**HIBERNATE**

Object Relational Mapping(ORM) Concept used in Hibernate. (Mapping nPOJO Objects to columns of the table)

Persistence storage of data so it is a persistent framework.

Hibernate is a persistent framework used to persist the data into the database. It is an open source light weight ORM based framework. To communicate with the datatabase.

ORM is a programming technique that maps the POJOs to the columns of the table in database.

4 Programs:

1. ***POJO class or Persistent*** class which contains the properties and its getter and setters based upon the columns of the table.
2. ***Hibernate.cfg.xml(present in src folder)*** file which contains the information about the database like

a. *hibernate.connection.driver-class*🡪 represents JDBC drivername

b. *hibernate.connection.url*🡪 JDBC connection URL

c*. hibernate.connection.username*🡪 represents database username

d. *hibernate.connection.password*🡪 represents database password

e. *hibernate.connection.pool-size*🡪 maximum number of pooled connection

f. *hibernate.dialect*🡪the class name of a hibernate org.hibernate.dialect. Dialect which allows hibernate to generate sql query for a particular query database.

g. *hibernate*.*show\_sql*🡪write all generated sql statement in the console in form of true/false(default)

h. *hibernate.format\_sql🡪* format the generated sql on the console. Value contained True/false(default)

i. *hibernate.use\_sql\_comments*🡪hibernate will generate comments inside the sql for easier debugging. It will contain True/False(default)

j. *hibernate.hbm2.ddl.auto*🡪 automatically validates or exports the schema DDL to the database. The values are:

i. validate🡪validates the schema makes not changes to the database.

ii. update🡪 if table not their, it creates . If table presents, it just inserts.

iii. create🡪creates a schema destroying the previous data.

iv. create-drop🡪 drop and create a clean database each time we deploy.

k. *hibernate.mapping*🡪give the information about the mapping file to the database.

1. Hibernate mapping file ***POJOCLASSNAME.hbm.xml*** file used to provide the mapping between the persistent object to the columns of the table.
2. <class>🡪represents the persistence class.
3. <id>🡪 represents the primary key for the column

* <generator>🡪 used to provide the value for primary key or auto generate.

1. <property>🡪 used to map the other columns
2. ***Main class*** used to store the data into the database for that we use

**4 interfaces in org.hibernate.\*** package.

1. *Configuration class*🡪 allows the application to access the database properties and mapping documents to be used when creating session factory.
2. *SessionFactory interface*🡪 used to create a session instance
3. *Session Interface* 🡪 The main interface between java application and hibernate and used to create read and delete operations for instance of mapped entity classes. It has ***3 states***:
4. Transcient: never persistent, not associated with any session
5. Persistent: associated with unique session.
6. Detached: previously persistent, now not associated with any session.

***Methods:***

1. **save()🡪** stores an object into the database and will fail if the primary key is already persistent.
2. **persist()**🡪same like save method but save() returns serializable object i.e. primary key, whereas persist method returns void.

Eg: System.out.print(session.save(emp));🡪returns the primary key

System.out.print(session.persist(emp));🡪returns error

1. **saveOrUpdate()🡪** call save() or update() based on operation, if the identifiers exist, it will call update() or else it calls save().
2. **update()🡪** used for updating the object using identifier. If the identifier is missing or doesn’t exists, it will throw exception.
3. **delete()🡪** removes the persistant instance from datasource based on the identifier.
4. **get()🡪**used to retrieve the object from the database. It always hits the database and returns the real object. Returns NULL if no row is found.

Eg: Employee e=(Employee) session.get(Employee.class,”Hundred”)

Hunderd🡪 Primary Key

Employee.class🡪 Class

1. **load()🡪** also used to retrieve the object, if no rows found, Throws ObjectNotFoundException.

Eg: Employee e=(Employee) session.load(Employee.class,”Hundred”)

1. **clear()🡪** clears the session.
2. **flush()🡪**flushing the session, simply makes the data i.e. currently in the session synchronized with the database.
3. **refresh()🡪**used to reload an object.
4. *Transaction Interface:* Used to specify the atomic unit of work

**Method:**

1. *commit()🡪*
2. *rollback()🡪*

**Generator:**

<id name="id" column="Emp\_id">

<generator **class="assigned**"></generator>

</id>

Assigned🡪

Auto🡪

Identity🡪

Hilo🡪

Native🡪

Foreign🡪

Seqhilo🡪

Uuid🡪

Guid🡪

<class dynamic-insert=”true”>🡪which means exclude the null property value in sql insert statement.

