**Hibernate :**

* Hibernate is ORM solution for Java Applications.
* Hibernate is ORM tool given to the transfer the data between a java(object)application and a database(Relational) in the form of the objects. Hibernate is the open source, light weight tool given by **Gavin King**.
* Hibernate is non-invasive framework, which means it doesn't forces the programmer to extend or implement any class or interface.
* Hibernate can run with-in or with-out server, hence Hibernate is suitable for both stand-alone applications(desktop application) and web-applications.
* Hibernate is purely meant for persistence.

**Persistence :** The process of storing enterprise data in to relational database.

###### Draw Backs of JDBC :

1. In JDBC, to establish a simple database connection also, we need to write that code inside try block. And if any exceptions raised then we need to handle those exceptions inside catch block. Here try and catch blocks are mandatory.
2. After performing our required task, as a programmer we must close the connection manually, otherwise we may get out of connection error at runtime. Closing the connection and other resource-reallocating code we need to write inside finally block.
3. If we are not closing the connection manually then JDBC is not responsible to close the connection automatically.
4. After developing the JDBC program, if there is any changes occurred in the database table structure, then our JDBC program doesn't work because we will write static SQL Queries inside program.   
   So we need to rewrite the SQL commands according to the latest table structure. The re-compilation and re-deployment is mandatory.
5. Because of static SQL queries, JDBC will make our applications as database dependent. If we change the database then our applications doesn't work.
6. In JDBC, while retrieving the data from database, we will get that data in the form of ResultSet. Programmer is responsible to cover the data from ResultSet into our required objects.
7. JDBC will generate database specific exceptions and errors which are not familiar to java programmers.
8. In an Enterprise application, the data flow from one layer to another layer will be in the form of objects, but finally while transferring the object(data) from Data-Access layer to database, that object has to convert into text. Because JDBC cannot transfer the objects directly into the database.

###### Advantages of Hibernate :

1. Hibernate persists java objects directly into the database.
2. Hibernate generates efficient queries at runtime.
3. Hibernate has its own query language called as HQL which is database independent.
4. Hibernate applications are database independent.
5. Hibernate supports implicit cashing mechanism.
6. Hibernate supports collections like List, Set, Map.
7. Hibernate have translators which will convert checked exceptions into unchecked exceptions. So that try, catch blocks are not required.
8. Hibernate have different types of algorithms to generate primary key implicitly , while storing the objects in to database.

package hibernateex;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

/\*@Entity is an persistence class stores the data in database

\*@Table stores the data in database with the mentioned table name

@Id -->is the primary key value for every table we will have one primary key

@Column -->it is the column name

\*/

@Table(name="student\_mst\_tbl")

public class Student {

/\*

Here all are private varaibles .we cannot access private varaibles outside the class for accessing of outside members or classes

we are using setters and getters

setters--> are called mutator methods it only sets the data it will not return any data

its return type is void

public void setId(int id) {

this.id = id;

}

getters--> these are accessor methods ,only meant for get the data it will return data

its return type is based on private variables datatype,here it is int

private int id;

public int getId() {

return id;

}

Note: If a Class contains private variables and for accessing setters and getters ,we call that class as Pojo Class

Note:If that Pojo class saving data in database we call it as Entity(Persistence Class)

\*/

@Id

@Column(name="ID")

private int id;

@Column(name="NAME")

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

package hibernateex;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.AnnotationConfiguration;

import org.hibernate.cfg.Configuration;

public class StudentTest {

public static void main(String[] args) {

/\* To communicate with the database we need parameters like connection,url,username,schema we can place database details in properties file called hibernate.properties, or as an XML file named hibernate.cfg.xml. here we placed all database details in hibernate.cfg.xml

\* Configuration class read the configuration details (hibernate.cfg.xml)

\*

\* Configuration cfg= new AnnotationConfiguration().configure("hibernateex/hibernate.configuration.xml");

\* if we use annotations we create the configuration class as mentioned above (AnnotationConfiguration())

\*

\* By using configuration reference (cfg) ,SessionFactory establish the jdbc connection because configuration will have database details

SessionFactory sfac= cfg.buildSessionFactory();

it gives jdbc connection

\* We have jdbc connection ,to perform crud operations on database just like in jdbc we don't write statements or prepared statements ,hibernate internally it communicates with jdbc and generates jdbc queries

\* To do crud operations we have to get Session(it is mediator between java and database,it is physical connection between java and database) because all appropriate methods are present in session only

\* Ex: save() method for insertion ,update() method for update, get() method for selection of data ,delete() method for delete data

\* To do operations we have to get first jdbc connection ,by the help of SessionFactory(sfac) we will open one Session to do crud operations

\* Session ses= sfactory.openSession();

\* now we got one session to perform database manipulations and

\* to apply changes on the database(like insert,update,delete) we have to take transactions ,all our methods are in session only that's y we have to begin transactions on session only

\* Transaction tx= ses.beginTransaction();

\* if we want apply changes on database we have to call explicitly

\* tx.commit();

\*

\* if we don't want to apply changes on database we have to call explicitly

\* tx.rollback();

\* for insert,update,delete only we have to call explicitly

tx.commit();

for select statements we dont call

tx.commit();

\*

\*

\*

\* In this I want to insert student details on database

\* Create the student object and set the data to it

\*

\* Student stu= new Student();

stu.setId(15);

stu.setName("srinadh");

after setting the data to student object and call save method in session

ses.save(stu);

Internally save() method will have insert query, session internally communicates with jdbc and generates prepared statement like

\*

\* Hibernate:

insert into student\_mst\_tbl(NAME, ID)

values(?, ?)

after calling save() method commit the transaction

tx.commit();

\*

\* after commit its better to close SessionFactory,Session

sfactory.close();

\* ses.close();

\*

\*

\*

\*/

Configuration cfg= new AnnotationConfiguration().configure("hibernate.configuration.xml");

SessionFactory sfactory = cfg.buildSessionFactory();

Session ses= sfactory.openSession();

Transaction tx= ses.beginTransaction();

Student stu= new Student();

stu.setId(20);

stu.setName("srinadh");

ses.save(stu);

tx.commit();ses.close();

}

}

<?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>

<!-- database details starts -->

<!-- The JDBC driver class. -->

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

<!-- The JDBC URL to the database instance. -->

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

<!-- database username -->

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

<!-- database password -->

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

<!-- database details ends it is common for every database here we are using mysql database -->

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<!-- Below properties additionally provided by hibernate -->

<!-- This property makes Hibernate generate the appropriate SQL for the chosen database. -->

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

<!-- Below one is hibernate tool(hbm2ddl.auto) it automatically creates table in db

hibernate.hbm2ddl.auto"=create-> it will always creates the table

hibernate.hbm2ddl.auto"=update ->first it checks table is created or not if created ,it doesnt create otherwise it create

-->

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

<!-- hibernate.show\_sql"=true -> it will generates sql queries to the console -->

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

<!-- hibernate.format\_sql"=true ->it will generates sql queries in proper format in console -->

<property name="hibernate.format\_sql">true</property>

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<!-- if we use annotations at class level we have to refer (mapping class="hibernateex.Student" -->

<mapping class="hibernateex.Student"></mapping>

</session-factory>

</hibernate-configuration>

package com.pioneercoders.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="course\_mst\_tbl",schema="hibernate\_db")

public class CourseEntity {

@Id

@GeneratedValue

@Column(name="COURSE\_ID")

private Integer courseId;

@Column(name="COURSE\_NAME")

private String courseName;

@Column(name="FACULTY\_NAME")

private String facultyName;

@Column(name="COURSE\_FEE")

private float courseFee;

@Column(name="COURSE\_DURATION")

private Integer courseDuration;

public Integer getCourseId() {

return courseId;

}

public void setCourseId(Integer courseId) {

this.courseId = courseId;

}

public String getCourseName() {

return courseName;

}

public void setCourseName(String courseName) {

this.courseName = courseName;

}

public String getFacultyName() {

return facultyName;

}

public void setFacultyName(String facultyName) {

this.facultyName = facultyName;

}

public float getCourseFee() {

return courseFee;

}

public void setCourseFee(float courseFee) {

this.courseFee = courseFee;

}

public Integer getCourseDuration() {

return courseDuration;

