int vehicleId;

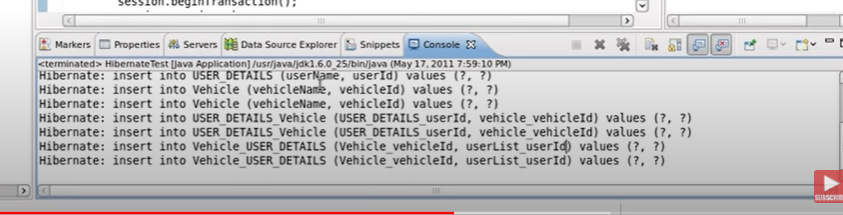
private String vehicleName;

/\* this is a reverse relation ship.or Bi-directional relationship \*/

/\*Collection of UserDetails in Vehicle table \*/

@ManyToMany(mappedBy="vehicleList")

**/\* Note : If we will not use mappedBy .Hibernate will create 2 mapping table.**



**So to inform hibernate not to create 2 mapping table.we have to use mappedBy .**

**\*/**

private Collection<UserDetails> user\_DetailsList=new ArrayList<UserDetails>();

public int getVehicleId() {

return vehicleId;

}

public void setVehicleId(int vehicleId) {

this.vehicleId = vehicleId;

}

public String getVehicleName() {

return vehicleName;

}

public void setVehicleName(String vehicleName) {

this.vehicleName = vehicleName;

}

public Collection<UserDetails> getUser\_DetailsList() {

return user\_DetailsList;

}

public void setUser\_DetailsList(Collection<UserDetails> user\_DetailsList) {

this.user\_DetailsList = user\_DetailsList;

}

}

------------------------------------------------------------------------------------

public class HibernateTest {

public static void main(String[] args) throws Exception{

System.out.println("helllloooo");

UserDetails userdetails=new UserDetails();

userdetails.setUserName("SANJEEV");

Vehicle vehicle1=new Vehicle();

vehicle1.setVehicleName("HONDA CITY");

Vehicle vehicle2=new Vehicle();

vehicle2.setVehicleName("Baleno");

userdetails.getVehicleList().add(vehicle1);

userdetails.getVehicleList().add(vehicle2);

vehicle1.getUser\_DetailsList().add(userdetails);

vehicle2.getUser\_DetailsList().add(userdetails);

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

session.save(userdetails);

session.save(vehicle1);

session.save(vehicle2);

session.getTransaction().commit();

session.close();

}

}

Note: If we will not use mapped by hibernate will create 4 tables:

USER\_DETAILS: userName, userId

Vehicle:vehicleName,vehicleId

USER\_DETAILS\_Vehicle:User\_Details\_userId,vehicle\_vehicleId

Vehicle\_User\_Details:vehicle\_vehicleId,userList\_userId

To tell hibernate only 1 table we have to user mappedBy. So pick 1 class and do mappedBy="vehicleList"

so it will help us to have mapping only in 1 table.

USER\_DETAILS: userName, userId

Vehicle:vehicleName,vehicleId

USER\_DETAILS\_Vehicle:user\_DetailsList\_userId,vehicleList\_vehicleId

-----------------------------------------------------------------------------------------

**\*\*\*\*\*\*CascadeTypes and Other Things\*\*\*\*\*\*\*\***

Hibernate Collections:

1.>Bag semantic-List /ArrayList

2.>Bag semantic with IDENTITY

3.>List semantic

4.>Set semantic

5.>Map semantic

When we have relationship between entities, then we need to define how the different operations will affect the other entity. This is done by cascading and there are different types of it.

Here is a simple example of applying cascading between primary and secondary entities.

cascade attribute::, which means the dependent object(Student) will be persisted/updated/deleted automatically on subsequent persist/update/delete on University object.

