SESSION-17

HIBERNATE CONFIGURATIONS-MAPPINGS

Hibernate Configurations:

Hibernate requires to know in advance — where to find the mapping information that defines how your Java classes relate to the database tables. Hibernate also requires a set of configuration settings related to database and other related parameters. All such information is usually supplied as a standard Java properties file called **hibernate.properties**, or as an XML file named **hibernate.cfg.xml**.

I will consider XML formatted file **hibernate.cfg.xml** to specify required Hibernate properties in my examples. Most of the properties take their default values and it is not required to specify them in the property file unless it is really required. This file is kept in the root directory of your application's classpath.

Hibernate Properties

Following is the list of important properties, you will be required to configure for a databases in a standalone situation −

Sr.No.	Properties & Description
	hibernate.dialect This property makes Hibernate generate the appropriate SQL for the chosen database.
	hibernate.connection.driver_class The JDBC driver class.
	hibernate.connection.url The JDBC URL to the database instance.
	hibernate.connection.username The database username.
	hibernate.connection.password The database password.
	hibernate.connection.pool_size Limits the number of connections waiting in the Hibernate database connection pool.
	hibernate.connection.autocommit Allows autocommit mode to be used for the JDBC connection.

Hibernate Mapping

Hibernate mapping and also explore the relationships that can be established between entities.
hibernate mappings are one of the key features of hibernate. they establish the relationship between two database tables as attributes in your model. that allows you to easily navigate the associations in your model and criteria queries.

you can establish either unidirectional or bidirectional i.e you can either model them as an attribute on only one of the associated entities or on both. it will not impact your database mapping tables, but it defines in which direction you can use the relationship in your model and criteria queries.

the relationship that can be established between entities are-

one to one — it represents the one to one relationship between two tables.
one to many/many to one — it represents the one to many relationship between two tables.
many to many — it represents the many to many relationship between two tables.
you all must have heard about these relationships. in this article, you will learn about all relationships using hibernate.

one to one

there is the one to one relationship between address . address must not be an entity as it is a value type, but for this example, we will consider address as a separate entity.

for the one to one relationship, we need to simply annotate the data member of the corresponding class with @onetoone. after running the application, if we look after the table created by the hibernate, we will find that the user_details table has all the fields i.e id , username, along with a foreign key of address table column (in this example). if we want to achieve two-way binding like user object should have address and vice-versa. then we have to make the user object inside address class and annotate it with @onetoone. then, the address table will also contain the foreign key of the user column.

you will see the name of the field in both the table will be default depending on table name but to change the default implementation, we have to annotate it with @joincolumn with an attribute name.

address.java

```
package com.innovatnm.demohibernate.model;
import javax.persistence.entity;
import javax.persistence.generatedvalue;
import javax.persistence.id;
import javax.persistence.onetoone;

@entity
public class address {

@id @generatedvalue

private int id;

private string street;

private string city;

@onetoone(mappedby="address")

private user user;

public address() {}

public address(int id, string street, string city) {

this.id = id;

this.street = street;

this.city = city;

}

public string getstreet() {
return street;
}


public void setstreet(string street) {

this.street = street;

}
```

```
public string getcity() {

return city;

}


public user getuser() {

return user;

}


public void setuser(user user) {

this.user = user;

}


public void setcity(string city) {

this.city = city;

}


public int getid() {

return id;

}

public void setid(int id) {

this.id = id;

}

}
```

one to many/many to one

there is a one to many relationship between user and mobile . as one user can have more than one mobile. and a many to one relationship between mobile and user .

for the one to many relationship, we have to annotate in the same manner as above with @onetomany to collection type data member . after running the application, if we look after the table created by hibernate, we will find that hibernate is making a new table for mapping of user_details table and mobile table(in this example). for two-way binding, we annotate the user inside the mobile class with @manytoone.

you will see the mapping table and its column name will have a default name generated based on the two tables but to change the default implementation, andwe will annotate it @jointable and the attribute name in this is to change the name of mapping table and attribute joincolumns for the primary key of the same class (user in this case) and inversejoincolumns for the primary key of corresponding class (mobile in this case).

mobile.java

```
package com.innovatnm.demohibernate.model;
import javax.persistence.entity;
import javax.persistence.generatedvalue;
import javax.persistence.id;
import javax.persistence.joincolumn;
import javax.persistence.manytoone;


@entity
public class mobile {

@id @generatedvalue
private int id;
private string brand;
private string model;



@manytoone
@joincolumn(name="user_id")
private user user;


public int getid() {
return id;
}
```

```
public void setid(int id) {
this.id = id;
}

public string getbrand() {
return brand;
}

public void setbrand(string brand) {
this.brand = brand;
}

public string getmodel() {
return model;
}

public void setmodel(string model) {
this.model = model;
}

public user getuser() {

return user;

}

public void setuser(user user) {
this.user = user;

}}
```

many to many

there is a many to many relationship between user and vehicle . we will let consider a world where there are multiple owner of a vehicle and a owner can also own multiple vehicle . you can consider that the vehicle is used for the rent and one user can rent multiple vehicle and a vehicle can be rented by multiple users .

for a many to many relationship, we have to annotate in the same manner as above with @manytomany to collection type data member and for two-way binding, we have to do the same in the corresponding class. after running the application if we look after the table created by the hibernate, we will found that the there is two mapping table formed one made by first mapping (by user in this case) and other mapping table by vehicle class. so, in order to restrict it, we have to tell hibernate that the mapping is already done by other class with the help of attribute mappedby in @manytomany. so, we will add this attribute in either of the class user or vehicle.