}

public void setCourseDuration(Integer courseDuration) {

this.courseDuration = courseDuration;

}

@Override

public String toString() {

return "CourseEntity [courseId=" + courseId + ", courseName="

+ courseName + ", facultyName=" + facultyName + ", courseFee="

+ courseFee + ", courseDuration=" + courseDuration + "]";

}

}

package com.pioneercoders.util;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class HibernateUtil {

public SessionFactory sFactory =null;

public static SessionFactory getSessionFactory(){

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

SessionFactory sFactory = cfg.buildSessionFactory();

return sFactory;

}

}

package com.pioneercoders.test;

import com.pioneercoders.dao.CourseDao;

import com.pioneercoders.dao.CourseDaoImpl;

public class CourseEntityDeleteTest {

public static void main(String[] args) {

CourseDao cdao = new CourseDaoImpl();

int deletedata = cdao.deleteCourseInfo(4);

System.out.println(deletedata);

}

}

package com.pioneercoders.test;

import com.pioneercoders.dao.CourseDao;

import com.pioneercoders.dao.CourseDaoImpl;

import com.pioneercoders.model.CourseEntity;

public class CourseEntityGetById {

public static void main(String[] args) {

CourseDao cdao = new CourseDaoImpl();

CourseEntity entity = cdao.getById(3);

if(entity!=null){

System.out.println("Details are"+entity.getCourseFee()+" "+entity.getCourseName()+" "+entity.getFacultyName());

}

}

}

package com.pioneercoders.test;

import com.pioneercoders.dao.CourseDao;

import com.pioneercoders.dao.CourseDaoImpl;

public class CourseEntityInsertTest {

public static void main(String[] args) {

CourseDao cdao = new CourseDaoImpl();

//cdao.insertCourseInfo("core java", "nagoor babu", 1000.0f, 4);

//cdao.insertCourseInfo("adv java", "nagoor babu", 1000.0f, 3);

//cdao.insertCourseInfo("scjp", "durga", 2000.0f, 4);

//cdao.insertCourseInfo("scwcd", "durga", 2000.0f, 4);

cdao.insertCourseInfo("spring", "sriman", 2000.0f, 4);

System.out.println("inserted successfully ");

}

}

package com.pioneercoders.test;

import java.util.List;

import com.pioneercoders.dao.CourseDao;

import com.pioneercoders.dao.CourseDaoImpl;

import com.pioneercoders.model.CourseEntity;

public class CourseEntityListTest {

public static void main(String[] args) {

CourseDao cdao = new CourseDaoImpl();

List<CourseEntity> li = cdao.getAllCourses();

for(CourseEntity data:li){

System.out.println(data);

}

}

}

package com.pioneercoders.test;

import com.pioneercoders.dao.CourseDao;

import com.pioneercoders.dao.CourseDaoImpl;

public class CourseEntityUpdateTest {

public static void main(String[] args) {

CourseDao cdao = new CourseDaoImpl();

int updatedata = cdao.updateCourseInfo("hibernate", "sriman", 3);

System.out.println(updatedata);

}

}

package com.pioneercoders.dao;

import java.util.List;

import com.pioneercoders.model.CourseEntity;

public interface CourseDao {

public List<CourseEntity> getAllCourses();

public CourseEntity getById(int courseid);

public void insertCourseInfo(String coursename,String facultyname,float coursefee,int courseduration);

public int updateCourseInfo(String coursename,String facultyname,int courseid);

public int deleteCourseInfo(int courseid);

}

package com.pioneercoders.dao;

import java.util.List;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.model.CourseEntity;

import com.pioneercoders.util.HibernateUtil;

public class CourseDaoImpl implements CourseDao{

public List<CourseEntity> getAllCourses() {

// TODO Auto-generated method stub

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

Query query = ses.createQuery("from CourseEntity");

List<CourseEntity> li= query.list();

return li;

}

public CourseEntity getById(int courseid) {

// TODO Auto-generated method stub

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

Query query = ses.createQuery("from CourseEntity where courseId=:courseid");

query.setInteger("courseid", courseid);

CourseEntity centity = new CourseEntity();

centity = (CourseEntity)query.uniqueResult();

tx.commit();

return centity;

}

public void insertCourseInfo(String coursename,

String facultyname, float coursefee, int courseduration) {

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

CourseEntity coentity = new CourseEntity();

coentity.setCourseName(coursename);

coentity.setFacultyName(facultyname);

coentity.setCourseFee(coursefee);

coentity.setCourseDuration(courseduration);

ses.save(coentity);

tx.commit();

// TODO Auto-generated method stub

}

public int updateCourseInfo(String coursename, String facultyname,

int courseid) {

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx=ses.beginTransaction();

Query query = ses.createQuery("update CourseEntity set courseName=:coursename,facultyName=:facultyname where courseId=:courseid");

query.setString("coursename",coursename);

query.setString("facultyname", facultyname);

query.setInteger("courseid",courseid);

int updated = query.executeUpdate();

tx.commit();

return updated;

}

public int deleteCourseInfo(int courseid) {

// TODO Auto-generated method stub

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx=ses.beginTransaction();

Query query = ses.createQuery("delete from CourseEntity where courseId=:courseid");

query.setInteger("courseid", courseid);

int deleted = query.executeUpdate();

return deleted;

}

}

package com.pioneercoders.model;

public class Employee {

private int id;

private String firstname;

private String lastname;

public int getId() {

return id;

}

public void setId(int 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) {

this.lastname = lastname;

}

}

package com.pioneercoders.test;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.model.Employee;

import com.pioneerscoders.util.HibernateUtil;

public class GeneratorInsertTest {

public static void main(String[] args) {

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx = ses.beginTransaction();

//create the object and set the data

Employee emp = new Employee();

emp.setFirstname("srinadh");

emp.setLastname("m");

//save the data

ses.save(emp);

//commit the transaction

tx.commit();

System.out.println("success");

}

}

package com.pioneercoders.test;

public class KeyGeneratorsTheory {

/\*

\* Id generators in hibernate are generators to generate identifier or

\* primary key value for a persistent object while saving an object in

\* database

\*

\*

\*

\* Hibernate provides different primary key generator algorithms. All

\* hibernate generator classes implements hibernate.id.IdentifierGenerator

\* interface, and overrides the generate(SessionImplementor,Object) method

\* to generate the ‘identifier or primary key value‘. If we want our own

\* user defined generator, then we should implement IdentiferGenerator

\* interface and override the generate() <generator /> tag (which is sub

\* element of <id /> tag) is used to configure generator class in mapping

\* file.

\* Hibernate built-in generator classes

assigned (default key generator ) applicable for all databases

increment

sequence

hilo

native

identity

seqhilo

uuid

guid

select

1) Assigned

Assigned generator class is the default generator if there is no <generator> tag and supports in all the databases.

Developer should assign the identifier value to entity object before saving into the database.

Assigned generator configuration in hibernate mapping file.

How to declare generators in hibernate mapping file u can see below we can place key generators in below the primary key column we have to mention generator

<id name="empId" column="EMPNO">

<generator class="assigned"/>

</id>

OR

<id name="empId" column="EMPNO">

<generator class="org.hibernate.id.Assigned"/>

</id>

OR

<id name="empId" column="EMPNO"/>

2.Increment

Increment generator supports in all databases and generates identifier value for new records by using below formula.

Max of Id value in Database + 1

For first record it assigns 1 to the identifier. For second record it assigns based on above formula. i.e.( Max of Id value in Database + 1) =( 1+1 ) = 2.

Increment generator configuration in hibernate mapping file.