No need to perform operation separately on Student.

cascade means: CASCADE types

CascadeType.ALL

CascadeType.PERSIST

CascadeType.DETACH

CascadeType.REMOVE

CascadeType.MERGE

CascadeType.REFRESH

----------------------------

@Entity

@Table(name="User\_Details")

public class UserDetails {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int userId;

private String userName;

@OneToMany(cascade=CascadeType.ALL)

@JoinTable(name="USER\_VEHICLE",joinColumns=@JoinColumn(name="USER\_ID"),

inverseJoinColumns=@JoinColumn(name="VEHICLE\_ID")

)

/\*Collection of vehicles in UserDetails \*/

private Collection<Vehicle> vehicleList=new ArrayList<Vehicle>();

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public Collection<Vehicle> getVehicleList() {

return vehicleList;

}

public void setVehicleList(Collection<Vehicle> vehicleList) {

this.vehicleList = vehicleList;

}

}

-----------------------------------------------------------------------------------------------

@Entity

public class Vehicle {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int vehicleId;

private String vehicleName;

public int getVehicleId() {

return vehicleId;

}

public void setVehicleId(int vehicleId) {

this.vehicleId = vehicleId;

}

public String getVehicleName() {

return vehicleName;

}

public void setVehicleName(String vehicleName) {

this.vehicleName = vehicleName;

}

}

------------------------------------------------------------------------------------------------

public class HibernateTest {

public static void main(String[] args) throws Exception{

UserDetails userdetails=new UserDetails();

userdetails.setUserName("SANJEEV");

Vehicle vehicle1=new Vehicle();

vehicle1.setVehicleName("HONDA CITY");

Vehicle vehicle2=new Vehicle();

vehicle2.setVehicleName("Baleno");

userdetails.getVehicleList().add(vehicle1);

userdetails.getVehicleList().add(vehicle2);

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

session.save(userdetails);

session.getTransaction().commit();

session.close();

}

}

--------------------------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*Implementing Inheritance\*\*\*\*\*\*\*\*\*\*\*\*\*

in this example only 1 table will be there

----------------------------

@Entity

public class Vehicle {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int vehicleId;

private String vehicleName;

public int getVehicleId() {

return vehicleId;

}

public void setVehicleId(int vehicleId) {

this.vehicleId = vehicleId;

}

public String getVehicleName() {

return vehicleName;

}

public void setVehicleName(String vehicleName) {

this.vehicleName = vehicleName;

}

}

---------------------------------------------

@Entity

public class TwoWheelerVehicle extends Vehicle {

private String SteeringHandle;

public String getSteeringHandle() {

return SteeringHandle;

}

public void setSteeringHandle(String steeringHandle) {

SteeringHandle = steeringHandle;

}

}

---------------------------------------------

@Entity

public class FourWheelerVehicle extends Vehicle {

private String Steeringwheel;

public String getSteeringwheel() {

return Steeringwheel;

}

public void setSteeringwheel(String steeringwheel) {

Steeringwheel = steeringwheel;

}

}

--------------------------------------------

public class HibernateTest {

public static void main(String[] args) throws Exception{

System.out.println("helllloooo");

Vehicle vehicle=new Vehicle();

vehicle.setVehicleName("Car");

TwoWheelerVehicle twoWheelerVehicle=new TwoWheelerVehicle();

twoWheelerVehicle.setVehicleName("Bike");

twoWheelerVehicle.setSteeringHandle("Bike Steering Handle");

FourWheelerVehicle fourWheelerVehicle=new FourWheelerVehicle();

fourWheelerVehicle.setVehicleName("Jeep");

fourWheelerVehicle.setSteeringwheel("Jeep Steering wheel");

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

session.save(vehicle);

session.save(twoWheelerVehicle);

session.save(fourWheelerVehicle);

session.getTransaction().commit();

session.close();

}

}

-----------------------------------------------------------------------------------------------------------------------

output::

Hibernate: insert into Vehicle (vehicleName, DTYPE, vehicleId) values (?, 'Vehicle', ?)

Hibernate: insert into Vehicle (vehicleName, SteeringHandle, DTYPE, vehicleId) values (?, ?, 'TwoWheelerVehicle', ?)

Hibernate: insert into Vehicle (vehicleName, Steeringwheel, DTYPE, vehicleId) values (?, ?, 'FourWheelerVehicle', ?)

here DTYPE is additional column ,it represents whether it is Viechle or TwoWheelerVehicle,FourWheelerVehicle

DTYPE VEHICLEID VEHICLENAME SteeringWheel SteeringHandle

Vehicle 1 Car null null

TwoWheelerVehicle 2 Bike null Bike Steering Handle

FourWheelerVehicle 3 Jeep Jeep Steering wheel null

Note:tBy default hibernate handles inheritance by @Inheritance(strategy=InheritanceType.SINGLE\_TABLE)

-------------------------------------------------------------------------------------------------------------------------

other types::

@Inheritance(strategy=InheritanceType.TABLE\_PER\_CLASS) --->

@Inheritance(strategy=InheritanceType.JOINED

)