<class dynamic-update=”true”>🡪 excludes the unmodified properties in the sql update statement.

<property name=*"name"* column=*"Emp\_name" type=”data\_type” unique=”true” not-null=”true”*></property>

Data\_type🡪 of property

Unique=true🡪 unique constraint

Not-null🡪 constraint

Size🡪 by default 255. / size=100

**HIBERNATE ANNOTATION**

Used instead of hbm mapping file.

Programs are:

1. ***POJO class or Persistent:*** which contains the annotation used to map the persistent object with columns of the table. Present in javax.persistance package.

***@Entity***🡪represents the class is a persistent class.

***@Table(name=”Emp200”)*🡪** optional annotation used to map the persistent class with the table.

*Eg:* public class Employee{

*}*

***@Id(column=”emp\_id”)****🡪*infront of primary key to denote primary key

***@GeneratedValue(strategy=GenerationType.AUTO/IDENTITY/TABLE/SEQUENCE)***

Eg: private int id;

***@Column(name=”emp\_name”)***

private String name;

private String salary;

***@Temporal(TemporalType.DATE/TIME/TIMESTAMP)***

***@Column(name=”date\_of\_birth”)***

private Date dob;

private String designation;

***@Enumerated(EnumerationType.ORDINAL/STRING)***

private Gender gender;

}

Public enum Gender {male,female}

***Create object for Annnotation Configuration instead of using Configuration().***

**HIBERNATE INHERITANCE**

In Hibernate, the Persistence class can also be inherited .

3Types:

1. ***Table Per Hierarchy Inheritance🡪*** where it will create a single table for all persistent class.

***In Annotation:***

@Inheritance(strategy=InheritanceType.***SINGLE\_TABLE***)

@DiscriminatorColumn(name="emptype",discriminatorType=DiscriminatorType.***STRING***) //discriminate between the POJO classes

@DiscriminatorValue(value="emp")

***In HBM:***

<hibernate-mapping>

<class name=*"com.pack.Employee"* table=*"Employee300"*

discriminator-value=*"emp"*>

<!-- Column Attribute Used to denote the table name shown in db -->

<!-- Column used to denote primary key id in database -->

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<!-- Normal column generated -->

<discriminator column=*"empType"* type=*"string"*></discriminator> <!-- Always located above the property tag -->

<property name=*"name"* column=*"Emp\_name"*></property>

<!-- Sub-class details given -->

<subclass name=*"com.pack.Regular\_Employee"*

discriminator-value=*"reg\_emp"*>

<property name=*"salary"*></property>

<property name=*"bonus"*></property>

</subclass>

<subclass name=*"com.pack.Contract\_Employee"*

discriminator-value=*"con\_emp"*>

<property name=*"pay\_hour"*></property>

<property name=*"contract\_duration"*></property>

</subclass>

</class>

</hibernate-mapping>

1. ***Table Per Concrete Class Inheritance:***

Where the table will be created for individual persistent class.

***In HBM:***

<hibernate-mapping>

<class name=*"com.pack.Employee"* table=*"Employee400"*>

<!-- Column Attribute Used to denote the table name shown in db -->

<!-- Column used to denote primary key id in database -->

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<!-- Normal column generated -->

<property name=*"name"* column=*"Emp\_name"*></property>

<!-- Sub-class details given -->

<union-subclass name=*"com.pack.Regular\_Employee"* table=*"Employee401"*>

<property name=*"salary"*></property>

<property name=*"bonus"*></property>

</union-subclass>

<union-subclass name=*"com.pack.Contract\_Employee"* table=*"Employee402"* >

<property name=*"pay\_hour"*></property>

<property name=*"contract\_duration"*></property>

</union-subclass>

</class>

</hibernate-mapping>

***In Annotation:***

***Parent Class***

import javax.persistence.DiscriminatorColumn;

import javax.persistence.DiscriminatorType;

import javax.persistence.DiscriminatorValue;

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="emp401")

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

public class Employee {

}

***Sub Class***

@Entity

@Table(name="emp402")

//AttrubuteOverride is optional inorder to override the parent class column names.

@AttributeOverrides({

@AttributeOverride(name="id",column=@Column(name="id")),@AttributeOverride(name="name",column=@Column(name="name"))

})

**public** **class** Regular\_Employee **extends** Employee { }

1. ***Table Per Concrete Class Inheritance:***

Where the tables are linked(joined) using primary key and foreign key relationship.