<id name="empId" column="EMPNO">

<generator class="increment"/>

</id>

OR

<id name="empId" column="EMPNO">

<generator class="org.hibernate.id.IncrementGenerator"/>

</id>

In this example we are showing assigned and increments generators first, and there working is

keep the comment for increment generator and try with assigned after assigned generator working ,

let's keep comment for assigned an try with increment first we used assigned generator it inserts value in db value is 1 and then

second if we use increment generator it increments to insertedvalue+1 (initialvalue+1) value is two

\*

\*

\*

\*

\*

\*

\*

\*

\*/

}

<?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>

<!-- database details starts -->

<!-- The JDBC driver class. -->

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

<!-- The JDBC URL to the database instance. -->

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

<!-- database username -->

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

<!-- database password -->

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

<!-- database details ends it is common for every database here we are using mysql database -->

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<!-- Below properties additionally provided by hibernate -->

<!-- This property makes Hibernate generate the appropriate SQL for the chosen database. -->

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

<!-- Below one is hibernate tool(hbm2ddl.auto) it automatically creates table in db

hibernate.hbm2ddl.auto"=create-> it will always creates the table

hibernate.hbm2ddl.auto"=update ->first it checks table is created or not if created ,it doesnt create otherwise it create

-->

<!-- hibernate.show\_sql"=true -> it will generates sql queries to the console -->

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

<!-- hibernate.format\_sql"=true ->it will generates sql queries in proper format in console -->

<property name="hibernate.format\_sql">true</property>

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<!-- if we use annotations at class level we have to refer (mapping class="hibernateex.Student" -->

<mapping resource="employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="com.pioneercoders.model.Employee" table="employee\_generate\_tbl">

<id name="id">

<!--<generator class="assigned"></generator>-->

<generator class="increment"/>

</id>

<property name="firstname"></property>

<property name="lastname"></property>

</class>

</hibernate-mapping>

package com.pioneercoders.dao;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entity.ContractEmployee;

import com.pioneercoders.util.HibernateUtil;

public class ContractEmployeeDao {

public void insertContractEmployeeInfo(ContractEmployee contractemployee){

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

ses.save(contractemployee);

tx.commit();

}

}

package com.pioneercoders.dao;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entity.Employee;

import com.pioneercoders.util.HibernateUtil;

public class EmployeeDao {

public void insertEmployeeInfo(Employee employee){

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

ses.save(employee);

tx.commit();

}

}

package com.pioneercoders.dao;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entity.RegularEmployee;

import com.pioneercoders.util.HibernateUtil;

public class RegularEmployeeDao {

public void insertRegularEmployeeInfo(RegularEmployee regularemployee){

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

ses.save(regularemployee);

tx.commit();

}

}

package com.pioneercoders.entity;

import javax.persistence.AttributeOverride;

import javax.persistence.AttributeOverrides;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Table;

@Entity

@Table(name="contractemployee")

@AttributeOverrides({

@AttributeOverride(name="EMPLOYEE\_ID", column=@Column(name="EMPLOYEE\_ID")),

@AttributeOverride(name="EMPLOYEE\_NAME", column=@Column(name="EMPLOYEE\_NAME"))

})

public class ContractEmployee extends Employee{

@Column(name="PAY\_PER\_HOUR")

private float pay\_per\_hour;

@Column(name="CONTRACT\_DURATION")

private String contract\_duration;

public float getPay\_per\_hour() {

return pay\_per\_hour;

}

public void setPay\_per\_hour(float pay\_per\_hour) {

this.pay\_per\_hour = pay\_per\_hour;

}

public String getContract\_duration() {

return contract\_duration;

}

public void setContract\_duration(String contract\_duration) {

this.contract\_duration = contract\_duration;

}

}

package com.pioneercoders.entity;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Inheritance;

import javax.persistence.InheritanceType;

import javax.persistence.Table;

@Entity

@Table(name = "employee\_tbl")

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

public class Employee {

@Id

@Column(name="EMPLOYEE\_ID")

private int empid;

@Column(name="EMPLOYEE\_NAME")

private String empname;

public int getEmpid() {

return empid;

}

public void setEmpid(int empid) {

this.empid = empid;

}

public String getEmpname() {

return empname;

}

public void setEmpname(String empname) {

this.empname = empname;

}

}

package com.pioneercoders.entity;

import javax.persistence.AttributeOverride;

import javax.persistence.AttributeOverrides;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Table;

@Entity

@Table(name="regularemployee")

@AttributeOverrides({

@AttributeOverride(name="EMPLOYEE\_ID", column=@Column(name="employeeid")),

@AttributeOverride(name="EMPLOYEE\_NAME", column=@Column(name="EMPLOYEE\_NAME"))

})

public class RegularEmployee extends Employee {

@Column(name="SALARY")

private float salary;

@Column(name="BONUS")

private int bonus;

public float getSalary() {

return salary;

}

public void setSalary(float salary) {

this.salary = salary;

}

public int getBonus() {

return bonus;

}

public void setBonus(int bonus) {

this.bonus = bonus;

}

}

package com.pioneercoders.entity;

public class TablePerClassTheory {

/\*

\* In this strategy we will create tables per each class

\*

\* employee -- >employee table contractemployee -->contractemployee table

\* regular table--> regularemployee table

\*

\* to inherit parent class(Employee) variables in sub class

\* (ContractEmployee,RegularEmployee) we will

\*

\* use @AttribiuteOverrides annotation and override those variables as

\* mentioned below

\*

\* @AttributeOverrides({

\*

\* @AttributeOverride(name="EMPLOYEE\_ID",

\* column=@Column(name="employeeid")),

\*

\* @AttributeOverride(name="EMPLOYEE\_NAME",

\* column=@Column(name="employeename")) })

\*

\* in this no need of discriminator column ,because we storing data

\* different tables

\*

\* Note :In this approach we will find some benefits and drawbacks some of

\* them are

\*

\* Advantage:

\*

\* Possible to define NOT NULL constraints on the table.

\* Slightly Normalized (data is separating in three tables )

\*

\* Disadvantage

\*

\* Duplicate columns are created in the subclass tables.

\*

\*/

}

package com.pioneercoders.test;

import com.pioneercoders.dao.ContractEmployeeDao;

import com.pioneercoders.dao.EmployeeDao;

import com.pioneercoders.dao.RegularEmployeeDao;

import com.pioneercoders.entity.ContractEmployee;

import com.pioneercoders.entity.Employee;

import com.pioneercoders.entity.RegularEmployee;

public class InsertInfoTest {

public static void main(String[] args) {

EmployeeDao edao = new EmployeeDao();

ContractEmployeeDao cdao = new ContractEmployeeDao();

RegularEmployeeDao rdao = new RegularEmployeeDao();

Employee emp = new Employee();

emp.setEmpid(12);

emp.setEmpname("rajesh");

ContractEmployee cemployee = new ContractEmployee();

cemployee.setEmpid(20);

cemployee.setEmpname("shivanarayan");

cemployee.setPay\_per\_hour(1000.0f);

cemployee.setContract\_duration("one year");

RegularEmployee regemployee = new RegularEmployee();

regemployee.setEmpid(33);

regemployee.setEmpname("sunil");

regemployee.setBonus(3000);

regemployee.setSalary(20000.0f);

edao.insertEmployeeInfo(emp);

cdao.insertContractEmployeeInfo(cemployee);

rdao.insertRegularEmployeeInfo(regemployee);

System.out.println("success");

/\*

\* Atter running this class it generates queries as

\*

\* Hibernate: insert into employee\_tbl (EMPLOYEE\_NAME, EMPLOYEE\_ID)

\* values (?, ?)

\*

\* Hibernate: insert into contractemployee (EMPLOYEE\_NAME,

\* CONTRACT\_DURATION, PAY\_PER\_HOUR, EMPLOYEE\_ID) values (?, ?, ?, ?)

\*

\*

\* Hibernate: insert into regularemployee (EMPLOYEE\_NAME, BONUS, SALARY,

\* EMPLOYEE\_ID) values (?, ?, ?, ?)