vehicle.java

```
package com.innovatnm.demohibernate.model;
import java.util.arraylist;
import java.util.collection;
import javax.persistence.entity;
import javax.persistence.generatedvalue;
import javax.persistence.generationtype;
import javax.persistence.id;
import javax.persistence.manytomany;


@entity
public class vehicle {

@id
@generatedvalue(strategy=generationtype.auto)
private int id;
private string name;



@manytomany(mappedby="vehicle")
private collection&lt;user&gt; user=new arraylist&lt;&gt;();;

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;
}

public collection&lt;user&gt; getuser() {
return user;
}

public void setuser(collection&lt;user&gt; user) {

this.user = user;

}}
```

now, after looking up to user class, you will need to understand things clearly. so here is user class

user.java

```
package com.innovatnm.demohibernate.model;
import java.util.arraylist;
import java.util.collection;
import javax.persistence.cascadetype;
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.onetomany;
import javax.persistence.onetoone;
import javax.persistence.table;



@entity
```

```
@table(name="user_details")

public class user {



@generatedvalue(strategy=generationtype.auto)

@id

private int id;

private string username;



@manytomany(cascade=cascadetype.all)

@jointable(name="usr_vehicle",joincolumns=@joincolumn(name="user_id"),inversejoinc
olumns=@joincolumn(name="vehicle_id") )

private collection&lt;vehicle&gt; vehicle=new arraylist&lt;&gt;();



@onetoone(cascade=cascadetype.all)

@joincolumn(name="address_id")

private address address;



@onetomany(cascade=cascadetype.all)

@jointable(name="user_mobile_mapping",joincolumns=@joincolumn(name="user_id"),i
nversejoincolumns=@joincolumn(name="mobile_id"))

private collection&lt;mobile&gt; mobile=new arraylist&lt;&gt;();;




public collection&lt;mobile&gt; getmobile() {

return mobile;
```

```
}

public void setmobile(collection&lt;mobile&gt; mobile) {

this.mobile = mobile;

}

public collection&lt;vehicle&gt; getvehicle() {

return vehicle;

}

public void setvehicle(collection&lt;vehicle&gt; vehicle) {

this.vehicle = vehicle;

}

public address getaddress() {

return address;

}

public void setaddress(address address) {

this.address = address;

}

public int getid() {

return id;

}

public void setid(int id) {

this.id = id;

}

public string getusername() {
```

```
return username;

}

public void setusername(string username) {

this.username = username;

}

}
```

hibernatemain.java

```
package com.innovatnm.demohibernate;



import org.hibernate.session;

import org.hibernate.sessionfactory;

import org.hibernate.cfg.configuration;



import com.innovatnm.demohibernate.model.address;

import com.innovatnm.demohibernate.model.mobile;

import com.innovatnm.demohibernate.model.user;

import com.innovatnm.demohibernate.model.vehicle;



public class hibernatemain {



public static void main(string[] args) {

user user=new user();
```

```
user user=new user();

user.setusername("dev");

user.setusername("ankit");


address address= new address();

address.setstreet("sector ");

address.setcity("noida");

address address=new address();

address.setcity("muzaffarpur");

address.setstreet("sahebganj");



vehicle veh=new vehicle();

veh.setname("car");

vehicle vehicle=new vehicle();

vehicle.setname("jeep");

vehicle vehicle= new vehicle();

vehicle.setname("bike");

vehicle vehicle= new vehicle();

vehicle.setname("bus");

vehicle vehicle=new vehicle();

vehicle.setname("cycle");

vehicle vehicle= new vehicle();
```

```
vehicle.setname("truck");



mobile mobile =new mobile();

mobile.setbrand("sony");

mobile.setmodel("xperia z");

mobile mobile = new mobile();

mobile.setbrand("redmi");

mobile.setmodel("note  pro");

mobile mobile = new mobile();

mobile.setbrand("nokia");

mobile.setmodel(" plus");



user.setaddress(address);

user.setaddress(address);
address.setuser(user);
address.setuser(user);

user.getmobile().add(mobile);
user.getmobile().add(mobile);
mobile.setuser(user);
mobile.setuser(user);
user.getmobile().add(mobile);
mobile.setuser(user);
user.getvehicle().add(veh);
user.getvehicle().add(vehicle);
user.getvehicle().add(vehicle);
veh.getuser().add(user);
vehicle.getuser().add(user);
vehicle.getuser().add(user);
user.getvehicle().add(vehicle);
user.getvehicle().add(vehicle);
user.getvehicle().add(vehicle);
vehicle.getuser().add(user);
vehicle.getuser().add(user);
```

```
vehicle.getuser().add(user);
sessionfactory sf=new configuration().configure().buildsessionfactory();
session session=sf.opensession();
session.begintransaction();
session.save(user);
session.save(user);
session.gettransaction().commit();
session.close();
}
}
```

after executing the main class, you will find the table as shown:

address table

id	city	street
2	noida	sector 15
9	Muzaffarpur	saheboani

mobile table

id	brand	model	user
3	sonv	xperia z3	1
4	Redmi	Note 5 pro	1
10	Nokia	7 plus	8

user_details table

id	userName	address_id
1	Ankit	2
8	Dev	9

usr_vehicle table

user_id	vehicle_id
1	5
1	6
1	7
8	11
8	12
8	13

user_mobile_mapping table

user_id	mobile_id
1	3
1	4
8	10

vehicle table

id	name
5	car
6	ieep
7	Bike
11	Bus
12	cycle
13	Truck