***In HBM:***

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

<!DOCTYPE hibernate-mapping PUBLIC

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

"src/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.pack.Employee"* table=*"Employee500"*>

<!-- Column Attribute Used to denote the table name shown in db -->

<!-- Column used to denote primary key id in database -->

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<!-- Normal column generated -->

<property name=*"name"* column=*"Emp\_name"*></property>

<!-- Sub-class details given -->

<joined-subclass name=*"com.pack.Regular\_Employee"* table=*"Employee501"*>

<key column=*"eid"*></key>

<property name=*"salary"*></property>

<property name=*"bonus"*></property>

</joined-subclass>

<joined-subclass name=*"com.pack.Contract\_Employee"* table=*"Employee502"* >

<key column=*"eid"*></key>

<property name=*"pay\_hour"*></property>

<property name=*"contract\_duration"*></property>

</joined-subclass>

</class>

</hibernate-mapping>

***IN Annotation:***

import javax.persistence.PrimaryKeyJoinColumn;

import javax.persistence.Table;

@Entity

@Table(name="emp503")

@PrimaryKeyJoinColumn(name="id")

public class Contract\_Employee extends Employee{

}

**HIBERNATE QUERY LANGUAGE(HQL)**

Used to **query the Persistent classes** instead of database using **org.hibernate.Query Interface.**

1. FROM *Employee🡪*select everything from Employee class
2. FROM *Employee As e*🡪
3. SELECT *e.firstname,e.lastname*

FROM *Employee e*

1. FROM Employee e

WHERE e.id>10;

1. FROM Employee e

WHERE e.id>10

ORDER BY e.salary ASC/DESC;

1. Using Named Parameter:

FROM *Employee e*

WHERE e.id=:a; a-> named parameter

1. void setParameter(a,100);
2. Query setFirstResult(int start); 🡪 takes an integer that represents the first row in your resultset, starts from 0.
3. Query setMaxResult(int start); 🡪retrieve a fixed number of objects.
4. SELECT count(e.FirstName) FROM *Employee e*;
5. **Query q= session.createQuery(“*FROM Employee*”);**
6. **List l=q.list(); 🡪 contains list of all properties. And returns a list**
7. Q**uery q=session.createQuery(“FROM employee e WHERE e.id=10)**
8. Employee e1=(Employee) q.UniqueResult()

**HCQL(Hibernate Criteria Query Language)**

Used to build up a criteria query object programmatically where you can apply filtration and logical condition.

Using **org.hibernate.Criteria Interface.**

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

List l=c.list();

***Methods:***

Criteria c1=session.createCriteria(Employee.class)

c1.add(Restrictions.eq(“salary”,10000))

list l=c1.list();

c1.add(Restrictions.gt(“salary”,1000));

c1.add(Restrictions.lt(“salary”,10000));

c1.add(Restrictions.like(name,”Sunandan”);

c1.add(Restrictions.ilike(name,”Sunandan”));

c1.add(Restrictions.between(salary,1000,2000))

c1.add(Restrictions.ge(salary,1000))

c1.add(Restrictions.le(salary,1000))

c1.add(Restrictions.ne(salary,10000));

c1.add(Restrictions.isNull(salary))🡪no values

c1.add(Restrictions.isNotNull(salary))

c1.add(Restrictions.isEmpty(salary))🡪means 0

c1.add(Restrictions.isNotEmpty (salary))

Criteria c2=session.createCriteria(Employee.class);

Criteria c3=Restrictions.gt(salary,2000);

Criteria c4=Restrictions.lt(age,20);

Logical Expression logical=Restrictions.OR(c3,c4);

C2.add(l);

Logical Expression l1=Restrictions.AND(c3,c4);

C2.add(l1);

Criteria c5=session.createCriteria(Employee.class);

C5.add(Restrictions.gt(salary,1000)); -🡪 Employee whose salary greater than 1000 will be printed.

C5.addOrder(Order.desc(salary))

C5.addOrder(Order.asc(salary))

**Projections:**

The criteria API provide org.hibernate.criterion.Projections Class used to apply

Avg(),sum(),min(),max() in criteria instead of using HQL.

Criteria c6= session.createCriteria(Employee.class)

C6.setProjection(Projections.avg(salary)

C6.setProjection(Projections.sum(salary)

C6.setProjection(Projections.min(salary)

C6.setProjection(Projections.max(salary)

C6.setProjection(Projections.count(salary)

C6.setProjection(Projections.countDistinct(salary)

**Hibernate Mapping**

1. One-One Mapping

Fetchtype.lazy is used for collection. If we set fetchtype.lazy then until we fetch the collection, the collection will not be loaded.

Fetchtype.eager🡪 the collection will be loaded at the time of entity loading.