\*/

}

}

<?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>

<!-- database details starts -->

<!-- The JDBC driver class. -->

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

<!-- The JDBC URL to the database instance. -->

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

<!-- database username -->

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

<!-- database password -->

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

<!-- database details ends it is common for every database here we are using mysql database -->

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<!-- Below properties additionally provided by hibernate -->

<!-- This property makes Hibernate generate the appropriate SQL for the chosen database. -->

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

<!-- Below one is hibernate tool(hbm2ddl.auto) it automatically creates table in db

hibernate.hbm2ddl.auto"=create-> it will always creates the table

hibernate.hbm2ddl.auto"=update ->first it checks table is created or not if created ,it doesnt create otherwise it create

-->

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

<!-- hibernate.show\_sql"=true -> it will generates sql queries to the console -->

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

<!-- hibernate.format\_sql"=true ->it will generates sql queries in proper format in console -->

<property name="hibernate.format\_sql">true</property>

<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

<!-- if we use annotations at class level we have to refer (mapping class="hibernateex.Student" -->

<mapping class="com.pioneercoders.entity.Employee"></mapping>

<mapping class="com.pioneercoders.entity.RegularEmployee"></mapping>

<mapping class="com.pioneercoders.entity.ContractEmployee"></mapping>

</session-factory>

</hibernate-configuration>

package com.pioneercoders.dao;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entity.Contract\_Employee;

import com.pioneercoders.util.HibernateUtil;

public class ContractEmployeeDao {

public void insertContractEmployeeInfo(Contract\_Employee contractemployee){

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

ses.save(contractemployee);

tx.commit();

}

}

package com.pioneercoders.dao;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entity.Employee;

import com.pioneercoders.util.HibernateUtil;

public class EmployeeDao {

public void insertEmployeeInfo(Employee employee){

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

ses.save(employee);

tx.commit();

}

}

package com.pioneercoders.dao;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entity.Regular\_Employee;

import com.pioneercoders.util.HibernateUtil;

public class RegularEmployeeDao {

public void insertRegularEmployeeInfo(Regular\_Employee regularemployee){

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

ses.save(regularemployee);

tx.commit();

}

}

package com.pioneercoders.entity;

import javax.persistence.Column;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

@Entity

@DiscriminatorValue(value="contractemployee")

public class Contract\_Employee extends Employee{

@Column(name="PAY\_PER\_HOUR")

private float pay\_per\_hour;

@Column(name="CONTRACT\_DURATION")

private String contract\_duration;

public float getPay\_per\_hour() {

return pay\_per\_hour;

}

public void setPay\_per\_hour(float pay\_per\_hour) {

this.pay\_per\_hour = pay\_per\_hour;

}

public String getContract\_duration() {

return contract\_duration;

}

public void setContract\_duration(String contract\_duration) {

this.contract\_duration = contract\_duration;

}

}

package com.pioneercoders.entity;

import javax.persistence.Column;

import javax.persistence.DiscriminatorColumn;

import javax.persistence.DiscriminatorType;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Inheritance;

import javax.persistence.InheritanceType;

import javax.persistence.Table;

@Entity

@Table(name="employee\_mst\_tbl")

@Inheritance(strategy=InheritanceType.SINGLE\_TABLE)//singletable strategy

@DiscriminatorColumn(name="EMPLOYEE\_TYPE",

discriminatorType=DiscriminatorType.STRING)//to show details of particular class

@DiscriminatorValue(value="employee")

public class Employee {

@Id

@Column(name="EMP\_ID")

@GeneratedValue

private int empId;

@Column(name="EMP\_NAME")

private String empName;

public int getEmpId() {

return empId;

}

public void setEmpId(int empId) {

this.empId = empId;

}

public String getEmpName() {

return empName;

}

public void setEmpName(String empName) {

this.empName = empName;

}

}

package com.pioneercoders.entity;

public class InheritanceMappingTheory {

/\* In hibernate we can map inheritance relationships also

\*

\*

\* We know inheritance functionality

\*

\* if we need features are functionality that already present in base class or parent simply we can inherit that class on top of that we can add additional features

\*

\* ex: In employee class

\* public class {

\* int empId;

\* String empname ;

\*

\* it is common for every employee if we want we can inherit and add other functionality

\*

\* ContractEmployee extends Employee RegularEmployee extends Employee

\* float pay\_per\_hour float salary

\* int contract\_period int bonus

\*

\*

\*

\* for contract employee empId ,empname same but salary may be different from regular employee for this Scenario we are applying Inheritance mapping

\*

\* we can map inheritance mapping in three strategies

\*

\* Table Per Hierarchy

Table Per Concrete class

Table Per Subclass

\* In Table Per Hierarchy we need only single table to map the whole hierarchy to identify the data belongs to which class in single table we are adding additional column Discriminator column

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*/

}

package com.pioneercoders.entity;

import javax.persistence.Column;

import javax.persistence.DiscriminatorValue;

import javax.persistence.Entity;

@Entity

@DiscriminatorValue(value="regularemployee")

public class Regular\_Employee extends Employee {

@Column(name="SALARY")

private float salary;

@Column(name="BONUS")

private int bonus;

public float getSalary() {

return salary;

}

public void setSalary(float salary) {

this.salary = salary;

}

public int getBonus() {

return bonus;

}

public void setBonus(int bonus) {

this.bonus = bonus;

}

/\* Note :Table per hierarchy this approach will have some advantages and disadvantages i'm discussing below

\*

\* Single Table Strategy

In Single table per subclass, the union of all the properties from the inheritance hierarchy is mapped to one table. As all the data goes in one table, a discriminator is used to differentiate between different type of data.

Advantages of Single Table per class hierarchy

Simplest to implement.

Only one table to deal with.

Performance wise better than all strategies because no joins or sub-selects need to be performed.

Single Insert is needed to insert data

Disadvantages:

Most of the column of table are nullable so the NOT NULL constraint cannot be applied.

Tables are not normalized.

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*/

}

package com.pioneercoders.test;

import com.pioneercoders.dao.ContractEmployeeDao;

import com.pioneercoders.dao.EmployeeDao;

import com.pioneercoders.dao.RegularEmployeeDao;

import com.pioneercoders.entity.Contract\_Employee;

import com.pioneercoders.entity.Employee;

import com.pioneercoders.entity.Regular\_Employee;

public class InsertTest {

public static void main(String[] args) {

EmployeeDao edao = new EmployeeDao();

ContractEmployeeDao cdao = new ContractEmployeeDao();

RegularEmployeeDao rdao = new RegularEmployeeDao();

Employee emp = new Employee();

emp.setEmpName("Rajesh");

Contract\_Employee cemployee = new Contract\_Employee();

cemployee.setContract\_duration("six months");

cemployee.setEmpName("Kulkarni");

cemployee.setPay\_per\_hour(500.0f);

Regular\_Employee regemployee = new Regular\_Employee();

regemployee.setBonus(5000);

regemployee.setEmpName("durga");

regemployee.setSalary(25000.0f);

edao.insertEmployeeInfo(emp);

cdao.insertContractEmployeeInfo(cemployee);

rdao.insertRegularEmployeeInfo(regemployee);

/\*

\* Note: After Running this test class u will find three insert queries

\*

\* Hibernate: insert into employee\_mst\_tbl (EMP\_NAME, EMPLOYEE\_TYPE)

\* values (?, 'employee') //id is automatically generated ,employee is

\* discriminator value

\*

\* Hibernate: insert into employee\_mst\_tbl (EMP\_NAME, CONTRACT\_DURATION,

\* PAY\_PER\_HOUR, EMPLOYEE\_TYPE) values (?, ?, ?, 'contractemployee')

\* contractemployee is discriminator value

\*

\*

\* Hibernate: insert into employee\_mst\_tbl (EMP\_NAME, BONUS, SALARY,

\* EMPLOYEE\_TYPE) values (?, ?, ?, 'regularemployee') regularemployee is

\* discriminator value

\*/

}

}

package com.pioneercoders.dao;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entities.ContractEmployee;

import com.pioneercoders.util.HibernateUtility;

public class ContractEmployeeDao {

public void insertContractEmployeeInfo(ContractEmployee contractemployee){

Session ses = HibernateUtility.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

ses.save(contractemployee);

tx.commit();

}

}

package com.pioneercoders.dao;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entities.Employee;

import com.pioneercoders.util.HibernateUtility;

public class EmployeeDao {

public void insertEmployeeInfo(Employee employee){

Session ses = HibernateUtility.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

ses.save(employee);

tx.commit();

}

}

package com.pioneercoders.dao;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entities.RegularEmployee;

import com.pioneercoders.util.HibernateUtility;

public class RegularEmployeeDao {

public void insertRegularEmployeeInfo(RegularEmployee regularemployee){

Session ses = HibernateUtility.getSessionFactory().openSession();

Transaction tx= ses.beginTransaction();

ses.save(regularemployee);

tx.commit();

}

}

package com.pioneercoders.dao;

public class TablePerSubClassTheory {

/\*

\* in case of table per subclass strategy, tables are created as per

\* persistent classes but they are related using primary and foreign key. So

\* there will not be duplicate columns in the relation.