--------------

here instead of DTYPE we will get Vehicle\_Type

@Entity

@Inheritance(strategy=InheritanceType.SINGLE\_TABLE)

@DiscriminatorColumn(

name="Vehicle\_Type",

discriminatorType=DiscriminatorType.STRING

)

--------------------------------------------------------------------------------------------------------------------------

\*\*\*\*\*\*\* CURD operations \*\*\*\*\*\*\*\*\*\*\*\*

C-Create

U-Update

R-Read

D-Delete

--------------------------------------------------------------------------------------------------------------------------

@Entity

@Table(name="User\_Details")

public class UserDetails {

@Id @GeneratedValue(strategy=GenerationType.AUTO)

private int userId;

private String userName;

public int getUserId() {

return userId;

}

public void setUserId(int userId) {

this.userId = userId;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

}

--------------------------------------------------------------------------------

public class HibernateTest {

public static void main(String[] args) throws Exception{

SessionFactory sessionFactory=new Configuration().configure().buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

UserDetails userdetails=null;

//inseting rows

for(int i=1;i<=10;i++){

userdetails=new UserDetails();

userdetails.setUserName("user :"+i);

session.save(userdetails);

}

session.getTransaction().commit();

session.close();

//Read opearion

session=sessionFactory.openSession();

userdetails=(UserDetails) session.get(UserDetails.class,6); //6th row

System.out.println("record pulled up is::"+userdetails.getUserId() +" : "+userdetails.getUserName());

//Delete operation

session.beginTransaction();

session.delete(userdetails);

session.getTransaction().commit();

session.close();

//update operation

session=sessionFactory.openSession();

userdetails=(UserDetails) session.get(UserDetails.class,7); //6th row

System.out.println("record pulled up is::"+userdetails.getUserId() +" : "+userdetails.getUserName());

userdetails.setUserName("SANJEEV");

session.beginTransaction();

session.update(userdetails);

session.getTransaction().commit();

session.close();

}

}

------------------------------------------------------------------------------------------------

Hibernate Embeddable Composite Primary Key using @Embeddable, @EmbeddedId

in this example , we will be declaring the IDs (Primary Key fields) as a separate class annotated with @Embeddable annotation.

An Employee is identified by its EmployeeId, which is defined by empId and department.

----------

@Embeddable

public class EmployeeId implements Serializable{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

@Column(name="EMP\_ID")

private int empId;

@Column(name="DEPARTMENT")

private String department;

public EmployeeId()

{

super();

}

public EmployeeId(int empId, String department)

{

super();

this.empId = empId;

this.department = department;

}

public int getEmpId(){

return empId;

}

public void setEmpId(int empId)

{

this.empId = empId;

}

public String getDepartment()

{

return department;

}

public void setDepartment(String department)

{

this.department = department;

}

/\*

@Override

public int hashCode()

{

final int prime = 31;

int result = 1;

result = prime \* result + ((department == null) ? 0 : department.hashCode());

result = prime \* result + empId;

return result;

}

@Override

public boolean equals(Object obj)

{

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

EmployeeId other = (EmployeeId) obj;

if (department == null)

{

if (other.department != null)

return false;

} else if (!department.equals(other.department))

return false;

if (empId != other.empId)

return false;

return true;

}

\*/

}

------------------

@Entity

@Table(name="Employee\_Details")

public class Employee implements Serializable{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

@EmbeddedId

private EmployeeId id;

@Column(name="EMP\_NAME")

private String empName;

public Employee()

{

super();

}

public Employee(EmployeeId id, String empName)

{

super();

this.id = id;

this.empName = empName;

}

public EmployeeId getId()

{

return id;

}

public void setId(EmployeeId id)

{

this.id = id;

}

public String getEmpName()

{

return empName;

}

public void setEmpName(String empName)

{

this.empName = empName;

}

}

-----------------------

public class EmployeeTest {

public static void main(String[] args) {

Employee employee=new Employee();

EmployeeId employeeId=new EmployeeId(1,"Blogging");

employee.setEmpName("JavaInterviewPoint");

employee.setId(employeeId);

Configuration cfg=new Configuration().configure();

SessionFactory sessionFactory=cfg.buildSessionFactory();

Session session=sessionFactory.openSession();

session.beginTransaction();

session.save(employee);

session.getTransaction().commit();

session.close();

}

}

----------------------

**Caching In Hibernate**