\*

\*

\* To do table per subclass we have to follow the strategy

\*

\* We need to specify @Inheritance(strategy=InheritanceType.JOINED) in the

\* parent class and @PrimaryKeyJoinColumn annotation in the subclasses.

\*

\*

\* Note : With this approach we have some pros and cons

\*

\* With Joined Strategy It's highly normalized but performance is not good.

\* Advantage: Tables are normalized.

\* Able to define NOT NULL constraint.

\*

\* Disadvantage: Does not perform as well as SINGLE\_TABLE strategy

\*/

}

package com.pioneercoders.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.PrimaryKeyJoinColumn;

import javax.persistence.Table;

@Entity

@Table(name="contractemployee\_sub\_tbl")

@PrimaryKeyJoinColumn(name="ID")

public class ContractEmployee extends Employee{

@Column(name="PAY\_PER\_HOUR")

private float pay\_per\_hour;

@Column(name="CONTRACT\_DURATION")

private String contract\_duration;

public float getPay\_per\_hour() {

return pay\_per\_hour;

}

public void setPay\_per\_hour(float pay\_per\_hour) {

this.pay\_per\_hour = pay\_per\_hour;

}

public String getContract\_duration() {

return contract\_duration;

}

public void setContract\_duration(String contract\_duration) {

this.contract\_duration = contract\_duration;

}

}

package com.pioneercoders.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Inheritance;

import javax.persistence.InheritanceType;

import javax.persistence.Table;

@Entity

@Table(name = "employee\_mst\_tbl")

@Inheritance(strategy=InheritanceType.JOINED)

public class Employee {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

@Column(name = "EMPLOYEE\_ID")

private int employeeid;

@Column(name = "EMPLOYEE\_NAME")

private String employeename;

public int getEmployeeid() {

return employeeid;

}

public void setEmployeeid(int employeeid) {

this.employeeid = employeeid;

}

public String getEmployeename() {

return employeename;

}

public void setEmployeename(String employeename) {

this.employeename = employeename;

}

}

package com.pioneercoders.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.PrimaryKeyJoinColumn;

import javax.persistence.Table;

@Entity

@Table(name="regularemployee\_sub\_tbl")

@PrimaryKeyJoinColumn(name="ID")

public class RegularEmployee extends Employee{

@Column(name="SALARY")

private float salary;

@Column(name="BONUS")

private int bonus;

public float getSalary() {

return salary;

}

public void setSalary(float salary) {

this.salary = salary;

}

public int getBonus() {

return bonus;

}

public void setBonus(int bonus) {

this.bonus = bonus;

}

}

package com.pioneercoders.test;

import com.pioneercoders.dao.ContractEmployeeDao;

import com.pioneercoders.dao.EmployeeDao;

import com.pioneercoders.dao.RegularEmployeeDao;

import com.pioneercoders.entities.ContractEmployee;

import com.pioneercoders.entities.Employee;

import com.pioneercoders.entities.RegularEmployee;

public class InsertInformationTest {

public static void main(String[] args) {

EmployeeDao edao = new EmployeeDao();

ContractEmployeeDao cdao = new ContractEmployeeDao();

RegularEmployeeDao rdao = new RegularEmployeeDao();

Employee emp = new Employee();

emp.setEmployeename("anilll");

ContractEmployee cemployee = new ContractEmployee();

cemployee.setContract\_duration("one year");

cemployee.setPay\_per\_hour(300);

RegularEmployee regemployee = new RegularEmployee();

regemployee.setBonus(2000);

regemployee.setSalary(20000.0f);

// call the dao and set data to it

edao.insertEmployeeInfo(emp);

cdao.insertContractEmployeeInfo(cemployee);

rdao.insertRegularEmployeeInfo(regemployee);

System.out.println("successs");

/\*

\* Note: if we run this it generate the query as

\*

\* Hibernate: insert into employee\_mst\_tbl (EMPLOYEE\_NAME) values (?)

\*

\*

\* Hibernate: insert into employee\_mst\_tbl (EMPLOYEE\_NAME) values (?)

\* Hibernate: insert into contractemployee\_sub\_tbl (CONTRACT\_DURATION,

\* PAY\_PER\_HOUR, ID) values (?, ?, ?)

\*

\*

\* Hibernate: insert into employee\_mst\_tbl (EMPLOYEE\_NAME) values (?)

\* Hibernate: insert into regularemployee\_sub\_tbl (BONUS, SALARY, ID)

\* values (?, ?, ?)

\*/

}

}

package com.pioneercoders.entities;

import java.util.Set;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

@Entity

@Table(name="Department\_mst\_tbl")

public class Department {

@Id

@GeneratedValue

@Column(name="DEPARTMENT\_ID")

private Long departmentId;

@Column(name="DEPT\_NAME")

private String departmentName;

@OneToMany(mappedBy="department")

private Set<Employee> employees;

public Long getDepartmentId() {

return departmentId;

}

public void setDepartmentId(Long departmentId) {

this.departmentId = departmentId;

}

public String getDepartmentName() {

return departmentName;

}

public void setDepartmentName(String departmentName) {

this.departmentName = departmentName;

}

public Set<Employee> getEmployees() {

return employees;

}

public void setEmployees(Set<Employee> employees) {

this.employees = employees;

}

}

package com.pioneercoders.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

@Entity

@Table(name="Employee\_mst\_tbl")

public class Employee {

@Id

@GeneratedValue

@Column(name="employee\_id")

private Long employeeId;

@Column(name="firstname")

private String firstname;

@Column(name="lastname")

private String lastname;

@Column(name="cell\_phone")

private String cellphone;

@ManyToOne

@JoinColumn(name="department\_id")

private Department department;

public Long getEmployeeId() {

return employeeId;

}

public void setEmployeeId(Long employeeId) {

this.employeeId = employeeId;

}

public String getFirstname() {

return firstname;

}

public void setFirstname(String firstname) {

this.firstname = firstname;

}

public String getLastname() {

return lastname;

}

public void setLastname(String lastname) {

this.lastname = lastname;

}

public String getCellphone() {

return cellphone;

}

public void setCellphone(String cellphone) {

this.cellphone = cellphone;

}

public Department getDepartment() {

return department;

}

public void setDepartment(Department department) {

this.department = department;

}

}

package com.pioneercoders.entities;

public class OneToManyMappingTheory {

/\*

\* create the tables in database schema as per below

\*

\* CREATE TABLE `Department\_mst\_tbl` ( `department\_id` BIGINT(20) NOT NULL

\* AUTO\_INCREMENT, `dept\_name` VARCHAR(50) NOT NULL DEFAULT '0', PRIMARY KEY

\* (`department\_id`) );

\*

\*

\*

\*

\* CREATE TABLE `Employee\_mst\_tbl` ( `employee\_id` BIGINT(10) NOT NULL

\* AUTO\_INCREMENT, `firstname` VARCHAR(50) NULL DEFAULT NULL, `lastname`

\* VARCHAR(50) NULL DEFAULT NULL,

\* `cell\_phone` VARCHAR(15) NULL DEFAULT NULL, `department\_id` BIGINT(20)

\* NULL DEFAULT NULL, PRIMARY KEY (`employee\_id`), INDEX `FK\_DEPT`

\* (`department\_id`), CONSTRAINT `FK\_DEPT` FOREIGN KEY (`department\_id`)

\* REFERENCES `Department\_mst\_tbl` (`department\_id`) );

\*

\*

\* Note: Employee and Department table exhibits One-to-many relationship.

\* Each Department can be associated with multiple Employees and each

\* Employee can have only one Department.

\*

\* @ManyToOne annotation defines a single-valued association to another

\* entity class that has many-to-one multiplicity. It is not normally

\* necessary to specify the target entity explicitly since it can usually be

\* inferred from the type of the object being referenced.

\*

\* @JoinColumn is used to specify a mapped column for joining an entity

\* association.

\* @OneToMany annotation defines a many-valued association with one-to-many multiplicity.

If the collection is defined using generics to specify the element type, the associated target entity type need not be specified; otherwise the target entity class must be specified.

The association may be bidirectional. In a bidirectional relationship, one of the sides (and only one) has to be the owner: the owner is responsible for the association column(s) update.

To declare a side as not responsible for the relationship, the attribute mappedBy is used. ' mappedBy' refers to the property name of the association on the owner side. In our case, this is department. As you can see, you don’t have to (must not) declare the join column since it has already been declared on the owners side.

\*

\*

\*

\*/

}

package com.pioneercoders.test;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entities.Department;

import com.pioneercoders.entities.Employee;

import com.pioneercoders.util.HibernateUtil;

public class InsertTest {

public static void main(String[] args) {

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx = ses.beginTransaction();

// first set the department data

Department department = new Department();

department.setDepartmentName("Sales");

ses.save(department);

Employee emp1 = new Employee();

emp1.setCellphone("9642123124");

emp1.setFirstname("lokesh");

emp1.setLastname("bb");

Employee emp2 = new Employee();

emp2.setCellphone("9642102129");

emp2.setFirstname("ashish");

emp2.setLastname("sh");

emp1.setDepartment(department);

emp2.setDepartment(department);

ses.save(emp1);

ses.save(emp2);

ses.getTransaction().commit();

ses.close();

System.out.println("success fully inserted");

/\*

\* Hibernate: insert into Department\_mst\_tbl (DEPT\_NAME) values (?)

\* Hibernate: insert into Employee\_mst\_tbl (cell\_phone, department\_id,

\* firstname, lastname) values (?, ?, ?, ?)

\* Hibernate:

\*

\*insert into Employee\_mst\_tbl (cell\_phone, department\_id, firstname, lastname)

\* values (?, ?, ?, ?)

\*/

}

}

package com.pioneercoders.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

/\*

\* @ManyToOne indicates that Many student tuples can refer to one University tuple.

\* Also note that we have provided optional=false means this relationship becomes mandatory , no student row can be saved without a university tuple reference.@JoinColumn says that there is a column UNIVERSITY\_ID in Student table which will refer(foreign key) to primary key of the University table. In this example only Student to University entity navigation is possible. Not viceversa.

\*/

@Entity

@Table(name="student\_mst\_tbl")

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 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) {

this.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;

}

}

package com.pioneercoders.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="university\_mst\_tbl")

public class University {

@Id

@GeneratedValue

@Column(name = "UNIVERSITY\_ID")

private long id;

@Column(name = "NAME")

private String name;

@Column(name = "COUNTRY")

private String 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;

}

}

package com.pioneercoders.test;

import org.hibernate.Session;

import com.pioneercoders.entities.Student;

import com.pioneercoders.entities.University;

import com.pioneercoders.util.HibernateUtil;

public class InsertInfoTest {

public static void main(String[] args) {

Session session = HibernateUtil.getSessionFactory().openSession();

session.beginTransaction();

Student student1 = new Student();

student1.setFirstName("bhargav");

student1.setLastName("su");

student1.setSection("science");

Student student2 = new Student();

student2.setFirstName("chiatanya");

student2.setLastName("p");

student2.setSection("chemistry");

University university = new University();

university.setName("Usmania");

university.setCountry("india");

student1.setUniversity(university);

student2.setUniversity(university);

session.persist(university);

session.persist(student1);

session.persist(student2);

session.getTransaction().commit();

System.out.println("success");

/\*

\* Hibernate: insert into university\_mst\_tbl (COUNTRY, NAME) values (?,?)

\*

\* Hibernate: insert into student\_mst\_tbl (FIRST\_NAME, LAST\_NAME,

\* SECTION, UNIVERSITY\_ID) values (?, ?, ?, ?)

\* Hibernate: insert into

\* student\_mst\_tbl (FIRST\_NAME, LAST\_NAME, SECTION, UNIVERSITY\_ID)

\* values (?, ?, ?, ?)

\*/

}

}

package com.pioneercoders.util;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class HibernateUtil {

public SessionFactory sFactory =null;

public static SessionFactory getSessionFactory(){

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

SessionFactory sFactory = cfg.buildSessionFactory();

return sFactory;

}

}

package com.pioneercoders.entities;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.JoinTable;

import javax.persistence.ManyToMany;

import javax.persistence.Table;

@Entity

@Table(name = "CART")

public class Cart {

@Id

@Column(name = "cart\_id")

@GeneratedValue(strategy = GenerationType.IDENTITY)

private long id;

@Column(name = "cart\_total")

private double total;

@ManyToMany(targetEntity = Item.class, cascade = { CascadeType.ALL })

@JoinTable(name = "CART\_ITEMS",

joinColumns = { @JoinColumn(name = "cart\_id") },

inverseJoinColumns = { @JoinColumn(name = "item\_id") })

private Set<Item> items;

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public double getTotal() {

return total;

}

public void setTotal(double total) {

this.total = total;

}

public Set<Item> getItems() {

return items;

}

public void setItems(Set<Item> items) {

this.items = items;

}

/\*

\* Most important part here is the use of ManyToMany annotation and JoinTable annotation where we provide table name and columns to be used for many-to-many mapping.

\*/

}

package com.pioneercoders.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="ITEM")

public class Item {

@Id

@Column(name="item\_id")

@GeneratedValue(strategy=GenerationType.IDENTITY)

private long id;

@Column(name="item\_price")

private double price;

@Column(name="item\_desc")

private String description;

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

}

package com.pioneercoders.entities;

public class ManyToManyTheory {

/\* Many to Many association will have third table

\* CREATE TABLE `Cart` ( `cart\_id` int(11) unsigned NOT NULL AUTO\_INCREMENT,

\* `cart\_total` decimal(10,0) NOT NULL, PRIMARY KEY (`cart\_id`) )

\* ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

\*

\* CREATE TABLE `Item` ( `item\_id` int(11) unsigned NOT NULL AUTO\_INCREMENT,

\* `item\_desc` varchar(20) NOT NULL, `item\_price` decimal(10,0) NOT NULL,

\* PRIMARY KEY (`item\_id`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

\*

\* CREATE TABLE `Cart\_Items` ( `cart\_id` int(11) unsigned NOT NULL,

\* `item\_id` int(11) unsigned NOT NULL, PRIMARY KEY (`cart\_id`,`item\_id`),

\* CONSTRAINT `fk\_cart` FOREIGN KEY (`cart\_id`) REFERENCES `Cart`

\* (`cart\_id`), CONSTRAINT `fk\_item` FOREIGN KEY (`item\_id`) REFERENCES

\* `Item` (`item\_id`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

\*/

}

package com.pioneercoders.test;

import java.util.HashSet;

import java.util.Set;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.entities.Cart;

import com.pioneercoders.entities.Item;

import com.pioneercoders.util.HibernateUtil;

public class InsertInfo {

public static void main(String[] args) {

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx = ses.beginTransaction();

Item item1 = new Item();

item1.setDescription("samsung");

item1.setPrice(300);

Item item2 = new Item();

item2.setDescription("nokia");

item2.setPrice(200);

Cart cart = new Cart();

cart.setTotal(500);

Set<Item> items = new HashSet<Item>();

items.add(item1);

items.add(item2);

cart.setItems(items);

ses.save(cart);

ses.getTransaction().commit();

}

/\*

\* Hibernate: insert into CART (cart\_total) values (?)

\* Hibernate: insert

\* into ITEM (item\_desc, item\_price) values (?, ?)

\* Hibernate: insert into

\* ITEM (item\_desc, item\_price) values (?, ?)

\* Hibernate: insert into

\* CART\_ITEMS (cart\_id, item\_id) values (?, ?)

\* Hibernate: insert into

\* CART\_ITEMS (cart\_id, item\_id) values (?, ?)

\*/

}

package com.pioneercoders.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="course\_mst\_tbl")

public class Course {

@Id

@Column(name="COURSE\_ID")

private int courseid;

@Column(name="COURSE\_NAME")

private String coursename;

@Column(name="COURSE\_FEE")

private float coursefee;

@Column(name="COURSE\_DURATION")

private String courseduration;

public int getCourseid() {

return courseid;

}

public void setCourseid(int courseid) {

this.courseid = courseid;

}

public String getCoursename() {

return coursename;

}

public void setCoursename(String coursename) {

this.coursename = coursename;

}

public float getCoursefee() {

return coursefee;

}

public void setCoursefee(float coursefee) {

this.coursefee = coursefee;

}

public String getCourseduration() {

return courseduration;

}

public void setCourseduration(String courseduration) {

this.courseduration = courseduration;

}

@Override

public String toString() {

return "Course [courseid=" + courseid + ", coursename=" + coursename

+ ", coursefee=" + coursefee + ", courseduration="

+ courseduration + "]";

}

}

package com.pioneercoders.test;

public class HQLTheory {

/\*

\* HQL (Hibernate Query Language)

\*

\*

\* Hibernate Query Language (HQL) is an object-oriented query language,

\* similar to SQL, but instead of operating on tables and columns, HQL works

\* with persistent objects and their properties. HQL queries are translated

\* by Hibernate into conventional SQL queries which in turns perform action

\* on database.

\*

\* Although you can use SQL statements directly with Hibernate using Native

\* SQL but we would recommend to use HQL .we directly operate on entity

\* class variables it internally generates sql queries

\*

\* Advantage of HQL

\*

\* There are many advantages of HQL. They are as follows:

\*

\* database independent

\* supports polymorphic queries

\* easy to learn for Java Programmer

\*

\*

\*

\* Ex: If we want to select data from database

\* in sql select \* from table\_name;

\* select \* from course\_mst\_tbl

\* it retruns all the data from database

\*

\* in hql

\*

\* from Entityclassname ;

\* from Course

\* it returns all the data from database

\*

\*

\* Query Interface

It is an object oriented representation of Hibernate Query. The object of Query can be obtained by calling the createQuery() method Session interface.

The query interface provides many methods. There is given commonly used methods:

public int executeUpdate() is used to execute the update or delete query.

public List list() returns the result of the ralation as a list.

public Query setFirstResult(int rowno) specifies the row number from where record will be retrieved.

public Query setMaxResult(int rowno) specifies the no. of records to be retrieved from the relation (table).

public Query setParameter(int position, Object value) it sets the value to the JDBC style query parameter.

public Query setParameter(String name, Object value) it sets the value to a named query parameter.

\*

\*

\*/

}

package com.pioneercoders.test;

import java.util.List;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.model.Course;

import com.pioneercoders.util.HibernateUtil;

public class SELECTLIST {

public static void main(String[] args) {

Session ses = HibernateUtil.getSessionFactory().openSession();

//in this we are showing example on hql ,in that how iterate list of data from database using hql

//first write the query to fetch from database

Transaction tx = ses.beginTransaction();

Query query =ses.createQuery("from Course");

//it will returns all the data its return type is list

List<Course> course = query.list();

//print that collection or use foreach loop to iterate

System.out.println(course);

for(Course element:course){

System.out.println(element);

}

//if we want to only some of the properties in course\_mst\_tbl like only coursenames we will select clause

Query queryselect= ses.createQuery("SELECT C.coursename FROM Course C");

List results = queryselect.list();

System.out.println(results);

//generally we use where clause to minimize the results i want to now where duration is only one month

//i'm getting only one result here i'm calling unique result from query

Query whereselect = ses.createQuery("FROM Course C WHERE C.courseduration='one'");

Course result =(Course) whereselect.uniqueResult();

System.out.println(result);

tx.commit();

ses.close();

}

}

package com.pioneercoders.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.Table;

@NamedQueries(

{

@NamedQuery(

name = "findCourseByName",

query = "from Course c where c.coursename = :coursename"

),

@NamedQuery(

name="findCourseNameByDuration",

query="from Course c where c.courseduration=:courseduration"

)

}

)

@Entity

@Table(name="course\_mst\_tbl")

public class Course {

@Id

@Column(name="COURSE\_ID")

private int courseid;

@Column(name="COURSE\_NAME")

private String coursename;

@Column(name="COURSE\_FEE")

private float coursefee;

@Column(name="COURSE\_DURATION")

private String courseduration;

public int getCourseid() {

return courseid;

}

public void setCourseid(int courseid) {

this.courseid = courseid;

}

public String getCoursename() {

return coursename;

}

public void setCoursename(String coursename) {

this.coursename = coursename;

}

public float getCoursefee() {

return coursefee;

}

public void setCoursefee(float coursefee) {

this.coursefee = coursefee;

}

public String getCourseduration() {

return courseduration;

}

public void setCourseduration(String courseduration) {

this.courseduration = courseduration;

}

@Override

public String toString() {

return "Course [courseid=" + courseid + ", coursename=" + coursename

+ ", coursefee=" + coursefee + ", courseduration="

+ courseduration + "]";

}

}

package com.pioneercoders.test;

import java.util.List;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.model.Course;

import com.pioneercoders.util.HibernateUtil;

public class InfoUsingNamedQuery {

public static void main(String[] args) {

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx = ses.beginTransaction();

// to get named queries where we mentioned in entity class call

// getNamedQuery()

Query query = ses.getNamedQuery("findCourseByName");

// pass the value to query

query.setString("coursename", "corejava");

List<Course> li = query.list();

System.out.println(li);

// getting information of courses based on courseduration

Query namedquery = ses.getNamedQuery("findCourseNameByDuration");

namedquery.setString("courseduration", "three");

List<Course> coursesbasedonduration = namedquery.list();

System.out.println(coursesbasedonduration);

/\*

\* the queries generated are

\*

\* findCourseByName

\*

\* Hibernate: select course0\_.COURSE\_ID as COURSE\_I1\_0\_,

\* course0\_.COURSE\_DURATION as COURSE\_D2\_0\_, course0\_.COURSE\_FEE as

\* COURSE\_F3\_0\_, course0\_.COURSE\_NAME as COURSE\_N4\_0\_ from

\* course\_mst\_tbl course0\_ where course0\_.COURSE\_NAME=? [Course

\* [courseid=1, coursename=corejava, coursefee=2000.0,

\* courseduration=four], Course [courseid=5, coursename=corejava,

\* coursefee=1000.0, courseduration=three]]

\*

\* findCourseNameByDuration

\*

\* Hibernate: select

\* course0\_.COURSE\_ID as COURSE\_I1\_0\_, course0\_.COURSE\_DURATION as

\* COURSE\_D2\_0\_, course0\_.COURSE\_FEE as COURSE\_F3\_0\_,

\* course0\_.COURSE\_NAME as COURSE\_N4\_0\_ from course\_mst\_tbl course0\_

\* where course0\_.COURSE\_DURATION=? [Course [courseid=3,

\* coursename=hibernate, coursefee=1500.0, courseduration=three], Course

\* [courseid=5, coursename=corejava, coursefee=1000.0,

\* courseduration=three]]

\*/

tx.commit();

ses.close();

}

}

package com.pioneercoders.test;

public class NamedQueriesTheory {

/\*

\*

\* Named Queries are useful to know the purpose of the query ex:

\* @NamedQuery( name = "findEmployeeByName", (the user can understand this

\* query is getting information of names) query =

\* "from Employee e where e.name = :name"

\*

\* The hibernate named query is way to use any query by some meaningful

\* name. It is like using alias names. The Hibernate framework provides the

\* concept of named queries so that application programmer need not to

\* scatter queries to all the java code.

\*

\* There are two ways to define the named query in hibernate:

\*

\* by annotation by mapping file.

\*

\*

\* Hibernate Named Query by annotation

\*

\* If you want to use named query in hibernate, you need to have knowledge

\* of @NamedQueries and @NamedQuery annotations.

\*

\* @NameQueries annotation is used to define the multiple named queries.

\*

\* @NameQuery annotation is used to define the single named query.

\*

\* Let's see the example of using the named queries:

\*

\* @NamedQueries( {

\*

\* @NamedQuery( name = "findEmployeeByName", query =

\* "from Course c where c.coursename = :name" ) } )

\*/

}

package com.pioneercoders.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="course\_mst\_tbl")

public class Course {

@Id

@Column(name="COURSE\_ID")

private int courseid;

@Column(name="COURSE\_NAME")

private String coursename;

@Column(name="COURSE\_FEE")

private float coursefee;

@Column(name="COURSE\_DURATION")

private String courseduration;

public int getCourseid() {

return courseid;

}

public void setCourseid(int courseid) {

this.courseid = courseid;

}

public String getCoursename() {

return coursename;

}

public void setCoursename(String coursename) {

this.coursename = coursename;

}

public float getCoursefee() {

return coursefee;

}

public void setCoursefee(float coursefee) {

this.coursefee = coursefee;

}

public String getCourseduration() {

return courseduration;

}

public void setCourseduration(String courseduration) {

this.courseduration = courseduration;

}

@Override

public String toString() {

return "Course [courseid=" + courseid + ", coursename=" + coursename

+ ", coursefee=" + coursefee + ", courseduration="

+ courseduration + "]";

}

}

package com.pioneercoders.test;

public class CriteriaApiTheory {

/\*

\* CriteriaApi introduced mainly to operate on database where logical operations are required to get data

\*

\* Hibernate provides alternate ways of manipulating objects and in turn data available in RDBMS tables.

\* One of the methods is Criteria API which allows you to build up a criteria query object programmatically where you can apply filtration rules and logical conditions(gt,lt,eq)

\*

\*

\* The Criteria interface provides methods to apply criteria such as retreiving all the records of table whose salary is greater than 50000 etc.

\*

\*

\*

\* We can retrieve data from database in hibernate they provided several efficient techniques

\*

\*

\* 1.HQL(if we want operate queries independent of database we will use HQL)

\* 2.Criteria Api (if largest amount of data is present in database we want to apply logical operations like (lessthan,greaterthan ,equal ) to suppress data we will use Criteria Api

\* 3.Native Sql (if we have strong knowledge on sql and we want apply db operations directly on database we will use Native Sql ,but in this approach db to db query may change

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*/

}

package com.pioneercoders.test;

import java.util.List;

import org.hibernate.Criteria;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.criterion.Restrictions;

import com.pioneercoders.model.Course;

import com.pioneercoders.util.HibernateUtil;

public class SelectCriteriaList {

public static void main(String[] args) {

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx = ses.beginTransaction();

//for select queries(select) we don't need to begin Transaction in hibernate

//for non-select queries(insert,update,delete) we have to begin transactions because we have to apply changes on db after saving changes until we call tx.commit() changes doesn't effect on db

//in this by using criteriaapi i'm getting list of data from database

Criteria c=ses.createCriteria(Course.class);//passing Entity Class as argument

List criterialist=c.list();

//while selecting data it's better to override toString() method otherwise it will prints hashcode values of Course Class

System.out.println(criterialist);

//if i want to get only some of the records from one to third record we will use below scenario

Criteria criteria=ses.createCriteria(Course.class);

c.setFirstResult(0);//'0' index it display first value in database

c.setMaxResults(3); //it will display three results

List minrecords=c.list();

System.out.println(minrecords);

//if want to display courses where coursefee greater than 1000 for this we have to restrictions (where clause in criteria)

Criteria crite=ses.createCriteria(Course.class);

c.add(Restrictions.gt("coursefee",1000.0f));//coursefee is the propertyname

List restrictresult=c.list();

System.out.println(restrictresult);

/\*

\* Hibernate:

select

this\_.COURSE\_ID as COURSE\_I1\_0\_0\_,

this\_.COURSE\_DURATION as COURSE\_D2\_0\_0\_,

this\_.COURSE\_FEE as COURSE\_F3\_0\_0\_,

this\_.COURSE\_NAME as COURSE\_N4\_0\_0\_

from

course\_mst\_tbl this\_

where

this\_.COURSE\_FEE>? limit ?

[Course [courseid=1, coursename=corejava, coursefee=2000.0, courseduration=four], Course [courseid=3, coursename=hibernate, coursefee=1500.0, courseduration=three], Course [courseid=4, coursename=spring, coursefee=1500.0, courseduration=four]]

\*/

tx.commit();

ses.close();

}

}

package com.pioneercoders.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="course\_mst\_tbl")

public class Course {

@Id

@Column(name="COURSE\_ID")

private int courseid;

@Column(name="COURSE\_NAME")

private String coursename;

@Column(name="COURSE\_FEE")

private float coursefee;

@Column(name="COURSE\_DURATION")

private String courseduration;

public int getCourseid() {

return courseid;

}

public void setCourseid(int courseid) {

this.courseid = courseid;

}

public String getCoursename() {

return coursename;

}

public void setCoursename(String coursename) {

this.coursename = coursename;

}

public float getCoursefee() {

return coursefee;

}

public void setCoursefee(float coursefee) {

this.coursefee = coursefee;

}

public String getCourseduration() {

return courseduration;

}

public void setCourseduration(String courseduration) {

this.courseduration = courseduration;

}

@Override

public String toString() {

return "Course [courseid=" + courseid + ", coursename=" + coursename

+ ", coursefee=" + coursefee + ", courseduration="

+ courseduration + "]";

}

}

package com.pioneercoders.test;

import java.util.List;

import org.hibernate.SQLQuery;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.model.Course;

import com.pioneercoders.util.HibernateUtil;

public class ListInfoEntityQuery {

public static void main(String[] args) {

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx = ses.beginTransaction();

// in this we are selecting data from db using native sql queries

// write the sql query to get data from db

String nativesql = "select \* from course\_mst\_tbl";

SQLQuery query = ses.createSQLQuery(nativesql);

// after writing sql in hibernate we are only dealing with objects

// mention which is the Entity class for equivalent course\_mst\_tbl

query.addEntity(Course.class);

List<Course> courses = query.list();

for (Course course : courses) {

System.out.println("Course details are " + " "

+ course.getCourseduration() + " " + course.getCoursefee()

+ " " + course.getCoursename());

}

/\*

\* Hibernate: select

\*

\* from course\_mst\_tbl

\* Course details are four 2000.0 corejava

\* Course details are one 1000.0 advjava

\*

\* Course details are three 1500.0 hibernate

\* Course details are four 1500.0 spring

\* Course details are three 1000.0 corejava

\*

\*/

tx.commit();

ses.close();

}

}

package com.pioneercoders.test;

import java.util.List;

import java.util.Map;

import org.hibernate.Criteria;

import org.hibernate.SQLQuery;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.pioneercoders.util.HibernateUtil;

public class ListInfoScalarQuery {

public static void main(String[] args) {

Session ses = HibernateUtil.getSessionFactory().openSession();

Transaction tx = ses.beginTransaction();

String sqlquery = "select course\_name,course\_fee,course\_duration from course\_mst\_tbl";

SQLQuery query = ses.createSQLQuery(sqlquery);

query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);// im iterating

// based on

// map

List courses = query.list();

for (Object course : courses) {

Map row = (Map) course;// it will map the row after getting course

System.out.println("Details are " + " " + row.get("course\_name")

+ " " + row.get("course\_fee") + " "

+ row.get("course\_duration"));

}

tx.commit();

ses.close();

/\*

\* Hibernate: select course\_name, course\_fee, course\_duration from

\* course\_mst\_tbl

\* Details are corejava 2000.0 four

\* Details are advjava 1000.0 one

\*

\* Details are hibernate 1500.0 three

\* Details are spring 1500.0 four

\*

\* Details are corejava 1000.0 three

\*/

}

}

package com.pioneercoders.test;

public class NativeSqlTheory {

/\* Hibernate also provides feature to write sql queries according to database which are database dependent

\* we can write sql queries using createSqlQuery()

\*

\* public SQLQuery createSQLQuery(String sqlString) throws HibernateException

\*

\* After you pass a string containing the SQL query to the createSQLQuery() method, you can associate the SQL result with either an existing Hibernate entity, a join, or a scalar result using addEntity(), addJoin(), and addScalar() methods respectively.

Scalar queries:

The most basic SQL query is to get a list of scalars (values) from one or more tables. Following is the syntax for using native SQL for scalar values:

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

SQLQuery query = session.createSQLQuery(sql);

query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);

List results = query.list();

Entity queries:

The above queries were all about returning scalar values, basically returning the "raw" values from the resultset. The following is the syntax to get entity objects as a whole from a native sql query via addEntity().

String sql = "SELECT \* FROM EMPLOYEE";

SQLQuery query = session.createSQLQuery(sql);

query.addEntity(Employee.class);

List results = query.list();

after writing sqlqueries if we want to get list of data we have two approaches

1) using Scalar Query

2) using Entity Query

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*/

}